Cab Fare Prediction using Machine Learning to Transforming the travel Industry

1. INTRODUCTION

1.1 OVERVIEW

This project is about creating a Cab Fare Predictor, which can be used to predict the fare of a cab ride based on various factors like cab type, name, product id, source, and destination.

1.2 Purpose

The purpose of this project is to provide a tool to predict the cab fare, which can be helpful for people who frequently use cabs. It can also be useful for cab companies to estimate the cost of a ride in advance and make better pricing decisions.

2 LITERATURE SURVEY

2.1 Existing problem

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.

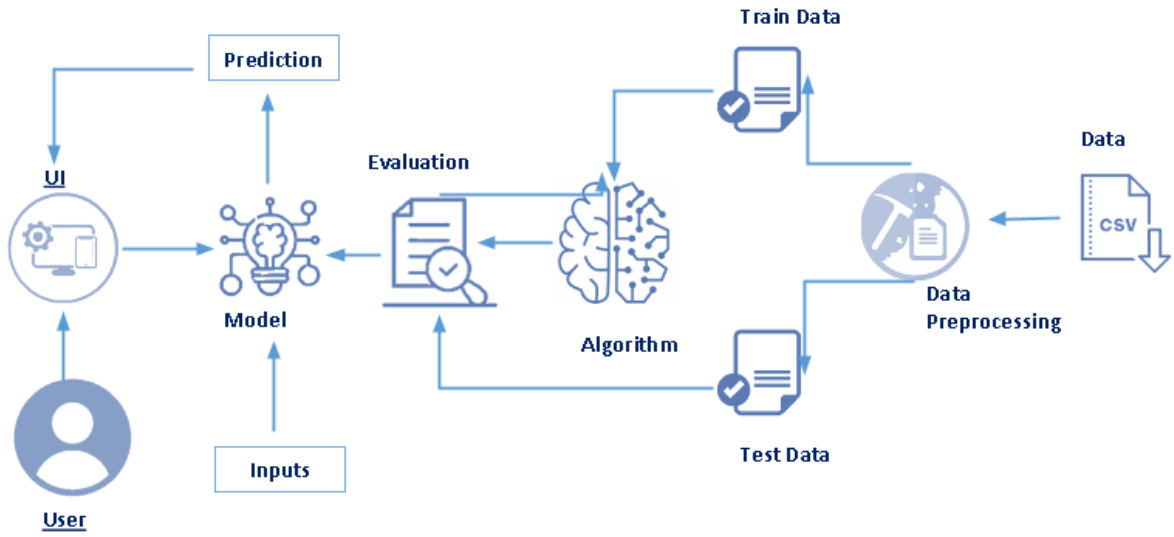
2. Surge Pricing – Companies such as Uber respond dynamically to changes in supply and demand in order to price their services differently. Like most of us have noticed, this frequently happens on stormy evenings and nights when more people request for cabs. Taxify also not so long ago introduced dynamic pricing to ensure the drivers are encouraged to go online and offer services when the demand is high.

2.2 Proposed solution

The proposed solution is to use a machine learning model to predict the cab fare. The model will be trained on historical data, taking into account various factors that could affect the fare. This will provide a more accurate estimate of the cab fare, and can be easily updated as new data becomes available.

3 THEORATICAL ANALYSIS

3.1 Block diagram



3.2 HARDWARE AND SOFTWARE DESIGNING

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It was created by Guido van Rossum , and first released on February 20, 1991. Its high-level built in data structures, combined with dynamic typing and dynamic binding , make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

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, crossplatform, 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.

Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

Spyder

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features. Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.Spyder is extensible with first-party and third party plugins includes support for interactive tools for data inspection and embeds Pythonspecific code. Spyder is also pre-installed in Anaconda Navigator, which is included in Anaconda.

Flask

Webframework used for building. It is a web application framework written in python which will be running in local browser with a user interface. In this application, whenever the user interacts with UI and selects emoji, it will suggest the best and top movies of that genre to the user.

