
Food
Delivery

Importing Libraries

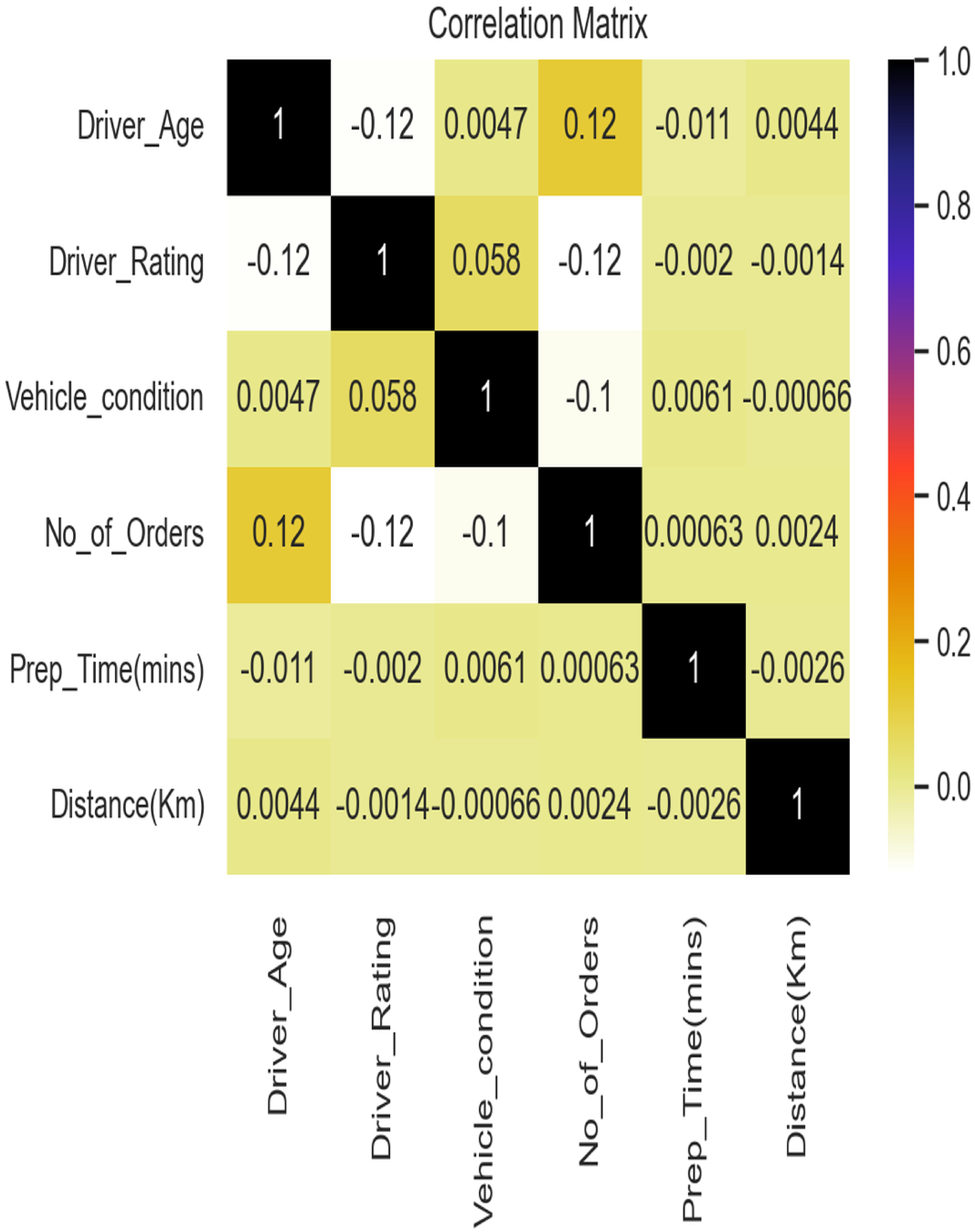
```
#Data Analysis
import pandas as pd
import numpy as np

#Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.offline import init_notebook_mode, iplot, plot
from plotly.subplots import make_subplots
import plotly.express as px
import plotly as py
init_notebook_mode(connected=True)
import plotly.graph_objs as go

#Feature selection and engineering
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn.decomposition import PCA

#Machine Learning
#import libraries for machine learning models
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
import xgboost as xg
from sklearn.metrics import r2_score, mean_squared_error

#Notebook Settings
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings('ignore')
```

