Project 9: Predicting Food Delivery Times

Submitted By: Andrew Stambaugh, Undergraduate Student

Dept. of Mechanical Engineering

Northwestern University

Date Submitted: 12/09/2022

Table of contents

Predicting Food Delivery Times

Andrew Stambaugh

Abstract: This project details the use of the six principles of Mechanistic Data Science to translate a Kaggle dataset of food deliveries to a Random Forest Regression model and Neural Network which can accurately predict the arrival time of food deliveries upon being sent out. Through deep data mining, creative features were engineered to finely optimize the model and accurately represent nonlinear and non-numeric relationships within the data.

Keywords: *Data Science, Machine Learning, Deep Learning, Neural Network, and Feature Engineering*

Introduction

Food delivery is a service which has become increasingly popular in the food industry over the past few years, especially with the COVID-19 pandemic limiting physical contact across the world. As more companies funnel into this market, there is an increased need for fast, reliable delivery times as these best communicate to customers the efficiency of their service. Many companies, including DoorDash, Uber, and more have begun implementing forms of machine learning to provide delivery time estimates. However, can a machine learning model give reliably accurate delivery time estimates when given enough data? This project seeks to answer this question through following the six modules of Mechanistic Data Science (MDS): Multimodal Data Generation and Collection, Extraction of Mechanistic Features, KnowledgeDriven Dimension Reduction, Reduced Order Surrogate Models, Deep Learning for Regression and Classification, and, finally, System and Design [1]. These modules form the crux of this project's methodology, and their applications span across a diverse array of projects, including heart failure classification, audio translation, or even optimizing a sports team model. The main dataset for this project was retrieved from Kaggle and contains over 45,000 entries of food deliveries across multiple platforms and companies. All coding was done in Python through standard libraries such as pandas, numpy, sklearn, and tensorflow. Through completing this project, I sought to not only learn more about MDS as a whole, but also to learn new technical skills in data science such as using tensorflow and coding my first neural network model.

Multimodal Data Generation and Collection

(https://www.kaggle.com/datasets/gauravmalik26/food-delivery-dataset)

This project's dataset was retrieved from Kaggle through the link above and contains over 45,000 entries of food deliveries across different platforms, food companies, and third-party service providers. It contains 19 input features and one dependent variable, the time it takes for the food to be delivered upon leaving the restaurant. The inputs features include Delivery ID, Delivery Person ID, Delivery Person Age (Years), Delivery Person Average Rating (1-5),
Restaurant Latitude, Restaurant Longitude, Delivery Location Latitude, Delivery Location Longitude, Order Date (M/D/Y), Time Ordered Placed (hh:mm:ss), Time Picked Up (hh:mm:ss), Weather Conditions, Road Traffic Density, Type of Vehicle, Vehicle Condition, Type of Order

(Snack, Meal or Buffet), Multiple Deliveries? (0-3), Delivering to a Festival? (Binary), and Urban Density. Table 1 is an example of some of the dataset's features. Upon importing the data, I converted it into a panda data frame in Python.

Table 1: Example Portion of Main Dataset (Raw)

Road_traffic_density	Type_of_vehicle	Weatherconditions	Festival	Urban Density	Time_taken(min)
High	motorcycle	conditions Sunny	No	Urban	(min) 24
Jam	scooter	conditions Stormy	No	Metropolitian	(min) 33
Low	motorcycle	conditions Sandstorms	No	Urban	(min) 26

Mechanistic Features Extraction Part 1

The raw dataset from Kaggle contained improper formatting for many fields. Thus, the data needed to be heavily preprocessed and the proper features be extracted where necessary.

Preprocessing

The dataset required an immense amount of preprocessing before serious feature extraction could ensue. The data cleaning process involved removing NULL values, dividing categorical variables into multiple binary variables, converting all scales to numeric values, universalizing date time formats, and removing any unnecessary strings from numeric entries. Table 2 shows Table 1 after data cleaning.