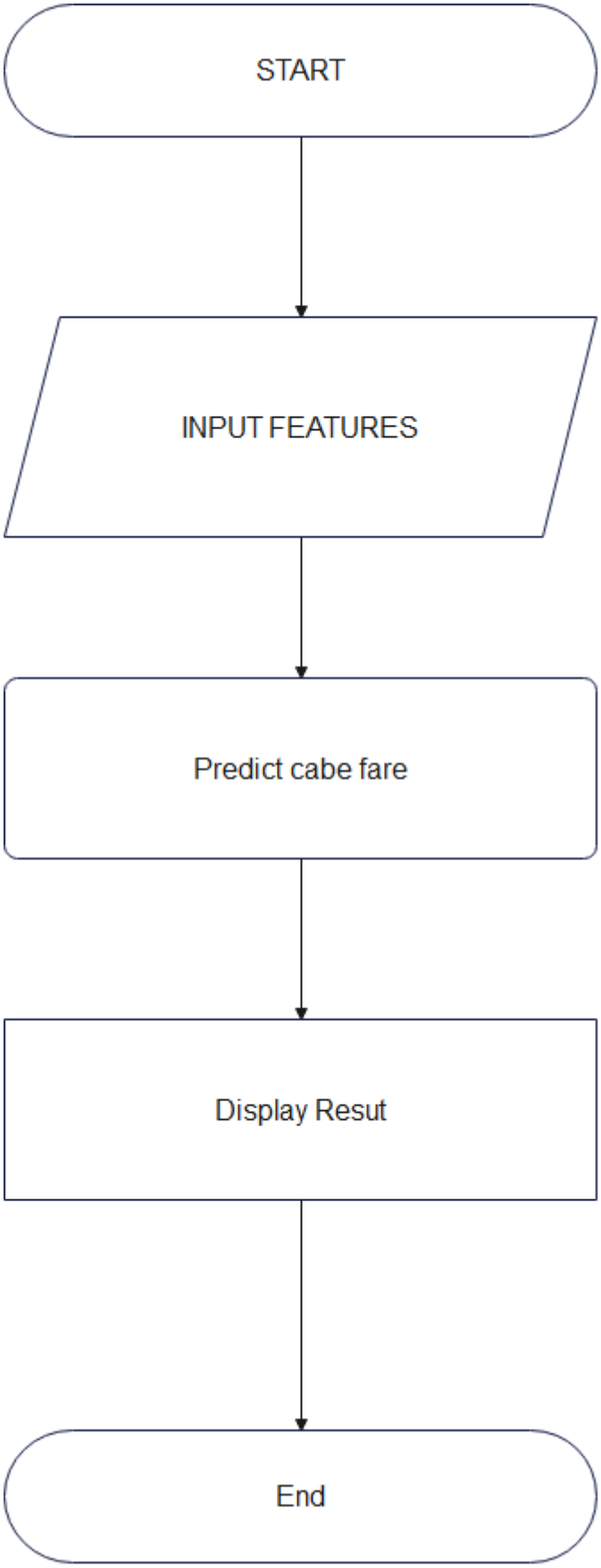
Hardware Requirements:

Operating system: window 7 and above with 64bit
Processor Type -Intel
Core i3-3220
RAM: 4Gb and above
Hard disk: min 100GB

4 EXPERIMENTAL INVESTIGATIONS

The experimental investigation involved collecting data on various cab rides and analysing the factors that affect the cab fare. This data was then used to train the machine learning model and test its accuracy. The text data need to be organized before proceeding with the project. The original dataset has a single folder. We will be using the cab_rides.csv and weather.csv files to fetch the text data of training data. The data need to be unique and all fields need to be filled. The dataset images are to be pre-processed before giving to the model. We will create a function that uses the pre-trained model for predicting custom outputs. Then we have to test and train the model. After the model is build, we will be integrating it to a web application

