

NYC TAXI FARE PREDICTION AND VISUALIZATION

Project report submitted in partial fulfillment of the requirement for the degree of
Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

(Sparsh Batta (191296))

(Mehul Kansal (191299))

Under the supervision of

(Dr. Jagpreet Sidhu)



to

Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology Wagnaghat, Solan-173234,
Himachal Pradesh**

Candidate's Declaration

I hereby declare that the work presented in this report entitled “**NYC Taxi Fare Prediction and Visualization**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Dr. Jagpreet Sidhu** (Assistant Professor (SG), Department of CSE & IT).

I also authenticate that I have carried out the above mentioned project work under the proficiency stream "**Data Science**".

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)

Sparsh Batta (191296)

(Student Signature)

Mehul Kansal (191299)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)

Supervisor Name

Designation

Department name

Dated:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date:

Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper

Name: _____ Department: _____ Enrolment No _____

Contact No. _____ E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none">• All Preliminary Pages• Bibliography/Images/Quotes• 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

.....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

Acknowledgement

We are quite grateful to have this as the conclusion of our mission because it took a lot of perseverance and assistance from many people to complete and make such a significant effect on. All that we have accomplished is directly related to their oversight and assistance, for which we are really grateful.

We honor and thank Dr. Jagpreet Sidhu for enabling us to work on the project at Jaypee University of Information Technology and for providing all of us with the guidance and instruction that enabled us to successfully finish the project. Despite the fact that he had involved time management, we are really thankful to him for providing such above-average support and guidance. We are also grateful towards Mr. Mohan Sharma and Mr. Ravi Raina, our respective lab coordinators who provided us with required resources for the successful accomplishment of our project.

We owe a huge debt of gratitude to our project manager Dr. Jagpreet Sidhu, who showed a genuine interest in our errand work and led the group of us until the project's conclusion by providing all the necessary information for developing a strong framework.

We are grateful and sufficiently honored to have received ongoing assistance, which enabled us to successfully complete our endeavor task. In a similar vein, we should extend our sincere gratitude to each and every one of them for their helpful assistance.

Sparsh Batta

191296

Mehul Kansal

191299

TABLE OF CONTENT

Title	Page No.
Main Title	–
Certification [Candidate’s Declaration]	I
Plagiarism Certificate	II
Acknowledgment	III
Table of Content	IV
List of Abbreviations	VI
List of Figures	VIII
List of Tables	XI
Abstract	XII
CHAPTER 1: INTRODUCTION	(1)
1.1 Introduction	1
1.2 Problem Statement	2
1.3 Objectives	2
1.4 Methodology	3
1.4.1 Python	4
1.4.2 Jupyter Notebook	5
1.4.3 NumPy	6
1.4.4 Pandas	7
1.4.5 Sklearn	7
1.4.6 Plotly	8
1.4.7 XGBoost	8
1.4.8 TensorFlow	9
1.4.9 Keras	10
1.4.10 Shapely	11
1.5 Organization	15
CHAPTER 2: LITERATURE SURVEY	(17)
CHAPTER 3: SYSTEM DEVELOPMENT	(28)
3.1 Analytical	28
3.1.1 Vaex	29
3.1.2 Dask	31

3.1.3 Dataset splitting in chunks	31
3.1.4 Drawbacks of Vaex and Dask	32
3.2 Computational	33
3.2.1 Working of Haversine Formula	33
3.3 Experimental	34
3.3.1 Using Shapely library	36
3.4 Mathematical	37
3.5 Statistical	39
CHAPTER 4: EXPERIMENTS AND RESULT ANALYSIS	(42)
4.1 Evaluation Metrics	42
4.1.1 Mean Absolute Error (MAE)	42
4.1.2 Mean Squared Error (MSE)	43
4.1.3 Root Mean Squared Error (RMSE)	43
4.1.4 Root Mean Squared Logarithmic Error (RMSLE)	43
4.1.5 R-Squared Error	44
4.1.6 Generalized function for evaluation_metrics	45
4.2 Linear Regression	45
4.3 SVR (Support Vector Regression)	47
4.4 Random Forest Regression Model	48
4.5 XGB (Extreme Gradient Boosting) Regression	50
4.6 Voting Regressor	51
4.7 Bagging Regressor	53
4.8 HistGradientBoosting Regressor	54
4.9 Artificial Neural Network (ANN)	55
4.10 Extended EDA for Evaluation Metrics	58
4.11 K-Fold Cross Validation	59
CHAPTER 5: CONCLUSIONS	(61)
5.1 Conclusions	61
5.2 Future Scope	61
5.3 Applications	61
REFERENCES	(62)
APPENDIX	(63)

LIST OF ABBREVIATIONS

Abbreviation No.	Abbreviation	Description	Page No.
Abbr. 1.1	NYC	New York City	2
Abbr. 1.2	BQML	Big Query Machine Learning	3
Abbr. 1.3	SVR	Support Vector Regression	3
Abbr. 1.4	RAM	Random Access Memory	3
Abbr. 1.5	XGB	Xtreme Gradient Boosting	4
Abbr. 1.6	CPU	Central Processing Unit	4
Abbr. 1.7	GUI	Graphical Unit Interface	5
Abbr. 1.8	API	Application Programming Interface	8
Abbr. 1.9	GPU	Graphics Processing Unit	9
Abbr. 1.10	TPU	Tensor Processing Unit	9
Abbr. 1.11	GEOS	Open Source Geometry Engine	11
Abbr. 1.12	GIS	Geographic Information Systems	12
Abbr. 1.13	EDA	Exploratory Data Analysis	12
Abbr. 1.14	MAE	Mean Absolute Error	16
Abbr. 1.15	MSE	Mean Squared Error	16
Abbr. 1.16	RMSE	Root Mean Squared Error	16
Abbr. 1.17	RMSLE	Root Mean Squared Logarithmic Error	16
Abbr. 2.1	SVM	Support Vector Machine	17
Abbr. 2.2	OLS	Ordinary Least Squares	20
Abbr. 2.3	ITS	Intelligent Transport System	23
Abbr. 2.4	RNN	Recurrent Neural Network	26
Abbr. 2.5	ANN	Artificial Neural Network	26
Abbr. 3.1	JDA	James Donald Armstrong	31
Abbr. 3.2	NVIDIA	Next Version/ Nevada Invidia	31
Abbr. 3.3	NASA	National Aeronautics and Space Administration	31
Abbr. 3.4	CSV	Comma Separated Value	32
Abbr. 3.5	JFK	John F. Kennedy	34

Abbr. 3.6	UTC	Coordinated Universal Time	41
Abbr. 4.1	SSE	Sum of Squared Errors	45
Abbr. 4.2	ReLU	Rectified Linear Activation Unit	56

LIST OF FIGURES

Figure	Caption	Page No.
Figure 1.1	Python logo	4
Figure 1.2	Jupyter Notebook logo	5
Figure 1.3	NumPy logo	6
Figure 1.4	Pandas logo	7
Figure 1.5	Sklearn logo	7
Figure 1.6	Plotly logo	8
Figure 1.7	XGBoost logo	8
Figure 1.8	TensorFlow logo	9
Figure 1.9	Keras logo	10
Figure 1.10	Shapely logo	11
Figure 1.11	Matplotlib logo	13
Figure 1.12	Seaborn logo	13
Figure 1.13	Ensemble learning	14
Figure 1.14	Voting and Bagging	14
Figure 1.15	Plotly library uses	15
Figure 2.1	SVR outliers	17
Figure 2.2	Random Forest	18
Figure 2.3	XGB framework	19
Figure 2.4	Lasso v/s Linear Regression	21
Figure 2.5	Forward Selection	21
Figure 2.6	Duration and Amount graph	22
Figure 2.7	SVM in ITS	23
Figure 2.8	House Price Prediction with ANN	27
Figure 3.1	Structure of dataset	28
Figure 3.2	Vaex logo	29
Figure 3.3	Out-of-core ML	30
Figure 3.4	Vaex Mini Batches	30
Figure 3.5	Dask logo	31
Figure 3.6	Syntax for chunk size	32

Figure 3.7	Haversine formula	33
Figure 3.8	Extended Haversine	34
Figure 3.9	Haversine in succinct	34
Figure 3.10	JFK airport	35
Figure 3.11	LaGuardia airport	35
Figure 3.12	Newark airport	35
Figure 3.13	Manhattan borough	35
Figure 3.14	Shapely logo	36
Figure 3.15	Boundaries in shapely	36
Figure 3.16	Bearing angle	37
Figure 3.17	Haversine function	37
Figure 3.18	Point class in Shapely	38
Figure 3.19	Lambda function on airports	38
Figure 3.20	Inclusion of features	39
Figure 3.21	Correlation Heat Map	39
Figure 3.22	Correlation Bar Plot	40
Figure 3.23	Pearson's correlation coefficient	40
Figure 3.24	Different types of correlations	41
Figure 4.1	Mean Absolute Error	42
Figure 4.2	Mean Squared Error	43
Figure 4.3	Root Mean Squared Error	43
Figure 4.4	RMSE v/s RMSLE	44
Figure 4.5	R-Squared Error	44
Figure 4.6	evaluation_metrics description	45
Figure 4.7	Best fit line	45
Figure 4.8	Linear regression equation	46
Figure 4.9	Determining β_0 and β_1	46
Figure 4.10	Pipeline class description	46
Figure 4.11	Linear Regression Model	47
Figure 4.12	SVR epsilon tube	48
Figure 4.13	SVR model	48
Figure 4.14	Random Forest Regressor	49

Figure 4.15	Random Forest Model	49
Figure 4.16	Boosting method	50
Figure 4.17	XGB model	50
Figure 4.18	Voting Regressor Working	51
Figure 4.19	Voting Regressor Model	52
Figure 4.20	Bagging Regressor Working	53
Figure 4.21	Bagging Regressor Model	53
Figure 4.22	HistGradientBoosting Model and Working	55
Figure 4.23	Standard Scaler for ANN	56
Figure 4.24	ANN Model and Working	56
Figure 4.25	ANN on Early Stopping	57
Figure 4.26	Evaluation Result of ANN	57
Figure 4.27	RMSE v/s RSquared plot	58
Figure 4.28	Pie chart of RMSE	59
Figure 4.29	Pie chart of RSquared	59
Figure 4.30	RMSE under K-Fold	60
Figure 4.31	RSquared under K-Fold	60
Figure 4.32	Generalized working of K-Fold Cross Validation	60

LIST OF TABLES

Figure	Description of Table	Page No.
Table 2.1	Comparison of different models on the basis of final inference	25

Abstract

The term regression analysis coincides with finding a real value parameter that can be used for prediction purposes. It differs from classification in the sense that, in classification the categorical or discrete values form the basis of model as opposed to regression where the numerical parameters are employed for the same.

Computationally regression analysis is generally more expensive than classification analysis though the same can differ in different contexts.

The prediction analysis is nothing but a synonym of regression and classification analysis as in both of these we do nothing but try to predict the numerical value or categorize our data point on the basis of class labels.

With the advancement of Artificial Intelligence during the globalization era, its subfields including Machine Learning, Deep Learning and the contemporary domain of Reinforcement Learning have shown promising results that extend its functionality and application from medicine, health care, transportation and evolution to politics, geo-science and various other viably hidden technical domains like forensics, criminal activities, court procedures and other various aspects.

The taxi fare prediction is one such small application of the vast array of contributions that has been bestowed upon by this promising field.

For our considerations we have used the data of New York City and its corresponding borough has the basis for developing our regression model.

The sprawling city of NYC is over cumbered with a huge string of taxi chains running across the city which provide a seemingly accurate description of the data that we are going to need to able to process as a preliminary requirement for our model.

The abstraction is done in terms of variable charging across different sections of the city and other aspects like weather or road conditions or traffic jams which are not considered while forming the data but they surely have an implicit impact on the overall fare result that we are going to obtain at the end of the formulation of this whole process.

The inclusion of inflated charges is an important indicator of the fact that this project can be utilized for future purposes as well even when the fare prices change by a certain margin.

Chapter 1: - INTRODUCTION

1.1 Introduction

Predictive analysis is the cornerstone of many upstream activities in data science as well as the machine learning paradigm. The main aim is to generate a user defined data set from some preprocessed information that is available via repositories, documents or articles over the websites. These are most often investigated in the fields of illness, natural catastrophe, and fare prediction. Additionally, it may be used to discover how to prevent future disasters and how to decrease the damage caused by a particular one. Regression analysis may be used to investigate the effects of variables measured on several scales, such as the influence of price modifications and the amount of promotional activities. These benefits make it easier for market researchers, data analysts, and data scientists to find and pick the best set of variables to incorporate into predictive modeling.

When used in conjunction with other business analytical techniques, regression analysis helps firms better understand the significance of their data points and utilize them to inform decision-making. Regression analysis makes it possible to see how, when one of the independent variables is changed, the dependent variable's typical value changes while the other independent variables stay the same. This powerful statistical technique is thus used by business analysts and other data specialists to get rid of unimportant variables and concentrate on the important ones.

The benefit of regression analysis is that it makes it possible to use data analysis to help companies make better decisions. A firm's future weeks, months, and years may be altered by having a better understanding of the issues.

Regression is a generic machine learning approach used to predict a continuous real value and here we are likely to predict dependent variables by using the independent features embedded in our regression equation, and the project dubbed "NYC Taxi Fare Prediction" is a regression model. For this objective, a regression model is used to forecast the taxi prices in and around the various boroughs of New York City in accordance with the variables that the dataset creator has supplied. The distances are given in the form of coordinates which are nothing but the latitudes and longitudes in degrees. The dataset, which is nothing more than the corresponding data-time that indicates the pickup time of a specific passenger or group of passengers, is associated with the key.

When making the necessary projection, fare inflation is also taken into account.

1.2 Problem Statement

The main aim of the problem associated with the project is to estimate the “**fare charges**” that are to be paid by the passengers for a trip around NYC and the fares have some independent precursors associated with it that we need to entertain under our regression model so that we be able to get near to approximate value of the fare charges.

According to the specified parameters, it considers the pickup time, the latitude and longitude of the start location, the latitude and longitude of the end location, the count of passengers and the fare amount per passenger.

It employs several operations, including the Haversine formula and Manhattan distance. The number of passengers and the likelihood of traffic congestion are also considered when assessing the fee.

It uses the assistance of predictive assessment, which makes certain adjustments to data from already-existing data sets in order to identify new trends and fashions. Then, future outcomes and trends are predicted using those advances and styles. Acting predictive analysis helps us forecast next developments and overall performance. Additionally, it is known as prognosis assessment or prognostic method prediction.

1.3 Objectives

The major objective is already been described in the problem statement, which is predicting the fare charges of the passengers for taxi rides that they take across the NYC [**Abbr. 1.1**].

Other objectives include the determination of best possible technique for regression, or in simple terms, the best possible regression model that is most optimal for our current scenario and how we can find the distance between two longitudes and latitudes on the earth using different methodologies or trigonometric formulae and functions.

We will also need to evaluate inflation costs because each year the prices of the taxi fares are increased leading us to carefully consider what fare charges to consider in that particular session with the use of inflation factor that is embedded in the code and is separately devised into a column called as “**inflation_costs**”.

Another major objective could be to reduce the error which again falls under selecting the best possible model or mechanism for determining the taxi fares using the available information.

1.4 Methodology

The methodology we're employing is a subset of BQML [Abbr. 1.2] models, where data is handled in the frontend as database queries. But we can just use the conventional basic machine learning protocols and concepts for its backend implementation.

For regression, prediction, and forecasting purposes, BQML uses linear regression; however, for classification issues, it uses binary logistic regression and multiclass logistic regression approaches.

BQML processing works similarly to any generalized machine learning model creation process in that we first choose the pertinent features from our raw data, then process and clean our dataset by removing or replacing any null or missing values, then apply our model (either Linear Regression or Logistic Regression), and finally validate using any model performance evaluation, such as RMSE (Root Mean Square Error) for evaluation of our model.

We may also employ the Random Forest and SVR [Abbr. 1.3] approaches in contrast to Linear Regression since they can also be used to predict or forecast data.

The cost of a taxi ride depends on the distance travelled and how long it takes to get there (the sum of the drop-off fee, distance fee, and time fee). Although it is simple to estimate the distance and drop charge, it is more difficult to determine the time. It is the end outcome of intricate, nonlinear traffic dynamics.

The dataset for the underlying research, known as "**new- york - city - taxi- fare - prediction**," was given by **Google** as part of a competition that was organised in 2018 in conjunction with an exclusive cooperation with **Coursera**. The event was referred to as the "Playground Prediction Competition."

The key, fee amount, date and time of the journey, latitude and longitude of the start location, latitude and longitude of the end location, and count of the passengers are the eight parameters that make up the dataset.

The original training dataset had close to 55 million data points, which were reduced for sampling reasons and RAM [Abbr. 1.4] constraints to 1 million.

By analyzing data gathered from taxis, we are attempting to resolve a similar issue: calculating travel duration without real-time data. Making such estimations would improve projections for the future.

We have also used SVR (Support Vector Regression) for regression purposes in this context.

Support vector machines are expected to perform well for taxi fare analysis because they have better generalization capabilities and guarantee global minima for given training data. Our findings demonstrate that the SVR predictor may dramatically lower both relative mean errors and root-mean-squared errors of anticipated fares when compared to alternative.