Table 2: Example Portion of Main Dataset (Clean)

Road Traffic Density	Motorcycle	Scooter	Sunny	Stormy	Sandstorms	Festival	Urban Density	Time Taken (min)
2	1	0	1	0	0	0	2	24
3	0	1	0	1	0	0	3	33
0	1	0	0	0	1	0	2	26

Feature Extraction

After preprocessing, the correct features needed to be extracted from the restaurant coordinates, delivery coordinates, order date, and order time. The coordinates themselves possess little interpretation in a machine learning model besides the distance between them. Using the distance method from the geopy Python library, the geodesic between the coordinates was calculated and the distance recorded. As for the date and time, I used the date time Python library to extract various temporal features from them. These include the day of the week, month, season, hour, rush hour?, night time?, and weekend? These are all features I hypothesized may affect an order's delivery time.

Knowledge Driven Dimension Reduction

Processing the data's features into usable formats consequently increased its overall dimensionality to over 35 features. Due to this, I sought methods of dimension reduction.

K-means Clustering

The first of these methods was to cluster the numerical data with the K-means algorithm in hopes of discovering a latent variable. However, as shown in Figure 1, the clusters only divide a dense dataset into irrelevant blocks. Though some interpretations are possible, such as the blue cluster being longer distance deliveries with good to mediocre delivery ratings, these do not point to a latent variable. Other attempts to cluster other features resulted in the same outcomes, leading to the conclusion that the data contains too much noise to possess any meaningful K-means clusters and, therefore, other dimension reduction methods should be sought.

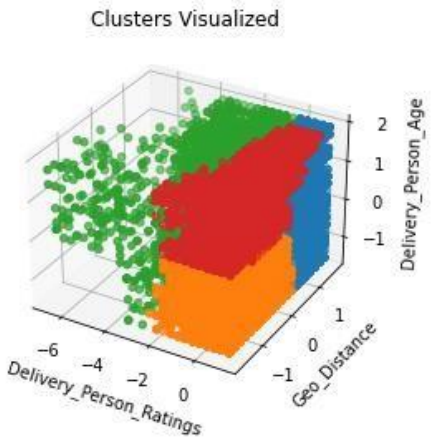

Figure 1: K-means Clusters Visualized

Correlation Matrix

A simple yet effective method for determining a variable's redundancy and relevance is the correlation matrix. In regard to redundancy, I implemented a maximum of 0.6 input-input correlation, but even with this strict criterion, no variables possessed such a correlation together, meaning redundancy was likely not an issue. On the other hand, in regard to relevance, I implemented a minimum of 0.2 input-output correlation, meaning if a variable's correlation with delivery time was less than 0.2, then it was deemed irrelevant and removed from the dataset. With this criterion, the number of dimensions was reduced from 35+ to only 13, including age, driver ratings, traffic level, vehicle condition, geo distance, urban density, multiple deliveries?, festival?, nighttime?, rush hour?, sunny?, fog?, and motorcycle?

Reduced Order Surrogate Model

In my pursuit of dimension reduction, I attempted to find a latent space within the dataset in order to potentially create a reduced order model. However, with the majority of the dataset being binary or categorical variables, finding a proper latent space spanning all of the data necessitated an algorithm beyond the scope of my data science knowledge Hence, I focused on reducing the order of only the non-binary features.

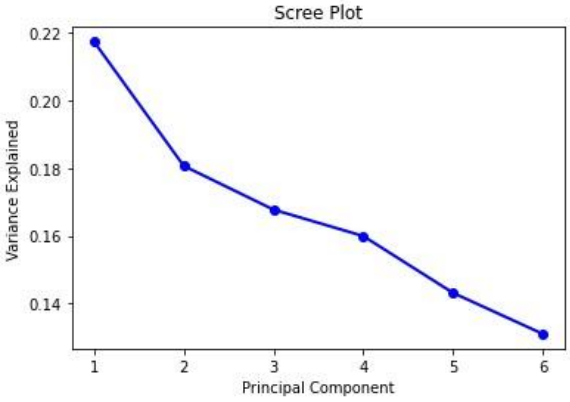

Figure 2: PCA Scree Plot

Principle Component Analysis

Using Principle Component Analysis, the results for a latent space were unsatisfactory. As shown in the scree plot in Figure 2, the first principle component only explains about 22% of the variance in the data. Furthermore, in order to capture at least 95% of data's variance, all 6 components are needed. This results in no dimension reduction. Thus, a reduced order model was not implemented for the dataset.

