MINI PROJECT REPORT

FOOD DELIVERY TIME PREDICTION

Abstract

This program focuses on predicting delivery time based on various features using different regression models. The code begins with data reading, exploration, and visualization using libraries like Pandas, NumPy, Matplotlib, Seaborn, and Plotly express. It includes data cleaning steps, creating distance features, and one-hot encoding categorical variables. The modeling section involves Linear regression, Decision tree regression, K-nearest neighbors regression, Lasso, and Ridge regression. The models are trained, and predictions are evaluated using metrics like mean squared error, mean absolute error, and R-squared score. Visualizations such as scatter plots and box plots are used to compare predicted and actual values.
The program concludes with an example of predicting delivery time for new data using the trained models. Standard scaling is applied to the input features, and the predicted delivery times for each model are displayed. Overall, the code provides a comprehensive analysis of delivery time prediction using different regression techniques.

Introduction

The modern era has witnessed a significant transformation in consumer behavior, with the advent of on-demand food delivery services. As the food delivery industry continues to thrive, optimizing the delivery process becomes paramount for service providers to meet customer expectations. The timely delivery of orders plays a crucial role in customer satisfaction and loyalty. This project focuses on predicting food delivery times, leveraging various regression models, to enhance operational efficiency in the food delivery domain.

Objective

The primary objective of this project is to develop a predictive model for food delivery time based on relevant features such as the age and ratings of delivery personnel, geographical coordinates of the restaurant and delivery location, and the type of order and vehicle used. By employing machine learning regression algorithms, we aim to create models that can forecast delivery times, allowing food delivery platforms to optimize their logistics and provide customers with more accurate delivery estimates.

Scope of the Project

The scope of this project encompasses data analysis, visualization, and the application of regression models to predict food delivery times. The dataset used includes information on delivery personnel, restaurant and delivery location coordinates, and various order and vehicle types. By exploring the relationships between these variables, we aim to build models that can generalize well to unseen data, providing valuable insights for the food delivery industry.

Significance of the Study

The significance of predicting food delivery times lies in its potential to revolutionize the operational efficiency of food delivery services. Accurate predictions can assist businesses in optimizing resource allocation, reducing delivery times, and improving customer satisfaction. As the competition in the food delivery industry intensifies, the ability to provide reliable and precise delivery time estimates can be a key differentiator for service providers. In the subsequent sections of this project report, we delve into the data exploration and preprocessing steps, the application of various regression models, and the evaluation of their performance. Additionally, visualizations and analyses are presented to provide a comprehensive understanding of the factors influencing food delivery times.

The project concludes with recommendations and insights derived from the predictive models, offering valuable guidance for stakeholders in the food delivery ecosystem.

Procedure

This Python program employs various regression models, including **Linear Regression** and **Ridge Regression**, to predict delivery times based on features such as delivery person attributes, geographic coordinates, and order details. The dataset is first loaded using **Pandas** and visualized using **Seaborn** and **Plotly**. The data is then cleaned, including the computation of distances between locations. Feature engineering involves creating dummy variables for categorical features. Model performance is compared using cross-validation, and the models are trained and evaluated using metrics like **Mean Squared Error** and **R-squared**. The program also includes visualizations of predicted vs. actual points and real data vs. predictions for both **Linear Regression** and **Ridge Regression**. Finally, the models are applied to new data for predicting delivery times in real-world scenarios.

1. Importing required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor,plot_tree
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score,KFold
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
import plotly.express as px
```

The key libraries imported are **pandas** for data manipulation, **numpy** for numerical operations, **matplotlib.pyplot** for basic plotting, various modules from **scikit-learn** for machine learning tasks such as regression and model evaluation, **seaborn** for advanced data visualization, and **plotly express** for interactive plotting.

2. Reading the dataset

```
fd1=pd.read_csv("deliverytime.csv")
fd=pd.DataFrame(fd1)
print(fd.head())
print(fd.shape)
print(fd.info())
print(fd.describe())
print(fd.isnull().sum())
```

Our dataset named **deliverytime.csv** is read into a pandas dataframe. The information about its columns and data types is given by **fd.info()**. The count of missing values in each column is given by **fd.isnull().sum()**.

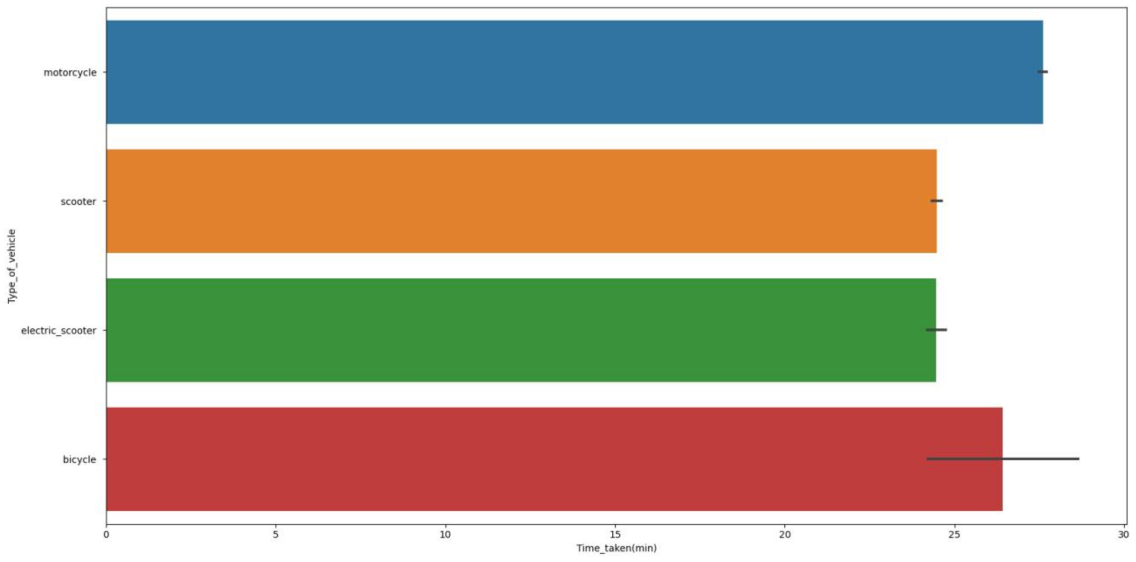
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           45593 non-null  object
 1   Delivery_person_ID           45593 non-null  object
 2   Delivery_person_Age          45593 non-null  int64
 3   Delivery_person_Ratings      45593 non-null  float64
 4   Restaurant_latitude          45593 non-null  float64
 5   Restaurant_longitude         45593 non-null  float64
 6   Delivery_location_latitude   45593 non-null  float64
 7   Delivery_location_longitude  45593 non-null  float64
 8   Type_of_order                45593 non-null  object
 9   Type_of_vehicle              45593 non-null  object
 10  Time_taken(min)              45593 non-null  int64
dtypes: float64(5), int64(2), object(4)
```