But a better alternative would be to use Random Forest or Extreme Gradient Boosting (XGB) (a more effective technique called **LightBGM** can also be used, [see **Appendix A.1**]). While, Random Forest stays more true to the accuracy, the XGB [Abbr. 1.5] is used to efficiently compute the taxi fares even without taking significant amount of time which is truly remarkable considering the constraints that we have with respect to the size of the RAM and the performance of the CPU [Abbr. 1.6] of our device.

Let us look at the libraries and platforms that we have used for creating our project.

1.4.1 Python



Figure 1.1: - Python logo

Python is a general purpose, high-level interpreted programming language and it finds applications in wide variety of contexts like data analysis, machine learning, general-purpose problem solving, web server development, environment testing and content creation.

It is widely known for its simplicity and readability. Python code is easy to read and understand, even for people who are not experienced programmers.

This makes it an ideal language for beginners, as well as for experienced developers who want to quickly prototype and test new ideas.

Python comes with a huge amount of versatility. Python can be used for a wide range of applications, from small scripts to large-scale web applications and data processing pipelines. It has a large and active community of developers who contribute to its ecosystem by developing libraries, frameworks, and tools that extend its functionality.

1.4.2 Jupyter Notebook

Jupyter Notebook serves as the platform for creating the project-specific notebook. Project Jupyter is where the Jupyter Notebook is suggested. Project Jupyter's goal is to offer interactive computing services, open standards, and open-source software for a range of programming languages. In 2014, Brian Granger and Fernando Pérez split it from IPython.

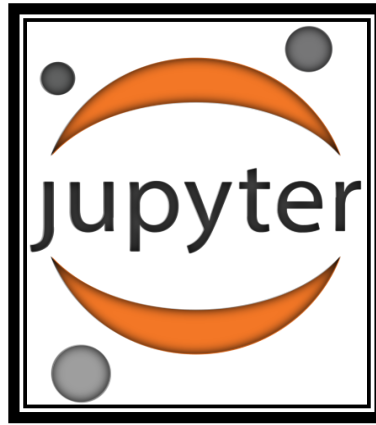


Figure 1.2: - Jupyter Notebook logo

The interface of Jupyter Notebook is divided into two main areas: the notebook area and the kernel area. The notebook area is where users can create and edit notebooks, which are essentially documents that contain a series of cells. Each cell can contain code, text, or multimedia elements such as images and videos. The kernel area is where the code is executed, and the results are displayed in the notebook area. This architecture allows for easy experimentation and data exploration.

Jupyter Notebook offers many features that make it a popular choice for data scientists and researchers. One of its most useful features is its ability to display data visualizations directly in the notebook.

This is made possible by the use of libraries such as Matplotlib, Seaborn, and Plotly. These libraries allow users to create high-quality visualizations such as line charts, scatter plots, heat maps, and more.

Jupyter Notebook has an excellent support for interactive widgets. These widgets allow users to create graphical user interfaces (GUIs) [Abbr. 1.7] for their code, making it easier to interact with and manipulate data. Jupyter Notebook also supports the integration of external tools and libraries, such as TensorFlow and Keras, which are commonly used for machine learning and deep learning.

1.4.3 NumPy

NumPy is a powerful Python library for numerical computations, especially for arrays and matrices. It provides an array object that is much more efficient than Python's built-in lists for numerical computations. NumPy is widely used in the scientific and data analysis communities.



Figure 1.3: - NumPy Logo

Python now has access to the computing power of languages like C and FORTRAN thanks to NumPy. Using Python's NumPy module, you may work with multidimensional arrays and matrices. It is perfect for scientific or mathematical operations due to its effectiveness and quickness.

Signal processing and linear algebra are also supported by NumPy. Therefore, if you need to do any mathematical operations on your data, NumPy is unquestionably the library for you.

There are several ways that Python lists and NumPy arrays are different from one another. First off, NumPy arrays contain more dimensions than Python lists do. Second, whereas NumPy arrays are homogeneous, Python lists are varied.

This means that each member of a NumPy array must be of the same type. Third, NumPy arrays are more efficient than Python lists. NumPy arrays may be created in a variety of ways. One method is to create an array from a Python list.

Once it has been created, a NumPy array can be modified in a variety of ways. For example, you may change an array's shape or use an index to retrieve its elements. Mathematical operations like addition, multiplication, and division may also be performed on NumPy arrays.

1.4.4 Pandas



Figure 1.4: - Pandas Logo

Pandas is a popular open-source data library for Python widely used for working with CSV and Excel files. It provides data structures and functions for working with structured data, including tables and time series. Pandas has a deep rooted integration with NumPy which makes it suitable for numerical and mathematical computations associated with the data sets.

Pandas mainly processes the data in the form of Data Frame, which is a two-dimensional table with labeled columns and rows. Data Frames can be manipulated and analyzed using a wide range of functions provided by Pandas.

Pandas also provides a Series object, which is a one-dimensional array with labels. Series can be used to represent time series data or any other type of data where each value has a label. Series are often used as the columns of a Data Frame.

1.4.5 Sklearn



Figure 1.5: - Sklearn Logo

It provides a wide range of machine learning algorithms like classification, regression, clustering and also for several very specific visualization purposes like statistics involved with the inferences drawn from various models.

Sklearn also provides functions for data preprocessing, model selection, and evaluation. Sklearn is built on top of NumPy, SciPy, and Matplotlib, making it a powerful tool for data analysis and machine learning.

1.4.6 Plotly



Figure 1.6: - Plotly Logo

Plotly is a powerful open-source data visualization library for Python. It provides a variety of visualization tools, including scatter plots, line plots, bar charts, and heat maps. Plotly can create interactive visualizations that can be easily customized, including hover effects, zooming, and panning.

Plotly also provides a range of APIs [Abbr. 1.8] for integrating visualizations into web applications, as well as the ability to export visualizations to various file formats. With Plotly, users can create high-quality and professional-looking visualizations with ease, making it a popular tool for data analysts, scientists, and engineers.

1.4.7 XGBoost



Figure 1.7: - XGBoost Logo

XGBoost is an open-source machine learning library for Python that is widely used for building high-performance, scalable, and accurate models for classification, regression, and ranking problems. It is

designed to be highly efficient and scalable, with support for distributed computing on clusters. It also includes a range of advanced algorithms for feature selection, regularization, and optimization.

1.4.8 TensorFlow



Figure 1.8: - TensorFlow Logo

Google created the open-source, high-performance machine learning package known as TensorFlow. With a large developer and research community that contribute to its development and use it for a variety of applications, it was first released in 2015 and has since then; it has been widely employed in many day-to-day deep learning tasks and automations.

TensorFlow's primary goal is to streamline the creation and application of machine learning models. It offers an adaptable and expandable framework for developing models utilizing a variety of algorithms and methods, such as deep learning, reinforcement learning, and others. Developers can quickly build and train models that can handle huge datasets and challenging tasks with TensorFlow.

Making computational graphs, which are a set of mathematical operations that specify the structure of a machine learning model, is one of TensorFlow's core capabilities. Faster model training and deployment are made possible by the fact that this graph may be optimized for and run on a range of hardware, including CPUs, GPUs [Abbr. 1.9], and TPUs [Abbr. 1.10].

Another crucial aspect of TensorFlow is its high-level APIs, which give programmers a straightforward and user-friendly interface for creating and refining machine learning models. Many different programming languages, including Python, C++, and Java, are supported by TensorFlow, making it usable by a variety of developers.

TensorFlow offers various high-level APIs and tools which make it easier to create, share, analyze and save machine learning models in addition to its core library. There are two of these: TensorFlow Lite,

which enables models to be deployed on mobile and embedded devices, and TensorFlow Hub, which offers a repository of pre-trained models that can be readily included into new applications.

TensorFlow is a robust and adaptable machine learning toolkit that has emerged as a crucial tool for academics and developers working on a variety of applications like image recognition, text analysis and for simple regression or classification purposes. Its widespread use, scalability, and adaptability are evidence of its use, and it will probably continue to be a significant force in the machine learning industry for years to come.

1.4.9 Keras



Figure 1.9: - Keras Logo

Keras is an open source high-end deep learning module built on top of TensorFlow library. Due to its simplicity, usability, and versatility, it has grown in popularity since its initial release in 2015 and is now among the most widely used deep learning libraries.

Keras is made to make creating and experimenting with deep learning models simple for developers. It offers a user-friendly interface for creating neural networks, allowing programmers to concentrate on the model's design rather than the implementation's minute details.

One of the primary characteristics of Keras is its modularity, which enables programmers to construct sophisticated neural networks out of layer combinations of basic building components. Convolutional, recurrent, and dense pre-built layers, among many others, are all included in Keras and are simply coupled to produce unique neural network topologies.

The ease with which Keras can transition between several backends, such as TensorFlow or Theano, without modifying the code, is another key aspect. As a result, developers can test out many deep learning frameworks and pick the one that best suits their requirements.

Additionally, Keras offers a variety of helpful tools for deep learning model training. It offers a variety of optimizers, like Adam and RMSprop that can be used to effectively train models on huge datasets. It also offers a selection of evaluation metrics and loss functions that can be used to gauge a model's effectiveness during training and testing.

Along with its fundamental features, Keras has a number of high-level APIs that make it simple to create and train deep learning models for particular tasks. For example, Keras has an image preprocessing API for jobs like image recognition and a text preprocessing API that makes it simple to preprocess text data for tasks like natural language processing.

Keras is a strong and adaptable deep learning package that makes it simple for developers to construct and test neural networks. Its simplicity and usability make it the perfect choice for deep learning novice developers, while its versatility and adaptability make it a potent tool for deep learning experts.

1.4.10 Shapely



Figure 1.10: - Shapely Logo

Shapely is a Python module that offers resources for working with planar geometric objects. The GEOS [Abbr. 1.11] library, which offers an interface to several computational geometry techniques, is built within it. Shapely is utilized in many different fields, including robotics, computer-aided design, and geographic information systems.

The capacity of Shapely to represent and manipulate several kinds of geometric objects, including as points, lines, polygons, and collections of these things, is one of its primary characteristics. These objects can be built from the ground up or imported from different sources, including shape files or geojson files.

Additionally, Shapely offers a broad selection of geometric operations that can be applied to these objects. These procedures cover both spatial analysis (such as buffering and convex hull computation) and spatial predicates (such as contains, intersects, and touches). Developers who need to interact with geometric objects in their applications frequently choose Shapely's API because it is simple to use and straightforward.

Integrating Shapely with other Python libraries like Matplotlib and Fiona is another one of its important features. Shapely objects can be visualized using Matplotlib, and Fiona offers utilities for reading and publishing GIS [Abbr. 1.12] data in a variety of formats. As a result, integrating Shapely into current GIS workflows and projects is simple.

Additionally, Shapely supports a number of coordinate systems, such as projected, geographic, and Cartesian coordinate systems. Developers are able to work with data which are extracted from different references and translate it between various coordinate systems as necessary thanks to this.

Shapely has a sizable and active community of developers and users, and it is open source and regularly updated. Users can get help and resources from this group, and the library is always being developed and improved.

Shapely is a strong and adaptable Python module for working with geometric objects. Its user-friendly API and wealth of capabilities make it the perfect option for developers who must work with geometric data in their applications, and its connection with other Python libraries and support for different coordinate systems make it a flexible tool for working with GIS data.

Now, let's have a close look at the structure of the project. The project is organized in the form of six stages.

- i.) Loading
- ii.) Formulation
- iii.) Cleaning
- iv.) EDA [Abbr. 1.13]
- v.) Model Examination
- vi.) Extended EDA along with K- Fold Cross Validation

The first stage includes the loading up of dataset from local machine and the dataset has been taken from **kaggle**. The data is then analyzed for further changes, or for inclusions and exclusions.

The second stage is where we implement several functions like the **Haversine formula** and some helper functions. The Haversine formula is used to calculate the distance between two latitudes and longitudes situated over the surface of the earth. Along with that, several features are also incorporated that will make our computation easier, more logical and more concise, like the inclusion of all the points that are near the Newark airport under that particular new feature called as ‘**Newark**’, which will allow us to distribute all the data points close to that particular airport or any other specific location for that matter, and in turn it would become easier for our model to actually look into the geographical pattern of the city that we are trying to portray here. The third step is the data cleaning stage, where redundant features are removed and the null or empty values are taken care of.

Since, our dataset does not contain too many features, so the methods like feature selection, feature reduction would not be preferable to be applied here. Several other factors are also taken into account, like for example; we drop the ‘**key**’ feature as it has no practical significance; we drop the passenger count under 1 which basically means that the particular ride has no passengers and along with that, we also drop the negative fare count.

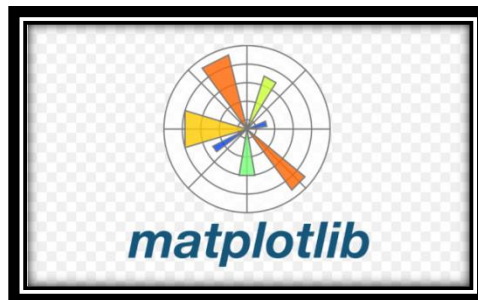


Figure 1.11: - Matplotlib is a famous graph plotting library employed for data science and machine learning operations and is designed for Python



Figure 1.12: - Seaborn is a library that uses matplotlib as a base library for plotting the graphs but additionally incorporate some more features to make the graphs look more attractive and user friendly.

For visualization purposes, we have used Matplotlib and Seaborn (which is built on top of Matplotlib for having redefined and user friendly functionality).

For modeling purposes, the generalized approach of ensemble learning has been employed. The power of ensemble learning can simply be judged from the fact of the number of models that it encompasses within it.

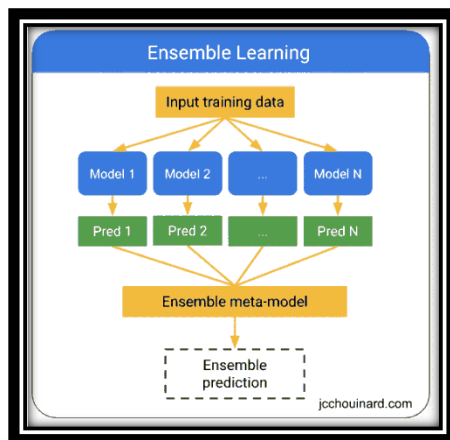


Figure 1.13: - A broad idea about ensemble learning (credits: jchouinard.com)

A voting regressor, a type of ensemble meta-estimator, fits a number of base regressors to the whole dataset one at a time.

The many predictions are averaged to produce the final projection. Machine learning algorithms may be made more accurate and efficient by using a technique known as bagging, also known as Bootstrap aggregating.

We can manage whether we need to reduce the bias error or variance error using ensemble learning techniques. To avoid the data being overfit, both classification and regression models require bagging, particularly decision tree approaches. The graphic below illustrates the distinction between a voting regressor and a bagging regressor:

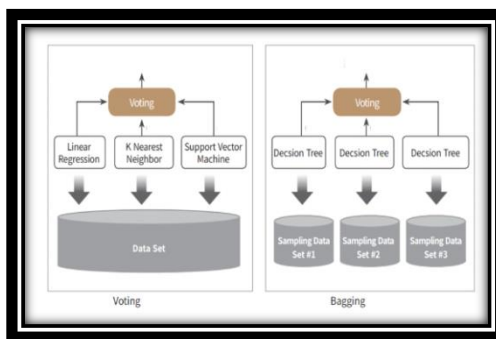


Figure 1.14: - Difference between Voting and Bagging Methods

The extended EDA is performed on the models in the fifth stage in order to statistically monitor how well various models that we have used on our dataset are performing. For this, we've chosen "Plotly," a more capable and user-friendly graph plotting toolkit that allows for more user input. The Plotly Python graphing module produces the interactive, publishable graphs. We can easily construct several graphs like pie charts, bar plots, histograms, polar charts, heat maps and more using the interactive Plotly library. Its user interaction is most powerful tool that manages it to suppress Plotly. Below is an illustration of a common graph made using the Plotly library.

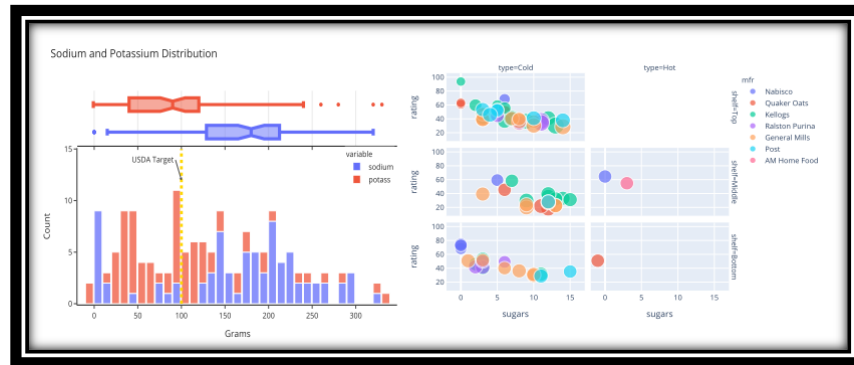


Figure 1.15: - Examples of different plots created using Plotly library

The sixth stage also includes the use of K Fold Cross Validation in order to determine whether a better way is there in order to verify the error stats of our model using the evaluation metrics.