Mechanistic Features Extraction Part 2

Although the proper mechanistic features have already been extracted, creative feature engineering must be done in order to best model each feature's relationship with delivery time, specifically non-linear or non-continuous relationships. Deep data mining was done on the dataset, and this section goes over the major takeaways found in this process.

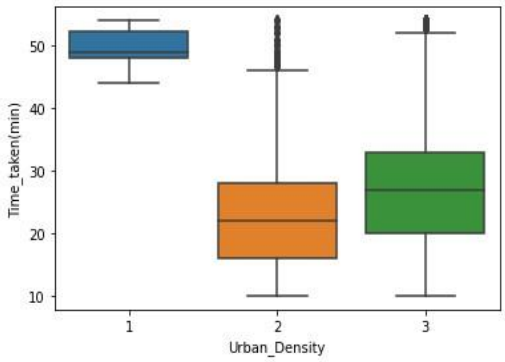

Figure 3: Delivery Time by Urban Density

Lack of Suburban Data

Suburban food deliveries, across all features, behave entirely different than those in urban or metropolitan areas. Not only do they have significantly higher delivery times on average (Figure 3), but features such as geo distance or delivery ratings no longer have any correlation to delivery time. In other words, suburban deliveries need a separate model entirely. With these deliveries only making up 0.3% of the data, I felt it was necessary to narrow the scope of my project to only urban and metropolitan areas as I did not have a sufficient amount of data to create a separate, reliable model for suburban food deliveries.

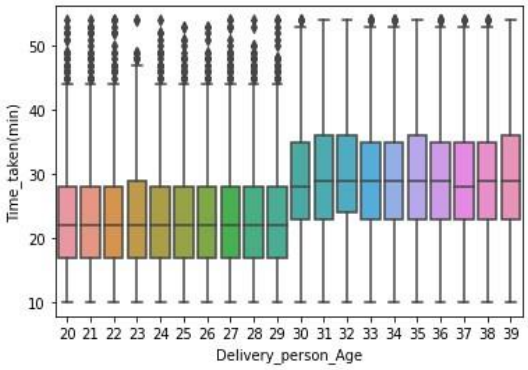

Figure 5: Boxplots of Delivery Time by Age

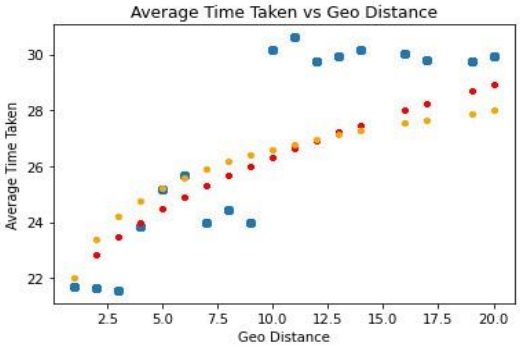

Figure 4: Average Delivery Time by Geo Distance vs Square Root and Log Functions

Dichotomous Variables

Many variables not only showed a nonlinear relationship with delivery time, but also a discontinuous, dichotomous relationship. Examples of this are seen in a delivery person's age (Figure 4), driver ratings, and the geo distance between the restaurant and destination. In Figure 4, the delivery person's age has no effect on the median delivery time until the age of 30, in which it abruptly increases and levels out again. This exact phenomenon happens for driver ratings, as well, at 4.5 stars, except decreasing instead. In order to account for this, these features were transformed into a binary format, splitting the data at the aforementioned dichotomous lines. However, despite having a similar split behavior at values greater than 10 (Figure 5), geo distance still possesses somewhat of a positive correlation on the left side of the split. To account for this, rather than using a simple binary variable, I split geo distance into two variables, geo distance * (geo distance >= 10) and geo distance * (geo distance < 10).

Other Important Relationships

Although not surprising behavior, multiple deliveries possesses a visually linear behavior in relation to the time taken to complete a delivery (Figure 6). This is an important discovery as it supports the hypothesis that under similar conditions, delivery times will have similar values (assuming each delivery in these multiple deliveries is under similar conditions). Simply put, delivery times can be predicted given the correct knowledge and relationships of its conditions. Finally, vehicle conditions have little effect on the delivery time, except when the vehicle is in the worst condition possible. Thus, it was best to represent this feature as a binary rather than a 0-2 scale.