Feature Engineering

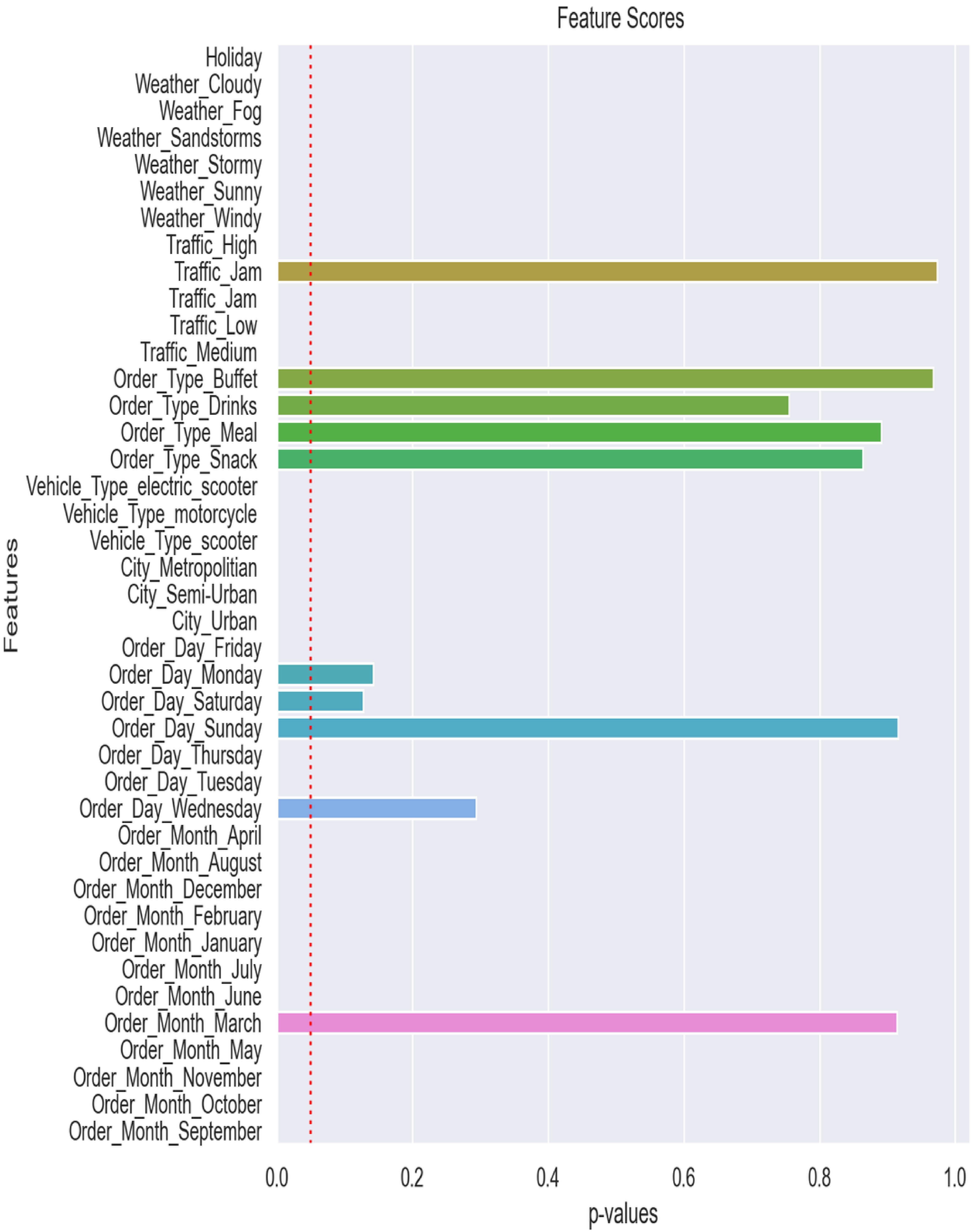
```
#converting datatypes to the appropriate format
delivery['Time_Orderd'] = pd.to_datetime(delivery['Time_Orderd'])
delivery['Time_Order_picked'] = pd.to_datetime(delivery['Time_Order_picked'])
delivery['Order_Date'] = pd.to_datetime(delivery['Order_Date'])

#calculating prep time
delivery['Preparation_Time'] = delivery['Time_Order_picked']  - delivery['Time_Orderd']

#extracting all components of the timedelta variable
Prep_Mins= pd.to_timedelta(delivery['Preparation_Time']).dt.components

#adding minutes components of Prep_Mins df to main delivery df as Prep_Tiime(mins)
delivery['Prep_Time(mins)']= Prep_Mins['minutes']


#extracting minutes compoment from the datetime column
delivery['Order_Day'] = delivery['Order_Date'].dt.day_name()
delivery['Order_Month'] = delivery['Order_Date'].dt.month_name()

#Calculating distance using Haversine Formula
# Set the earth's radius (in kilometers)
R = 6371

# Convert degrees to radians
def deg_to_rad(degrees):
    return degrees * (np.pi/180)

# Function to calculate the distance between two points using the haversine formula
def distcalculate(lat1, lon1, lat2, lon2):
    d_lat = deg_to_rad(lat2-lat1)
    d_lon = deg_to_rad(lon2-lon1)
    a = np.sin(d_lat/2)**2 + np.cos(deg_to_rad(lat1)) * np.cos(deg_to_rad(lat2)) * np.sin(d_lon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    return R * c

# Calculate the distance between each pair of points
delivery['Distance(Km)'] = np.nan

for i in range(len(delivery)):
    delivery.loc[i, 'Distance(Km)'] = distcalculate(delivery.loc[i, 'Restaurant_latitude'],
                                        delivery.loc[i, 'Restaurant_longitude'],
                                        delivery.loc[i, 'Delivery_location_latitude'],
                                        delivery.loc[i, 'Delivery_location_longitude'])
```