We have used a total of 5 folds in order to determine the evaluation metrics corresponding to only linear regression and XGB (Extreme Gradient Boosting) due to the device and RAM constraints.

1.5 Organization

The organization of the report is as follows: -

Chapter 1: - It provides details about the project and the notion behind it. The problem that we are working upon, the major objective of the project, its different features and how we are going to ensemble everything is the core content of chapter 1

Chapter 2: - Different journals, articles and research papers over the internet have been explored in order to get a definite idea about the various intuitions of the project and how different models can be used.

The effect of different models on the evaluation metrics of our project is also taken into consideration.

Chapter 3: - This chapter let us to go deeper into the analytical, mathematical and experimental interpretation of our data set and how we can manipulate or modify our data set to suit our conditions.

It also takes into account the EDA, which is termed as the statistical interpretation of our data.

The EDA in most of the forms is achieved by matplotlib and seaborn libraries, but we can also utilize better libraries such as Plotly having more user friendliness and more user intervention with robust features like data hovering capabilities and is available in vibrant colors and unique designs.

Chapter 4: - It basically covers the evaluation metrics that will define the accuracy of our models and each and every model is discussed in detail.

A total of 5 evaluation metrics, namely the MAE [Abbr. 1.14], MSE [Abbr. 1.15], RMSE [Abbr. 1.16], RMSLE [Abbr. 1.17] and R-Squared Error are taken into account for determining the accuracy of our model. And along with that why and how each model is used in a particular context is also mentioned here.

Chapter 5: - Under this chapter, the conclusion of our project and the future scope associated with it have been discussed in regards to further techniques and optimizations that can be implemented in our dataset for better accuracy in a relatively less amount of time.

Along with that, optimizations related to collecting massive data on a much quicker basis would also be considered in the light of training over large datasets.

Note: -

The references and appendices are appended at the end for indulging into the various contexts related with the source of the project and for describing as from where the said text was referenced.

Note that some references may be in the form of textual publications while some have been extracted from some publications that were conceived in the form of seminars or virtual meetings.

The appendices have been referenced in the inner text wherever they have been referred and are purely meant for gathering on some extra insight into that topic which otherwise would have been a futile and irrelevant exercise to include within the main context.

These include some of the side-referenced topics such as Lasso Regression, Ridge Regression, LightBGM and Intelligent Transportation System (ITS).

The Figures, tables, sections and subsections corresponding to each chapter have been described above and have been excluded from the main content for modularity.

Chapter 2: - LITERATURE SURVEY

Regression analysis is a statistical method used to analyze the relationship between a dependent variable and one or more independent variables. It involves the use of a regression model to estimate the values of the dependent variable based on the values of the independent variables. The objective of regression analysis is to find the best-fitting line or curve that can predict the values of the dependent variable based on the independent variables.

Regression and classification are two common types of statistical modeling techniques used in machine learning and data analysis.

Linear regression is the simplest and the most basic method of regression that is used for prediction purposes. But there are several other regression analysis techniques that we need to look forward to.

The SVR method, commonly called as the Support Vector Regression method is a subclass of SVM [Abbr. 2.1] algorithms that is used for regression purposes. Though, it is majorly used as a classifier rather than a regressor, but we can still utilize the power of SVM kernels in order to get a best fit line that will help us to arrive at an optimal result and reduce the RMSE (Root Mean Squared Errors).

According to a [sciencedirect.com](https://www.sciencedirect.com) article by Helman written in 2019, SVR is described as follows: -
“SVR applies the same premise as SVM, but to regression issues. SVMs are frequently used to categorize and organize issues. Regression using SVMs is not always well-documented. SVRs are these models. Finding a function that approximates the mapping from the input domain to the actual number in the training sample is difficult when using regression. SVR focuses mostly on taking into account factors within a decision's parameters. The widely used SVR method gives you freedom and allows you to select your level of error tolerance through the use of both a reasonable error margin (ϵ) and an acceptance adjustment that is higher than the acceptable error rate.”

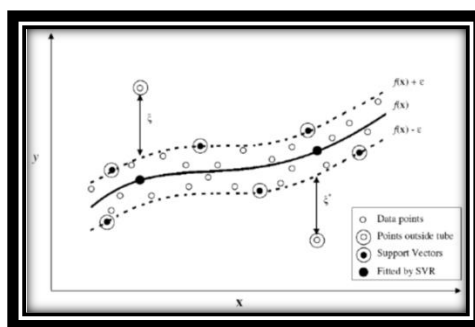


Figure 2.1: - A diagram showing the hyper planes within which our regression line is contained. The black dots represent the points fitted by SVR while the concentric dots are outliers

Unlike linear regression, SVR does not assume a linear relationship between the variables. Instead, it tries to find a nonlinear relationship between the variables by transforming the data into a higher-dimensional space. SVR aims to find a hyperplane that has the maximum distance from the closest data points to it, while still keeping the error within a certain tolerance level. This tolerance level is set by a parameter called epsilon. SVR is less sensitive to outliers than linear regression, and it can handle nonlinear relationships between variables.

Random Forest, probably one the best algorithm for prediction purposes is nothing but a collection of several decision trees which when ensemble together provides a series of outputs that is determined using “**Majority Voting**”

Higher accuracy is obtained and overfitting is avoided because to the larger number of trees in the forest.

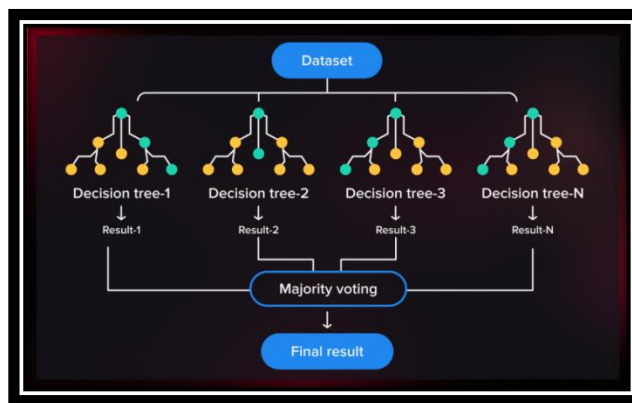


Figure 2.2: - A diagram which depicts the general representation of the working of a typical Random Forest architecture.

The gradient boosting techniques like **XGB** (Extreme Gradient Boosting) Regressor generally outperform Random Forests but with a little tweaking in the data set or by taking extremely large amount of data, Random Forests can yield better results.

XGBoost is generally faster than Random Forest as it is optimized for speed and performance but on the other hand, Random Forest is generally more interpretable than XGBoost as it provides feature importance scores for each feature, while XGBoost's feature importance is more difficult to interpret.

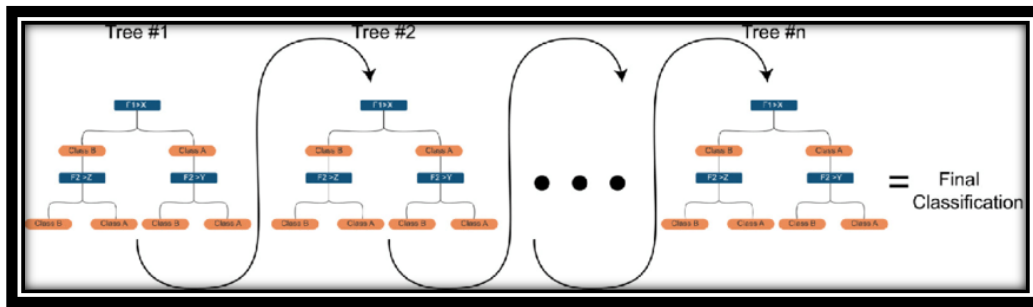


Figure 2.3: - A basic representation of how a typical XGB model performs deep penetration of trees in a sequential fashion, one after the other

According to Wikipedia, The XGB has some salient features which make it better than the usual Random Forests. These are as follows: -

- i.) Using single, distributed systems and performing outside core computation
- ii.) Additional randomization criteria
- iii.) Selecting features automatically
- iv.) A corresponding decrease in leaf nodes
- v.) Faster processing due to the effects of gradient boosting

XGBoost can handle large datasets and distributed computing more efficiently than Random Forest.

The Random Forests, though works well on small datasets are typically employed for large data where we have near about millions of records because of its strong accuracy. The XGB, even though has gradient boosting can sometimes fail to surpass the Random forests in terms of speed as it uses sequential generation of trees whereas Random Forests make use of parallel generation of trees and finally the voting algorithm predicts the results.

The research paper proposed by **Christophoros Antoniadis, Delara Fadavi, Antoine Foba Amon Jr.** titled, **“Fare and Duration Prediction: A Study of New York City Taxi Rides”** which was presented in **December 16, 2016** at **Stanford University** proposes the same idea about Random Forest. In the section 5.2 they stated that,

“Given that it can simulate the nonlinearities of traffic and location effect, the random forest model performs better than any other models. Although the pickup and drop-off sites are taken into account, the model does not have a mechanism to estimate the effects of the places along the route. When travelling through high-speed zones with little to no traffic, regions with heavy traffic can still be reached reasonably quickly. They are regarded to reasonably forecast duration and fare given the factors that have been taken into consideration in the models. Rotating the geographic coordinates

does not result in a substantial increase in forecast accuracy, even if employing rides per hour and average speed per hour improves the models and hence serves as proxies for traffic modeling.”

The dataset is termed as “**new- york – city - taxi- fare - prediction**” that issued for the underlying project and has been provided by **Google** in context of a competition held in 2018 in association with an exclusive partnership with **Coursera**. The competition was titled “**Playground Prediction Competition**”.

The dataset consists of **8 columns** consisting of the **key, taxi fare charges, the latitudes and longitudes of start location and end location, the number of passengers or travellers and the cost per head**.

In the context of predicting taxi fares across a city, linear regression can be used to model the relationship between the fare amount (dependent variable) and various predictor variables, such as distance traveled, time of day, and location.

To use linear regression for predicting taxi fares across the city, you would typically start by collecting data on past taxi rides from a variety of sources, such as taxi companies, government data sets, or online APIs. This data should include information on the fare amount, distance traveled, time of day, and location (such as latitude and longitude).

Once you have collected the data, you would typically split it into two sets: a training set and a testing set. The training set is used to train the linear regression model, while the testing set is used to evaluate its accuracy.

To train the linear regression model, you would typically use an algorithm such as OLS [**Abbr. 2.2**] or gradient descent to find the coefficients that minimize the sum of squared errors between the predicted fares and the actual fares in the training set. The resulting model can then be used to predict fares for new taxi rides based on their distance, time of day, and location. It is important to note that the accuracy of the predictions depends on the quality and quantity of the data used to train the model. Other factors that can affect the accuracy of the predictions include the choice of predictor variables, the functional form of the model (e.g., linear, quadratic, or higher-order), and the presence of outliers or other sources of noise in the data.

Though, not necessary here, we can regularize the coefficients of our linear regression using the lasso technique [**see Appendix A.2**], however a better method than lasso is ridge regression [**see Appendix A.4**].

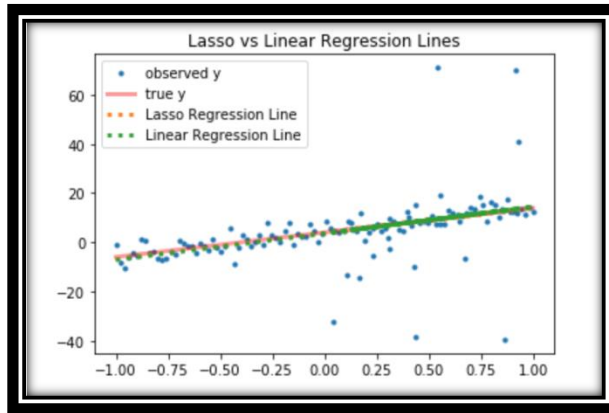


Figure 2.4: - Difference between best-fit lines considered under Linear Regression and Lasso Regression

The lasso approach may be used to decrease coefficients in order to further find the best collection of variables to employ. A model that is sparser and simpler to interpret is produced by using lasso.

The penalizing parameter in Lasso is run across a range of values for both the above said prediction models. In order to choose the value of to be utilized, the lasso model with the lowest error is found using cross validation. The values that produced the minimum cross validation error in each case are very close to zero. The optimal parameter, for example, is $1.19 \cdot 10^{-4}$ for the fare regression. Due to these findings, we must utilize the forward-selected linear regression model rather than the lasso method penalizes 5 variables.

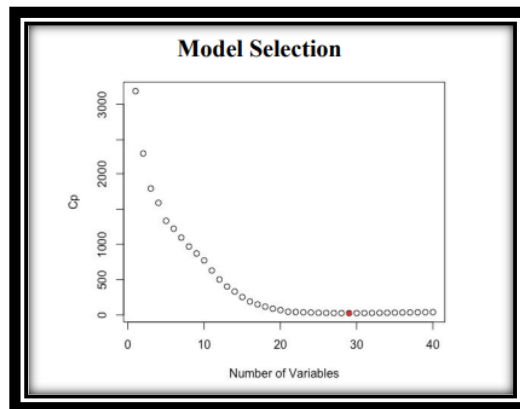


Figure 2.5: - Forward selection using linear regression

Following feature selection in, linear regression can be a viable approach for understanding the effect of number of variables on our parameter C_p (complexity parameter). The longitude of the pickup location is a continuous variable that represents the east-west position of the location relative to the

prime meridian, which is a reference line that runs through Greenwich, England and is used as a reference for determining longitude.

Including pickup longitude as a predictor variable in a linear regression model can be useful because it provides information about the geographic location of the pickup, which can be an important factor in determining the fare amount. For example, taxi fares in urban areas may be higher than in suburban or rural areas due to factors such as higher demand and congestion.

When using pickup longitude as a predictor variable in linear regression, it is important to ensure that the variable is appropriately scaled and centered to avoid issues such as Multicollinearity and bias in the coefficient estimates. One important thing that we can observe is that distribution of ride duration and fare amount is almost similar with some fluctuations in between.

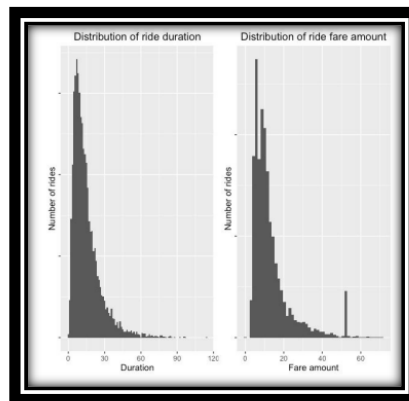


Figure 2.6: - Duration and fare amount, both follow the same distribution

This implies that just like fare amounts, travel time also increases at the same pace. So, instead of fare amounts, we can also focus on journals that help us to compute the predicted time travelled between various locations.

In a 2004 **IEEE** research paper published by **Chun-Hsin Wu, Jan-Ming Ho** and **D. T. Lee**, the idea of SVM is proposed and that is used for time travel prediction.

In many transportation analyses, travel-time data serve as the foundational components for a variety of performance indicators.

Transportation systems can benefit greatly from the application of regression techniques. These techniques can be used to aid in the planning, designing, operation, and evaluation of such systems. When it comes to advanced traveler information systems, the use of travel-time data can be especially valuable for providing pre-trip and en route information. By analyzing such data, drivers and passengers can make more informed decisions, such as adjusting their timetables or route choices.

Support Vector Machines (SVM) are a popular machine learning technique that can be used in transportation systems and route guidance systems for a variety of purposes. SVMs can be trained on large datasets of traffic and travel time information to predict future travel times, traffic congestion, and other factors that impact route planning and guidance.

In transportation systems, SVMs can be used for traffic prediction, vehicle tracking, and real-time route optimization. For example, SVMs can predict traffic congestion and suggest alternative routes to drivers to help them avoid delays. They can also be used to optimize traffic signals to improve traffic flow and reduce congestion.

In route guidance systems, SVMs can be used to suggest the most efficient route for drivers based on real-time traffic information. By analyzing traffic data from multiple sources, SVMs can identify the fastest and most reliable routes for drivers to take, even in high traffic areas.

SVMs can also be used in transportation planning and design. By analyzing historical traffic data, SVMs can predict future traffic patterns and help transportation planners design more efficient and effective transportation systems.

Support Vector Machines (SVMs) are a type of machine learning algorithm that are relatively robust to noisy or faulty data. SVMs work by finding the hyperplane that best separates the classes of data points in a high-dimensional space.

There are several studies that use SVM to vision-based intelligent cars for (ITSs) [Abbr. 2.3] [see Appendix A.3], such as head identification, traffic-pattern recognition, and vehicle detection.

