DoorDash Delivery Duration Prediction

DoorDash is an American company that operates an online food ordering and food delivery platform. They show the expeted time of delivery. It is very important for DoorDash to get this right, as it has a big impact on comsumer experience.

In this project, I am going to build a model to predict the estimated time of delivery.

For a given delivery, one must predict the total delivery duration (in seconds), i.e., the time taken from

START : the time customer submits the order (created_at) to

END : When the order will be delivered to the customer (actual_delivery_time)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
historical_data = pd.read_csv("/content/drive/MyDrive/historical_data.csv")
historical_data.head(5)
```

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	orc
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	1845	american	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	5477	mexican	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	5477	NaN	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	5477	NaN	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	5477	NaN	

```
historical_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 16 columns):
 #   Column                                        Non-Null Count   Dtype
---  ------                                        --------------   -----
 0   market_id                                     196441 non-null  float64
 1   created_at                                    197428 non-null  object
 2   actual_delivery_time                          197421 non-null  object
```

```
 3   store_id                                      197428 non-null  int64
 4   store_primary_category                        192668 non-null  object
 5   order_protocol                                196433 non-null  float64
 6   total_items                                   197428 non-null  int64
 7   subtotal                                      197428 non-null  int64
 8   num_distinct_items                            197428 non-null  int64
 9   min_item_price                                197428 non-null  int64
 10  max_item_price                                197428 non-null  int64
 11  total_onshift_dashers                         181166 non-null  float64
 12  total_busy_dashers                            181166 non-null  float64
 13  total_outstanding_orders                      181166 non-null  float64
 14  estimated_order_place_duration                197428 non-null  int64
 15  estimated_store_to_consumer_driving_duration  196902 non-null  float64
dtypes: float64(6), int64(7), object(3)
memory usage: 24.1+ MB
```

Looks like, the "created_at" and "actual_delivery_time" being date and time data type are set as object. We need to change the data type, so as to get the desired results.

```
historical_data['created_at'] = pd.to_datetime(historical_data['created_at'])
historical_data['actual_delivery_time'] = pd.to_datetime(historical_data['actual_delivery_
```

Feature Creation

```
#create target variable for regression
from datetime import datetime
historical_data['actual_total_delivery_duration'] = historical_data['actual_delivery_time'
```

```
historical_data['busy_dashers_ratio']= historical_data['total_busy_dashers']/historical_da
```

```
historical_data['estimated_non_prep_duration']= historical_data['estimated_store_to_consum
```

Data Preperation

```
historical_data['market_id'].nunique()
```

```
6
```

```
historical_data['store_id'].nunique()
```

```
6743
```

```
historical_data['order_protocol'].nunique()
```

```
7
```

```
#dummies for order protocol
order_protocol_dummy = pd.get_dummies(historical_data['order_protocol'])
order_protocol_dummy = order_protocol_dummy.add_prefix('order_protocol_')
```

```
#dummies for market id
market_id_dummy = pd.get_dummies(historical_data['market_id'])
market_id_dummy = market_id_dummy.add_prefix('market_id_')
market_id_dummy.head()
```

	market_id_1.0	market_id_2.0	market_id_3.0	market_id_4.0	market_id_5.0	market_
0	1	0	0	0	0	
1	0	1	0	0	0	
2	0	0	1	0	0	
3	0	0	1	0	0	
4	0	0	1	0	0	

```
#dictionary with most repeated categories of each store to fill null rows where it is poss
store_id_unique = historical_data['store_id'].unique().tolist()
store_id_and_category = {store_id: historical_data[historical_data.store_id == store_id].s
```

```
def fill(store_id):
  """Return primary store category from the dictionary"""
  try:
    return store_id_and_category[store_id].values[0]
  except:
    return np.nan

#fill null values
historical_data["nan_free_store_primary_category"] = historical_data.store_id.apply(fill)
```

```
#dummies for store primary category
store_primary_category_dummy = pd.get_dummies(historical_data.nan_free_store_primary_categ
store_primary_category_dummy = store_primary_category_dummy.add_prefix('category_')
store_primary_category_dummy.head()
```

