PROFESSIONAL TRAINING REPORT

at

Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

K Pavan Chandra

Reg No:39110518

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119, TAMILNADU

April-2021

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956) JEPPIAAR

NAGAR, RAJIV GANDHI SALAI

CHENNAI– 600119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of K Pavan chandra (Reg No:39110518) who carried out the project entitled " Dynamic Pricing Prediction For Cabs Using Machine Learning" under my supervision from feb 2022 to apr 2022.

Internal Guide

Mr. A. Yovan Felix

Submitted for Viva voce Examination held on

_____

InternalExaminer ExternalExaminer

DECLARATION

I K Pavan Chandra hereby declare that the project report titled " Dynamic Pricing Prediction For Cabs Using Machine Learning" done by me under the guidance of Mr. A. Yovan Felix is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering.

DATE:

PLACE: SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board of Management of SATHYABAMA for their kind encouragement and support in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr. T. Sasikala M.E., Ph.D., Dean, School of Computing, Dr. S. Vigneshwari, M.E., Ph.D. and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Mr. A. Yovan Felix for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE

ABSTRACT:- Ride-on-demand (RoD) services are becoming more and more common, such as Uber and OLA cabs. To help both drivers and customers, RoD services use dynamic pricing to balance supply and demand in an attempt to increase service quality. Dynamic prices, however, often generate problems for passengers: often "unpredictable" prices prevent them from easily making fast decisions. In order to address this problem, it is therefore important to give passengers more detail, and forecasting dynamic prices is a feasible solution. Taking the Rapido dataset as an example in this paper, we focus on the estimation of dynamic prices, forecasting the price for each individual passenger order. Predicting prices will help passengers understand whether they could get a lower price in nearby locations or in a short period of time, thus alleviating their concerns. By learning the relationship between dynamic prices and features derived from the dataset, the prediction is carried out. As a representative, we train one linear model and test its output based on real service data from various perspectives. Furthermore, we view the

contribution of features based on the model at different levels and find out what features contribute most to dynamic prices. Finally, we predict dynamic prices using an efficient linear regression model based on evaluation results. Our hope is that the study helps to make passengers happier as an accurate forecast.

TABLE OF CONTENTS

Dynamic Pricing Prediction For Cabs Using Machine Learning

Introduction:

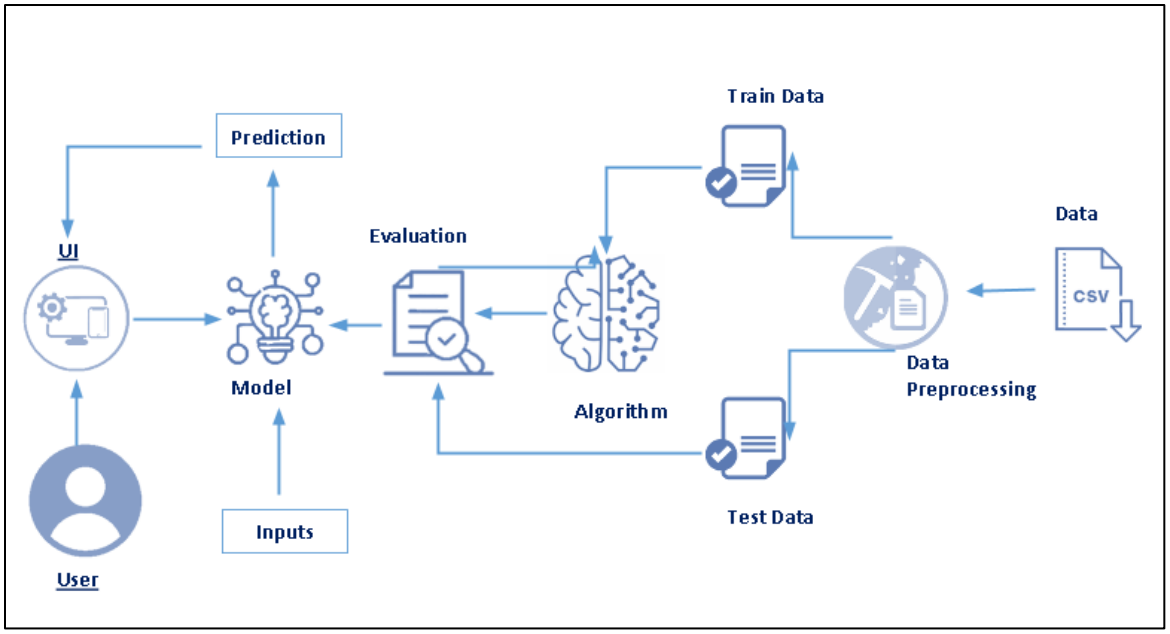
Many organizations do not have a direct role in travel and tourism but offer related products and services. Some examples would be offering travel insurance, parking facilities at airports, theatre and event tickets, car hire, and travel by rail or coach to airports, etc. at competitive rates. There are various different forms of dynamic pricing:
1. Peak Pricing – This is a strategy that is common in transportation businesses. Airlines are a good example. Airlines often charge a higher price to travel during rush hour mostly on weekdays and sometimes on weekends.
2. Surge Pricing – Companies such as Uber respond dynamically to changes in supply and demand in order to price their services differently. Like most of us have noticed, this frequently happens on stormy evenings and nights when more people request for cabs. Taxify also not so long ago introduced dynamic pricing to ensure the drivers are encouraged to go online and offer services when the demand is high.

Aim:-The primary Aim of the project is to Predict Dynamic increase of price of cabs using Machine Learning

Solution Requirement:

Every day the price of travel was changed due to the demand for public uses. The framework developed for the price prediction is analyzed for the travel plans. For the same travel plan offered at a fixed price for a particular group of customers, our proposed model saw a final fare with a lesser number of errors in predicting customer planning. As time progresses and more data are collected, the supervised learning will produce more accurate results and will be helpful in determining fare optimizer and dynamic availability of adjustments and continuously improve future recommendations.

Technical Architecture:



Pre-Requisites

In order to develop this project we need to install following softwares/packages:

Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video.

To make a responsive python script you must require the following packages

Requests: Allows you to send HTTP requests using Python.

Flask: Web framework used for building Web applications.

If you are using **anaconda navigator**, follow below steps to download required packages:

- Open the anaconda prompt.
- Type "pip install requests" and click enter.
- Type "pip install Flask" and click enter.

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Prior Knowledge : One should have knowledge on the following Concepts:

Requests: Allows you to send HTTP requests using Python.

Flask: Web framework used for building Web applications.

Project Objectives

Write what are all the technical aspects that students would get if they complete this project.

1. Knowledge on Machine Learning Algorithms.
2. Knowledge on Python Language with Machine Learning
3. Knowledge on Statistics and Graphs and their relations
4. You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
5. You will be able to know how to pre-process / clean the data using different data pre-processing techniques.
6. Applying different algorithms according to the dataset and based on visualization.
7. Real Time Analysis of Project
8. Building an ease of User Interface (UI)
9. Navigation of ideas towards other projects(creativeness)
10. Knowledge on building ML Model..
11. You will be able to know how to find the accuracy of the model.
12. How to Build web applications using the Flask framework.

