A Comparative Analysis of Different Machine Learning Models for Flight Fare Estimation Use Case

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology (B. Tech)

in

COMPUTER SCIENCE AND ENGINEERING WITH SPECIALIZATION IN INFORMATION SECURITY

by

MISHAL THOMAS SABU

17BCI0174

MOHIT MAYANK

17BCI0118

Under the guidance of

Prof. Usha K

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT, Vellore

MAY 2021

DECLARATION BY THE CANDIDATE

I hereby declare that the thesis entitled "**A COMPARATIVE ANALYSIS OF DIFFERENT MACHINE LEARNING MODELS FOR FLIGHT FARE ESTIMATION USE CASE**" submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering with specialization in Information Security to Vellore Institute of Technology, Vellore is a record of bonafide work carried out by me under the supervision of **Prof**. **Usha K.**

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of the Candidate(s)

Name: Mishal Thomas Sabu

Reg. Number: 17BCI0174

Signature of the Candidate(s)

Name: Mohit Mayank

Reg. Number: 17BCI0118

Place : Vellore

Date :

CERTIFICATE

This is to certify that the thesis entitled "**A COMPARATIVE ANALYSIS OF DIFFERENT MACHINE LEARNING MODELS FOR FLIGHT FARE ESTIMATION USE CASE"** submitted by **Mishal Thomas Sabu, 17BCI0174 & Mohit Mayank, 17BCI0118** to VIT, for the award of the degree of **Bachelor of Technology** in **Computer Science with Specialization in Information Security** is a record of bonafide work undertaken by them under my supervision during the period, 01.01.2021 to 31.05.2021, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date: 31/05/2021

Signature of the Guide

Internal Examiner (s) **External Examiner (s)**

Dr. Prabu Sevugan

Head of Department, Information Security

B. Tech (Computer Science with spec. in Information Security)

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our Honorable Chancellor *Dr. G. Viswanathan,* Vice-Presidents *Mr. Sankar Viswanathan, Mr Sekar Viswanathan* and Mr. *G. V. Selvam,* Vice Chancellor *Dr. Anand Samuel* and Pro-Vice Chancellor *Dr. S. Narayanan* of VIT University, Vellore for providing the opportunity and facilities to carry out our research work in this esteemed institution.

We would like to express our special thanks of gratitude to our mentor and guide Prof. Usha K, who gave us the golden opportunity to do this wonderful and enlightening project on Flight Fare Estimation using Machine Learning and who moderated this project and channeled us to do research by digging deep into the Machine learning domain. We had the opportunity to learn the vast topic by building the models from scratch.

We acknowledge the role research papers and online journals which helped us to get idea about the research efforts done in ML field. We would also like to thank our parents and friends who helped us in finalizing this project within the limited time period.
We would like to thank Dr. Prabu Sevugan, Head of the Department, Information Security for his kind help, support, interaction and extending laboratory and database facilities.

Mishal Thomas Sabu, 17BCI0174

Mohit Mayank, 17BCI0118

Place : Vellore

Date : 31/05/2021

EXECUTIVE SUMMARY

Flights are an important mode of transportation in our present society. Today the flight prices depend on a variety of factors and changes on an hourly basis. Our project deals with comparing publicly available datasets and evaluates various models commonly used for flight fare estimation. The machine learning models used are Random Forest Regression, Decision Tree Regressor, Bagging Tree Regressor, Gradient Boosting Regressor and Weighted Combination of three regressors. Each dataset we studied contain various unique attributes which affect flight fare. We also analyzed how the unique features affects the models to estimate the flight fare. Research project highlighted the robustness of random forest regression model in all the datasets and how the trends in the data affect the various models. A Web application was designed using python and flask to estimate and display flight fare.

Tags: Flight Fare Estimation, Random Forest Regression, Decision Tree Regressor, Bagging Tree Regressor, Gradient Boosting Regressor and Weighted Combination of Decision Tree, Bagging Tree and Gradient Boosting Regressors

CONTENTS

List of Figures

Figure No. **Title** **Page No.**

List of Tables

Table No. Title **Page No.**

List of Abbreviations

BTR	Bagging Tree Regressor
D.T	Decision Tree
DTR	Decision Tree Regressor
GBR	Gradient Boosting Regressor
IDE	Integrated Development Environment
ML	Machine Learning
R2	R squared
RFR	Random Forest Regressor
RMSE	Root Mean Square Error
Qtr	Quarter
Yr	Year

Symbols and Notations

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to

1. INTRODUCTION

1.1 OBJECTIVE

In today's world aero-planes are the fastest means to travel across continents. It is commonly used by people to travel around the world to explore new places and visit loved ones. The world we live in today is a world based on profits. The prices of flight tickets to travel around the world keep on fluctuating and never stays the same. Many factors such as demand, distance to travel, type of seating, etc. influences the price of flight tickets and causes it to fluctuate. The objective of the project is to develop a comparative analysis of different machine learning models for flight fare use cases based on DB1B Market dataset to estimate flight fare.

COVID-19 pandemic hit worldwide, uniting all countries into one entity with sole ambition to wipe the dreaded virus away from its face. The impact on travel industry was devastating with restrictions imposed by many countries kept people at their homes and not willing to venture outside, not even within the country where they are residing.

Lockdowns forced many airlines minimize their operations and in extreme cases to shutdown operations. Global outbreak was a shock to the booming industry and it became necessary to adapt to the 'new normal' in order to survive. Outbreak prompted airports to adopt new measures to ensure safety, hygiene and risk management, of employees and passengers in conjunction with airlines operating in the sector. Travel Industry has responded to challenges and is emerging as resilient model to drive nation's economy and global competitiveness.

Small and large companies have adopted the new norm of "online meeting" by acknowledging the fact that it is not necessary to travel for every meeting. Most probably 50% of business related travels will be curtailed in the post COVID era. The business models of some high cost, long-haul flight operators are faltering and in turn terminating many transatlantic flight operations.

2020 was a game changer for all and the greatest challenge was faced by travel industry. Insight into global trends is shown in figure 1.1

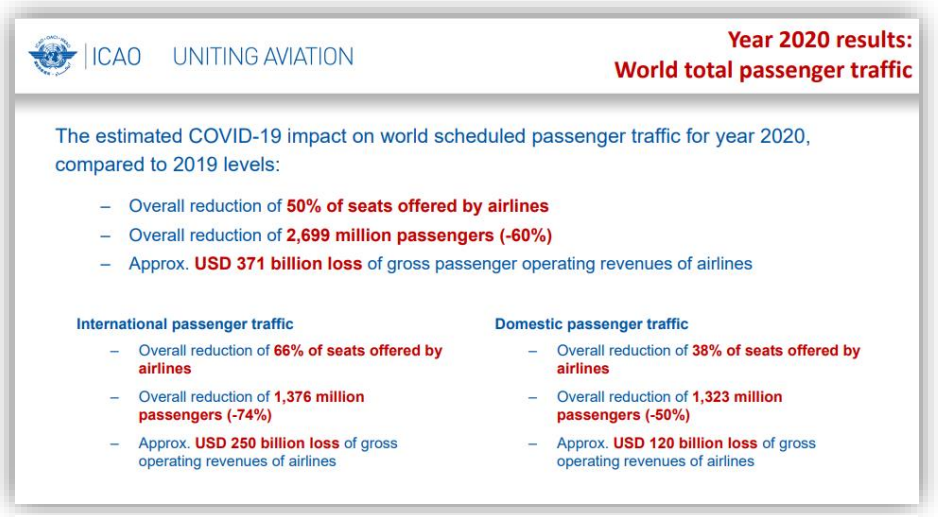


Figure 1 Effects of COVID -19 on Civil Aviation Industry [19]

Some of the companies worldwide took quick and decisive actions by taking measures to reduce its overhead fixed costs and introduced flexible "work from home" policy. For business to evolve beyond COVID –19, they have to take up initiatives that will enable them to reshape the industry in the years and decades to come. Expectations of passengers are evolving continuously based on the on-going COVID situation and the demand for tools and capabilities in numerous frontlines like self-service has grown significantly.

Global scheduled seat capacity comparison over the years to monitor the recovery is depicted in Figure 1.2 below.

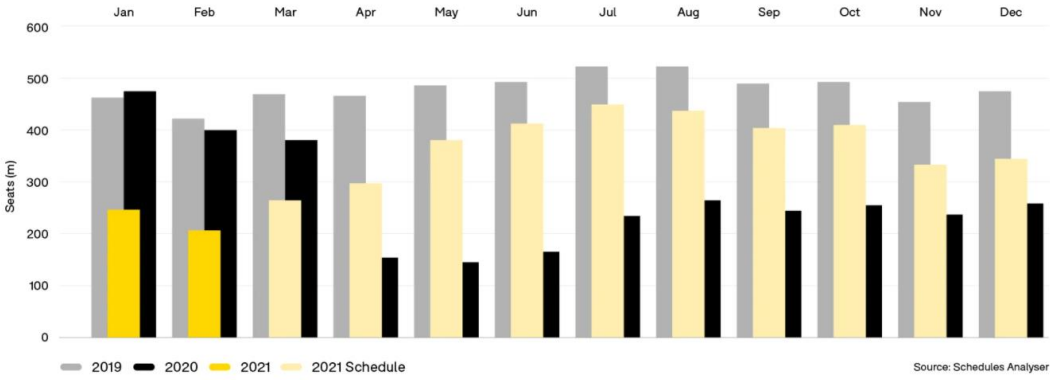


Figure 2: Percentage Change in Capacity Compared to Previous Year to Monitor Recovery against Pre-Pandemic Capacity [14]

Airline Capacity Trend in the recent months continues to fluctuate, with most of the airlines reducing the capacity in alignment with COVID 19 protocol.

Region	vs. Apr-19	vs. Apr-20	vs. Mar-21
Global	-43.0%	73.6%	2.8%
Australia	-36.5%	409.1%	18.9%
Brazil	-61.3%	329.7%	-28.2%
China	5.5%	95.0%	-0.5%
France	-76.0%	173.7%	8.1%
Germany	-81.0%	152.9%	32.1%
India	-19.3%	134.8%	-4.4%
Japan	-51.0%	-9.3%	14.4%
Mexico	-21.6%	83.3%	7.6%
Singapore	-84.5%	131.5%	2.6%
South Africa	-44.8%	115.2%	14.8%
South Korea	-43.3%	41.7%	-0.2%
Spain	-75.5%	251.8%	31.1%
UAE	-53.1%	288.0%	2.6%
United Kingdom	-89.8%	0.9%	13.6%
USA	-31.2%	52.8%	1.6%

Region	20/05/2019	18/05/2020	10/05/2021
Global	-41.8%	90.0%	3.2%
Australia	-31.4%	629.0%	2.0%
Brazil	-53.6%	435.7%	1.5%
China	9.8%	57.9%	1.5%
France	-69.1%	289.3%	4.0%
Germany	-78.4%	137.0%	13.2%
India	-50.4%	-31.3%	-12.8%
Japan	-61.0%	-23.9%	-9.0%
Mexico	-17.1%	505.8%	0.0%
Singapore	-82.9%	289.6%	-1.0%
South Africa	-46.8%	968.8%	7.5%
South Korea	-45.4%	13.4%	-2.0%
Spain	-67.0%	146.9%	10.5%
UAE	-51.2%	237.8%	-5.4%
United Kingdom	-82.4%	6.8%	72.9%
USA	-24.8%	212.4%	1.3%

Source: Schedules Analyser

Figure 3 Capacity Variance by Current Month and Week [14]

Though passenger flight schedules declined due to COVID -19, Cargo Traffic by air had a boost with delivery of critical supplies. With diminished passenger operations, the flights were used as Cargo only operations to serve commerce and industries. These flights were termed as "preighters" with cargo in the belly and even additional cargo in the passenger cabin.

1.2 MOTIVATION

Dynamic pricing was first introduced by American Airlines in 1980s. It is now widely used by companies to respond to demand fluctuations and to drive revenue significantly. In the digital age, passengers can compare prices of flights of different airlines. Price is one of the major factors on which purchasing decisions are made by passengers.

Dynamic Pricing Machine Learning Models are based on concept of delivering the right price for every customer while increasing revenue for the business. Flight predictor by Microsoft, Bing was the first entrant into the market of "Big Data" and it was withdrawn from the market in 2014 from Travel Portal. The other web portals like Kayak Fare Predictor and Skyscanner are in the market offering the forecast options to give a rough idea of flight fares.

Cheapflights and Wego are other search engines used widely by passengers and strong competitors in the market.

Airlines has calculated algorithms to maximize the profits by varying individual ticket fares based on capacity of the plane. The three parameters which is considered by airlines to fix the fare is current demand of a flight, current number of seats and timing of the booking.

Airways always overbook the seats to maximize revenue by ensuring there are no empty seats. Everyday each flight has passengers who do not show up. Factoring this aspect of passengers, airlines overbook the flights so that each flight has maximum number of passengers as close as the capacity of the flight or seats on the plane.

The revenue earned by airlines when the passenger has not turned up is more than the cost of the seat when it flies empty. Because no-show passenger has already paid the ticket price. When a flight is full and when there are more passengers who are willing to purchase the ticket, the passengers turned away resorts to competing airlines to travel. Airways tries to accommodate the demand by overbooking so that difference between demand, no-shows and supply does not hurt the airlines or his reputation.

1.3 BACKGROUND

OAG has tweeted that there will be increase in demand for flight seats and will exceed the sixty million barrier over rolling twelve months. Europe has increased the seat capacity in flights by 20%. South East Asia has shown 10%, Southwest pacific 11% and North Africa has shown 13% growth in the capacity. Still the it is way below 60 % of the pre-pandemic capacity of flight seats.

Summer months of 2021 is expected to thaw the “freeze” set in the air travel, airlines loosening the restrictions imposed due to COVID-19 like opening up middle seats that was blocked during the pandemic in United States. Also some airlines are considering to restart serving food and snacks to passengers, which was stopped to limit contact between flight personnel and passengers.

Accurate data enables better decision making for passengers and efforts to bring transparency in the demand will help the industry to come out of red.

1.4 LITERATURE SURVEY

Table 1 Literature Survey

Index	Research Paper	About	Advantages	Disadvantages
1	Airfare prices prediction using machine learning techniques [2]	The article is attempting to predict the air fare in Greece by investigating the features which influence or affect the airfare. The articles gave detailed brief about the four phases of the study such, selection of features that influenced the airfare, the flight data collection for the training and testing of the ML models, comparison of various ML Models and finally the evaluation of the experimental models. The article goes on to compare the accuracy of models when using various features to determine the important factors when predicting airfare.	1. We are able to see how various features affect airfare.	1.Only certain features are being considered and the others are not being taken into account 2. 1817 data entries are being used to train and validate the models. Which does not seem like enough data to create valid models
2	An Airfare Prediction Model for Developing Markets [3]	The research paper focused on Vietnam aviation scene to evaluate the features that have significant impact on airfare fluctuations. Predictions based on developed countries aviation industry is not sensitive in giving a precise prediction model which is relevant to developing	We see how well different models accurately predict airfare. We see that there is a significant increase in accuracy when using weights to combine the Random Forest model and Multilayer Perceptron model.	Only Two models are combined to improve accuracy. We do not know the effects of combining others. Data is gathered only 21 days prior to departure.

Index	Research Paper	About	Advantages	Disadvantages
		countries. The information was gathered 21 days previous to travel. To understand the airfare variation over the days, the price feature was used to create two additional features like fluctuation index and fluctuation amount. The graph plotting these additional new features till the departure date gave insight about the airfare price variations. The Random Forest model and Multilayer Perceptron were combined with the use of weights to improve accuracy.		
3	Predicting the Price of a Flight Ticket with The Use of Machine Learning Algorithms [4]	The research paper aims to identify the price of a ticket given a day based on the data collected from makemytrip.com for the first quarter of 2011 for flights from Bombay to Delhi. This data is fed to variety of machine learning models, like KNN, SVM, Decision tree, to predict the price. After predicting the prices, the values are compared with the actual value and is plotted on a graph based on number of days before departure. This graph is used to study the trend of price with respect to days before departure.	The trend of price is visualized. It predicts price based on readily available data.	The data of the first quarter of 2011 is only being used to predict price. Other Important features like bulk fare and passenger count are not taken into account.

Index	Research Paper	About	Advantages	Disadvantages
4	Random Forests[1]	The notion of random forests is explained in this article, which is a statistical tool that combines tree predictors in which each tree is reliant on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Each Tree classifier depends on the individual tree in the forest and correlation between them. By combining trees to form a forest by considering the random features gives improved accuracy to the model. The mean squared generalization error, which is obtained from the correlation between residuals and the mean squared error of the predictor trees, is bound in random forests for regression. Random split selection Forest, bagging random forest, introduction of random noise into the outputs for lowering generalization error. When randomly selected features are used to split each node, resultant rate of errors can be compared to that of Adaboost as boosting is believed to be a	1. Random Forests' accuracy is on par with Adaboost's, if not even better. 2. It's fairly resistant to outliers and noise. 3. It is quicker than the boosting or bagging methods. 4. It can be used of classification and regression problems It's simple and easily parallelized.	1. For statistical modelers, random forest is a black box method in which they have very little influence over what the model does. 2. For regression problems it cannot predict beyond the range in the training data

Index	Research Paper	About	Advantages	Disadvantages
		special case of random forests.		
5	A Framework for Airfare Price Prediction: A Machine Learning Approach [9]	The datasets used to estimate flight fares here are publicly available datasets. The Office of Airline Information within the United States Bureau of Transportation Statistics collects and maintains two datasets: the DB1B and the T-100 databases (BTS). This information, along with macroeconomic information, is utilized to forecast the quarterly average airfare at the market segment level. Models like LR, SVM, MLP, XGBoost and Random Forest were used to predict the price. Random Forest was found to predict the price 80% accuracy.	1. Shows us that we can predict airfare using market level segment data. 2. Describes the important factors to predict airfare	1. Does not tell us of which particular years data was used. 2. The destination and origin of flight is not considered in predicting price.
6	ACER: An Adaptive Context-Aware Ensemble Regression Model for Airfare Price Prediction [10]	This study offers the ACER context-aware ensemble regression model, which is a hybrid of several context-aware models that adapts context features adaptively. It's a system that can use context-aware data to train basis learners and combine several base learners to make up for the lack of a single basis learner. The trial data was gathered from one of China's most well-known Online Travel Agencies (OTAs). The	1. ACER identifies features that can be used as context information in order to train context-aware models.	1. Does not detail about the data source. 2. Does not R2 score. Large RAM required for training.

Index	Research Paper	About	Advantages	Disadvantages
		were scored based on MAPE. It was shown to out perform a lot of base models like Random Forest, KNN and Bayesian.		
7	Design and implementation of ticket price forecasting system [8]	This article uses Random Forest model in combination with ARMA model to predict air ticket prices. The data was collected up to 60 days in advance to departure in 12-hour intervals. The information gathered included the following: city of departure, destination, ticket purchase date, departure date, ticket selections with prices, and departure time. It uses the model in an system to predict airfare. The system consists of 3 layers. It has view layer as its interface, Functional layer where the forecasting and data management occurs and the data layer is the layer where flight data is stored	1. From the paper we see that the combined model gives better accuracy than standalone models.	1. Contains data from 60 hours in advance. 2. Does not how accurate the models are. The System implementation is not explained but rather represented as abstract parts in a diagram.
8	Creating a prediction model of passenger preference between low cost and legacy airlines [7]	The goal of this study is to forecast passenger preferences between low-cost and legacy airlines. The data was collected using Mturk. Out of the nine factors studied, four were determined to be significant predictors. They were the participant's annual	1. It allows for people's preferences to be determined easily. This type of study allows flight finders to readily give acceptable suggestions to its users.	1. For more accurate predictions sensitive information has to be disclosed. The data was gathered from MTurk and therefore is likely to be biased.

Index	Research Paper	About	Advantages	Disadvantages
		travel frequency, annual income, seat type, and educational level. The study was divided into 2 stages. Stage 1 was regression analysis. Stage 2 to validate the results from stage 1.		
9	To buy or not to buy: mining airfare data to minimize ticket purchase price [6]	This research paper deals with predicting the optimal time to buy tickets for cheap prices. They used models like Hamlet, Q learning, Ripper, Time Series to test its workings. The result of the various tests was that Hamlet was robust and outperformed the other models.	It was able predict prices and tell whether to buy the ticket now or wait the price will decrease.	1. The data used only spanned 21 days in advance. The entries were all from January 2003
10	Airline ticket price and demand prediction: A survey [5]	It is a research paper that is a comprehensive study on existing works. It describes ticket pricing as dynamic and separates the existing models into customer side, uses restricted features extracted from historical data, and airline side model which uses limited internal factors like seat availability, fare class, recent demand etc. It also discusses the strength and weakness of existing works like performance issues and dataset issues. It also suggests using social media-based data to improve ticket	1. Clearly summarizes the existing works. Suggests direction for future improvements	1. Does not talk about models that are a mix of airline side models and customer side model Is a theoretical work.

Index	Research Paper	About	Advantages	Disadvantages
		price and demand prediction		

2. PROJECT DESCRIPTION AND GOALS

The project aims to compare various datasets and to evaluate the models obtained from the datasets to create realistic estimations of historical flight prices. The goal of the project is to understand the trends in the dataset and its features effecting the model.

Anyone who has purchased a plane ticket understands how pricing can fluctuate dramatically. "Revenue management" or "yield management" are sophisticated quasi-academic strategies used by airlines. Over time, the cheapest available ticket for a specific date becomes more or less expensive. This frequently occurs as a result of a desire to increase revenue. -

1. Patterns of buying times (ensuring that last-minute purchases are costly)

2. Keeping the plane as full as possible (increasing the price of a flight that is nearly full in order to limit sales and hold inventory for those expensive last-minute purchases)

Technology adopted by airlines were based on Operation Research framework for yield management, which is unable to respond to competitors' strategy or changing to COVID conditions in real time.

Flexible rules which responds to Customer – Airlines management intervention will usher in flexible models for unbundled ancillaries like differential baggage fare, which would show profitable product combinations. Flexible models that responds to changing market dynamics during COVID times.

Stand-alone models followed by each airlines generate data silos. In order to be market oriented, airlines need to understand how passengers are responding to different pricing models adopted by competitors in the aviation industry. Implementation of machine generated models that learn from real time transactional data will help the policy making decisions. And also segment passengers into right categories and airlines will be able to give personalized offers catering to customers. It would ensure the right price bundle to the loyal customers at the right time. Airlines industry will surge forward from intuition driven to a precise automated data-driven decision making model.

With the roll-out of vaccination programs globally and people are gaining confidence to venture out on “Staycations” and business trips. It is expected that there would be steady increase in demand for domestic & international air travel during the summer months of 2021.

3. TECHNICAL SPECIFICATION

The environment used to run the programs was based on Kaggle.

Kaggle is a Google-owned company that hosts an online community of data scientists and machine learning experts. In a web-based data-science environment, it lets users to search and publish data sets, study and construct models. It provides a cloud-based workbench to run programs. The kernel consists of 4-cores, 16gb RAM, 20GB Dataset space and 5GB Disk Space.

Flask API was used in the development of the web application.

Flask is an API of python with which simple web applications can be build. It is classified as micro web framework as it does not require any tools or libraries. It has no abstraction layer for databases and requires no form validation.

The Python libraries used are:

Numpy or Numerical Python is the homogeneous multi-dimensional array that contains same type elements for example integers or string or characters. It is a fundamental package used for scientific computing in Python language. It gives fast mathematical computations on arrays and matrices.

Sklearn or Scikit-learn features algorithms like Random Forest, SVM’s and k-neighbours.

Pandas is a python library used for data manipulation & analysis. It supports multi-dimensional arrays.

Matplotlib is a Python charting toolkit that may be used to make static, animated, and interactive displays. It has an object-oriented API for integrating charts into apps.

Seaborn is a matplotlib-based library that provides an interface for creating visually appealing and useful statistical visuals.

Glob library is used to retrieve files/ pathnames matching specified pattern.

Prophet library is used to make forecasts for univariate time series datasets, which automatically identifies hyperparameters for the ML model to make forecasts for data with trends and seasonality.

The time library is used in programs to represent time in objects, integers, and texts. It also includes other features, such as waiting during code execution and calculating the code's efficiency.

The python pickle module is used to serialize and de-serialize python objects so that it can be saved on a disk. Serialization occur by converting the object into a character stream.

3.1 COLLECTING DATA SAMPLES

We need historical datasets to work with for any prediction/classification task. Past flight fares for each route were gathered for this project from multiple websites - Bureau of Transportation and Statistics of Unites States, Kaggle and Flight fare prediction hackathon.

Detailed process is mentioned in 4.1.

3.2 CLEANING AND PREPARING DATA

It is concerned with the conversion of data into a form that may be used. In most cases, the raw data received in the first step is considered worthless by machine learning algorithms. Data wrangling is the process of cleaning, manipulating, and mapping data from one form to another so that it can be consumed later in the project life cycle. Missing data imputation, typecasting, handling duplicates and outliers, and so on are all part of this process. (Refer 4.1)

3.3 BUILDING MODEL, MERGING MODELS AND ACCURACY CALCULATION

Modeling is a process of feeding the data attributes into a ML algorithm to train the model.

Having built various models, we compare the models based on performance metrics like time take to train the model, the R2 value, adjusted R2 value, and RMSE value.
In a regression model, R2 is a statistical measure that quantifies the percent of the dependent variable's variation that can be explained by the independent variables.

R2 that has been corrected for the number of predictors in the model is known as adjusted R2. By taking into account how many independent variables are introduced, it can provide a more exact understanding of the correlation that is occurring.

RMSE is the standard deviation of the prediction errors (residuals). This measure is dependent on the scale of the numbers used and hence will vary with different scales of data.

To improve model performance, we combine the models using weights. The weights used can be positive or negative, to counteract enhanced trends.

4. DESIGN APPROACH AND DETAILS

4.1. DESIGN APPROACH / MATERIALS & METHODS

A machine learning pipeline is an end-to-end workflow that incorporates several components of a data-intensive endeavor. After the identification of project objective/ goals, the following steps were followed for model building of the project. The following diagram depicts the ML pipeline or process with various sub-components:

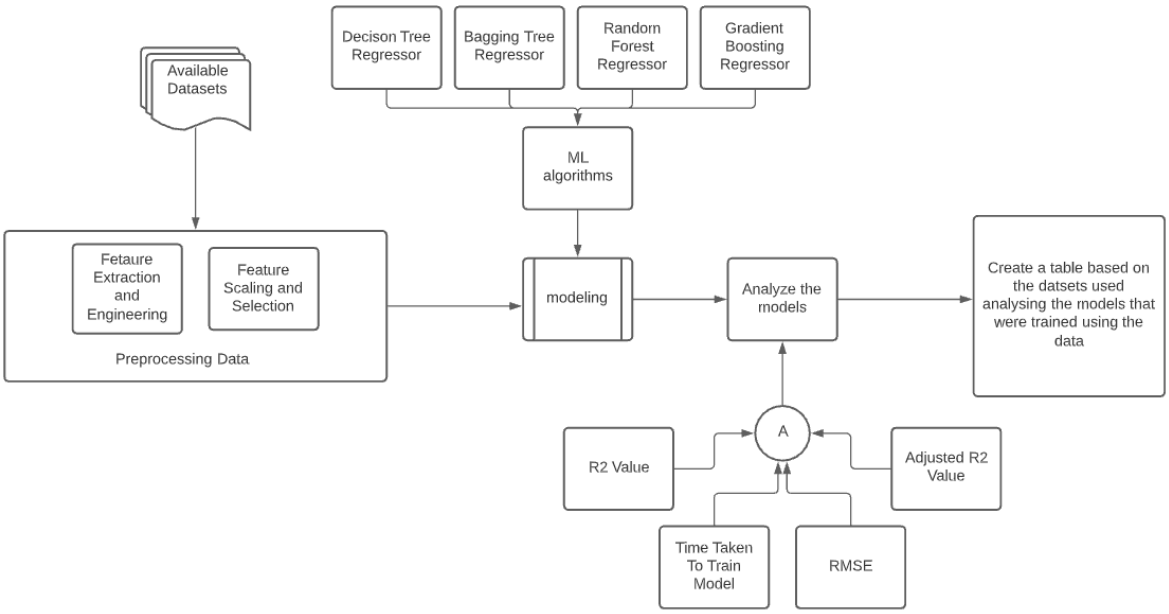


Figure 4 Dataset Analysis Workflow

The datasets used in the projects are publicly available ones. **DB1B dataset [12]** has 25 years of available data of United States of America, all local and international airline routes, as well as code-share partner (foreign) airline routes, and provide coupon-specific information for each

DB1B Survey itinerary (from 1993 up through 2018), aggregated by quarters. It provides patterns of air traffic, air carrier market shares, and passenger flows. It is accessible to public from the website of Bureau of Transportation and Statistics of Unites States. It is a 10% sample of data on airline tickets collected from reporting carriers. [18]

For the project, DB1B market dataset as the base and imported the seat class column from DB1B ticket dataset. In addition to this a new column called pandemic was introduced. It is a binary type column where zero represent not a pandemic and one represents a pandemic situation. Data ranging from Quarter 1 of 2018 – Quarter 3 2020, to & fro of route from California to Florida was used and is named as Normal View. The Normal view dataset was split into 45 sub-routes which contains more than 10, 016 entries each.

Predict the Flight Fare Hackathon [13] is a competition hosted by machinehack.com and contenders were given a dataset with price of flight tickets of various airlines, between various Indian cities over the period ranging from March – June 2019. Machine hack was used to test the accuracy of models.

EaseMyTrip dataset [11] with Indian flight details was downloaded from Kaggle to tune the model performance.

Data processing and wrangling

DB1B Normal view and sub-route view dataset has multiple files, year-wise, quarter-wise and year & quarter wise. All these multiple files were used for building the models.

Feature Engineering and extraction

The data has been preprocessed and wrangled to the point where it can be used by the feature engineering and extraction step. In this step, we'll leverage current characteristics to derive and extract context/use-case-specific attributes or features that will be used by machine learning algorithms in the future. [15] Depending on the type of data, we used various strategies.

Feature scaling and selection

The sheer number of features available could have a negative impact on the overall solution. Not only is processing and handling a dataset with a large number of attributes tough, but it also makes interpretation and visualization complex. As a result, feature selection assists us

in identifying representative sets of characteristics that can be used in the modeling step with little information loss. Data was then split into training and testing data.

Modeling

Modeling is a process of feeding the data attributes into a ML algorithm to train the model. It is an iterative process using multiple algorithms to compare the models based on performance metrics.

Model Evaluation

The models that are used are:

- Decision Tree Regressor: It is a supervised classification model in the form of tree. Dataset is broken down into small subsets while incrementally developing decision tree. The decision of making strategic splits affects the accuracy of the tree. Each tree is a model, complete with branches, nodes, and leaves.

- Random Forest Regressor: It's a supervised learning algorithm which uses ensemble methods (bagging), by constructing multitude of decision trees at training phase and output is generated as mean/ mode of prediction of individual trees. [16]

- Bagging Tree Regressor: It's an ensemble meta-estimator that fits base regressors to random subsets of the original dataset and then combines their individual predictions into a final prediction.

- Gradient Boost Regressor: It's an additive model that enables for the optimization of arbitrary differentiable loss functions by relying on the best possible model, which, when combined with earlier models, reduces total prediction error. If a slight modification in a case's prediction generates no change in error, the case's next target result is zero. [17]

- C1: It consists of the models DTR, RFR and BTR. With the use of only positive weights

- C2: It consists of the models DTR, RFR and BTR. With the use of positive and negative weights

- C3: It consists of the models GBR, RFR and BTR. With the use of only positive weights

- C4: It consists of the models GBR, RFR and BTR. With the use of positive and negative weights

Each model is a data representation that has been generalized, as well as the algorithm that was used to learn this representation. As a result, model evaluation is the process of assessing the performance of a built model against a set of criteria. Model performance is typically a function defined to provide a numerical value that can be used to determine a model's effectiveness. Cost or loss functions are frequently optimized in order to construct a precise model based on these evaluation metrics.

Created a summary table based on the model used. Train and test size, R2 value, root mean RMSE, the weights used, time taken for training models and various other parameters that are dependent on the model used. Comparison of model performance was easy with this table.

It's just as important to prepare and evaluate the model as it is to tune it. Working with various machine learning frameworks/libraries that come with a standard set of algorithms, we rarely use them out of the box.

Deployment and Monitoring

Following the completion of model development and evaluation, as well as multiple iterations to improve the results, the final stage of model deployment was carried out using a flask-based web application.

About the Flight Fare Estimator Web Application:

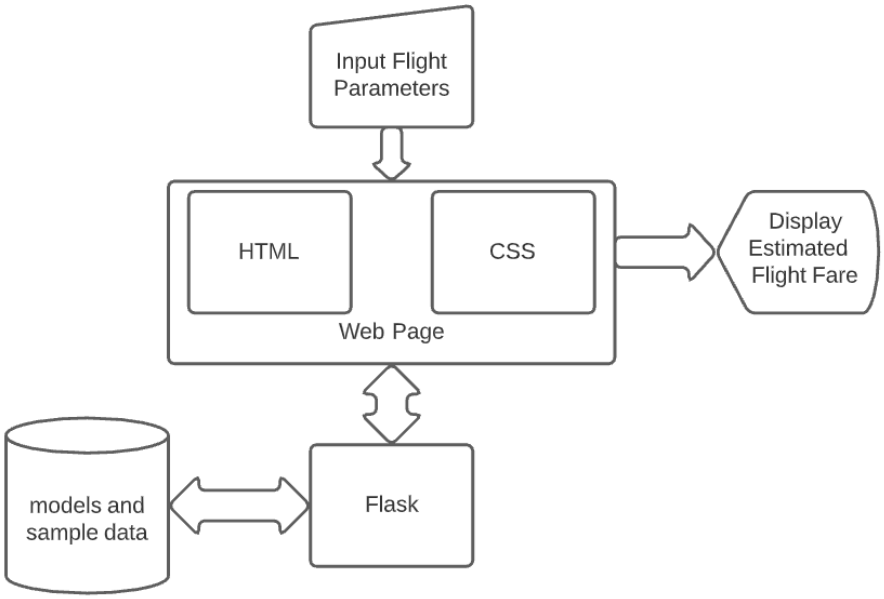


Figure 5 Flight Fare Estimator Web Application Architecture Diagram

The application is built using flask. It renders a webpage that is designed using HTML and CSS. This webpage has a form which contains various flight parameters that need to be entered. After this form is submitted the flask application uses the entered data and transforms it as well as arranges it into a format that can be accepted by the models (these are extracted from the models saved in the pickle format files). The model using the input parameters estimates the flight fare and the flask API renders it on the HTML webpage.

The home() function present in the flask program allows the HTML page to be rendered.

When the form is submitted the predict() function from the flack program is called and is used to estimate the fare.

4.2. CODES AND STANDARDS

For Random Forest Regressor the model training code is:

```
from sklearn.ensemble import RandomForestRegressor

def rf(X_train, y_train,<parameters to fill summary table>):

    start = time.time()

    model = RandomForestRegressor().fit(X_train, y_train)

    stop = time.time()
```

```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

Figure 6 Random forest training function for the DB1B Normal View

For Decision Tree Regressor the model training code is:

from sklearn.tree import DecisionTreeRegressor

def dt(X_train, y_train, ,<parameters to fill summary table>):

start = time.time()

model = DecisionTreeRegressor().fit(X_train, y_train)

stop = time.time()

```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

Figure 7 Decision Tree training function for the DB1B Sub-Route View

For Bagging Tree Regressor the model training code is:

from sklearn.ensemble import BaggingRegressor

def br(X_train, y_train, ,<parameters to fill summary table>):

start = time.time()

model = BaggingRegressor().fit(X_train, y_train)

stop = time.time()

```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

Figure 8 Decision Tree training function for the DB1B Sub-Route View

For Gradient Boosting Regressor the model training code is:

from sklearn.experimental import enable_hist_gradient_boosting

from sklearn.ensemble import HistGradientBoostingRegressor

def gbr(X_train, y_train, ,<parameters to fill summary table>):

params = {

'learning_rate': 0.3,

'loss': 'poisson'

}

start = time.time()

model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)

stop = time.time()

```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
def gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    params = {
             #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson',#['least_squares', 'least_absolute_deviation', 'poisson'],
            'verbose':2}
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

Figure 9 Gradient Boosting Regressor training function for the DB1B Normal View

The general syntax used to train the models:

model=<model name>().fit(X_train, y_train)

X_train corresponds to the data-frame containing the Independent variables.

y_train corresponds to the data-frame containing the Dependent variables or the variable that is to be predicted.

The. fit() function is used to pass the values for training.

The variables start and stop are used to time how long it takes to train a model.

Functions c1, c2, c3 and c4 represent weighted model combinations used.

All the weighted models consist of RFR and BTR in the outer most loop and the inner most loop respectively

The middle loop either contains the DT model or GBR model. Based upon which model is present there the function is called c1 or c2 id DT model is present otherwise it is called c3 or c4.

The presence of an odd number in the function name (i.e., 1,3) represents the usage of only positive weights in the function.

The presence of an even number in the function name (i.e., 2,4) represents the usage of negative and positive weights in the function.