This initial exploration helps understand the structure, size, and content of the dataset, as well as identify any missing values that may need to be addressed in the subsequent data cleaning steps.

3. Data visualization

Data visualization offer insights into the relationships between different features and the target variable **Time_taken(min)** in the dataset. They can aid in understanding patterns, correlations, and potential influences on delivery times.

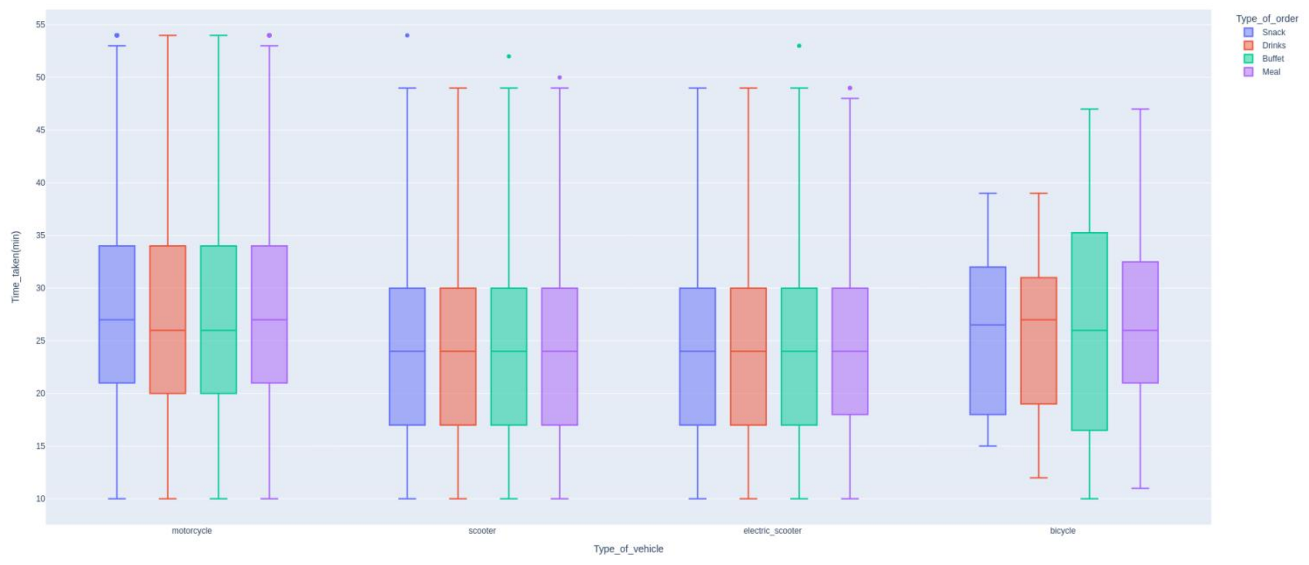
```
sns.barplot(data=fd,y='Type_of_vehicle',x='Time_taken(min)')
plt.show()
```



The plot visualizes the relationship between the categorical variable **Type_of_vehicle** on the y-axis and the numerical variable **Time_taken(min)** on the x-axis. Each bar represents a type of vehicle and the height of the bar corresponds to the average time taken by that vehicle.

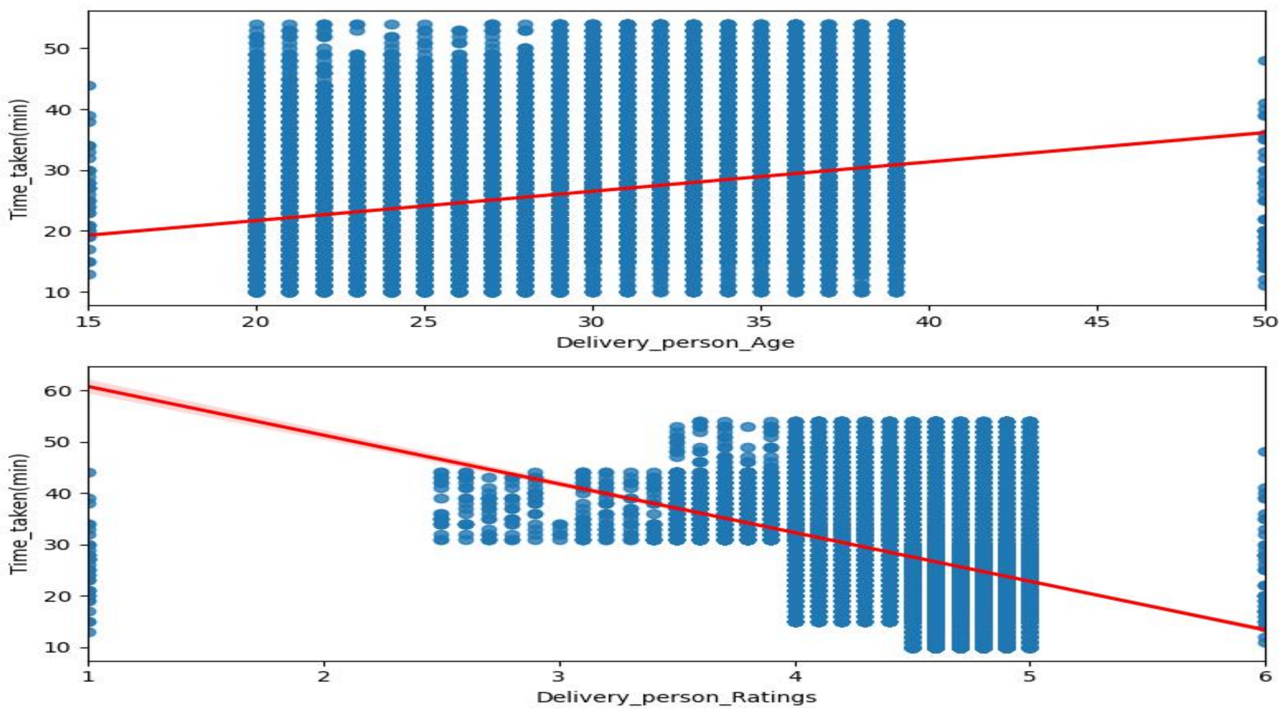
```
fig=px.box(fd,x='Type_of_vehicle',y='Time_taken(min)',color='Type_of_order')
fig.show()
```