Project Flow

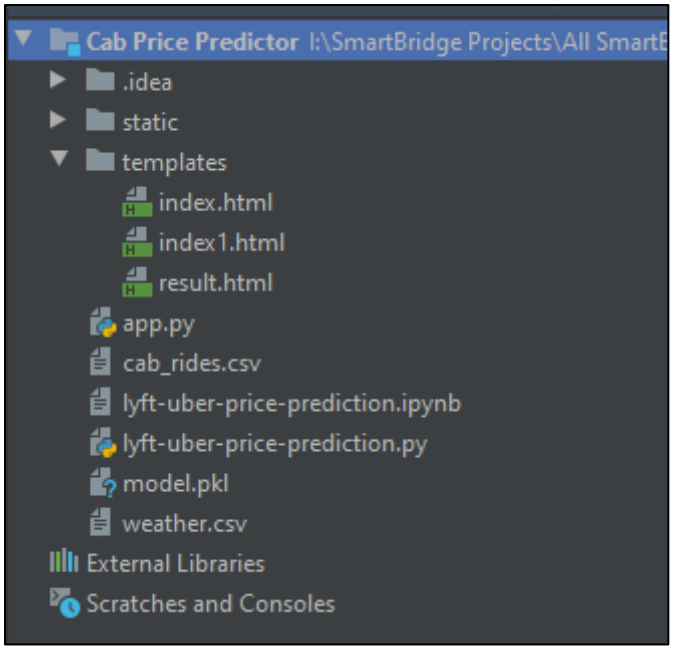
1. Install Required Libraries.
2. Data Collection.
    - Collect the dataset or Create the dataset
3. Data Pre- processing.
    - Import the Libraries.

- Importing the dataset.
- Understanding Data Type and Summary of features.
- Take care of missing data & create columns.
- Data Visualization.
- Drop the column from dataframe ,merge the dataframes.
- Observing Target,Numerical and Categorical Columns
- Label Encoding & Splitting the Dataset into Dependent and Independent variables
- Splitting Data into Train and Test.

4. Model Building
    - Training and testing the model
    - Evaluation of Model
    - Saving the Model
5. Application Building
    - Create an HTML file
    - Build a Python Code
6. Final UI
    - Dashboard Of the flask app.

Project Structure

Create a Project folder which contains files as shown below.



- We are building a Flask Application which needs HTML pages "**index.html**" ,"**index1.html**" ,"**result.html**" stored in the templates folder and a python script **app.py** for server side scripting
- The model is built in the notebook **lyft-uber-price-prediction.ipynb**

- We need the model which is saved and the saved model in this content is **model.pkl.**
- The static folder will contain a css file which is used in the html file.
- The templates mainly used here are “**index.html**”,”**index1.html**”, “**result.html**” for showcasing the UI
- The flask app is denoted as **app.py**

Install Required Packages.

- To complete the project, you need the following packages and libraries.
  1. Anaconda Navigator
  2. Jupyter
  3. Numpy
  4. Pandas
  5. Matplotlib
  6. Scikit-Learn

Install Anaconda software
To install Anaconda on your local system, go through the below links according to your system requirements. After Anaconda is installed, run the .exe folder.

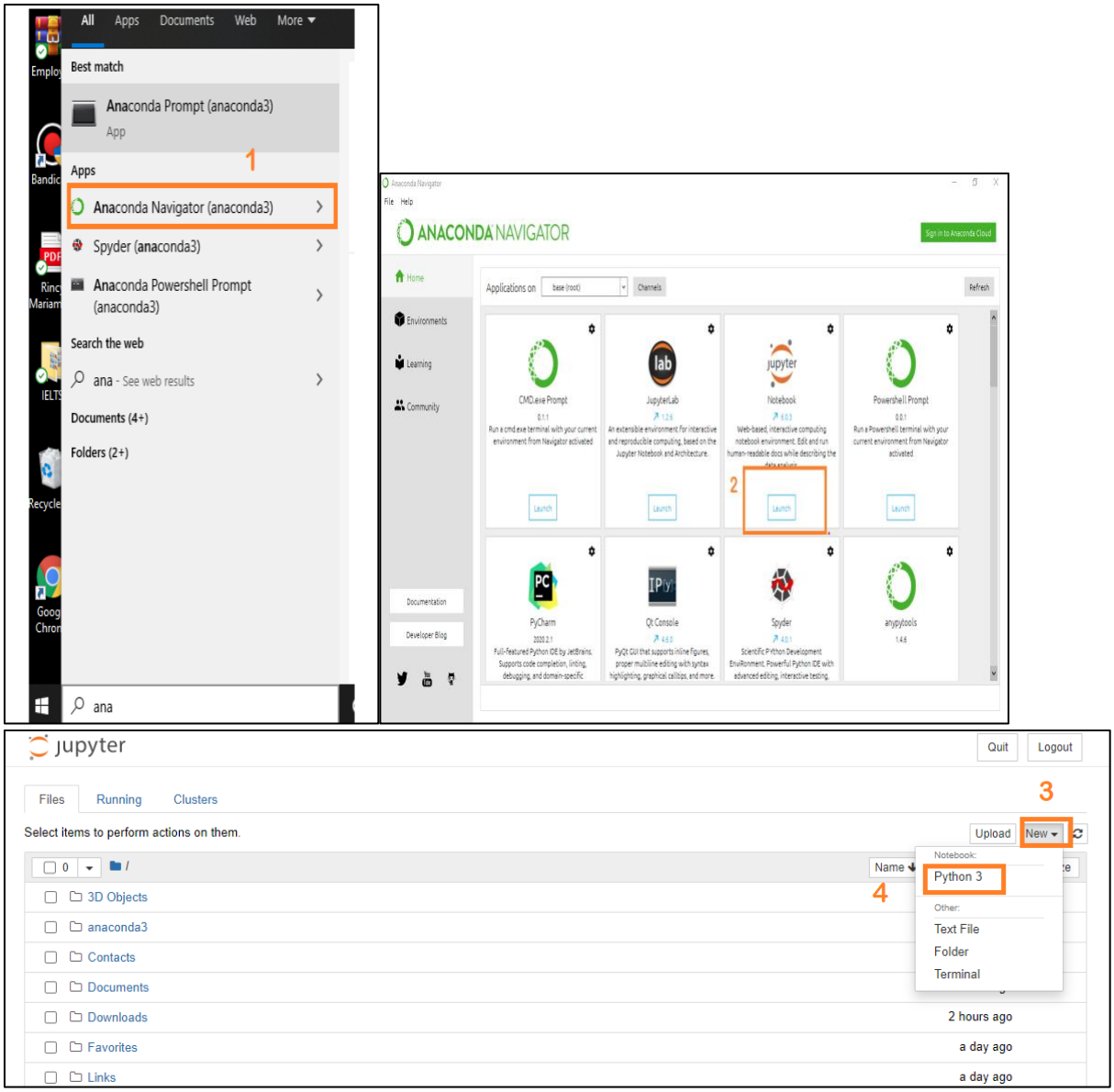
For Windows- Link

For MacOS- Link

For Linux- Link

Install Required Libraries.

Search Anaconda Navigator and open a Jupyter notebook.



Installation of Numpy Library :

Using Anaconda Navigator: conda install numpy

OR

Using command prompt: pip install numpy.

Installation of Pandas Library :

Using Anaconda Navigator: conda install pandas

OR

Using command prompt: pip install pandas.

Installation of Matplotlib Library :

Using Anaconda Navigator: conda install matplotlib

OR

Using command prompt: pip install matplotlib

Installation of Scikit-Learn Library :

Using Anaconda Navigator: conda install -c conda-forge scikit-learn

OR

Using command prompt: pip install -U scikit-learn

Dataset Collection

Collect The Dataset Or Create The Dataset

- ML depends heavily on data, without data, it is impossible for an “AI” to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

- There are many features which are responsible for Dynamic price prediction for travel, e.g. Cab type,Cab Name,Source ,Destination, Cab service type etc. For better prediction of the Dynamic price for travel, we should consider as many relevant features as possible.

- You can collect dataset from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.

- Here we are using a data set which you can find in the below link and you can download it from the link

Data Pre-Processing

Import The Libraries

- Import the required libraries for the model to run.