```
train = historical_data.drop(columns = ['created_at', 'market_id', 'store_id','store_prima
train.head(5)
```

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_ite
0	1.0	4	3441	4	557	
1	2.0	1	1900	1	1400	
2	1.0	1	1900	1	1900	
3	1.0	6	6900	5	600	
4	1.0	3	3900	3	1100	

```
train = pd.concat([train, order_protocol_dummy, market_id_dummy, store_primary_category_du
train.head()
```

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_ite
0	1.0	4	3441	4	557	
1	2.0	1	1900	1	1400	
2	1.0	1	1900	1	1900	
3	1.0	6	6900	5	600	
4	1.0	3	3900	3	1100	

5 rows × 100 columns

```
#align dtype
train = train.astype("float32")
train.head()
```

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_ite
0	1.0	4.0	3441.0	4.0	557.0	
1	2.0	1.0	1900.0	1.0	1400.0	
2	1.0	1.0	1900.0	1.0	1900.0	
3	1.0	6.0	6900.0	5.0	600.0	
4	1.0	3.0	3900.0	3.0	1100.0	

5 rows × 100 columns

```
train.describe()
```

	order_protocol	total_items	subtotal	num_distinct_items	min_item_pri
count	196433.000000	197428.000000	197428.000000	197428.000000	197428.0000
mean	2.882352	3.196391	2682.331543	2.670791	686.2185
std	1.503771	2.666546	1823.093750	1.630255	522.0386
min	1.000000	1.000000	0.000000	1.000000	-86.0000
25%	1.000000	2.000000	1400.000000	1.000000	299.0000
50%	3.000000	3.000000	2200.000000	2.000000	595.0000
75%	4.000000	4.000000	3395.000000	3.000000	949.0000
max	7.000000	411.000000	27100.000000	20.000000	14700.0000

8 rows × 100 columns

```
train['busy_dashers_ratio'].describe()
```

```
count    1.775900e+05
mean              NaN
std               NaN
min              -inf
25%      8.269231e-01
50%      9.622642e-01
75%      1.000000e+00
max               inf
Name: busy_dashers_ratio, dtype: float64
```

```
#check infinity values with using numpy isfinite() function
np.where(np.any(~np.isfinite(train),axis=0) == True)
```

```
(array([ 0,  6,  7,  8, 10, 11, 12]),)
```

```
#replace inf values with nan to drop all nans
train.replace([np.inf, -np.inf], np.nan, inplace = True)
train.dropna(inplace = True)
```

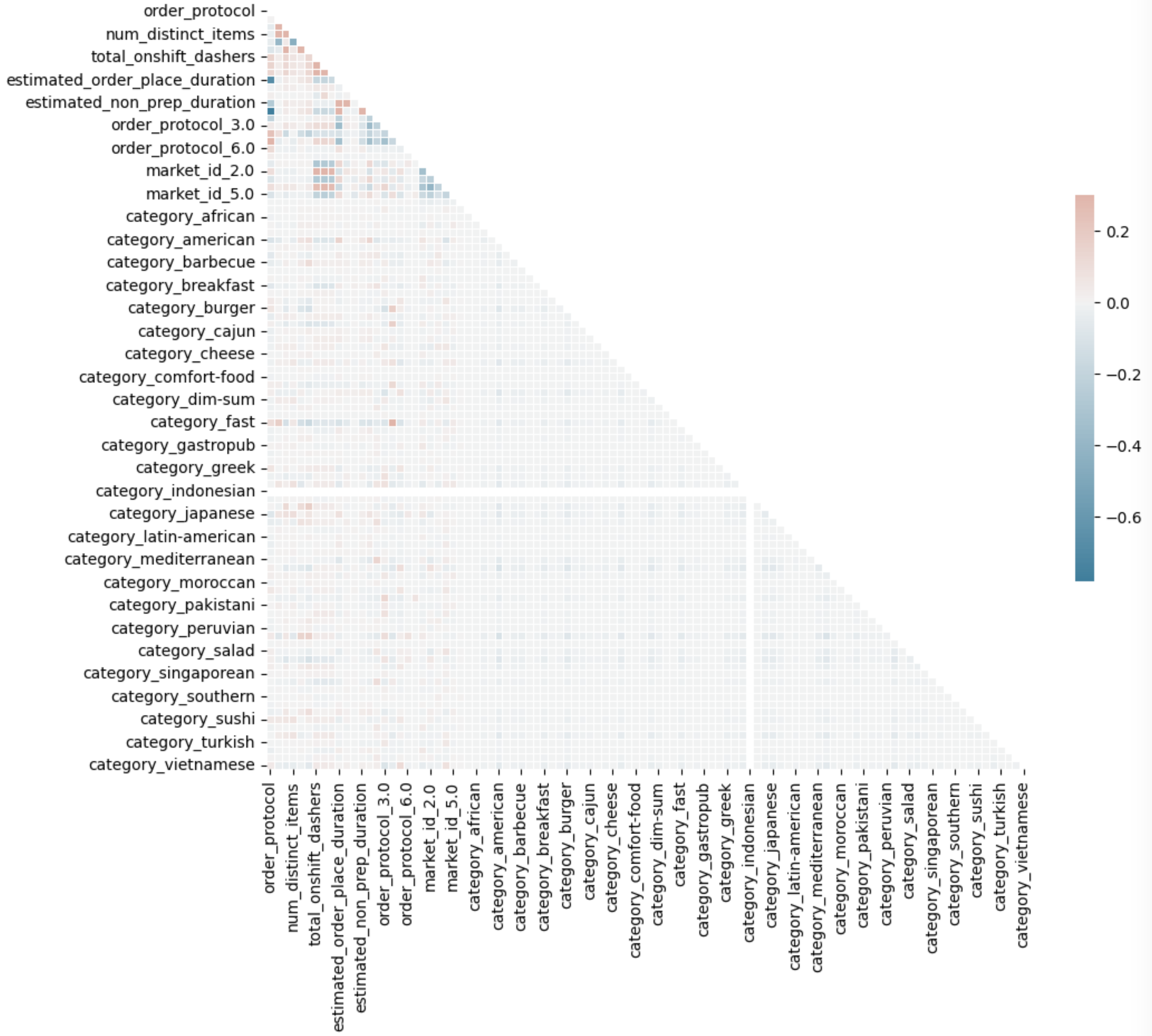
```
train.shape
```