This plot is effective for visualizing the distribution of a numerical variable **Time_taken(min)** across different categories of two categorical variables **Type_of_vehicle** and **Type_of_order**. The color parameter is used to differentiate the boxes based on the **Type_of_order**.



This plot helps to understand the distribution of delivery times for different types of vehicles, with further distinction based on the type of order. The box plot provides insights into the central tendency, spread, and potential outliers within each category.

```
fig=plt.figure(figsize=(8,7))
fig.add_subplot(211)
sns.regplot(data=fd,x='Delivery_person_Age',y='Time_taken(min)',line_kws={'color':'red'})
fig.add_subplot(212)
sns.regplot(data=fd,x='Delivery_person_Ratings',y='Time_taken(min)',line_kws={'color':'red'})
plt.tight_layout()
plt.show()
```



This plot allows for the visualization of the linear relationships between delivery time and both the delivery person's age and ratings. The red regression lines help in understanding the trend in the data.

4. Data cleaning

Data cleaning steps contribute to preparing the data for further analysis and modeling. They address issues such as removing unnecessary columns, deriving new features, and encoding categorical variables.

4.1 Removing columns

Dropping irrelevant columns is a common data cleaning step to focus on relevant features for analysis or modeling.

```
fd=fd.drop(['ID','Delivery_person_ID'],axis=1)
print(fd.head())
```

This method is used to drop columns **ID** and **Delivery_person_ID** along the columns axis.

4.2 Calculating distance

```
def dis(lat1, lon1, lat2, lon2):

    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    newlon = lon2 - lon1
    newlat = lat2 - lat1

    haver_formula = np.sin(newlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(newlon/2.0)**2

    dist = 2 * np.arcsin(np.sqrt(haver_formula ))
    km = 6367 * dist
    return km
dis_cols = ['Restaurant_latitude', 'Restaurant_longitude', 'Delivery_location_latitude','Delivery_location_longitude']
fd['Distance'] = fd[dis_cols].apply(
    lambda x: dis(x[0], x[1], x[2], x[3]),
    axis=1)

print(fd.head())
print(fd.tail())
```

The function defined as **dis** calculates the distance between two sets of latitude and longitude coordinates.This step effectively adds a new column **Distance** to the dataframe, containing the calculated distances between the latitude and longitude coordinates for restaurant and delivery locations. This distance metric can be useful in various analyses, particularly in predicting delivery times based on spatial information.

4.3 Categorical variable encoding

```
dummy_fd=pd.get_dummies(fd[['Type_of_order', 'Type_of_vehicle']], dtype='int64')
fd=pd.concat([fd,dummy_fd],axis=1)
fd=fd.drop(['Type_of_order', 'Type_of_vehicle'],axis=1)
print(fd.info())
```

Here we convert categorical variables **Type_of_order** and **Type_of_vehicle** into dummy variables and concatenate them back to the dataframe. The original categorical columns are then dropped.