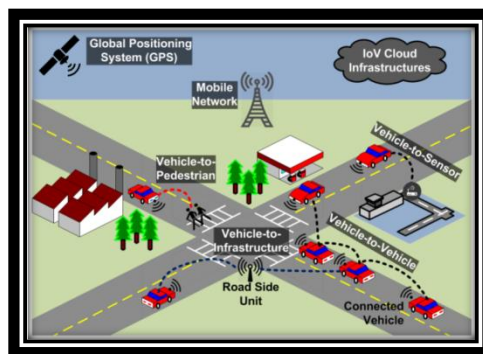


Figure 2.7: - SVM is regarded as one of the pioneer methods for Intelligent Transport System (ITS's)

SVM has recently been used to forecast time series, Support vector regression (SVR), which has demonstrated numerous innovations and convincing results, like predicting forecasts of the financial market, power price, power consumption calculation, reconstruction of Systems in chaos. Except for the forecast of traffic flow,

But there aren't many SVR outcomes for time-series analysis. Given the numerous positive outcomes of time-varying SVR prediction applications drive our research in modeling travel times with SVR.

It seems that SVR is one of the best algorithms for travel-time prediction and seemingly outperforms many regression models that are in the same pace.

But this is not always the case.

The study article "Fare and Duration Prediction: A Study of New York City Taxi Rides" suggested using the Random Forest approach to forecast the fare charges.

The time it takes to travel from one place to another depends on the destination because there are more cars on the road at different times of the day.

While it's impossible to account for every location between the beginning and ending points of a route, we can use the pickup and drop-off locations to simulate the effects of traffic and congestion. However, linear models don't account for nonlinear effects of locations on traffic and travel time.

Random forest is a machine learning technique that can be used for predicting taxi fares. This technique works by combining many decision trees that are created using bootstrapped samples of the training data.

Each tree segments non-overlapping areas of the predictor space, which is the set of all potential values for the qualities that may impact the fare amount.

To improve the accuracy of the random forest model, the range of predictor space can be narrowed down to include only the most relevant variables that have the most significant impact on the fare amount.

Additionally, averaging over more trees also helps to improve the accuracy of the model.

We have considered four publications that revolve around the whole scenario of our

The below table describes the different research papers proposing the different models, their methodologies and the final inference that they make.

Author(s)	Journal/Conference, year	Published By (IEEE, Elsevier, Springer)	Methodology	Inference
Chun- Hsin Wu, Jan-Ming Ho, D.T. Lee	2004	IEEE	SVR (Support Vector Regression)	Moderately Optimal
Kunal Soni	2019	IJSRCSEIT	Linear Regression	Moderately Optimal
Christophoros Antoniadis, Delara Fadavi, Antoine Foba Amon Jr.	2016	IEEE	Random Forest, SVR and Linear Regression	Random Forest outperforms SVR and Linear Regression
Jun Xu, Rouhollah Rahmatizadeh, Ladislau Boloni and Damala Turgut	2017	IEEE	Neural Networks	Neural Networks can be a pretty good choice for regression purposes for huge data set

Table 2.1: Comparison of different research papers on the basis of methodologies and inference drawn at the end

There are very rare research papers which signify the use of Artificial Neural Networks for predicting real values, or in simple terms, solving the regression problems.

But we can look at a similar research paper that focuses on finding taxi demand using RNN [Abbr. 2.4] RNN's are mostly used for predicting text and speech records but they can also be employed for regression purposes.

Real-time prediction of taxi demand is a challenging problem that has gained a lot of attention in recent years. One popular approach to solving this problem is to use recurrent neural networks (RNNs). RNNs are a class of artificial neural networks that can process sequential data and have been used in various fields, including speech recognition, image captioning, and natural language processing.

To predict taxi demand, the first step is to collect historical data on taxi demand, including the time of day, day of the week, and other relevant factors such as weather, events, and holidays. This data can then be used to train an RNN model, which can learn to recognize patterns and relationships between these factors and the demand for taxis.

Once the model has been trained, it can be used to make real-time predictions of taxi demand based on current and historical data. For example, the model can be used to predict the demand for taxis in the next hour based on the current time of day, the day of the week, and other relevant factors.

There are several challenges associated with real-time prediction of taxi demand using RNNs. One challenge is the need for large amounts of historical data to train the model effectively. Another challenge is the need to process data quickly and efficiently in real-time to make accurate predictions.

Despite these challenges, real-time prediction of taxi demand using RNNs has shown promising results in various studies. These models have the potential to improve the efficiency of taxi services and reduce wait times for customers.

Moving on to artificial neural networks, which are the simplest form of Neural Networks, they can be easily employed for regression purposes.

A typical example of how ANN [Abbr. 2.5] can be used to formulate a regression problem, such as predicting the price of a house using different features or parameters associated with it like its weight, age, doors etc.

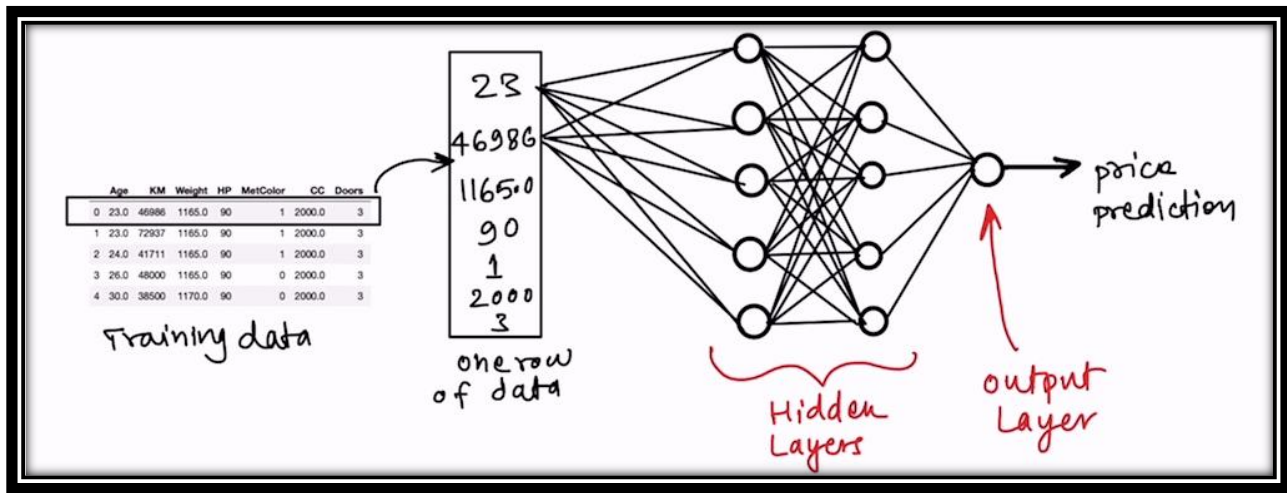


Figure 2.8: - Prediction of house price using ANN

Artificial neural networks can be utilized to predict a numerical value in regression problems by training the network to learn the correlation between the input variables and the continuous output variable.

The ANNs model the relationship between inputs and the output by learning the mapping between them. The basic type of ANN for regression is a feed-forward neural network with a single hidden layer. Input layer receives input variables, the hidden layer processes this information, and the output layer provides the predicted value.

During training, the network adjusts the weights and biases to minimize the difference between the predicted output and the actual output, using a loss function such as mean squared error.

To avoid overfitting, regularization techniques such as dropout or weight decay can be used. Hyper parameters, such as the number of neurons in the hidden layer, learning rate, and activation functions, can be optimized to enhance the model's performance.

And the main difference between them lies in their architecture and their ability to handle sequential data.

The basic neural networks have a simple feed forward architecture where data flows through the input layer, hidden layers and finally through the output layer, while the latter, more complex have loops in their architecture that allow the previous output to be fed back into the network as input.

Chapter 3: - SYSTEM DEVELOPMENT

3.1 Analytical

The analytical treatment of the model is based on the fact that we have to analyze our dataset and pick up the best and the most reasonable features that we require and drop off the features that are redundant.

The structure of the dataset is as follows: -

A	B	C	D	E	F	G	H	I
key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	
2009-06-15 17:26:	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.712278	1	
2010-01-05 16:52:	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1	
2011-08-18 00:35:	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.991242	40.750562	2	
2012-04-21 04:30:	7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.991567	40.758092	1	
2010-03-09 07:51:	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1	
2011-01-06 09:50:	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.73163	-73.972892	40.758233	1	
2012-11-20 20:35:	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	40.764842	1	
2012-01-04 17:22:	16.5	2012-01-04 17:22:00 UTC	-73.9513	40.774138	-73.990095	40.751048	1	
2012-12-03 13:10:	9	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	40.731628	1	
2009-09-02 01:11:	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.99154	40.758138	2	
2012-04-08 07:30:	5.3	2012-04-08 07:30:50 UTC	-73.996335	40.737142	-73.980721	40.733559	1	
2012-12-24 11:24:	5.5	2012-12-24 11:24:00 UTC	0	0	0	0	3	
2009-11-06 01:04:	4.1	2009-11-06 01:04:03 UTC	-73.991601	40.744712	-73.983081	40.744682	2	
2013-07-02 19:54:	7	2013-07-02 19:54:00 UTC	-74.00536	40.728867	-74.008913	40.710907	1	
2011-04-05 17:11:	7.7	2011-04-05 17:11:05 UTC	-74.001821	40.737547	-73.99806	40.722788	2	
2013-11-23 12:57:	5	2013-11-23 12:57:00 UTC	0	0	0	0	1	
2014-02-19 07:22:	12.5	2014-02-19 07:22:00 UTC	-73.98643	40.760465	-73.98899	40.737075	1	
2009-07-22 16:08:	5.3	2009-07-22 16:08:00 UTC	-73.98106	40.73769	-73.994177	40.728412	1	
2010-07-07 14:52:	5.3	2010-07-07 14:52:00 UTC	-73.969505	40.784843	-73.958732	40.783357	1	
2014-12-06 20:36:	4	2014-12-06 20:36:22 UTC	-73.979815	40.751902	-73.979446	40.755481	1	
2010-09-07 13:18:	10.5	2010-09-07 13:18:00 UTC	-73.985382	40.747858	-73.978377	40.76207	1	
2013-02-12 12:15:	11.5	2013-02-12 12:15:46 UTC	-73.957954	40.779252	-73.96125	40.758787	1	
2009-08-06 18:17:	4.5	2009-08-06 18:17:23 UTC	-73.991707	40.770505	-73.985459	40.763671	1	

Figure 3.1:- The structure of the dataset which consists of 8 features by default (including the fare_amount)

The dataset contains a total of **55 million** data points corresponding to the taxi rides around the different boroughs of New York City from 2008-2016. This dataset was proposed by Google and was provided as a problem for the “**Playground Prediction Competition**” which was held in **2018** under the partnership of Google and Coursera

The **key** is redundant so can be removed from our dataset as it contains the same information as that of pickup_date_time.

The **pickup_date_time** is in UTC format, so it can be split into weekdays, hours, minutes, and seconds for easy computation.

The **fare_amount** provided here is not treated analytically with respect to the **inflation** that occurs in the prices every year.

So, we need to take extra care for that as well in order to get a better predicted answer that will stand the test of time even after decades.

The **passenger_count** is denoted by a number that denotes the number of people that are present in a particular ride.

As for the **RAM** and device constraints it would not be possible to account for 55 million rows and most of the operations that we are trying to perform (some of them like Random Forests, SVR and Voting Regressor may be quite costly for a large number of data points) would end up choking the whole internal memory.

As for our purposes we are only accounting for about 80,000 data points as of now.

Some of the libraries and the methods that we tried to use for the purpose of achieving the desired speed are as follows: -

3.1.1 Vaex



Figure 3.2: - Vaex library logo

Vaex is a Python module for exploring and visualizing large tabular datasets using lazy Out-of-Core Data Frames (comparable to Pandas). On an N-dimensional grid, it can compute statistics like mean, total, count, standard deviation, etc. up to a billion (10^9) objects/rows per second. Histograms, density charts, and 3D volume rendering are used in visualization to enable interactive large data exploration. For optimum speed, Vaex makes advantage of memory mapping, a zero memory copy policy, and lazy calculations (no memory wasted).

As Vaex uses the concept of Out-Of-Core data frame, so it is library that is categorized under Out-Of-Core Machine Learning which is an emerging field in ML that relies on the concept of multi-cored CPU for parallel processing to achieve maximum throughput.

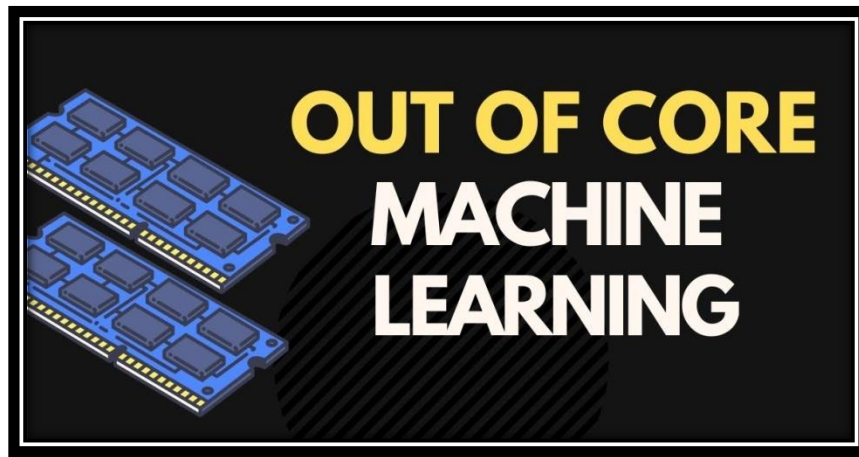


Figure 3.3: - Vaex Out-of-Core Machine Learning embraces the idea of Multi-Core CPU

Processing data that is too vast to fit within a computer's main memory is often referred to as being **out-of-core**.

Randomly accessing portions of a dataset often results in a (relatively) minimal performance hit when the dataset neatly fits into the main memory of a machine.

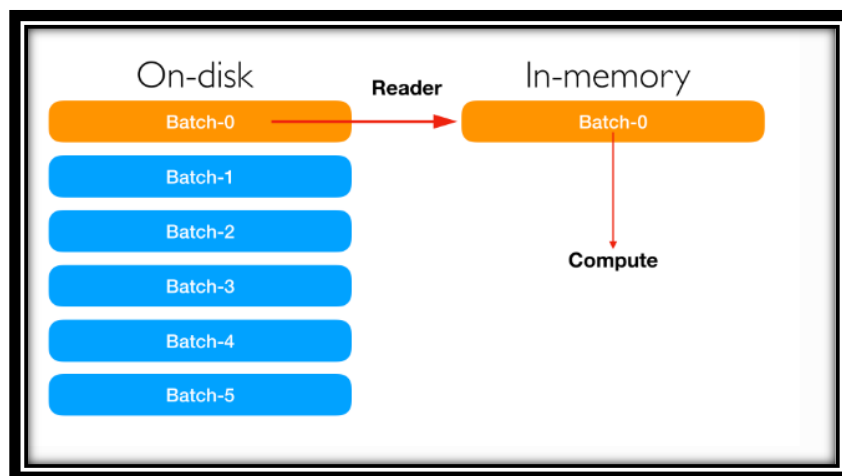


Figure 3.4: - Vaex creates mini batches of our dataset and only one particular batch is loaded into the memory at a particular time

It becomes exceedingly costly to randomly search to a chunk of data or to analyze the same data more than once when data must be stored in a medium like a huge rotating hard disc or an external computer network.

An out-of-core algorithm would attempt to retrieve all pertinent data in a single sequence in such a situation.

However, given the deep memory structure of current computers, switching from random to sequential access can improve speed even for datasets that fit in memory.

Actually, as it ensures that there will always be a little bit of data in the main memory at any one moment, the capacity to learn gradually from a mini-batch of instances is essential to out-of-core learning.

It may take some adjusting to get a mini-batch size that balances relevance and memory footprint.

3.1.2 Dask



Figure 3.5:- Dask library logo

The working principle of Dask is similar to that of the Vaex library.

Dask is a parallel computing-focused open-source Python library. Python programming can be scaled via Dask from single-core local workstations to massive distributed cloud clusters.

On GitHub, Dask was built in December 2014 by Matthew Rocklin and has received over 9.8k ratings and 500 contributors.

Retail, governmental, financial, as well as life science and geophysical institutes, utilize Dask. Among the companies that utilize Dask are Wal-Mart, Wayfair, Grub Hub, JDA [**Abbr. 3.1**], General Motors, NVIDIA [**Abbr. 3.2**], Harvard Medical School, Capital One, and NASA [**Abbr. 3.3**].

3.1.3 Dataset splitting in chunks

Even with compression, your data file may occasionally be too big to load entirely into memory. So how do you swiftly process it?

You can only put a portion of the file into memory at once by importing and processing the data in batches. So you can process files that take up more RAM.

Pandas allows us to read data in parts rather than reading everything into memory. With CSV [Abbr. 3.4], we can only load a small number of the lines into memory at once.

In particular, an iterator across DataFrames rather than a single DataFrame is returned if the chunk size parameter is used with pandas.read_csv.