Feature Selection

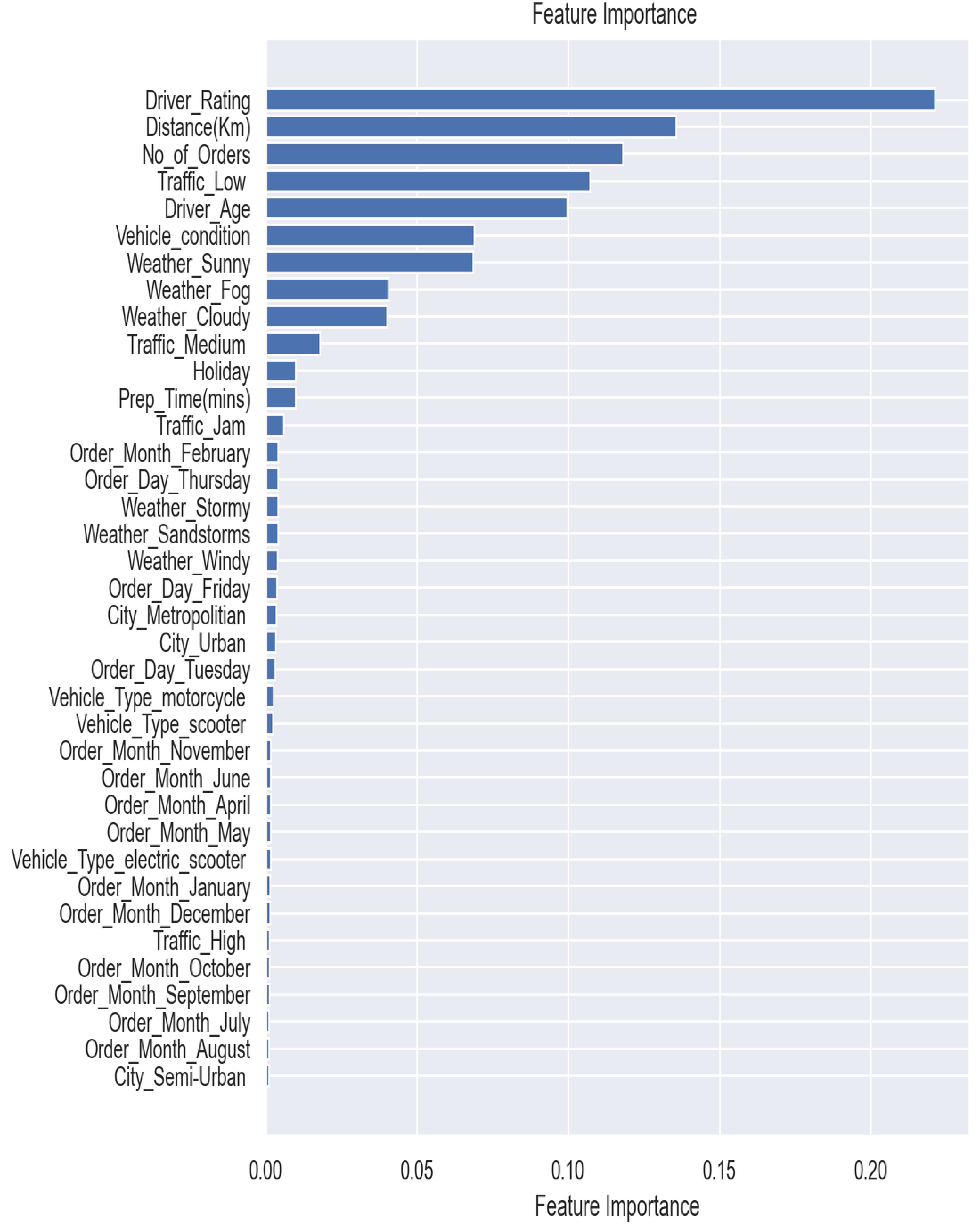
```python
#checking column names
delivery.columns

#drop columns that will not be used for prediction
delivery.drop(columns = ['Order_id', 'Driver_ID','Restaurant_latitude',
                         'Restaurant_longitude','Time_Orderd',
                         'Delivery_location_latitude',
                         'Delivery_location_longitude','Order_Date',
                         'Time_Order_picked', 'Preparation_Time'], axis = 1, inplace = True)


#Splitting dataset into numerical and categorical variables
#numerical variables
num_data = delivery.select_dtypes(include=[np.number])
num_data.head()

#categorical variables
cat_data = delivery.select_dtypes(exclude=[np.number])
cat_data.head()

#Feature selection on numerical variables using Correlation Matrix
# Target variable is Time_taken(min) is excluded
cor_num = num_data.loc[:, num_data.columns!='Time_taken(min)']
cor = cor_num.corr()

#visualizing correlation matrix
plt.figure(figsize=(6,4)) #increase heatmap size
sns.heatmap(cor, annot = True, cmap=plt.cm.CMRmap_r)

#Feature selection on categorical variables using p-values
#using label encoding for ordinal and binary categorical variables
ord_enc = OrdinalEncoder()
cat_data["Holiday"] = ord_enc.fit_transform(cat_data[["Holiday"]])

# one-hot encoding nominal categeorical variables get_dummies method
cat_data_encoded = pd.get_dummies(cat_data)
cat_data_encoded.head()
```