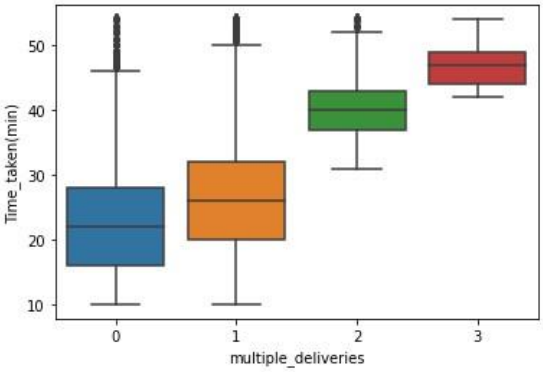

Figure 6: Boxplots of Delivery Time by Additional Number of Delivery on Route

Deep Learning for Regression

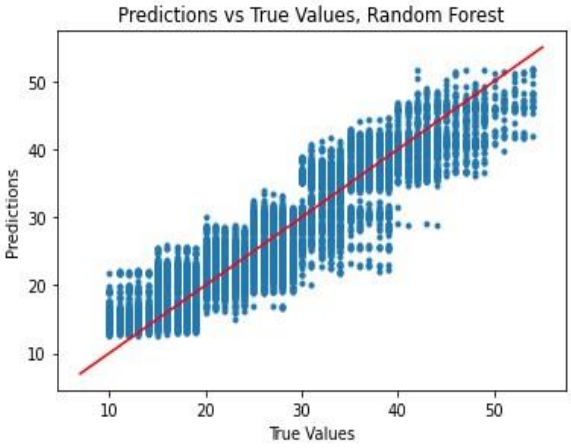

Figure 7: Random Forest Predictions

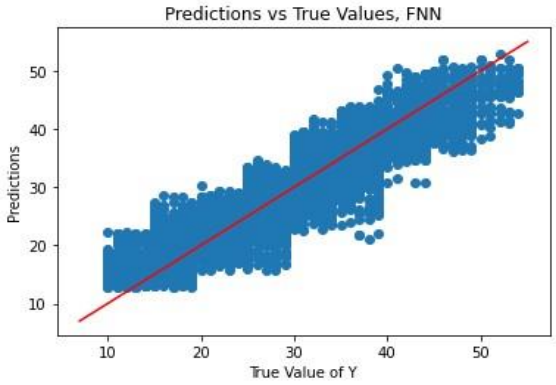

Figure 8: FNN Predictions

Although deep learning more specifically refers to neural networks, I initially implemented a more simple machine learning model with Random Forest Regression. My initial thoughts were that the data in this project was simple, and, thus did not need the more computationally expensive neural network model in order to be sufficiently accurate. Although this proved to be true, I decided to implement a Feedforward Neural Network otherwise for the sake of comparison and to better follow the Deep Learning aspect of this module. *Random Forest vs. FNN*

Other than its simpler implementation than a neural network, I chose to implement this algorithm as it excels in deciphering convoluted, nonlinear relationships as opposed to other forms of regression. Using the RandomForestRegressor() function from sklearn.ensemble in Python, I implemented this model with 500 trees, each with a depth of 11, and only 8 features used, maximizing the R-squared at 0.8431.

For the FNN's implementation, KerasRegressor() was used from tensorflow in Python with one input layer and two hidden layers, each with 25 neurons and the Rectified Linear Unit activation function. This model's performance peaked around 30 epochs with a batch size of 5 at an R-squared of 0.8331. Though close, the Random Forest model

is still slightly more accurate, despite having a simpler algorithm. This is reflected in the plots of their predictions (Figure 7 and 8), as well. While, the Random Forest is limited to integers as that is what it has been trained on, the predictions are generally tighter to the line, indicating that it's more accurate on average. However, as this was my first neural network, there is certainly room for fine tuning and improving that model much further. All in all, both models are respectably accurate and prove that given sufficient features of a food delivery order, it's possible to accurately predict most delivery times.

System and Design

For system and design, the fitted model can be directly implemented into a food delivery service's UI. Upon an order being sent out for delivery, the real time data is recorded by the system and input into this model's algorithm, including the data cleaning and feature extraction process. One concern with this algorithm is that it evidently does not take into account that the conditions of a delivery, such as weather or traffic level, are not constant. However, in such cases, the algorithm can be run again, except instead of the restaurant's location, the driver's current location on the road can be input, imitating a new order being placed. Finally, all data collected by the system can be fed back into the algorithm. In turn, it will continue learning, and its performance will improve over time.