5. FLOWCHART



OUTPUT

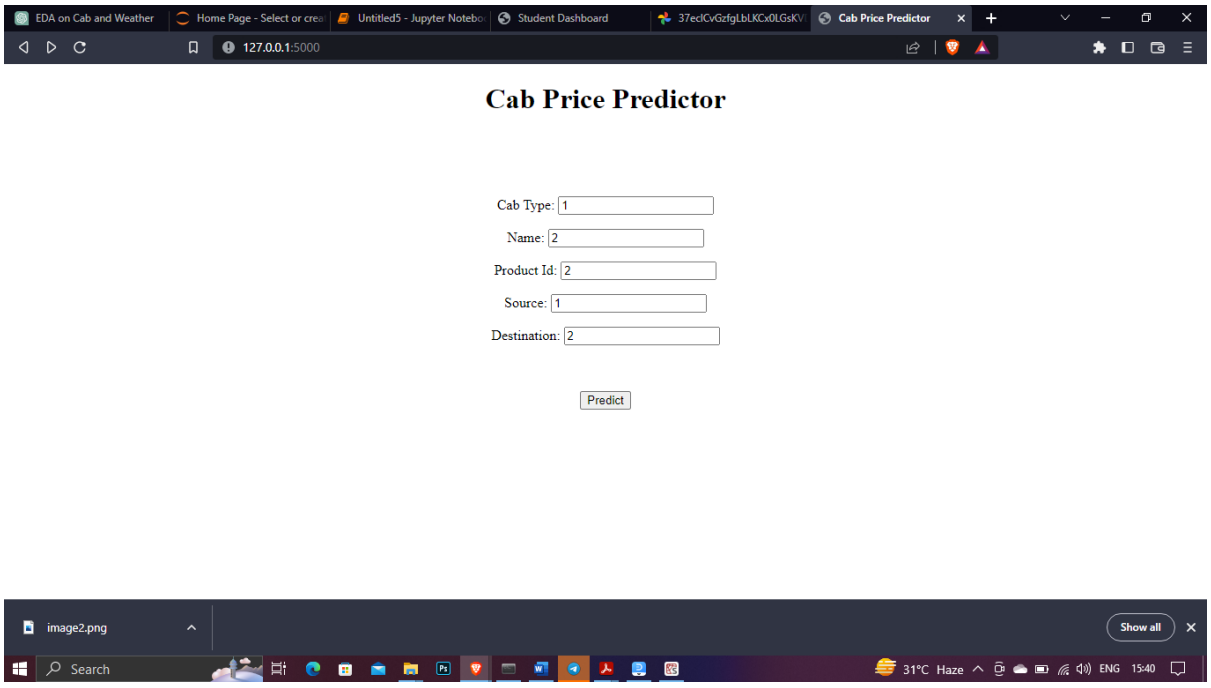

EDA on Cab and Weather
Home Page - Select or crea
Untitled5 - Jupyter Notebo
Student Dashboard
37eclCvGzfgLbLKCx0LGsKV
Cab Price Predictor
127.0.0.1:5000
Cab Price Predictor
Cab Type: 1
Name: 2
Product Id: 2
Source: 1
Destination: 2
Predict

image2.png
Show all
Search
31°C Haze
ENG
15:40

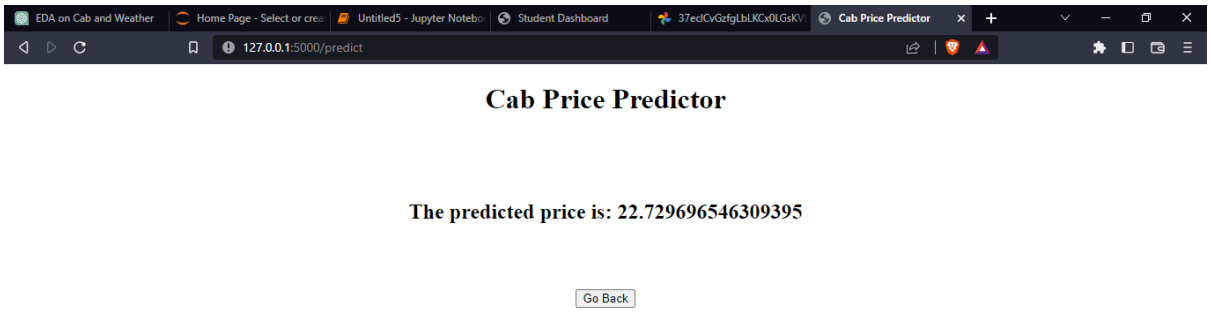

EDA on Cab and Weather
Home Page - Select or crea
Untitled5 - Jupyter Notebo
Student Dashboard
37eclCvGzfgLbLKCx0LGsKV
Cab Price Predictor
127.0.0.1:5000/predict
Cab Price Predictor
The predicted price is: 22.729696546309395
Go Back


image2.png
Show all
Search
31°C Haze
ENG
15:40

7 ADVANTAGES & DISADVANTAGES

Advantages:

- Accurate prediction of cab fare
- Takes into account various factors that could affect the fare
- Easy to update as new data becomes available

Disadvantages:

Limited to the data used for training the model
May not be accurate in all cases, especially if new data is significantly different from the training data.

8 APPLICATIONS

The cab fare predictor can be used by cab companies to estimate the cost of a ride in advance and make better pricing decisions. It can also be used by people who frequently use cabs to get a rough estimate of the cost of a ride.

9 CONCLUSIONS

In conclusion, the cab fare predictor is a useful tool that can provide an accurate estimate of the cost of a cab ride based on various factors. The machine learning model used in this project showed promising results, and with further improvements, it can be a valuable tool for both cab companies and customers.