Correlation Matrix
Driver_Age
Driver_Rating
Vehicle_condition
No_of_Orders
Prep_Time(mins)
Distance(Km)
1 -0.12 0.0047 0.12 -0.011 0.0044
-0.12 1 0.058 -0.12 -0.002 -0.0014
0.0047 0.058 1 -0.1 0.0061 -0.00066
0.12 -0.12 -0.1 1 0.00063 0.0024
-0.011 -0.002 0.0061 0.00063 1 -0.0026
0.0044 -0.0014 -0.00066 0.0024 -0.0026 1
1.0
0.8
0.6
0.4
0.2
0.0



Feature Scores
Holiday
Weather_Cloudy
Weather_Fog
Weather_Sandstorms
Weather_Stormy
Weather_Sunny
Weather_Windy
Traffic_High
Traffic_Jam
Traffic_Jam
Traffic_Low
Traffic_Medium
Order_Type_Buffet
Order_Type_Drinks
Order_Type_Meal
Order_Type_Snack
Vehicle_Type_electric_scooter
Vehicle_Type_motorcycle
Vehicle_Type_scooter
City_Metropolitian
City_Semi-Urban
City_Urban
Order_Day_Friday
Order_Day_Monday
Order_Day_Saturday
Order_Day_Sunday
Order_Day_Thursday
Order_Day_Tuesday
Order_Day_Wednesday
Order_Month_April
Order_Month_August
Order_Month_December
Order_Month_February
Order_Month_January
Order_Month_July
Order_Month_June
Order_Month_March
Order_Month_May
Order_Month_November
Order_Month_October
Order_Month_September
Features
0.0
0.2
0.4
0.6
0.8
1.0
p-values