Conclusion

Overall, it is concluded in this project that it is indeed possible to accurately predict the time of a food delivery's arrival through machine or deep learning methods. Whether it be through Random Forest, a Neural Network, and likely many other algorithms, it's a possibility that companies can rely on. By doing this project, I've gained a greater understanding of how to implement neural networks and how they work in tensorflow. Furthermore, I've experienced firsthand the deep data mining required to creatively feature engineer. Finally, I've better learned the six modules of MDS and how they relate data science to the scientific principles of the world. As I continue to pursue a career a machine learning, these principles of MDS will certainly be applicable across every project, especially any project related to scientific or mathematical principles.

Potential Areas to Improve

As this was my first major project in Mechanistic Data Science, there is a great deal to learn and improve on for the future. Beside the correlation matrix, my efforts to reduce the dimensionality of the data were not satisfactory due to the existence of binary and categorical variables in the data. In the future, I should do research on dimensionality reduction algorithms specifically catered to these types of variables. Potentially, this could also lead to a reduced order surrogate model, something I was unable to implement for the same reasons. While the neural network was a success, my unfamiliarity with tensorflow as a whole hindered my ability to fine tune the model and optimize it to its fullest potential. Furthermore, there are many other types of neural networks other than FNNs which could produce better results, something to consider going forward. Overall, despite these shortcomings, they serve as future opportunities to learn, and do not take away from the machine learning skills I did learn while completing this project.

References

Liu, W. K., Z. Gan, and M. Fleming, *Mechanistic Data Science*. Springer Cham, 2021: p. 17-18. https://doi.org/10.1007/978-3-030-87832-0