Function c1 is defined as:

```
def c1(model1,model2,model3, <parameters to fill summary table>):
    predictions1 = model1.predict(X_test)     #RF
    predictions2 = model2.predict(X_test)    #DT
    predictions3 = model3.predict(X_test)     #BR
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
```

```
#positive weight DT
def c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

Figure 10 function c1 for the Ease-My-Trip Dataset

Function c2 is defined as:

```
def c2(model1,model2,model3, ,<parameters to fill summary table>):
    predictions1 = model1.predict(X_test)     #RF
    predictions2 = model2.predict(X_test)    #DT
    predictions3 = model3.predict(X_test)     #BR
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
```

mi=i

mj=j

mk=k

predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3

```
#negative weight DT
def c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

Figure 11 function c2 for the Ease-My-Trip Dataset

Function c3 is defined as:

def c3(model1,model2,model3, <parameters to fill summary table>):

predictions1 = model1.predict(X_test) #RF

predictions2 = model2.predict(X_test) #GBR

predictions3 = model3.predict(X_test) #BR

max_r2=0

mi=0

mj=0

mk=0

for i in range(0,101):

for j in range(0,101):

```
        for k in range(0,101):
            if((i+j+k)==100):
                #print(i," ",j," ",k)
                pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                r2 = r2_score(y_test, pred)
                if(r2>max_r2):
                    max_r2=r2
                    mi=i
                    mj=j
                    mk=k
predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
```

```
#positive weight GBR
def c3(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

Figure 12 function c3 for the Machine Hack Dataset

Function c4 is defined as:

```
    def c4(model1,model2,model3, <parameters to fill summary table>):
        predictions1 = model1.predict(X_test)    #RF
        predictions2 = model2.predict(X_test)    #GBR
        predictions3 = model3.predict(X_test)    #BR
        max_r2=0
```

```
mi=0
mj=0
mk=0
for i in range(-100,301):
    for j in range(-100,301):
        for k in range(-100,301):
            if((i+j+k)==100):
                #print(i," ",j," ",k)
                pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                r2 = r2_score(y_test, pred)
                if(r2>max_r2):
                    max_r2=r2
                    mi=i
                    mj=j
                    mk=k
predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
```

Figure 13 function c4 for the Machine Hack Dataset

In function c1 and c2 model2 is a DT model whereas in c3 and c4 model2 is GBR model.

In the cases where the Datasets are split based on year and quarter there exists a function called modelT() which is used to execute all the model training functions (i.e., dt(), gbr(), rf(),

br()) as well as the weight based model combining function (i.e., c1(), c2(), c3(), c4()). It is also used to save the models as .pkl file

```
def modelT(X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
k,yr,qt):
    #RF
    print("RF")
    model1=rf(X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
0,k,yr,qt)
    #DT
    print("DT")
    model2=dt(X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
0,k,yr,qt)
    #BR
    print("BR")
    model3=br(X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
0,k,yr,qt)
    #GBR
    print("GBR")
    model4=gbr(X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
0,k,yr,qt)
    #positive weight
    print("c1")
    c1(model1,model2,model3,X_train, X_test, y_train,
y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
k,yr,qt)

    #negative weight
    print("c2")
```

```
        c2(model1,model2,model3,X_train, X_test, y_train,
    y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
    k,yr,qt)
        print("c3")
        c3(model1,model4,model3,X_train, X_test, y_train,
    y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
    k,yr,qt)
        print("c4")
        c4(model1,model4,model3,X_train, X_test, y_train,
    y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,
    k,yr,qt)

        #model saving
        filename = 'RFmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
        pickle.dump(model1, open(filename, 'wb'))

        filename = 'DTmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
        pickle.dump(model2, open(filename, 'wb'))

        filename = 'BRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
        pickle.dump(model3, open(filename, 'wb'))

        filename = 'GBRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
        pickle.dump(model4, open(filename, 'wb'))
return
```

Figure 14 modelT() function used in the normal view divided based on yrs and qtrs

```
def modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,yr,qt):
    #RF
    print("RF")
    model1=rf(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #DT
    print("DT")
    model2=dt(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #BR
    print("BR")
    model3=br(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #GBR
    print("GBR")
    model4=gbr(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #positive weight
    print("c1")
    c1(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l

    #negative weight
    print("c2")
    c2(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l
    print("c3")
    c3(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l
    print("c4")
    c4(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l

    #model saving
    filename = 'RFmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model1, open(filename, 'wb'))

    filename = 'DTmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model2, open(filename, 'wb'))

    filename = 'BRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model3, open(filename, 'wb'))

    filename = 'GBRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model4, open(filename, 'wb'))
    return
```

<parameters to fill summary table> is used to refer to the following variables:

Tr- Training Data size list

Te- Testing Data size list

r2_value- The r2 value list

adj_r2_value-The adjusted r2 value list

RMSE_value- The root-mean-square-error list

Mn- The model name list

max_i_ - The list of weight assigned to the Random Forest model

max_j_ - The list of weight assigned to the Decision tree model

max_k_ - The list of weight assigned to the Bagging tree model

max_l_ - The list of weight assigned to the Gradient boosting model

o- The parameter used to indicate whether predictions are to be performed. If 0 is equal to zero then the predictions are made the summary table is updated

k/tst- The value indicating percent of data used in training

rn- The list of route names

The function modelT is used only by the DB1B dataset when it is testing the data when it is divided based on year and quarter

The modelT function in addition to the parameters of other functions passes additional parameters like:

y_test – It corresponds to the data-frame containing the Dependent variables for testing.

X_test- It corresponds to the data-frame containing the Independent variables for testing.

yr- It corresponds to the year being used.

qt- It corresponds to the quarter being used.

In the web application there are 2 functions. They are:

home():

It is used to render the home page upon running the flask. The @app.route(‘/’) allows this function to be called first whenever the flask code is run.

```
@app.route('/')
def home():
    return render_template("interface.html")
```

Figure 15 Web-App home() function

predict():

This function is called when the form that is in the homepage is submitted. Here these values are converted into a format that can be passed as input to the models. These models using the input estimate the flight fare and this estimated flight fare is rendered on the page.

```
@app.route("/predict", methods = ["GET", "POST"])
def predict():

    if request.method == "POST":
        global cols1
        cols=cols1
        print(cols)
        #coupons nums
        mkc=request.form["MktCoupons"]
        cols=cols.replace("MktCoupons",mkc)

        #Year
        mkc=request.form["Year"]
        cols=cols.replace("Year",mkc)

        #Quarter
        mkc=request.form["Quarter"]
        cols=cols.replace("Quarter",mkc)

        #BulkFare
        mkc=request.form["BulkFare"]
        cols=cols.replace("BulkFare",mkc)


        #Passengers
        mkc=request.form["Passengers"]
        cols=cols.replace("Passengers",mkc)

        #Pandemic
        mkc=request.form["Pandemic"]
        cols=cols.replace("Pandemic",mkc)

        #airline
        mkc=request.form["airline"]
        cols=cols.replace(mkc,1)

        #Routes
        mkc=request.form["Routes"]
        cols=cols.replace(mkc,1)

        #Seat Class
        mkc=request.form["Seat Class"]
        cols=cols.replace(mkc,1)
        print(cols)


        for i in cols:
            if str(i).isdigit():
                b=0
            else:
                cols=cols.replace(i,0)

        print(cols)


        p1=m1.predict([cols])
        p3=m3.predict([cols])
        p4=m4.predict([cols])
        output=p1*0.24+p3*0.01+p4*0.75
        return render_template('interface.html',prediction_text="Your Flight price is $. {}".format(output))
    return
```

Figure 16 Web-App predict() function

4.3. LIMITATIONS AND CONSTRAINTS

In order to estimate flight fares, sufficient data point are required to compute specific fare statistics. Using historical flight details to show current flight prices compared to historical data aggregates could lead to incorrect conclusions.

The gradient boosting regressor does not use default parameters unlike the other models. This was done so inorder to get better results from the model.

The HistGradientBoostingRegressor() is being used here instead of the GradientBoostingRegressor() as the HistGradientBoostingRegressor() is faster when the sample size is greater than 10000 for large datasets.

The combination models includes only 3 models because it cannot allow for a 4 model as it would exponentially increase the time taken to compute can range from 100^3K to (400) ^3K. Hence it is not used.

The Flight price estimator web application is based only upon the normal view of the DB1B database as the values that need to be entered for other models vary.

5. SCHEDULE TASKS AND MILESTONES

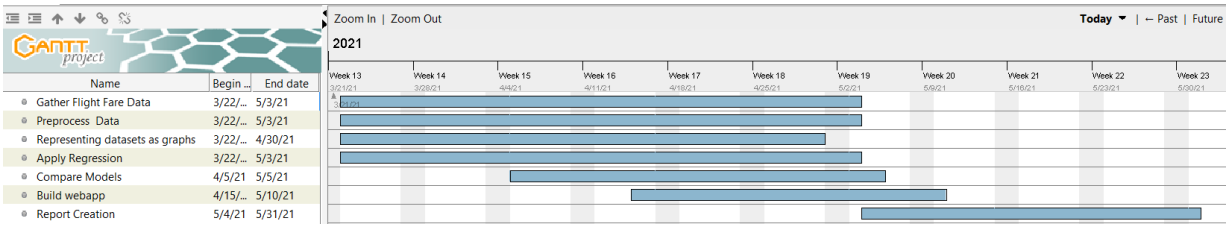


Figure 17 Gantt chart

The tasks are:

1. Gather data from various sources
   Starting date: 22 March 2021
   Ending Date: 3 May 2021
2. Process the gathered Data
   Starting date: 22 March 2021
   Ending Date: 3 May 2021
3. Understanding the data
   Starting date: 22 March 2021
   Ending Date: 3 May 2021

4. Applying the regression
   Starting date: 22 March 2021
   Ending Date: 3 May 2021
5. Comparison of models
   Starting date: 22 March 2021
   Ending Date: 3 May 2021
6. Web application building
   Starting date: 15 April 2021
   Ending Date: 10 May 2021
7. Report Creation
   Starting date: 4 May 2021
   Ending Date: 31 May 2021

The milestones for the project are:

1. Creation of the summary table which contains the model evaluation results for each dataset used.
2. Creation of Webapp using flask to predict flight fare.

6. PROJECT DEMONSTRATION

Feature Engineering:

DB1B Dataset:

```
In [1]: import pandas as pd
In [2]: df=pd.read_csv("cal-fl-rvi-all-1.csv")
In [3]: df1=pd.read_csv("cal-fl-rvi-all-2.csv")
In [4]: df=pd.concat([df,df1])
In [5]: df1=pd.read_csv("cal-fl-rvi-all-3.csv")
In [6]: df=pd.concat([df,df1])
In [7]: df1=pd.read_csv("cal-fl-rvi-all-4.csv")
In [8]: df=pd.concat([df,df1])
```

Figure 18 - 2018 Document Merging

```
In [9]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 813348 entries, 0 to 218599
Data columns (total 43 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   Unnamed: 0          813348 non-null  int64
 1   ItinID              813348 non-null  int64
 2   MktID               813348 non-null  int64
 3   MktCoupons          813348 non-null  int64
 4   Year                813348 non-null  int64
 5   Quarter             813348 non-null  int64
 6   OriginAirportID     813348 non-null  int64
 7   OriginAirportSeqID  813348 non-null  int64
 8   OriginCityMarketID  813348 non-null  int64
 9   Origin              813348 non-null  object
 10  OriginCountry       813348 non-null  object
 11  OriginStateFips     813348 non-null  int64
 12  OriginState         813348 non-null  object
 13  OriginStateName     813348 non-null  object
 14  OriginWac           813348 non-null  int64
 15  DestAirportID       813348 non-null  int64
 16  DestAirportSeqID    813348 non-null  int64
 17  DestCityMarketID    813348 non-null  int64
 18  Dest                813348 non-null  object
 19  DestCountry         813348 non-null  object
 20  DestStateFips       813348 non-null  int64
 21  DestState           813348 non-null  object
 22  DestStateName       813348 non-null  object
 23  DestWac             813348 non-null  int64
 24  AirportGroup        813348 non-null  object
 25  WacGroup            813348 non-null  object
 26  TkCarrierChange     813348 non-null  float64
 27  TkCarrierGroup      813348 non-null  object
 28  OpCarrierChange     813348 non-null  float64
 29  OpCarrierGroup      813348 non-null  object
 30  RPCarrier           813348 non-null  object
 31  TkCarrier           813348 non-null  object
 32  OpCarrier           813348 non-null  object
 33  BulkFare            813348 non-null  float64
 34  Passengers          813348 non-null  float64
 35  MktFare             813348 non-null  float64
 36  MktDistance         813348 non-null  float64
 37  MktDistanceGroup    813348 non-null  int64
 38  MktMilesFlown       813348 non-null  float64
 39  NonStopMiles        813348 non-null  float64
 40  ItinGeoType         813348 non-null  int64
 41  MktGeoType          813348 non-null  int64
 42  Unnamed: 41         0 non-null       float64
dtypes: float64(9), int64(19), object(15)
memory usage: 273.0+ MB

In [10]: df.to_csv("cal-fl-rvi-all-18.csv")
```

Figure 19 - 2019 Document Merging

```
In [1]: import pandas as pd

In [2]: df=pd.read_csv("cal-fl-all-1-19.csv")

In [3]: df1=pd.read_csv("cal-fl-all-2-19.csv")

In [4]: df=pd.concat([df,df1])

In [5]: df1=pd.read_csv("cal-fl-all-3-19.csv")

In [6]: df=pd.concat([df,df1])

In [7]: df1=pd.read_csv("cal-fl-all-4-19.csv")

In [8]: df=pd.concat([df,df1])

In [9]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 221149 entries, 0 to 59719
Data columns (total 43 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             221149 non-null  int64
 1   ITIN_ID                221149 non-null  int64
 2   MKT_ID                 221149 non-null  int64
 3   MARKET_COUPONS         221149 non-null  int64
 4   YEAR                   221149 non-null  int64
 5   QUARTER                221149 non-null  int64
 6   ORIGIN_AIRPORT_ID      221149 non-null  int64
 7   ORIGIN_AIRPORT_SEQ_ID  221149 non-null  int64
 8   ORIGIN_CITY_MARKET_ID  221149 non-null  int64
 9   ORIGIN                 221149 non-null  object
 10  ORIGIN_COUNTRY         221149 non-null  object
 11  ORIGIN_STATE_FIPS      221149 non-null  int64
 12  ORIGIN_STATE_ABR       221149 non-null  object
 13  ORIGIN_STATE_NM        221149 non-null  object
 14  ORIGIN_WAC             221149 non-null  int64
 15  DEST_AIRPORT_ID        221149 non-null  int64
 16  DEST_AIRPORT_SEQ_ID    221149 non-null  int64
 17  DEST_CITY_MARKET_ID    221149 non-null  int64
 18  DEST                   221149 non-null  object
 19  DEST_COUNTRY           221149 non-null  object
 20  DEST_STATE_FIPS        221149 non-null  int64
 21  DEST_STATE_ABR         221149 non-null  object
 22  DEST_STATE_NM          221149 non-null  object
 23  DEST_WAC               221149 non-null  int64
 24  AIRPORT_GROUP          221149 non-null  object
 25  WAC_GROUP              221149 non-null  object
 26  TK_CARRIER_CHANGE      221149 non-null  float64
 27  TK_CARRIER_GROUP       221149 non-null  object
 28  OP_CARRIER_CHANGE      221149 non-null  float64
 29  OP_CARRIER_GROUP       221149 non-null  object
 30  REPORTING_CARRIER      221149 non-null  object
 31  TICKET_CARRIER         221149 non-null  object
 32  OPERATING_CARRIER      221149 non-null  object
 33  BULK_FARE              221149 non-null  float64
 34  PASSENGERS             221149 non-null  float64
 35  MARKET_FARE            221149 non-null  float64
 36  MARKET_DISTANCE        221149 non-null  float64
 37  DISTANCE_GROUP         221149 non-null  int64
 38  MARKET_MILES_FLOWN     221149 non-null  float64
 39  NONSTOP_MILES          221149 non-null  float64
 40  ITIN_GEO_TYPE          221149 non-null  int64
 41  MKT_GEO_TYPE           221149 non-null  int64
 42  Unnamed: 41            0 non-null       float64
dtypes: float64(9), int64(19), object(15)
memory usage: 74.2+ MB

In [10]: df.to_csv("cal-fl-all-19.csv")
```

Figure 20 - 2020 document merging:

```
In [1]: import pandas as pd

In [2]: df=pd.read_csv("cal-fl-all-1-20.csv")

In [4]: df1=pd.read_csv("cal-fl-all-2-20.csv")

In [5]: df=pd.concat([df,df1])

In [6]: df1=pd.read_csv("cal-fl-all-3-20.csv")

In [7]: df=pd.concat([df,df1])

In [8]: #df1=pd.read_csv("cal-fl1-4.csv")

In [9]: #df=pd.concat([df,df1])

In [10]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75793 entries, 0 to 20479
Data columns (total 43 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          75793 non-null  int64
 1   ItinID              75793 non-null  int64
 2   MktID               75793 non-null  int64
 3   MktCoupons          75793 non-null  int64
 4   Year                75793 non-null  int64
 5   Quarter             75793 non-null  int64
 6   OriginAirportID     75793 non-null  int64
 7   OriginAirportSeqID  75793 non-null  int64
 8   OriginCityMarketID  75793 non-null  int64
 9   Origin              75793 non-null  object
 10  OriginCountry       75793 non-null  object
 11  OriginStateFips     75793 non-null  int64
 12  OriginState         75793 non-null  object
 13  OriginStateName     75793 non-null  object
 14  OriginWac           75793 non-null  int64
 15  DestAirportID       75793 non-null  int64
 16  DestAirportSeqID    75793 non-null  int64
 17  DestCityMarketID    75793 non-null  int64
 18  Dest                75793 non-null  object
 19  DestCountry         75793 non-null  object
 20  DestStateFips       75793 non-null  int64
 21  DestState           75793 non-null  object
 22  DestStateName       75793 non-null  object
 23  DestWac             75793 non-null  int64
 24  AirportGroup        75793 non-null  object
 25  WacGroup            75793 non-null  object
 26  TkCarrierChange     75793 non-null  float64
 27  TkCarrierGroup      75793 non-null  object
 28  OpCarrierChange     75793 non-null  float64
 29  OpCarrierGroup      75793 non-null  object
 30  RPCarrier           75793 non-null  object
 31  TkCarrier           75793 non-null  object
 32  OpCarrier           75793 non-null  object
 33  BulkFare            75793 non-null  float64
 34  Passengers          75793 non-null  float64
 35  MktFare             75793 non-null  float64
 36  MktDistance         75793 non-null  float64
 37  MktDistanceGroup    75793 non-null  int64
 38  MktMilesFlown       75793 non-null  float64
 39  NonStopMiles        75793 non-null  float64
 40  ItinGeoType         75793 non-null  int64
 41  MktGeoType          75793 non-null  int64
 42  Unnamed: 41         0 non-null      float64
dtypes: float64(9), int64(19), object(15)
memory usage: 25.4+ MB

In [11]: df.to_csv("cal-fl-all-20.csv")
```

Figure 21 Merging the yearly Documents

In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv("rvi/cal-fl-rvi-all-18.csv")
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 813348 entries, 0 to 813347
Data columns (total 44 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   Unnamed: 0          813348 non-null  int64  
 1   Unnamed: 0.1        813348 non-null  int64  
 2   ItinID              813348 non-null  int64  
 3   MktID               813348 non-null  int64  
 4   MktCoupons          813348 non-null  int64  
 5   Year                813348 non-null  int64  
 6   Quarter             813348 non-null  int64  
 7   OriginAirportID     813348 non-null  int64  
 8   OriginAirportSeqID  813348 non-null  int64  
 9   OriginCityMarketID  813348 non-null  int64  
 10  Origin              813348 non-null  object 
 11  OriginCountry       813348 non-null  object 
 12  OriginStateFips     813348 non-null  int64  
 13  OriginState         813348 non-null  object 
 14  OriginStateName     813348 non-null  object 
 15  OriginWac           813348 non-null  int64  
 16  DestAirportID       813348 non-null  int64  
 17  DestAirportSeqID    813348 non-null  int64  
 18  DestCityMarketID    813348 non-null  int64  
 19  Dest                813348 non-null  object 
 20  DestCountry         813348 non-null  object 
 21  DestStateFips       813348 non-null  int64  
 22  DestState           813348 non-null  object 
 23  DestStateName       813348 non-null  object 
 24  DestWac             813348 non-null  int64  
 25  AirportGroup        813348 non-null  object 
 26  WacGroup            813348 non-null  object 
 27  TkCarrierChange     813348 non-null  float64
 28  TkCarrierGroup      813348 non-null  object 
 29  OpCarrierChange     813348 non-null  float64
 30  OpCarrierGroup      813348 non-null  object 
 31  RPCarrier           813348 non-null  object 
 32  TkCarrier           813348 non-null  object 
 33  OpCarrier           813348 non-null  object 
 34  BulkFare            813348 non-null  float64
 35  Passengers          813348 non-null  float64
 36  MktFare             813348 non-null  float64
 37  MktDistance         813348 non-null  float64
 38  MktDistanceGroup    813348 non-null  int64  
 39  MktMilesFlown       813348 non-null  float64
 40  NonStopMiles        813348 non-null  float64
 41  ItinGeoType         813348 non-null  int64  
 42  MktGeoType          813348 non-null  int64  
 43  Unnamed: 41         0 non-null       float64
dtypes: float64(9), int64(20), object(15)
memory usage: 273.0+ MB
```

In [4]:
```
df1=pd.read_csv("rvi/cal-fl-rvi-all-19.csv")
```

```
In [5]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852046 entries, 0 to 852045
Data columns (total 44 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   Unnamed: 0          852046 non-null  int64
 1   Unnamed: 0.1        852046 non-null  int64
 2   ItinID              852046 non-null  float64
 3   MktID               852046 non-null  float64
 4   MktCoupons          852046 non-null  int64
 5   Year                852046 non-null  int64
 6   Quarter             852046 non-null  int64
 7   OriginAirportID     852046 non-null  int64
 8   OriginAirportSeqID  852046 non-null  int64
 9   OriginCityMarketID  852046 non-null  int64
 10  Origin              852046 non-null  object
 11  OriginCountry       852046 non-null  object
 12  OriginStateFips     852046 non-null  int64
 13  OriginState         852046 non-null  object
 14  OriginStateName     852046 non-null  object
 15  OriginWac           852046 non-null  int64
 16  DestAirportID       852046 non-null  int64
 17  DestAirportSeqID    852046 non-null  int64
 18  DestCityMarketID    852046 non-null  int64
 19  Dest                852046 non-null  object
 20  DestCountry         852046 non-null  object
 21  DestStateFips       852046 non-null  int64
 22  DestState           852046 non-null  object
 23  DestStateName       852046 non-null  object
 24  DestWac             852046 non-null  int64
 25  AirportGroup        852046 non-null  object
 26  WacGroup            852046 non-null  object
 27  TkCarrierChange     852046 non-null  int64
 28  TkCarrierGroup      852046 non-null  object
 29  OpCarrierChange     852046 non-null  int64
 30  OpCarrierGroup      852046 non-null  object
 31  RPCarrier           852046 non-null  object
 32  TkCarrier           852046 non-null  object
 33  OpCarrier           852046 non-null  object
 34  BulkFare            852046 non-null  int64
 35  Passengers          852046 non-null  int64
 36  MktFare             852046 non-null  float64
 37  MktDistance         852046 non-null  int64
 38  MktDistanceGroup    852046 non-null  int64
 39  MktMilesFlown       852046 non-null  int64
 40  NonStopMiles        852046 non-null  int64
 41  ItinGeoType         852046 non-null  int64
 42  MktGeoType          852046 non-null  int64
 43  Unnamed: 41         0 non-null       float64
dtypes: float64(4), int64(25), object(15)
memory usage: 286.0+ MB
```

```
In [6]: df=pd.concat([df,df1])
```

```
In [ ]:
```

```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1665394 entries, 0 to 852045
Data columns (total 44 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   Unnamed: 0          1665394 non-null  int64
 1   Unnamed: 0.1        1665394 non-null  int64
 2   ItinID              1665394 non-null  float64
 3   MktID               1665394 non-null  float64
 4   MktCoupons          1665394 non-null  int64
 5   Year                1665394 non-null  int64
 6   Quarter             1665394 non-null  int64
 7   OriginAirportID     1665394 non-null  int64
 8   OriginAirportSeqID  1665394 non-null  int64
 9   OriginCityMarketID  1665394 non-null  int64
 10  Origin              1665394 non-null  object
 11  OriginCountry       1665394 non-null  object
 12  OriginStateFips     1665394 non-null  int64
 13  OriginState         1665394 non-null  object
 14  OriginStateName     1665394 non-null  object
 15  OriginWac           1665394 non-null  int64
 16  DestAirportID       1665394 non-null  int64
 17  DestAirportSeqID    1665394 non-null  int64
 18  DestCityMarketID    1665394 non-null  int64
 19  Dest                1665394 non-null  object
 20  DestCountry         1665394 non-null  object
 21  DestStateFips       1665394 non-null  int64
 22  DestState           1665394 non-null  object
 23  DestStateName       1665394 non-null  object
 24  DestWac             1665394 non-null  int64
 25  AirportGroup        1665394 non-null  object
 26  WacGroup            1665394 non-null  object
 27  TkCarrierChange     1665394 non-null  float64
 28  TkCarrierGroup      1665394 non-null  object
 29  OpCarrierChange     1665394 non-null  float64
 30  OpCarrierGroup      1665394 non-null  object
 31  RPCarrier           1665394 non-null  object
 32  TkCarrier           1665394 non-null  object
 33  OpCarrier           1665394 non-null  object
 34  BulkFare            1665394 non-null  float64
 35  Passengers          1665394 non-null  float64
 36  MktFare             1665394 non-null  float64
 37  MktDistance         1665394 non-null  float64
 38  MktDistanceGroup    1665394 non-null  int64
 39  MktMilesFlown       1665394 non-null  float64
 40  NonStopMiles        1665394 non-null  float64
 41  ItinGeoType         1665394 non-null  int64
 42  MktGeoType          1665394 non-null  int64
 43  Unnamed: 41         0 non-null        float64
dtypes: float64(11), int64(18), object(15)
memory usage: 571.8+ MB
```

```
In [8]: df1=pd.read_csv("rvi/cal-fl-rvi-all-20.csv")
```

```
In [9]: df=pd.concat([df,df1])
```

```
In [10]: #df1=pd.read_csv("cal-fl1-4.csv")
```

```
In [11]: #df=pd.concat([df,df1])
```

```
In [12]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1930382 entries, 0 to 264987
Data columns (total 44 columns):
 #   Column              Dtype
---  ------              -----
 0   Unnamed: 0          int64
 1   Unnamed: 0.1        int64
 2   ItinID              float64
 3   MktID               float64
 4   MktCoupons          int64
 5   Year                int64
 6   Quarter             int64
 7   OriginAirportID     int64
 8   OriginAirportSeqID  int64
 9   OriginCityMarketID  int64
 10  Origin              object
```

```
 11  OriginCountry       object
 12  OriginStateFips     int64
 13  OriginState         object
 14  OriginStateName     object
 15  OriginWac           int64
 16  DestAirportID       int64
 17  DestAirportSeqID    int64
 18  DestCityMarketID    int64
 19  Dest                object
 20  DestCountry         object
 21  DestStateFips       int64
 22  DestState           object
 23  DestStateName       object
 24  DestWac             int64
 25  AirportGroup        object
 26  WacGroup            object
 27  TkCarrierChange     float64
 28  TkCarrierGroup      object
 29  OpCarrierChange     float64
 30  OpCarrierGroup      object
 31  RPCarrier           object
 32  TkCarrier           object
 33  OpCarrier           object
 34  BulkFare            float64
 35  Passengers          float64
 36  MktFare             float64
 37  MktDistance         float64
 38  MktDistanceGroup    int64
 39  MktMilesFlown       float64
 40  NonStopMiles        float64
 41  ItinGeoType         int64
 42  MktGeoType          int64
 43  Unnamed: 41         float64
dtypes: float64(11), int64(18), object(15)
memory usage: 662.7+ MB
```

In [13]:
```
df.to_csv("rvi/cal-fl-rvi-all.csv")
```

Adding additional columns that represent seating class from another dataset called the DB1B ticket dataset:

In [1]:
```
import pandas as pd
df=pd.read_csv("../input/calfl-reverse-as-well-non-sparce/cal-fl-rvi-all.csv")
```

In [2]:
```
import numpy as np
```

In [3]:
```
df1=pd.read_csv("../input/2020-db1b-coupon/res2018C.csv", usecols=['ItinID',"FareClass"])
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41360566 entries, 0 to 41360565
Data columns (total 2 columns):
 #   Column     Dtype 
---  ------     ----- 
 0   ItinID     int64 
 1   FareClass  object
dtypes: int64(1), object(1)
memory usage: 631.1+ MB
```

In [4]:
```
df1=df1.append(pd.read_csv("../input/2020-db1b-coupon/res2019C.csv", usecols=['ItinID',"FareClass"]))
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84620419 entries, 0 to 43259852
Data columns (total 2 columns):
 #   Column     Dtype 
---  ------     ----- 
 0   ItinID     int64 
 1   FareClass  object
dtypes: int64(1), object(1)
memory usage: 1.9+ GB
```

In [4]:
```
df1=df1.append(pd.read_csv("../input/2020-db1b-coupon/res2019C.csv", usecols=['ItinID',"FareClass"]))
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84620419 entries, 0 to 43259852
Data columns (total 2 columns):
 #   Column     Dtype 
---  ------     ----- 
 0   ItinID     int64 
 1   FareClass  object
dtypes: int64(1), object(1)
memory usage: 1.9+ GB
```

In [5]:
```
df1=df1.append(pd.read_csv("../input/2020-db1b-coupon/res2020f.csv", usecols=['ItinID',"FareClass"]))
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 98828944 entries, 0 to 14208524
Data columns (total 2 columns):
 #   Column     Dtype 
---  ------     ----- 
 0   ItinID     int64 
 1   FareClass  object
dtypes: int64(1), object(1)
memory usage: 2.2+ GB
```

In [6]:
```
df1.head()
```

Out[6]:

	ItinID	FareClass
0	201813056144	G
1	201813494948	X
2	201813324543	Y
3	201813723842	Y
4	201813483054	Y

In [7]:
```
df2=pd.merge(df, df1,on=["ItinID"])
```

In [8]:
```
df2.head()
```

Out[8]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	ItinID	MktID	MktCoupons	Year	Quarter	OriginAirportID	OriginAirportSeqID	...	Passengers	MktFare	M
0	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	1289208	...	1.0	1338.27	
1	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	1289208	...	1.0	1338.27	
2	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	1289208	...	1.0	1338.27	
3	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	1289208	...	1.0	1338.27	
4	1	1	1389	2.018177e+10	2.018177e+12	1	2018	1	14893	1489302	...	1.0	66.47	

5 rows × 46 columns

In [9]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4611555 entries, 0 to 4611554
Data columns (total 46 columns):
 #   Column              Dtype  
---  ------              -----  
 0   Unnamed: 0          int64  
 1   Unnamed: 0.1        int64  
 2   Unnamed: 0.1.1      int64  
 3   ItinID              float64
 4   MktID               float64
 5   MktCoupons          int64  
 6   Year                int64  
 7   Quarter             int64  
 8   OriginAirportID     int64  
 9   OriginAirportSeqID  int64  
 10  OriginCityMarketID  int64  
```

```
 11  Origin              object
 12  OriginCountry       object
 13  OriginStateFips     int64
 14  OriginState         object
 15  OriginStateName     object
 16  OriginWac           int64
 17  DestAirportID       int64
 18  DestAirportSeqID    int64
 19  DestCityMarketID    int64
 20  Dest                object
 21  DestCountry         object
 22  DestStateFips       int64
 23  DestState           object
 24  DestStateName       object
 25  DestWac             int64
 26  AirportGroup        object
 27  WacGroup            object
 28  TkCarrierChange     float64
 29  TkCarrierGroup      object
 30  OpCarrierChange     float64
 31  OpCarrierGroup      object
 32  RPCarrier           object
 33  TkCarrier           object
 34  OpCarrier           object
 35  BulkFare            float64
 36  Passengers          float64
 37  MktFare             float64
 38  MktDistance         float64
 39  MktDistanceGroup    int64
 40  MktMilesFlown       float64
 41  NonStopMiles        float64
 42  ItinGeoType         int64
 43  MktGeoType          int64
 44  Unnamed: 41         float64
 45  FareClass           object
dtypes: float64(11), int64(19), object(16)
memory usage: 1.6+ GB
```

```
In [10]: df2.to_csv("cl-flwithFC.csv")
```

Converting them into a machine-readable format:

```
In [1]: #version 2 sparce-all-rts-cal-fl.csv create
        import pandas as pd
```

```
In [2]: df=pd.read_csv("../input/calfl-reverse-as-well-non-sparce/cl-flwithFC.csv")
        df
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Unnamed: 0.1.1.1	ItinID	MktID	MktCoupons	Year	Quarter	OriginAirportID	...	Passengers	MktFare	Mk
0	0	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
1	1	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
2	2	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
3	3	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
4	4	1	1	1389	2.018177e+10	2.018177e+12	1	2018	1	14893	...	1.0	66.47	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4611550	4611550	264985	64401	2631169	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	153.00	
4611551	4611551	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	1.0	168.00	
4611552	4611552	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	1.0	168.00	
4611553	4611553	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	168.00	
4611554	4611554	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	168.00	

4611555 rows × 47 columns

In [3]:
```
import numpy as np
```

In [4]:
```
df['Pandemic']=np.where(np.logical_and(df['Year']==2020 , np.logical_or(df['Quarter']==2,df['Quarter']==3)), 1, 0)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
       'ItinID', 'MktID', 'MktCoupons', 'Year', 'Quarter', 'OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID', 'Origin', 'OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID', 'Dest',
       'DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'DestWac',
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup', 'RPCarrier', 'TkCarrier',
       'OpCarrier', 'BulkFare', 'Passengers', 'MktFare', 'MktDistance',
       'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41', 'FareClass', 'Pandemic'],
      dtype='object')
```

In [6]:
```
df["route"]=df['Origin']+"-"+df['Dest']
```

In [7]:
```
df
```

Out[7]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Unnamed: 0.1.1.1	ItinID	MktID	MktCoupons	Year	Quarter	OriginAirportID	...	MktDistance	MktDistance(
0	0	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
1	1	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
2	2	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
3	3	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
4	4	1	1	1389	2.018177e+10	2.018177e+12	1	2018	1	14893	...	373.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4611550	4611550	264985	64401	2631169	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	
4611551	4611551	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	204.0	
4611552	4611552	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	204.0	
4611553	4611553	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	
4611554	4611554	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	

4611555 rows × 49 columns

In [8]:
```
df["route"].value_counts()
```

Out[8]:
```
LAX-SFO    110821
SFO-LAX    109955
LAX-MCO    103128
MCO-LAX    101071
FLL-LAX     70205
            ...
TPA-MCO         2
PSP-SBA         2
MMH-ONT         2
JAX-MCO         1
MCO-JAX         1
Name: route, Length: 1026, dtype: int64
```

In [9]:
```
v = df["route"].value_counts()
df.shape
```

Out[9]:
```
(4611555, 49)
```

In [10]:
```
df=df[df.route.isin(v.index[v.gt(10016)])]
```

In [11]:
```
df.shape
```

Out[11]:
```
(3394381, 49)
```

In [12]:
```
df["route"].value_counts()
```

Out[12]:
```
LAX-SFO    110821
SFO-LAX    109955
LAX-MCO    103128
MCO-LAX    101071
FLL-LAX     70205
            ...
SNA-FLL     10546
LAX-PNS     10545
MIA-TPA     10441
PNS-LAX     10334
FLL-JAX     10310
Name: route, Length: 119, dtype: int64
```

In [13]:
```
#a = df1['route'].unique()
```

In [14]:
```
#grps=df1.groupby('route')
```

In [15]:
```
#for i in a:
 #   df2=grps.get_group(i)
  #  print(i," ",df2.shape)
   # df2.to_csv(i+".csv")
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
       'ItinID', 'MktID', 'MktCoupons', 'Year', 'Quarter', 'OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID', 'Origin', 'OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID', 'Dest',
       'DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'DestWac',
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup', 'RPCarrier', 'TkCarrier',
       'OpCarrier', 'BulkFare', 'Passengers', 'MktFare', 'MktDistance',
       'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41', 'FareClass', 'Pandemic', 'route'],
      dtype='object')
```

In [17]:
```
df=df.drop(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
            'ItinID', 'MktID','MktDistance','OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID','OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',"Origin",'Dest',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID','DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'Des
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup','TkCarrier',
       'OpCarrier', 'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41'],axis=1)
```

In [18]:
```
df=df.drop_duplicates()
```

In [19]:
```
df["route"].value_counts()
v = df["route"].value_counts()
```

In [20]:
```
df=df[df.route.isin(v.index[v.gt(10016)])]
```

In [21]:
```
df.shape
```

Out[21]:
```
(862459, 10)
```

In [22]:
```
df["route"].value_counts()
```

Out[22]:
```
LAX-SFO    48135
SFO-LAX    47627
LAX-MCO    40285
MCO-LAX    39668
MIA-LAX    27502
LAX-MIA    27500
SAN-SFO    26842
SFO-SAN    26638
LAX-FLL    24954
FLL-LAX    24915
MCO-SFO    22883
SNA-SFO    22663
SFO-SNA    22596
SFO-MCO    22480
MIA-SFO    21694
SFO-MIA    20963
LAX-TPA    19504
TPA-LAX    18832
LAX-SJC    18765
SJC-LAX    18481
FLL-SFO    17483
SFO-FLL    17385
SAN-MCO    16525
MCO-SAN    16380
LAX-SMF    14829
SMF-LAX    14567
SAN-SJC    14119
SJC-SAN    14084
SAN-SMF    13141
SMF-SAN    12944
SFO-TPA    12689
TPA-SFO    12451
BUR-SFO    12022
SFO-BUR    11942
LAX-OAK    11851
```

```
OAK-LAX    11601
MCO-SMF    11067
SJC-SNA    10955
SMF-MCO    10932
PSP-SFO    10706
SNA-SJC    10618
SFO-PSP    10563
SAN-TPA    10429
TPA-SAN    10140
FLL-SAN    10109
Name: route, dtype: int64
```