```
(176173, 100)
```

```
corr = train.corr()
mask = np.triu(np.ones_like(corr, dtype = bool))
```

```
f, ax = plt.subplots(figsize=(11,9))
cmap = sns.diverging_palette(230,20, as_cmap = True)
sns.heatmap(corr, mask = mask, cmap = cmap, vmax = 0.3, center = 0,
            square = True, linewidths = 0.5, cbar_kws={'shrink': 0.5})
```

```
<Axes: >
```




```
train['category_indonesian'].describe()
```

```
count    176173.0
mean          0.0
```

```
std           0.0
min           0.0
25%           0.0
50%           0.0
75%           0.0
max           0.0
Name: category_indonesian, dtype: float64
```

```
def get_redundant_pairs(df):
  """Get diagonal and lower triangular pairs of correlation matrix"""
  pairs_to_drop = set()
  cols = df.columns
  for i in range(0, df.shape[1]):
    for j in range(0, i+1):
      pairs_to_drop.add((cols[i],cols[j]))
  return pairs_to_drop
```

```
def get_top_abs_correlations(df,n=5):
  """Sort correlations in the descending order and return n highest results"""
  au_corr = df.corr().abs().unstack()
  labels_to_drop = get_redundant_pairs(df)
  au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
  return au_corr[0:n]
```

```
print('Top Absolute Correlation')
print(get_top_abs_correlations(train,20))
```

```
Top Absolute Correlation
total_onshift_dashers                        total_busy_dashers              0.94
                                             total_outstanding_orders        0.93
total_busy_dashers                           total_outstanding_orders        0.93
estimated_store_to_consumer_driving_duration estimated_non_prep_duration     0.92
estimated_order_place_duration               order_protocol_1.0              0.90
order_protocol                               order_protocol_1.0              0.78
                                             order_protocol_5.0              0.76
total_items                                  num_distinct_items              0.75
order_protocol                               estimated_order_place_duration  0.68
subtotal                                     num_distinct_items              0.68
total_items                                  subtotal                        0.55
min_item_price                               max_item_price                  0.54
subtotal                                     max_item_price                  0.50
order_protocol_4.0                           category_fast                   0.49
num_distinct_items                           min_item_price                  0.44
market_id_2.0                                market_id_4.0                   0.40
total_items                                  min_item_price                  0.38
order_protocol_1.0                           order_protocol_3.0              0.37
estimated_order_place_duration               order_protocol_3.0              0.36
                                             estimated_non_prep_duration     0.36
dtype: float64
```