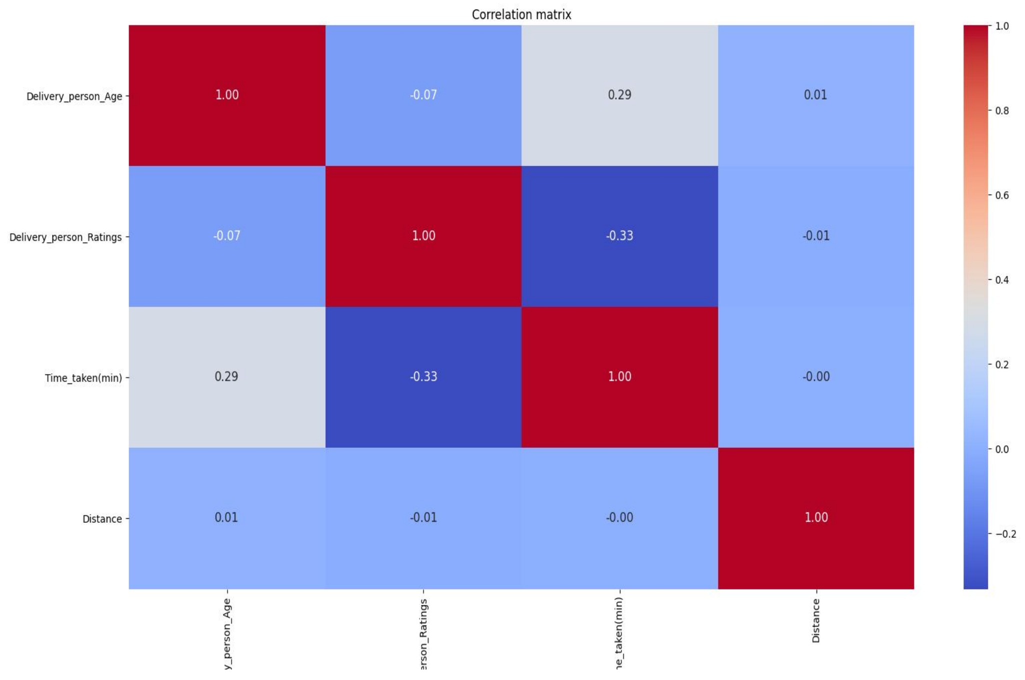
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 16 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Delivery_person_Age               45593 non-null  int64
 1   Delivery_person_Ratings           45593 non-null  float64
 2   Restaurant_latitude               45593 non-null  float64
 3   Restaurant_longitude              45593 non-null  float64
 4   Delivery_location_latitude        45593 non-null  float64
 5   Delivery_location_longitude       45593 non-null  float64
 6   Time_taken(min)                   45593 non-null  int64
 7   Distance                          45593 non-null  float64
 8   Type_of_order_Buffet              45593 non-null  int64
 9   Type_of_order_Drinks              45593 non-null  int64
 10  Type_of_order_Meal                45593 non-null  int64
 11  Type_of_order_Snack               45593 non-null  int64
 12  Type_of_vehicle_bicycle           45593 non-null  int64
 13  Type_of_vehicle_electric_scooter  45593 non-null  int64
 14  Type_of_vehicle_motorcycle        45593 non-null  int64
 15  Type_of_vehicle_scooter           45593 non-null  int64
dtypes: float64(6), int64(10)
```

The result is the dataframe **fd** that includes the original features along with the new one-hot encoded columns representing the categorical variables.

5. Correlation matrix

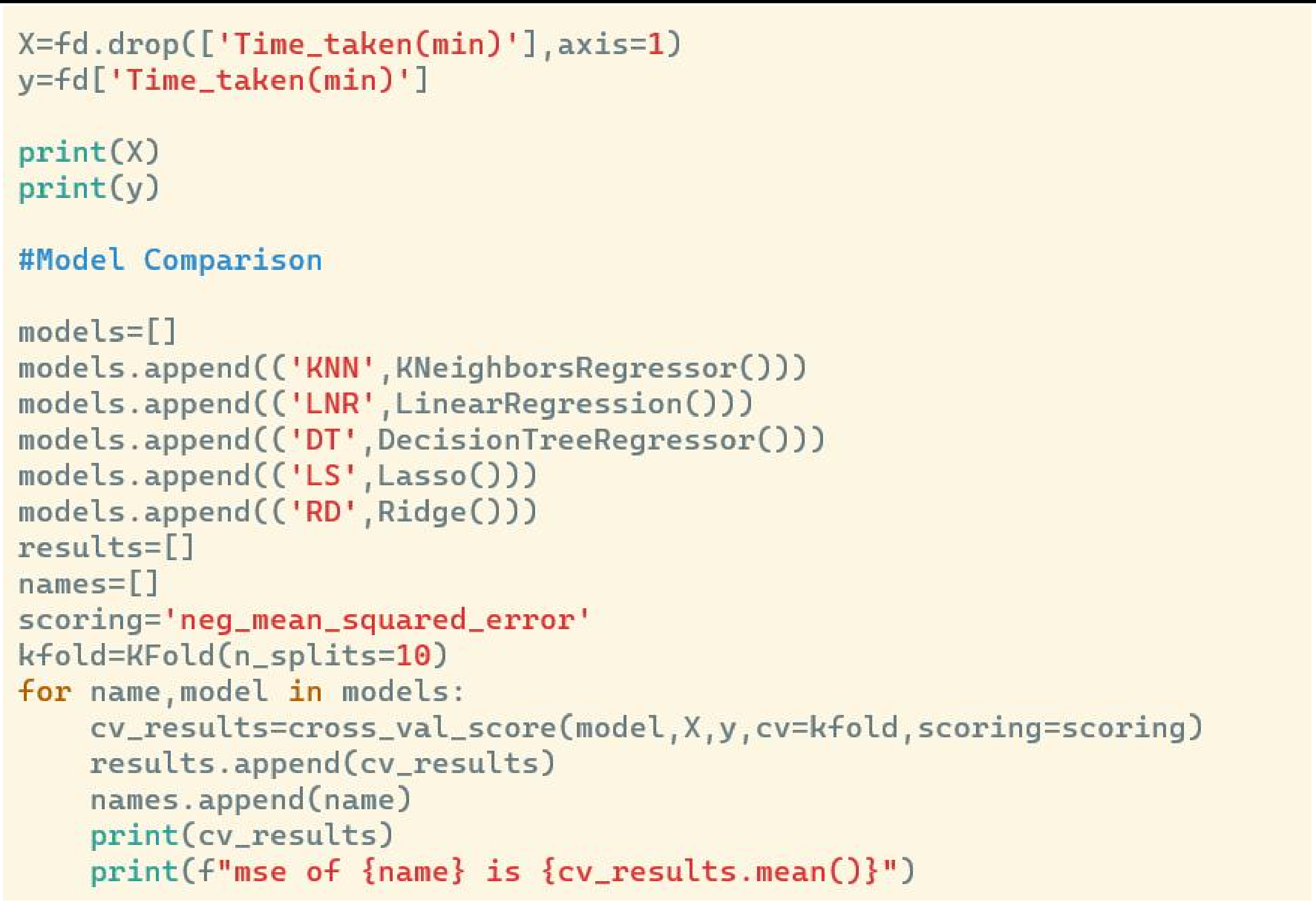
The heatmap provides a visual representation of the correlations between the selected numerical variables. It helps identify patterns, relationships, and potential multicollinearity between features. The color intensity and annotation values indicate the strength and direction of the correlations.