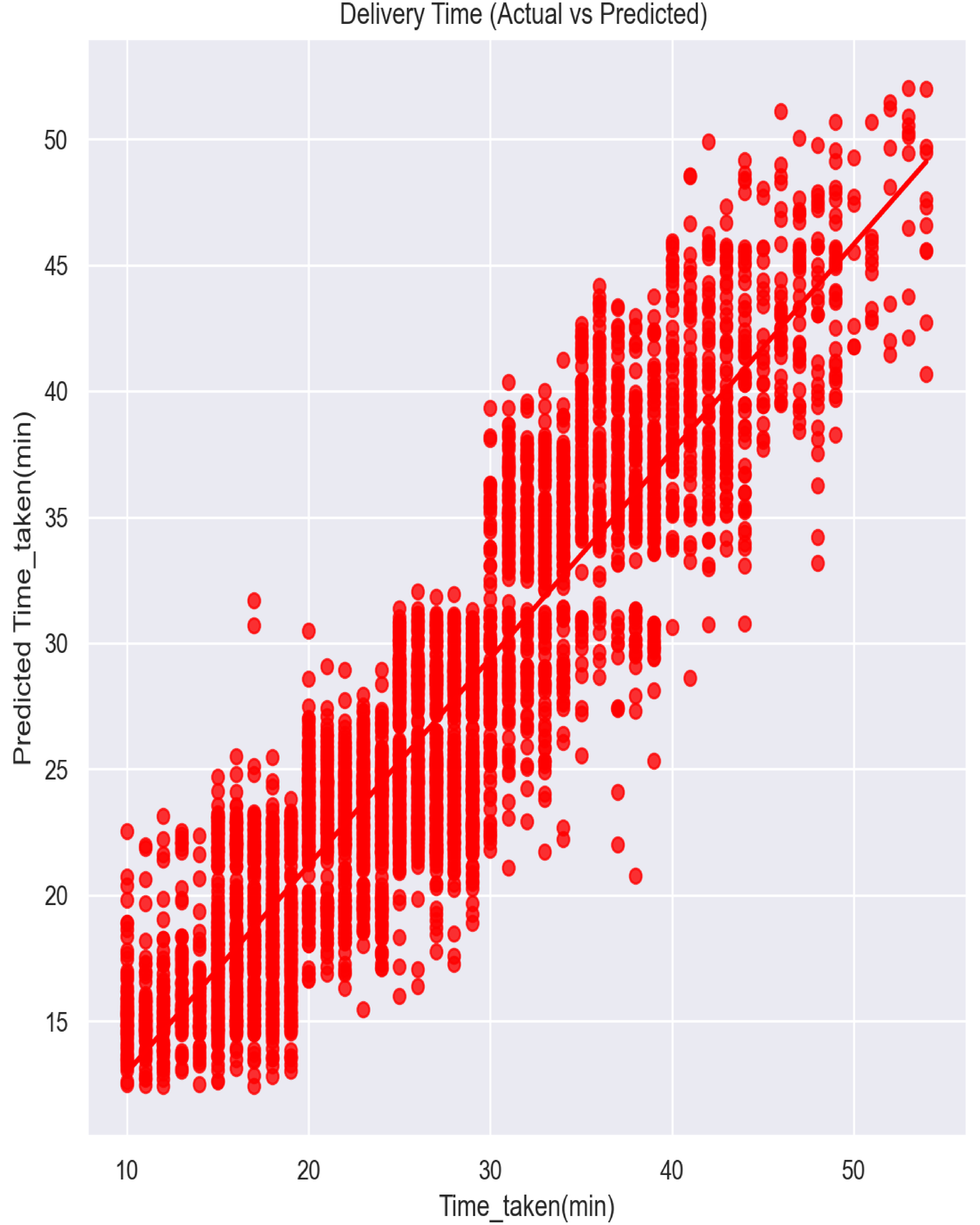

Feature Importance
Driver_Rating
Distance(Km)
No_of_Orders
Traffic_Low
Driver_Age
Vehicle_condition
Weather_Sunny
Weather_Fog
Weather_Cloudy
Traffic_Medium
Holiday
Prep_Time(mins)
Traffic_Jam
Order_Month_February
Order_Day_Thursday
Weather_Stormy
Weather_Sandstorms
Weather_Windy
Order_Day_Friday
City_Metropolitian
City_Urban
Order_Day_Tuesday
Vehicle_Type_motorcycle
Vehicle_Type_scooter
Order_Month_November
Order_Month_June
Order_Month_April
Order_Month_May
Vehicle_Type_electric_scooter
Order_Month_January
Order_Month_December
Traffic_High
Order_Month_October
Order_Month_September
Order_Month_July
Order_Month_August
City_Semi-Urban
0.00
0.05
0.10
0.15
0.20
Feature Importance