```
train.columns
```

```
Index(['order_protocol', 'total_items', 'subtotal', 'num_distinct_items',
       'min_item_price', 'max_item_price', 'total_onshift_dashers',
```

```
       'total_busy_dashers', 'total_outstanding_orders',
       'estimated_order_place_duration',
       'estimated_store_to_consumer_driving_duration', 'busy_dashers_ratio',
       'estimated_non_prep_duration', 'order_protocol_1.0',
       'order_protocol_2.0', 'order_protocol_3.0', 'order_protocol_4.0',
       'order_protocol_5.0', 'order_protocol_6.0', 'order_protocol_7.0',
       'market_id_1.0', 'market_id_2.0', 'market_id_3.0', 'market_id_4.0',
       'market_id_5.0', 'market_id_6.0', 'category_afghan', 'category_african',
       'category_alcohol', 'category_alcohol-plus-food', 'category_american',
       'category_argentine', 'category_asian', 'category_barbecue',
       'category_belgian', 'category_brazilian', 'category_breakfast',
       'category_british', 'category_bubble-tea', 'category_burger',
       'category_burmese', 'category_cafe', 'category_cajun',
       'category_caribbean', 'category_catering', 'category_cheese',
       'category_chinese', 'category_chocolate', 'category_comfort-food',
       'category_convenience-store', 'category_dessert', 'category_dim-sum',
       'category_ethiopian', 'category_european', 'category_fast',
       'category_filipino', 'category_french', 'category_gastropub',
       'category_german', 'category_gluten-free', 'category_greek',
       'category_hawaiian', 'category_indian', 'category_indonesian',
       'category_irish', 'category_italian', 'category_japanese',
       'category_korean', 'category_kosher', 'category_latin-american',
       'category_lebanese', 'category_malaysian', 'category_mediterranean',
       'category_mexican', 'category_middle-eastern', 'category_moroccan',
       'category_nepalese', 'category_other', 'category_pakistani',
       'category_pasta', 'category_persian', 'category_peruvian',
       'category_pizza', 'category_russian', 'category_salad',
       'category_sandwich', 'category_seafood', 'category_singaporean',
       'category_smoothie', 'category_soup', 'category_southern',
       'category_spanish', 'category_steak', 'category_sushi',
       'category_tapas', 'category_thai', 'category_turkish', 'category_vegan',
       'category_vegetarian', 'category_vietnamese'],
      dtype='object')
```

```
train = historical_data.drop(columns =['created_at','market_id','store_id','store_primary_
```

```
train = pd.concat([train, order_protocol_dummy, store_primary_category_dummy], axis = 1)
train = train.drop(columns = ['total_onshift_dashers','total_busy_dashers','category_indon

train = train.astype("float32")

train.replace([np.inf, -np.inf], np.nan, inplace = True)
train.dropna(inplace = True)

train.head()
```

	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_c
0	4.0	3441.0	4.0	557.0	1239.0	

```
train.shape
```

```
(177077, 89)
```

```
def get_top_abs_correlations(df,n=5):
  """Sort correlations in the descending order and return n highest results"""
  au_corr = df.corr().abs().unstack()
  labels_to_drop = get_redundant_pairs(df)
  au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
  return au_corr[0:n]
print('Top Absolute Correlation')
print(get_top_abs_correlations(train,20))
```

```
Top Absolute Correlation
estimated_order_place_duration  order_protocol_1.0    0.897649
total_items                     num_distinct_items    0.758153
subtotal                        num_distinct_items    0.682892
total_items                     subtotal              0.557181
min_item_price                  max_item_price        0.541239
subtotal                        max_item_price        0.507949
order_protocol_4.0              category_fast         0.489986
num_distinct_items              min_item_price        0.446735
total_items                     min_item_price        0.389280
order_protocol_1.0              order_protocol_3.0    0.373582
estimated_order_place_duration  order_protocol_3.0    0.364171
order_protocol_1.0              order_protocol_5.0    0.342341
estimated_order_place_duration  order_protocol_5.0    0.333288
order_protocol_3.0              order_protocol_5.0    0.332530
order_protocol_1.0              order_protocol_2.0    0.226904
estimated_order_place_duration  order_protocol_2.0    0.221216
order_protocol_2.0              order_protocol_3.0    0.220401
max_item_price                  order_protocol_4.0    0.215635
order_protocol_1.0              order_protocol_4.0    0.202153
order_protocol_2.0              order_protocol_5.0    0.201970
dtype: float64
```