```
Correlation_matrix=fd[['Delivery_person_Age','Delivery_person_Ratings','Time_taken(min)','Distance']].corr()
plt.figure(figsize=(8,6))
sns.heatmap(Correlation_matrix,annot=True,cmap='coolwarm',fmt="0.2f",annot_kws={"size":12})
plt.title('Correlation matrix')
plt.show()
```



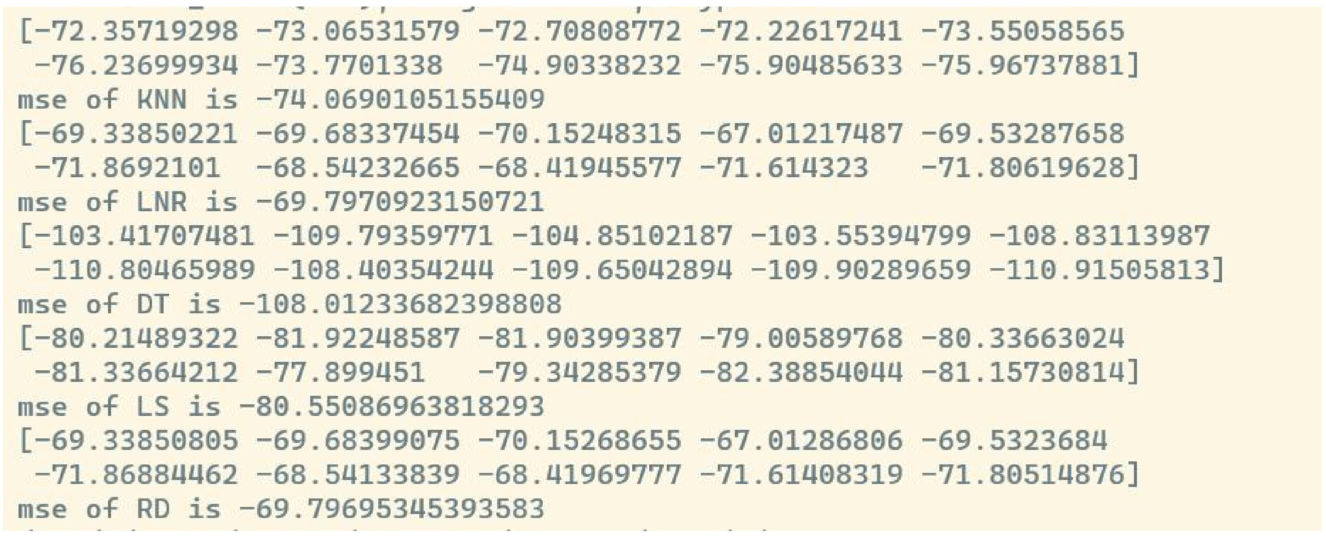
6. Model comparison

Here we are preparing the data for modeling and comparing the performance of different regression algorithms using cross-validation. This helps us compare the performance of different regression models on our dataset. The negative MSE is used, and lower values indicate better performance. The results are printed for each model, allowing us to assess which algorithm performs best on our data.



```
X=fd.drop(['Time_taken(min)'],axis=1)
y=fd['Time_taken(min)']

print(X)
print(y)

#Model Comparison

models=[]
models.append(('KNN',KNeighborsRegressor()))
models.append(('LNR',LinearRegression()))
models.append(('DT',DecisionTreeRegressor()))
models.append(('LS',Lasso()))
models.append(('RD',Ridge()))
results=[]
names=[]
scoring='neg_mean_squared_error'
kfold=KFold(n_splits=10)
for name,model in models:
    cv_results=cross_val_score(model,X,y,cv=kfold,scoring=scoring)
    results.append(cv_results)
    names.append(name)
    print(cv_results)
    print(f"mse of {name} is {cv_results.mean()}")
```

Here we create a list of regression models **KNN**, **Linear Regression**, **Decision Tree**, **Lasso**, **Ridge** and use k-fold cross-validation to evaluate the models. After that we calculate the negative mean squared error (MSE) as the evaluation metric.

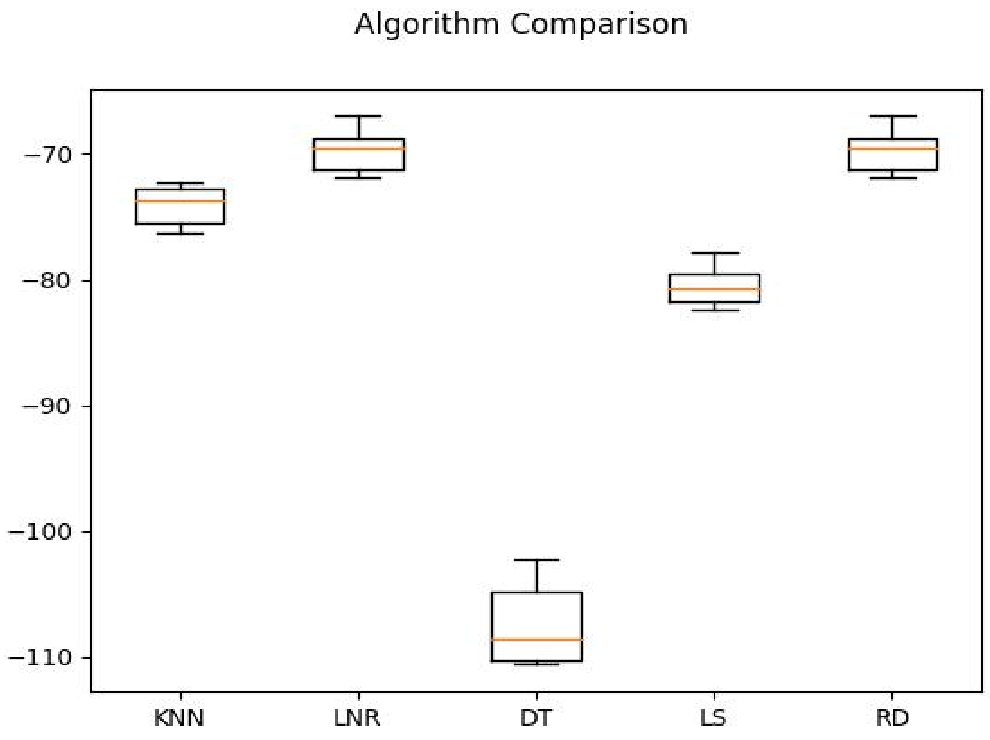