The syntax and the working of the modified Pandas function is highlighted below: -

```
In [2]: import pandas
        for chunk in pandas.read_csv("redwinequality.csv", chunksize=100):
            print(chunk)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.700	0.00	1.9	0.076	
1	7.8	0.880	0.00	2.6	0.098	
2	7.8	0.760	0.04	2.3	0.092	
3	11.2	0.280	0.56	1.9	0.075	
4	7.4	0.700	0.00	1.9	0.076	
..	
95	4.7	0.600	0.17	2.3	0.058	
96	6.8	0.775	0.00	3.0	0.102	
97	7.0	0.500	0.25	2.0	0.070	
98	7.6	0.900	0.06	2.5	0.079	
99	8.1	0.545	0.18	1.9	0.080	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

Figure 3.6:- Syntax of chunk size for pandas.read_csv method

The other issue with enormous volumes of data is that computing may also create a bottleneck, which chunking doesn't alleviate. So, the first two techniques, viz. the Vaex and the Dask libraries seem to be the better choice.

3.1.4 Drawbacks of Vaex and Dask

Though Dask and Vaex provide speed but it comes at a functional cost. So, have preferred to use traditional pandas in our final choice because of the following reasons: -

- Dask performs weirdly with some very crucial and useful Pandas functions like **shape** or **describe** and as such cannot be used as a dedicated data frame management library.
- Vaex though is better than Dask at most aspects still lacks the capability to manipulate the axis of the dataset and lacks the incorporation of lambda functions inside the features of the dataframe.

But these old obsolete trigonometric functions can be written in the form of simpler functions like sine, cosine etc.

For instance, haversine $(\theta) = \sin^2(\theta/2)$.

But this is not the formula that we define while considering the latitudes and longitudes.

$$\begin{aligned} a &= \sin^2(\phi_B - \phi_A/2) + \cos \phi_A * \cos \phi_B * \sin^2(\lambda_B - \lambda_A/2) \\ c &= 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a}) \\ d &= R * c \end{aligned}$$

Figure 3.8: - Extended haversine formula for including the latitudes and longitudes

The ‘d’ in the above formula signifies the distance between the two points given their latitudes and longitudes.

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Figure 3.9: - Haversine formula written in a more succinct form

3.3 Experimental

Apart from the haversine formula, we have also made some extensions in case of dimensionality of our dataset and performed experimental splitting on our dataset to increment the possible values of the evaluation metrics.

There are a total of 3 airports in **New York City**, namely **JFK** [Abbr. 3.5], **LaGuardia** and **Newark**.

The points that are near a particular specific airport has been included in what is called a virtual circle so that all the coordinates that near to any of those particular airports are considered as a part of that area.

Same thing, we have done with all the places that are within or around the **Manhattan** borough of New York City. The coordinates we got from online resources corresponding to the following places that we mentioned above are as follows: -

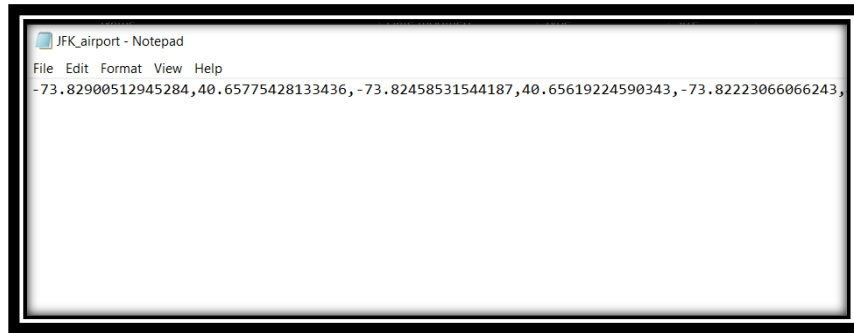


Figure 3.10: - Coordinates of JFK airport

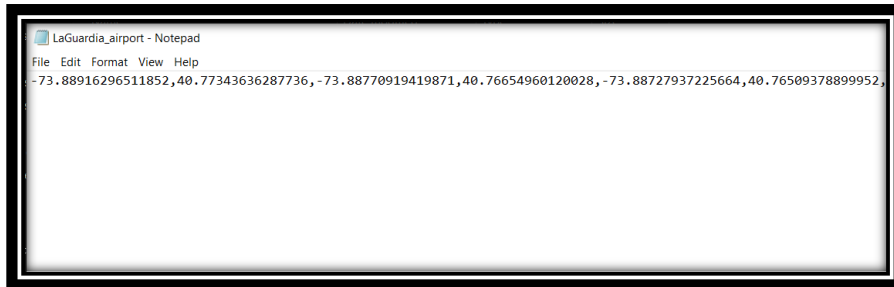


Figure 3.11: - Coordinates of LaGuardia airport

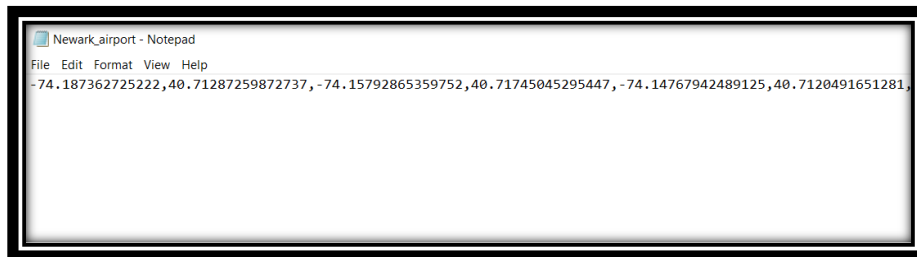


Figure 3.12: - Coordinates of Newark airport

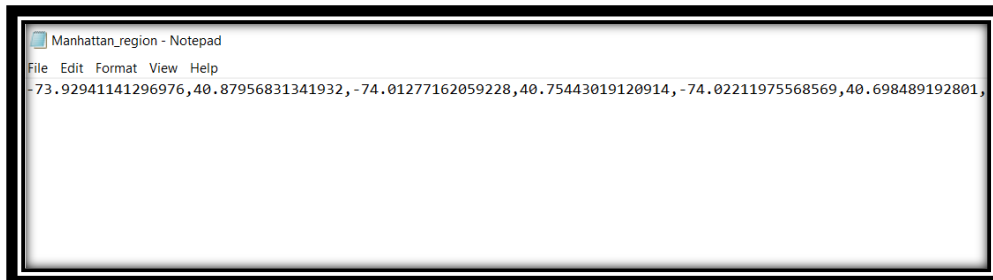


Figure 3.13: - Coordinates of Manhattan borough of NYC

In these files the coordinates are separated by commas and we cannot practically utilize them in this format unless of course, we modify them to suit our requirements, specifications and conditions. For that purpose, we have used **Shapely** library.

3.3.1 Using Shapely library



Figure 3.14: - Shapely library logo

If you wish to mix them, the library lets you interact with the three primary types of geometric objects: Point, Line String, and Polygons+ geometry collections.

There are many more, including linear rings, multi-points, multi-polygons, etc., but for now they will work just well because the approaches are extremely portable.

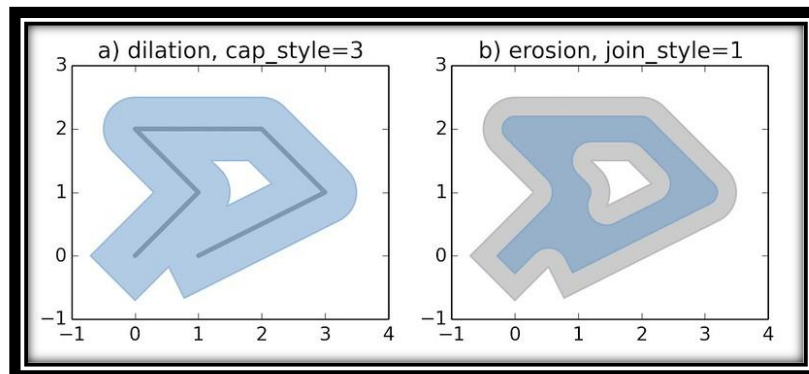


Figure 3.15: - The boundaries as defined by shapely, dilation defines the central path while erosion defines the spatial area around the path

The main aim of using Shapely library is to predict the given geometrical shape from the set of coordinates and this would be our aim in order to define a particular region where that particular coordinates marked under the text files as shown above, are present.

And after that we would have rough idea and when we will loop over all the data points, we would be able to categorize each one of these points as belonging to or not belonging to that particular region.

3.4 Mathematical

The mathematical portion of the project mainly relies on computing the value of the angle in which will denote the value in which we are facing. This function is a pre- requisite parameter for our regression models. It is also known as the bearing angle and its value lies between -180 to 180 degrees (or $-\pi$ to $+\pi$ in radians).

```
In [105]: # The formula used is the following:
#          $\vartheta = \text{atan2}(\sin(\Delta\text{Long}) \cdot \cos(\text{Lat2}),$ 
#          $\cos(\text{Lat1}) \cdot \sin(\text{Lat2}) - \sin(\text{Lat1}) \cdot \cos(\text{Lat2}) \cdot \cos(\Delta\text{Long}))$ 

def get_direction(lat1,lon1,lat2,lon2):
    lon1=lon1.to_numpy()
    lat1=lat1.to_numpy()
    lon2=lon2.to_numpy()
    lat2=lat2.to_numpy()
    diff_lon_rad = np.deg2rad(lon2-lon1)
    x=np.sin(diff_lon_rad)*np.cos(lat2)
    y=np.cos(lat1)*np.sin(lat2)-(np.sin(lat1)*np.cos(lat2)*np.cos(diff_lon_rad))
    initial_bearing=np.arctan2(x,y) #the theta angle that we are looking for

    initial_bearing=np.degrees(initial_bearing) #as the value of initial bearing is between -180 to 180 degrees
    direction=(initial_bearing+360)%360
    return direction
```

Figure 3.16: - This function is used to compute the direction in which we are facing by using simple trigonometric functions using NumPy library

```
In [106]: # Basically formula for Haversine is as follows
#  $a = \sin^2(\Delta\text{LatDifference}/2) + \cos(\text{Lat1}) \cdot \cos(\text{Lat2}) \cdot \sin^2(\Delta\text{LonDifference}/2)$ 
#  $c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$ 
#  $d = R \cdot c$ 

def haversine(lat1,lon1,lat2,lon2):
    lat1=np.deg2rad(lat1.to_numpy())
    lon1=np.deg2rad(lon1.to_numpy())
    lat2=np.deg2rad(lat2.to_numpy())
    lon2=np.deg2rad(lon2.to_numpy())
    #haversine formula
    diff_lat=lat2-lat1
    diff_lon=lon2-lon1
    a=np.sin(diff_lat/2)**2+np.cos(lat1)*np.cos(lat2)*np.sin(diff_lon/2)**2
    c=2*np.arcsin(np.sqrt(a))
    r=6372.8 #radius of earth in km
    return np.around(c*r,decimals=2)
```

Figure 3.17: - The Haversine function is used to compute the distance between two points over a geographical area given their latitudes and longitudes

```
In [112]: def in_airport(x1,y1,x2,y2,airport):
          for icoord, (x,y) in enumerate(zip([x1,x2], [y1,y2])):
              point = Point(float(x), float(y))
              found = 0
              polygon = Polygon(airport)
              if polygon.contains(point) == True:
                  found = 1
              return found
          return found
```

Figure 3.18: - The Point class which is provided by Shapely module is used for marking regions near the airports

The positions which are near the airports have been marked by **Points** class of Shapely module and those points which are within the boundary of that particular area are marked as true (or 1) otherwise they are marked as 0.