```
train = historical_data.drop(columns = ['created_at','market_id','store_id','store_primary_
                                        'nan_free_store_primary_category','order_protocol'

train = pd.concat([train, store_primary_category_dummy], axis=1)
train = train.drop(columns = ['total_onshift_dashers','total_busy_dashers','category_indon

train = train.astype('float32')

train.replace([np.inf, -np.inf], np.nan, inplace = True)
train.dropna(inplace= True)
train.head()


print("Top Absolute Correlations")
print(get_top_abs_correlations(train, 20))
```

```
Top Absolute Correlations
total_items                   num_distinct_items              0.758153
subtotal                      num_distinct_items              0.682892
total_items                   subtotal                        0.557181
min_item_price                max_item_price                  0.541239
subtotal                      max_item_price                  0.507949
num_distinct_items            min_item_price                  0.446735
total_items                   min_item_price                  0.389280
total_outstanding_orders      estimated_order_place_duration  0.171010
total_items                   category_fast                   0.170968
max_item_price                category_italian                0.169774
                              category_fast                   0.166186
                              category_pizza                  0.157573
estimated_order_place_duration  category_american             0.150171
min_item_price                category_pizza                  0.149579
subtotal                      total_outstanding_orders        0.131141
max_item_price                total_outstanding_orders        0.129543
min_item_price                category_fast                   0.127986
max_item_price                category_burger                 0.120936
subtotal                      category_fast                   0.119187
                              category_italian                0.118715
dtype: float64
```

```
#Simplifying model and increasing accuracy by using new variables as predcitors

train['percent_distinct_item_of_total'] = train['num_distinct_items']/train['total_items']
train['avg_price_per_item'] = train['subtotal']/train['total_items']
train.drop(columns = ['num_distinct_items','subtotal'], inplace = True)

print('Top Absolute Correlation')
print(get_top_abs_correlations(train,20))
```

```
Top Absolute Correlation
min_item_price                avg_price_per_item              0.860581
max_item_price                avg_price_per_item              0.770380
min_item_price                max_item_price                  0.541239
total_items                   percent_distinct_item_of_total  0.445763
                              min_item_price                  0.389280
                              avg_price_per_item              0.310765
percent_distinct_item_of_total  avg_price_per_item            0.226718
category_pizza                avg_price_per_item              0.225507
max_item_price                percent_distinct_item_of_total  0.178017
category_fast                 avg_price_per_item              0.175971
min_item_price                percent_distinct_item_of_total  0.173534
total_outstanding_orders      estimated_order_place_duration  0.171010
total_items                   category_fast                   0.170968
max_item_price                category_italian                0.169774
                              category_fast                   0.166186
category_italian              avg_price_per_item              0.158193
max_item_price                category_pizza                  0.157573
category_fast                 percent_distinct_item_of_total  0.153581
estimated_order_place_duration  category_american             0.150171
min_item_price                category_pizza                  0.149579
dtype: float64
```

```
train['price_range']=train['max_item_price']-train['min_item_price']
train.drop(columns = ['max_item_price','min_item_price'], inplace = True)
```

```
print('Top Absolute Correlation')
print(get_top_abs_correlations(train,20))
```

```
Top Absolute Correlation
total_items                     percent_distinct_item_of_total    0.445763
                                price_range                       0.333309
                                avg_price_per_item                0.310765
percent_distinct_item_of_total  avg_price_per_item                0.226718
category_pizza                  avg_price_per_item                0.225507
category_fast                   avg_price_per_item                0.175971
total_outstanding_orders        estimated_order_place_duration    0.171010
total_items                     category_fast                     0.170968
category_italian                avg_price_per_item                0.158193
category_fast                   percent_distinct_item_of_total    0.153581
estimated_order_place_duration  category_american                 0.150171
category_american               category_pizza                    0.106997
estimated_order_place_duration  category_fast                     0.106748
category_american               category_mexican                  0.106458
category_burger                 avg_price_per_item                0.104433
total_outstanding_orders        price_range                       0.100249
category_mexican                category_pizza                    0.097821
total_outstanding_orders        avg_price_per_item                0.088976
                                category_breakfast                0.088308
category_american               category_burger                   0.084004
dtype: float64
```