Appendix A: Data Cleaning Code

```
# import pandas as pd
# from datetime import datetime, timedelta
# import numpy as np
# from geopy.distance import geodesic as GD

# def isfloat(num):
#     try:
#         float(num)
#         return True
#     except ValueError:
#         return False

# train = pd.read_csv("train.csv")
# train = train.iloc[:45593,:]
# train= train.replace('NaN', float(np.nan), regex=True)
# train = train.dropna()
# train = train.reset_index(drop=True)

# def clean_data(panda):

#     # takes in a panda as an input, cleans it the way as intended for
#     # this project, and then outputs the result
#     ## Remove id and delivery driver id
#     panda1 = panda.iloc[:,2:]
#     panda = pd.concat([panda["ID"], panda1], axis = 1)

#     ## Format Order Date to Datetime
#     Date = []
```

```
#     for i in range(len(panda['Order_Date'])):
#         Date.append(datetime.strptime(panda['Order_Date'][i], '%m/%d/%Y')) #
panda['Order_Date']=Date

#     ## Geographic Distance

#     panda['Geo_Distance']=np.zeros(len(panda))
#     res_coordinates=panda[['Restaurant_latitude','Restaurant_longitude']].to_numpy()
#     del_coordinates=panda[['Delivery_location_latitude','Delivery_location_longitude']].to_numpy()

#     for i in range(len(train)):
#         panda['Geo_Distance'].loc[i]=GD(res_coordinates[i],del_coordinates[i])

#     panda['Geo_Distance']=panda['Geo_Distance'].astype("str").str.extract('(\d+)') #
panda['Geo_Distance']=panda['Geo_Distance'].astype("float")

#     ## Remove any outliers
#     panda = panda[panda['Geo_Distance'] < 100]
#     panda = panda.reset_index(drop=True)

#     ## Create Time Features
#     panda['Weatherconditions']=panda['Weatherconditions'].str.split(" ", expand=True)[1]
#     panda["month"] = panda.Order_Date.dt.month
#     season = []
#     panda["month_even"] = np.where(panda['month'].isin([2,4,6,8,10,12]),1,0)
#     panda['day_of_week'] = panda.Order_Date.dt.day_of_week.astype(int)
#     panda['weekend'] = np.where(panda['day_of_week'].isin([5,6]),1,0)
#     panda = panda.drop(columns = ["month", "day_of_week"], axis = 1)

#     ## Format order times to timedelta. Allows us to use dt operations easier
```

```
#     Time = []
#     for i in range(len(panda['Time_Orderd'])):
#     Time.append(datetime.strptime(panda['Time_Orderd'][i], '%H:%M:%S'))
#     panda['Time_Orderd']=Time

#     Time = []
#     for i in range(len(panda['Time_Order_picked'])):
#         Time.append(datetime.strptime(panda['Time_Order_picked'][i], '%H:%M:%S')) #
panda['Time_Order_picked']=Time

#     Hour = []
#     Night = []
#     Rush_Hour = []
#     for i in range(len(panda['Time_Order_picked'])):
#         Hour.append(panda['Time_Orderd'][i].hour) #
if Hour[i] in [22,23,24,1,2,3,4]:
#             Night.append(1) #
else:
#             Night.append(0)
#         if Hour[i] in [7,8,9,4,5,6]:
#             Rush_Hour.append(1)
#         else:
#             Rush_Hour.append(0)

#     panda['Night']=Night
#     panda['Rush_Hour']=Rush_Hour

#     ## Format Weather
#     Sunny = []
```

```
#     Fog = []

#     for i in range(len(panda)):
#         weather = panda['Weatherconditions'][i] #
if weather in ['Sunny']:
#             Sunny.append(1) #
else:
#             Sunny.append(0)

#         if weather == 'Fog' or weather == 'Cloudy':
#             Fog.append(1) #
else:
#             Fog.append(0)

#     panda['Sunny'] = Sunny
#     panda['Fog'] = Fog

#     ## Road_Traffic Density
#     Density = []
#     for i in range(len(panda)): #         den =
panda['Road_traffic_density'][i] #         if
den == 'Low ': #             Density.append(1) #
if den == 'Medium ': #
Density.append(2) #         if den == 'High ': #
Density.append(3) #         if den == 'Jam ':
#             Density.append(4)

#     panda['Road_traffic_density'] = Density

#     ## Format Type of Vehicle
```

```
#     Motorcycle = []


#     for i in range(len(panda)): #
vehicle = panda['Type_of_vehicle'][i] #
if vehicle == 'motorcycle ':
#           Motorcycle.append(1) #
else:
#           Motorcycle.append(0)


#     panda['Motorcycle'] = Motorcycle

#     ## Format Festical
#     Festival = []
#     for i in range(len(panda)):
#         if panda['Festival'][i] == 'Yes ':
#           Festival.append(1) #
else:
#           Festival.append(0)

#     panda['Festival'] = Festival

#     ## Format City
#     City = []
#     for i in range(len(panda)):
#         city = panda['City'][i]
#         if city == 'Semi-Urban ':
#           City.append(1)
#         if city == 'Urban ':
```

```
#             City.append(2)
#         if city == 'Metropolitian ':
#             City.append(3)

#     panda['Urban_Density'] = np.zeros(len(panda))
#     panda['Urban_Density'] = City

#     ## Remove the "(min)" from the time take

#     panda['Time_taken(min)']=panda['Time_taken(min)'].str.split(" ", expand=True)[1]

#     ## Drop the columns that we transformed to other columns to avoid redundancy

#     panda = panda.drop(columns=['Weatherconditions','Order_Date', 'Time_Orderd',
#                         'Time_Order_picked', 'Type_of_vehicle', 'City', 'Type_of_order',
#                         'Delivery_location_latitude', 'Delivery_location_longitude',
#                         'Restaurant_latitude', 'Restaurant_longitude', 'weekend'],axis=1)
#     return panda
# train_clean=clean_data(train)
# train_clean.to_csv('train_clean.csv', index=False)
```