First step is usually importing the libraries that will be needed in the program.

```
#Importing Libraries

import numpy as np #linear algebra
import pandas as pd #data processing
import matplotlib.pyplot as plt
import seaborn as sns #used for data visualization
import pickle
from collections import Counter as c # return counts
from sklearn.preprocessing import LabelEncoder #importing the LabelEncoding from sklearn
from sklearn.model_selection import train_test_split #split data in train and test array
from sklearn.ensemble import RandomForestRegressor #regression ML algorithm
```

Numpy- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. Pandas objects are very much dependent on NumPy objects.
Pandas- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Counter: Python Counter is a container that will hold the count of each of the elements present in the container.

Matplotlib and Seaborn: Both are the data visualization library used for plotting graphs which will help us to understand the data.

Accuracy score: used in classification type problems and for finding accuracy.

Train_test_split: used for splitting data arrays into training data and for testing data.

Pickle: to serialize your machine learning algorithms and save the serialized format to a file.

Random Forest Regressor: random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Label Encoding: It is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Reading The Dataset

- Here, we are reading the dataset(.csv) from the system using pandas and storing it in a variable 'df'. It's time to begin building your text classifier! The data has been loaded into a DataFrame called df. The .head() method is particularly informative.
- You might have your data in .csv files, .excel files or .tsv files or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.

- Let's load a .csv data file into pandas. There is a function for it, called read_csv().We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- There are two dataset one is representing the data of Cab Rides and another one is representing for weather data.

Dataset

```
rides_df = pd.read_csv('cab_rides.csv')
weather_df = pd.read_csv('weather.csv')
```

- If the dataset is in the same directory of your program, you can directly read it.
- To check the top five records of the dataset,simply write the dataframe name.head().
- Rides_df represent the data for the cab rides like type of cab,location,name etc.

```
#Cab rides represent the for the type of cab,location,name etc.
rides_df.head()
```

	distance	cab_type	time_stamp	destination	source	price	surge_multiplier	id	product_id	name
0	0.44	Lyft	1544952607890	North Station	Haymarket Square	5.0	1.0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	lyft_line	Shared
1	0.44	Lyft	1543284023677	North Station	Haymarket Square	11.0	1.0	4bd23055-6827-41c6-b23b-3c491f24e74d	lyft_premier	Lux
2	0.44	Lyft	1543366822198	North Station	Haymarket Square	7.0	1.0	981a3613-77af-4620-a42a-0c0866077d1e	lyft	Lyft
3	0.44	Lyft	1543553582749	North Station	Haymarket Square	26.0	1.0	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	lyft_luxsuv	Lux Black XL
4	0.44	Lyft	1543463360223	North Station	Haymarket Square	9.0	1.0	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	lyft_plus	Lyft XL

- Weather_df represents data for the weather like temperature ,humidity etc

```
#Weather data represents the temperature ,humidity etc.
weather_df.head()
```

	temp	location	clouds	pressure	rain	time_stamp	humidity	wind
0	42.42	Back Bay	1.0	1012.14	0.1228	1545003901	0.77	11.25
1	42.43	Beacon Hill	1.0	1012.15	0.1846	1545003901	0.76	11.32
2	42.50	Boston University	1.0	1012.15	0.1089	1545003901	0.76	11.07
3	42.11	Fenway	1.0	1012.13	0.0969	1545003901	0.77	11.09
4	43.13	Financial District	1.0	1012.14	0.1786	1545003901	0.75	11.49

Understanding Data Type And Summary Of Features

- How the information is stored in a DataFrame or Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data : numeric and text data types.

- Numeric data types include integers and floats.

- Text data type is known as Strings in Python, or Objects in Pandas. Strings can contain numbers and / or characters.

- For example, a string might be a word, a sentence, or several sentences.
- Will see how our dataset is, by using info() method.
- info() method provides the summary of dataset.

```
rides_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   distance          693071 non-null  float64
 1   cab_type          693071 non-null  object
 2   time_stamp        693071 non-null  int64
 3   destination       693071 non-null  object
 4   source            693071 non-null  object
 5   price             637976 non-null  float64
 6   surge_multiplier  693071 non-null  float64
 7   id                693071 non-null  object
 8   product_id        693071 non-null  object
 9   name              693071 non-null  object
dtypes: float64(3), int64(1), object(6)
memory usage: 52.9+ MB
```

```
weather_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6276 entries, 0 to 6275
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   temp        6276 non-null   float64
 1   location    6276 non-null   object
 2   clouds      6276 non-null   float64
 3   pressure    6276 non-null   float64
 4   rain        894 non-null    float64
 5   time_stamp  6276 non-null   int64
 6   humidity    6276 non-null   float64
 7   wind        6276 non-null   float64
dtypes: float64(6), int64(1), object(1)
```

- As you can see in our dataset both numerical and categorical data are present, but it is not necessary that all the continuous data which we are seeing has to be continuous in nature. There may be a case that some categorical data is in the form of numbers but when we perform info() operation we will get numerical output. So, we need to take care of those types of data also.

Take Care Of Missing Data & Create Columns

- Sometimes you may find some data missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

- Word “True” that the particular column has missing values, we can also see the count of missing values in each column by using isna().sum function.

```
rides_df.isna().sum()
```

```
distance                0
cab_type                0
time_stamp              0
destination             0
source                  0
price                55095
surge_multiplier        0
id                      0
product_id              0
name                    0
dtype: int64
```

- The fillna() function is used to fill NA/NaN values using the specified method.

```
: weather_df.isna().sum()
```

```
: temp             0
  location         0
  clouds           0
  pressure         0
  rain          5382
  time_stamp       0
  humidity         0
  wind             0
  dtype: int64
```

```
: weather_df = weather_df.fillna(0)
```

- Converting the timestamp data into real date format & Creating the new column that contain the location and source.
- Using the datetime method to convert the time stamp into real date format

```
rides_df['date'] = pd.to_datetime(rides_df['time_stamp']/ 1000, unit = 's')
weather_df['date'] = pd.to_datetime(weather_df['time_stamp'], unit = 's')
```

```
rides_df['merged_date'] = rides_df['source'].astype('str') + ' - ' + rides_df['date'].dt.strftime('%Y-%m-%d').astype('str') + '
weather_df['merged_date'] = weather_df['location'].astype('str') + ' - ' + weather_df['date'].dt.strftime('%Y-%m-%d').astype('st
```

- The rides and weather data have been joined by merged_date column.
- Join the weather date on rides data

```
df_joined = rides_df.join(weather_df, on = ['merged_date'], rsuffix ='_w')
```

- Check the joined data info with the info() method.

```
df_joined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1167730 entries, 0 to 637975
Data columns (total 22 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   distance          1167730 non-null  float64
 1   cab_type          1167730 non-null  object
 2   time_stamp        1167730 non-null  int64
 3   destination       1167730 non-null  object
 4   source            1167730 non-null  object
 5   price             1167730 non-null  float64
 6   surge_multiplier  1167730 non-null  float64
 7   id                1167730 non-null  object
 8   product_id        1167730 non-null  object
 9   name              1167730 non-null  object
 10  date              1167730 non-null  datetime64[ns]
 11  merged_date       1167730 non-null  object
 12  temp              1164996 non-null  float64
 13  location          1164996 non-null  object
 14  clouds            1164996 non-null  float64
 15  pressure          1164996 non-null  float64
 16  rain              1164996 non-null  float64
 17  time_stamp_w      1164996 non-null  float64
 18  humidity          1164996 non-null  float64
 19  wind              1164996 non-null  float64
 20  date_w            1164996 non-null  datetime64[ns]
 21  merged_date_w     1164996 non-null  object
dtypes: datetime64[ns](2), float64(10), int64(1), object(9)
memory usage: 204.9+ MB
```