```
[-72.35719298 -73.06531579 -72.70808772 -72.22617241 -73.55058565
 -76.23699934 -73.7701338  -74.90338232 -75.90485633 -75.96737881]
mse of KNN is -74.0690105155409
[-69.33850221 -69.68337454 -70.15248315 -67.01217487 -69.53287658
 -71.8692101  -68.54232665 -68.41945577 -71.614323   -71.80619628]
mse of LNR is -69.7970923150721
[-103.41707481 -109.79359771 -104.85102187 -103.55394799 -108.83113987
 -110.80465989 -108.40354244 -109.65042894 -109.90289659 -110.91505813]
mse of DT is -108.01233682398808
[-80.21489322 -81.92248587 -81.90399387 -79.00589768 -80.33663024
 -81.33664212 -77.899451   -79.34285379 -82.38854044 -81.15730814]
mse of LS is -80.55086963818293
[-69.33850805 -69.68399075 -70.15268655 -67.01286806 -69.5323684
 -71.86884462 -68.54133839 -68.41969777 -71.61408319 -71.80514876]
mse of RD is -69.79695345393583
```

Linear regression and **Ridge regression** has the least error value which indicates that they are the best models for our prediction.

Now we create a boxplot to visually compare the performance of different regression algorithms based on their negative mean squared errors (MSE) obtained through cross-validation.

```
fig=plt.figure()
fig.suptitle("Algorithm Comparison")
ax=fig.add_subplot(111)
plt.boxplot(results)

ax.set_xticklabels(names)
plt.show()
```



Lower values on the y-axis indicate better performance.

7. Feature scaling and train-test split

In this part we are preparing the data for training and testing a regression model. The steps include splitting the data into training and testing sets, and then scaling the numerical features using **StandardScaler**.

```
X=fd.drop(['Time_taken(min)'],axis=1)
y=fd['Time_taken(min)']


X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

Split the data into training and testing sets. 80% of the data is used for training **X_train**, **y_train**, and 20% for testing **X_test**, **y_test**. The **random_state** parameter ensures reproducibility.

```
scaler= StandardScaler()
X_train1= scaler.fit_transform(X_train)

X_test1= scaler.transform(X_test)
```

Use **StandardScaler** to standardize the numerical features separately for the training and testing sets. The **fit_transform** method is used on the training set, and the **transform** method is used on the testing set to ensure consistent scaling.

```
print(X_train1.shape)      (36474, 15)
print(X_test1.shape)       (9119, 15)
print(y_train.shape)       (36474,)
print(y_test.shape)        (9119,)
print(y.shape)             (45593,)
print(X.shape)             (45593, 15)
```

This section is crucial for preparing the data in a standardized form before feeding it into machine learning models.

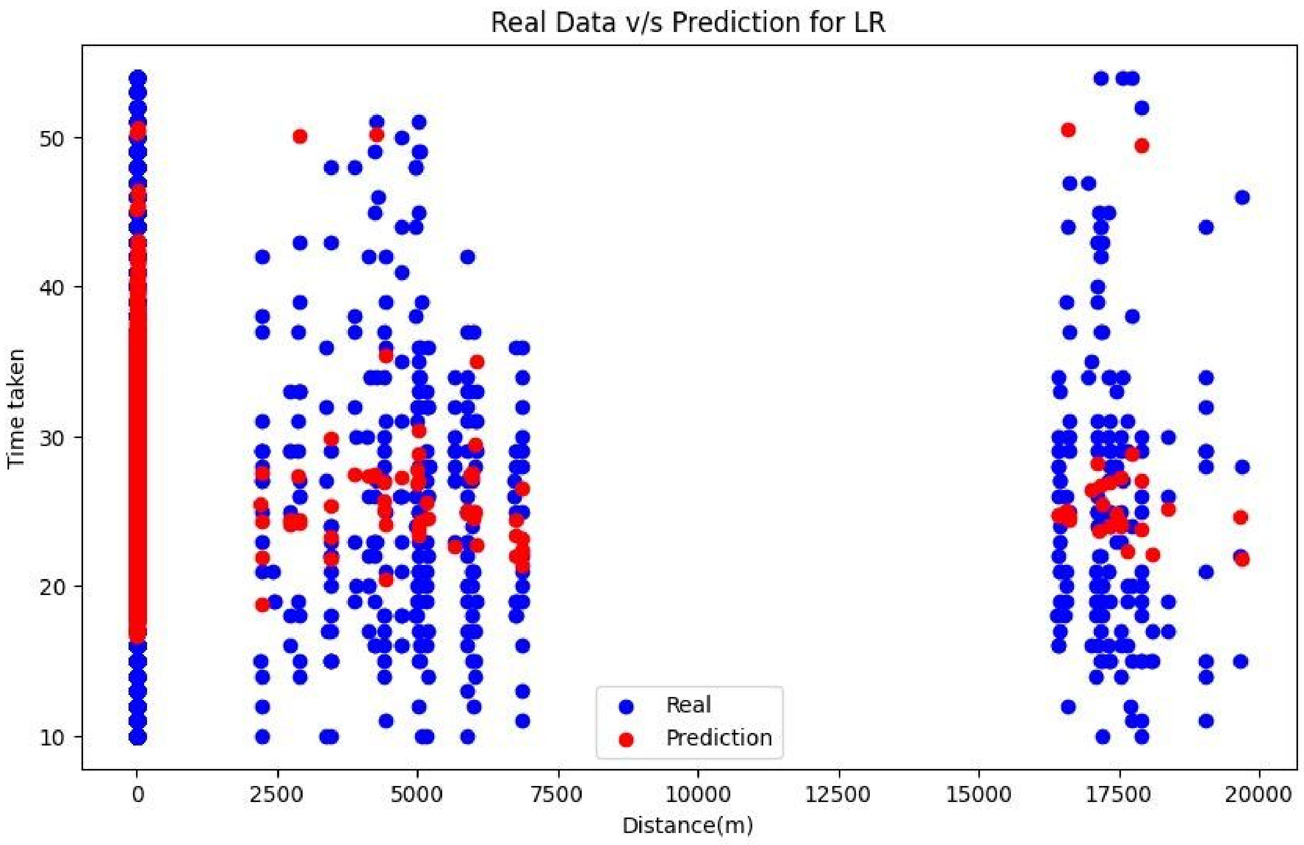
8. Prediction using Linear regression

In this part, we train a **Linear Regression model** on the scaled training data **X_train1**, **y_train** and then make predictions on the scaled testing data **X_test1**. Subsequently, we evaluate the performance of the model using various regression metrics such as **Mean Squared Error** (MSE), **Root Mean Squared Error** (RMSE), **Mean Absolute Error** (MAE), **R-squared score**(R2 score).

```
lr=LinearRegression()
lr.fit(X_train1,y_train)
plr=lr.predict(X_test1)

print(y)
print(plr)

print(mean_squared_error(y_test,plr))
print(np.sqrt(mean_squared_error(y_test,plr)))
print(mean_absolute_error(y_test,plr))
print(r2_score(y_test,plr))
```

```
MSE= 69.91629696506209
RMSE= 8.361596555985113
MAE= 6.610221176330825
R2= 0.20258003222258936
```

The comparison of true vs predicted values helps in visualizing how well the model predictions align with the actual outcomes.