Computing ML Algorithm scores

```
#Linear Regression - Fitting, Predicing, Accuracy Scores
linreg = LinearRegression() #call the model
linreg.fit(X_train,y_train) #fit the model
y_pred_linreg = linreg.predict(X_test) #use model for prediction
linreg_score = round(metrics.r2_score(y_test, y_pred_linreg) * 100, 2) #calculate R2
linreg_rmse = mean_squared_error(y_test, y_pred_linreg) #calculate RMSE


#Ridge Regression - Fitting, Predicing, Accuracy Scores
ridgereg = Ridge() ridgereg.fit(X_train,y_train)
y_pred_ridgereg = ridgereg.predict(X_test)
ridgereg_score = round(metrics.r2_score(y_test, y_pred_ridgereg) * 100, 2)
ridgereg_rmse = mean_squared_error(y_test, y_pred_ridgereg)

#Lasso Regression - Fitting, Predicing, Accuracy Scores
lassoreg = Lasso() lassoreg.fit(X_train, y_train)
y_pred_lassoreg = lassoreg.predict(X_test)
lassoreg_score = round(metrics.r2_score(y_test, y_pred_lassoreg) * 100, 2)
lassoreg_rmse = mean_squared_error(y_test, y_pred_lassoreg)

#XGBRegressor - Fitting, Predicing, Accuracy Scores
XGBR = xg.XGBRegressor() XGBR.fit(X_train, y_train)
y_pred_XGBR = XGBR.predict(X_test)
XGBR_score = round(metrics.r2_score(y_test, y_pred_XGBR) * 100, 2)
XGBR_rmse = mean_squared_error(y_test, y_pred_XGBR)

#HistGradientBoostingRegressor - Fitting, Predicing, Accuracy Scores
HGBR = HistGradientBoostingRegressor() HGBR.fit(X_train, y_train)
y_pred_HGBR = HGBR.predict(X_test)
HGBR_score = round(metrics.r2_score(y_test, y_pred_HGBR) * 100, 2)
HGBR_rmse = mean_squared_error(y_test, y_pred_HGBR)

#Random Forest - Fitting, Predicing, Accuracy Scores
rf = RandomForestRegressor(n_estimators=150) rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rf_score = round(metrics.r2_score(y_test, y_pred_rf) * 100, 2)
rf_rmse = mean_squared_error(y_test, y_pred_rf)

# Creating dataframe to display results of all models
Model_Comparison = pd.DataFrame({
    'Model': ['Linear Regression', 'Ridge Regression', 'Lasso Regression','XGBR',
              'HGBR', 'Random Forest'],
    'R2_Score': [linreg_score, ridgereg_score, lassoreg_score,XGBR_score,
              HGBR_score, rf_score],
    'RMSE': [linreg_rmse, ridgereg_rmse, lassoreg_rmse,XGBR_rmse,
              HGBR_rmse, rf_rmse]})
```

Model Evaluation

```
# create regressor object
regressor = HistGradientBoostingRegressor().fit(X, y)
regressor.score(X,y)

# fit the regressor with x and y data
regressor.fit(X, y)

#Evaluating predictive capacity of the model using pre-scaled DF
data = delivery2.iloc[[18049]]

#Spliting data into features set and target set
X_data = data.drop(columns = 'Time_taken(min)', axis = 1)
y_data = data['Time_taken(min)']

print(X_data)
print(y_data)

# changing input_data to a numpy array
X_data_numpy_array = np.asarray(X_data)
y_data_numpy_array = np.asarray(y_data)


# reshape the array
X_data_reshaped = X_data_numpy_array.reshape(1,-1)
y_data_reshaped = y_data_numpy_array.reshape(1,-1)

prediction = regressor.predict(X_data_reshaped)
actual_time = y_data_reshaped

print('Actual Delivery mins =', actual_time[0])
print('Predicted1 Delivery mins =', round(prediction[0],2))

Actual Delivery mins = 25
Predicted1 Delivery mins = 24.18
```


Delivery Time (Actual vs Predicted)
Predicted Time_taken(min)
Time_taken(min)
10
20
30
40
50
15
25
35
45