- As we can see from the joined data information, the rides data expanded from 693,070 records to 1,164,996. The data has bee expanded because the data has been joined by hour information, but the weather data has record more than one for same hour. I will show below.

```
df_joined['id'].value_counts()
```

```
865b44b9-4235-4e8e-b6fd-bc8373e95b63    15
849b0941-5ea2-497d-85a3-e5a3b1cf44f6    15
114775a3-877f-4439-bc0c-970130199d41    15
b4e2a936-5f8b-47fb-829f-5ebf2b4c9eff    15
82622350-1af5-481c-81f8-8994e694f745    15
                                        ..
339a7841-ca1d-408d-9e85-bcc830ac515c     1
c0e04bd2-7363-431c-9362-b40a605c1fce     1
4efcd2dc-af4e-4b43-aabd-a335ab7ba5c9     1
b8f71df6-9ef8-4a8a-9e6a-085505d81332     1
7253c99a-9092-4b96-bbe5-42125d4c8631     1
Name: id, Length: 637976, dtype: int64
```

```
df_joined[df_joined['id'] == '865b44b9-4235-4e8e-b6fd-bc8373e95b63'].iloc[:,10:22]
```

	date	merged_date	temp	location	clouds	pressure	rain	time_stamp_w	humidity	wind	date_w	merged_date_w
560855	2018-11-26 06:28:02.305999994	Financial District - 2018-11-26 - 6	40.51	Financial District	1.00	1014.18	0.0	1.543213e+09	0.92	1.28	2018-11-26 06:16:45	Financial District - 2018-11-26 - 6
560855	2018-11-26 06:28:02.305999994	Financial District - 2018-11-26 - 6	41.61	Financial District	0.98	1014.35	0.0	1.543215e+09	0.91	1.82	2018-11-26 06:49:02	Financial District - 2018-11-26 - 6
560855	2018-11-26 06:28:02.305999994	Financial District - 2018-11-26 - 6	40.50	Financial District	1.00	1014.18	0.0	1.543213e+09	0.92	1.27	2018-11-26 06:15:45	Financial District - 2018-11-26 - 6

- As you can see that there are 693,071 unique records. I chose the most repetitive one. All of the weather information are really similar, so I decided to make group by id.

```
id_group = pd.DataFrame(df_joined.groupby('id')['temp','clouds', 'pressure', 'rain', 'humidity', 'wind'].mean())
df_rides_weather = rides_df.join(id_group, on = ['id'])
```

```
C:\Users\Shivam\anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly con
verted to a tuple of keys) will be deprecated, use a list instead.
  """Entry point for launching an IPython kernel.
```

```
# Creating the columns for Month, Hour and Weekdays
df_rides_weather['Month'] = df_rides_weather['date'].dt.month
df_rides_weather['Hour'] = df_rides_weather['date'].dt.hour
df_rides_weather['Day'] =  df_rides_weather['date'].dt.strftime('%A')
```

- Creating the columns for Month, Hour and Weekdays

Data Visualization

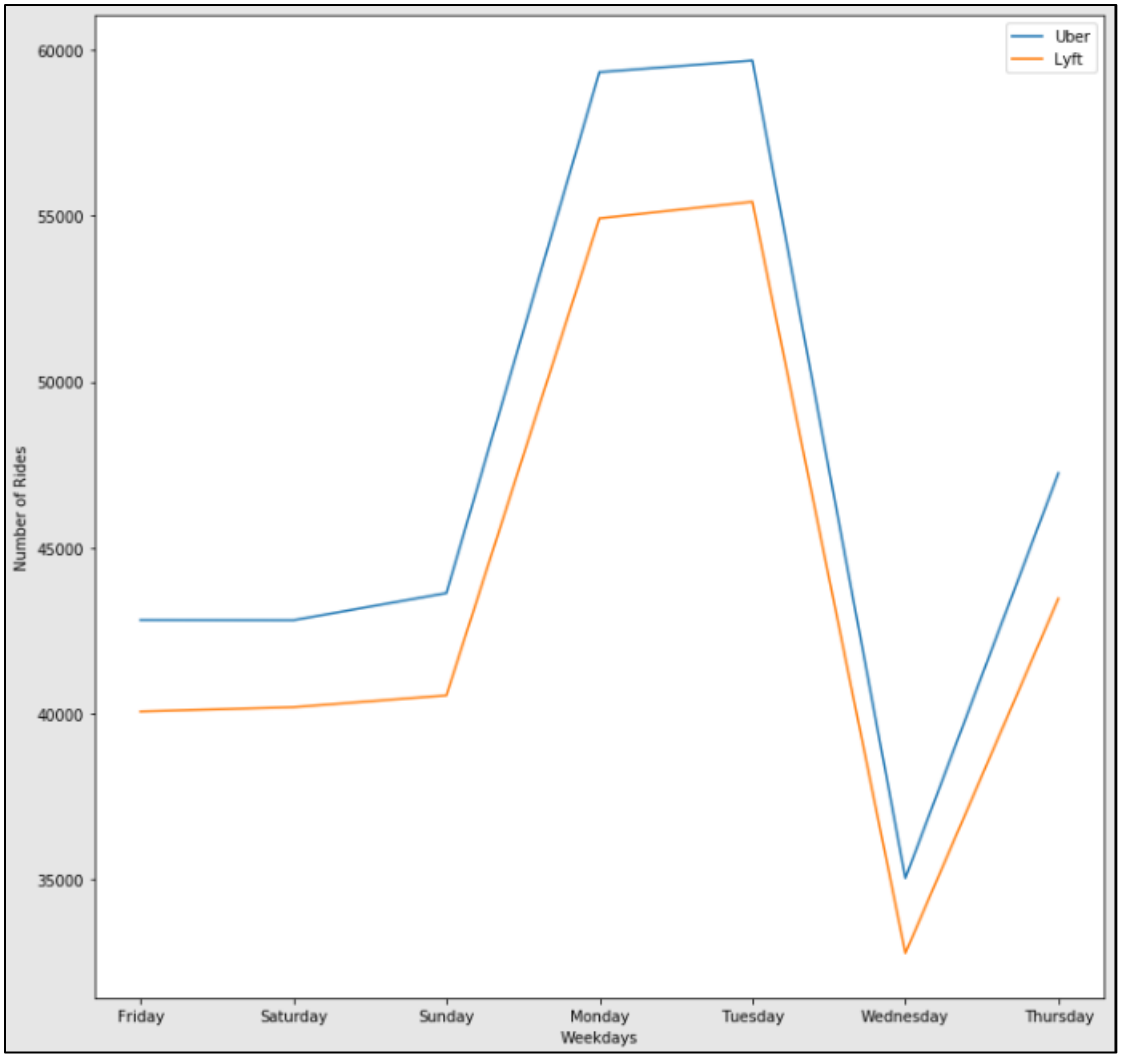
- Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends and correlations that might go undetected in text-based data.Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine learning model. In fact, even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.
- To visualize the dataset we need libraries called Matplotlib and Seaborn.The Matplotlib library is a Python 2D plotting library which allows you to generate plots, scatter plots, histograms, bar charts etc.
- Let's visualize our data using Matplotlib and the Seaborn library.