In [23]:
```
df.columns
```

Out[23]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'RPCarrier', 'BulkFare', 'Passengers',
       'MktFare', 'FareClass', 'Pandemic', 'route'],
      dtype='object')
```

In [24]:
```
df.shape
```

Out[24]:
```
(862459, 10)
```

In [25]:
```
df
```

Out[25]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route
0	3	2018	1	AA	0.0	1.0	1338.27	NaN	0	LAX-FLL
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA
...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA

862459 rows × 10 columns

In [26]:
```
df = df.dropna()
```

In [27]:
```
df
```

Out[27]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA
28	1	2018	1	AA	0.0	1.0	1651.23	D	0	MIA-LAX
...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA

831288 rows × 10 columns

In [28]:
```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    return(res)
```

In [29]:
```
#s=pd.Series(list(df["RPCarrier"]))
#d=pd.get_dummies(s)
#df=pd.concat([df,d],axis=1)
#df=df.drop(["RPCarrier"],axis=1)
df=encode_and_bind(df,"RPCarrier")
```

In [30]:
```
df.shape
```

Out[30]: (831288, 31)

In [31]:
```
df
```

Out[31]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route	...	RPCarrier_NK	RPCarrier_OH	RPCarrier_OO
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL	...	0	0	0
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX	...	0	0	0
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA	...	0	0	0
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA	...	0	0	0
28	1	2018	1	AA	0.0	1.0	1651.23	D	0	MIA-LAX	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO	...	0	0	0
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO	...	0	0	0
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO	...	0	0	0
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO	...	0	0	0
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA	...	0	0	0

831288 rows × 31 columns

In [32]:
```
df.columns
```

Out[32]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'RPCarrier', 'BulkFare', 'Passengers',
       'MktFare', 'FareClass', 'Pandemic', 'route', 'RPCarrier_9E',
       'RPCarrier_AA', 'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP',
       'RPCarrier_DL', 'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7',
       'RPCarrier_HA', 'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH',
       'RPCarrier_OO', 'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA',
       'RPCarrier_VX', 'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX'],
      dtype='object')
```

In [33]:
```
#s=pd.Series(list(df["route"]))#.isin(v.index[v.gt(10016)])))
#d=pd.get_dummies(s)
#df=pd.concat([df,d],axis=1)
#df=df.drop(["route",'Unnamed: 0.1.1'],axis=1)
df=encode_and_bind(df,"route")
df=df.drop(["route",'RPCarrier'],axis=1)
```

In [34]:
```
df.columns
```

Out[34]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers', 'MktFare',
       'FareClass', 'Pandemic', 'RPCarrier_9E', 'RPCarrier_AA', 'RPCarrier_AS',
       'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL', 'RPCarrier_EV',
       'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA', 'RPCarrier_MQ',
       'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO', 'RPCarrier_QX',
       'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX', 'RPCarrier_WN',
       'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO', 'route_FLL-LAX',
       'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL', 'route_LAX-MCO',
       'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO', 'route_LAX-SJC',
       'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX', 'route_MCO-SAN',
       'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX', 'route_MIA-SFO',
       'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO', 'route_SAN-SFO',
       'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA', 'route_SFO-BUR',
       'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO', 'route_SFO-MIA',
       'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA', 'route_SFO-TPA',
       'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA', 'route_SMF-LAX',
       'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO', 'route_SNA-SJC',
       'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO'],
      dtype='object')
```

In [35]:
```
df
```

Out[35]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	FareClass	Pandemic	RPCarrier_9E	RPCarrier_AA	...	route_SJC-SAN	route_SJC-SNA	route_S
1	3	2018	1	0.0	1.0	1338.27	D	0	0	1	...	0	0	
4	1	2018	1	0.0	1.0	66.47	X	0	0	1	...	0	0	
17	4	2018	1	0.0	1.0	5.54	X	0	0	1	...	0	0	
22	1	2018	1	0.0	1.0	521.46	X	0	0	1	...	0	0	
28	1	2018	1	0.0	1.0	1651.23	D	0	0	1	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611350	2	2020	3	0.0	1.0	78.00	X	1	0	0	...	0	0	
4611352	2	2020	3	0.0	1.0	98.00	X	1	0	0	...	0	0	
4611354	2	2020	3	0.0	1.0	213.00	X	1	0	0	...	0	0	
4611368	2	2020	3	0.0	1.0	103.50	X	1	0	0	...	0	0	
4611372	2	2020	3	0.0	1.0	103.50	X	1	0	0	...	0	0	

831288 rows × 74 columns

In [36]:
```
sub=pd.read_csv("../input/calfl-reverse-as-well-non-sparce/L_FARE_CLASS.csv_")
```

In [37]:
```
sub
```

Out[37]:

	Code	Description
0	-	Ground Segment
1	C	Unrestricted Business Class
2	D	Restricted Business Class
3	F	Unrestricted First Class
4	G	Restricted First Class
5	U	Unknown
6	X	Restricted Coach Class
7	Y	Unrestricted Coach Class

In [38]:
```
df["FareClass"]=df["FareClass"].map({"C":"Unrestricted Business Class","D":"Restricted Business Class",
                                     "F":"Unrestricted First Class","G":"Restricted First Class",
                                     "X":"Restricted Coach Class","Y":"Unrestricted Coach Class",
                                     "U":"Others"})
```

In [39]:
```
df
```

Out[39]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	FareClass	Pandemic	RPCarrier_9E	RPCarrier_AA	...	route_SJC-SAN	route_SJC-SNA	route_S
1	3	2018	1	0.0	1.0	1338.27	Restricted Business Class	0	0	1	...	0	0	
4	1	2018	1	0.0	1.0	66.47	Restricted Coach Class	0	0	1	...	0	0	
17	4	2018	1	0.0	1.0	5.54	Restricted Coach Class	0	0	1	...	0	0	
22	1	2018	1	0.0	1.0	521.46	Restricted Coach Class	0	0	1	...	0	0	
28	1	2018	1	0.0	1.0	1651.23	Restricted Business Class	0	0	1	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611350	2	2020	3	0.0	1.0	78.00	Restricted Coach Class	1	0	0	...	0	0	
4611352	2	2020	3	0.0	1.0	98.00	Restricted Coach Class	1	0	0	...	0	0	
4611354	2	2020	3	0.0	1.0	213.00	Restricted Coach Class	1	0	0	...	0	0	
4611368	2	2020	3	0.0	1.0	103.50	Restricted Coach Class	1	0	0	...	0	0	
4611372	2	2020	3	0.0	1.0	103.50	Restricted Coach Class	1	0	0	...	0	0	

831288 rows × 74 columns

In [40]:
```
df=encode_and_bind(df,"FareClass")
df=df.drop(["FareClass"],axis=1)
```

In [41]:
```
df
```

Out[41]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	Pandemic	RPCarrier_9E	RPCarrier_AA	RPCarrier_AS	...	route_TPA-LAX	route_TPA-SAN	rou
1	3	2018	1	0.0	1.0	1338.27	0	0	1	0	...	0	0	
4	1	2018	1	0.0	1.0	66.47	0	0	1	0	...	0	0	
17	4	2018	1	0.0	1.0	5.54	0	0	1	0	...	0	0	
22	1	2018	1	0.0	1.0	521.46	0	0	1	0	...	0	0	
28	1	2018	1	0.0	1.0	1651.23	0	0	1	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611350	2	2020	3	0.0	1.0	78.00	1	0	0	0	...	0	0	
4611352	2	2020	3	0.0	1.0	98.00	1	0	0	0	...	0	0	
4611354	2	2020	3	0.0	1.0	213.00	1	0	0	0	...	0	0	
4611368	2	2020	3	0.0	1.0	103.50	1	0	0	0	...	0	0	
4611372	2	2020	3	0.0	1.0	103.50	1	0	0	0	...	0	0	

831288 rows × 80 columns

In [42]:
```
df.shape
```

Out[42]:
```
(831288, 80)
```

In [43]:
```
df.columns
```

Out[43]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers', 'MktFare',
       'Pandemic', 'RPCarrier_9E', 'RPCarrier_AA', 'RPCarrier_AS',
       'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL', 'RPCarrier_EV',
       'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA', 'RPCarrier_MQ',
       'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO', 'RPCarrier_QX',
       'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX', 'RPCarrier_WN',
       'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO', 'route_FLL-LAX',
       'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL', 'route_LAX-MCO',
       'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO', 'route_LAX-SJC',
       'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX', 'route_MCO-SAN',
       'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX', 'route_MIA-SFO',
       'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO', 'route_SAN-SFO',
       'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA', 'route_SFO-BUR',
       'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO', 'route_SFO-MIA',
       'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA', 'route_SFO-TPA',
       'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA', 'route_SMF-LAX',
       'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO', 'route_SNA-SJC',
       'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO', 'FareClass_Others',
       'FareClass_Restricted Business Class',
       'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
       'FareClass_Unrestricted Business Class',
       'FareClass_Unrestricted Coach Class',
       'FareClass_Unrestricted First Class'],
      dtype='object')
```

In [44]:
```
df.to_csv("sparce-all-rts-cal-fl-withfareclass.csv")
```

In [45]:
```
c=0
v=df.columns
for i in v:

    if "route" in i:
        print(i)
        c=c+1
print(c)
```

```
route_BUR-SFO
route_FLL-LAX
route_FLL-SAN
route_FLL-SFO
route_LAX-FLL
route_LAX-MCO
route_LAX-MIA
route_LAX-OAK
route_LAX-SFO
route_LAX-SJC
route_LAX-SMF
route_LAX-TPA
route_MCO-LAX
route_MCO-SAN
route_MCO-SFO
route_MCO-SMF
route_MIA-LAX
route_MIA-SFO
route_OAK-LAX
route_PSP-SFO
route_SAN-MCO
route_SAN-SFO
route_SAN-SJC
route_SAN-SMF
route_SAN-TPA
route_SFO-BUR
route_SFO-FLL
route_SFO-LAX
route_SFO-MCO
route_SFO-MIA
route_SFO-PSP
route_SFO-SAN
route_SFO-SNA
route_SFO-TPA
route_SJC-LAX
route_SJC-SAN
route_SJC-SNA
route_SMF-LAX
route_SMF-MCO
route_SMF-SAN
route_SNA-SFO
route_SNA-SJC
route_TPA-LAX
route_TPA-SAN
route_TPA-SFO
45
```

In [46]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 831288 entries, 1 to 4611372
Data columns (total 80 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   MktCoupons                              831288 non-null  int64
 1   Year                                    831288 non-null  int64
 2   Quarter                                 831288 non-null  int64
 3   BulkFare                                831288 non-null  float64
 4   Passengers                              831288 non-null  float64
 5   MktFare                                 831288 non-null  float64
 6   Pandemic                                831288 non-null  int64
 7   RPCarrier_9E                            831288 non-null  uint8
 8   RPCarrier_AA                            831288 non-null  uint8
 9   RPCarrier_AS                            831288 non-null  uint8
 10  RPCarrier_B6                            831288 non-null  uint8
 11  RPCarrier_CP                            831288 non-null  uint8
 12  RPCarrier_DL                            831288 non-null  uint8
 13  RPCarrier_EV                            831288 non-null  uint8
 14  RPCarrier_F9                            831288 non-null  uint8
 15  RPCarrier_G7                            831288 non-null  uint8
 16  RPCarrier_HA                            831288 non-null  uint8
 17  RPCarrier_MQ                            831288 non-null  uint8
 18  RPCarrier_NK                            831288 non-null  uint8
 19  RPCarrier_OH                            831288 non-null  uint8
 20  RPCarrier_OO                            831288 non-null  uint8
 21  RPCarrier_QX                            831288 non-null  uint8
 22  RPCarrier_SY                            831288 non-null  uint8
 23  RPCarrier_UA                            831288 non-null  uint8
 24  RPCarrier_VX                            831288 non-null  uint8
 25  RPCarrier_WN                            831288 non-null  uint8
 26  RPCarrier_YV                            831288 non-null  uint8
 27  RPCarrier_YX                            831288 non-null  uint8
 28  route_BUR-SFO                           831288 non-null  uint8
 29  route_FLL-LAX                           831288 non-null  uint8
 30  route_FLL-SAN                           831288 non-null  uint8
 31  route_FLL-SFO                           831288 non-null  uint8
 32  route_LAX-FLL                           831288 non-null  uint8
 33  route_LAX-MCO                           831288 non-null  uint8
 34  route_LAX-MIA                           831288 non-null  uint8
 35  route_LAX-OAK                           831288 non-null  uint8
 36  route_LAX-SFO                           831288 non-null  uint8
 37  route_LAX-SJC                           831288 non-null  uint8
 38  route_LAX-SMF                           831288 non-null  uint8
 39  route_LAX-TPA                           831288 non-null  uint8
 40  route_MCO-LAX                           831288 non-null  uint8
 41  route_MCO-SAN                           831288 non-null  uint8
 42  route_MCO-SFO                           831288 non-null  uint8
 43  route_MCO-SMF                           831288 non-null  uint8
 44  route_MIA-LAX                           831288 non-null  uint8
 45  route_MIA-SFO                           831288 non-null  uint8
 46  route_OAK-LAX                           831288 non-null  uint8
 47  route_PSP-SFO                           831288 non-null  uint8
 48  route_SAN-MCO                           831288 non-null  uint8
 49  route_SAN-SFO                           831288 non-null  uint8
 50  route_SAN-SJC                           831288 non-null  uint8
 51  route_SAN-SMF                           831288 non-null  uint8
 52  route_SAN-TPA                           831288 non-null  uint8
 53  route_SFO-BUR                           831288 non-null  uint8
 54  route_SFO-FLL                           831288 non-null  uint8
 55  route_SFO-LAX                           831288 non-null  uint8
 56  route_SFO-MCO                           831288 non-null  uint8
 57  route_SFO-MIA                           831288 non-null  uint8
 58  route_SFO-PSP                           831288 non-null  uint8
 59  route_SFO-SAN                           831288 non-null  uint8
 60  route_SFO-SNA                           831288 non-null  uint8
 61  route_SFO-TPA                           831288 non-null  uint8
 62  route_SJC-LAX                           831288 non-null  uint8
 63  route_SJC-SAN                           831288 non-null  uint8
 64  route_SJC-SNA                           831288 non-null  uint8
 65  route_SMF-LAX                           831288 non-null  uint8
 66  route_SMF-MCO                           831288 non-null  uint8
 67  route_SMF-SAN                           831288 non-null  uint8
 68  route_SNA-SFO                           831288 non-null  uint8
 69  route_SNA-SJC                           831288 non-null  uint8
 70  route_TPA-LAX                           831288 non-null  uint8
 71  route_TPA-SAN                           831288 non-null  uint8
 72  route_TPA-SFO                           831288 non-null  uint8
 73  FareClass_Others                        831288 non-null  uint8
 74  FareClass_Restricted Business Class     831288 non-null  uint8
 75  FareClass_Restricted Coach Class        831288 non-null  uint8
 76  FareClass_Restricted First Class        831288 non-null  uint8
 77  FareClass_Unrestricted Business Class   831288 non-null  uint8
 78  FareClass_Unrestricted Coach Class      831288 non-null  uint8
 79  FareClass_Unrestricted First Class      831288 non-null  uint8
dtypes: float64(3), int64(4), uint8(73)
memory usage: 208.6 MB
```

Dividing the data based on sub-routes:

In [1]:
```
#version 10 creating subroutes
import pandas as pd
```

In [2]:
```
df=pd.read_csv("../input/calfl-reverse-as-well-non-sparce/cl-flwithFC.csv")
df
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Unnamed: 0.1.1.1	ItinID	MktID	MktCoupons	Year	Quarter	OriginAirportID	...	Passengers	MktFare	Mkt
0	0	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
1	1	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
2	2	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
3	3	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	1.0	1338.27	
4	4	1	1	1389	2.018177e+10	2.018177e+12	1	2018	1	14893	...	1.0	66.47	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611550	4611550	264985	64401	2631169	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	153.00	
4611551	4611551	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	1.0	168.00	
4611552	4611552	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	1.0	168.00	
4611553	4611553	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	168.00	
4611554	4611554	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	1.0	168.00	

4611555 rows × 47 columns

In [3]:
```
import numpy as np
```

In [4]:
```
df['Pandemic']=np.where(np.logical_and(df['Year']==2020 , np.logical_or(df['Quarter']==2,df['Quarter']==3)), 1, 0)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
       'ItinID', 'MktID', 'MktCoupons', 'Year', 'Quarter', 'OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID', 'Origin', 'OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID', 'Dest',
       'DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'DestWac',
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup', 'RPCarrier', 'TkCarrier',
       'OpCarrier', 'BulkFare', 'Passengers', 'MktFare', 'MktDistance',
       'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41', 'FareClass', 'Pandemic'],
      dtype='object')
```

In [6]:
```
df["route"]=df['Origin']+"-"+df['Dest']
```

In [7]:
```
df
```

Out[7]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Unnamed: 0.1.1.1	ItinID	MktID	MktCoupons	Year	Quarter	OriginAirportID	...	MktDistance	MktDistanceG
0	0	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
1	1	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
2	2	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
3	3	0	0	1050	2.018177e+10	2.018177e+12	3	2018	1	12892	...	2399.0	
4	4	1	1	1389	2.018177e+10	2.018177e+12	1	2018	1	14893	...	373.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4611550	4611550	264985	64401	2631169	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	
4611551	4611551	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	204.0	
4611552	4611552	264986	64402	2631170	2.020319e+11	2.020319e+13	1	2020	3	13303	...	204.0	
4611553	4611553	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	
4611554	4611554	264987	64403	2631171	2.020319e+11	2.020319e+13	1	2020	3	15304	...	204.0	

4611555 rows × 49 columns

In [8]:
```
df["route"].value_counts()
```

Out[8]:
```
LAX-SFO    110821
SFO-LAX    109955
LAX-MCO    103128
MCO-LAX    101071
FLL-LAX     70205
              ...
SCK-VPS         2
ONT-MCE         2
TPA-MCO         2
MCO-JAX         1
JAX-MCO         1
Name: route, Length: 1026, dtype: int64
```

In [9]:
```
v = df["route"].value_counts()
df.shape
```

Out[9]:
```
(4611555, 49)
```

In [10]:
```
df=df[df.route.isin(v.index[v.gt(10016)])]
```

In [11]:
```
df.shape
```

Out[11]:
```
(3394381, 49)
```

In [12]:
```
df["route"].value_counts()
```

Out[12]:
```
LAX-SFO    110821
SFO-LAX    109955
LAX-MCO    103128
MCO-LAX    101071
FLL-LAX     70205
              ...
SNA-FLL     10546
LAX-PNS     10545
MIA-TPA     10441
PNS-LAX     10334
FLL-JAX     10310
Name: route, Length: 119, dtype: int64
```

In [13]:
```
#a = df1['route'].unique()
```

In [14]:
```
#grps=df1.groupby('route')
```

In [15]:
```
#for i in a:
 #   df2=grps.get_group(i)
  #  print(i," ",df2.shape)
   # df2.to_csv(i+".csv")
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
       'ItinID', 'MktID', 'MktCoupons', 'Year', 'Quarter', 'OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID', 'Origin', 'OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID', 'Dest',
       'DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'DestWac',
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup', 'RPCarrier', 'TkCarrier',
       'OpCarrier', 'BulkFare', 'Passengers', 'MktFare', 'MktDistance',
       'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41', 'FareClass', 'Pandemic', 'route'],
      dtype='object')
```

In [17]:
```
df=df.drop(['Unnamed: 0', 'Unnamed: 0.1', 'Unnamed: 0.1.1', 'Unnamed: 0.1.1.1',
            'ItinID', 'MktID','MktDistance','OriginAirportID',
       'OriginAirportSeqID', 'OriginCityMarketID','OriginCountry',
       'OriginStateFips', 'OriginState', 'OriginStateName', 'OriginWac',"Origin",'Dest',
       'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID','DestCountry', 'DestStateFips', 'DestState', 'DestStateName', 'Des
       'AirportGroup', 'WacGroup', 'TkCarrierChange', 'TkCarrierGroup',
       'OpCarrierChange', 'OpCarrierGroup','TkCarrier',
       'OpCarrier', 'MktDistanceGroup', 'MktMilesFlown', 'NonStopMiles', 'ItinGeoType',
       'MktGeoType', 'Unnamed: 41'],axis=1)
```

In [18]:
```
df=df.drop_duplicates()
```

In [19]:
```
df["route"].value_counts()
v = df["route"].value_counts()
```

```
In [20]: df=df[df.route.isin(v.index[v.gt(10016)])]
```

```
In [21]: df.shape
```

```
Out[21]: (862459, 10)
```

```
In [22]: df["route"].value_counts()
```

```
Out[22]: LAX-SFO    48135
         SFO-LAX    47627
         LAX-MCO    40285
         MCO-LAX    39668
         MIA-LAX    27502
         LAX-MIA    27500
         SAN-SFO    26842
         SFO-SAN    26638
         LAX-FLL    24954
         FLL-LAX    24915
         MCO-SFO    22883
         SNA-SFO    22663
         SFO-SNA    22596
         SFO-MCO    22480
         MIA-SFO    21694
         SFO-MIA    20963
         LAX-TPA    19504
         TPA-LAX    18832
         LAX-SJC    18765
         SJC-LAX    18481
         FLL-SFO    17483
         SFO-FLL    17385
         SAN-MCO    16525
         MCO-SAN    16380
         LAX-SMF    14829
         SMF-LAX    14567
         SAN-SJC    14119
         SJC-SAN    14084
         SAN-SMF    13141
         SMF-SAN    12944
         SFO-TPA    12689
         TPA-SFO    12451
         BUR-SFO    12022
         SFO-BUR    11942
         LAX-OAK    11851
         OAK-LAX    11601
         MCO-SMF    11067
         SJC-SNA    10955
         SMF-MCO    10932
         PSP-SFO    10706
         SNA-SJC    10618
         SFO-PSP    10563
         SAN-TPA    10429
         TPA-SAN    10140
         FLL-SAN    10109
         Name: route, dtype: int64
```

```
In [23]: df.columns
```

```
Out[23]: Index(['MktCoupons', 'Year', 'Quarter', 'RPCarrier', 'BulkFare', 'Passengers',
                'MktFare', 'FareClass', 'Pandemic', 'route'],
               dtype='object')
```

```
In [24]: df.shape
```

```
Out[24]: (862459, 10)
```

```
In [25]: df
```

Out[25]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route
0	3	2018	1	AA	0.0	1.0	1338.27	NaN	0	LAX-FLL
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA
...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA

862459 rows × 10 columns

In [26]:
```
df = df.dropna()
```

In [27]:
```
df
```

Out[27]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA
28	1	2018	1	AA	0.0	1.0	1651.23	D	0	MIA-LAX
...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA

831288 rows × 10 columns

In [28]:
```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    return(res)
```

In [29]:
```
#s=pd.Series(list(df["RPCarrier"]))
#d=pd.get_dummies(s)
#df=pd.concat([df,d],axis=1)
#df=df.drop(["RPCarrier"],axis=1)
df=encode_and_bind(df,"RPCarrier")
```

In [30]:
```
df.shape
```

Out[30]:
```
(831288, 31)
```

In [31]:
```
df
```

Out[31]:

	MktCoupons	Year	Quarter	RPCarrier	BulkFare	Passengers	MktFare	FareClass	Pandemic	route	...	RPCarrier_NK	RPCarrier_OH	RPCarrier_OO
1	3	2018	1	AA	0.0	1.0	1338.27	D	0	LAX-FLL	...	0	0	0
4	1	2018	1	AA	0.0	1.0	66.47	X	0	SMF-LAX	...	0	0	0
17	4	2018	1	AA	0.0	1.0	5.54	X	0	SFO-TPA	...	0	0	0
22	1	2018	1	AA	0.0	1.0	521.46	X	0	SFO-MIA	...	0	0	0
28	1	2018	1	AA	0.0	1.0	1651.23	D	0	MIA-LAX	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	YX	0.0	1.0	78.00	X	1	MIA-SFO	...	0	0	0
4611352	2	2020	3	YX	0.0	1.0	98.00	X	1	MIA-SFO	...	0	0	0
4611354	2	2020	3	YX	0.0	1.0	213.00	X	1	MIA-SFO	...	0	0	0
4611368	2	2020	3	YX	0.0	1.0	103.50	X	1	MIA-SFO	...	0	0	0
4611372	2	2020	3	YX	0.0	1.0	103.50	X	1	SFO-MIA	...	0	0	0

831288 rows × 31 columns

In [32]:
```
df.columns
```

Out[32]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'RPCarrier', 'BulkFare', 'Passengers',
       'MktFare', 'FareClass', 'Pandemic', 'route', 'RPCarrier_9E',
       'RPCarrier_AA', 'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP',
       'RPCarrier_DL', 'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7',
       'RPCarrier_HA', 'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH',
       'RPCarrier_OO', 'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA',
       'RPCarrier_VX', 'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX'],
      dtype='object')
```

In [33]:
```
#s=pd.Series(list(df["route"]))#.isin(v.index[v.gt(10016)]))
#d=pd.get_dummies(s)
#df=pd.concat([df,d],axis=1)
#df=df.drop(["route",'Unnamed: 0.1.1'],axis=1)
df=encode_and_bind(df,"route")
df=df.drop(['RPCarrier'],axis=1)
```

In [34]:
```
df.columns
```

Out[34]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers', 'MktFare',
       'FareClass', 'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA',
       'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL',
       'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA',
       'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO',
       'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX',
       'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO',
       'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
       'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
       'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
       'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
       'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
       'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
       'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
       'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
       'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
       'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
       'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO'],
      dtype='object')
```

In [35]:
```
df
```

Out[35]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	FareClass	Pandemic	route	RPCarrier_9E	...	route_SJC-SAN	route_SJC-SNA	route_SMF-LAX	ro
1	3	2018	1	0.0	1.0	1338.27	D	0	LAX-FLL	0	...	0	0	0	
4	1	2018	1	0.0	1.0	66.47	X	0	SMF-LAX	0	...	0	0	1	
17	4	2018	1	0.0	1.0	5.54	X	0	SFO-TPA	0	...	0	0	0	
22	1	2018	1	0.0	1.0	521.46	X	0	SFO-MIA	0	...	0	0	0	
28	1	2018	1	0.0	1.0	1651.23	D	0	MIA-LAX	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611350	2	2020	3	0.0	1.0	78.00	X	1	MIA-SFO	0	...	0	0	0	
4611352	2	2020	3	0.0	1.0	98.00	X	1	MIA-SFO	0	...	0	0	0	
4611354	2	2020	3	0.0	1.0	213.00	X	1	MIA-SFO	0	...	0	0	0	
4611368	2	2020	3	0.0	1.0	103.50	X	1	MIA-SFO	0	...	0	0	0	
4611372	2	2020	3	0.0	1.0	103.50	X	1	SFO-MIA	0	...	0	0	0	

831288 rows × 75 columns

In [36]:
```
sub=pd.read_csv("../input/calfl-reverse-as-well-non-sparce/L_FARE_CLASS.csv_")
```

In [37]:
```
sub
```

Out[37]:

	Code	Description
0	-	Ground Segment
1	C	Unrestricted Business Class
2	D	Restricted Business Class
3	F	Unrestricted First Class
4	G	Restricted First Class
5	U	Unknown
6	X	Restricted Coach Class
7	Y	Unrestricted Coach Class

In [38]:
```
df["FareClass"]=df["FareClass"].map({"C":"Unrestricted Business Class","D":"Restricted Business Class",
                                     "F":"Unrestricted First Class","G":"Restricted First Class",
                                     "X":"Restricted Coach Class","Y":"Unrestricted Coach Class",
                                     "U":"Others"})
```

In [39]:
```
df
```

Out[39]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	FareClass	Pandemic	route	RPCarrier_9E	...	route_SJC-SAN	route_SJC-SNA	route_SMF-LAX	ro
1	3	2018	1	0.0	1.0	1338.27	Restricted Business Class	0	LAX-FLL	0	...	0	0	0	
4	1	2018	1	0.0	1.0	66.47	Restricted Coach Class	0	SMF-LAX	0	...	0	0	1	
17	4	2018	1	0.0	1.0	5.54	Restricted Coach Class	0	SFO-TPA	0	...	0	0	0	
22	1	2018	1	0.0	1.0	521.46	Restricted Coach Class	0	SFO-MIA	0	...	0	0	0	
28	1	2018	1	0.0	1.0	1651.23	Restricted Business Class	0	MIA-LAX	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4611350	2	2020	3	0.0	1.0	78.00	Restricted Coach Class	1	MIA-SFO	0	...	0	0	0	
4611352	2	2020	3	0.0	1.0	98.00	Restricted Coach Class	1	MIA-SFO	0	...	0	0	0	
4611354	2	2020	3	0.0	1.0	213.00	Restricted Coach Class	1	MIA-SFO	0	...	0	0	0	
4611368	2	2020	3	0.0	1.0	103.50	Restricted Coach Class	1	MIA-SFO	0	...	0	0	0	
4611372	2	2020	3	0.0	1.0	103.50	Restricted Coach Class	1	SFO-MIA	0	...	0	0	0	

831288 rows × 75 columns

In [40]:
```
df=encode_and_bind(df,"FareClass")
df=df.drop(["FareClass"],axis=1)
```

In [41]:
```
df
```

Out[41]:

	MktCoupons	Year	Quarter	BulkFare	Passengers	MktFare	Pandemic	route	RPCarrier_9E	RPCarrier_AA	...	route_TPA-LAX	route_TPA-SAN	route_TPA-SFO
1	3	2018	1	0.0	1.0	1338.27	0	LAX-FLL	0	1	...	0	0	0
4	1	2018	1	0.0	1.0	66.47	0	SMF-LAX	0	1	...	0	0	0
17	4	2018	1	0.0	1.0	5.54	0	SFO-TPA	0	1	...	0	0	0
22	1	2018	1	0.0	1.0	521.46	0	SFO-MIA	0	1	...	0	0	0
28	1	2018	1	0.0	1.0	1651.23	0	MIA-LAX	0	1	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4611350	2	2020	3	0.0	1.0	78.00	1	MIA-SFO	0	0	...	0	0	0
4611352	2	2020	3	0.0	1.0	98.00	1	MIA-SFO	0	0	...	0	0	0
4611354	2	2020	3	0.0	1.0	213.00	1	MIA-SFO	0	0	...	0	0	0
4611368	2	2020	3	0.0	1.0	103.50	1	MIA-SFO	0	0	...	0	0	0
4611372	2	2020	3	0.0	1.0	103.50	1	SFO-MIA	0	0	...	0	0	0

831288 rows × 81 columns

In [42]:
```
df.shape
```

Out[42]:
```
(831288, 81)
```

In [43]:
```
df.columns
```

Out[43]:
```
Index(['MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers', 'MktFare',
       'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA', 'RPCarrier_AS',
       'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL', 'RPCarrier_EV',
       'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA', 'RPCarrier_MQ',
       'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO', 'RPCarrier_QX',
       'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX', 'RPCarrier_WN',
       'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO', 'route_FLL-LAX',
       'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL', 'route_LAX-MCO',
       'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO', 'route_LAX-SJC',
       'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX', 'route_MCO-SAN',
       'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX', 'route_MIA-SFO',
       'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO', 'route_SAN-SFO',
       'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA', 'route_SFO-BUR',
       'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO', 'route_SFO-MIA',
       'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA', 'route_SFO-TPA',
       'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA', 'route_SMF-LAX',
       'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO', 'route_SNA-SJC',
       'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO', 'FareClass_Others',
       'FareClass_Restricted Business Class',
       'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
       'FareClass_Unrestricted Business Class',
       'FareClass_Unrestricted Coach Class',
       'FareClass_Unrestricted First Class'],
      dtype='object')
```

In [44]:
```
a = df['route'].unique()
```

In [45]:
```
grps=df.groupby('route')
```

In [46]:
```
for i in a:
    df2=grps.get_group(i)
    print(i," ",df2.shape)
    df2.to_csv(i+"-verWFC.csv")
```

```
LAX-FLL   (24378, 81)
SMF-LAX   (14029, 81)
SFO-TPA   (12300, 81)
SFO-MIA   (19131, 81)
MIA-LAX   (25887, 81)
MCO-LAX   (38976, 81)
MIA-SFO   (19880, 81)
LAX-MIA   (25906, 81)
SJC-SAN   (13685, 81)
PSP-SFO   (10495, 81)
SFO-PSP   (10351, 81)
SNA-SJC   (10180, 81)
MCO-SAN   (16188, 81)
MCO-SMF   (10926, 81)
FLL-SAN   (9928, 81)
SAN-MCO   (16339, 81)
SAN-TPA   (10308, 81)
TPA-SAN   (10013, 81)
SAN-SFO   (25252, 81)
SFO-SAN   (25034, 81)
SAN-SJC   (13713, 81)
SAN-SMF   (12434, 81)
SMF-SAN   (12229, 81)
SFO-LAX   (45906, 81)
SFO-FLL   (16720, 81)
FLL-SFO   (16810, 81)
SFO-MCO   (21819, 81)
MCO-SFO   (22209, 81)
TPA-SFO   (12075, 81)
LAX-SFO   (46379, 81)
SFO-SNA   (21449, 81)
TPA-LAX   (18544, 81)
SJC-SNA   (10515, 81)
SMF-MCO   (10793, 81)
FLL-LAX   (24332, 81)
LAX-MCO   (39598, 81)
LAX-TPA   (19197, 81)
SJC-LAX   (18130, 81)
LAX-SMF   (14307, 81)
LAX-OAK   (11158, 81)
LAX-SJC   (18419, 81)
SFO-BUR   (11437, 81)
BUR-SFO   (11505, 81)
OAK-LAX   (10894, 81)
SNA-SFO   (21530, 81)
```

Machine Hack Dataset:

In [1]:
```
pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.0.7-py2.py3-none-any.whl (243 kB)
     |████████████████████████████████| 243 kB 862 kB/s eta 0:00:01
Collecting et-xmlfile
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.0.7
WARNING: You are using pip version 21.0.1; however, version 21.1.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
train_data=pd.read_excel('../input/flight-fare-prediction-mh/Data_Train.xlsx',engine='openpyxl')
#test_data=pd.read_excel('../input/flight-fare-prediction-mh/Test_set.xlsx',engine='openpyxl')
train_data.shape#test_data.shape
```

Out[2]:
```
(10683, 11)
```

In [3]:
```
train_data.head()
```

Out[3]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

In [4]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

In [5]:
```
train_data.isnull().values.any()
#test_data.isnull().values.any()
```

Out[5]:
```
True
```

In [6]:
```
train_data.isnull().sum()
```

Out[6]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

In [7]:
```
train_data['Duration'].value_counts()
```

Out[7]:
```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
47h          1
31h 30m      1
35h 20m      1
33h 20m      1
35h 35m      1
Name: Duration, Length: 368, dtype: int64
```

In [8]:
```
train_data.dropna(inplace=True)
```

In [9]:
```
train_data.shape
```

Out[9]:
```
(10682, 11)
```

In [10]:
```
train_data['Journey_day']=pd.to_datetime(train_data.Date_of_Journey,format='%d/%m/%Y').dt.day

train_data['Journey_month']=pd.to_datetime(train_data.Date_of_Journey,format='%d/%m/%Y').dt.month
```

In [11]:
```
train_data.head()
```

Out[11]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3

In [12]:
```
train_data.drop(['Date_of_Journey'],axis=1,inplace=True)
```

In [13]:
```
train_data['Departure_hour']=pd.to_datetime(train_data['Dep_Time']).dt.hour
train_data['Departure_Minute']=pd.to_datetime(train_data['Dep_Time']).dt.minute
train_data.drop(['Dep_Time'],axis=1,inplace=True)
```

In [14]:
```
train_data.head()
```

Out[14]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Departure_hour	Departure_
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3	22	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	5	5	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No info	13882	9	6	9	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No info	6218	12	5	18	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No info	13302	1	3	16	

In [15]:
```
train_data['Arrival_Hour']=pd.to_datetime(train_data['Arrival_Time']).dt.hour
train_data['Arrival_Minute']=pd.to_datetime(train_data['Arrival_Time']).dt.minute
train_data.drop(['Arrival_Time'],axis=1,inplace=True)
```

In [16]:
```
train_data.head()
```

Out[16]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arriva
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	20	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5	50	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12	5	18	5	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16	50	

In [17]:
```
duration = list(train_data["Duration"])
for i in range(len(duration)):
    if len(duration[i].split()) != 2:
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"
        else:
            duration[i] = "0h " + duration[i]
duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))
```

In [18]:
```
train_data['Duration_Hours']=duration_hours
train_data['Duration_Min']=duration_mins
```

In [19]:
```
train_data.head()
```

Out[19]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arriv
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	20	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5	50	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12	5	18	5	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16	50	

In [20]:
```
train_data.drop(['Duration'],axis=1,inplace=True)
```

In [21]:
```
train_data.head()
```

Out[21]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arrival_Hour	A
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	1	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50	13	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25	4	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	18	5	23	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	16	50	21	

In [22]:
```
train_data['Airline'].value_counts()
```

Out[22]:
```
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

In [23]:
```
Airline=train_data[['Airline']]
Airline=pd.get_dummies(Airline,drop_first=True)
Airline.head()
```

Out[23]:

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Airline_Vis Prem econ
0	0	0	1	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	

In [24]:
```
train_data['Source'].value_counts()
```

Out[24]:
```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [25]:
```
Source=train_data[['Source']]
Source=pd.get_dummies(Source,drop_first=True)
Source.head()
```

Out[25]:

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

In [26]:
```
train_data['Destination'].value_counts()
```

Out[26]:
```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [27]:
```
Destination=train_data['Destination']
Destination=pd.get_dummies(Destination,drop_first=1)
Destination.head()
```

Out[27]:

	Cochin	Delhi	Hyderabad	Kolkata	New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

In [28]:
```
train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

In [29]:
```
train_data["Total_Stops"].value_counts()
```

Out[29]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [30]:
```
train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

In [31]:
```
train_data.head()
```

Out[31]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arrival_Hour	Arrival_Minute	Duration_
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35	

In [32]:
```
data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
```

In [33]:
```
data_train.head()
```

Out[33]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arrival_Hour	...	Airline_Vistara Premium economy	Sourc
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	...	0	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	...	0	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	...	0	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	...	0	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	...	0	

5 rows × 33 columns

In [34]:
```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
```

In [35]:
```
data_train.head()
```

Out[35]:

	Total_Stops	Price	Journey_day	Journey_month	Departure_hour	Departure_Minute	Arrival_Hour	Arrival_Minute	Duration_Hours	Duration_Min	...	Airline
0	0	3897	24	3	22	20	1	10	2	50	...	
1	2	7662	1	5	5	50	13	15	7	25	...	
2	2	13882	9	6	9	25	4	25	19	0	...	
3	1	6218	12	5	18	5	23	30	5	25	...	
4	1	13302	1	3	16	50	21	35	4	45	...	

5 rows × 30 columns

In [36]:
```
data_train.to_csv("train_data.csv")
```

Ease-My-Trip Dataset:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df=pd.read_csv("../input/easemytrip-flight-fare-travel-listings/marketing_sample_for_easemytrip_in-easemytrip_flights_fares__2026
df
```