Appendix B: Model Code

```
# import pandas as pd
# import numpy as np
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LinearRegression
# from sklearn.decomposition import PCA
# from sklearn.cluster import KMeans
# import matplotlib.pyplot as plt
# from sklearn.preprocessing import MinMaxScaler
```

```
# import seaborn as sns
# from sklearn.preprocessing import StandardScaler
# from sklearn.metrics import r2_score
# from sklearn.ensemble import RandomForestRegressor

# train = pd.read_csv('train_clean.csv')
# train2 = pd.read_csv('train_clean.csv')
# sns.boxplot(x='Urban_Density', y='Time_taken(min)', data=train)
# plt.show()
# train = train[train["Urban_Density"] > 1].reset_index(drop=True)
# sns.boxplot(x='Sunny', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Fog', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='month_even', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Urban_Density', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Road_traffic_density', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Vehicle_condition', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='multiple_deliveries', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Festival', y='Time_taken(min)', data=train)
# plt.show()
# sns.boxplot(x='Delivery_person_Age', y='Time_taken(min)', data=train)
# plt.show()
```

```
# # ## Feature Engineering
# Geo_dist = []
# Geo_dist2 = []
# Road_dense = []
# Delivery_Age = []
# for i in range(len(train['Geo_Distance'])):
#     if (train['Geo_Distance']<10)[i] == True:
#         Geo_dist.append(train['Geo_Distance'][i])
#         Geo_dist2.append(0) #
else:
#         Geo_dist.append(0)
#         Geo_dist2.append(train['Geo_Distance'][i])

#     if train['Delivery_person_Age'][i] >= 30:
#         Delivery_Age.append(1) #
else:
#         Delivery_Age.append(0)


# train['Delivery_person_Age'] = Delivery_Age

# train['Geo_Distance < 10']=Geo_dist
# train['Geo_Distance >= 10']=Geo_dist2
# train['Delivery_person_Ratings'] = train['Delivery_person_Ratings'] >= 4.5 # exp= []
# for i in range(len(train['Vehicle_condition'])):
#     x = int(train['Vehicle_condition'][i])
#     if x == 0: #         exp.append(0) #
else:
#         exp.append(1)
```

```
# train['Vehicle_condition'] = exp

# train.corr().to_csv('Correlation_Matrix.csv')
# train = train.drop("Geo_Distance", axis = 1)

# x=train.drop("Time_taken(min)",axis=1).iloc[:,1:]
# y=train["Time_taken(min)"]

# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=(1/3), random_state=1)



# #%% Preparing data for Kmeans

# ## MUST COMMENT OUT CODE TO DROP GEO_DISTANCE IN LINE 86
# train_clus = pd.concat([train['Geo_Distance'], train['Delivery_person_Ratings'], #
train['Delivery_person_Age']], axis = 1)

# plt.title('Geo_Distance')
# plt.hist(train_clus['Geo_Distance'])
# plt.show()
# plt.title('Delivery Person Age')
# plt.hist(train_clus['Delivery_person_Age'])
# plt.show()
# plt.title('Delivery Person Ratings')
# plt.hist(train_clus['Delivery_person_Ratings'])
# plt.show()

# scaler = StandardScaler()
# train_clus_array = scaler.fit_transform(train_clus)
```

```
# train_clus = pd.DataFrame(train_clus_array, columns=train_clus.columns)

# #%% Running Kmeans algorithm
# inertia=[]
# for i in range(1,12):
#     Kmeans = KMeans(n_clusters=i, tol=1e-6, n_init=100, random_state=12345)
#     Kmeans.fit(train_clus)
#     inertia.append(Kmeans.inertia_)

# plt.title('Food Delivery Objective Cost of Kmeans vs # of clusters')
# plt.plot(range(1,12), inertia)
# plt.show()
# #%% Do 4 Clusters

# Kmeans = KMeans(n_clusters=4, tol=1e-6, n_init=100, random_state=12345)
# Kmeans.fit(train_clus)
# train_clus['clusters'] = Kmeans.labels_
# groups = train_clus.groupby('clusters')

# fig = plt.figure()
# ax = plt.axes(projection='3d') #
plt.title("Clusters Visualized") #
for name, group in groups:
#                                         ax.scatter3D(group["Delivery_person_Ratings"],
group['Delivery_person_Age'], group["Geo_Distance"], label = name)

# ax.set_ylabel('Geo_Distance')
# ax.set_zlabel('Delivery_Person_Age')
# ax.set_xlabel('Delivery_Person_Ratings')
```