10 FUTURE SCOPE

The Cab Fare Prediction using Machine Learning project has a lot of potential for future enhancements, including:

Incorporating more relevant factors into the prediction: The model can be improved by taking into account additional factors such as traffic conditions, weather, and road conditions, which can impact the fare of a cab ride.

Improving the accuracy of the model: The accuracy of the model can be improved by using more advanced machine learning techniques such as deep learning, reinforcement learning, or ensembling.

Real-time predictions: The model can be made real-time by incorporating real-time data such as traffic conditions and weather to provide even more accurate predictions.

Personalization: The model can be personalized to individual users by taking into account their preferred pickup and drop-off locations, travel patterns, and other factors.

Integration with cab booking platforms: The model can be integrated with cab booking platforms such as Uber and Ola to provide real-time predictions of fares for customers.

Exploration of alternative pricing models: The model can be used to explore alternative pricing models, such as dynamic pricing based on demand and supply, to provide even better pricing for customers.

Expansion to other cities: The model can be expanded to other cities to provide cab fare predictions for customers in those cities as well.

The future scope of the Cab Fare Prediction using Machine Learning project is vast and exciting, and there is a lot of potential for further development and improvement.

APPENDIX

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

rides_df = pd.read_csv('C:/Users/tsjis/Downloads/cab/cab_rides.csv')
weather_df = pd.read_csv('C:/Users/tsjis/Downloads/cab/weather.csv')

rides_df

rides_df.info()

weather_df

rides_df.isna().sum()

rides_df = rides_df.dropna(axis=0).reset_index(drop=True)

weather_df

weather_df.isna().sum()

weather_df = weather_df.fillna(0)
```

```
weather_df

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

#  df_rides['date'].dt.strftime('%m').head()
weather_df.index = weather_df['merged_date']

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

df_joined.info()

df_joined['id'].value_counts()

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

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'])

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')

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()

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

import seaborn as sns
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_type'] == 'Uber'].price , ax = ax[0,0], order =
uber_order)
ax2 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].name, y =
df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].price , ax = ax[0,1], order =
lyft_order)
ax3 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] ==
'Uber'].groupby('name').name.count().index, y =
df_rides_weather[df_rides_weather['cab_type'] == 'Uber'].groupby('name').name.count(),
ax = ax[1,0] ,order = uber_order)
ax4 = sns.barplot(x = df_rides_weather[df_rides_weather['cab_type'] ==
'Lyft'].groupby('name').name.count().index, y =
df_rides_weather[df_rides_weather['cab_type'] == 'Lyft'].groupby('name').name.count(), ax
= ax[1,1],order = lyft_order)
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', xytext = (0, 10), textcoords = 'offset points')
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', xytext = (0, 10), textcoords = 'offset points')
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()

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

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_rides_weather['name'] ==
col].groupby('distance').price.mean(), label = col)
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_rides_weather['name'] ==
col].groupby('distance').price.mean(), label = col)
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()

# the average rate per mile
df_rides_weather['rate_per_mile'] = round((df_rides_weather['price'] /
df_rides_weather['distance'] ),2)
# The average rate per mile plot
fig, ax = plt.subplots(1,2,figsize = (12,5))
```

```
ax1 = sns.lineplot(x = df_rides_weather.groupby(['distance'])['rate_per_mile'].mean().index,
y = df_rides_weather.groupby('distance')['rate_per_mile'].mean(), ax = ax[0])
ax2 = sns.lineplot(x = df_rides_weather.groupby(['distance'])['rate_per_mile'].mean().index,
y = df_rides_weather.groupby('distance')['rate_per_mile'].mean(), ax = ax[1])
plt.xticks(range(0, 10,1))
ax1.set(xlabel = 'Distance', ylabel = 'Rate per Mile in USD')
ax2.set(xlabel = 'Distance', ylabel = 'Rate per Mile in USD', ylim = (0,15))
ax1.set_title('The Average Rate per Mile', fontsize = 16)
ax2.set_title('ZOOM Average Rate per Mile', fontsize = 16)
plt.show()

rates_per_mile_pivot = df_rides_weather.pivot_table(index = ['cab_type', 'name', 'distance']
, values = ['rate_per_mile'])
rates_per_mile_pivot.reset_index(inplace = True)