- The distribution of rides in weekdays

```
import matplotlib.pyplot as plt
uber_day_count = df_rides_weather[df_rides_weather['cab_type'] == 'Uber']['Day'].value_counts()
uber_day_count = uber_day_count.reindex(index = ['Friday','Saturday','Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday'])
lyft_day_count = df_rides_weather[df_rides_weather['cab_type'] == 'Lyft']['Day'].value_counts()
lyft_day_count = lyft_day_count.reindex(index = ['Friday','Saturday','Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday'])

fig , ax = plt.subplots(figsize = (12,12))
ax.plot(uber_day_count.index, uber_day_count, label = 'Uber')
ax.plot(lyft_day_count.index, lyft_day_count, label = 'Lyft')
ax.set(ylabel = 'Number of Rides', xlabel = 'Weekdays')
ax.legend()
plt.show()
```

- Plot the figure 'Number of rides vs Weekdays' (For Uber And Lyft Rides).



- The number of rides by weekday, the ride distribution in one day

```
fig , ax = plt.subplots(figsize= (12,12))
ax.plot(df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].groupby('Hour').Hour.count().index, df_rides_weather[df_rides_we
ax.plot(df_rides_weather[df_rides_weather['cab_type'] == 'Uber'].groupby('Hour').Hour.count().index, df_rides_weather[df_rides_we
ax.legend()
ax.set(xlabel = 'Hours', ylabel = 'Number of Rides')
plt.xticks(range(0,24,1))
plt.show()
```

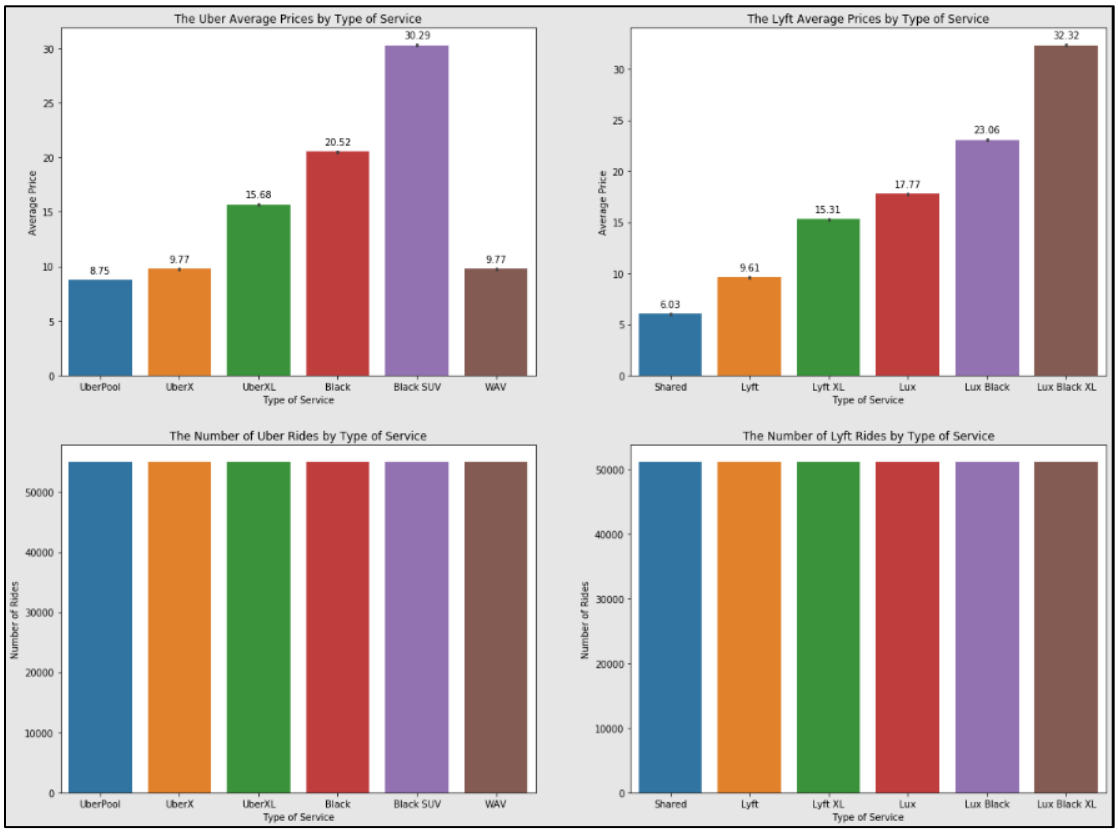
- Plot the figure 'Number of rides vs Hours (For Uber And Lyft Rides).

- The Average price of rides by type of service and number of rides by type of service.

```
uber_order =[ 'UberPool', 'UberX', 'UberXL', 'Black','Black SUV','WAV' ]
lyft_order = ['Shared', 'Lyft', 'Lyft XL', 'Lux', 'Lux Black', 'Lux Black XL']
fig, ax = plt.subplots(2,2, figsize = (20,15))
ax1 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] == 'Uber'].name, y = df_rides_weather[df_rides_weather['cab_t
ax2 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].name, y = df_rides_weather[df_rides_weather['cab_t
ax3 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] == 'Uber'].groupby('name').name.count().index, y = df_rides_w
ax4 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].groupby('name').name.count().index, y = df_rides_w
for p in ax1.patches:
    ax1.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', x
for p in ax2.patches:
    ax2.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', x
ax1.set(xlabel = 'Type of Service', ylabel = 'Average Price')
ax2.set(xlabel = 'Type of Service', ylabel = 'Average Price')
ax3.set(xlabel = 'Type of Service', ylabel = 'Number of Rides')
ax4.set(xlabel = 'Type of Service', ylabel = 'Number of Rides')
ax1.set_title('The Uber Average Prices by Type of Service')
ax2.set_title('The Lyft Average Prices by Type of Service')
ax3.set_title('The Number of Uber Rides by Type of Service')
ax4.set_title('The Number of Lyft Rides by Type of Service')
plt.show()
```

- Plot the figure for:-
    - The Uber Average Prices by Type of Service.
    - The Lyft Average Prices by Type of Service