```
In [113]: df['JFK']=df.apply(lambda x: in_airport(x['dropoff_latitude'], x['dropoff_longitude'],
df['LGA']=df.apply(lambda x: in_airport(x['dropoff_latitude'], x['dropoff_longitude'],
df['NWK']=df.apply(lambda x: in_airport(x['dropoff_latitude'], x['dropoff_longitude'],
```

Figure 3.19: - The lambda function has been applied on all the points to verify whether a particular pickup point lies under the boundary of that airport or not

Apart from that, in cleaning stage, we need to heed to some of the edge cases, like we need to identify whether: -

- There exists a negative fare amount for any given ride
- The passenger count is less than 1
- There exists a NaN or empty value
- There exists a redundant feature like the 'key'

We will take care of all of these steps in the cleaning stage of our project.

Apart from that, EDA (Exploratory Data Analysis) also forms a crucial part of mathematical interpretation of our model which we are going to discuss now in the statistical portion.

3.5 Statistical

The statistical section of our project mainly relies on the outputs and the inferences drawn under the EDA (Exploratory Data Analysis) which is nothing but the statistical interpretation of our dataset for better understanding of hidden patterns.

The first and foremost task is to map the correlation between different features of our dataset in order to get a clear cut idea about the definite relationships that they have with each other.

Our modified and updated dataset looks something like this: -

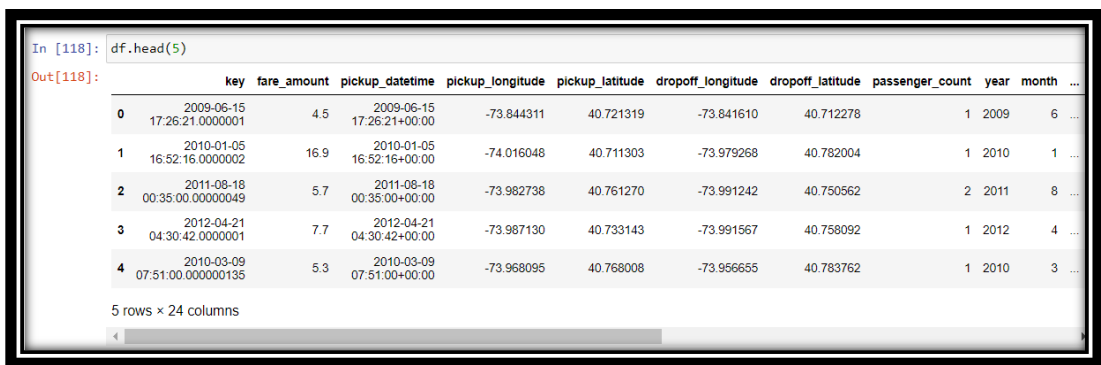


Figure 3.20: - The number of features is increased from 8 to 24 in view of the inclusion of new features (actually 23 as 'key' acts as redundant data)

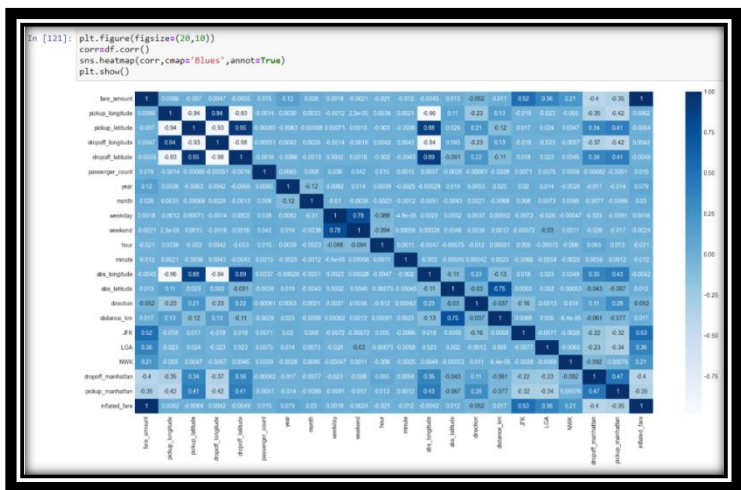


Figure 3.21: - Correlation heat map for features. The one in dark shades are highly correlated

But evaluating the correlation of every feature with every other feature seems to be a redundant task so; we would like to apply a more succinct approach.

As our main target variable is **inflated_fare** amount, so we would like to have a bar plot considering the correlation of fare amount with every other available

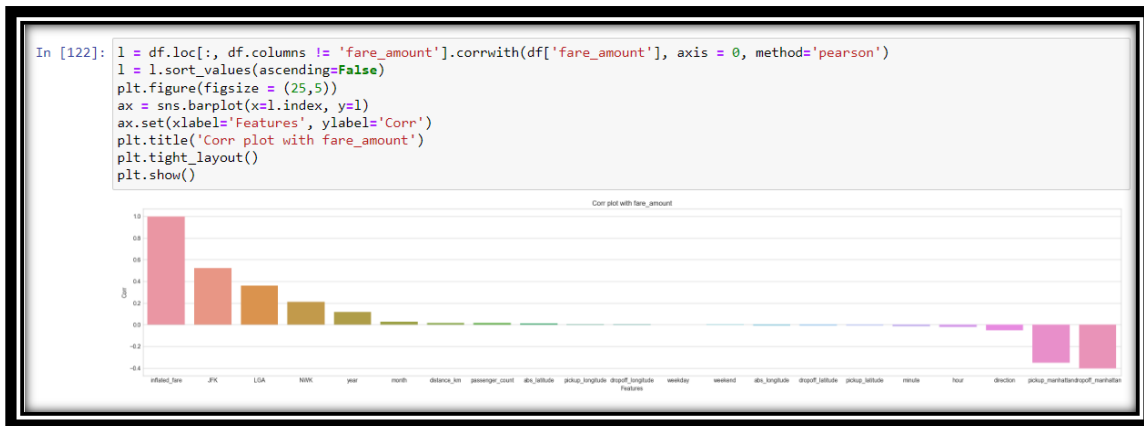


Figure 3.22: - Bar plot which accurately signifies the positive, negative and zero correlation between the features and fare amount considered after inflation

In the bar plot, the length of the bar signifies the magnitude of correlation, and the direction with respect to x and y axis signifies whether it is positively or negatively correlated.

The coefficient of correlation used here is Pearson coefficient which is defined by the following formula,

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient
 x_i = values of the x-variable in a sample
 \bar{x} = mean of the values of the x-variable
 y_i = values of the y-variable in a sample
 \bar{y} = mean of the values of the y-variable

Figure 3.23: - Pearson's correlation coefficient

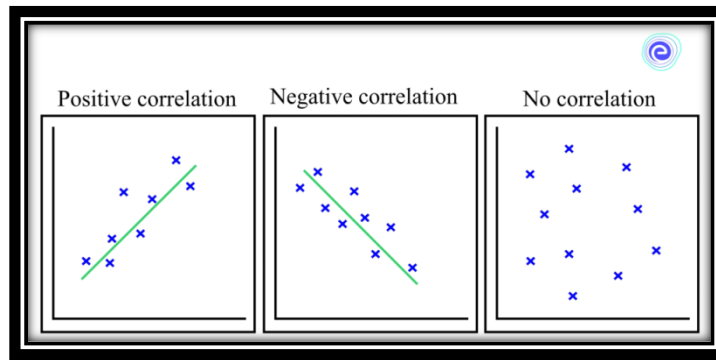


Figure 3.24: - Positive, negative and no correlation

Different types of correlations as described in the above figure are as follows: -

- i.) **Positive Correlation:** - It signifies a relationship between two variables where if the value of one increases, the value of other also increases, but maybe at a different pace.
- ii.) **Negative Correlation:** - It signifies a relationship between two variables where if the value of one increases, the value of other decrease, but maybe at a different pace.
- iii.) **No Correlation:** - It signifies a relationship between two variables where the increase or decrease in the value of one variable has no direct or apparent effect on the value of other variable.

The above plot signifies that among the airports **JFK** is most positively correlated with our inflated_fare amount followed by **LaGuardia** and **Newark**.

The **dropoff_manhattan** and **pickup_manhattan** are most negatively correlated with our inflated fare amount.

As the pickup and drop off locations were by default given in UTC [**Abbr. 3.6**] format, we had to split it up in terms of years, months, weekdays, hours, minutes and seconds.

On the basis of this we have devised several graphs corresponding to different parameters.

The parameter corresponding to y-axis would be fare amounts and corresponding on x-axis would be any date time parameter like hours, minutes, seconds, weekdays, months etc.

It would give us a definite idea about the variation of histograms with respect to different date time durations.

Chapter 4: - EXPERIMENTS AND RESULT ANALYSIS

Experimental result analysis is a specialized field that gives coaches and players unbiased data to observe or analyze the performance based on certain experimental inferences.

Systematic observation, which gives accurate, trustworthy, and comprehensive information on performance, serves as the foundation for this procedure.

There are total of six models that we have used for prediction purposes, namely the **Linear Regression**, **SVR (Support Vector Regression)**, **Random Forests**, **XGB (Extreme Gradient Boosting)**, **Voting Regressor** and **Bagging Regressor**.

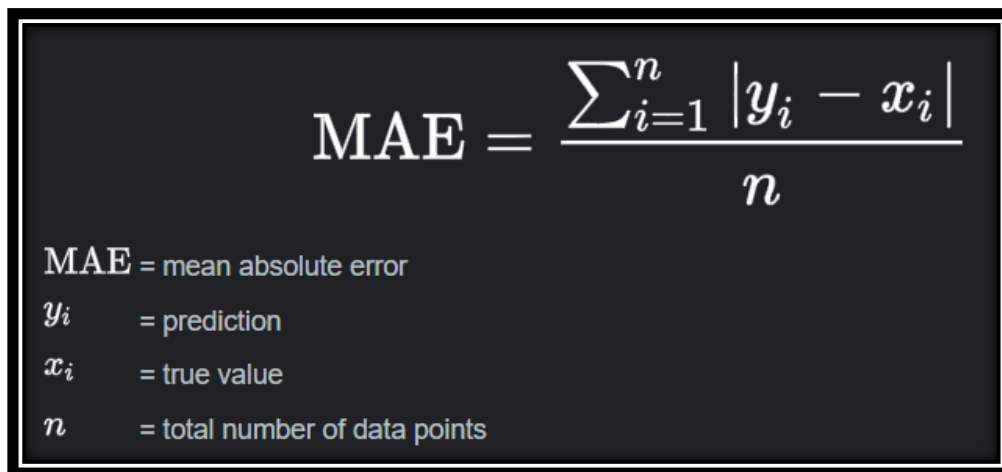
We will take each of these regressions one by one and analyze the performance based on a total of five evaluation metrics.

NOTE: - All the results generated in the subsequent models as shown below are fluctuating as the input seeded to these models has a random state between **37** and **40**

4.1 Evaluation Metrics Formulations

Some general formulations corresponding to evaluation metrics are described below.

4.1.1 Mean Absolute Error: - It measures the absolute difference between the predicted and actual values of a set of data.


$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

MAE = mean absolute error
 y_i = prediction
 x_i = true value
 n = total number of data points

Figure 4.1: - Formula for Calculating Mean Absolute Error

4.1.2 Mean Squared Error (MSE): - It measures the sum of square of differences between actual and predicted values.

Formula

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error
 n = number of data points
 Y_i = observed values
 \hat{Y}_i = predicted values

Figure 4.2: - Formula for Calculating Mean Squared Error

4.1.3 Root Mean Squared Error (RMSE): - It is just a derived formulation with having square root over the entire calculated MSE value.

Formula

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^N (x_i - \hat{x}_i)^2}{N}}$$

RMSE = root-mean-square deviation
 i = variable i
 N = number of non-missing data points
 x_i = actual observations time series
 \hat{x}_i = estimated time series

Figure 4.3: - Formula for Calculating Root Mean Squared Error

4.1.4 Root Mean Squared Logarithmic Error (RMSLE): - The Root Mean Squared Logarithmic Error (RMSLE) is the same as RMSE except that it adds a logarithmic function around both the predicted and the observed values and adds a 1 to them for avoiding the occurrence of 0 as natural log of 0 is undefined.

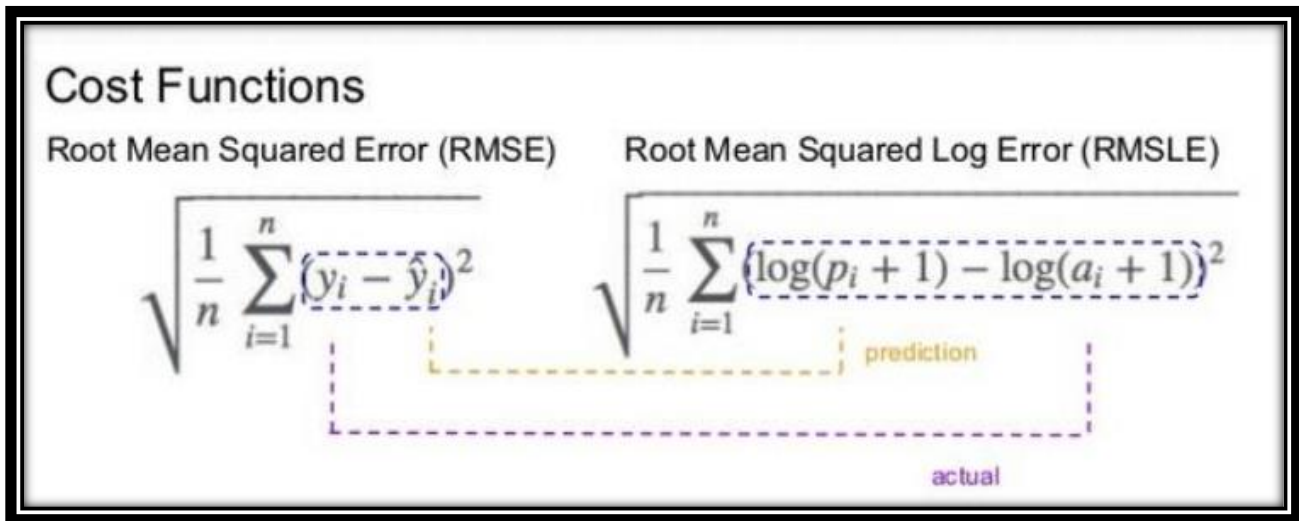


Figure 4.4: - Difference Between RMSE and RMSLE

4.1.5 R-Squared Error: - R-Squared is basically how good your model is in comparison to the average difference between sum of squares of original and mean value. The numerator is comprised of basically your residual while the denominator is composed of the average sum of squares corresponding to the regression problem.

The negative value of R-Square signifies the fact that the average score is better than the error score as prediction made by your model while the positive score signifies the complementary fact.

$$\begin{aligned} \text{Coefficient of Determination} &\rightarrow R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \\ \text{Sum of Squares Total} &\rightarrow SST = \sum (y - \bar{y})^2 \\ \text{Sum of Squares Regression} &\rightarrow SSR = \sum (y' - \bar{y}')^2 \\ \text{Sum of Squares Error} &\rightarrow SSE = \sum (y - y')^2 \end{aligned}$$

Figure 4.5: - Formulation of R-Squared Error

4.1.6 Generalized function for evaluation metrics: - We have incorporated a generalized function for calculating and storing the evaluation metrics in a dictionary for the purpose of EDA on the final results.

The function is as follows: -

```
In [100]: def evaluation_metrics(y_test,predictions, model, modelObj=None, input_test=None):
          if(model=='Neural Network'):
              loss, mse, rmse, mae, msle, rSquared = modelObj.evaluate(input_test, y_test, verbose=2, batch_size=batch_size)
              print('Mean Absolute Error:', mae)
              print('Mean Squared Error:', mse)
              print('Root Mean Squared Error:', rmse)
              print('Root Mean Squared Log Error:', np.sqrt(msle))
              print('R Squared Score:', rSquared)
          else:
              mae = metrics.mean_absolute_error(y_test, predictions)
              print('Mean Absolute Error:', mae)
              mse = metrics.mean_squared_error(y_test, predictions)
              print('Mean Squared Error:', mse)
              rmse = np.sqrt(mse)
              print('Root Mean Squared Error:', rmse)
              rmsle = np.log(rmse)
              print('Root Mean Squared Log Error:', rmsle)
              rSquared = metrics.r2_score(y_test,predictions)
              print('R Squared Score:', rSquared)
          rootMeanSquareDict[model] = rmse
          meanAbsoluteDict[model] = mae
          rSquaredDict[model] = rSquared
```

Figure 4.6: - evaluation_metrics generalized function

4.2 Linear Regression: - In linear regression, we try to accommodate a best fit line that minimizes the SSE [Abbr. 4.1] or RMSE (Root Mean Squared Error).

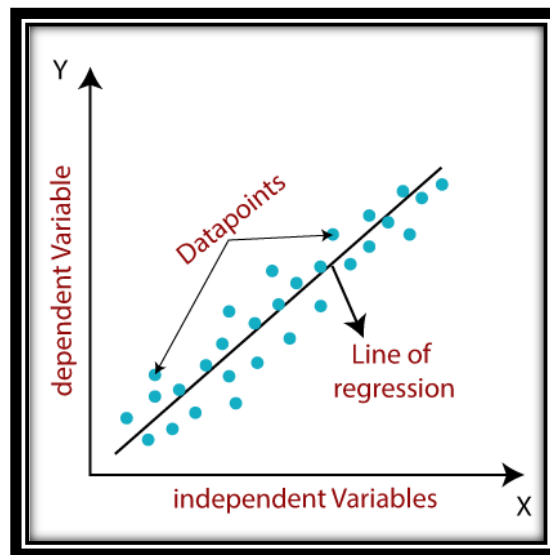


Figure 4.7: - Best fit line as devised under linear regression

The diagram shows the linear regression equation $Y_i = \beta_0 + \beta_1 X_i$ enclosed in a double-bordered box. Arrows point from labels to the corresponding parts of the equation: 'Constant/Intercept' points to β_0 , 'Independent Variable' points to X_i , 'Slope/Coefficient' points to β_1 , and 'Dependent Variable' points to Y_i .

Figure 4.8: - The linear regression line is defined by the regression coefficients β_0 and β_1

The value of β_0 and β_1 is as follows: -

The formulas for determining the regression coefficients are shown in a double-bordered box with a blue background. The formula for β_1 is $\beta_1 = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$ and the formula for β_0 is $\beta_0 = \frac{\sum y - \beta_1 \sum x}{n}$.

Figure 4.9: - Formulae for determining the regression coefficients β_0 and β_1

For the purpose of generating the model, we have used pipeline feature of Sklearn which works in the intended fashion as that of a typical generalized pipeline of model construction as mentioned in many theoretical contexts which includes the scaling method and afterwards the main model that we are going to use.

A typical example showcasing the use of pipeline method is as follows: -

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
model = LinearRegression()
pipe = Pipeline(steps=[('scaler', scaler), \
                        ('model', model)], memory='tmp')
```

Figure 4.10: - Use of sklearn.pipeline.Pipeline feature to create a virtual pipeline of events

```

scaler=StandardScaler()
model=LinearRegression(n_jobs=-1)
pipe=Pipeline(steps=[('scaler',scaler),('model',model)],memory='tmp')

beginTime=time.time()
pipe.fit(x_train,y_train)
predictions=pipe.predict(x_test)
print('Total processing time taken by Linear Regression is %s seconds'%(time.time()-beginTime))

Total processing time taken by Linear Regression is 1.7113978862762451 seconds

Evaluation Metrics for Linear Regression

evaluation_metrics(y_test,predictions,'LinearRegression')

Mean Absolute Error: 3.9976635237505347
Mean Squared Error: 42.667853028504126
Root Mean Squared Error: 6.532063458701556
Root Mean Squared Log Error: 1.8767228901102557
R Squared Score: 0.4947087775535346

```

Figure 4.11: - Construction of linear regression model and corresponding evaluation based on evaluation metrics

The RMSE corresponding to Linear Regression model is that of approximately **6.532** while the Mean Absolute Error is **3.997**, and R-Squared Error which is **0.4947** which is average at best. Being one the most basic and the fastest algorithms that are devised for regression, it is not performing up to the mark.

So, in order to reduce the value of evaluation metrics (or to increase the value of R-Squared Error) we need to take into picture, more complex regression models such as that of SVR (Support Vector Regression) and Random Forests.

4.3 SVR (Support Vector Regression)

Support vector regression is the regression form of SVM (Support Vector Machine) that tries to fit the line (called as the best-fit-line) within the constraints of the epsilon tube.

The epsilon here denotes the marginal distance between the positive and negative hyper planes and the corresponding best fit line.

The errors or outlier points are present outside the range of both of the positive and negative hyper planes and in between them is our regression line or the best-fit- line.

The distance between either of the planes and the regression line is denoted by epsilon (ϵ) which is also the radii of the tube.

The tube-like structure denotes the maximum capability of our model to tolerate or resist the outliers that are not defined or included under our model.

A typical illustrative design of SVR in a graphical representation is as follows: -

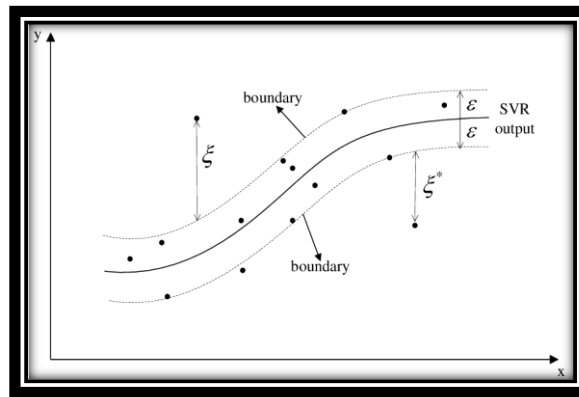


Figure 4.12: - A typical diagram of an epsilon SVR (Support Vector Regression) tube that signifies the marginal distance between the regression line and either of the hyper planes

```
svrClassifier = SVR(kernel='rbf',C=3,epsilon=0.1)
scaler = StandardScaler()
pipeline = Pipeline(steps=[('scaler', scaler),('SVR',svrClassifier)],memory='tmp')
beginTime = time.time()
pipeline.fit(x_train,y_train)
finalPredictions = pipeline.predict(x_test)
print('Total processing time taken by SVR is %s seconds'%(time.time()-beginTime))

Total processing time taken by SVR is 472.47876143455505 seconds
```

Evaluation Metrics for Support Vector Regression

```
evaluation_metrics(y_test,finalPredictions,'SVR')

Mean Absolute Error: 3.2679850955461895
Mean Squared Error: 35.62480814266628
Root Mean Squared Error: 5.9685212109621
Root Mean Squared Log Error: 1.7865211265526482
R Squared Score: 0.5781155699631009
```

Figure 4.13: - Construction of SVR model and corresponding evaluation based on evaluation metrics

The RMSE corresponding to SVR model is that of approximately **5.968** while the Mean Absolute Error is **3.267**, and R-Squared Error which is **0.578** which is slightly better than the linear regression model but still not up to the mark.

4.4 Random Forest Regression Model

The random forest is an ensemble learning technique that falls under the category of bagging method. It utilizes the power of multiple decision trees in order to make a prediction.

To decrease variation within a noisy dataset, ensemble learning techniques like bagging, often referred to as bootstrap aggregation, are frequently utilized.

In bagging, a random sample of data from a training set is picked with replacement, which allows for multiple selections of the individual data points.