Out[2]:

	Uniq Id	Crawl Timestamp	Source	Layover1	Layover2	Layover3	Destination	Flight Operator	Flight Number	Departure Date	Dep
0	53f192419756cfa95baa474506354f5	2020-04-01 01:23:42 +0000	Delhi	Patna	NaN	NaN	Guwahati	SpiceJet|SpiceJet	SG-8751|SG-426	23Apr2020	
1	7bb845050e88cc6a4f17abf24794c73b	2020-04-01 00:59:51 +0000	Mumbai	Bangalore	Delhi	NaN	Kochi	Vistara|Vistara|Vistara	UK-851|UK-812|UK-885	18Apr2020	
2	c9a84cc24d8a350eb41555c7bb76def6	2020-04-01 00:56:35 +0000	Ahmedabad	Delhi	Kolkata	NaN	Guwahati	Air India|Air India|Air India	AI-18|AI-401|AI-729	25Apr2020	
3	0c0a87ac2229b8c3f1b322baf1fc9f4e	2020-04-01 00:42:35 +0000	Kolkata	Delhi	NaN	NaN	Kochi	SpiceJet|SpiceJet	SG-254|SG-8561	16Apr2020	
4	b8f8af12fd6d9bf2f24389f26488478c	2020-04-01 01:52:13 +0000	Indore	Mumbai	NaN	NaN	Chennai	Indigo|Indigo	6E-5321|6E-323	18Apr2020	
...	...	...	...	...	...	...	...	...	...	...	
29995	35c0bbbb50478edb124b6987ae54864a	2020-04-01 01:50:52 +0000	Ranchi	Delhi	NaN	NaN	Chennai	Indigo|Indigo	6E-436|6E-2162	18Apr2020	
29996	80c354a117f6abea283fd44dd25a95d6	2020-04-01 01:11:27 +0000	Ahmedabad	Pune	NaN	NaN	Bengaluru	SpiceJet|SpiceJet	SG-6630|SG-517	16Apr2020	
29997	e26c8fd1922e77256b458da1c5895074	2020-04-01 00:57:21 +0000	Mumbai	Delhi	NaN	NaN	Ranchi	Vistara|Vistara	UK-970|UK-751	18Apr2020	
29998	6a81ee6ea557a970dab4499550f27a6c	2020-04-01 01:46:12 +0000	Bhopal	Raipur	Mumbai	NaN	Shirdi	Air India|Air India|Air India	AI-9683|AI-651|AI-9653	16Apr2020	
29999	29610e1ac73f6b2bbf2827ba3d8f8f33	2020-04-01 01:14:48 +0000	Delhi	NaN	NaN	NaN	Hyderabad	GoAir	G8-422	18Apr2020	

30000 rows × 16 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Uniq Id', 'Crawl Timestamp', 'Source', 'Layover1', 'Layover2',
       'Layover3', 'Destination', 'Flight Operator', 'Flight Number',
       'Departure Date', 'Departure Time', 'Arrival Date', 'Arrival Time',
       'Total Time', 'Number Of Stops', 'Fare'],
      dtype='object')
```

In [4]:
```
df=df.drop(['Uniq Id','Flight Number'],axis=1)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Crawl Timestamp', 'Source', 'Layover1', 'Layover2', 'Layover3',
       'Destination', 'Flight Operator', 'Departure Date', 'Departure Time',
       'Arrival Date', 'Arrival Time', 'Total Time', 'Number Of Stops',
       'Fare'],
      dtype='object')
```

In [6]:
```
df[df['Number Of Stops']==1].shape
```

Out[6]:
```
(16238, 14)
```

In [7]:
```
df1=df[np.logical_or(df['Number Of Stops']==1 , df['Number Of Stops']==0)]
```

In [8]:
```
df1
```

Out[8]:

	Crawl Timestamp	Source	Layover1	Layover2	Layover3	Destination	Flight Operator	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare
0	2020-04-01 01:23:42 +0000	Delhi	Patna	NaN	NaN	Guwahati	SpiceJet|SpiceJet	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0
3	2020-04-01 00:42:35 +0000	Kolkata	Delhi	NaN	NaN	Kochi	SpiceJet|SpiceJet	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0
4	2020-04-01 01:52:13 +0000	Indore	Mumbai	NaN	NaN	Chennai	Indigo|Indigo	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0
7	2020-04-01 01:20:18 +0000	Kochi	Mumbai	NaN	NaN	Delhi	SpiceJet|SpiceJet	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0
8	2020-04-01 01:10:52 +0000	Vadodara	Delhi	NaN	NaN	Mumbai	Air India|Air India	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29994	2020-04-01 02:16:24 +0000	Abudhabi	Colombo	NaN	NaN	Chennai	SriLankan Airlines|SriLankan Airlines	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0
29995	2020-04-01 01:50:52 +0000	Ranchi	Delhi	NaN	NaN	Chennai	Indigo|Indigo	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0
29996	2020-04-01 01:11:27 +0000	Ahmedabad	Pune	NaN	NaN	Bengaluru	SpiceJet|SpiceJet	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0
29997	2020-04-01 00:57:21 +0000	Mumbai	Delhi	NaN	NaN	Ranchi	Vistara|Vistara	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0
29999	2020-04-01 01:14:48 +0000	Delhi	NaN	NaN	NaN	Hyderabad	GoAir	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0

17848 rows × 14 columns

In [9]:
```
df1['Flight Operator'].unique()
```

Out[9]:
```
array(['SpiceJet|SpiceJet', 'Indigo|Indigo', 'Air India|Air India',
       'GoAir', 'British Airways|British Airways', 'AirAsia', 'Indigo',
       'Vistara|Vistara', 'AirAsia|AirAsia', 'GoAir|GoAir',
       'Etihad Airways|Etihad Airways',
       'National Air Services|National Air Services', 'SpiceJet',
       'Qatar Airways|Qatar Airways', 'AirArabia|AirArabia', 'Vistara',
       'Lufthansa Airways|Air India', 'Virgin Atlantic Airways|Air India',
       'Turkish Airlines|Turkish Airlines',
       'LOT Polish Airlines|LOT Polish Airlines',
       'SriLankan Airlines|SriLankan Airlines', 'Air India',
       'Nepal Airways', 'Gulf Airways|Gulf Airways',
       'Ethiopian Airlines|Ethiopian Airlines',
       'Saudi Arabian Airlines|Saudi Arabian Airlines', 'Etihad Airways',
       'EVA Airways|China Eastern Airlines', 'SriLankan Airlines',
       'Virgin Atlantic Airways|Virgin Atlantic Airways', 'Trujet',
       'Saudi Arabian Airlines', 'Delta Airlines|Delta Airlines',
       'American Airlines|American Airlines',
       'Lufthansa Airways|Etihad Airways', 'Gulf Airways|Air India',
       'Virgin Atlantic Airways|Vistara',
       'Malaysia Airlines|SriLankan Airlines',
       'Lufthansa Airways|LOT Polish Airlines',
       'Iberia Airlines|Air India', 'Sichuan Airlines|Air India',
       'Hong Kong Airlines|All Nippon Airways',
       'All Nippon Airways|All Nippon Airways', 'Alitalia|Air India',
       'Malaysia Airlines|Air India', 'Kenya Airways|Kenya Airways',
       'Singapore Airlines|Singapore Airlines', 'Rwandair|Air India',
       'Biman Bangladesh|Biman Bangladesh', 'All Nippon Airways|Vistara',
       'Ethiopian Airlines|Air India', 'Cathay Pacific Airways|Air India',
       'Thai Airways International|All Nippon Airways',
       'British Airways|Air India', 'Scandinavian Airlines|Air India',
       'Shanghai Airlines|Cathay Pacific Airways', 'Air India Express',
       'United Airlines|Air India', 'National Air Services',
       'Lufthansa Airways|British Airways', 'FlyDubai',
       'Shanghai Airlines|China Eastern Airlines',
       'Saudi Arabian Airlines|Air India',
       'Ukraine Intl Airlines|Ukraine Intl Airlines', 'Gulf Airways',
       'AirArabia', 'Gulf Airways|Etihad Airways',
       'Biman Bangladesh|Vistara',
       'Cathay Pacific Airways|All Nippon Airways',
       'Turkish Airlines|Air India', 'China Eastern Airlines|Air India',
       'All Nippon Airways|Air Pacific', 'British Airways|Vistara',
       'Air Canada|Air India', 'Alitalia|Virgin Atlantic Airways',
       'Swiss|Air India', 'Etihad Airways|Air India',
       'American Airlines|British Airways', 'Jetstar Asia|Air India',
       'Virgin Atlantic Airways', 'Singapore Airlines|Air India',
       'Austrian Airways|Air India', 'Oman Air|Qatar Airways',
       'Nepal Airways|Nepal Airways', 'Oman Air|Oman Air',
       'Lufthansa Airways|Lufthansa Airways', 'Ethiopian Airlines',
       'KLM Airlines|Virgin Atlantic Airways',
       'All Nippon Airways|Air India', 'Biman Bangladesh',
       'British Airways', 'Air France|British Airways',
       'Etihad Airways|SriLankan Airlines',
       'Scandinavian Airlines|Virgin Atlantic Airways',
       'China Eastern Airlines|China Eastern Airlines',
       'All Nippon Airways', 'EVA Airways|Cathay Pacific Airways',
       'Hong Kong Airlines|Cathay Pacific Airways',
       'Air Canada|Air Canada', 'SriLankan Airlines|Air India',
       'Turkish Airlines', 'Korean Airways|All Nippon Airways',
       'Lufthansa Airways|Virgin Atlantic Airways',
       'China Southern Airlines|China Eastern Airlines',
       'Air India Express|Air India Express', 'Swiss|British Airways',
       'Air France|Air India', 'Trujet|Trujet',
       'Ethiopian Airlines|Vistara', 'Korean Airways|Etihad Airways',
       'SriLankan Airlines|Biman Bangladesh', 'Jetstar Airways|Air India',
       'Oman Air', 'Qantas Airways|Air India',
       'Brussels Airlines|Air India', 'China Airlines|Air India',
       'Kenya Airways|Air India', 'Dragonair|Cathay Pacific Airways',
       'Oman Air|Air India', 'Singapore Airlines|Vistara',
       'Jazeera Airways', 'Deutsche Bahn Ag|Air India',
       'Myanmar Airways Intl', 'Kenya Airways',
       'Silk Air|Singapore Airlines', 'Etihad Airways|British Airways',
       'China Southern Airlines|Etihad Airways',
       'Austrian Airways|LOT Polish Airlines'], dtype=object)
```

In [10]:
```
df1['Flight Operator']=df1['Flight Operator']+'|'
val=df1['Flight Operator'].str.split("|")
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [11]:
```
val
```

Out[11]:
```
0                          [SpiceJet, SpiceJet, ]
3                          [SpiceJet, SpiceJet, ]
4                             [Indigo, Indigo, ]
7                          [SpiceJet, SpiceJet, ]
8                        [Air India, Air India, ]
                              ...
29994    [SriLankan Airlines, SriLankan Airlines, ]
29995                         [Indigo, Indigo, ]
29996                      [SpiceJet, SpiceJet, ]
29997                        [Vistara, Vistara, ]
29999                                 [GoAir, ]
Name: Flight Operator, Length: 17848, dtype: object
```

In [12]:
```
v1=[]
v2=[]
for i in val:
    v1.append(i[0])
    v2.append(i[1])
```

In [13]:
```
df1['F1']=v1
df1['F2']=v2
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [14]:
```
df1.columns
```

Out[14]:
```
Index(['Crawl Timestamp', 'Source', 'Layover1', 'Layover2', 'Layover3',
       'Destination', 'Flight Operator', 'Departure Date', 'Departure Time',
       'Arrival Date', 'Arrival Time', 'Total Time', 'Number Of Stops', 'Fare',
       'F1', 'F2'],
      dtype='object')
```

In [15]:
```
df1.shape
```

Out[15]:
```
(17848, 16)
```

In [16]:
```
df1=df1[np.logical_or(df1['Number Of Stops']==0,np.logical_and(df1['Number Of Stops']==1,df1['F1']==df1['F2']))]
```

In [17]:
```
df1
```

Out[17]:

	Crawl Timestamp	Source	Layover1	Layover2	Layover3	Destination	Flight Operator	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare
0	2020-04-01 01:23:42 +0000	Delhi	Patna	NaN	NaN	Guwahati	SpiceJet|SpiceJet|	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0
3	2020-04-01 00:42:35 +0000	Kolkata	Delhi	NaN	NaN	Kochi	SpiceJet|SpiceJet|	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0
4	2020-04-01 01:52:13 +0000	Indore	Mumbai	NaN	NaN	Chennai	Indigo|Indigo|	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0
7	2020-04-01 01:20:18 +0000	Kochi	Mumbai	NaN	NaN	Delhi	SpiceJet|SpiceJet|	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0
8	2020-04-01 01:10:52 +0000	Vadodara	Delhi	NaN	NaN	Mumbai	Air India|Air India|	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29994	2020-04-01 02:16:24 +0000	Abudhabi	Colombo	NaN	NaN	Chennai	SriLankan Airlines|SriLankan Airlines|	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0
29995	2020-04-01 01:50:52 +0000	Ranchi	Delhi	NaN	NaN	Chennai	Indigo|Indigo|	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0
29996	2020-04-01 01:11:27 +0000	Ahmedabad	Pune	NaN	NaN	Bengaluru	SpiceJet|SpiceJet|	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0
29997	2020-04-01 00:57:21 +0000	Mumbai	Delhi	NaN	NaN	Ranchi	Vistara|Vistara|	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0
29999	2020-04-01 01:14:48 +0000	Delhi	NaN	NaN	NaN	Hyderabad	GoAir|	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0

17340 rows × 16 columns

In [18]:
```
df1['F1'].unique()
```

Out[18]:
```
array(['SpiceJet', 'Indigo', 'Air India', 'GoAir', 'British Airways',
       'AirAsia', 'Vistara', 'Etihad Airways', 'National Air Services',
       'Qatar Airways', 'AirArabia', 'Turkish Airlines',
       'LOT Polish Airlines', 'SriLankan Airlines', 'Nepal Airways',
       'Gulf Airways', 'Ethiopian Airlines', 'Saudi Arabian Airlines',
       'Virgin Atlantic Airways', 'Trujet', 'Delta Airlines',
       'American Airlines', 'All Nippon Airways', 'Kenya Airways',
       'Singapore Airlines', 'Biman Bangladesh', 'Air India Express',
       'FlyDubai', 'Ukraine Intl Airlines', 'Oman Air',
       'Lufthansa Airways', 'China Eastern Airlines', 'Air Canada',
       'Jazeera Airways', 'Myanmar Airways Intl'], dtype=object)
```

In [19]:
```
df1=df1.drop(['Layover2', 'Layover3','Flight Operator','F2'],axis=1)
```

In [20]:
```
df1
```

Out[20]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1
0	2020-04-01 01:23:42 +0000	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet
3	2020-04-01 00:42:35 +0000	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet
4	2020-04-01 01:52:13 +0000	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo
7	2020-04-01 01:20:18 +0000	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet
8	2020-04-01 01:10:52 +0000	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India
...	...	...	...	...	...	...	...	...	...	...	...	...
29994	2020-04-01 02:16:24 +0000	Abudhabi	Colombo	Chennai	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0	SriLankan Airlines
29995	2020-04-01 01:50:52 +0000	Ranchi	Delhi	Chennai	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0	Indigo
29996	2020-04-01 01:11:27 +0000	Ahmedabad	Pune	Bengaluru	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0	SpiceJet
29997	2020-04-01 00:57:21 +0000	Mumbai	Delhi	Ranchi	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0	Vistara
29999	2020-04-01 01:14:48 +0000	Delhi	NaN	Hyderabad	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0	GoAir

17340 rows × 12 columns

In [21]:
```
df1.dtypes
```

Out[21]:
```
Crawl Timestamp     object
Source              object
Layover1            object
Destination         object
Departure Date      object
Departure Time      object
Arrival Date        object
Arrival Time        object
Total Time          object
Number Of Stops      int64
Fare               float64
F1                  object
dtype: object
```

In [22]:
```
from datetime import datetime
```

In [23]:
```
df1.head()
```

Out[23]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1
0	2020-04-01 01:23:42 +0000	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet
3	2020-04-01 00:42:35 +0000	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet
4	2020-04-01 01:52:13 +0000	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo
7	2020-04-01 01:20:18 +0000	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet
8	2020-04-01 01:10:52 +0000	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India

In [24]:
```
df1['Crawl Timestamp']=pd.to_datetime(df1['Crawl Timestamp'])
```

In [25]:
```
df1.head()
```

Out[25]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1
0	2020-04-01 01:23:42+00:00	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet
3	2020-04-01 00:42:35+00:00	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet
4	2020-04-01 01:52:13+00:00	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo
7	2020-04-01 01:20:18+00:00	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet
8	2020-04-01 01:10:52+00:00	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India

In [26]:
```
df1.dtypes
```

Out[26]:
```
Crawl Timestamp    datetime64[ns, UTC]
Source                          object
Layover1                        object
Destination                     object
Departure Date                  object
Departure Time                  object
Arrival Date                    object
Arrival Time                    object
Total Time                      object
Number Of Stops                  int64
Fare                           float64
F1                              object
dtype: object
```

In [27]:
```
df1['Departure']=df1['Departure Date']+" "+df1['Departure Time']
```

In [28]:
```
df1['Departure']=pd.to_datetime(df1['Departure'])
df1.dtypes
```

Out[28]:
```
Crawl Timestamp    datetime64[ns, UTC]
Source                          object
Layover1                        object
Destination                     object
Departure Date                  object
Departure Time                  object
Arrival Date                    object
Arrival Time                    object
Total Time                      object
Number Of Stops                  int64
Fare                           float64
F1                              object
Departure              datetime64[ns]
dtype: object
```

In [29]:
```
df1.head()
```

Out[29]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1	Departure
0	2020-04-01 01:23:42+00:00	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet	2020-04-23 13:10:00
3	2020-04-01 00:42:35+00:00	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet	2020-04-16 18:00:00
4	2020-04-01 01:52:13+00:00	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo	2020-04-18 07:25:00
7	2020-04-01 01:20:18+00:00	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet	2020-04-27 05:35:00
8	2020-04-01 01:10:52+00:00	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India	2020-04-18 20:20:00

In [30]:
```
df1['Crawl Timestamp']=pd.to_datetime(df1['Crawl Timestamp']).dt.tz_localize(None)
```

In [31]:
```
df1.dtypes
```

Out[31]:
```
Crawl Timestamp    datetime64[ns]
Source                     object
Layover1                   object
Destination                object
Departure Date             object
Departure Time             object
Arrival Date               object
Arrival Time               object
Total Time                 object
Number Of Stops             int64
Fare                      float64
F1                         object
Departure         datetime64[ns]
dtype: object
```

In [32]:
```
df1
```

Out[32]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1	Departure
0	2020-04-01 01:23:42	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet	2020-04-23 13:10:00
3	2020-04-01 00:42:35	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet	2020-04-16 18:00:00
4	2020-04-01 01:52:13	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo	2020-04-18 07:25:00
7	2020-04-01 01:20:18	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet	2020-04-27 05:35:00
8	2020-04-01 01:10:52	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India	2020-04-18 20:20:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
29994	2020-04-01 02:16:24	Abudhabi	Colombo	Chennai	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0	SriLankan Airlines	2020-04-23 23:00:00
29995	2020-04-01 01:50:52	Ranchi	Delhi	Chennai	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0	Indigo	2020-04-18 14:50:00
29996	2020-04-01 01:11:27	Ahmedabad	Pune	Bengaluru	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0	SpiceJet	2020-04-16 04:00:00
29997	2020-04-01 00:57:21	Mumbai	Delhi	Ranchi	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0	Vistara	2020-04-18 08:45:00
29999	2020-04-01 01:14:48	Delhi	NaN	Hyderabad	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0	GoAir	2020-04-18 20:55:00

17340 rows × 13 columns

In [33]:
```
delta = df1['Departure'] - df1['Crawl Timestamp']
```

In [34]:
```
delta
```

Out[34]:
```
0       22 days 11:46:18
3       15 days 17:17:25
4       17 days 05:32:47
7       26 days 04:14:42
8       17 days 19:09:08
              ...
29994   22 days 20:43:36
29995   17 days 12:59:08
29996   15 days 02:48:33
29997   17 days 07:47:39
29999   17 days 19:40:12
Length: 17340, dtype: timedelta64[ns]
```

In [35]:
```
d=[]
for i in delta:
    d.append(i.days)
```

In [36]:
```
df1['Days Left']=d
```

In [37]:
```
df1
```

Out[37]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1	Departure	Days Left
0	2020-04-01 01:23:42	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet	2020-04-23 13:10:00	22
3	2020-04-01 00:42:35	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet	2020-04-16 18:00:00	15
4	2020-04-01 01:52:13	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo	2020-04-18 07:25:00	17
7	2020-04-01 01:20:18	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet	2020-04-27 05:35:00	26
8	2020-04-01 01:10:52	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India	2020-04-18 20:20:00	17
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29994	2020-04-01 02:16:24	Abudhabi	Colombo	Chennai	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0	SriLankan Airlines	2020-04-23 23:00:00	22
29995	2020-04-01 01:50:52	Ranchi	Delhi	Chennai	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0	Indigo	2020-04-18 14:50:00	17
29996	2020-04-01 01:11:27	Ahmedabad	Pune	Bengaluru	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0	SpiceJet	2020-04-16 04:00:00	15
29997	2020-04-01 00:57:21	Mumbai	Delhi	Ranchi	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0	Vistara	2020-04-18 08:45:00	17
29999	2020-04-01 01:14:48	Delhi	NaN	Hyderabad	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0	GoAir	2020-04-18 20:55:00	17

17340 rows × 14 columns

In [38]:
```
days1=[]
m1=[]
yrs=[]
hrs=[]
mins=[]
for i in df1['Departure']:
    days1.append(i.day)
    yrs.append(i.year)
    m1.append(i.month)
    hrs.append(i.hour)
    mins.append(i.minute)
```

In [39]:
```
df1["Departure_day"]=days1
df1["Departure_month"]=m1
df1["Departure_year"]=yrs
df1["Departure_hour"]=hrs
df1["Departure_minute"]=mins
```

In [40]:
```
df1
```

Out[40]:

	Crawl Timestamp	Source	Layover1	Destination	Departure Date	Departure Time	Arrival Date	Arrival Time	Total Time	Number Of Stops	Fare	F1	Departure	Days Left	Departu
0	2020-04-01 01:23:42	Delhi	Patna	Guwahati	23Apr2020	13:10	23Apr2020	22:00	08h 50m	1	4378.0	SpiceJet	2020-04-23 13:10:00	22	
3	2020-04-01 00:42:35	Kolkata	Delhi	Kochi	16Apr2020	18:00	17Apr2020	11:10	17h 10m	1	5226.0	SpiceJet	2020-04-16 18:00:00	15	
4	2020-04-01 01:52:13	Indore	Mumbai	Chennai	18Apr2020	07:25	18Apr2020	13:35	06h 10m	1	3333.0	Indigo	2020-04-18 07:25:00	17	
7	2020-04-01 01:20:18	Kochi	Mumbai	Delhi	27Apr2020	05:35	27Apr2020	15:40	10h 05m	1	3563.0	SpiceJet	2020-04-27 05:35:00	26	
8	2020-04-01 01:10:52	Vadodara	Delhi	Mumbai	18Apr2020	20:20	19Apr2020	12:50	16h 30m	1	5309.0	Air India	2020-04-18 20:20:00	17	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
25554	2020-04-01 02:16:24	Abudhabi	Colombo	Chennai	23Apr2020	23:00	24Apr2020	08:50	08h 20m	1	35839.0	SriLankan Airlines	2020-04-23 23:00:00	22	
25555	2020-04-01 01:50:52	Ranchi	Delhi	Chennai	18Apr2020	14:50	18Apr2020	23:35	08h 45m	1	4161.0	Indigo	2020-04-18 14:50:00	17	
25556	2020-04-01 01:11:27	Ahmedabad	Pune	Bengaluru	16Apr2020	04:00	16Apr2020	17:05	13h 05m	1	2145.0	SpiceJet	2020-04-16 04:00:00	15	
25557	2020-04-01 00:57:21	Mumbai	Delhi	Ranchi	18Apr2020	08:45	19Apr2020	09:55	25h 10m	1	5009.0	Vistara	2020-04-18 08:45:00	17	
25559	2020-04-01 01:14:48	Delhi	NaN	Hyderabad	18Apr2020	20:55	18Apr2020	23:10	02h 15m	0	2351.0	GoAir	2020-04-18 20:55:00	17	

17340 rows × 19 columns

In [41]:
```
df1.columns
```

Out[41]:
```
Index(['Crawl Timestamp', 'Source', 'Layover1', 'Destination',
       'Departure Date', 'Departure Time', 'Arrival Date', 'Arrival Time',
       'Total Time', 'Number Of Stops', 'Fare', 'F1', 'Departure', 'Days Left',
       'Departure_day', 'Departure_month', 'Departure_year', 'Departure_hour',
       'Departure_minute'],
      dtype='object')
```

In [42]:
```
df1=df1.drop(['Crawl Timestamp','Departure Date', 'Departure Time', 'Arrival Date', 'Arrival Time',
       'Total Time','Departure'],axis=1)
```

In [43]:
```
df1.columns
```

Out[43]:
```
Index(['Source', 'Layover1', 'Destination', 'Number Of Stops', 'Fare', 'F1',
       'Days Left', 'Departure_day', 'Departure_month', 'Departure_year',
       'Departure_hour', 'Departure_minute'],
      dtype='object')
```

In [44]:
```
df1
```

Out[44]:

	Source	Layover1	Destination	Number Of Stops	Fare	F1	Days Left	Departure_day	Departure_month	Departure_year	Departure_hour	Departure_min
0	Delhi	Patna	Guwahati	1	4378.0	SpiceJet	22	23	4	2020	13	
3	Kolkata	Delhi	Kochi	1	5226.0	SpiceJet	15	16	4	2020	18	
4	Indore	Mumbai	Chennai	1	3333.0	Indigo	17	18	4	2020	7	
7	Kochi	Mumbai	Delhi	1	3563.0	SpiceJet	26	27	4	2020	5	
8	Vadodara	Delhi	Mumbai	1	5309.0	Air India	17	18	4	2020	20	
...	...	...	...	...	...	...	...	...	...	...	...	
29994	Abudhabi	Colombo	Chennai	1	35839.0	SriLankan Airlines	22	23	4	2020	23	
29995	Ranchi	Delhi	Chennai	1	4161.0	Indigo	17	18	4	2020	14	
29996	Ahmedabad	Pune	Bengaluru	1	2145.0	SpiceJet	15	16	4	2020	4	
29997	Mumbai	Delhi	Ranchi	1	5009.0	Vistara	17	18	4	2020	8	
29999	Delhi	NaN	Hyderabad	0	2351.0	GoAir	17	18	4	2020	20	

17340 rows × 12 columns

In [45]:
```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    return(res)
```

In [46]:
```
df1=encode_and_bind(df1,"F1")
df1=encode_and_bind(df1,"Source")
df1=encode_and_bind(df1,"Destination")
df1=encode_and_bind(df1,"Layover1")
```

In [47]:
```
df1
```

Out[47]:

	Source	Layover1	Destination	Number Of Stops	Fare	F1	Days Left	Departure_day	Departure_month	Departure_year	...	Layover1_Tiruchirapally	La
0	Delhi	Patna	Guwahati	1	4378.0	SpiceJet	22	23	4	2020	...	0	
3	Kolkata	Delhi	Kochi	1	5226.0	SpiceJet	15	16	4	2020	...	0	
4	Indore	Mumbai	Chennai	1	3333.0	Indigo	17	18	4	2020	...	0	
7	Kochi	Mumbai	Delhi	1	3563.0	SpiceJet	26	27	4	2020	...	0	
8	Vadodara	Delhi	Mumbai	1	5309.0	Air India	17	18	4	2020	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
29994	Abudhabi	Colombo	Chennai	1	35839.0	SriLankan Airlines	22	23	4	2020	...	0	
29995	Ranchi	Delhi	Chennai	1	4161.0	Indigo	17	18	4	2020	...	0	
29996	Ahmedabad	Pune	Bengaluru	1	2145.0	SpiceJet	15	16	4	2020	...	0	
29997	Mumbai	Delhi	Ranchi	1	5009.0	Vistara	17	18	4	2020	...	0	
29999	Delhi	NaN	Hyderabad	0	2351.0	GoAir	17	18	4	2020	...	0	

17340 rows × 369 columns

In [48]:
```
df1.to_csv("EaseMyTrip-discriptive.csv")
```

In [49]:
```
df1=df1.drop(['Source', 'Layover1', 'Destination','F1'],axis=1)
```

In [50]:
```
df1
```

Out[50]:

	Number Of Stops	Fare	Days Left	Departure_day	Departure_month	Departure_year	Departure_hour	Departure_minute	F1_Air Canada	F1_Air India	...	Layover1_Tiruchirapal
0	1	4378.0	22	23	4	2020	13	10	0	0	...	
3	1	5226.0	15	16	4	2020	18	0	0	0	...	
4	1	3333.0	17	18	4	2020	7	25	0	0	...	
7	1	3563.0	26	27	4	2020	5	35	0	0	...	
8	1	5309.0	17	18	4	2020	20	20	0	1	...	
...	...	...	...	...	...	...	...	...	...	...	...	
29994	1	35839.0	22	23	4	2020	23	0	0	0	...	
29995	1	4161.0	17	18	4	2020	14	50	0	0	...	
29996	1	2145.0	15	16	4	2020	4	0	0	0	...	
29997	1	5009.0	17	18	4	2020	8	45	0	0	...	
29999	0	2351.0	17	18	4	2020	20	55	0	0	...	

17340 rows × 365 columns

```
In [51]: df1.dtypes
```

```
Out[51]: Number Of Stops              int64
         Fare                       float64
         Days Left                    int64
         Departure_day                int64
         Departure_month              int64
                                      ...
         Layover1_Vadodara            uint8
         Layover1_Varanasi            uint8
         Layover1_Vijayawada          uint8
         Layover1_Vishakhapatnam      uint8
         Layover1_Warsaw              uint8
         Length: 365, dtype: object
```

```
In [52]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17340 entries, 0 to 29999
Columns: 365 entries, Number Of Stops to Layover1_Warsaw
dtypes: float64(1), int64(7), uint8(357)
memory usage: 7.1 MB
```

```
In [53]: df1.isnull().values.any()
```

```
Out[53]: False
```

```
In [54]: df1.to_csv("EaseMyTrip.csv")
```

Heat maps:

DB1B Dataset Normal View:

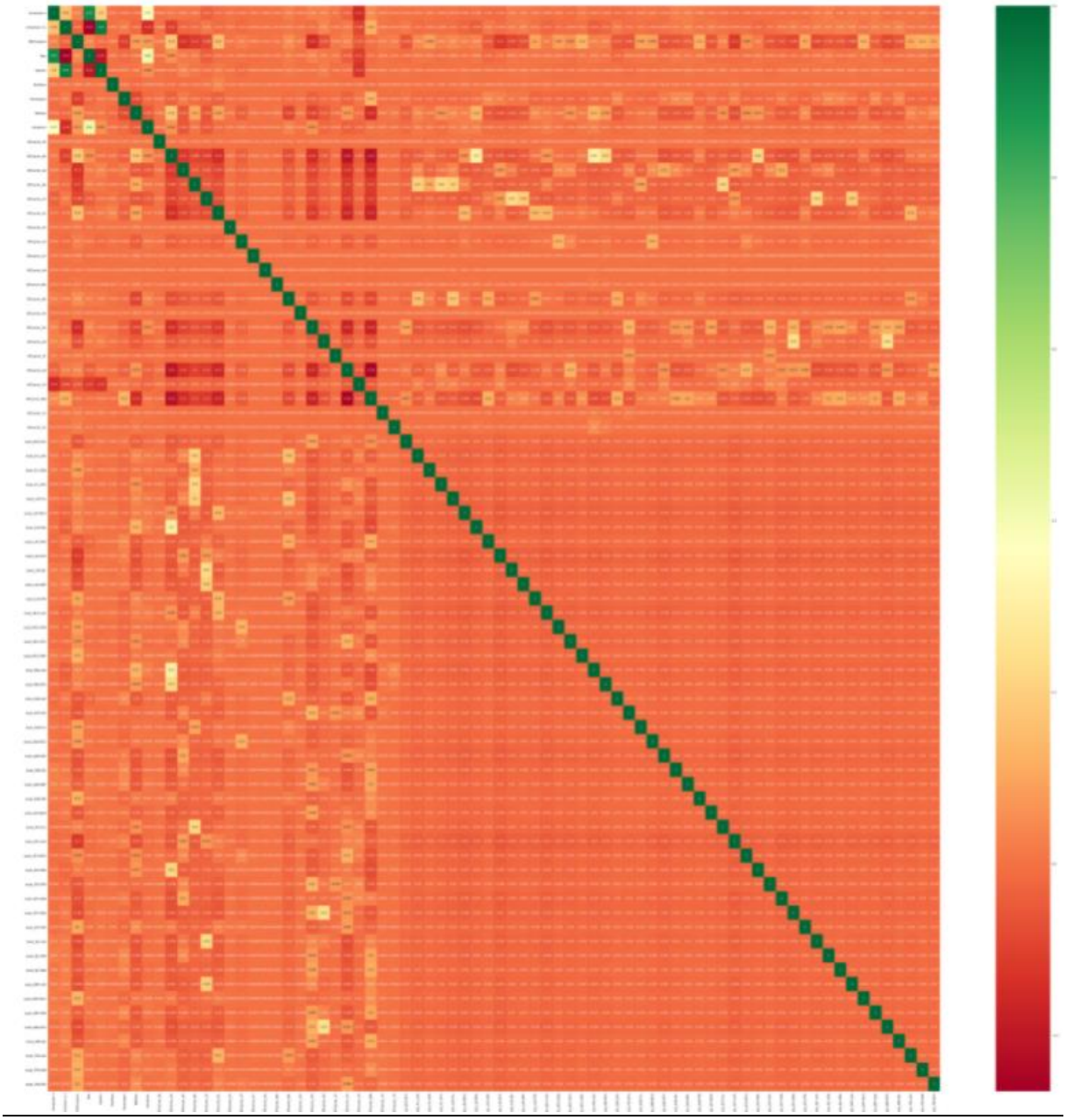
```
In [1]:
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import glob
g=glob.glob('../input/calfl1016combrts/*.csv')
```

```
In [2]:
```

```
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    k=file.split("/")
    print(k[-1])
    rt=k[-1].split(".")