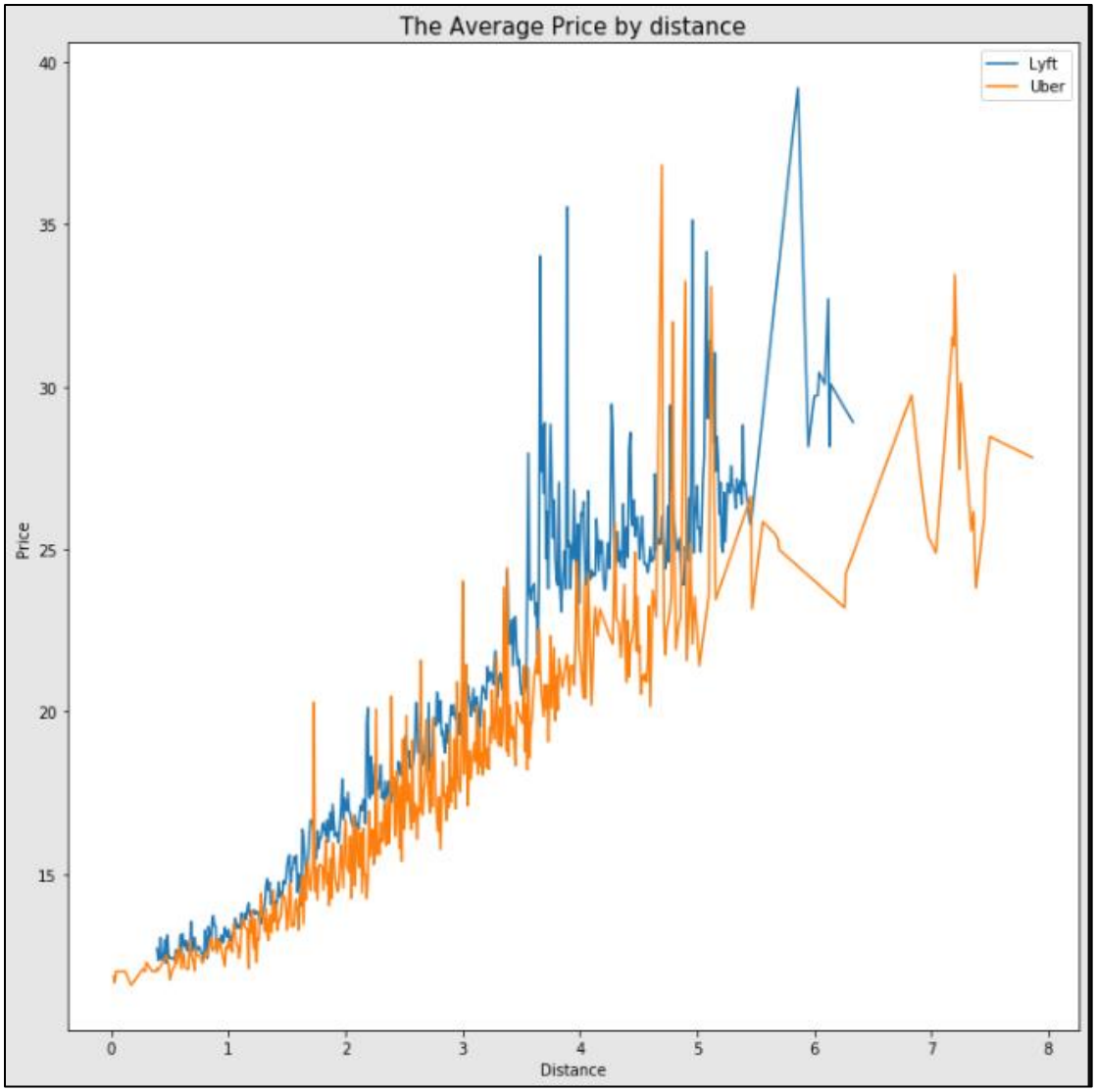
- The Number of Uber Rides by Type of Service
- The Number of Lyft Rides by Type of Service



- The first two charts in the first row show the comparison of prices by type of service for UBER and LYFT seperately.
- While UBER has extra service for the people with disabilities, LYFT hasn't this service.
- LYFT Shared prices is almost 30% lower that UberPool.
- The average price by distance

```
fig , ax = plt.subplots(figsize = (12,12))
ax.plot(df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].groupby('distance').price.mean().index, df_rides_weather[df_ride
ax.plot(df_rides_weather[df_rides_weather['cab_type'] == 'Uber'].groupby('distance').price.mean().index, df_rides_weather[df_ride
ax.set_title('The Average Price by distance', fontsize= 15)
ax.set(xlabel = 'Distance', ylabel = 'Price' )
ax.legend()
plt.show()
```

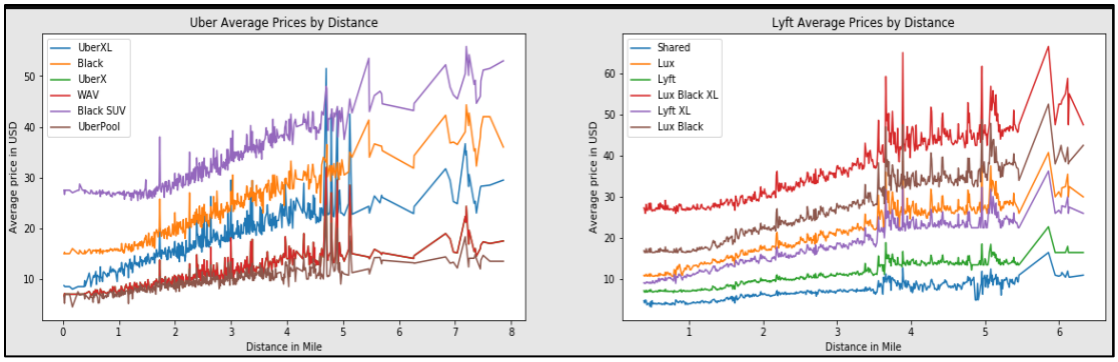
- Plot the figure 'Average price by distance (Price vs Distance For Uber And Lyft Rides).



- Average LYFT prices are slightly higher than UBER prices especially for higher distances.

```
fig, ax = plt.subplots(1,2 , figsize = (20,5))
for i,col in enumerate(df_rides_weather[df_rides_weather['cab_type'] == 'Uber']['name'].unique()):
    ax[0].plot(df_rides_weather[ df_rides_weather['name'] == col].groupby('distance').price.mean().index, df_rides_weather[ df_ri
ax[0].set_title('Uber Average Prices by Distance')
ax[0].set(xlabel = 'Distance in Mile', ylabel = 'Average price in USD')
ax[0].legend()
for i,col in enumerate(df_rides_weather[df_rides_weather['cab_type'] == 'Lyft']['name'].unique()):
    ax[1].plot(df_rides_weather[ df_rides_weather['name'] == col].groupby('distance').price.mean().index, df_rides_weather[ df_ri
ax[1].set(xlabel = 'Distance in Mile', ylabel = 'Average price in USD')
ax[1].set_title('Lyft Average Prices by Distance')
ax[1].legend()
plt.show()
```

- Plot the figure Uber Average Prices by Distance & Lyft Average Prices by Distance.



Drop The Column From Dataframe ,Merge The Dataframes

- Remove the column merged date from rides and weather DataFrame.

```
rides_df = rides_df.drop('merged_date', axis=1)
rides_df = rides_df.drop('date', axis=1)
```

```
weather_df = weather_df.drop('merged_date', axis=1)
weather_df = weather_df.drop('date', axis=1)
```

merged_date	temp	location	clouds	pressure	rain	time_stamp	humidity	wind
Back Bay - 2018-12-16 - 23	42.42	Back Bay	1.00	1012.14	0.1228	1545003901	0.77	11.25
Beacon Hill - 2018-12-16 - 23	42.43	Beacon Hill	1.00	1012.15	0.1846	1545003901	0.76	11.32
Boston University - 2018-12-16 - 23	42.50	Boston University	1.00	1012.15	0.1089	1545003901	0.76	11.07

- Rename the source weather dataframe column with new name.

```
source_weather_df = avg_weather_df.rename(
    columns={
        'location': 'source',
        'temp': 'source_temp',
        'clouds': 'source_clouds',
        'pressure': 'source_pressure',
        'rain': 'source_rain',
        'humidity': 'source_humidity',
        'wind': 'source_wind'
    }
)

source_weather_df
```

	source	source_temp	source_clouds	source_pressure	source_rain	source_humidity	source_wind
0	Back Bay	39.082122	0.678432	1008.447820	0.007925	0.764073	6.778528
1	Beacon Hill	39.047285	0.677801	1008.448356	0.008297	0.765048	6.810325
2	Boston University	39.047744	0.679235	1008.459254	0.007738	0.763786	6.692180
3	Fenway	38.964379	0.679866	1008.453289	0.007343	0.767266	6.711721
4	Financial District	39.410822	0.676730	1008.435793	0.008563	0.754837	6.860019

- Rename the destination weather dataframe column with new name.

```
destination_weather_df = avg_weather_df.rename(
    columns={
        'location': 'destination',
        'temp': 'destination_temp',
        'clouds': 'destination_clouds',
        'pressure': 'destination_pressure',
        'rain': 'destination_rain',
        'humidity': 'destination_humidity',
        'wind': 'destination_wind'
    }
)

destination_weather_df
```

	destination	destination_temp	destination_clouds	destination_pressure	destination_rain	destination_humidity	destination_wind
0	Back Bay	39.082122	0.678432	1008.447820	0.007925	0.764073	6.778528
1	Beacon Hill	39.047285	0.677801	1008.448356	0.008297	0.765048	6.810325
2	Boston University	39.047744	0.679235	1008.459254	0.007738	0.763786	6.692180
3	Fenway	38.964379	0.679866	1008.453289	0.007343	0.767266	6.711721
4	Financial District	39.410822	0.676730	1008.435793	0.008563	0.754837	6.860019