These weak models are then trained independently using many data samples, and depending on the task—for example, classification or regression—the average or majority of those predictions result in a more accurate estimate.

The random forests can be used as a classifier or a regressor. In a classifier, the final result is provided by the **Majority Voting**.

But in case of regressor, it is obtained by taking the mean or average of all the computed results taken from each of the individual decision trees.

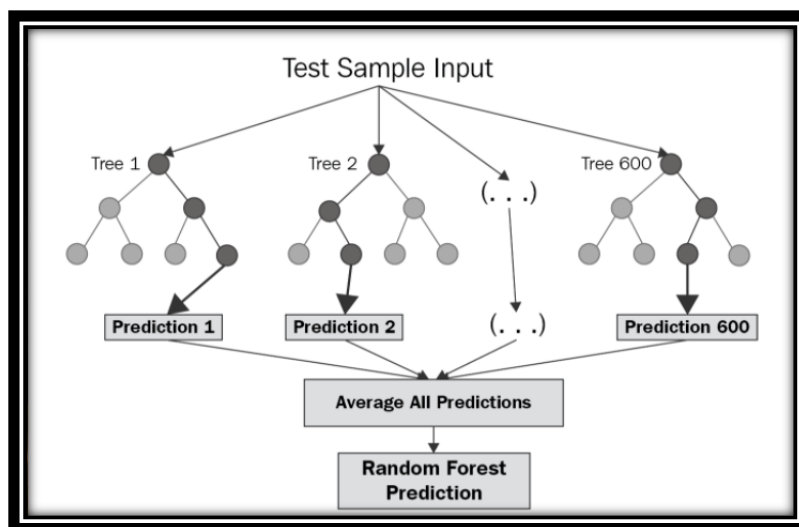


Figure 4.14: - A typical Random Forest Regressor

```
scaler = StandardScaler()
randomForest = RandomForestRegressor(n_estimators=2000,random_state=40,n_jobs=-1)
pipeline = Pipeline(steps=[('scaler',scaler),('randomForest',randomForest)],memory='tmp')
start_time = time.time()
pipeline.fit(x_train,y_train)
finalPredictions = pipeline.predict(x_test)
print('Total processing time taken by Random Forest Regressor is %s seconds'%(time.time()-beginTime))

Total processing time taken by Random Forest Regressor is 1040.8194539546967 seconds

Evaluation Metrics for Random Forest Regressor

evaluation_metrics(y_test,finalPredictions,'RandomForest')

Mean Absolute Error: 1.9316324005719687
Mean Squared Error: 20.422756568604594
Root Mean Squared Error: 4.519154408581831
Root Mean Squared Log Error: 1.5083248985955326
R Squared Score: 0.7581448584867176
```

Figure 4.15: - Construction of Random Forest Regressor model and corresponding evaluation based on evaluation metrics

Here the `n_estimators` parameter signifies the number of decision trees that our Random Forest model is using for creation of Random Forest Regressor.

The RMSE corresponding to Random Forest model is that of approximately **4.519** while the Mean Absolute Error is **1.931**, and R-Squared Error which is **0.758** which is quite good. For obtaining more accuracy, a bagging model like the random forest, needs more data which is outside the scope of the device constraints.

4.5 XGB (Extreme Gradient Boosting) Regression

It is an advanced form of Random Forests and instead of bagging uses the boosting technique of ensemble learning for prediction of the continuous values.

While both bagging and boosting converts a set of weak learners into strong learners, bagging achieves it through parallel computation and processing of all the weak learners and boosting uses the concept of sequential learning and processing. Moreover, boosting also takes into account the power and robustness of gradient boosting in order to achieve efficient result in less amount of time.

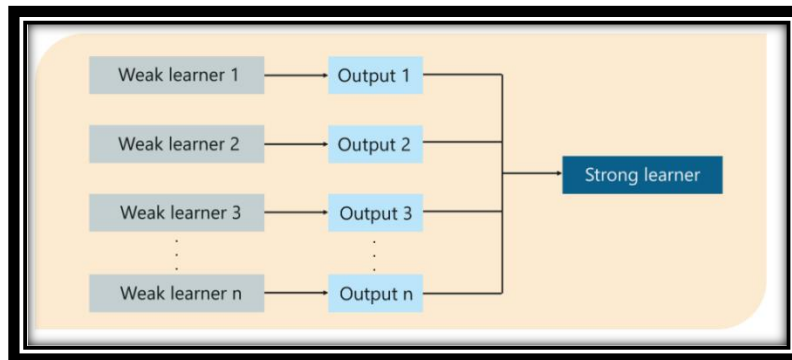


Figure 4.16: - The boosting method as used by XGB (Extreme Gradient Boosting)

```
scaler = StandardScaler()
xgbRegressor = xgb.XGBRegressor(learning_rate=0.15,gamma=0.5,max_depth=10)
pipeline = Pipeline(steps=[('scaler',scaler),('xgb',xgbRegressor)],memory='tmp')
beginTime = time.time()
pipeline.fit(x_train,y_train)
finalPredictions = pipeline.predict(x_test)
print('Total processing time taken by XGB Regressor is %s seconds'%(time.time()-beginTime))
Total processing time taken by XGB Regressor is 16.17334771156311 seconds

Evaluation Metrics for XGB Regressor

evaluation_metrics(y_test,finalPredictions,'XGB')
Mean Absolute Error: 1.8821662955917713
Mean Squared Error: 21.505257870130052
Root Mean Squared Error: 4.637376183805887
Root Mean Squared Log Error: 1.5341487286673494
R Squared Score: 0.7453254085466825
```

Figure 4.17: - Construction of XGB model and corresponding evaluation based on evaluation metrics

The learning rate, the gamma values and the max_depth are hyper parameters here. The gamma value defines the minimum loss reduction that is required for further partition and max_depth is used for determining the maximum depth of decision tree that would be used in XGB.

The RMSE corresponding to XGB regressor model is that of approximately **4.637** while the Mean Absolute Error is **1.882**, and R-Squared Error which is **0.745** which is quite good. For obtaining more accuracy, a boosting model like XGB also needs more data which is outside the scope of the device constraints.

The time taken by XGB is **16.17 seconds** while that of Random Forests is nearly **1040 seconds** which is approximately **17.33 minutes**.

From this data, we can easily verify how much faster XGB is as compared to Random Forests, even though both of them compute the result with nearly the same accuracy.

4.6 Voting Regressor

Voting regression technique is also a method of ensemble learning which utilizes the power of various machine learning models to create a better regressor model that finally undergoes voting to verify the best possible answer.

A typical voting regressor is as follows: -

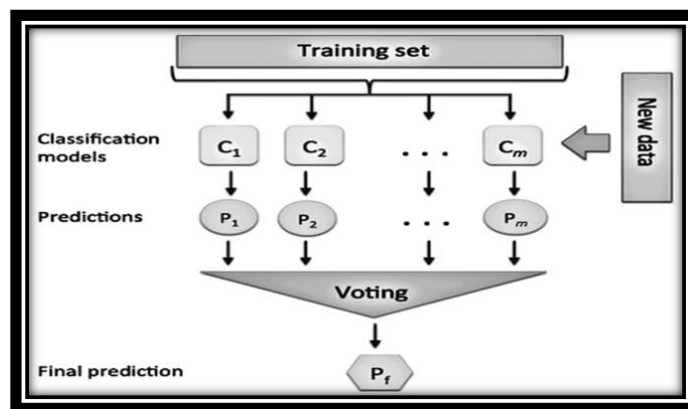


Figure 4.18: - A typical voting regressor model

Just like Random Forest, it also finds out the mean of all the predicted results and shows that as the final output.

The voting regressor is based on the fact that several classification models instead of being trained separately can be trained in a combined fashion and then voting is the done in the form of collecting the mean value of all the estimated results.

The voting regressor is one of the best models when it comes to the situation where the robustness of one model and the speed or efficiency of other model is required.

The predictions made by the voting regressor model on our dataset are as follows: -

```
scaler = StandardScaler()
votingRegressor = VotingRegressor([('randomForest',randomForest),('xgbRegressor',xgbRegressor),('svrClassifier',svrClassifier)])
pipeline = Pipeline(steps=[('scaler',scaler),('votingRegressor',votingRegressor)],memory='tmp')
beginTime = time.time()
pipeline.fit(x_train,y_train)
finalPredictions = pipeline.predict(x_test)
print('Total processing time taken by Voting Regressor is %s seconds'%(time.time()-beginTime))
Total processing time taken by Voting Regressor is 1427.8425126075745 seconds

Evaluation Metrics for Voting Regressor

evaluation_metrics(y_test,finalPredictions,'VotingRegressor')
Mean Absolute Error: 2.0614573616939715
Mean Squared Error: 21.17596547987652
Root Mean Squared Error: 4.601735051029831
Root Mean Squared Log Error: 1.526433417385152
R Squared Score: 0.7492250318603366
```

Figure 4.19: - Construction of voting regressor model and corresponding evaluation based on evaluation metrics

The models that we have considered under voting regressor are as follows: -

- i.) Random Forest
- ii.) XGB (Extreme Gradient Boosting)
- iii.) SVR (Support Vector Regression)

The constructor of voting regressor class takes a tuple of models as a parameter as shown above.

The RMSE corresponding to voting regressor model is that of approximately **4.601** while the Mean Absolute Error is **2.061**, and R-Squared Error which is **0.7492** which is almost accurate as Random Forests and XGB, but has a huge runtime of **1427.82** seconds which is near 23 minutes.

4.7 Bagging Regressor

The bagging regressor is yet another ensemble learning technique which takes into account a base model and then it allocated that base model over some k subsets of our data. The final predictions derived from these k subsets are then aggregated by taking the mean of all the available results. It falls under the bagging technique of ensemble method, just as random forests, the only difference being that, while random forests take into account, a decision tree as their core base model, the bagging regressor can consider any known regression model as one of its base model.

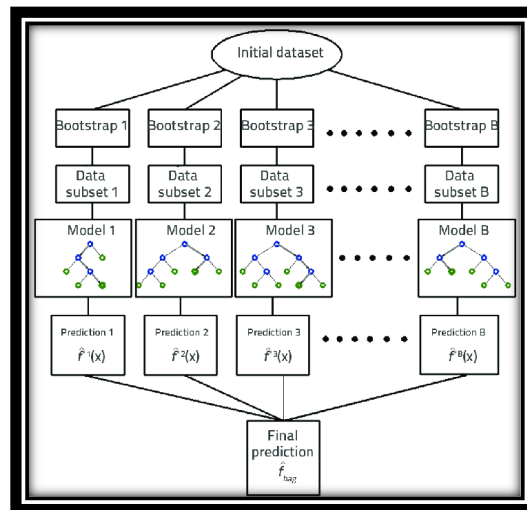


Figure 4.20: - A typical representation of the working of Bagging model in ensemble learning

The predictions and the values of the evaluation metrics as obtained under bagging regressor are obtained as shown below: -

```
scaler = StandardScaler()
baggingRegressor = BaggingRegressor(base_estimator=xgbRegressor,n_estimators=5,random_state=40)
pipeline = Pipeline(steps=[('scaler',scaler),('baggingRegressor',baggingRegressor)],memory='tmp')
beginTime=time.time()
pipeline.fit(x_train,y_train)
finalPredictions = pipeline.predict(x_test)
print('Total processing time taken by Bagging Regressor is %s seconds'%(time.time()-beginTime))
Total processing time taken by Bagging Regressor is 51.15070295333862 seconds

Evaluation Metrics for Bagging Regressor

evaluation_metrics(y_test,finalPredictions,'BaggingRegressor')
Mean Absolute Error: 1.8437971212327784
Mean Squared Error: 19.60656986487372
Root Mean Squared Error: 4.4279306526721625
Root Mean Squared Log Error: 1.48793235362894
R Squared Score: 0.7678104954475746
```

Figure 4.21: - Construction of bagging regressor model and corresponding evaluation based on evaluation metrics

The bagging regressor takes into account **n_estimators** parameter which denotes the number of subsets in which we have to divide our dataset and corresponding to that the number of base models that we require for that purpose.

Of all the regression models that we have applied so far, the bagging regressor outperforms all with the base model being XGB (Extreme Gradient Boosting). That's why it took only a mere **51 seconds** to complete the whole operation.

The RMSE corresponding to voting regressor model is that of approximately **4.4279** while the Mean Absolute Error is **1.843**, and R-Squared Error which is **0.7678** which is probably the most accurate score we have obtained so far.

4.8 HistGradientBoosting Regressor

In machine learning, the HistGradientBoosting gradient boosting algorithm is used for classification and regression applications. It shares similarities with other gradient boosting algorithms like XGBoost and LightGBM, but also differs in a few key ways that make it especially effective for big datasets.

Histogram-based gradient boosting is the main distinction between HistGradientBoosting and other gradient boosting techniques. This indicates that rather than doing the boosting operation directly on the individual data points, the algorithm discretizes the feature space into histograms. Because it requires fewer calculations to update the gradients at each step, this method can be substantially faster than other gradient boosting algorithms.

HistGradientBoosting also has the benefit of using an early halting method to help avoid over fitting. Every time a boosting iteration is performed, the algorithm analyses the validation loss and terminates training when the loss stops decreasing. By doing so, the model is better able to generalize to new data and avoid being over fit to the training set of data.

HistGradientBoosting is a versatile technique that may be applied to a variety of datasets because it handles both category and numerical data. It has the ability to handle missing values and automatically encrypt categorical data using a number of methods, including one-hot encoding and target encoding.

Early stopping is used in HistGradientBoosting to train models for regression and classification problems. It can handle both numerical and categorical data and is especially well suited for huge datasets. It is a versatile and effective tool for machine learning because of its capacity to prevent over fitting and support for missing values and categorical input.

The predictions and the values of the evaluation metrics as obtained under bagging regressor are obtained as shown below: -

```
scaler = StandardScaler()
beginTime = time.time()
histGradientBoostingRegressor = HistGradientBoostingRegressor().fit(x_train, y_train)
finalPredictions = histGradientBoostingRegressor.predict(x_test)
print('Total processing time taken by Hist Gradient Bagging Regressor is %s seconds'%(time.time()-beginTime))

Total processing time taken by Hist Gradient Bagging Regressor is 1.1213388442993164 seconds

Evaluation Metrics for HistGradientBoosting Regressor

evaluation_metrics(y_test, finalPredictions, 'HistGradientBoostingRegressor')

Mean Absolute Error: 1.923075918045883
Mean Squared Error: 19.970923532807
Root Mean Squared Error: 4.4688392474083
Root Mean Squared Log Error: 1.497138696183887
R Squared Score: 0.7634956612760528
```

Figure 4.22: - Construction of HistGradientBoosting regressor model and corresponding evaluation based on evaluation metrics

The RMSE corresponding to HistGradientBoosting regressor model is that of approximately **4.468** while the Mean Absolute Error is **1.9230**, and R-Squared Error which is **0.7634** which is again as nearly accurate as Random Forests and XGB.

Being extremely fast and an optimized version of traditional Gradient Boosting algorithm, it merely takes 1.12 seconds to complete the prediction process.

4.9 Artificial Neural Network (ANN)

In regression tasks, ANNs are used to predict a continuous output variable based on a set of input variables. The process of training an ANN for regression involves adjusting the weights of the connections between neurons so that the network can learn to approximate the underlying function that maps the input variables to the output variable.

The feed forward neural network is a popular ANN type used for regression analysis. An input layer, one or more hidden layers, and an output layer make up this kind of network. The anticipated output variable is produced by the output layer, which also receives the input variables. The calculations required to convert the input variables into the anticipated output variable are carried out by the hidden layers.

The network is initially initialized with random weights before being trained for regression using a feed-forward neural network. Following that, the network is shown the input-output pair-based training

data. For each input, the network computes its output, and an error value is determined by comparing the predicted and actual outputs.

The weights in the network are modified using the back propagation algorithm to bring the predicted output values closer to the actual output values. The procedure is repeated until the error on the training data is below a predetermined threshold or for a predetermined number of iterations.

Here are the steps that have been taken while formulating the prediction using Sequential ANN using Keras library.

```
Using standard scaler as a preprocessor to ANN

In [62]: scaler = StandardScaler()
x_train_dl[x_train_dl.columns] = scaler.fit_transform(x_train_dl[x_train_dl.columns])
x_test_dl[x_test_dl.columns] = scaler.fit_transform(x_test_dl[x_test_dl.columns])

Manually implementing R-Squared Metric as it is not provided by compile function of Keras model

In [63]: def r2_score(y_true, y_pred):
u = K.sum(K.square(y_true - y_pred))
v = K.sum(K.square(y_true - K.mean(y_true)))
return (1 - (u / v))
```

Figure 4.23: - Construction of Standard Scaler for Neural Network Model

```
In [64]: def ann_model():
model = Sequential()

#first Layer
model.add(Dense(48, activation='relu', input_dim=19))

#subsequent Layers
model.add(Dense(96, activation='relu'))
model.add(Dense(48, activation='relu'))
model.add(Dense(24, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(6, activation='relu'))

#Last Layer
model.add(Dense(1, activation='relu'))

ann_optimizer = opt.RMSprop()
model.compile(loss='mae', optimizer = ann_optimizer, metrics=[tf.keras.metrics.MeanSquaredError(),
tf.keras.metrics.RootMeanSquaredError(),tf.keras.metrics.MeanAbsoluteError(),
tf.keras.metrics.MeanSquaredLogarithmicError(), r2_score])
return model
```

Figure 4.24: - Construction of Deep Learning ANN Model

Here we have used ReLU as an activation function. ReLU [Abbr. 4.2] is a commonly used activation function in artificial neural networks, particularly in deep learning models. It is a simple function that applies a non-linear transformation to the input, mapping negative values to zero, and leaving positive values unchanged.