    plt.figure(figsize = (70,70))
    sns.heatmap(df.corr(), annot = True, cmap = "RdYlGn")

    plt.show()
```

```
begin
1
sparce-all-rts-cal-fl.csv
```

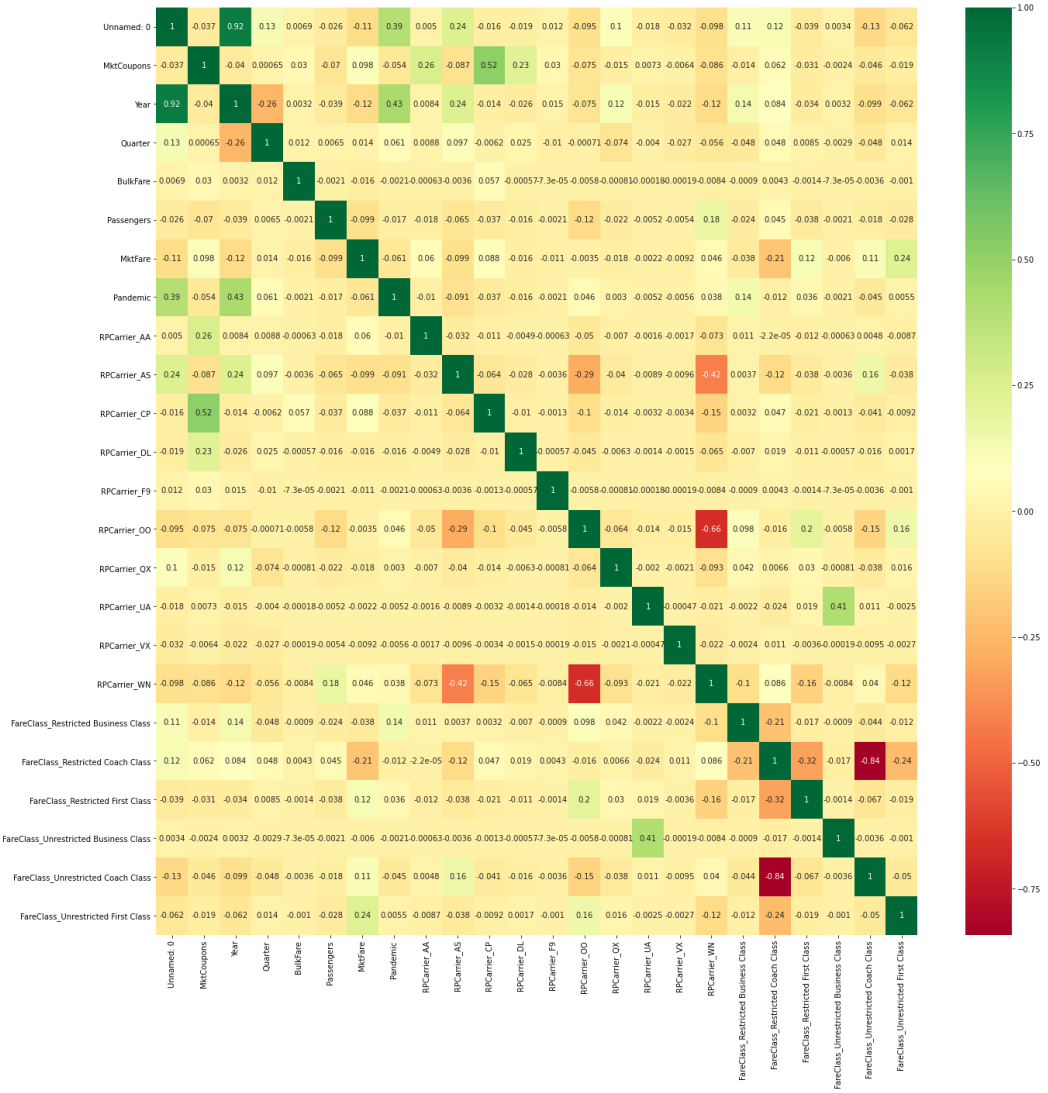


DB1B Dataset Sub-Route View:

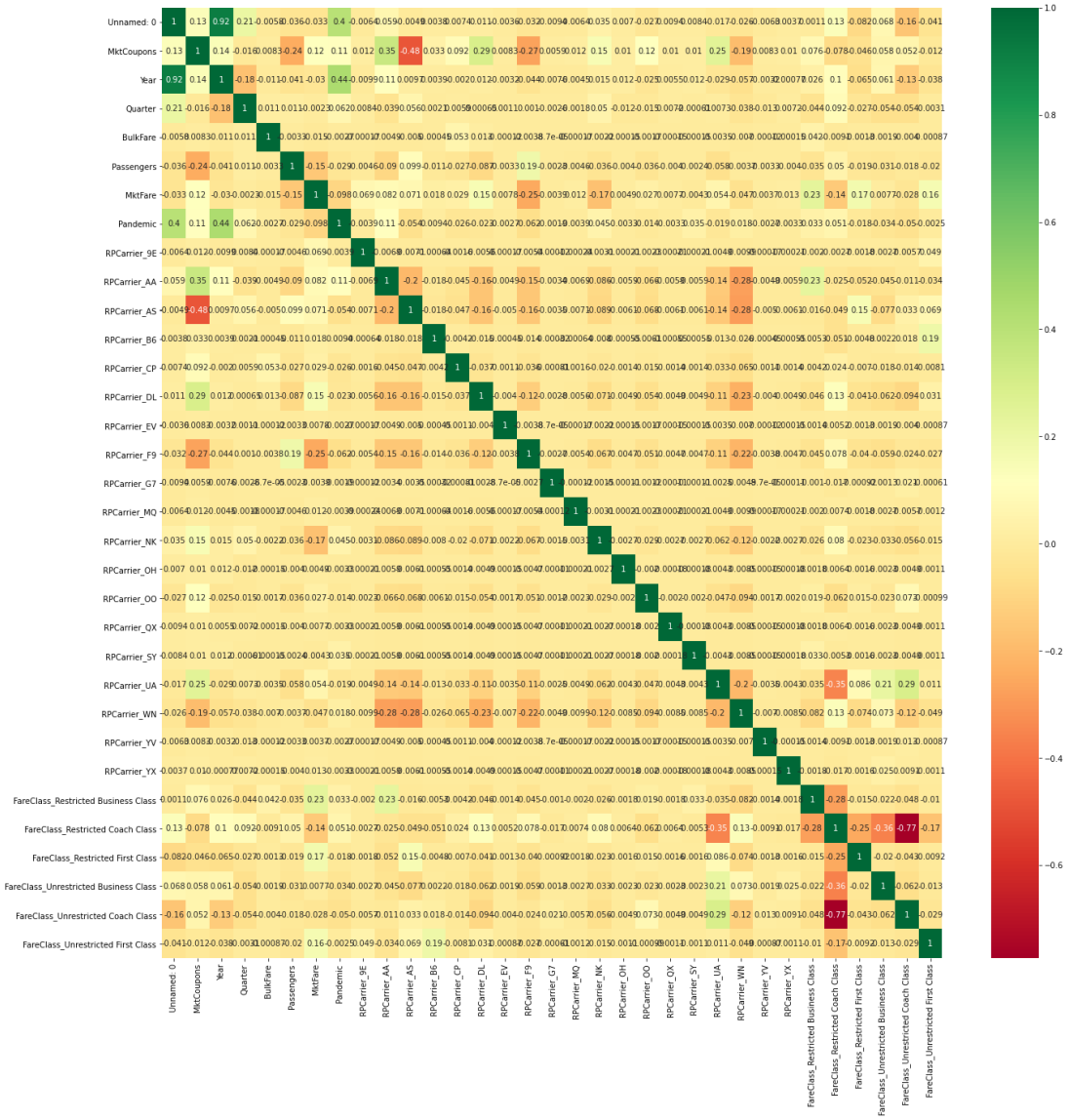
In [7]:
```
#heatmaps for subroutes(0-4)
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import glob
g=glob.glob('../input/sparce-routes-wfc/*.csv')
```

In [8]:
```
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    k=file.split("/")
    print(k[-1])
    rt=k[-1].split(".")

    nunique = df.apply(pd.Series.nunique)
    cols_to_drop = nunique[nunique == 1].index
    df=df.drop(cols_to_drop, axis=1)

    plt.figure(figsize = (22,22))
    sns.heatmap(df.corr(), annot = True, cmap = "RdYlGn")

    plt.show()

    if(F==5):
        break
```

```
begin
1
SAN-SJC-verWFC.csv
```

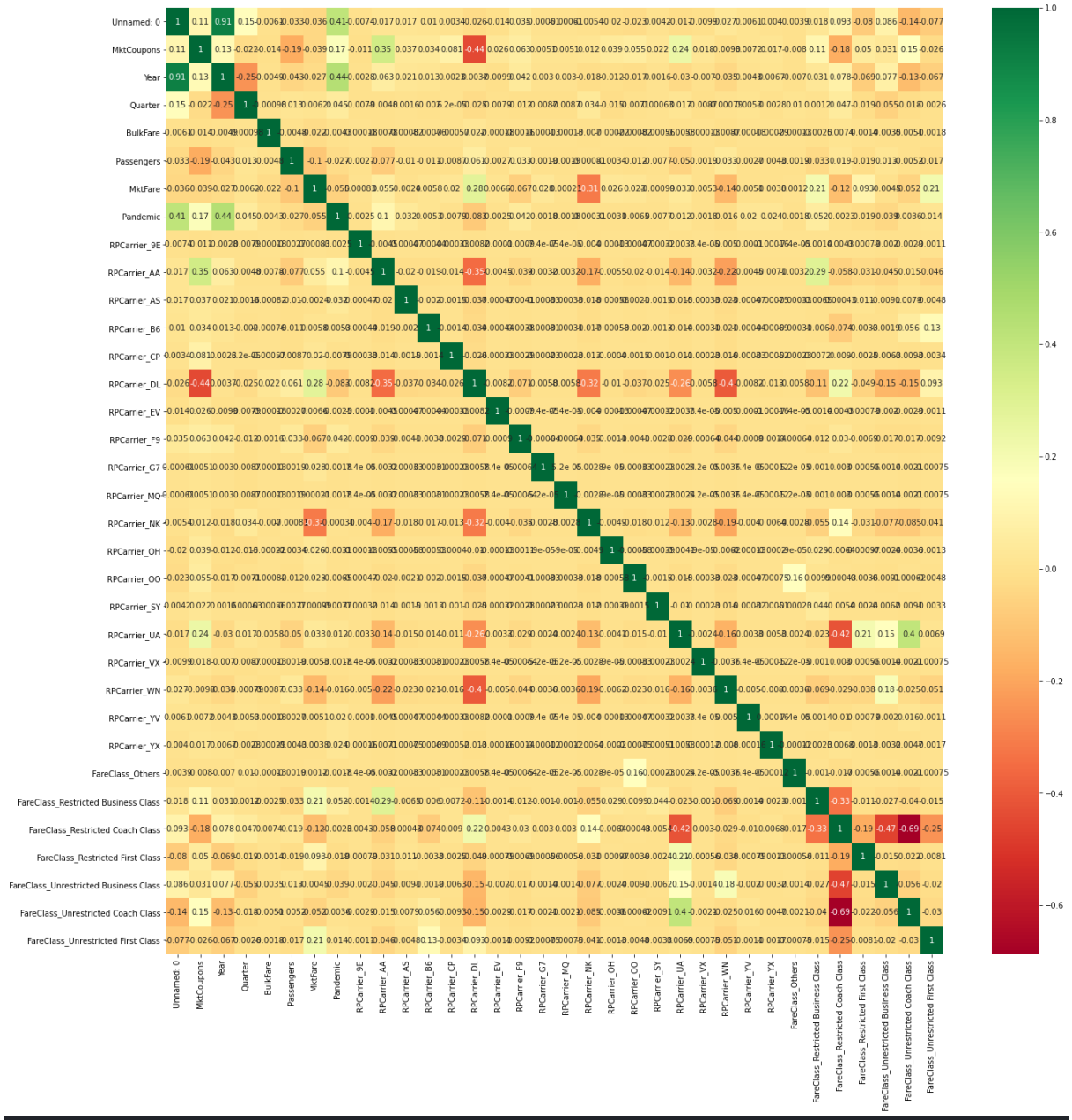



```
2
SAN-MCO-verWFC.csv
```

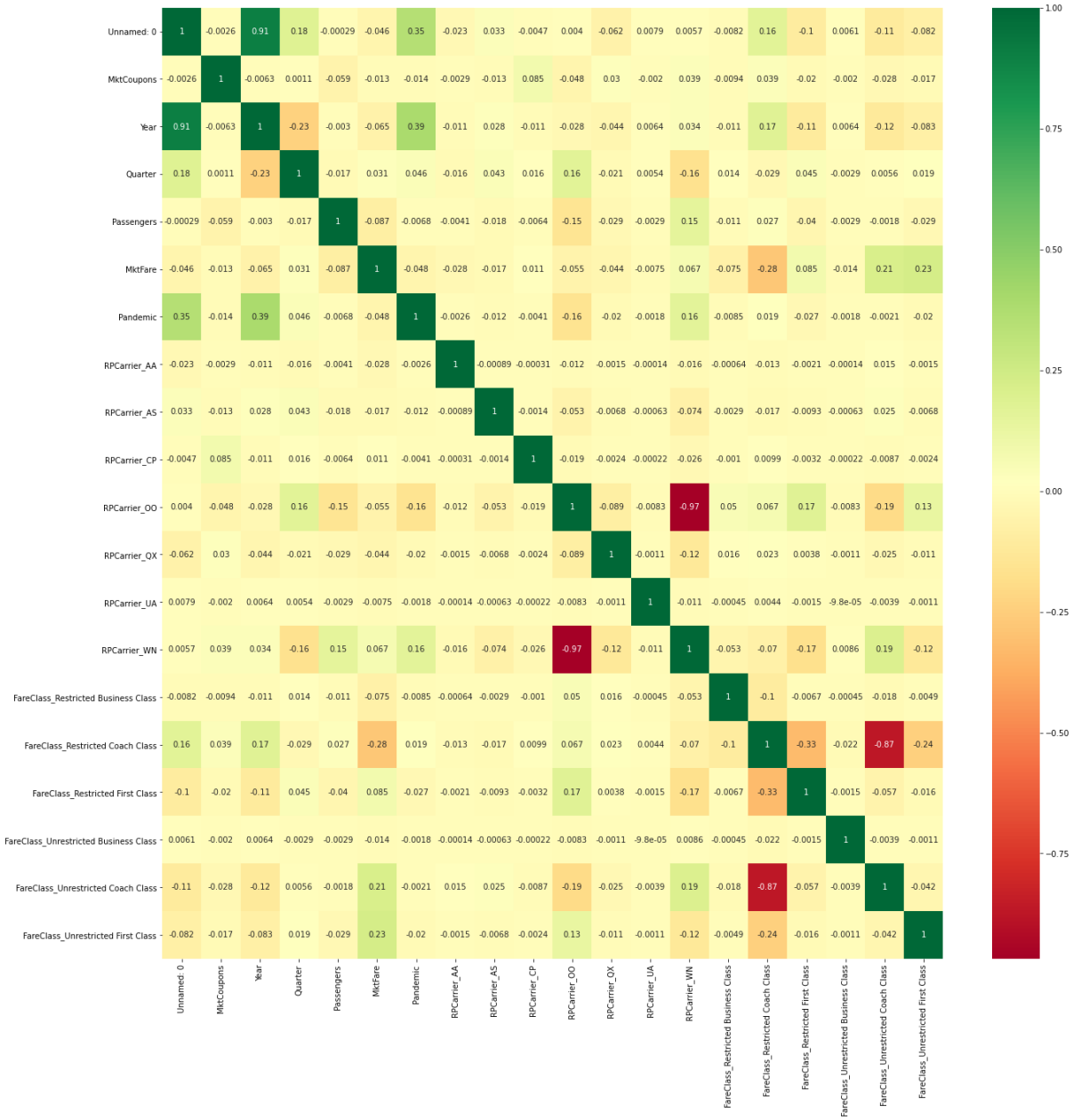


3

LAX-TPA-verWFC.csv




```
4
SNA-SJC-verWFC.csv
```



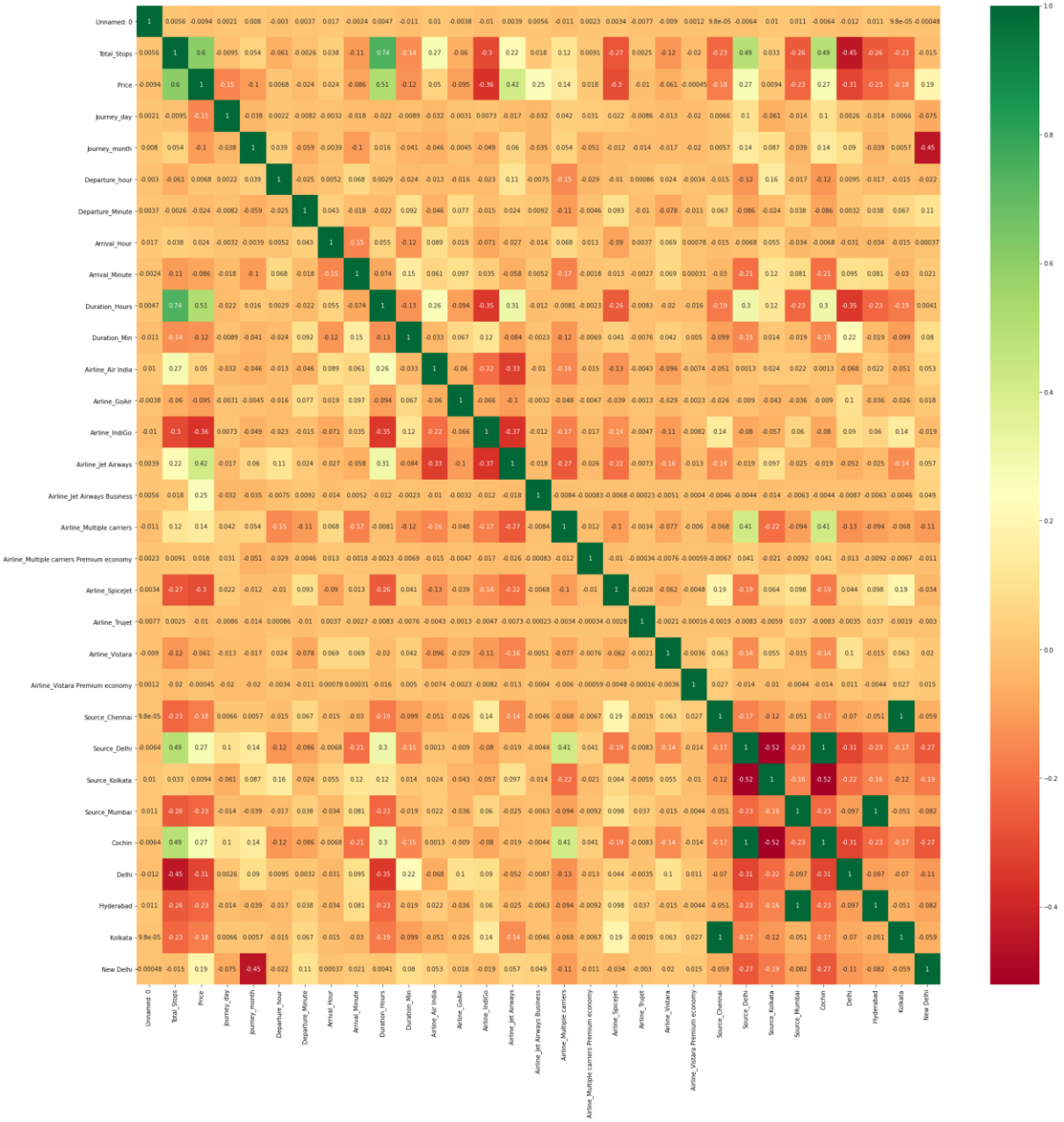
Machine Hack Dataset:

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
#df=pd.read_csv("../input/selectedfeatures-db1b/updatedsrv1.csv")#"../input/2019-q1-spacre-and-long/19_1S.csv",nrows=500000)#../i
#y = df['MktFare'].values
#X= df.drop(['MktFare'],axis=1).values
#df=pd.read_csv("../input/calfl1/cal-fl-rvi-all-sp.csv")#../input/dist-test/res2016C.csv")
from sklearn.model_selection import train_test_split
import glob
g=glob.glob('../input/dataset-2-for-airfare/*.csv')
```

In [2]:
```
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    k=file.split("/")
    print(k[-1])
    rt=k[-1].split(".")

    plt.figure(figsize = (30,30))
    sns.heatmap(df.corr(), annot = True, cmap = "RdYlGn")

    plt.show()
```

```
begin
1
train_data.csv
```

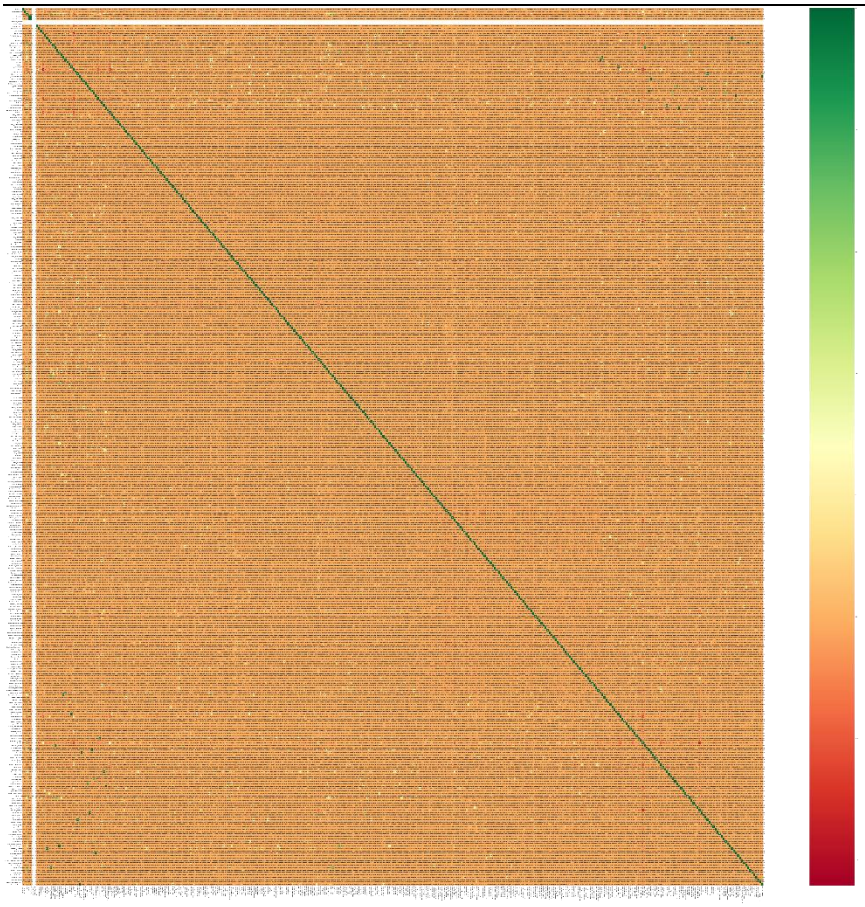


Ease-My-Trip Dataset:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
#df=pd.read_csv("../input/selectedfeatures-db1b/updatedsrv1.csv")#"../input/2019-q1-spacre-and-long/19_1S.csv",nrows=500000)#../i
#y = df['MktFare'].values
#X= df.drop(['MktFare'],axis=1).values
#df=pd.read_csv("../input/calfl1/cal-fl-rvi-all-sp.csv")#../input/dist-test/res2016C.csv")
from sklearn.model_selection import train_test_split
import glob
g=glob.glob('../input/easemytrip-ff-sample-data-processed/EaseMyTrip.csv')
```

In [2]:
```
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    k=file.split("/")
    print(k[-1])
    rt=k[-1].split(".")

    plt.figure(figsize = (150,150))
    sns.heatmap(df.corr(), annot = True, cmap = "RdYlGn")

    plt.show()
```

```
begin
1
EaseMyTrip.csv
```



Training models:

DB1B Dataset:
Normal view:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
g=glob.glob('../input/calfl1016combrts/sparce-all-rts-cal-fl-withfareclass.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
#route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/calfl1016combrts/sparce-all-rts-cal-fl-withfareclass.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [6]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
def gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    params = {
              #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson',#['least_squares', 'least_absolute_deviation', 'poisson'],
            'verbose':2}
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [11]:
```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    print(df.columns)
    y = df['MktFare'].values
    X= df.drop(['MktFare','Unnamed: 0'],axis=1).values
    y=y.astype(float)

    # Train the model
    for k in [0.3,0.2,0.1,0.05]:

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
        #RF
        print("RF")
        model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #DT
        print("DT")
        model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #BR
        print("BR")
        model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #GBR
        print("BR")
        model4=gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #positive weight DT
        print("c1")
        c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight DT
        print("c2")
        c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #positive weight GBR
        print("c3")
        c3(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight GBR
        print("c4")
        c4(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c5")
        #c5(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c6")
        #c6(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)
```

```
        #model saving
        filename = 'RFmodel'+str(k)+'.pkl'
        pickle.dump(model1, open(filename, 'wb'))

        filename = 'DTmodel'+str(k)+'.pkl'
        pickle.dump(model2, open(filename, 'wb'))

        filename = 'BRmodel'+str(k)+'.pkl'
        pickle.dump(model3, open(filename, 'wb'))

        filename = 'GBRmodel'+str(k)+'.pkl'
        pickle.dump(model4, open(filename, 'wb'))




print("end")
```

```
begin
1
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'RPCarrier_9E', 'RPCarrier_AA', 'RPCarrier_AS',
       'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL', 'RPCarrier_EV',
       'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA', 'RPCarrier_MQ',
       'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO', 'RPCarrier_QX',
       'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX', 'RPCarrier_WN',
       'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO', 'route_FLL-LAX',
       'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL', 'route_LAX-MCO',
       'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO', 'route_LAX-SJC',
       'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX', 'route_MCO-SAN',
       'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX', 'route_MIA-SFO',
       'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO', 'route_SAN-SFO',
       'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA', 'route_SFO-BUR',
       'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO', 'route_SFO-MIA',
       'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA', 'route_SFO-TPA',
       'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA', 'route_SMF-LAX',
       'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO', 'route_SNA-SJC',
       'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO', 'FareClass_Others',
       'FareClass_Restricted_Business_Class'
```

In [12]:
```
df=pd.DataFrame([pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_value),pd.Series(adj_r2_value),pd.Series(RMSE_value),
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[12]:

	0	1	2	3	4	5	6	7	
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomFores
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	
Test size	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	
r2	0.358345	0.340997	0.351828	0.37815	0.358345	0.359115	0.380679	0.380734	
Adjusted r2	0.358142	0.340788	0.351623	0.377953	0.358142	0.358912	0.380483	0.380537	

5 rows × 32 columns

In [13]:
```
df1=df.T
df1.head()
```

Out[13]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.358345	0.358142	194.906848	0	0	0	0	559.476681
1	DecisionTreeRegressor()	0.7	0.3	0.340997	0.340788	197.524137	0	0	0	0	8.926332
2	BaggingRegressor()	0.7	0.3	0.351828	0.351623	195.894151	0	0	0	0	62.403033
3	GradientBoostingRegressor()	0.7	0.3	0.37815	0.377953	191.875352	0	0	0	0	19.942588
4	weight 1()	0.7	0.3	0.358345	0.358142	194.906848	100	0	0	0	0.0

In [14]:
```
df1.to_csv("Table.csv")
```

In [15]:
```
df1
```

Out[15]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.358345	0.358142	194.906848	0	0	0	0	559.476681
1	DecisionTreeRegressor()	0.7	0.3	0.340997	0.340788	197.524137	0	0	0	0	8.926332
2	BaggingRegressor()	0.7	0.3	0.351828	0.351623	195.894151	0	0	0	0	62.403033
3	GradientBoostingRegressor()	0.7	0.3	0.37815	0.377953	191.875352	0	0	0	0	19.942588
4	weight 1()	0.7	0.3	0.358345	0.358142	194.906848	100	0	0	0	0.0
5	weight 2()	0.7	0.3	0.359115	0.358912	194.789966	135	-25	-10	0	0.0
6	weight 3()	0.7	0.3	0.380679	0.380483	191.484833	25	0	0	75	0.0
7	weight 4()	0.7	0.3	0.380734	0.380537	191.476344	35	0	-10	75	0.0
8	RandomForestRegressor()	0.8	0.2	0.350036	0.349727	198.391866	0	0	0	0	661.176472
9	DecisionTreeRegressor()	0.8	0.2	0.336134	0.335819	200.502359	0	0	0	0	10.71984
10	BaggingRegressor()	0.8	0.2	0.350193	0.349884	198.368023	0	0	0	0	72.555686
11	GradientBoostingRegressor()	0.8	0.2	0.371076	0.370777	195.154382	0	0	0	0	22.203737
12	weight 1()	0.8	0.2	0.351381	0.351072	198.186606	48	0	52	0	0.0
13	weight 2()	0.8	0.2	0.351979	0.351671	198.095179	70	-25	55	0	0.0
14	weight 3()	0.8	0.2	0.373476	0.373178	194.781699	3	0	22	75	0.0
15	weight 4()	0.8	0.2	0.373476	0.373178	194.781699	3	0	22	75	0.0
16	RandomForestRegressor()	0.9	0.1	0.337559	0.336929	200.558859	0	0	0	0	756.893783
17	DecisionTreeRegressor()	0.9	0.1	0.322531	0.321887	202.821053	0	0	0	0	12.920727
18	BaggingRegressor()	0.9	0.1	0.333662	0.333028	201.147984	0	0	0	0	86.079849
19	GradientBoostingRegressor()	0.9	0.1	0.355763	0.35515	197.784054	0	0	0	0	24.262442
20	weight 1()	0.9	0.1	0.337572	0.336942	200.55699	95	0	5	0	0.0
21	weight 2()	0.9	0.1	0.338295	0.337666	200.447464	116	-27	11	0	0.0
22	weight 3()	0.9	0.1	0.358415	0.357805	197.376525	20	0	6	74	0.0
23	weight 4()	0.9	0.1	0.358415	0.357805	197.376525	20	0	6	74	0.0
24	RandomForestRegressor()	0.95	0.05	0.335229	0.333963	195.912954	0	0	0	0	796.651448
25	DecisionTreeRegressor()	0.95	0.05	0.320504	0.31921	198.070804	0	0	0	0	12.797468
26	BaggingRegressor()	0.95	0.05	0.334605	0.333338	196.004826	0	0	0	0	95.799308
27	GradientBoostingRegressor()	0.95	0.05	0.356119	0.354893	192.810052	0	0	0	0	24.873753
28	weight 1()	0.95	0.05	0.335893	0.334628	195.815056	58	0	42	0	0.0
29	weight 2()	0.95	0.05	0.337868	0.336607	195.523623	110	-52	42	0	0.0
30	weight 3()	0.95	0.05	0.358076	0.356853	192.51696	0	0	22	78	0.0
31	weight 4()	0.95	0.05	0.358095	0.356872	192.514092	-7	0	29	78	0.0

.

Sub-route view:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
g=glob.glob('../input/sparce-routes-wfc/*.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/sparce-routes-wfc/SAN-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/SNA-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/OAK-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-PSP-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SNA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/MIA-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/PSP-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-OAK-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-MIA-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/BUR-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-SMF-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-BUR-verWFC.csv',
 '../input/sparce-routes-wfc/MIA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SMF-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-SNA-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-FLL-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-FLL-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-SNA-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-MIA-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SMF-verWFC.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [6]:
```
#
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor

def gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,rn):
    params = {
              #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson'#['least_squares', 'least_absolute_deviation', 'poisson'],
            }
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst,rn):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst,rn):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst,rn):
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2  r2_score(y_test, predictions)

    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst,rn):
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k

    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [11]:
```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    print(df.columns)
    rtn=df.route[1]
    print(rtn)
    y = df['MktFare'].values
    X= df.drop(['MktFare','route','Unnamed: 0'],axis=1).values
    y=y.astype(float)
    print(df.columns)
    # Train the model
    for k in [0.3,0.2,0.1,0.05]:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
        #RF
        print("RF")
        model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,rtn)
        #DT
        print("DT")
        model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,rtn)
        #BR
        print("BR")
        model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,rtn)
        #GBR
        print("BR")
        model4=gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,rtn)
        #positive weight DT
        print("c1")
        c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,rtn)

        #negative weight DT
        print("c2")
        c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,rtn)

        #positive weight GBR
        print("c3")
        c3(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,rtn)

        #negative weight GBR
        print("c4")
        c4(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,rtn)

        print("c5")
        #c5(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c6")
        #c6(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)
```

```
        #model saving
        filename = 'RFmodel'+rtn+str(k)+'.pkl'
        pickle.dump(model1, open(filename, 'wb'))

        filename = 'DTmodel'+rtn+str(k)+'.pkl'
        pickle.dump(model2, open(filename, 'wb'))

        filename = 'BRmodel'+rtn+str(k)+'.pkl'
        pickle.dump(model3, open(filename, 'wb'))

        filename = 'GBRmodel'+rtn+str(k)+'.pkl'
        pickle.dump(model4, open(filename, 'wb'))




print("end")
```

```
        'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
        'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
        'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
        'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
        'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
        'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
        'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
        'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
        'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
        'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
        'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO',
        'FareClass_Others', 'FareClass_Restricted Business Class',
        'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
        'FareClass_Unrestricted Business Class',
        'FareClass_Unrestricted Coach Class',
        'FareClass_Unrestricted First Class'],
       dtype='object')
SAN-SJC
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
```

In [12]:
```
df=pd.DataFrame([pd.Series(route_name),pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_value),pd.Series(adj_r2_value),pd.S
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Route Name","Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[12]:

	0	1	2	3	4	5	6	7	
Route Name	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	S
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomForestReg
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	
Test size	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	
r2	0.116189	0.087036	0.115022	0.154921	0.117897	0.117932	0.154921	0.156235	0

5 rows × 1440 columns

In [13]:
```
df1=df.T
df1.head()
```

Out[13]:

	Route Name	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	SAN-SJC	RandomForestRegressor()	0.7	0.3	0.116189	0.098881	62.75949	0	0	0	0	1.181429
1	SAN-SJC	DecisionTreeRegressor()	0.7	0.3	0.087036	0.069157	63.786174	0	0	0	0	0.01624
2	SAN-SJC	BaggingRegressor()	0.7	0.3	0.115022	0.097691	62.800921	0	0	0	0	0.156823
3	SAN-SJC	GradientBoostingRegressor()	0.7	0.3	0.154921	0.138372	61.368899	0	0	0	0	0.788022
4	SAN-SJC	weight 1()	0.7	0.3	0.117897	0.100622	62.698838	56	0	44	0	0.0

In [14]:
```
df1.to_csv("Table.csv")
```

In [15]:
```
df1
```

Out[15]:

	Route Name	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	SAN-SJC	RandomForestRegressor()	0.7	0.3	0.116189	0.098881	62.75949	0	0	0	0	1.181429
1	SAN-SJC	DecisionTreeRegressor()	0.7	0.3	0.087036	0.069157	63.786174	0	0	0	0	0.01624
2	SAN-SJC	BaggingRegressor()	0.7	0.3	0.115022	0.097691	62.800921	0	0	0	0	0.156823
3	SAN-SJC	GradientBoostingRegressor()	0.7	0.3	0.154921	0.138372	61.368899	0	0	0	0	0.788022
4	SAN-SJC	weight 1()	0.7	0.3	0.117897	0.100622	62.698838	56	0	44	0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1435	LAX-SMF	GradientBoostingRegressor()	0.95	0.05	0.102925	-0.008504	92.326655	0	0	0	0	0.291925
1436	LAX-SMF	weight 1()	0.95	0.05	0.074036	-0.040982	93.801516	100	0	0	0	0.0
1437	LAX-SMF	weight 2()	0.95	0.05	0.08835	-0.024889	93.073635	213	-100	-13	0	0.0
1438	LAX-SMF	weight 3()	0.95	0.05	0.10428	-0.006981	92.25691	17	0	0	83	0.0
1439	LAX-SMF	weight 4()	0.95	0.05	0.104669	-0.006544	92.236873	40	0	-23	83	0.0

1440 rows × 12 columns

Normal view divided into years and quarters:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
import time
g=glob.glob('../input/calfl1016combrts/sparce-all-rts-cal-fl-withfareclass.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
#route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
yrs=[]
qtrs=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/calfl1016combrts/sparce-all-rts-cal-fl-withfareclass.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print("MSE:", mse)
        rmse = np.sqrt(mse)
        print("RMSE:", rmse)
        r2 = r2_score(y_test, predictions)
        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print("MSE:", mse)
        rmse = np.sqrt(mse)
        print("RMSE:", rmse)
        r2 = r2_score(y_test, predictions)
        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [6]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
def gbr(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt):
    params = { #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson',#['least_squares', 'least_absolute_deviation', 'poisson'],
            'verbose':2}
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print("MSE:", mse)
        rmse = np.sqrt(mse)
        print("RMSE:", rmse)
        r2 = r2_score(y_test, predictions)
        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [11]:

```
def modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,yr,qt):
    #RF
    print("RF")
    model1=rf(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #DT
    print("DT")
    model2=dt(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #BR
    print("BR")
    model3=br(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #GBR
    print("GBR")
    model4=gbr(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt)
    #positive weight
    print("c1")
    c1(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_

    #negative weight
    print("c2")
    c2(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    print("c3")
    c3(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    print("c4")
    c4(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_

    #model saving
    filename = 'RFmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model1, open(filename, 'wb'))

    filename = 'DTmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model2, open(filename, 'wb'))

    filename = 'BRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model3, open(filename, 'wb'))

    filename = 'GBRmodel'+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model4, open(filename, 'wb'))
    return
```

In [12]:

```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    for k in [0.3,0.2,0.1,0.05]:
        df=pd.read_csv(file)
        print(df.columns)
        for i in [2018,2019,2020]:
            df1=df[df["Year"]==i]
            y = df1['MktFare'].values
            X= df1.drop(['MktFare','Unnamed: 0'],axis=1).values
            y=y.astype(float)
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
            modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,i,0)
        for i in [1,2,3,4]:
            df1=df[df["Quarter"]==i]
            y = df1['MktFare'].values
            X= df1.drop(['MktFare','Unnamed: 0'],axis=1).values
            y=y.astype(float)
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
            modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,0,i)
        for i in [2018,2019,2020]:
            for j in [1,2,3,4]:
                df1=df[np.logical_and(df["Year"]==i,df["Quarter"]==j)]
                y = df1['MktFare'].values
                X= df1.drop(['MktFare','Unnamed: 0'],axis=1).values
                y=y.astype(float)
                X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
                modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,

print("end")
```

```
begin
1
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'RPCarrier_9E', 'RPCarrier_AA', 'RPCarrier_AS',
       'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL', 'RPCarrier_EV',
       'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA', 'RPCarrier_MQ',
       'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO', 'RPCarrier_QX',
       'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX', 'RPCarrier_WN',
       'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO', 'route_FLL-LAX',
       'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL', 'route_LAX-MCO',
       'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO', 'route_LAX-SJC',
       'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX', 'route_MCO-SAN',
       'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX', 'route_MIA-SFO',
       'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO', 'route_SAN-SFO',
       'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA', 'route_SFO-BUR',
       'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO', 'route_SFO-MIA',
       'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA', 'route_SFO-TPA',
       'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA', 'route_SMF-LAX',
       'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO', 'route_SNA-SJC',
       'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO', 'FareClass_Others'
```