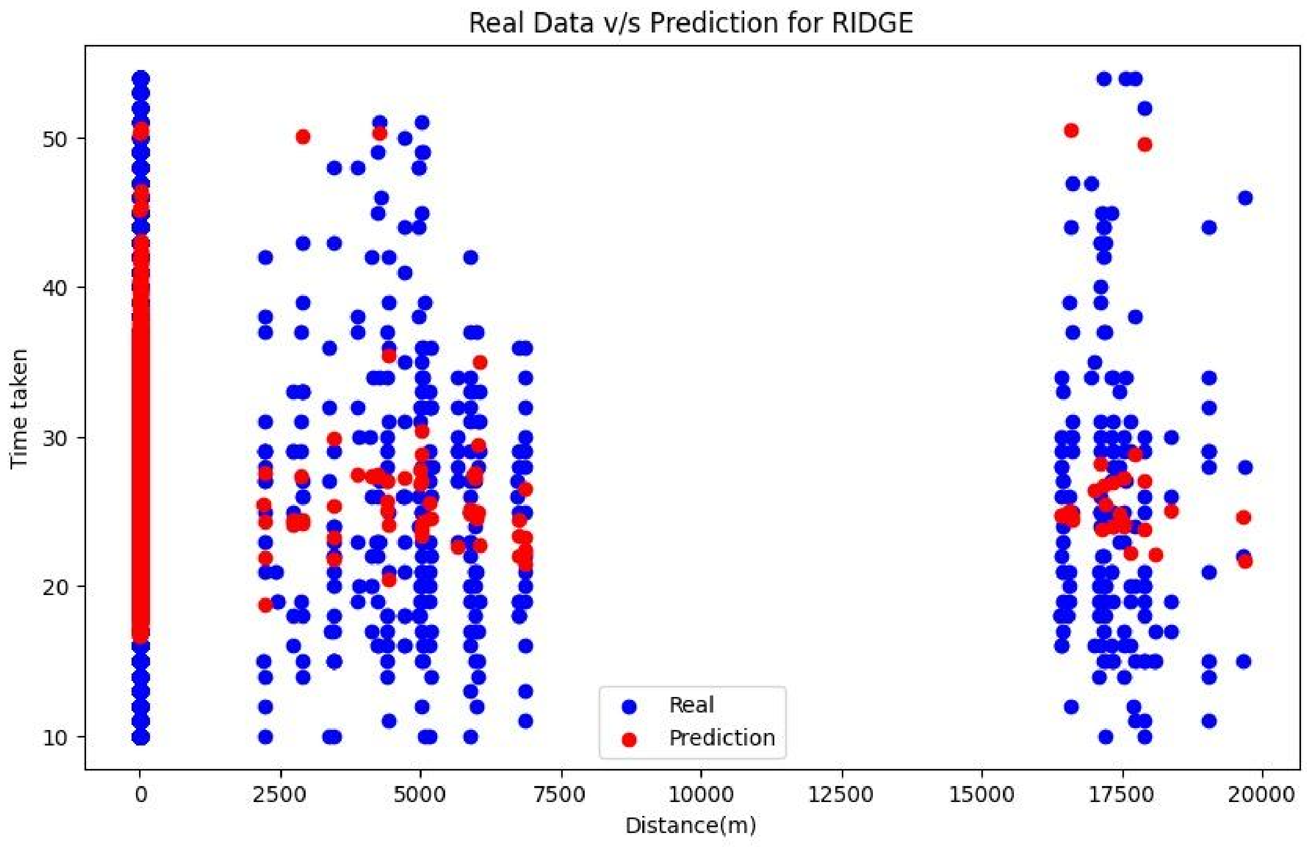


9. Prediction using Ridge regression

Here we train a **Ridge regression model** and follow the same steps as we did for Linear regression model.