- Merge the rides data with the destination & source weather dataframe and store in data variable.

```
data = rides_df\
    .merge(source_weather_df, on='source')\
    .merge(destination_weather_df, on='destination')

data
```

	distance	cab_type	time_stamp	destination	source	price	surge_multiplier	id	product_id	name	...	source_pressure	source_rai
0	0.44	Lyft	1544952607890	North Station	Haymarket Square	5.0	1.0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	lyft_line	Shared	...	1008.445239	0.0086
1	0.44	Lyft	1543284023677	North Station	Haymarket Square	11.0	1.0	4bd23055-6827-41c6-b23b-3c491f24e74d	lyft_premier	Lux	...	1008.445239	0.0086
2	0.44	Lyft	1543366822198	North Station	Haymarket Square	7.0	1.0	981a3613-77af-4620-a42a-0c0866077d1e	lyft	Lyft	...	1008.445239	0.0086

Observing Target, Numerical And Categorical Columns

- Categorical Columns:
- The below code is used for fetching all the object or categorical type of columns from our data and we are storing it as **set** in a variable **cat**.

```
cat=data.dtypes[data.dtypes=='O'].index.values
cat
```

```
array(['cab_type', 'destination', 'source', 'id', 'product_id', 'name'],
      dtype=object)
```

- As you can observe, it gives us the same count of columns which we found previously.

```
for i in cat:
    print("Column :",i)
    print('count of classes : ',data[i].nunique())
    print(c(data[i]))
    print('*'*120)
```

- In the above we are looping with each categorical column and printing the classes of each categorical column using the counter function so that we can detect which columns are categorical and which are not.

- If you observe some columns have a few classes and some have many, but in some columns there are similar types of classes present so let's make that similar class into a single class using numpy .

Numerical Columns

```
data.dtypes[data.dtypes!='O'].index.values
```

```
array(['distance', 'time_stamp', 'price', 'surge_multiplier',
       'source_temp', 'source_clouds', 'source_pressure', 'source_rain',
       'source_humidity', 'source_wind', 'destination_temp',
       'destination_clouds', 'destination_pressure', 'destination_rain',
       'destination_humidity', 'destination_wind'], dtype=object)
```

Same as we did with categorical columns, we are making use of dtypes for finding the continuous columns.

Label Encoding

- Typically, any structured dataset includes multiple columns with combinations of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too.We need to convert each text category to numbers in order for the machine to process those using mathematical equations.
- How should we handle categorical variables? There are Multiple ways to handle it, but I will see one of them is Label Encoding.
- Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.
- Let's see how to implement label encoding in Python using the scikit-learn library.
- As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it.

```
data1=data.copy()
from sklearn.preprocessing import LabelEncoder #importing the LabelEncoding from sklearn
x='*'
for i in cat:#looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LE = LabelEncoder()#creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LE.fit_transform(data[i]) # trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print(x*100)
```

```
LABEL ENCODING OF: cab_type
Counter({'Uber': 330568, 'Lyft': 307408})
Counter({1: 330568, 0: 307408})
****************************************************************************************************
LABEL ENCODING OF: destination
Counter({'Financial District': 54192, 'Back Bay': 53190, 'Theatre District': 53189, 'Boston University': 53171, 'Haymarket Squa
re': 53171, 'Fenway': 53166, 'Northeastern University': 53165, 'North End': 53164, 'South Station': 53159, 'West End': 52992,
'Beacon Hill': 52840, 'North Station': 52577})
Counter({4: 54192, 0: 53190, 10: 53189, 2: 53171, 5: 53171, 3: 53166, 8: 53165, 6: 53164, 9: 53159, 11: 52992, 1: 52840, 7: 525
77})
****************************************************************************************************
LABEL ENCODING OF: source
Counter({'Financial District': 54197, 'Back Bay': 53201, 'Theatre District': 53201, 'Boston University': 53172, 'North End': 53
```

- In the above code we are looping through all the selected text class categorical columns and performing label encoding.
- If you see output of the above code, after performing label encoding alphabetical classes are converted to numeric.

Splitting the Dataset into Dependent and Independent variables.

- In machine learning, the concept of dependent variable (y) and independent variables(x) is important to understand. Here, Dependent variable is nothing but output in the dataset and the independent variable is all inputs in the dataset. We can denote with any symbol (alphabets). In our dataset we can say that class is the dependent variable and all other columns are independent. But in order to select the independent columns we will be selecting only those columns which are highly correlated and of some value to our dependent column.
- With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.
- Let's create out independent and dependent variables:

Creating independent and dependent variable

```
x = data.drop(['price','distance','time_stamp','surge_multiplier','id','source_temp','source_clouds','source_pressure','source_r
x=pd.DataFrame(x)
y = data['price'] #dependent feature
y=pd.DataFrame(y)
```

- In the above code we are creating a DataFrame of the independent variable x with our selected columns and for dependent variable y we are only taking the class column.
- Where DataFrame is used to represent a table of data with rows and columns.

Split The Dataset Into Train Set And Test Set

- When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset.

For this, you will have a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

- But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is,'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.
- The train-test split is a technique for evaluating the performance of a machine learning algorithm.
- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.
- In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.We will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.
- There are a few other parameters that we need to understand before we use the class:
- test_size — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.4 as the value, the dataset will be split 40% as the test dataset
- train_size — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
- random_state — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.
- Now split our dataset into a train set and test using train_test_split class from scikit learn library.

Model Building

Train And Test The Model Using Random Forest Regressor.

- There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms.

Example: 1. Linear Regression.

2. Logistic Regression.

3. Random Forest Regression / Classification.

4. Decision Tree Regression / Classification.

- You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate.
- Now we apply the Random forest regressor algorithm on our dataset.
- A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Build the model with the Random Forest Regressor.

- We're going to use x_train and y_train obtained above in train_test_split section to train our Random forest regression model. We're using the fit method and passing the parameters as shown below.

```
from sklearn.ensemble import RandomForestRegressor
rand=RandomForestRegressor(n_estimators=20,random_state=52,n_jobs=-1,max_depth=4)
rand.fit(x_train,y_train)
```

```
C:\Users\Shivam\anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataConversionWarning: A column-vector y
1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
RandomForestRegressor(max_depth=4, n_estimators=20, n_jobs=-1, random_state=52)
```

Predict the values

- Once the model is trained, it's ready to make predictions. We can use the predict method on the model and pass x_test as a parameter to get the output as y_pred.
- Notice that the prediction output is an array of real numbers corresponding to the input array.

Predicting the Result

```
ypred=rand.predict(x_test)
print(ypred)
```

```
[33.44544798 19.16381383  9.54753035 ...  6.02421004 26.79738243
 17.55244465]
```

Model Evaluation

- Finally, we need to check to see how well our model is performing on the test data. There are many evaluation techniques. For this, we evaluate the score produced by the model.

Score of the model

```
rand.score(x_train,y_train)
```

```
0.7575275520145969
```

Saving The Model

- Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.
- Save our model by importing pickle files.

Saving Our Model ¶

```
import pickle
pickle.dump(rand, open("model.pkl", "wb"))
```

Here, rand is our random forest regressor class with saving as **model.pkl** file. Wb is the write binary in bytes.