ReLU is popular because it is computationally efficient, easy to implement, and has been shown to be effective in many different types of neural networks. Additionally, ReLU helps to address the vanishing

gradient problem that can occur in deep neural networks, which can prevent learning from taking place effectively.

However, one potential downside of ReLU is that it can lead to "dead neurons" in the network, where the neuron always outputs zero, due to a negative bias in the input. This can be mitigated by using variants of ReLU, such as Leaky ReLU or Parametric ReLU, which allow for a small positive output for negative inputs, or by using a different activation function altogether.

Here we have also used the concept of Early Stopping to stop our epoch iteration after the point where no significant improvement in efficiency is apparently visible.

```
In [93]: # create a model
regression_ann_model = ann_model()

# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

# train model
epochs = 60
batch_size = 32

start_time = time.time()
history = regression_ann_model.fit(
    x_train_dl, y_train, epochs=epochs, batch_size=batch_size, callbacks=[early_stop])
finalPredictions = regression_ann_model.predict(x_test_dl)

print('Total processing time taken by Artificial Neural Network is %s seconds'%(time.time()-beginTime))
```

Figure 4.25: -Running ANN based on early stopping procedure and corresponding evaluation based on evaluation metrics

```
Evaluation Metrics for Neural Network

In [101]: evaluation_metrics(y_test, finalPredictions, 'Neural Network', regression_ann_model, x_test_dl)

748/748 - 2s - loss: 2.1825 - mean_squared_error: 25.9539 - root_mean_squared_error: 5.0945 - mean_absolute_error: 2.1825 - mea
n_squared_logarithmic_error: 0.0905 - r2_score: 0.7300 - 2s/epoch - 2ms/step
Mean Absolute Error: 2.1825034618377686
Mean Squared Error: 25.95389747619629
Root Mean Squared Error: 5.094496726989746
Root Mean Squared Log Error: 0.30075955003018057
R Squared Score: 0.7299590110778809
```

Figure 4.26: - The final evaluation result of Neural Networks

As seen from the above figure, the RMSE corresponding to Artificial Neural Network regressor model is that of approximately **5.094** while the Mean Absolute Error is **2.182**, and R-Squared Error which is **0.729** which is just a little less than Random Forest and other ensemble learning techniques that we applied previously.

It so happens because Neural Networks tend to train on a million of rows and it is outside the device constraints to let the model train on such a large amount of data, which may otherwise have boosted the efficiency from 0.729 of R-Squared error to nearly about 0.9.

4.10 Extended EDA for Evaluation Metrics

Now that we have applied all of our regression models on our datasets, it's time for us to statistically analyze their performances using the evaluation metrics through which we have determined the performance of each of the following models.

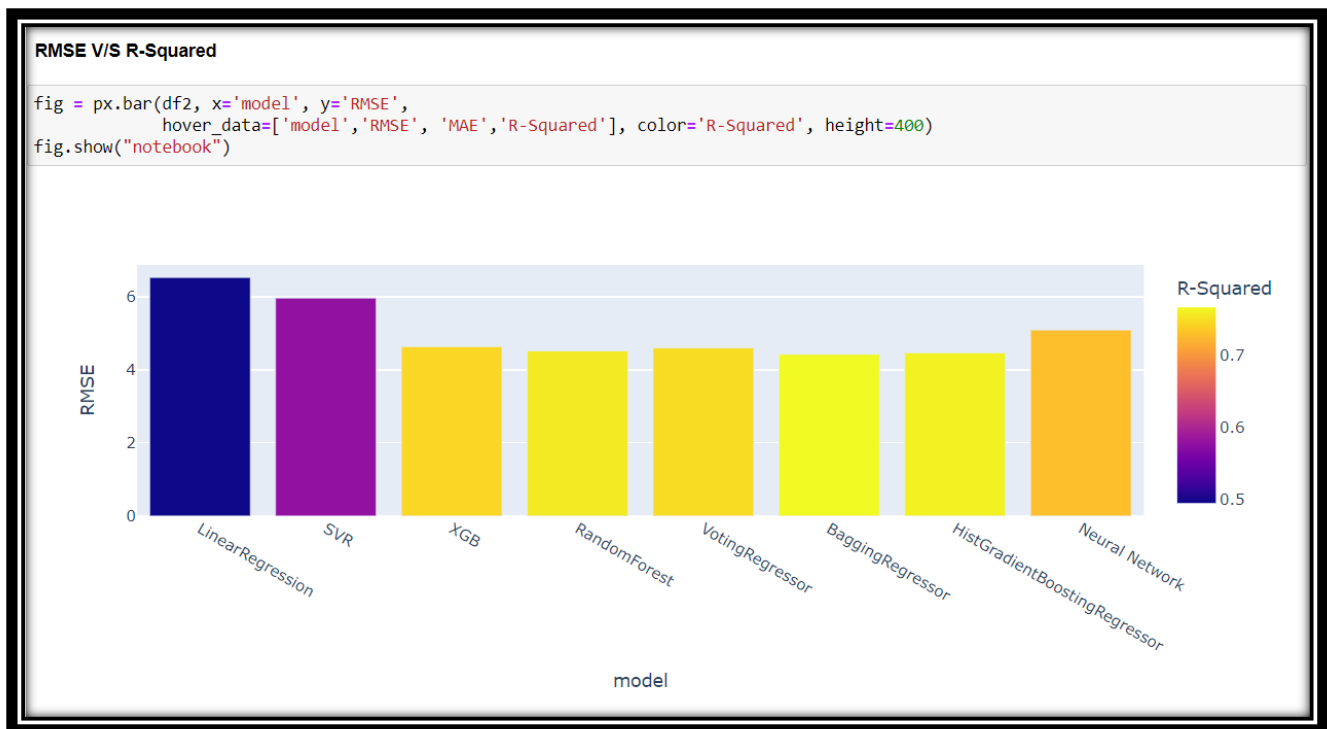


Figure 4.27: - The RMSE v/s R-Squared plot

By observing the data, we can see that **Linear Regression** has the worst error figure while the **Bagging Regressor, XGB and Random Forest** almost perform equally well in terms of RMSE.

And as R-Squared is inversely proportional to RMSE, so the models with poor RMSE score (higher value of RMSE) will obviously have corresponding lower value of R-Squared Error.

The pie plots present below divides the whole set of regression models on the basis of their share in the total RMSE and R-Squared errors obtained so far using all the models combined together.

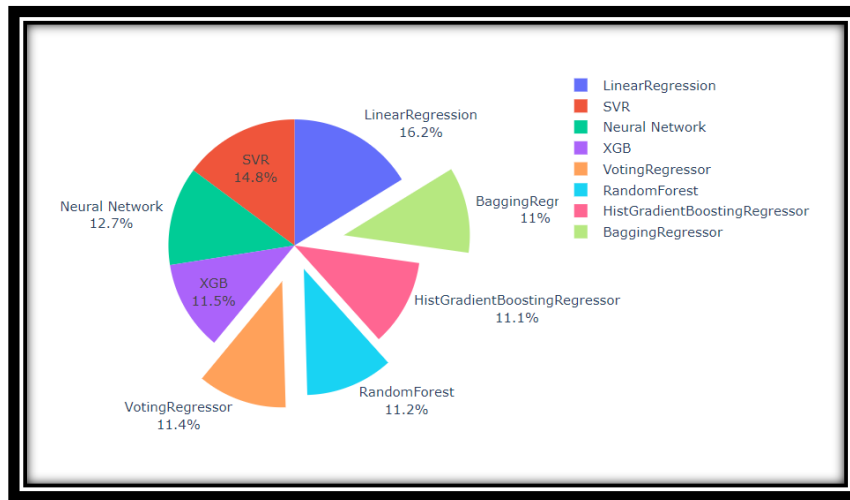


Figure 4.28: - Pie Chart corresponding to distribution of RMSE

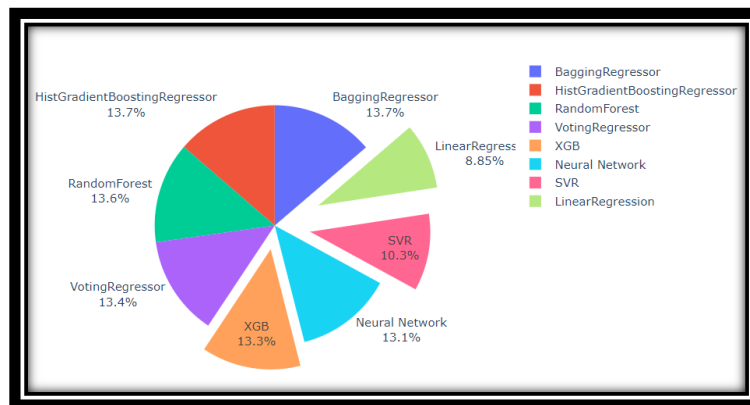


Figure 4.29: - Pie Chart corresponding to distribution of R-Squared Error

4.11 K Fold Cross Validation

As already mentioned before, K-Fold Cross Validation is a method of describing the validation by splitting the whole dataset into k parts, in which we train by k-1 parts and test by the remaining part.

An array of results is obtained after performing this operation and the function that we have employed for this purpose is `cross_val_score` which is provided under the `sklearn.model_selection` module.

The parameter `cv=5` denotes the number of cross validation sets or simply the value of k which in this case is 5. It simply means that we need to take into account 5 values for any of the evaluation metrics that we are trying to compute whether it is an RMSE or R-Squared Error.

As the k-fold cross validation will be applied for 5 times for each of the k data subsets, we have only applied it for Linear Regression as a means of demonstration.

The low specifications of our device have bounded us to test it over the Linear Regression and XGB model only in order to avoid crashing of system.

```

XGB (Extreme Gradient Boosting)

In [123]: model = xgb.XGBRegressor(learning_rate=0.15,gamma=0.5,max_depth=10)

Root Mean Squared Error

In [124]: beginTime=time.time()
score = cross_val_score(model, x_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
print('Time taken to compute Root Mean Square Error for XGB model is %s seconds'%(time.time()-beginTime))
Time taken to compute Root Mean Square Error for XGB model is 41.404168128967285 seconds

In [125]: for i in score:
print(-i,end=" ")

4.463262362368249 4.12841845998133 4.181103597283813 4.550858018366846 3.9757004928917308

```

Figure 4.30: - RMSE computation under K-Fold

```

R-Squared Error

In [126]: score = cross_val_score(model, x_train, y_train, scoring='r2', cv=5)

In [127]: for i in score:
print(i,end=" ")

0.7479452887923277 0.7918695440094901 0.7877820202409157 0.7610004992039835 0.7969448148858513

```

Figure 4.31: - R-Squared computation under K-Fold



Figure 4.32: - Generalized working of K-Fold Cross Validation with k=5

Chapter 5: - CONCLUSIONS

5.1 Conclusions

The major inference drawn from this project is that we were able to predict taxi fare prices near to a RMSE of **4.22** and R-Squared Error of about **0.79** which indicates an accuracy of about **79%** on approximate scale. It can be further optimized by taking the large data set but unfortunately we have considered only **90k** points out of a total of **55 million** extended rows due to the **RAM** and device constraints.

5.2 Future Scope

The major challenge that is up ahead in this project is the inclusion of **scaling factor** and how it can modify our results. As of now, we have only focused upon the simple regression models or some complex ensemble learning techniques which yielded some quite good results considering the amount of data set points that we inputted. Inclusion of lasso and ridge regression is one more thing to look forward to.

Along with that, it would also be a nice idea to look up to different techniques by which we would be able to handle large data as the one that we have currently faced. Use of efficient data processing libraries like **Dask** or **Vaex** is good but in the long run they don't help much in terms of providing the desired functionality which is fulfilled by Pandas.

5.3 Applications

The project of ours can find use in many of the scenarios as mentioned below: -

- i.) In the apps that are driven by real-life problems like the geo-location apps such as Maps, Tom-Tom API and much more for traffic analysis.
- ii.) Can be used by Traffic Surveillance Authorities and unicorn start-ups or companies like Uber to monitor the effect of increasing fare prices along with the inflation and the increased number of passengers.
- iii.) Can also be used by taxi passengers themselves for self- evaluation of the fare amount that they have spent in total within a particular year.
- iv.) Majorly the usage of this project is to predict the fare prices given any position of coordinates of pickup and drop off locations which will also be valid for the upcoming decades due to the inclusion of inflation.

REFERENCES

A) Books

[1] Andreas C. Muller, Sarah Guido, "Introduction to Machine Learning With Python". Sebastopol, California, USA: Shroff, O'Reilly Media, 2016.

[2] Geron Aurelien, "Hands-On Machine Learning with Scikit-Learn and Tensor Flow: Concepts, Tools, and Techniques to Build Intelligent Systems". Sebastopol, California, USA: O'Reilly Media, 2017.

B) Conferences and Conference Meetings

[1] Vanajakshi, L., S. C. Subramanian, and R. Sivanandan. "Travel time and fare prediction under heterogeneous traffic conditions using global positioning system data from buses." IET intelligent transport systems 3.1 (2009): 1-9.

[2] Wu, Chun-Hsin, Jan-Ming Ho, and Der -Tsai Lee. "Travel-time and fare prediction with support vector regression." IEEE transactions on intelligent transportation systems 5.4 (2004): 276-281.

[3] Yildirimoglu, Mehmet, and Nikolas Geroliminis. "Experienced travel time prediction for congested freeways." Transportation Research Part B: Methodological 53 (2013): 45-63.

[4] Biagioni, James, et al. "Easytracker: automatic transit tracking, mapping, and arrival time prediction using smart phones." Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, ACM, 2011.

C) Datasets

[1] Google LLC, "nyc-taxi-fare-prediction" Google-Coursera, Kaggle, "Playground Prediction Competition, 2016.

D) Journals

[1] Jamal I. Daoud, "Multicollinearity and Regression Analysis" Journal of Physics, International Islamic University Malaysia, Kuala Lumpur, Malaysia, Conf. Ser. 949 012009, 2017.

APPENDIX

A.1 LightGBM

An open-source package called Light Gradient Boosted Machine, or **Light GBM** for short, offers a practical and fast implementation of the gradient boosting technique.

By using a sort of autonomous feature selection and concentrating on boosting cases with greater gradients, LightGBM expands the gradient boosting technique. This can hasten training significantly and enhance prediction performance.

As a result, when using tabular data for regression and classification predictive modelling tasks, LightGBM has established itself as the de facto method for machine learning contests. As a result, together with Extreme Gradient Boosting, it bears some of the responsibility for the improved acceptance and widespread use of gradient boosting techniques in general (XGBoost).

A.2 Lasso Regression

It is a type of regularisation method used in regression problems. For a more accurate forecast, it is preferred over simple regression techniques without regularization.

Shrinkage is used in this model. When data values shrink toward the mean, this is referred to as shrinkage. Simple, sparse models are encouraged by the lasso procedure.

The coefficients of the remaining features are decreased to zero and a random feature is chosen from the highly linked ones.

Additionally, as the model parameters vary, the selected variable varies at random. Generally speaking, ridge regression performs better than this.

A.3 Intelligent Transportation System (ITS)

The use of sensing, analysis, control, and communications technology in ground transportation to increase security, mobility, and effectiveness is known as an intelligent transportation system. An intelligent transportation system consists of a variety of applications that process and exchange information to reduce traffic, enhance traffic management, lessen the impact on the environment, and boost the advantages of transportation for both business users and the general public.

The use of information and control technologies in the operation of transportation systems is the technological foundation of intelligent transportation systems (ITS).

Communications, automated control, and computer hardware and software are some of these technologies. It takes expertise from several engineering fields, such as civil, electrical, mechanical, industrial, and their allied disciplines, to apply these technologies to transportation.

The majority of transportation issues are brought on by a lack of timely and accurate information as well as by a lack of system-wide coordination amongst employees.

Therefore, information technology's positive contribution is to provide improved knowledge to enable system participants to reach mutually beneficial decisions.

A.4 Ridge Regression

In situations when the independent variables are strongly correlated, ridge regression is a technique for estimating the coefficients of multiple-regression models. It has been used to a variety of disciplines, including engineering, chemistry, and econometrics.

This technique carries out L2 regularization. Predicted values differ much from real values when the problem of Multicollinearity arises, least-squares are unbiased, and variances are significant.

It is optimally a better and a more robust method than lasso regression.

Ordinary least squares (OLS) regression is the analysis technique used to assess the association between independent variables (Features) and a dependent variable (Target). By minimising the sum of squares in the difference between the observed and predicted values of the dependent variable, the method predicts ties.

Ridge regression, on the other hand, refers to a type of linear regression model where the coefficients are estimated using a biased estimator rather than the Ordinary Least Squares (OLS) estimator and have a smaller variance.