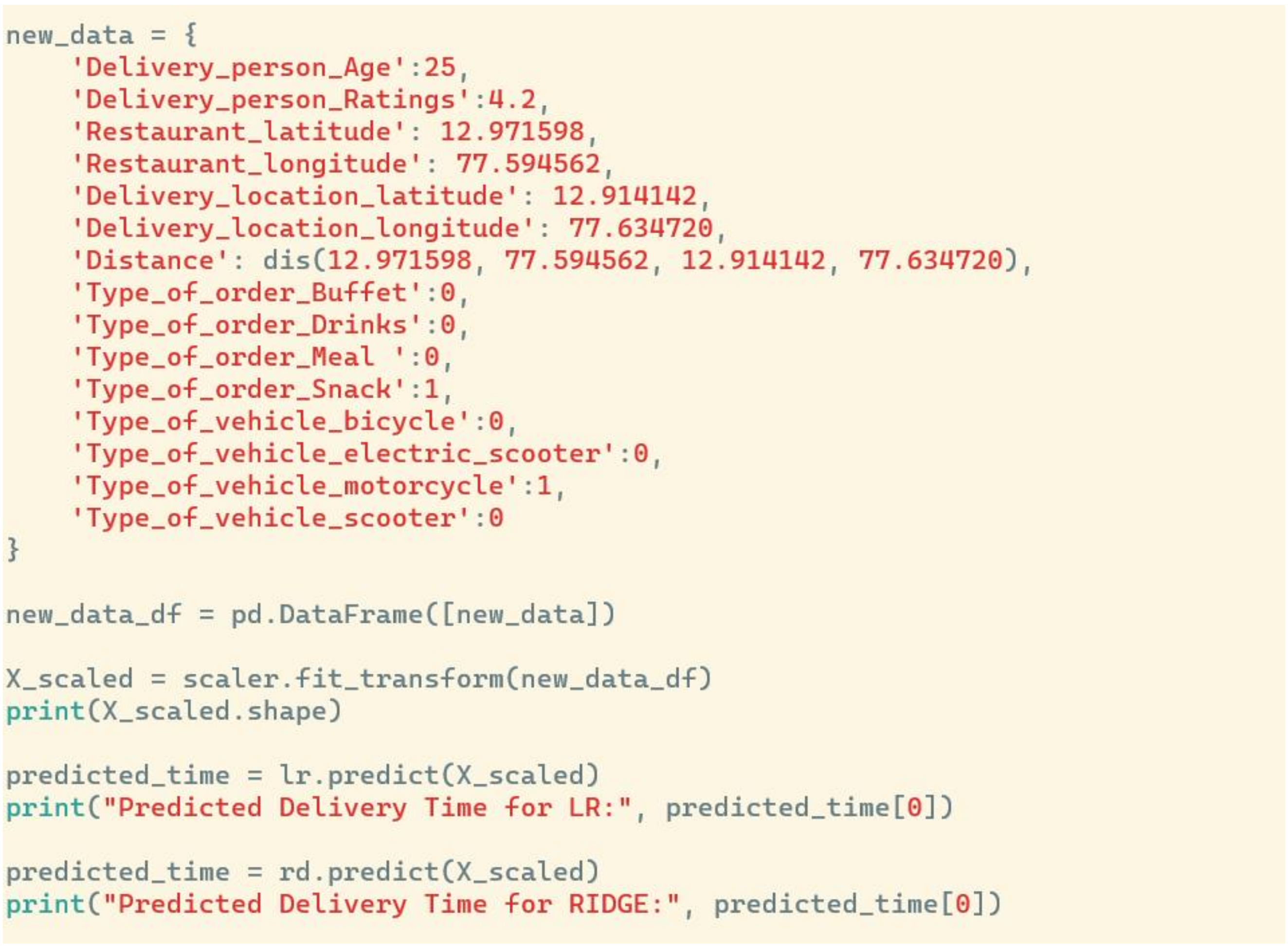
```
rd=Ridge(alpha=1.0)
rd.fit(X_train1,y_train)
prd=rd.predict(X_test1)

print(y)
print(prd)

print(mean_squared_error(y_test,prd))
print(np.sqrt(mean_squared_error(y_test,prd)))
print(mean_absolute_error(y_test,prd))
print(r2_score(y_test,prd))
```

```
MSE= 69.91628071599106
RMSE= 8.361595584336225
MAE= 6.610224880847482
R2= 0.20258021754896183
```

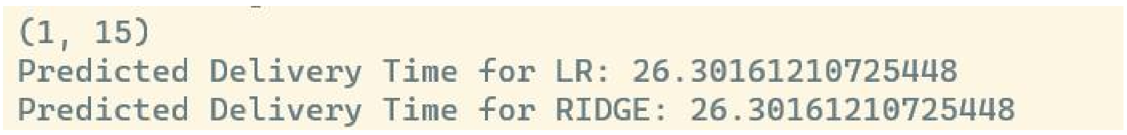


10. Model deployment

Now that we have the best-performing model, it can be deployed to make the prediction. The model takes inputs such as order details, rider details, restaurant details, and returns the predicted delivery time.



```
new_data = {
    'Delivery_person_Age':25,
    'Delivery_person_Ratings':4.2,
    'Restaurant_latitude': 12.971598,
    'Restaurant_longitude': 77.594562,
    'Delivery_location_latitude': 12.914142,
    'Delivery_location_longitude': 77.634720,
    'Distance': dis(12.971598, 77.594562, 12.914142, 77.634720),
    'Type_of_order_Buffet':0,
    'Type_of_order_Drinks':0,
    'Type_of_order_Meal ':0,
    'Type_of_order_Snack':1,
    'Type_of_vehicle_bicycle':0,
    'Type_of_vehicle_electric_scooter':0,
    'Type_of_vehicle_motorcycle':1,
    'Type_of_vehicle_scooter':0
}

new_data_df = pd.DataFrame([new_data])

X_scaled = scaler.fit_transform(new_data_df)
print(X_scaled.shape)

predicted_time = lr.predict(X_scaled)
print("Predicted Delivery Time for LR:", predicted_time[0])

predicted_time = rd.predict(X_scaled)
print("Predicted Delivery Time for RIDGE:", predicted_time[0])
```



```
(1, 15)
Predicted Delivery Time for LR: 26.30161210725448
Predicted Delivery Time for RIDGE: 26.30161210725448
```

Conclusion

In conclusion, the food delivery time prediction project involves extensive data analysis, visualization, cleaning, and modeling to understand and predict the time taken for food deliveries.
The project starts by loading and exploring the dataset, checking its structure, summary statistics, and identifying missing values. Visualizations, including bar plots, box plots, and regression plots, provide insights into the relationships between different variables and help understand the data distribution. Irrelevant columns are dropped. A distance metric is calculated based on geographical coordinates. Categorical variables are encoded using one-hot encoding. A correlation matrix is created to visualize the relationships between numerical features, highlighting potential correlations with the target variable. Several regression models are trained and evaluated using cross-validation. The models are compared based on metrics like Mean Squared Error (MSE) using k-fold cross-validation, providing insights into their performance. Numerical features are scaled using StandardScaler, and the dataset is split into training and testing sets. Linear Regression and Ridge Regression models are trained and evaluated on the testing set using metrics such as MSE, RMSE, MAE, and R-squared. The trained models are used to predict the delivery time for a new data point containing relevant features.
Overall, this project demonstrates the application of machine learning techniques for predicting food delivery times based on various factors. The chosen regression models provide insights into the relationships within the dataset and can be utilized in a real-world scenario for making delivery time predictions.