Build Flask Application

Flask Structure

To build this flask application you should have basic knowledge of “HTML, CSS, Bootstrap, flask framework and python”

- For more information regarding flask
- **Main Python Script**
- Let us build a flask file ’**lyft-uber-price-prediction.ipynb**’ which is a web framework written in python for server-side scripting. Let’s see step by step procedure for building the backend application.
- You can also run this on spyder, given as app.py
- App starts running when the “__name__” constructor is called in main.
- Render_template is used to return html files.
- “GET” method is used to take input from the user.
- “POST” method is used to display the output to the user.
- Three.html web pages are given.

Importing Libraries

Libraries required for the app to run are to be imported.

```
# importing the necessary dependencies
from flask import Flask,request,render_template
import numpy as np
import pandas as pd
import pickle
import os
```

Loading Flask And Assigning The Model Variable.

Loading Flask and assigning the model variable

```
app=Flask(__name__)# initializing a flask app


with open('model.pkl', 'rb') as handle:
    model = pickle.load(handle)
```

Routing To The Html Page

Basically we give routes of our html pages in order to showcase the UI. By giving the routes the built code in the html page is connected to our flask app. This is how a UI can be built and showcased.

```
@app.route('/')# route to display the home page
def home():
    return render_template('index.html') #rendering the home page
@app.route('/Prediction',methods=['POST','GET'])
def prediction(): # route which will take you to the prediction page
    return render_template('index1.html')
@app.route('/Home',methods=['POST','GET'])
def my_home():
    return render_template('index.html')
```

- We are routing the app to the html templates which we want to render.Firstly we are rendering the "**index.html**" template and from there we are navigating to our index1 page for the user input and then redirect onto the result or prediction page.

    - app – our flask application name
    - model – it will contain the model which we build
    - @app.route() – used for routing to the different pages.we have four route,
    - one for routing to the home page ('/')
    - route to the prediction page ('/Prediction')
    - route to the same home page itself('/Home')
    - routing to the result page ('/predict')
    - render_template () – it is used for render our html page from the templates folder
    - predict() – is taking the values form the prediction page and storing it into a variable and then we are creating a DataFrame along with the values and 4 independent features and finally we are predicting the values using or loaded model which we build and storing the output in a variable and returning it to the result page.
    - app.run(debug=False) – for running our app

```
@app.route('/predict',methods=["POST","GET"])# route to show the predictions in a web UI
def predict():
    #  reading the inputs given by the user
    input_feature=[float(x) for x in request.form.values() ]
    features_values=[np.array(input_feature)]
    feature_name=['cab_type', 'name','product_id','source','destination']
    x=pd.DataFrame(features_values,columns=feature_name)

     # predictions using the loaded model file
    prediction=model.predict(x)
    print("Prediction is:",prediction)
     # showing the prediction results in a UI
    return render_template("result.html",prediction=prediction[0])
```

Here the route for prediction is given and necessary steps are performed in order to get the predicted output.

```
if __name__=="__main__":
    port=int(os.environ.get('PORT',5000))
    app.run(port=port,debug=True,use_reloader=False)
```

Lastly, we run our app on the local host. Here we are running it on localhost:5000

Run The App In Local Browser

```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

I:\SmartBridge Projects\Cab predictor>python app.py run
```

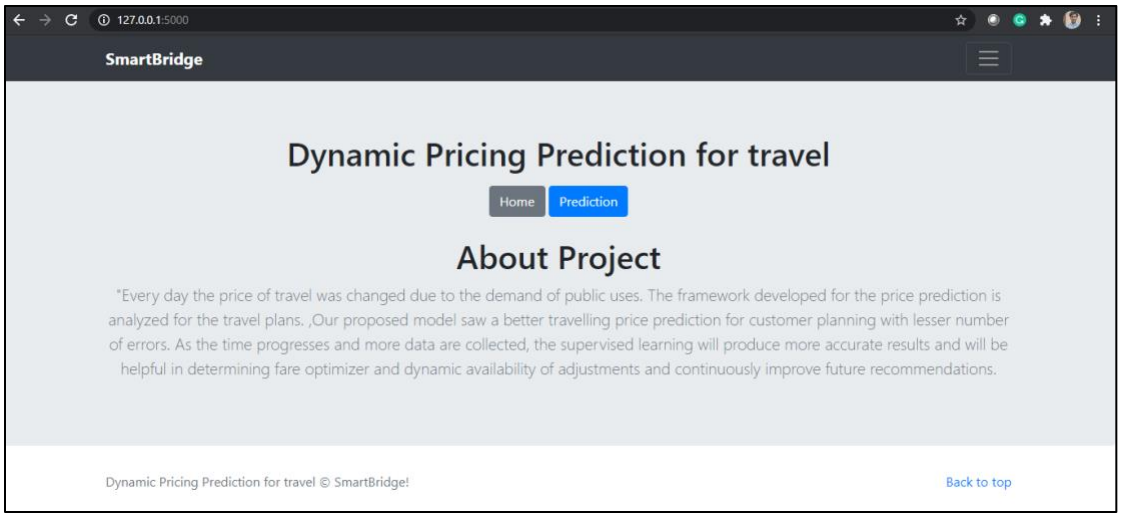
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page

```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

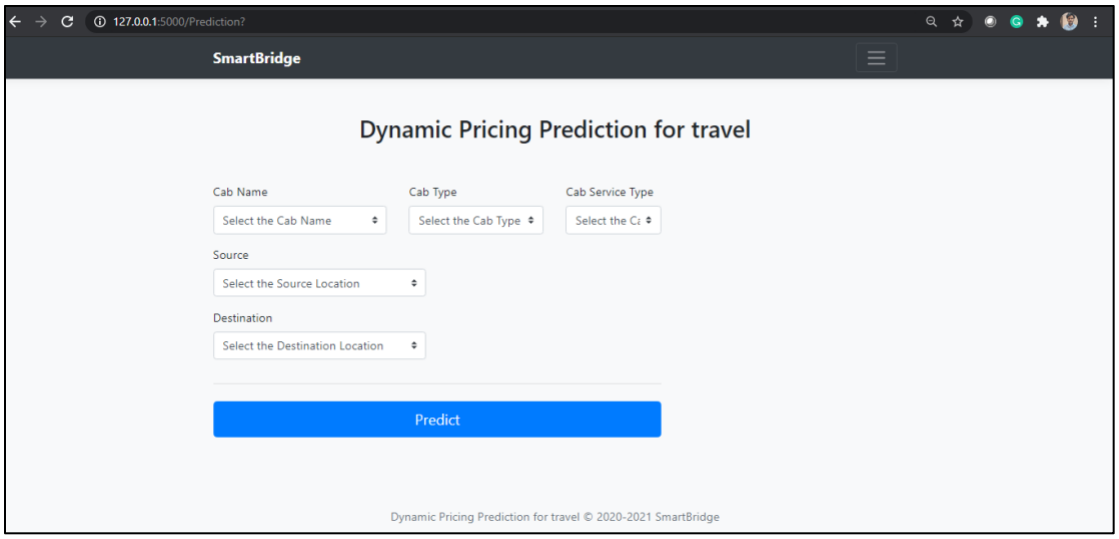
Final UI.

Dashboard of the flask application.

- This is the main page of Dynamic price predictor for travel .where you may know about the project and also from this page user can click onto the prediction button and they will redirect onto the prediction page for providing the inputs.



- The prediction page user gives the input for predicting the output where they can choose Cab Name ,Cab Type,Cab Service Type,Source & Destination then click to predict the output.



- In the predict page user will get the output based on the inputs they given in the prediction page.


127.0.0.1:5000/predict
SmartBridge
Final Fare Is :- "$ 19.16381383354415"