In [13]:
```
df=pd.DataFrame([pd.Series(yrs),pd.Series(qtrs),pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_value),pd.Series(adj_r2_va
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Year","Quarter","Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8
Year	2018	2018	2018	2018	2018	2018	2018	2018	2019
Quarter	0	0	0	0	0	0	0	0	0
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomForestRegressor()
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
Test size	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3

5 rows × 144 columns

In [14]:
```
df1=df.T
df1.head()
```

Out[14]:

	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	2018	0	RandomForestRegressor()	0.7	0.3	0.325286	0.324846	196.754955	0	0	0	0	261.951552
1	2018	0	DecisionTreeRegressor()	0.7	0.3	0.302979	0.302525	199.981007	0	0	0	0	4.59329
2	2018	0	BaggingRegressor()	0.7	0.3	0.320877	0.320435	197.396748	0	0	0	0	31.666346
3	2018	0	GradientBoostingRegressor()	0.7	0.3	0.347065	0.34664	193.553377	0	0	0	0	10.350508
4	2018	0	weight 1()	0.7	0.3	0.32535	0.32491	196.745621	89	0	11	0	0.0

In [15]:
```
df1.to_csv("Table.csv")
```

In [16]:
```
df1
```

Out[16]:

	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	2018	0	RandomForestRegressor()	0.7	0.3	0.325286	0.324846	196.754955	0	0	0	0	261.951552
1	2018	0	DecisionTreeRegressor()	0.7	0.3	0.302979	0.302525	199.981007	0	0	0	0	4.59329
2	2018	0	BaggingRegressor()	0.7	0.3	0.320877	0.320435	197.396748	0	0	0	0	31.666346
3	2018	0	GradientBoostingRegressor()	0.7	0.3	0.347065	0.34664	193.553377	0	0	0	0	10.350508
4	2018	0	weight 1()	0.7	0.3	0.32535	0.32491	196.745621	89	0	11	0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
139	2020	3	GradientBoostingRegressor()	0.7	0.3	0.427861	0.422714	123.1766	0	0	0	0	0.76778
140	2020	3	weight 1()	0.7	0.3	0.384467	0.37893	127.762365	100	0	0	0	0.0
141	2020	3	weight 2()	0.7	0.3	0.394558	0.389111	126.71081	217	-56	-61	0	0.0
142	2020	3	weight 3()	0.7	0.3	0.429171	0.424036	123.035453	15	0	0	85	0.0
143	2020	3	weight 4()	0.7	0.3	0.43203	0.426921	122.72695	67	0	-54	87	0.0

144 rows × 13 columns

Sub-route view divided into years and quarters:

In [1]:
```
# Version 8: table weight rename
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
import time
g=glob.glob('../input/sparce-routes-wfc/*.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
yrs=[]
qtrs=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/sparce-routes-wfc/SAN-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/SNA-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/OAK-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-PSP-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SNA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/MIA-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/PSP-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-OAK-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-MIA-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/BUR-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-SMF-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-BUR-verWFC.csv',
 '../input/sparce-routes-wfc/MIA-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SMF-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-SNA-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/FLL-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-FLL-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SJC-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-TPA-verWFC.csv',
 '../input/sparce-routes-wfc/SAN-SFO-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-FLL-verWFC.csv',
 '../input/sparce-routes-wfc/SJC-LAX-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-SNA-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-MCO-verWFC.csv',
 '../input/sparce-routes-wfc/SMF-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-MIA-verWFC.csv',
 '../input/sparce-routes-wfc/TPA-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/MCO-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/SFO-SAN-verWFC.csv',
 '../input/sparce-routes-wfc/LAX-SMF-verWFC.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt,rn):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt,rn):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt,rn):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [6]:
```
#
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor

def gbr(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k,yr,qt,rn):
    params = {
              #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson'#['least_squares', 'least_absolute_deviation', 'poisson'],
            }
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        route_name.append(rn)
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
        yrs.append(yr)
        qtrs.append(qt)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k

    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_K_.append(mk)
    max_l_.append(0)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_K_.append(mk)
    max_l_.append(mj)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    #RF
    #model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions1 = model1.predict(X_test)
    #DT
    #model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions2 = model2.predict(X_test)
    #BR
    #model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,1)
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    route_name.append(rn)
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    yrs.append(yr)
    qtrs.append(qt)
    return
```

In [11]:
```
def modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,yr,qt,rn):
    #RF
    print("RF")
    model1=rf(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt,r
    #DT
    print("DT")
    model2=dt(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt,r
    #BR
    print("BR")
    model3=br(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt,r
    #GBR
    print("GBR")
    model4=gbr(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k,yr,qt,
    #positive weight
    print("c1")
    c1(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_

    #negative weight
    print("c2")
    c2(model1,model2,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    print("c3")
    c3(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_
    print("c4")
    c4(model1,model4,model3,X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_

    #model saving
    filename = 'RFmodel-'+rn+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model1, open(filename, 'wb'))

    filename = 'DTmodel-'+rn+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model2, open(filename, 'wb'))

    filename = 'BRmodel-'+rn+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model3, open(filename, 'wb'))

    filename = 'GBRmodel-'+rn+str(k)+'y'+str(yr)+'q'+str(qt)+'.pkl'
    pickle.dump(model4, open(filename, 'wb'))
    return
```

In [12]:
```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    for k in [0.3,0.2,0.1]:
        df=pd.read_csv(file)
        print(df.columns)
        rtn=df.route[1]
        print(rtn)
        for i in [2018,2019,2020]:
            df1=df[df["Year"]==i]
            y = df1['MktFare'].values
            X= df1.drop(['MktFare','route','Unnamed: 0'],axis=1).values
            y=y.astype(float)
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
            modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,i,0,
        for i in [1,2,3,4]:
            df1=df[df["Quarter"]==i]
            y = df1['MktFare'].values
            X= df1.drop(['MktFare','route','Unnamed: 0'],axis=1).values
            y=y.astype(float)
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
            modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,0,i,
        for i in [2018,2019,2020]:
            for j in [1,2,3,4]:
                if((i==2020) and (j==4)):
                    continue
                df1=df[np.logical_and(df["Year"]==i,df["Quarter"]==j)]
                y = df1['MktFare'].values
                X= df1.drop(['MktFare','route','Unnamed: 0'],axis=1).values
                y=y.astype(float)
                X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
                modelT(X_train, X_test, y_train, y_test,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k,
print("end")
```

```
begin
1
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA',
       'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL',
       'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA',
       'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO',
       'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX',
       'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO',
       'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
       'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
       'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
       'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
       'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
       'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
       'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
       'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
       'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
       'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
       'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO',
       'FareClass_Others', 'FareClass_Restricted_Business_Class',
```

In [13]:
```
df=pd.DataFrame([pd.Series(route_name),pd.Series(yrs),pd.Series(qtrs),pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_val
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Route Name","Year","Quarter","Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8
Route Name	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC	SAN-SJC
Year	2018	2018	2018	2018	2018	2018	2018	2018	2019
Quarter	0	0	0	0	0	0	0	0	0
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomForestRegressor()
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7

5 rows × 4160 columns

In [14]:
```
df1=df.T
df1.head()
```

Out[14]:

	Route Name	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	SAN-SJC	2018	0	RandomForestRegressor()	0.7	0.3	0.171659	0.135881	58.340432	0	0	0	0	0.449817
1	SAN-SJC	2018	0	DecisionTreeRegressor()	0.7	0.3	0.131166	0.093639	59.749392	0	0	0	0	0.006109
2	SAN-SJC	2018	0	BaggingRegressor()	0.7	0.3	0.175634	0.140027	58.200291	0	0	0	0	0.063537
3	SAN-SJC	2018	0	GradientBoostingRegressor()	0.7	0.3	0.182612	0.147306	57.95345	0	0	0	0	0.580065
4	SAN-SJC	2018	0	weight 1()	0.7	0.3	0.176399	0.140825	58.173299	29	0	71	0	0.0

In [15]:
```
df1.to_csv("Table.csv")
```

In [16]:
```
df1
```

Out[16]:

	Route Name	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	SAN-SJC	2018	0	RandomForestRegressor()	0.7	0.3	0.171659	0.135881	58.340432	0	0	0	0	0.449817
1	SAN-SJC	2018	0	DecisionTreeRegressor()	0.7	0.3	0.131166	0.093639	59.749392	0	0	0	0	0.006109
2	SAN-SJC	2018	0	BaggingRegressor()	0.7	0.3	0.175634	0.140027	58.200291	0	0	0	0	0.063537
3	SAN-SJC	2018	0	GradientBoostingRegressor()	0.7	0.3	0.182612	0.147306	57.95345	0	0	0	0	0.580065
4	SAN-SJC	2018	0	weight 1()	0.7	0.3	0.176399	0.140825	58.173299	29	0	71	0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4155	TPA-LAX	2020	1	GradientBoostingRegressor()	0.8	0.2	0.258062	0.055948	190.863986	0	0	0	0	1.102082
4156	TPA-LAX	2020	1	weight 1()	0.8	0.2	0.253838	0.050574	191.4065	48	33	19	0	0.0
4157	TPA-LAX	2020	1	weight 2()	0.8	0.2	0.253838	0.050574	191.4065	48	33	19	0	0.0
4158	TPA-LAX	2020	1	weight 3()	0.8	0.2	0.259881	0.058262	190.629886	34	0	0	66	0.0
4159	TPA-LAX	2020	1	weight 4()	0.8	0.2	0.260537	0.059097	190.545442	69	0	-43	74	0.0

4160 rows × 14 columns

Machine-Hack Dataset:

In [1]:
```
# Version 8: table weight rename
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
g=glob.glob('../input/dataset-2-for-airfare/*.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
#route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/dataset-2-for-airfare/train_data.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [6]:
```
#
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
def gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    params = {
              #'n_trees_per_iteration_':2,
          #'max_depth': 4,
          #'min_samples_split': 5,
          'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
          'loss': 'poisson',#['least_squares', 'least_absolute_deviation', 'poisson'],
            'verbose':2}
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [11]:
```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    print(df.columns)
    y = df['Price'].values
    X= df.drop(['Price','Unnamed: 0'],axis=1).values
    y=y.astype(float)
    # Train the model
    for k in [0.3,0.2,0.1,0.05]:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
        #RF
        print("RF")
        model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #DT
        print("DT")
        model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #BR
        print("BR")
        model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #GBR
        print("GBR")
        model4=gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #positive weight DT
        print("c1")
        c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight DT
        print("c2")
        c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #positive weight GBR
        print("c3")
        c3(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight GBR
        print("c4")
        c4(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c5")
        #c5(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c6")
        #c6(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)
```

```
        #model saving
        filename = 'RFmodel'+str(k)+'Bench.pkl'
        pickle.dump(model1, open(filename, 'wb'))

        filename = 'DTmodel'+str(k)+'Bench.pkl'
        pickle.dump(model2, open(filename, 'wb'))

        filename = 'BRmodel'+str(k)+'Bench.pkl'
        pickle.dump(model3, open(filename, 'wb'))

        filename = 'GBRmodel'+str(k)+'Bench.pkl'
        pickle.dump(model4, open(filename, 'wb'))




print("end")
```

```
begin
1
Index(['Unnamed: 0', 'Total_Stops', 'Price', 'Journey_day', 'Journey_month',
       'Departure_hour', 'Departure_Minute', 'Arrival_Hour', 'Arrival_Minute',
       'Duration_Hours', 'Duration_Min', 'Airline_Air India', 'Airline_GoAir',
       'Airline_IndiGo', 'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Cochin', 'Delhi', 'Hyderabad', 'Kolkata', 'New Delhi'],
      dtype='object')
RF
MSE: 3958103.128738998
RMSE: 1989.4982102879605
R2: 0.8166693207422192
Adjusted R2: 0.8149948043017544
DT
MSE: 6815788.454702201
RMSE: 2610.706504895217
R2: 0.684307587135562
```

In [12]:
```
df=pd.DataFrame([pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_value),pd.Series(adj_r2_value),pd.Series(RMSE_value),
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[12]:

	0	1	2	3	4	5	6	7	
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomFores
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	
Test size	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	
r2	0.816669	0.684308	0.783886	0.790258	0.816669	0.822237	0.82989	0.831558	
Adjusted r2	0.814995	0.681424	0.781912	0.788342	0.814995	0.820614	0.828336	0.83002	

5 rows × 32 columns

In [13]:
```
df1=df.T
df1.head()
```

Out[13]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.816669	0.814995	1989.49821	0	0	0	0	2.582972
1	DecisionTreeRegressor()	0.7	0.3	0.684308	0.681424	2610.706505	0	0	0	0	0.036717
2	BaggingRegressor()	0.7	0.3	0.783886	0.781912	2160.067146	0	0	0	0	0.275899
3	GradientBoostingRegressor()	0.7	0.3	0.790258	0.788342	2127.985559	0	0	0	0	0.881431
4	weight 1()	0.7	0.3	0.816669	0.814995	1989.49821	100	0	0	0	0.0

In [14]:
```
df1.to_csv("Table.csv")
```

In [15]:
```
df1
```

Out[15]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.816669	0.814995	1989.49821	0	0	0	0	2.582972
1	DecisionTreeRegressor()	0.7	0.3	0.684308	0.681424	2610.706505	0	0	0	0	0.036717
2	BaggingRegressor()	0.7	0.3	0.783886	0.781912	2160.067146	0	0	0	0	0.275899
3	GradientBoostingRegressor()	0.7	0.3	0.790258	0.788342	2127.985559	0	0	0	0	0.881431
4	weight 1()	0.7	0.3	0.816669	0.814995	1989.49821	100	0	0	0	0.0
5	weight 2()	0.7	0.3	0.822237	0.820614	1959.05349	155	-14	-41	0	0.0
6	weight 3()	0.7	0.3	0.82989	0.828336	1916.421586	63	0	0	37	0.0
7	weight 4()	0.7	0.3	0.831558	0.83002	1907.000295	98	34	-32	0	0.0
8	RandomForestRegressor()	0.8	0.2	0.809268	0.806643	2115.460628	0	0	0	0	2.926064
9	DecisionTreeRegressor()	0.8	0.2	0.703309	0.699225	2638.431958	0	0	0	0	0.041016
10	BaggingRegressor()	0.8	0.2	0.7995	0.79674	2168.957805	0	0	0	0	0.298089
11	GradientBoostingRegressor()	0.8	0.2	0.772295	0.769161	2311.423636	0	0	0	0	0.915973
12	weight 1()	0.8	0.2	0.809811	0.807193	2112.448105	81	0	19	0	0.0
13	weight 2()	0.8	0.2	0.813344	0.810775	2092.734842	99	-22	23	0	0.0
14	weight 3()	0.8	0.2	0.821138	0.818677	2048.575709	32	0	34	34	0.0
15	weight 4()	0.8	0.2	0.821138	0.818677	2048.575709	32	34	34	0	0.0
16	RandomForestRegressor()	0.9	0.1	0.803841	0.798366	2207.976945	0	0	0	0	3.278046
17	DecisionTreeRegressor()	0.9	0.1	0.697117	0.688663	2743.645478	0	0	0	0	0.046289
18	BaggingRegressor()	0.9	0.1	0.785099	0.779101	2311.053051	0	0	0	0	0.338494
19	GradientBoostingRegressor()	0.9	0.1	0.804029	0.798559	2206.921769	0	0	0	0	0.924786
20	weight 1()	0.9	0.1	0.803841	0.798366	2207.976945	100	0	0	0	0.0
21	weight 2()	0.9	0.1	0.813856	0.808661	2150.873654	168	-38	-30	0	0.0
22	weight 3()	0.9	0.1	0.814606	0.809432	2146.536895	50	0	0	50	0.0
23	weight 4()	0.9	0.1	0.81727	0.81217	2131.060663	98	53	-51	0	0.0
24	RandomForestRegressor()	0.95	0.05	0.784054	0.771653	2489.715883	0	0	0	0	3.326445
25	DecisionTreeRegressor()	0.95	0.05	0.681867	0.663598	3021.908978	0	0	0	0	0.048383
26	BaggingRegressor()	0.95	0.05	0.80492	0.793718	2366.371781	0	0	0	0	0.350303
27	GradientBoostingRegressor()	0.95	0.05	0.75836	0.744484	2633.670346	0	0	0	0	1.828986
28	weight 1()	0.95	0.05	0.80492	0.793718	2366.371781	0	0	100	0	0.0
29	weight 2()	0.95	0.05	0.82447	0.81439	2244.668112	-11	-57	168	0	0.0
30	weight 3()	0.95	0.05	0.805225	0.79404	2364.522753	0	0	93	7	0.0
31	weight 4()	0.95	0.05	0.81079	0.799925	2330.497746	-86	22	164	0	0.0

Ease-My-Trip Dataset:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import glob
g=glob.glob('../input/easemytrip-ff-sample-data-processed/EaseMyTrip.csv')
r2_value=[]
adj_r2_value=[]
RMSE_value=[]
#route_name=[]
Tr=[]
te=[]
mn=[]
max_i_=[]
max_j_=[]
max_k_=[]
max_l_=[]
times=[]
```

In [2]:
```
g
```

Out[2]:
```
['../input/easemytrip-ff-sample-data-processed/EaseMyTrip.csv']
```

In [3]:
```
#descision tree
from sklearn.tree import DecisionTreeRegressor
def dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = DecisionTreeRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("DecisionTreeRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [4]:
```
#Random Forest
from sklearn.ensemble import RandomForestRegressor
def rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = RandomForestRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("RandomForestRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [5]:
```
#Bagging Regressor
from sklearn.ensemble import BaggingRegressor
def br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    start = time.time()
    model = BaggingRegressor().fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("BaggingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [6]:
```
#
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor

def gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,o,k):
    params = {
            #'n_trees_per_iteration_':2,
         #'max_depth': 4,
         #'min_samples_split': 5,
         'learning_rate': 0.3,#[0.1,0.15,0.2,0.175,0.225],
         'loss': 'poisson',#['least_squares', 'least_absolute_deviation', 'poisson'],
          'verbose':2}
    start = time.time()
    model = HistGradientBoostingRegressor(**params).fit(X_train, y_train)
    stop = time.time()
    if o==0:
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)

        print("MSE:", mse)

        rmse = np.sqrt(mse)

        print("RMSE:", rmse)

        r2 = r2_score(y_test, predictions)

        print("R2:", r2)
        ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
        print("Adjusted R2:",ar2)
        Tr.append(1-k)#train
        te.append(k)#test
        r2_value.append(r2)
        adj_r2_value.append(ar2)
        RMSE_value.append(rmse)
        #route_name.append(rt[0])
        mn.append("GradientBoostingRegressor()")
        max_i_.append(0)
        max_j_.append(0)
        max_k_.append(0)
        max_l_.append(0)
        times.append(stop-start)
    return model
```

In [7]:
```
#positive weight DT
def c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 1()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [8]:
```
#negative weight DT
def c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 2()")
    max_i_.append(mi)
    max_j_.append(mj)
    max_k_.append(mk)
    max_l_.append(0)
    times.append(0)
    return
```

In [9]:
```
#positive weight GBR
def c3(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(0,101):
        for j in range(0,101):
            for k in range(0,101):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 3()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [10]:
```
#negative weight GBR
def c4(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,tst):
    #RF
    predictions1 = model1.predict(X_test)
    #DT
    predictions2 = model2.predict(X_test)
    #BR
    predictions3 = model3.predict(X_test)
    max_r2=0
    mi=0
    mj=0
    mk=0
    for i in range(-100,301):
        for j in range(-100,301):
            for k in range(-100,301):
                if((i+j+k)==100):
                    #print(i," ",j," ",k)
                    pred=(i/100)*predictions1+(j/100)*predictions2+(k/100)*predictions3
                    r2 = r2_score(y_test, pred)
                    if(r2>max_r2):
                        max_r2=r2
                        mi=i
                        mj=j
                        mk=k
    predictions = (mi/100)*predictions1+(mj/100)*predictions2+(mk/100)*predictions3
    mse = mean_squared_error(y_test, predictions)
    print("MSE:", mse)
    rmse = np.sqrt(mse)
    print("RMSE:", rmse)
    r2 = r2_score(y_test, predictions)
    print("R2:", r2)
    ar2=1-(1-r2)*((len(X_test)-1)/(len(X_test)-X_test.shape[1]-1))
    print("Adjusted R2:",ar2)
    Tr.append(1-tst)#train
    te.append(tst)#test
    r2_value.append(r2)
    adj_r2_value.append(ar2)
    RMSE_value.append(rmse)
    #route_name.append(rt[0])
    mn.append("weight 4()")
    max_i_.append(mi)
    max_j_.append(0)
    max_k_.append(mk)
    max_l_.append(mj)
    times.append(0)
    return
```

In [11]:
```
import pickle
print("begin")
F=1
for file in g:
    print(F)
    F=F+1
    df=pd.read_csv(file)
    print(df.columns)
    y = df['Fare'].values
    X= df.drop(['Fare','Unnamed: 0'],axis=1).values
    y=y.astype(float)

    # Train the model
    for k in [0.3,0.2,0.1,0.05]:

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=k, random_state=0)
        #RF
        print("RF")
        model1=rf(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #DT
        print("DT")
        model2=dt(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)
        #BR
        print("BR")
        model3=br(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #GBR
        print("BR")
        model4=gbr(X_train, y_train,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,0,k)

        #positive weight DT
        print("c1")
        c1(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight DT
        print("c2")
        c2(model1,model2,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #positive weight GBR
        print("c3")
        c3(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #negative weight GBR
        print("c4")
        c4(model1,model4,model3,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c5")
        #c5(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        print("c6")
        #c6(model1,model2,model3,model4,Tr,te,r2_value,adj_r2_value,RMSE_value,mn,max_i_,max_j_,max_k_,max_l_,k)

        #model saving
        filename = 'RFmodel'+str(k)+'.pkl'
        pickle.dump(model1, open(filename, 'wb'))

        filename = 'DTmodel'+str(k)+'.pkl'
        pickle.dump(model2, open(filename, 'wb'))

        filename = 'BRmodel'+str(k)+'.pkl'
        pickle.dump(model3, open(filename, 'wb'))

        filename = 'GBRmodel'+str(k)+'.pkl'
        pickle.dump(model4, open(filename, 'wb'))



print("end")
```

```
begin
1
Index(['Unnamed: 0', 'Number Of Stops', 'Fare', 'Days Left', 'Departure_day',
       'Departure_month', 'Departure_year', 'Departure_hour',
       'Departure_minute', 'F1_Air Canada',
       ...
       'Layover1_Tiruchirapally', 'Layover1_Tirupati', 'Layover1_Tokyo',
       'Layover1_Udaipur', 'Layover1_VDY', 'Layover1_Vadodara',
       'Layover1_Varanasi', 'Layover1_Vijayawada', 'Layover1_Vishakhapatnam',
       'Layover1_Warsaw'],
      dtype='object', length=366)
RF
MSE: 45807131.11414554
RMSE: 6768.096565072453
R2: 0.8860247570420065
Adjusted R2: 0.8774477488888723
DT
MSE: 43318177.032963604
RMSE: 6581.654581711471
R2: 0.8922176606186811
Adjusted R2: 0.8841066886247179
```

In [12]:
```
df=pd.DataFrame([pd.Series(mn),pd.Series(Tr),pd.Series(te),pd.Series(r2_value),pd.Series(adj_r2_value),pd.Series(RMSE_value),
                 pd.Series(max_i_),pd.Series(max_j_),pd.Series(max_k_),pd.Series(max_l_),pd.Series(times)],
                 index = ["Model Name","Train size","Test size","r2","Adjusted r2","RMSE",
                          "RF weight","DT weight","BR weight value","GBR weight","Time"])
df.head()
```

Out[12]:

	0	1	2	3	4	5	6	7	
Model Name	RandomForestRegressor()	DecisionTreeRegressor()	BaggingRegressor()	GradientBoostingRegressor()	weight 1()	weight 2()	weight 3()	weight 4()	RandomFores
Train size	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	
Test size	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	
r2	0.886025	0.892218	0.877796	-139.188039	0.897912	0.897912	0.886025	0.886066	
Adjusted r2	0.877448	0.884107	0.8686	-149.737645	0.890229	0.890229	0.877448	0.877492	

5 rows × 32 columns

In [13]:
```
df1=df.T
df1.head()
```

Out[13]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.886025	0.877448	6768.096565	0	0	0	0	84.802701
1	DecisionTreeRegressor()	0.7	0.3	0.892218	0.884107	6581.654582	0	0	0	0	1.613518
2	BaggingRegressor()	0.7	0.3	0.877796	0.8686	7008.146996	0	0	0	0	4.898724
3	GradientBoostingRegressor()	0.7	0.3	-139.188039	-149.737645	237365.044706	0	0	0	0	0.604006
4	weight 1()	0.7	0.3	0.897912	0.890229	6405.449524	28	60	12	0	0.0

In [14]:
```
df1.to_csv("Table.csv")
```

In [15]:
```
df1
```

Out[15]:

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.886025	0.877448	6768.096565	0	0	0	0	84.802701
1	DecisionTreeRegressor()	0.7	0.3	0.892218	0.884107	6581.654582	0	0	0	0	1.613518
2	BaggingRegressor()	0.7	0.3	0.877796	0.8686	7008.146996	0	0	0	0	4.898724
3	GradientBoostingRegressor()	0.7	0.3	-139.188039	-149.737645	237365.044706	0	0	0	0	0.604006
4	weight 1()	0.7	0.3	0.897912	0.890229	6405.449524	28	60	12	0	0.0
5	weight 2()	0.7	0.3	0.897912	0.890229	6405.449524	28	60	12	0	0.0
6	weight 3()	0.7	0.3	0.886025	0.877448	6768.096565	100	0	0	0	0.0
7	weight 4()	0.7	0.3	0.886066	0.877492	6766.884799	108	0	-8	0	0.0
8	RandomForestRegressor()	0.8	0.2	0.862254	0.846096	8162.523116	0	0	0	0	106.325064
9	DecisionTreeRegressor()	0.8	0.2	0.866768	0.851139	8027.666747	0	0	0	0	1.978331
10	BaggingRegressor()	0.8	0.2	0.853239	0.836023	8425.413329	0	0	0	0	5.772634
11	GradientBoostingRegressor()	0.8	0.2	-163.945189	-183.294222	282458.993666	0	0	0	0	0.544914
12	weight 1()	0.8	0.2	0.876082	0.861546	7741.996094	7	60	33	0	0.0
13	weight 2()	0.8	0.2	0.876082	0.861546	7741.996094	7	60	33	0	0.0
14	weight 3()	0.8	0.2	0.862254	0.846096	8162.523116	100	0	0	0	0.0
15	weight 4()	0.8	0.2	0.862926	0.846846	8142.610642	136	0	-36	0	0.0
16	RandomForestRegressor()	0.9	0.1	0.932839	0.914981	5745.045158	0	0	0	0	127.107208
17	DecisionTreeRegressor()	0.9	0.1	0.912803	0.889618	6546.127918	0	0	0	0	2.325367
18	BaggingRegressor()	0.9	0.1	0.934902	0.917594	5656.095177	0	0	0	0	6.019999
19	GradientBoostingRegressor()	0.9	0.1	-511.844736	-648.203746	502026.509371	0	0	0	0	0.570025
20	weight 1()	0.9	0.1	0.934951	0.917655	5653.978624	9	3	88	0	0.0
21	weight 2()	0.9	0.1	0.934951	0.917655	5653.978624	9	3	88	0	0.0
22	weight 3()	0.9	0.1	0.934935	0.917636	5654.65367	11	0	89	0	0.0
23	weight 4()	0.9	0.1	0.934935	0.917636	5654.65367	11	0	89	0	0.0
24	RandomForestRegressor()	0.95	0.05	0.934646	0.887258	4332.024397	0	0	0	0	137.427253
25	DecisionTreeRegressor()	0.95	0.05	0.929297	0.87803	4505.822117	0	0	0	0	2.357306
26	BaggingRegressor()	0.95	0.05	0.912079	0.848328	5024.588259	0	0	0	0	6.460227
27	GradientBoostingRegressor()	0.95	0.05	-233.010429	-402.691298	259222.05488	0	0	0	0	0.593173
28	weight 1()	0.95	0.05	0.941133	0.898449	4111.400808	57	43	0	0	0.0
29	weight 2()	0.95	0.05	0.946555	0.907802	3917.495545	143	39	-82	0	0.0
30	weight 3()	0.95	0.05	0.934646	0.887258	4332.024397	100	0	0	0	0.0
31	weight 4()	0.95	0.05	0.941124	0.898433	4111.717424	189	0	-89	0	0.0

Forecasting using Prophet:

DB1B Dataset Normal View:

In [ ]:
```
# v1: normal view forecast
```

In [3]:
```
!pip install prophet
```

```
Collecting prophet
  Downloading prophet-1.0.1.tar.gz (65 kB)
     |████████████████████████████████| 65 kB 748 kB/s eta 0:00:011
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Collecting cmdstanpy==0.9.68
  Downloading cmdstanpy-0.9.68-py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 1.7 MB/s eta 0:00:01
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Collecting ujson
  Downloading ujson-4.0.2-cp37-cp37m-manylinux1_x86_64.whl (179 kB)
     |████████████████████████████████| 179 kB 3.6 MB/s eta 0:00:01
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Building wheels for collected packages: prophet
  Building wheel for prophet (setup.py) ... done
  Created wheel for prophet: filename=prophet-1.0.1-py3-none-any.whl size=6644013 sha256=b110c029935b182c58892578b0bd6852b5a32e
01e7682e6e190de3e48d7d09a8
  Stored in directory: /root/.cache/pip/wheels/4e/a0/1a/02c9ec9e3e9de6bdbb3d769d11992a6926889d71567d6b9b67
Successfully built prophet
Installing collected packages: ujson, cmdstanpy, prophet
  Attempting uninstall: cmdstanpy
    Found existing installation: cmdstanpy 0.9.5
    Uninstalling cmdstanpy-0.9.5:
      Successfully uninstalled cmdstanpy-0.9.5
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
e source of the following dependency conflicts.
fbprophet 0.7.1 requires cmdstanpy==0.9.5, but you have cmdstanpy 0.9.68 which is incompatible.
Successfully installed cmdstanpy-0.9.68 prophet-1.0.1 ujson-4.0.2
```

In [8]:
```
df1=df[["MktFare"]].copy()
df1.columns
```

Out[8]:
```
Index(['MktFare'], dtype='object')
```

In [9]:
```
df.loc[df['Quarter'] == 1, 'month'] = 2
df.loc[df['Quarter'] == 2, 'month'] = 5
df.loc[df['Quarter'] == 3, 'month'] = 8
df.loc[df['Quarter'] == 4, 'month'] = 11
```

In [10]:
```
df['day']=1
```

In [11]:
```
df1['ds']=pd.to_datetime(df[['Year','month','day']])
```

In [12]:
```
df1.columns=['y','ds']
```

In [13]:
```
df1.head()
```

Out[13]:

	y	ds
0	1338.27	2018-02-01
1	66.47	2018-02-01
2	5.54	2018-02-01
3	521.46	2018-02-01
4	1651.23	2018-02-01

In [14]:
```
m = p.Prophet()
m.fit(df1)
```

Out[14]:
```
<prophet.forecaster.Prophet at 0x7f1d84f29390>
```

In [15]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[15]:

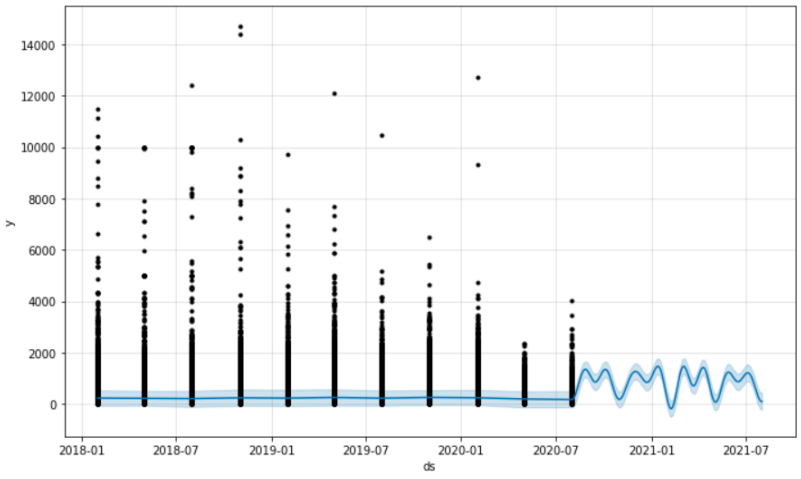
	ds
371	2021-07-28
372	2021-07-29
373	2021-07-30
374	2021-07-31
375	2021-08-01

In [16]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[16]:

	ds	yhat	yhat_lower	yhat_upper
371	2021-07-28	156.104509	-158.754008	491.452424
372	2021-07-29	128.281693	-166.154823	459.255533
373	2021-07-30	108.307655	-197.521098	469.326130
374	2021-07-31	96.565770	-231.168379	409.675853
375	2021-08-01	93.313938	-212.154563	428.087878

In [17]:
```
fig1 = m.plot(forecast)
```



DB1B Dataset Sub-Route View:

BUR-SFO Route:

In [18]:
```
# v2: subroute view forecast -0
```

In [19]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.0.1)
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
Requirement already satisfied: cmdstanpy==0.9.68 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.9.68)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
```

```
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: ujson in /opt/conda/lib/python3.7/site-packages (from cmdstanpy==0.9.68->prophet) (4.0.2)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
```

In [20]:
```
import prophet as p
import pandas as pd
```

In [21]:
```
import glob
g=glob.glob('../input/sparce-routes-wfc/*.csv')
```

In [22]:
```
#!pip install openpyxl
```

In [23]:
```
df=pd.read_csv(g[0])
#m = p.Prophet()
#m.fit(df)
df.columns
```

Out[23]:
```
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA',
       'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL',
       'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA',
       'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO',
       'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX',
       'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO',
       'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
       'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
       'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
       'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
       'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
       'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
       'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
       'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
       'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
       'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
       'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO',
       'FareClass_Others', 'FareClass_Restricted Business Class',
       'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
       'FareClass_Unrestricted Business Class',
       'FareClass_Unrestricted Coach Class',
       'FareClass_Unrestricted First Class'],
      dtype='object')
```

In [24]:
```
df1=df[["MktFare"]].copy()
df1.columns
```

Out[24]:
```
Index(['MktFare'], dtype='object')
```

In [25]:
```
df.loc[df['Quarter'] == 1, 'month'] = 2
df.loc[df['Quarter'] == 2, 'month'] = 5
df.loc[df['Quarter'] == 3, 'month'] = 8
df.loc[df['Quarter'] == 4, 'month'] = 11
```

In [26]:
```
df['day']=1
```

In [27]:
```
df1['ds']=pd.to_datetime(df[['Year','month','day']])
```

In [28]:
```
df1.columns=['y','ds']
```

In [29]:
```
df1.head()
```

Out[29]:

	y	ds
0	182.00	2018-02-01
1	277.00	2018-02-01
2	387.00	2018-02-01
3	147.50	2018-02-01
4	86.14	2018-02-01

In [30]:
```
m = p.Prophet()
m.fit(df1)
```

Out[30]:
```
<prophet.forecaster.Prophet at 0x7f1d5ba9df10>
```

In [31]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[31]:

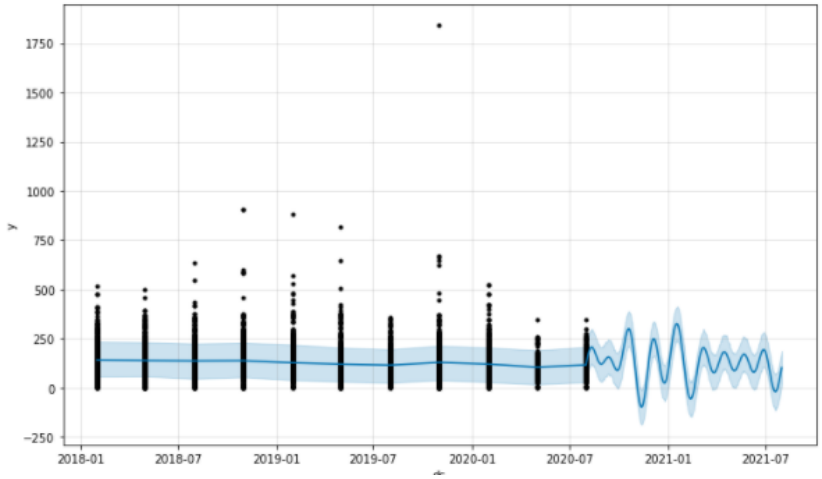
	ds
371	2021-07-28
372	2021-07-29
373	2021-07-30
374	2021-07-31
375	2021-08-01

In [32]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[32]:

	ds	yhat	yhat_lower	yhat_upper
371	2021-07-28	42.151641	-47.532547	130.902545
372	2021-07-29	56.571733	-31.407168	142.989617
373	2021-07-30	71.600896	-14.572802	158.791470
374	2021-07-31	86.918316	-4.285318	174.723653
375	2021-08-01	102.202539	21.847048	187.401235

In [33]:
```
fig1 = m.plot(forecast)
```



FLL-LAX Route:

In [34]:
```
# v3: subroute view forecast -1
```

In [35]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.0.1)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: cmdstanpy==0.9.68 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.9.68)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
```

```
Requirement already satisfied: ujson in /opt/conda/lib/python3.7/site-packages (from cmdstanpy==0.9.68->prophet) (4.0.2)
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
```

In [36]:
```
import prophet as p
import pandas as pd
```

In [37]:
```
import glob
g=glob.glob('../input/sparce-routes-wfc/*.csv')
```

In [38]:
```
#!pip install openpyxl
```

In [39]:
```
df=pd.read_csv(g[1])
#m = p.Prophet()
#m.fit(df)
df.columns
```

Out[39]:
```
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA',
       'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL',
       'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA',
       'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO',
       'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX',
       'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO',
       'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
       'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
       'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
       'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
       'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
       'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
       'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
       'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
       'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
       'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
       'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO',
       'FareClass_Others', 'FareClass_Restricted Business Class',
       'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
       'FareClass_Unrestricted Business Class',
       'FareClass_Unrestricted Coach Class',
       'FareClass_Unrestricted First Class'],
      dtype='object')
```

In [40]:
```
df1=df[["MktFare"]].copy()
df1.columns
```

Out[40]:
```
Index(['MktFare'], dtype='object')
```

In [41]:
```
df.loc[df['Quarter'] == 1, 'month'] = 2
df.loc[df['Quarter'] == 2, 'month'] = 5
df.loc[df['Quarter'] == 3, 'month'] = 8
df.loc[df['Quarter'] == 4, 'month'] = 11
```

In [42]:
```
df['day']=1
```

In [43]:
```
df1['ds']=pd.to_datetime(df[['Year','month','day']])
```

In [44]:
```
df1.columns=['y','ds']
```

In [45]:
```
df1.head()
```

Out[45]:

	y	ds
0	5.0	2018-02-01
1	125.0	2018-02-01
2	130.0	2018-02-01
3	140.0	2018-02-01
4	150.0	2018-02-01

In [46]:
```
m = p.Prophet()
m.fit(df1)
```

Out[46]:
```
<prophet.forecaster.Prophet at 0x7f1d5b95df50>
```

In [47]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[47]:

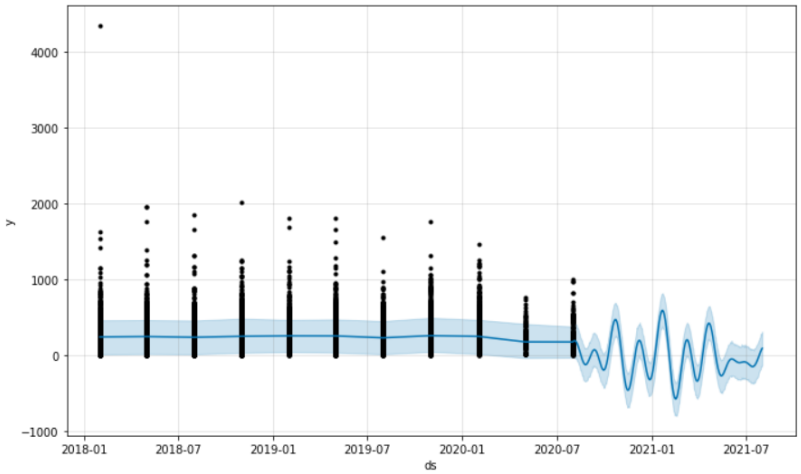
	ds
371	2021-07-28
372	2021-07-29
373	2021-07-30
374	2021-07-31
375	2021-08-01

In [48]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[48]:

	ds	yhat	yhat_lower	yhat_upper
371	2021-07-28	28.211630	-195.560499	233.562250
372	2021-07-29	45.561737	-183.754842	282.544791
373	2021-07-30	61.486813	-146.585990	270.506212
374	2021-07-31	75.573053	-149.671036	294.453685
375	2021-08-01	87.438682	-124.329951	307.265937

In [49]:
```
fig1 = m.plot(forecast)
```



LAX-FLL Route:

In [82]:
```
# v6: subroute view forecast -4
```

In [83]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.0.1)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
Requirement already satisfied: cmdstanpy==0.9.68 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.9.68)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Requirement already satisfied: ujson in /opt/conda/lib/python3.7/site-packages (from cmdstanpy==0.9.68->prophet) (4.0.2)
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
```

```
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
```

In [84]:
```
import prophet as p
import pandas as pd
```

In [85]:
```
import glob
g=glob.glob('../input/sparce-routes-wfc/*.csv')
```

In [86]:
```
#!pip install openpyxl
```

In [87]:
```
df=pd.read_csv(g[3])
#m = p.Prophet()
#m.fit(df)
df.columns
```

Out[87]:
```
Index(['Unnamed: 0', 'MktCoupons', 'Year', 'Quarter', 'BulkFare', 'Passengers',
       'MktFare', 'Pandemic', 'route', 'RPCarrier_9E', 'RPCarrier_AA',
       'RPCarrier_AS', 'RPCarrier_B6', 'RPCarrier_CP', 'RPCarrier_DL',
       'RPCarrier_EV', 'RPCarrier_F9', 'RPCarrier_G7', 'RPCarrier_HA',
       'RPCarrier_MQ', 'RPCarrier_NK', 'RPCarrier_OH', 'RPCarrier_OO',
       'RPCarrier_QX', 'RPCarrier_SY', 'RPCarrier_UA', 'RPCarrier_VX',
       'RPCarrier_WN', 'RPCarrier_YV', 'RPCarrier_YX', 'route_BUR-SFO',
       'route_FLL-LAX', 'route_FLL-SAN', 'route_FLL-SFO', 'route_LAX-FLL',
       'route_LAX-MCO', 'route_LAX-MIA', 'route_LAX-OAK', 'route_LAX-SFO',
       'route_LAX-SJC', 'route_LAX-SMF', 'route_LAX-TPA', 'route_MCO-LAX',
       'route_MCO-SAN', 'route_MCO-SFO', 'route_MCO-SMF', 'route_MIA-LAX',
       'route_MIA-SFO', 'route_OAK-LAX', 'route_PSP-SFO', 'route_SAN-MCO',
       'route_SAN-SFO', 'route_SAN-SJC', 'route_SAN-SMF', 'route_SAN-TPA',
       'route_SFO-BUR', 'route_SFO-FLL', 'route_SFO-LAX', 'route_SFO-MCO',
       'route_SFO-MIA', 'route_SFO-PSP', 'route_SFO-SAN', 'route_SFO-SNA',
       'route_SFO-TPA', 'route_SJC-LAX', 'route_SJC-SAN', 'route_SJC-SNA',
       'route_SMF-LAX', 'route_SMF-MCO', 'route_SMF-SAN', 'route_SNA-SFO',
       'route_SNA-SJC', 'route_TPA-LAX', 'route_TPA-SAN', 'route_TPA-SFO',
       'FareClass_Others', 'FareClass_Restricted Business Class',
       'FareClass_Restricted Coach Class', 'FareClass_Restricted First Class',
       'FareClass_Unrestricted Business Class',
       'FareClass_Unrestricted Coach Class',
       'FareClass_Unrestricted First Class'],
      dtype='object')
```

In [88]:
```
df1=df[["MktFare"]].copy()
df1.columns
```

Out[88]:
```
Index(['MktFare'], dtype='object')
```

In [89]:
```
df.loc[df['Quarter'] == 1, 'month'] = 2
df.loc[df['Quarter'] == 2, 'month'] = 5
df.loc[df['Quarter'] == 3, 'month'] = 8
df.loc[df['Quarter'] == 4, 'month'] = 11
```

In [90]:
```
df['day']=1
```

In [91]:
```
df1['ds']=pd.to_datetime(df[['Year','month','day']])
```

In [92]:
```
df1.columns=['y','ds']
```

In [93]:
```
df1.head()
```

Out[93]:

	y	ds
0	1.58	2018-02-01
1	1.58	2018-02-01
2	30.18	2018-02-01
3	30.18	2018-02-01
4	45.98	2018-02-01

In [94]:
```
m = p.Prophet()
m.fit(df1)
```

Out[94]:
```
<prophet.forecaster.Prophet at 0x7f1d70e52150>
```

In [95]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[95]:

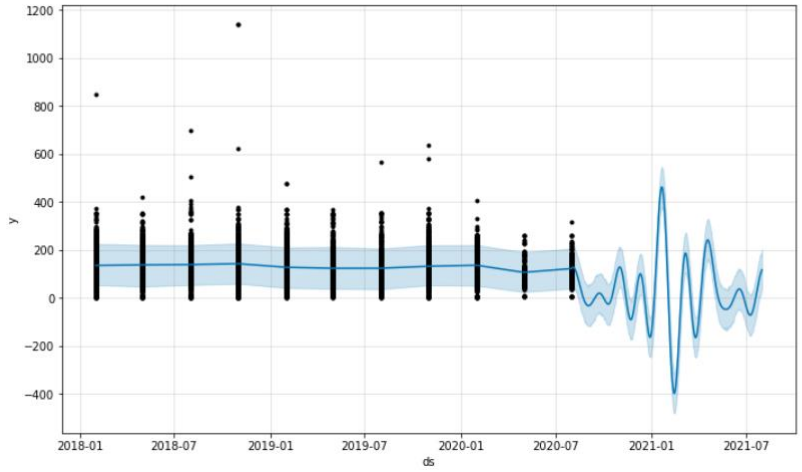
	ds
371	2021-07-28
372	2021-07-29
373	2021-07-30
374	2021-07-31
375	2021-08-01

In [96]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[96]:

	ds	yhat	yhat_lower	yhat_upper
371	2021-07-28	89.205989	0.062405	176.584794
372	2021-07-29	97.682129	13.491005	175.424766
373	2021-07-30	105.044955	14.270335	186.010673
374	2021-07-31	111.205904	26.467354	189.335744
375	2021-08-01	116.099746	39.424642	201.173454

In [97]:
```
fig1 = m.plot(forecast)
```



Machine Hack Dataset:

In [1]:
```
# v9: machine hack data
```

In [2]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.0.1)
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: cmdstanpy==0.9.68 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.9.68)
Requirement already satisfied: ujson in /opt/conda/lib/python3.7/site-packages (from cmdstanpy==0.9.68->prophet) (4.0.2)
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
```

```
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
```

In [3]:
```
import prophet as p
import pandas as pd
```

In [4]:
```
import glob
g=glob.glob('../input/dataset-2-for-airfare/train_data.csv')
```

In [5]:
```
#!pip install openpyxl
```

In [6]:
```
df=pd.read_csv(g[0])
#m = p.Prophet()
#m.fit(df)
df.columns
```

Out[6]:
```
Index(['Unnamed: 0', 'Total_Stops', 'Price', 'Journey_day', 'Journey_month',
       'Departure_hour', 'Departure_Minute', 'Arrival_Hour', 'Arrival_Minute',
       'Duration_Hours', 'Duration_Min', 'Airline_Air India', 'Airline_GoAir',
       'Airline_IndiGo', 'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Cochin', 'Delhi', 'Hyderabad', 'Kolkata', 'New Delhi'],
      dtype='object')
```

In [8]:
```
df1=df[["Price"]].copy()
df1.columns
```

Out[8]:
```
Index(['Price'], dtype='object')
```

In [9]:
```
df2=df[['Journey_month','Journey_day']].copy()
df2.columns
```

Out[9]:
```
Index(['Journey_month', 'Journey_day'], dtype='object')
```

In [11]:
```
df2['Departure_year']=2019
df2.head()
```

Out[11]:

	Journey_month	Journey_day	Departure_year
0	3	24	2019
1	5	1	2019
2	6	9	2019
3	5	12	2019
4	3	1	2019

In [12]:
```
df2.columns=['month','day','Year']
df2.head()
```

Out[12]:

	month	day	Year
0	3	24	2019
1	5	1	2019
2	6	9	2019
3	5	12	2019
4	3	1	2019

In [13]:
```
df1['ds']=pd.to_datetime(df2[['Year','month','day']])
```

In [14]:
```
df1.columns=['y','ds']
```

In [15]:
```
df1.head()
```

Out[15]:

	y	ds
0	3897	2019-03-24
1	7662	2019-05-01
2	13882	2019-06-09
3	6218	2019-05-12
4	13302	2019-03-01

In [16]:
```
m = p.Prophet()
m.fit(df1)
```

Out[16]:
```
<prophet.forecaster.Prophet at 0x7f28b140d050>
```

In [17]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[17]:

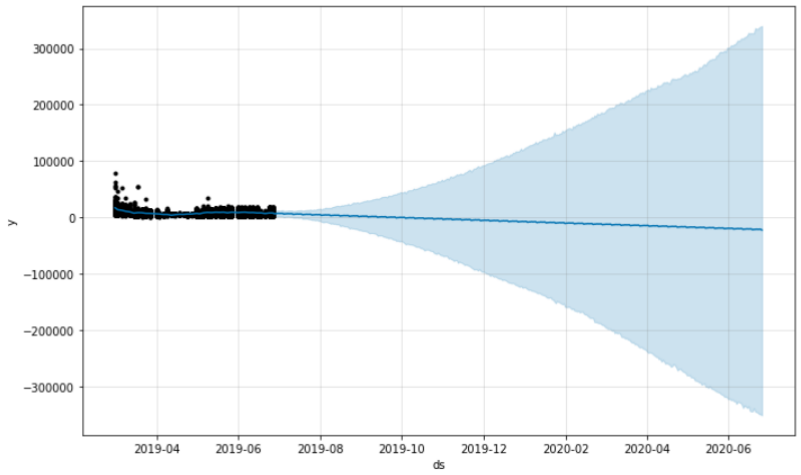
	ds
400	2020-06-22
401	2020-06-23
402	2020-06-24
403	2020-06-25
404	2020-06-26

In [18]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[18]:

	ds	yhat	yhat_lower	yhat_upper
400	2020-06-22	-20466.044952	-344640.406582	334718.893296
401	2020-06-23	-20787.422772	-346082.361795	334557.166006
402	2020-06-24	-20688.903894	-348348.100822	337317.755994
403	2020-06-25	-20459.675186	-348650.042487	338723.577188
404	2020-06-26	-22076.052972	-350606.102597	339816.583973

In [19]:
```
fig1 = m.plot(forecast)
```



Ease-My-Trip Dataset:

In [8]:
```
# v8: EaseMyTrip data
```

In [9]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.7/site-packages (1.0.1)
Requirement already satisfied: Cython>=0.22 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.29.23)
Requirement already satisfied: pystan~=2.19.1.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.19.1.1)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.19.5)
Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2.3)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.8.1)
Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.7/site-packages (from prophet) (4.59.0)
Requirement already satisfied: cmdstanpy==0.9.68 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.9.68)
Requirement already satisfied: convertdate>=2.1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (2.3.2)
Requirement already satisfied: LunarCalendar>=0.0.9 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.0.9)
Requirement already satisfied: holidays>=0.10.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (0.11.1)
Requirement already satisfied: setuptools-git>=1.2 in /opt/conda/lib/python3.7/site-packages (from prophet) (1.2)
Requirement already satisfied: matplotlib>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from prophet) (3.4.1)
Requirement already satisfied: ujson in /opt/conda/lib/python3.7/site-packages (from cmdstanpy==0.9.68->prophet) (4.0.2)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet)
(0.5.11)
```

```
Requirement already satisfied: pytz>=2014.10 in /opt/conda/lib/python3.7/site-packages (from convertdate>=2.1.2->prophet) (202
1.1)
Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (2.1.
1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet) (1.15.0)
Requirement already satisfied: korean-lunar-calendar in /opt/conda/lib/python3.7/site-packages (from holidays>=0.10.2->prophet)
(0.2.1)
Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.7/site-packages (from LunarCalendar>=0.0.9->prophet)
(3.7.7.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (2.
4.7)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (7.2.
0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib>=2.0.0->prophet) (0.10.
0)
```

In [10]:
```
import prophet as p
import pandas as pd
```

In [11]:
```
import glob
g=glob.glob('../input/easemytrip-ff-sample-data-processed/EaseMyTrip.csv')
```

In [12]:
```
#!pip install openpyxl
```

In [13]:
```
df=pd.read_csv(g[0])
#m = p.Prophet()
#m.fit(df)
df.columns
```

Out[13]:
```
Index(['Unnamed: 0', 'Number Of Stops', 'Fare', 'Days Left', 'Departure_day',
       'Departure_month', 'Departure_year', 'Departure_hour',
       'Departure_minute', 'F1_Air Canada',
       ...
       'Layover1_Tiruchirapally', 'Layover1_Tirupati', 'Layover1_Tokyo',
       'Layover1_Udaipur', 'Layover1_VDY', 'Layover1_Vadodara',
       'Layover1_Varanasi', 'Layover1_Vijayawada', 'Layover1_Vishakhapatnam',
       'Layover1_Warsaw'],
      dtype='object', length=366)
```

In [14]:
```
df1=df[["Fare"]].copy()
df1.columns
```

Out[14]:
```
Index(['Fare'], dtype='object')
```

In [20]:
```
df2=df[['Departure_year','Departure_month','Departure_day']].copy()
df2.columns
```

Out[20]:
```
Index(['Departure_year', 'Departure_month', 'Departure_day'], dtype='object')
```

In [21]:
```
df2.columns=['Year','month','day']
df2.head()
```

Out[21]:

	Year	month	day
0	2020	4	23
1	2020	4	16
2	2020	4	18
3	2020	4	27
4	2020	4	18

In [22]:
```
df1['ds']=pd.to_datetime(df2[['Year','month','day']])
```

In [23]:
```
df1.columns=['y','ds']
```

In [24]:
```
df1.head()
```

Out[24]:

	y	ds
0	4378.0	2020-04-23
1	5226.0	2020-04-16
2	3333.0	2020-04-18
3	3563.0	2020-04-27
4	5309.0	2020-04-18

In [25]:
```
m = p.Prophet()
m.fit(df1)
```

Out[25]:
```
<prophet.forecaster.Prophet at 0x7fbb5a22b890>
```

In [26]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[26]:

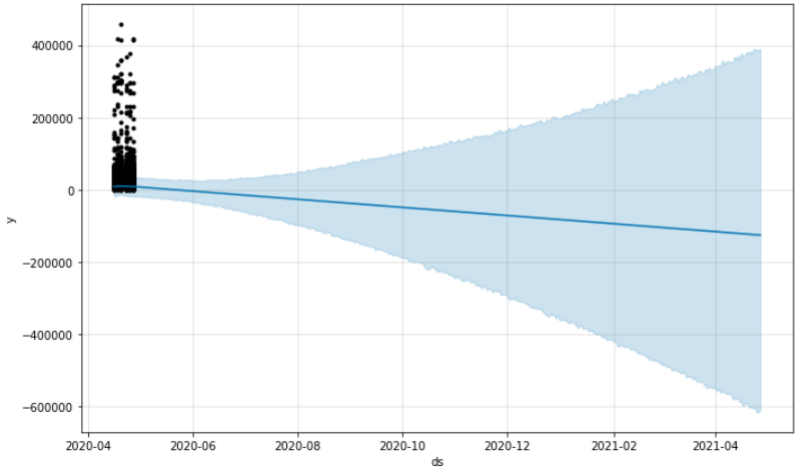
	ds
366	2021-04-23
367	2021-04-24
368	2021-04-25
369	2021-04-26
370	2021-04-27

In [27]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[27]:

	ds	yhat	yhat_lower	yhat_upper
366	2021-04-23	-124230.255066	-606048.765533	389774.453374
367	2021-04-24	-124598.970475	-604318.010856	387670.297455
368	2021-04-25	-124967.685883	-616863.034913	388336.174907
369	2021-04-26	-125336.401291	-615713.269859	385744.473603
370	2021-04-27	-125705.116700	-609882.247079	389533.969781

In [28]:
```
fig1 = m.plot(forecast)
```



Web-app made using flask code:

Flask code:

```
import pickle
import pandas as pd
from flask import request,render_template
from FlaskWebProject1 import app
from os.path import join, dirname, realpath
path=join(dirname(realpath(__file__)), 'static/stylesheets/')

m1=pickle.load(open(path+"RFmodel0.3.pkl", "rb"))
m3=pickle.load(open(path+"BRmodel0.3.pkl", "rb"))
m4=pickle.load(open(path+"GBRmodel0.3.pkl", "rb"))

df=pd.read_csv(path+"sp.csv",nrows=1)
cols1=df.columns
cols1=pd.Series(cols1)
cols1=cols1.drop(0)
cols1=cols1.drop(6)

@app.route('/')
@cross_origin()
def home():
    return render_template("interface.html")

@app.route("/predict", methods = ["GET", "POST"])
@cross_origin()
def predict():

    if request.method == "POST":
        global cols1
        cols=cols1
        print(cols)
        #coupons nums
        mkc=request.form["MktCoupons"]
        cols=cols.replace("MktCoupons",mkc)

        #Year
        mkc=request.form["Year"]
        cols=cols.replace("Year",mkc)

        #Quarter
        mkc=request.form["Quarter"]
        cols=cols.replace("Quarter",mkc)
```

```
        #BulkFare
        mkc=request.form["BulkFare"]
        cols=cols.replace("BulkFare",mkc)
```

```
        #Passengers
        mkc=request.form["Passengers"]
        cols=cols.replace("Passengers",mkc)

        #Pandemic
        mkc=request.form["Pandemic"]
        cols=cols.replace("Pandemic",mkc)

        #airline
        mkc=request.form["airline"]
        cols=cols.replace(mkc,1)

        #Routes
        mkc=request.form["Routes"]
        cols=cols.replace(mkc,1)

        #Seat Class
        mkc=request.form["Seat Class"]
        cols=cols.replace(mkc,1)
        print(cols)


    for i in cols:
        if str(i).isdigit():
            b=0
        else:
            cols=cols.replace(i,0)

    print(cols)


    p1=m1.predict([cols])
    p3=m3.predict([cols])
    p4=m4.predict([cols])
    output=p1*0.24+p3*0.01+p4*0.75
    return render_template('interface.html',prediction_text="Your Flight price is $. {}".format(output))
return
```

```
    cols=cols.replace("BulkFare",mkc)

    #Passengers
    mkc=request.form["Passengers"]
    cols=cols.replace("Passengers",mkc)

    #Pandemic
    mkc=request.form["Pandemic"]
    cols=cols.replace("Pandemic",mkc)

    #airline
    mkc=request.form["airline"]
    cols=cols.replace(mkc,1)

    #Routes
    mkc=request.form["Routes"]
    cols=cols.replace(mkc,1)

    #Seat Class
    mkc=request.form["Seat Class"]
    cols=cols.replace(mkc,1)
    print(cols)

    for i in cols:
        if str(i).isdigit():
            b=0
        else:
            cols=cols.replace(i,0)

    print(cols)


    p1=m1.predict([cols])
    p3=m3.predict([cols])
    p4=m4.predict([cols])
    output=p1*0.24+p3*0.01+p4*0.75
    return render_template('interface.html',prediction_text="Your Flight price is $. {}".format(output))
return render_template('interface.html',prediction_text="")
```

HTML page code:

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flight Price Prediction</title>

    <!-- css -->
    <link rel="stylesheet" href="{{ url_for('static', filename='stylesheets/styles.css') }}">


</head>

<body>

    <!-- As a heading -->
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand" href="/">FLIGHT PRICE ESTIMATOR</a>
            </div>
        </div>
    </nav>

    <br><br><br>


    <div class="container">



        <form action="\predict" method="post">


            <div class="toprow">
                    <div class="card">
                        <div class="card-body">
                         <h5 class="card-title">Number of Stops?</h5>
                            <select name="MktCoupons" required="required">
                                <option value="0">Non-Stop</option>
                                <option value="1">1</option>
                                <option value="2">2</option>
                                <option value="3">3</option>
                                <option value="4">4</option>
```

```
                    </select>
                </div>
            </div>
        </div>

        <br>
        <br>
        <br>
        <div class="row">
            <div class="col-sm-6">
                <div class="card">
                    <div class="card-body">
                        <h5 class="card-title">Year</h5>
                        <!-- Year -->
                        <select name="Year" id="Year" required="required">
                            <option value="2017">2017</option>
                            <option value="2018">2018</option>
                            <option value="2019">2019</option>
                            <option value="2020">2020</option>
                            <option value="2021">2021</option>
                            <option value="2022">2022</option>
                            <option value="2023">2023</option>
                            <option value="2024">2024</option>
                            <option value="2025">2025</option>
                            <option value="2026">2026</option>
                            <option value="2027">2027</option>
                            <option value="2028">2028</option>
                            <option value="2029">2029</option>
                        </select>

                    </div>
                </div>
            </div>
            <br>
            <br>
            <br>

            <div class="col-sm-6">
                <div class="card">
                    <div class="card-body">
                        <h5 class="card-title">Quarter</h5>
                        <!-- Quarter -->
                            <select name="Quarter" id="Quarter" required="required">
                                <option value="1">Quarter 1</option>
                                <option value="2">Quarter 2</option>
                                <option value="3">Quarter 3</option>
                                <option value="4">Quarter 4</option>
                            </select>

                         </div>
                    </div>
                </div>
            </div>

            <br>
            <br>
            <br>
```

```
<div class="row">
    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Extra Bagguage?</h5>
                <!-- BulkFare -->
                <select name="BulkFare" required="required">
                    <option value="0">No</option>
                    <option value="1">Yes</option>
                </select>
            </div>
        </div>
    </div>
    <br>
    <br>
    <br>
    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Number of Passengers?</h5>
                <!-- Passengers -->
                <select name="Passengers" required="required">
                    <option value="1">1</option>
                    <option value="2">2</option>
                    <option value="3">3</option>
                    <option value="4">4</option>
                    <option value="5">5</option>
                </select>
            </div>
        </div>
    </div>
</div>

<br>
<br>
<br>

<div class="row">
    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Pandemic?</h5>
                <!-- Pandemic -->
                <select name="Pandemic" id="Pandemic" required="required">
                    <option value="0">No</option>
                    <option value="1">Yes</option>
                </select>
            </div>
        </div>
    </div>

    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title"> Airline Preference</h5>
                <!-- Airline -->
                <select name="airline" id="airline" required="required">
                    <option value="RPCarrier_9E"> Endeavor Air</option>
                    <option value="RPCarrier_AA"> American Airlines</option>
                    <option value="RPCarrier_AS"> Alaska Airlines</option>
                    <option value="RPCarrier_B6"> JetBlue Airways</option>
                    <option value="RPCarrier_CP"> Compass Airlines</option>
                    <option value="RPCarrier_DL"> Delta Air Lines</option>
                    <option value="RPCarrier_EV"> ExpressJet Airlines</option>
                    <option value="RPCarrier_F9"> Frontier Airlines</option>
                    <option value="RPCarrier_G7"> GoJet Airlines</option>
                    <option value="RPCarrier_HA"> Hawaiian Airlines</option>
                    <option value="RPCarrier_MQ"> Envoy Air</option>
                    <option value="RPCarrier_NK"> Spirit Air Lines</option>
                    <option value="RPCarrier_OH"> PSA Airlines</option>
                    <option value="RPCarrier_OO"> SkyWest Airlines</option>
                    <option value="RPCarrier_QX"> Horizon Air</option>
                    <option value="RPCarrier_SY"> Sun Country Airlines</option>
                    <option value="RPCarrier_UA"> United Air Lines</option>
                    <option value="RPCarrier_VX"> Virgin America</option>
                    <option value="RPCarrier_WN"> Southwest Airlines</option>
                    <option value="RPCarrier_YV"> Mesa Airlines</option>
                    <option value="RPCarrier_YX"> Republic Airline</option>
                </select>
```

```
                    </div>
                </div>
            </div>
        </div>



        <br>
        <br>
        <br>

        <div class="row">
            <div class="col-sm-6">
                <div class="card">
                    <div class="card-body">
                        <!-- Routes -->
                        <h5 class="card-title">Routes</h5>
                        <select name="Routes" id="Routes" required="required">
                            <option value="route_BUR-SFO"> Burbank, USA (BUR) to San Francisco, USA (SFO) </option>
                            <option value="route_FLL-LAX"> Fort Lauderdale, USA (FLL) to Los Angeles, USA (LAX) </option>
                            <option value="route_FLL-SAN"> Fort Lauderdale, USA (FLL) to San Diego, USA (SAN) </option>
                            <option value="route_FLL-SFO"> Fort Lauderdale, USA (FLL) to San Francisco, USA (SFO) </option>
                            <option value="route_LAX-FLL"> Los Angeles, USA (LAX) to Fort Lauderdale, USA (FLL) </option>
                            <option value="route_LAX-MCO"> Los Angeles, USA (LAX) to Orlando, USA (MCO) </option>
                            <option value="route_LAX-MIA"> Los Angeles, USA (LAX) to Miami, USA (MIA) </option>
                            <option value="route_LAX-OAK"> Los Angeles, USA (LAX) to Oakland, USA (OAK) </option>
                            <option value="route_LAX-SFO"> Los Angeles, USA (LAX) to San Francisco, USA (SFO) </option>
                            <option value="route_LAX-SJC"> Los Angeles, USA (LAX) to San Jose, USA (SJC) </option>
                            <option value="route_LAX-SMF"> Los Angeles, USA (LAX) to Sacramento, USA (SMF) </option>
                            <option value="route_LAX-TPA"> Los Angeles, USA (LAX) to Tampa, USA (TPA) </option>
                            <option value="route_MCO-LAX"> Orlando, USA (MCO) to Los Angeles, USA (LAX) </option>
                            <option value="route_MCO-SAN"> Orlando, USA (MCO) to San Diego, USA (SAN) </option>
                            <option value="route_MCO-SFO"> Orlando, USA (MCO) to San Francisco, USA (SFO) </option>
                            <option value="route_MCO-SMF"> Orlando, USA (MCO) to Sacramento, USA (SMF) </option>
                            <option value="route_MIA-LAX"> Miami, USA (MIA) to Los Angeles, USA (LAX) </option>
                            <option value="route_MIA-SFO"> Miami, USA (MIA) to San Francisco, USA (SFO) </option>
                            <option value="route_OAK-LAX"> Oakland, USA (OAK) to Los Angeles, USA (LAX) </option>
                            <option value="route_PSP-SFO"> Palm Springs, USA (PSP) to San Francisco, USA (SFO) </option>
                            <option value="route_SAN-MCO"> San Diego, USA (SAN) to Orlando, USA (MCO) </option>
                            <option value="route_SAN-SFO"> San Diego, USA (SAN) to San Francisco, USA (SFO) </option>
                            <option value="route_SAN-SJC"> San Diego, USA (SAN) to San Jose, USA (SJC) </option>
                        </select>
                    </div>
                </div>
            </div>
            <div class="col-sm-6">
                <div class="card">
                    <div class="card-body">
                        <h5 class="card-title">Seat Class</h5>
                        <!-- Seat Class-->
                        <select name="Seat Class" id="Seat Class" required="required">
                            <option value="FareClass_Others"> Others </option>
                            <option value="FareClass_Restricted Business Class"> Restricted Business Class </option>
                            <option value="FareClass_Restricted Coach Class"> Restricted Coach Class </option>
                            <option value="FareClass_Restricted First Class"> Restricted First Class </option>
                            <option value="FareClass_Unrestricted Business Class"> Unrestricted Business Class </option>
                            <option value="FareClass_Unrestricted Coach Class"> Unrestricted Coach Class </option>
                            <option value="FareClass_Unrestricted First Class"> Unrestricted First Class </option>
                        </select>
                    </div>
                </div>
            </div>
        </div>



        <br>
        <br>
        <br>
        <!-- Submit -->
        <input type="submit" value="Submit" class="btn-secondary">
    </form>
        <br>
        <br>
        <h3>{{ prediction_text }}</h3>

        <br>
        <br>


    </div>
</body>

</html>
```

CSS page code:

```css
body {
    background-color: #e1f4f3;
    text-align: center;
}

.navbar {
    background-color: #333333;
    justify-content: space-between;
    padding: .5rem 1rem;
}


.navbar-brand {
    display: inline-block;
    padding-top: .3125rem;
    padding-bottom: .3125rem;
    margin-right: 1rem;
    font-size: 2rem;
    line-height: inherit;
    white-space: nowrap
}

.col-sm-6 {
    -ms-flex: 0 0 50%;
    flex: 0 0 50%;
    max-width: 50%
}

a {
    color: #f1f9f9;
}

a:hover {
    color: #f0f0f0;
 }

.h5, h5 {
    font-size: 1.25rem
}

.card {
    position: relative;
    display: -ms-flexbox;
    display: flex;
    -ms-flex-direction: column;
    flex-direction: column;
    min-width: 0;
    word-wrap: break-word;
    background-color: #fff;
    background-clip: border-box;
    border: 1px solid rgba(0,0,0,.125);
    border-radius: .25rem
}


.card-body {
    -ms-flex: 1 1 auto;
    flex: 1 1 auto;
    min-height: 1px;
    padding: 1.25rem
}

.card-title {
    margin-bottom: .75rem
}

.row {
    display: -ms-flexbox;
    display: flex;
    -ms-flex-wrap: wrap;
    flex-wrap: wrap;
    margin-right: -15px;
    margin-left: -15px
}
.btn-secondary {
    color: #fff;
    background-color: #6c757d;
    border-color: #6c757d
}
```

Webpage:


FLIGHT PRICE ESTIMATOR
Number of Stops?
Non-Stop
Year
2017
Quarter
Quarter 1
Extra Bagguage?
No
Number of Passengers?
1
Pandemic?
No
Airline Preference
Endeavor Air
Routes
Burbank, USA (BUR) to San Francisco, USA (SFO)
Seat Class
Others
Submit
FLIGHT PRICE ESTIMATOR
Number of Stops?
Year
Quarter
Extra Bagguage?
Number of Passengers?
Pandemic?
Airline Preference
Routes
Seat Class
Submit
Your Flight price is $. [28.79819257]

7. RESULT AND DISCUSSION

When looking at the DB1B dataset under normal view:

Table 2 DB1B Dataset Normal View

Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
RandomForestRegressor()	0.7	0.3	0.358345266	0.358141939	194.9068483	0	0	0	0	559.476681
DecisionTreeRegressor()	0.7	0.3	0.340996756	0.340787932	197.5241374	0	0	0	0	8.926332
BaggingRegressor()	0.7	0.3	0.351828184	0.351622792	195.894151	0	0	0	0	62.4030325
GradientBoostingRegressor()	0.7	0.3	0.37815008	0.377953029	191.8753518	0	0	0	0	19.9425881
weight 1()	0.7	0.3	0.358345266	0.358141939	194.9068483	100	0	0	0	0
weight 2()	0.7	0.3	0.359114611	0.358911528	194.7899665	135	-25	-10	0	0
weight 3()	0.7	0.3	0.380678775	0.380482526	191.4848326	25	0	0	75	0
weight 4()	0.7	0.3	0.380733683	0.380537451	191.4763441	35	0	-10	75	0
RandomForestRegressor()	0.8	0.2	0.350036444	0.349727456	198.3918664	0	0	0	0	661.176472
DecisionTreeRegressor()	0.8	0.2	0.336134262	0.335818665	200.5023595	0	0	0	0	10.7198403
BaggingRegressor()	0.8	0.2	0.350192665	0.349883751	198.3680229	0	0	0	0	72.555686
GradientBoostingRegressor()	0.8	0.2	0.371076396	0.37077741	195.154382	0	0	0	0	22.2037375
weight 1()	0.8	0.2	0.351380681	0.351072332	198.1866059	48	0	52	0	0
weight 2()	0.8	0.2	0.351978984	0.351670919	198.0951787	70	-25	55	0	0
weight 3()	0.8	0.2	0.373476194	0.373178348	194.7816987	3	0	22	75	0
weight 4()	0.8	0.2	0.373476194	0.373178348	194.7816987	3	0	22	75	0
RandomForestRegressor()	0.9	0.1	0.337559386	0.336929243	200.5588593	0	0	0	0	756.893783
DecisionTreeRegressor()	0.9	0.1	0.322531176	0.321886737	202.8210528	0	0	0	0	12.9207265
BaggingRegressor()	0.9	0.1	0.333661944	0.333028093	201.147984	0	0	0	0	86.0798488
GradientBoostingRegressor()	0.9	0.1	0.355762802	0.355149974	197.7840539	0	0	0	0	24.2624419
weight 1()	0.9	0.1	0.337571737	0.336941605	200.5569898	95	0	5	0	0
weight 2()	0.9	0.1	0.338295052	0.337665608	200.4474643	116	-27	11	0	0
weight 3()	0.9	0.1	0.358414936	0.357804631	197.3765248	20	0	6	74	0
weight 4()	0.9	0.1	0.358414936	0.357804631	197.3765248	20	0	6	74	0
RandomForestRegressor()	0.95	0.05	0.335228699	0.333962773	195.9129537	0	0	0	0	796.651448
DecisionTreeRegressor()	0.95	0.05	0.32050403	0.319210063	198.0708038	0	0	0	0	12.7974684
BaggingRegressor()	0.95	0.05	0.33460507	0.333337957	196.0048261	0	0	0	0	95.7993083
GradientBoostingRegressor()	0.95	0.05	0.356119458	0.354893315	192.8100519	0	0	0	0	24.8737531
weight 1()	0.95	0.05	0.335892907	0.334628246	195.8150557	58	0	42	0	0
weight 2()	0.95	0.05	0.337868228	0.336607329	195.5236226	110	-52	42	0	0
weight 3()	0.95	0.05	0.358075503	0.356853084	192.5169603	0	0	22	78	0
weight 4()	0.95	0.05	0.358094632	0.356872249	192.5140919	-7	0	29	78	0

The time taken for training follows the order DT<GBR<BTR<RFR

The R squared value follows the trend:

weight4>= weight3>= weight2>= weight1>GBR>RF>BR>DT

With an increase in training size, we see a decrease in the R-Squared value.

The order of priority for weights in the combined model seems to be

GBR>RF>BTR>DT

The R2 value range is 0.3205- 0.3807

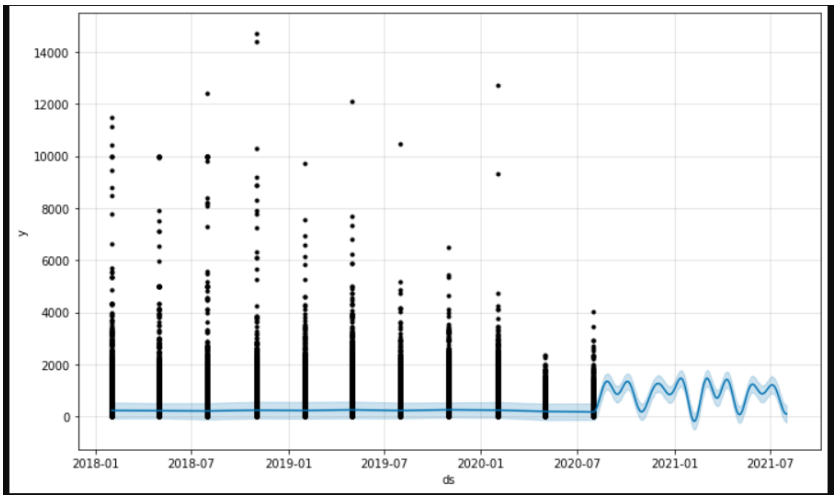


Figure 22 Prophet DB1B Dataset normal view

When using Facebook's Prophet algorithm, which is used to predict using time series data, to understand the trend in the DB1B dataset Normal view see a pattern of rise and falls but the scale at which this occurs is larger than inputted value's scale which is likely a result of outliers.