```
# # ax.set_ylim(16,53)
# # ax.set_zlim(0.45, 0.51)
# # ax.set_xlim(10,17.5)


# #%% Do Random Forest Regression
# rf = RandomForestRegressor(max_depth=11, n_estimators=500,
#                            random_state=317, max_features=8).fit(x_train, y_train)
# y_pred = rf.predict(x_test)
# Accuracy=r2_score(y_test,y_pred)*100
# print(" Accuracy of the model is %.2f" %Accuracy)

# plt.scatter(y_test, y_pred, marker = '.')
# plt.plot([7,20,30,40,50,55], [7,20,30,40,50,55], color = "red")
# plt.xlabel('True Values')
# plt.ylabel('Predictions')
# plt.title('Predictions vs True Values, Random Forest')
# plt.show()

# plt.scatter(y_test, y_pred-y_test, marker = '.')
# plt.xlabel('y')
# plt.ylabel('residuals')
# plt.title('Residuals')
# plt.show()

# #%% Plot the random forest regression
# train_sizes, train_score, test_score = learning_curve(rf, x_train,
y_train, n_jobs=1,train_sizes=np.linspace(0.1,1,5)) # train_score_mean = np.mean(train_score,
1)
```

```
# test_score_mean = np.mean(test_score, 1)
# plt.plot(train_sizes, train_score_mean, 'rs--', label='Training Score')
# plt.plot(train_sizes, test_score_mean, 'go-', label='Cross Validation Score')
# plt.legend(loc='lower right')
# plt.show()

# #%% FFNN

# import numpy as np
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Dense, Dropout
# from tensorflow.keras.optimizers import Adam
# from tensorflow.keras.models import load_model, save_model
# import matplotlib.pyplot as plt
# from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

# xnump = x_train.to_numpy().astype('float32')
# xnump2 = x_test.to_numpy().astype('float32')
# ynump = y_train.to_numpy().astype('float32')
# # x_retrain = xnump.reshape((len(xnump), 1))
# ynump = ynump.reshape((len(ynump), 1))

# scaler_X = MinMaxScaler()
# scaler_y = MinMaxScaler()
# X = scaler_X.fit_transform(xnump)
# y = scaler_y.fit_transform(ynump)

# I followed a similar format to that of the diamond example as I found it worked best #
def NN():
```

```
#     model=Sequential()
#     model.add(Dense(input_dim=15,activation="relu",units=25))
#     model.add(Dense(input_dim=15,activation="relu",units=25))
#     model.add(Dense(kernel_initializer="normal",
#     activation="relu", units=25))
#     model.add(Dense(kernel_initializer="normal", units=1))
#     model.compile(loss="mean_squared_error", optimizer="adam") #
return model



#
################################################################################
########

# FNN = KerasRegressor(build_fn=NN, epochs=30,
# batch_size=5)
# FNN.fit(xnump, ynump)
# y_pred = FNN.predict(xnump2)

# Accuracy=r2_score(y_test.reset_index(drop=True) ,y_pred)*100
# print(" Accuracy of the model is %.2f" %Accuracy)

# plt.scatter(y_test, y_pred)
# plt.plot([7,20,30,40,50,55], [7,20,30,40,50,55], color = "red")
# plt.xlabel('True Value of Y')
# plt.ylabel('Predictions')
# plt.title('Predictions vs True Values, FNN')
# plt.show()
```

```
# #%% Perform PCA on data

# x=train2.drop("Time_taken(min)",axis=1).iloc[:,1:]
# y=train2["Time_taken(min)"]

# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=(1/3), random_state=1)

# x_cont = pd.concat([x_train['Delivery_person_Age'], x_train['Delivery_person_Ratings'],
#                     x_train['Geo_Distance'], x_train['Urban_Density'], x_train['Road_traffic_density'],  #
x_train['Vehicle_condition']], axis = 1)

# x_cont_test = pd.concat([x_test['Delivery_person_Age'], x_test['Delivery_person_Ratings'],
#                     x_test['Geo_Distance'], x_test['Urban_Density'], x_test['Road_traffic_density'],  #
x_test['Vehicle_condition']], axis = 1)

# sc = StandardScaler()

# x_cont_train = sc.fit_transform(x_cont)
# x_cont_test = sc.transform(x_cont_test)
# pca = PCA(n_components = 6)

# x_pca_train = pca.fit_transform(x_cont_train)
# x_pca_test = pca.transform(x_cont_test)

# PC_values = np.arange(pca.n_components_) + 1

# plt.plot(PC_values, pca.explained_variance_ratio_, 'o-', linewidth=2, color='blue') #
plt.title('Scree Plot')
# plt.xlabel('Principal Component')
```

```
# plt.ylabel('Variance Explained')
```

```
# plt.show()
```