fig, ax = plt.subplots(2,2, figsize = (20,8))
ax1 = sns.scatterplot(x = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['distance'], y = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['rate_per_mile'], hue = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['name'], ax = ax[0,0])
ax2 = sns.scatterplot(x = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['distance'], y = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['rate_per_mile'], hue = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Uber']['name'], ax = ax[1,0])
ax2.set( ylim = (0,20))
ax3 = sns.scatterplot(x = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['distance'], y = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['rate_per_mile'], hue = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['name'], ax = ax[0,1])
ax4 = sns.scatterplot(x = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['distance'], y = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['rate_per_mile'], hue = rates_per_mile_pivot[rates_per_mile_pivot['cab_type'] ==
'Lyft']['name'], ax = ax[1,1])
ax4.set( ylim = (0,20))
handles_uber, labels_uber = ax1.get_legend_handles_labels()
handles_uber =
[handles_uber[6],handles_uber[3],handles_uber[4],handles_uber[5],handles_uber[1],handl
es_uber[2]]
labels_uber =
[labels_uber[6],labels_uber[3],labels_uber[4],labels_uber[5],labels_uber[1],labels_uber[2]]
ax1.legend(handles_uber, labels_uber)
ax2.legend(handles_uber, labels_uber)
handles_lyft, labels_lyft = ax3.get_legend_handles_labels()
```

```
handles_lyft =
[handles_lyft[6],handles_lyft[4],handles_lyft[5],handles_lyft[1],handles_lyft[2],handles_lyft[
3]]
labels_lyft =
[labels_lyft[6],labels_lyft[4],labels_lyft[5],labels_lyft[1],labels_lyft[2],labels_lyft[3]]
ax3.legend(handles_lyft, labels_lyft)
ax4.legend(handles_lyft, labels_lyft)
ax1.set_title('Uber Rate per Mile')
ax1.set(ylabel = 'Rate per Mile in USD', xlabel = ' ')
ax2.set_title('Uber Rate Zoom(0 to 20 USD)')
ax2.set(ylabel = 'Rate per Mile in USD', xlabel = 'Distance')
ax3.set_title('Lyft Rate per Mile')
ax3.set(ylabel = ' ', xlabel = ' ')
ax4.set_title('Lyft Rate Zoom(0 to 20 USD)')
ax4.set(ylabel = ' ', xlabel = 'Distance')
plt.show()

high_mile_rates = df_rides_weather[df_rides_weather['rate_per_mile'] > 80]
# The number of overrated rides by cab type
high_mile_rates['cab_type'].value_counts()

high_mile_rates[high_mile_rates['cab_type'] == 'Lyft'].loc[:,['distance', 'cab_type', 'price',
'surge_multiplier','name', 'rate_per_mile']]

high_mile_rates[high_mile_rates['cab_type'] == 'Uber'].loc[:,['distance', 'cab_type', 'price',
'surge_multiplier','name', 'rate_per_mile']].sort_values(by = 'rate_per_mile', ascending =
False).head(20)

over_rated_pivot = high_mile_rates[high_mile_rates['cab_type'] ==
'Uber'].pivot_table(index = ['name', 'distance', 'price'], values = ['id'], aggfunc =
len).rename(columns = {'id' : 'count_rides'})
over_rated_pivot.reset_index(inplace =True)
over_rated_pivot.sort_values(by = ['count_rides', 'name'], ascending = False).head(15)

weather_df.groupby('location').mean()

avg_weather_df = weather_df.groupby('location').mean().reset_index(drop=False)
avg_weather_df = avg_weather_df.drop('time_stamp', axis=1)
avg_weather_df

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

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

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

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

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

data

data.name.unique()

data.source.unique()

item_counts = data["source"].value_counts()
item_counts

data.destination.unique()
```

```
item_counts = data["destination"].value_counts()
item_counts


data.product_id.unique()

item_counts = data["name"].value_counts()
item_counts

item_counts = data["product_id"].value_counts()
item_counts

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

from collections import Counter as c # return counts
for i in cat:
   print("Column :",i)
   print('count of classes : ',data[i].nunique())
   print(c(data[i]))
   print('*'*120)

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

data.isnull().any()#it will return true if any columns is having null values

data.isnull().sum()

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)

data.head()

data.info()
```

```
x =
data.drop(['price','distance','time_stamp','surge_multiplier','id','source_temp','source_cloud
s','source_pressure','source_rain','source_humidity','source_wind','destination_temp','desti
nation_clouds','destination_pressure','destination_rain','destination_humidity','destination
_wind'],axis=1) #independet features
x=pd.DataFrame(x)
y = data['price'] #dependent feature
y=pd.DataFrame(y)

x.head()

y.head()

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
print(x_train.shape)
print(x_test.shape)

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)

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

rand.score(x_train,y_train)


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