When the DB1B dataset normal view is split based on years and quarters we get the table 7.2

	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	2018	0	RandomForestRegressor()	0.7	0.3	0.324288568	0.323848332	196.9003254	0	0	0	0	251.1734965
1	2018	0	DecisionTreeRegressor()	0.7	0.3	0.303094647	0.302640604	199.9644074	0	0	0	0	4.041873932
2	2018	0	BaggingRegressor()	0.7	0.3	0.321049098	0.320606752	197.3717471	0	0	0	0	27.63488698
3	2018	0	GradientBoostingRegressor()	0.7	0.3	0.351153326	0.350730594	192.9464719	0	0	0	0	8.570236683
4	2018	0	weight 1()	0.7	0.3	0.324535866	0.324095791	196.8642911	79	0	21	0	0
5	2018	0	weight 2()	0.7	0.3	0.326014345	0.325575233	196.6487215	110	-34	24	0	0
6	2018	0	weight 3()	0.7	0.3	0.352024569	0.351602404	192.8168884	15	0	0	85	0
7	2018	0	weight 4()	0.7	0.3	0.352025359	0.351603195	192.8167708	16	0	-1	85	0
8	2019	0	RandomForestRegressor()	0.7	0.3	0.351695035	0.351129574	201.0604674	0	0	0	0	168.0506074
9	2019	0	DecisionTreeRegressor()	0.7	0.3	0.336649111	0.336070526	203.3802008	0	0	0	0	2.907724619
10	2019	0	BaggingRegressor()	0.7	0.3	0.349916735	0.349349723	201.3360329	0	0	0	0	17.54399347
11	2019	0	GradientBoostingRegressor()	0.7	0.3	0.361664155	0.361107389	199.5086041	0	0	0	0	5.474424601
12	2019	0	weight 1()	0.7	0.3	0.351786799	0.351221418	201.0462374	82	0	18	0	0
13	2019	0	weight 2()	0.7	0.3	0.355185144	0.354622727	200.5185388	150	-75	25	0	0
14	2019	0	weight 3()	0.7	0.3	0.363370785	0.362815507	199.2417265	5	0	22	73	0
15	2019	0	weight 4()	0.7	0.3	0.363370785	0.362815507	199.2417265	5	0	22	73	0
16	2020	0	RandomForestRegressor()	0.7	0.3	0.444436511	0.443260416	177.4489986	0	0	0	0	67.04489088
17	2020	0	DecisionTreeRegressor()	0.7	0.3	0.415936922	0.414700495	181.9435152	0	0	0	0	1.118443489
18	2020	0	BaggingRegressor()	0.7	0.3	0.43861488	0.437426461	178.3763007	0	0	0	0	5.441787004
19	2020	0	GradientBoostingRegressor()	0.7	0.3	0.470898281	0.469778204	173.171449	0	0	0	0	2.52421999
20	2020	0	weight 1()	0.7	0.3	0.444439609	0.44326352	177.448504	98	0	2	0	0
21	2020	0	weight 2()	0.7	0.3	0.447559871	0.446390388	176.9494893	139	-46	7	0	0
22	2020	0	weight 3()	0.7	0.3	0.471723198	0.470604868	173.0364014	10	0	5	85	0
23	2020	0	weight 4()	0.7	0.3	0.471723198	0.470604868	173.0364014	10	0	5	85	0
24	0	1	RandomForestRegressor()	0.7	0.3	0.407350998	0.406688858	190.6123601	0	0	0	0	119.3511918
25	0	1	DecisionTreeRegressor()	0.7	0.3	0.384262892	0.383574957	194.2897744	0	0	0	0	2.077735186
26	0	1	BaggingRegressor()	0.7	0.3	0.405672094	0.405008078	190.8821604	0	0	0	0	13.99219155
27	0	1	GradientBoostingRegressor()	0.7	0.3	0.435266081	0.434635129	186.0690828	0	0	0	0	5.235390186

Figure 23 DB1B Dataset Normal divided into years and quarters

The time taken for training follows the order DT<GBR<BTR<RFR

The R squared value follows the trend:

weight4>= weight3>weight2>= weight1>GBR>RF>BR>DT

With an increase in training size, we see a decrease in the R-Squared value.

The order of priority for weights in the combined model seems to be

GBR>RF>=BTR>DT

The R2 value range is 0.2506- 0.4717

Since an increase was made when the data was divided into years and quarters. Dividing the normal Dataset into sub-routes result in Table 7.3

Table 3 DB1B Dataset Sub-route View

	Route Name	Model Name	Train siz	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	SAN-SJC	RandomForestRegressor()	0.7	0.3	0.116189113	0.098880967	62.75948981	0	0	0	0	1.181429386
1	SAN-SJC	DecisionTreeRegressor()	0.7	0.3	0.087036017	0.06915695	63.78617401	0	0	0	0	0.016239882
2	SAN-SJC	BaggingRegressor()	0.7	0.3	0.115021812	0.097690806	62.80092121	0	0	0	0	0.156823158
3	SAN-SJC	GradientBoostingRegressor()	0.7	0.3	0.1549212	0.138371565	61.36889937	0	0	0	0	0.788021564
4	SAN-SJC	weight 1()	0.7	0.3	0.117896552	0.100621843	62.69883781	56	0	44	0	0
5	SAN-SJC	weight 2()	0.7	0.3	0.117931961	0.100657946	62.69757937	61	-4	43	0	0
6	SAN-SJC	weight 3()	0.7	0.3	0.1549212	0.138371565	61.36889937	0	0	0	100	0
7	SAN-SJC	weight 4()	0.7	0.3	0.156234839	0.13971093	61.32118314	-31	0	11	120	0
8	SAN-SJC	RandomForestRegressor()	0.8	0.2	0.118496573	0.092346078	62.87284131	0	0	0	0	1.401438713
9	SAN-SJC	DecisionTreeRegressor()	0.8	0.2	0.083743796	0.056562331	64.10022425	0	0	0	0	0.020223141
10	SAN-SJC	BaggingRegressor()	0.8	0.2	0.106358889	0.079848319	63.30421893	0	0	0	0	0.188565969
11	SAN-SJC	GradientBoostingRegressor()	0.8	0.2	0.153605591	0.128496632	61.60805211	0	0	0	0	0.462889194
12	SAN-SJC	weight 1()	0.8	0.2	0.118496573	0.092346078	62.87284131	100	0	0	0	0
13	SAN-SJC	weight 2()	0.8	0.2	0.121383106	0.095318241	62.76981659	166	-33	-33	0	0
14	SAN-SJC	weight 3()	0.8	0.2	0.153605591	0.128496632	61.60805211	0	0	0	100	0
15	SAN-SJC	weight 4()	0.8	0.2	0.154247357	0.129157436	61.584691	18	0	-25	107	0
16	SAN-SJC	RandomForestRegressor()	0.9	0.1	0.137333788	0.084585622	66.03328831	0	0	0	0	1.416697264
17	SAN-SJC	DecisionTreeRegressor()	0.9	0.1	0.115031868	0.06092004	66.88139862	0	0	0	0	0.020126581
18	SAN-SJC	BaggingRegressor()	0.9	0.1	0.125011211	0.071509575	66.50323546	0	0	0	0	0.187899113
19	SAN-SJC	GradientBoostingRegressor()	0.9	0.1	0.176282739	0.125916126	64.5253864	0	0	0	0	0.278117418
20	SAN-SJC	weight 1()	0.9	0.1	0.137333788	0.084585622	66.03328831	100	0	0	0	0
21	SAN-SJC	weight 2()	0.9	0.1	0.137980746	0.085272138	66.00852279	130	-3	-27	0	0
22	SAN-SJC	weight 3()	0.9	0.1	0.176282739	0.125916126	64.5253864	0	0	0	100	0
23	SAN-SJC	weight 4()	0.9	0.1	0.178226095	0.12797831	64.44922565	42	0	-49	107	0

The time taken for training follows the order DT <BTR<GBR <RFR

The R squared value follows the trend:

weight4>= weight3>weight2>= weight1>GBR>RF>BR>DT

Depending on the sub-route in question, an increase in training size causes a decrease in the R-Squared value. Whereas in some cases R-Squared value increases.

The order of priority for weights in the combined model seems to be

GBR>RF>=BTR>DT

The R2 value range is -3.0511- 0.5612

When looking at Facebook's prophet's prediction for each various routes in images 7.2-4 we see that the expected trends are drastically magnified likely due the outlier distribution. Though the trends maybe magnified we are able to see how the price is expected to change with time.

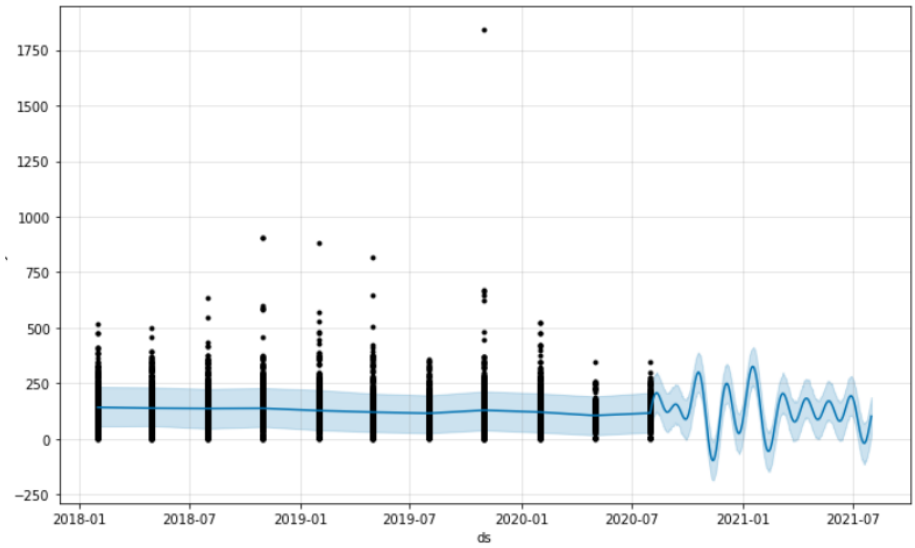


Figure 24 Prophet BUR-SFO Route

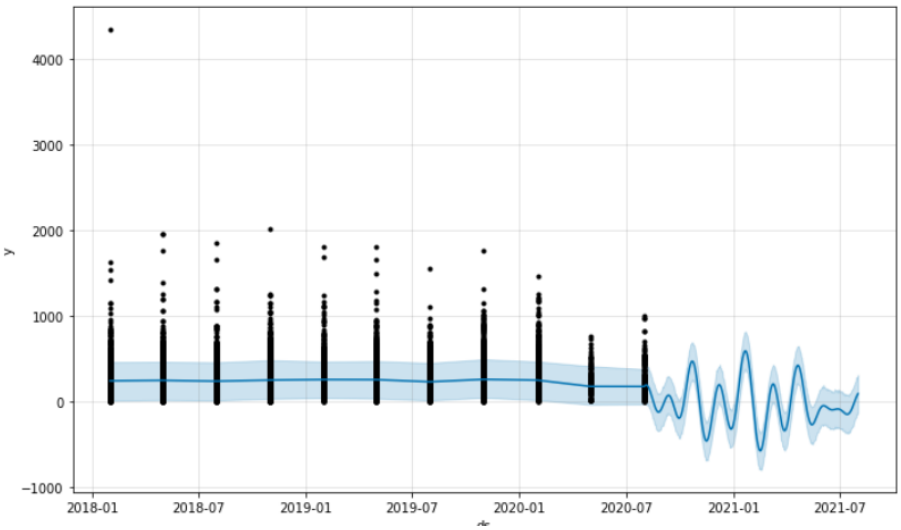


Fig 7.3: Prophet FLL-LAX Route

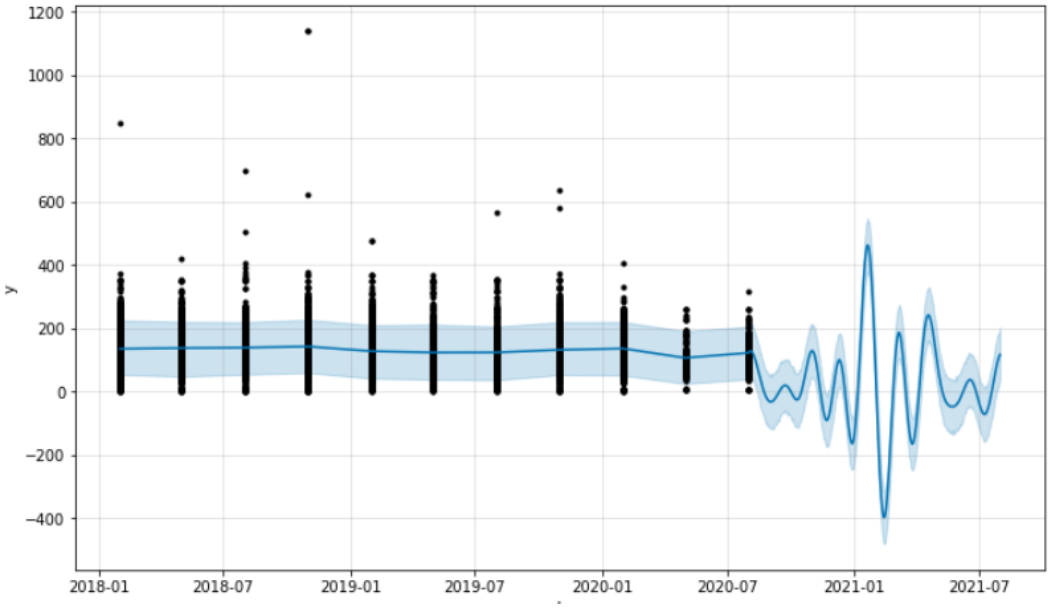


Fig 7.4: Prophet LAX- FLL Route

Sub-Routes are divided based on years and quarters to see the effects of a time period has.

Table 4 DB1B Dataset Sub-route View divided based on years and quarters

Route Name	Year	Quarter	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
SAN-SJC	2018	0	RandomForestRegressor()	0.7	0.3	0.1716593	0.135880779	58.34043192	0	0	0	0	0.449816942
SAN-SJC	2018	0	DecisionTreeRegressor()	0.7	0.3	0.131166218	0.093638679	59.74939184	0	0	0	0	0.006108761
SAN-SJC	2018	0	BaggingRegressor()	0.7	0.3	0.175634067	0.140027228	58.20029121	0	0	0	0	0.063536882
SAN-SJC	2018	0	GradientBoostingRegressor()	0.7	0.3	0.182611902	0.147306456	57.95344997	0	0	0	0	0.580065012
SAN-SJC	2018	0	weight 1()	0.7	0.3	0.176398545	0.140824726	58.17329888	29	0	71	0	0
SAN-SJC	2018	0	weight 2()	0.7	0.3	0.176768178	0.141210324	58.1602433	37	-10	73	0	0
SAN-SJC	2018	0	weight 3()	0.7	0.3	0.188424778	0.153370408	57.74701357	0	0	40	60	0
SAN-SJC	2018	0	weight 4()	0.7	0.3	0.189030583	0.15400238	57.72545676	-28	0	62	66	0
SAN-SJC	2019	0	RandomForestRegressor()	0.7	0.3	0.00100714	-0.055770152	64.89041901	0	0	0	0	0.318933487
SAN-SJC	2019	0	DecisionTreeRegressor()	0.7	0.3	-0.016915776	-0.074711709	65.46993034	0	0	0	0	0.004737616
SAN-SJC	2019	0	BaggingRegressor()	0.7	0.3	-0.008893678	-0.066233678	65.21118416	0	0	0	0	0.042857647
SAN-SJC	2019	0	GradientBoostingRegressor()	0.7	0.3	0.011664617	-0.044506962	64.54335822	0	0	0	0	0.678832769
SAN-SJC	2019	0	weight 1()	0.7	0.3	0.00100714	-0.055770152	64.89041901	100	0	0	0	0
SAN-SJC	2019	0	weight 2()	0.7	0.3	0.001276623	-0.055485354	64.88166618	122	-10	-12	0	0
SAN-SJC	2019	0	weight 3()	0.7	0.3	0.011668997	-0.044502333	64.54321518	2	0	0	98	0
SAN-SJC	2019	0	weight 4()	0.7	0.3	0.012081871	-0.044065994	64.52973238	23	0	-23	100	0
SAN-SJC	2020	0	RandomForestRegressor()	0.7	0.3	0.091518172	-0.017720892	59.9525675	0	0	0	0	0.2315588
SAN-SJC	2020	0	DecisionTreeRegressor()	0.7	0.3	0.046358036	-0.068311242	61.42460057	0	0	0	0	0.002711058
SAN-SJC	2020	0	BaggingRegressor()	0.7	0.3	0.078086214	-0.032767955	60.39414243	0	0	0	0	0.030804634
SAN-SJC	2020	0	GradientBoostingRegressor()	0.7	0.3	0.075056607	-0.036161853	60.49329512	0	0	0	0	0.630218983
SAN-SJC	2020	0	weight 1()	0.7	0.3	0.091518172	-0.017720892	59.9525675	100	0	0	0	0
SAN-SJC	2020	0	weight 2()	0.7	0.3	0.095648649	-0.013093751	59.81612292	194	-36	-58	0	0
SAN-SJC	2020	0	weight 3()	0.7	0.3	0.092546911	-0.016568453	59.91861359	80	0	0	20	0
SAN-SJC	2020	0	weight 4()	0.7	0.3	0.093541323	-0.015454469	59.88577435	114	0	-35	21	0
SAN-SJC	2018	0	RandomForestRegressor()	0.9	0.1	0.105890625	-0.020922016	63.12019842	0	0	0	0	0.618160725
SAN-SJC	2018	0	DecisionTreeRegressor()	0.9	0.1	0.034072941	-0.10292569	65.60624737	0	0	0	0	0.008749008
SAN-SJC	2018	0	BaggingRegressor()	0.9	0.1	0.098516019	-0.029342571	63.37997127	0	0	0	0	0.088543653
SAN-SJC	2018	0	GradientBoostingRegressor()	0.9	0.1	0.164068883	0.045507737	61.03209486	0	0	0	0	0.708754301
SAN-SJC	2018	0	weight 1()	0.9	0.1	0.105890625	-0.020922016	63.12019842	100	0	0	0	0
SAN-SJC	2018	0	weight 2()	0.9	0.1	0.130648442	0.007347234	62.24016701	210	-100	-10	0	0

The time taken for training follows the order DT <BTR <RFR<GBR

The general R squared value follows the trend:

weight4>= weight3>weight2>= weight1>GBR>RF>BR>DT

The order of priority for weights in the combined model seems to be

GBR>RF>BTR>DT

The R2 value range is -5.0394- 0.6640

We see that by splitting the data based on year and quarter can increase the R2 value.

The value of the R2 now not only depends on quality of the data for that specific quarter and year. Though it is likely to increase R2 value by increasing training size up to a certain extent.

For the Machine Hack Dataset:

Table 5 Machine Hack Dataset model summary table

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.81666932	0.814994804	1989.49821	0	0	0	0	2.58297157
1	DecisionTreeRegressor()	0.7	0.3	0.68430759	0.681424097	2610.7065	0	0	0	0	0.03671741
2	BaggingRegressor()	0.7	0.3	0.78388618	0.781912225	2160.06715	0	0	0	0	0.27589893
3	GradientBoostingRegressor(	0.7	0.3	0.790258	0.788342251	2127.98556	0	0	0	0	0.88143134
4	weight 1()	0.7	0.3	0.81666932	0.814994804	1989.49821	100	0	0	0	0
5	weight 2()	0.7	0.3	0.8222373	0.820613644	1959.05349	155	-14	-41	0	0
6	weight 3()	0.7	0.3	0.82988988	0.828336119	1916.42159	63	0	0	37	0
7	weight 4()	0.7	0.3	0.83155832	0.830019798	1907.00029	98	0	-32	34	0
8	RandomForestRegressor()	0.8	0.2	0.8092682	0.806643032	2115.46063	0	0	0	0	2.92606401
9	DecisionTreeRegressor()	0.8	0.2	0.70330856	0.699225003	2638.43196	0	0	0	0	0.04101563
10	BaggingRegressor()	0.8	0.2	0.79949952	0.796739898	2168.95781	0	0	0	0	0.29808927
11	GradientBoostingRegressor(	0.8	0.2	0.77229513	0.76916108	2311.42364	0	0	0	0	0.91597295
12	weight 1()	0.8	0.2	0.80981103	0.80719334	2112.44811	81	0	19	0	0
13	weight 2()	0.8	0.2	0.81334414	0.810775074	2092.73484	99	-22	23	0	0
14	weight 3()	0.8	0.2	0.82113834	0.818676551	2048.57571	32	0	34	34	0
15	weight 4()	0.8	0.2	0.82113834	0.818676551	2048.57571	32	0	34	0	0
16	RandomForestRegressor()	0.9	0.1	0.80384131	0.798366233	2207.97695	0	0	0	0	3.27804637
17	DecisionTreeRegressor()	0.9	0.1	0.69711729	0.688663398	2743.64548	0	0	0	0	0.04628873
18	BaggingRegressor()	0.9	0.1	0.78509906	0.77910086	2311.05305	0	0	0	0	0.33849406
19	GradientBoostingRegressor(	0.9	0.1	0.80402875	0.798558906	2206.92177	0	0	0	0	0.92478561
20	weight 1()	0.9	0.1	0.80384131	0.798366233	2207.97695	100	0	0	0	0
21	weight 2()	0.9	0.1	0.81385632	0.808660782	2150.87365	168	-38	-30	0	0
22	weight 3()	0.9	0.1	0.8146062	0.80943159	2146.5369	50	0	0	50	0
23	weight 4()	0.9	0.1	0.81726989	0.812169627	2131.06066	98	0	-51	53	0
24	RandomForestRegressor()	0.95	0.05	0.78405373	0.771652857	2489.71588	0	0	0	0	3.3264451
25	DecisionTreeRegressor()	0.95	0.05	0.68186688	0.663597852	3021.90898	0	0	0	0	0.04838324
26	BaggingRegressor()	0.95	0.05	0.8049203	0.793717704	2366.37178	0	0	0	0	0.35030341
27	GradientBoostingRegressor(	0.95	0.05	0.75835993	0.744483571	2633.67035	0	0	0	0	1.82898593
28	weight 1()	0.95	0.05	0.8049203	0.793717704	2366.37178	0	0	100	0	0
29	weight 2()	0.95	0.05	0.82447039	0.814390468	2244.66811	-11	-57	168	0	0
30	weight 3()	0.95	0.05	0.80522504	0.794039946	2364.52275	0	0	93	7	0
31	weight 4()	0.95	0.05	0.81079026	0.799924746	2330.49775	-86	0	164	22	0

Table 7.5: Machine Hack Dataset model summary table

The time taken for training follows the order DT <GBR <BTR <RFR

The general R squared value follows the trend:

weight4>= weight3>weight2>= weight1> RF >GBR>BR>DT

The order of priority for weights in the combined model seems to be

RF>BTR>= GBR>DT

The R2 value range is 0.6818- 0.8315

We observe the GBR is worsening when compared to DB1B dataset this can be due to decrease in number of columns present in the dataset.

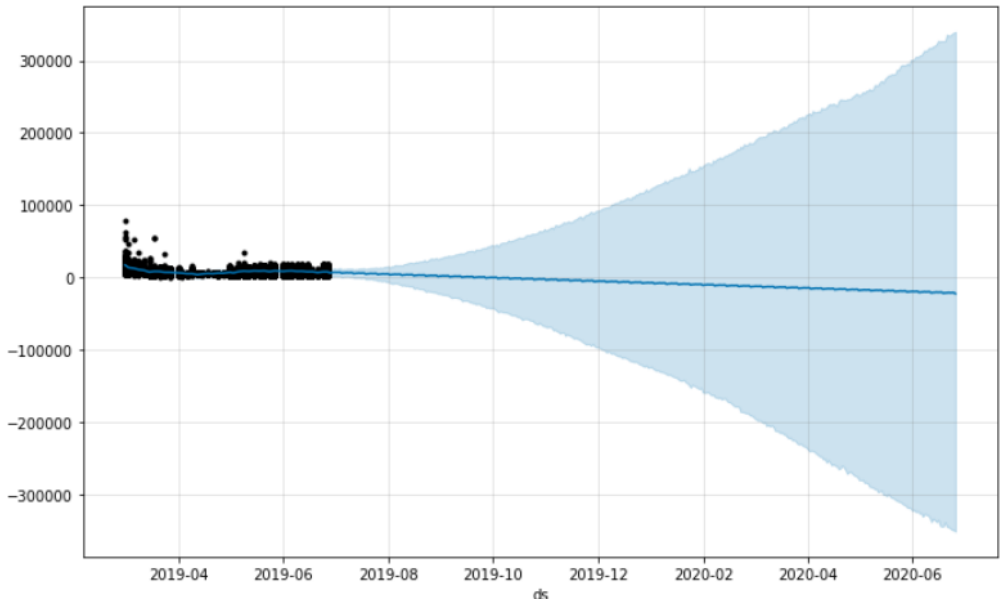


Figure 25 Prophet's – Machine Hack Dataset

Based on Prophet we can expect a steady decrease in average price but as more time passes it is more likely for large variations to occur. This can be due to the fact that a small period of data is used to predict prices.

For the Dataset acquired from Ease-My-Trip:

Table 6 Ease-My-Trip Dataset model summary table

	Model Name	Train size	Test size	r2	Adjusted r2	RMSE	RF weight	DT weight	BR weight value	GBR weight	Time
0	RandomForestRegressor()	0.7	0.3	0.886024757	0.877447749	6768.09657	0	0	0	0	84.8027005
1	DecisionTreeRegressor()	0.7	0.3	0.892217661	0.884106689	6581.65458	0	0	0	0	1.61351776
2	BaggingRegressor()	0.7	0.3	0.877796445	0.86860023	7008.147	0	0	0	0	4.89872432
3	GradientBoostingRegressor(	0.7	0.3	-139.188039	-149.737645	237365.045	0	0	0	0	0.60400581
4	weight 1()	0.7	0.3	0.897911538	0.890229049	6405.44952	28	60	12	0	0
5	weight 2()	0.7	0.3	0.897911538	0.890229049	6405.44952	28	60	12	0	0
6	weight 3()	0.7	0.3	0.886024757	0.877447749	6768.09657	100	0	0	0	0
7	weight 4()	0.7	0.3	0.886065566	0.877491629	6766.8848	108	0	-8	0	0
8	RandomForestRegressor()	0.8	0.2	0.862254416	0.846096055	8162.52312	0	0	0	0	106.325064
9	DecisionTreeRegressor()	0.8	0.2	0.866768319	0.851139466	8027.66675	0	0	0	0	1.97833133
10	BaggingRegressor()	0.8	0.2	0.853238795	0.83602285	8425.41333	0	0	0	0	5.77263355
11	GradientBoostingRegressor(	0.8	0.2	-163.945189	-183.294222	282458.994	0	0	0	0	0.54491448
12	weight 1()	0.8	0.2	0.876081904	0.861545588	7741.99609	7	60	33	0	0
13	weight 2()	0.8	0.2	0.876081904	0.861545588	7741.99609	7	60	33	0	0
14	weight 3()	0.8	0.2	0.862254416	0.846096055	8162.52312	100	0	0	0	0
15	weight 4()	0.8	0.2	0.862925657	0.846846037	8142.61064	136	0	-36	0	0
16	RandomForestRegressor()	0.9	0.1	0.932838638	0.914981271	5745.04516	0	0	0	0	127.107208
17	DecisionTreeRegressor()	0.9	0.1	0.912802993	0.889618398	6546.12792	0	0	0	0	2.32536674
18	BaggingRegressor()	0.9	0.1	0.934902244	0.917593564	5656.09518	0	0	0	0	6.01999879
19	GradientBoostingRegressor(	0.9	0.1	-511.844736	-648.203746	502026.509	0	0	0	0	0.57002497
20	weight 1()	0.9	0.1	0.934950955	0.917655226	5653.97862	9	3	88	0	0
21	weight 2()	0.9	0.1	0.934950955	0.917655226	5653.97862	9	3	88	0	0
22	weight 3()	0.9	0.1	0.934935421	0.917635562	5654.65367	11	0	89	0	0
23	weight 4()	0.9	0.1	0.934935421	0.917635562	5654.65367	11	0	89	0	0
24	RandomForestRegressor()	0.95	0.05	0.93464586	0.887257599	4332.0244	0	0	0	0	137.427253
25	DecisionTreeRegressor()	0.95	0.05	0.929296746	0.878029845	4505.82212	0	0	0	0	2.35730624
26	BaggingRegressor()	0.95	0.05	0.91207907	0.848327638	5024.58826	0	0	0	0	6.46022725
27	GradientBoostingRegressor(	0.95	0.05	-233.010429	-402.691298	259222.055	0	0	0	0	0.59317327
28	weight 1()	0.95	0.05	0.941133129	0.898448785	4111.40081	57	43	0	0	0
29	weight 2()	0.95	0.05	0.946554845	0.907801785	3917.49554	143	39	-82	0	0
30	weight 3()	0.95	0.05	0.93464586	0.887257599	4332.0244	100	0	0	0	0
31	weight 4()	0.95	0.05	0.941124062	0.898433144	4111.71742	189	0	-89	0	0

The time taken for training follows the order GBR <DT < BTR <RFR

The general R squared value follows the trend:

weight4>= weight3>weight2>= weight1> RF>DT >BR >0>GBR

The order of priority for weights in the combined model seems to be

RF>DT >BTR > GBR=0

The R2 value range is -511.8447- 0.9466

The fact that Decision Tree give Better Results over Bagging Tree Regressor here might be due to inherent bias present in the dataset.

But Due to Other Reasons like Bad configurement GBR performs poorly.

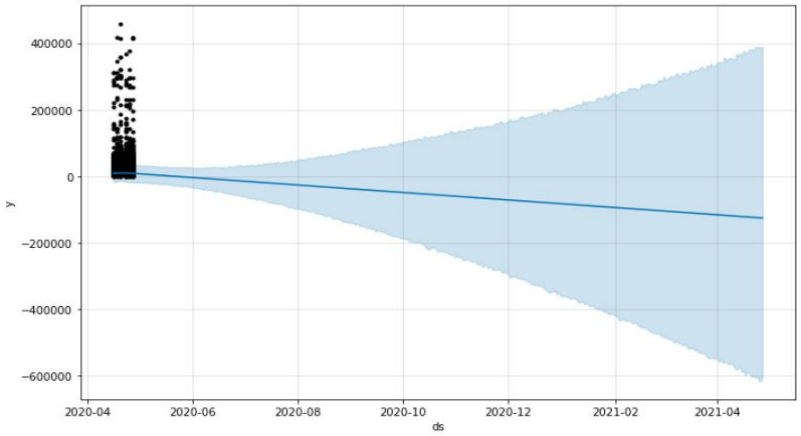


Figure 26 Prophet's – Ease-My-Trip Dataset

Based on Prophet's forecast due to even shorter period it is more likely for large increase in variance as time period increases.

8. SUMMARY

From the experiments conducted we see that:

On the bases of the 3 datasets used here, we can conclude that larger the range of datapoints the estimation would be more accurate and variance would be negligible.

The Random Forest Regressor is found to be an optimal regressor when predicting flight fare.

The more homogenous the data used the more likely it is to accurately predict flight flare.

The DB1B Dataset allowed for flight fare prediction at the market level accurately for over a year in advance but did not help in predicting the daily trends in flight fare.

The Machine Hack dataset and the Ease-My-Trip Dataset had shorter time periods; they came with accurate dates allowing for better predictions to be made on the daily levels.

The Ease-My-Trip Dataset came with an additional column ‘days left’ which accounted for the days left before flight which allowed for the daily variations in price to be accounted for.

The Observed trend is: The more homogenous the data and the more descriptive the features in the dataset are able to describe the flight scenario the better accurately the flight fare can be predicted.

To add to our learning curve, we plan to develop and execute codes for Catboost and other ML models along with the usage of ensemble technique to develop our own new ML model.

9. REFERENCES

1. Leo Breiman “Random Forests” 2001 Machine Learning, 45, 5-32, 2001 @ Kluwer Academic Publishers, Manufactured in The Netherlands
2. K. Tziridis, Th. Kalampokas, G. A. Papakostas and K. I. Diamantaras, "Airfare prices prediction using machine learning techniques," 2017 25th European Signal Processing Conference (EUSIPCO), Kos, 2017, pp. 1036-1039, doi: 10.23919/EUSIPCO.2017.8081365.
3. Viet Hoang Vu, Quang Tran Minh, Phu H Phung “An Airfare Prediction Model for Developing Markets” 2018 International Conference on Information Networking (ICOIN) 10.1109/ICOIN.2018.8343221
4. Supriya rajankar, Neha Sakharkar, Omprakash Rajankar “Predicting The Price Of A Flight Ticket With The Use Of machine Learning Algorithms” International Journal of Scientific & Technology Research Volume 8, Issue 12, December 2019, ISSN 2277-8616
5. Juhar Ahmed Abdella, Nazar Zaki, Khaled Shuaib, Fahad Khan, Airline ticket price and demand prediction: A survey, Journal of King Saud University - Computer and Information Sciences, 2019, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2019.02.001. (https://www.sciencedirect.com/science/article/pii/S131915781830884X)
6. Oren Etzioni and Rattapoom Tuchinda and Craig A. Knoblock and Alexander Yates “To buy or not to buy: mining airfare data to minimize ticket purchase price”, In Proceedings of KDD’03, 2003, ACM Press
7. Rian Mehta, Stephen Rice, John Deaton, Scott R. Winter, Creating a prediction model of passenger preference between low cost and legacy airlines, Transportation Research Interdisciplinary Perspectives, Volume 3, 2019, 100075, ISSN 2590-1982, https://doi.org/10.1016/j.trip.2019.100075. (https://www.sciencedirect.com/science/article/pii/S2590198219300740)
8. Li, Y., & Li, Z. (2018). Design and implementation of ticket price forecasting system. doi:10.1063/1.5039083
9. Wang, Tianyi & Pouyanfar, Samira & Tian, Haiman & Tao, Yudong & Alonso, Miguel & Luis, Steven & Chen, Shu-Ching. (2019). A Framework for Airfare Price Prediction: A Machine Learning Approach. 200-207. 10.1109/IRI.2019.00041.
10. T. Liu, J. Cao, Y. Tan and Q. Xiao, "ACER: An adaptive context-aware ensemble regression model for airfare price prediction," 2017 International Conference on Progress in Informatics and Computing (PIC), Nanjing, China, 2017, pp. 312-317, doi: 10.1109/PIC.2017.8359563.
11. PromptCloud (February 2021) EaseMyTrip Flight Fare Travel Listings, Version 1, Retrieved On May 2021 from https://www.kaggle.com/data/46091
12. U.S. Department of Transportation, Research and Innovative Technology Administration, Bureau of Transportation Statistics, Freight Transportation: Global Highlights, 2010 (Washington, DC: 2010)
13. Machine Hack (March 2019) Predict The Flight Ticket Price Hackathon, Retrieved on April 2021 from https://machinehack.com/hackathons/predict_the_flight_ticket_price_hackathon/data
14. https://www.oag.com/
15. https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788831307/1/ch01lvl1sec13/standard-ml-workflow
16. https://towardsdatascience.com/a-quick-and-dirty-guide-to-random-forest-regression

17. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor

18. https://airlab.fiu.edu/data-overview-the-origin-and-destination-survey-db1b

19. https://www.icao.int/Pages/default.aspx

10.Appendix A

# ItinID	# MktID	# MktCoupons	# Year	# Quarter	OriginAirportID	OriginAirportSeqID	OriginCityMark
20.2m 202b	2.02b 20203b	1 8	2018 2020	1 4	10.2k 15.6k	1.02m 1.56m	30.2k
20181768137.0	2018176813702.0	3	2018	1	12892	1289208	32575
20181768137.0	2018176813702.0	3	2018	1	12892	1289208	32575
20181768137.0	2018176813702.0	3	2018	1	12892	1289208	32575
20181768137.0	2018176813702.0	3	2018	1	12892	1289208	32575
20181768329.0	2018176832903.0	1	2018	1	14893	1489302	33192
20181768329.0	2018176832903.0	1	2018	1	14893	1489302	33192
20181768329.0	2018176832903.0	1	2018	1	14893	1489302	33192
20181774381.0	2018177438103.0	2	2018	1	15304	1530402	33195
20181774381.0	2018177438103.0	2	2018	1	15304	1530402	33195

Figure 27 DB1B Dataset Unprocessed

p	Source The source from where the data is being extracted	Layover1 The Layover place while travel	Layover2 The second layover place if any	Layover3 The 3rd layover place if there is any	Destination The destination of the flight	Flight Operator The name of the flight	Flight Number The number of the flight	Depa The dat from pla
1Apr20	Mumbai 10% Bengaluru 9% Other (24330) 81%	Delhi 32% Mumbai 18% Other (15098) 50%	[null] 59% Delhi 16% Other (7492) 25%	[null] 100% Kolkata 0% Other (22) 0%	Delhi 9% Chennai 7% Other (25221) 84%	Air India|Air India|... 25% Air India|Air India 17% Other (17580) 59%	21665 unique values	16Apr20
3:42	Delhi	Patna			Guwahati	SpiceJet|SpiceJet	SG-8751|SG-426	23Apr2
9:51	Mumbai	Bangalore	Delhi		Kochi	Vistara|Vistara|Vistara	UK-851|UK-812|UK-885	18Apr2
6:35	Ahmedabad	Delhi	Kolkata		Guwahati	Air India|Air India|Air India	AI-18|AI-401|AI-729	25Apr2
2:35	Kolkata	Delhi			Kochi	SpiceJet|SpiceJet	SG-254|SG-8561	16Apr2
2:13	Indore	Mumbai			Chennai	Indigo|Indigo	6E-5321|6E-323	18Apr2
2:33	Guwahati	Delhi	Bhopal		Raipur	Air India|Air India|Air India	AI-890|AI-437|AI-9683	23Apr2
3:33	Pune	Goa	Mumbai		Visakhapatnam	Air India|Air India|Air India	AI-9561|AI-34|AI-652	23Apr2

Figure 28 Ease-My-Trip Dataset Unprocessed

Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration
Jet Airways 36% IndiGo 19% Other (4781) 45%	18/05/2019 5% 6/06/2019 5% Other (9676) 91%	Delhi 42% Kolkata 27% Other (3275) 31%	Cochin 42% Banglore 27% Other (3275) 31%	DEL → BOM → COK 22% BLR → DEL 15% Other (6755) 63%	18:55 2% 17:00 2% Other (10223) 96%	19:00 4% 21:00 3% Other (9900) 93%	2h 50m 1h 30m Other (9747)
IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m
Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m
Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h
IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m
IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m
SpiceJet	24/06/2019	Kolkata	Banglore	CCU → BLR	09:00	11:25	2h 25m
Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	18:55	10:25 13 Mar	15h 30m
Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:00	05:05 02 Mar	21h 5m
Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:55	10:25 13 Mar	25h 30m
Multiple carriers	27/05/2019	Delhi	Cochin	DEL → BOM → COK	11:25	19:15	7h 50m
Air India	1/06/2019	Delhi	Cochin	DEL → BLR → COK	09:45	23:00	13h 15m

Figure 29 Machine Hack Dataset Unprocessed