```
train.shape
```

```
(177077, 81)
```

Multicollinearity is when one predictor variable in a multiple regression model can be predicted from the other variables. Which makes it harder to interpret your model and may cause other factors like overfitting

We will use variable inflation factor (VIF) for that and will remove the features that have VIF > 20

```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
```

```
def compute_vif(features):
  """Compute VIF score using vif() function"""
  vif_data = pd.DataFrame()
  vif_data['feature']= features
  vif_data['VIF'] = [vif(train['features'].values, i) for i in range(len(features))]
  return vif_data.sort_values(by=['VIF']).reset_index(drop=True)
```

```
features = train.columns.to_list()
features
```

```
['total_items',
 'total_outstanding_orders',
 'estimated_order_place_duration',
 'estimated_store_to_consumer_driving_duration',
 'busy_dashers_ratio',
```

```
       'category_afghan',
       'category_african',
       'category_alcohol',
       'category_alcohol-plus-food',
       'category_american',
       'category_argentine',
       'category_asian',
       'category_barbecue',
       'category_belgian',
       'category_brazilian',
       'category_breakfast',
       'category_british',
       'category_bubble-tea',
       'category_burger',
       'category_burmese',
       'category_cafe',
       'category_cajun',
       'category_caribbean',
       'category_catering',
       'category_cheese',
       'category_chinese',
       'category_chocolate',
       'category_comfort-food',
       'category_convenience-store',
       'category_dessert',
       'category_dim-sum',
       'category_ethiopian',
       'category_european',
       'category_fast',
       'category_filipino',
       'category_french',
       'category_gastropub',
       'category_german',
       'category_gluten-free',
       'category_greek',
       'category_hawaiian',
       'category_indian',
       'category_irish',
       'category_italian',
       'category_japanese',
       'category_korean',
       'category_kosher',
       'category_latin-american',
       'category_lebanese',
       'category_malaysian',
       'category_mediterranean',
       'category_mexican',
       'category_middle-eastern',
       'category_moroccan',
       'category_nepalese',
       'category_other',
       'category_pakistani',
```

```
vif_data = compute_vif(features)
vif_data
```

	feature	VIF
0	category_alcohol-plus-food	1.000370
1	category_chocolate	1.000489
2	category_belgian	1.000749
3	category_russian	1.003226
4	category_african	1.003820
...	...	...
76	busy_dashers_ratio	6.369592
77	category_american	7.033601
78	estimated_store_to_consumer_driving_duration	7.210810
79	estimated_order_place_duration	13.472033
80	percent_distinct_item_of_total	30.336791

81 rows × 2 columns

```
multicollinearity = True

while multicollinearity:
    highest_vif_feature = vif_data['feature'].values.tolist()[-1]
    print('I will remove', highest_vif_feature)
    feature.remove(highest_vif_feature)
    vif_data = compute_vif(features)
    multicollinearity = False if len (vif_data[vif_data.VIF > 20 ]) == 0 else True

selected_features = vif_data['feature'].value.tolist()
vif_data
```

```
selected_features = train.drop(['percent_distinct_item_of_total'], axis =1)
```

	feature	VIF
0	category_alcohol-plus-food	1.000222
1	category_chocolate	1.000362
2	category_belgian	1.000451
3	category_russian	1.002016
4	category_gluten-free	1.002364
...	...	...
75	category_american	4.505927
76	avg_price_per_item	5.958957
77	busy_dashers_ratio	6.357882
78	estimated_store_to_consumer_driving_duration	7.192609
79	estimated_order_place_duration	13.339134

80 rows × 2 columns

Feature Selection

Feature selection works to reduce the dimension of the dataset and getting rid of the features that do not have a significant effect on the model. Also it helps our algorithm to work faster

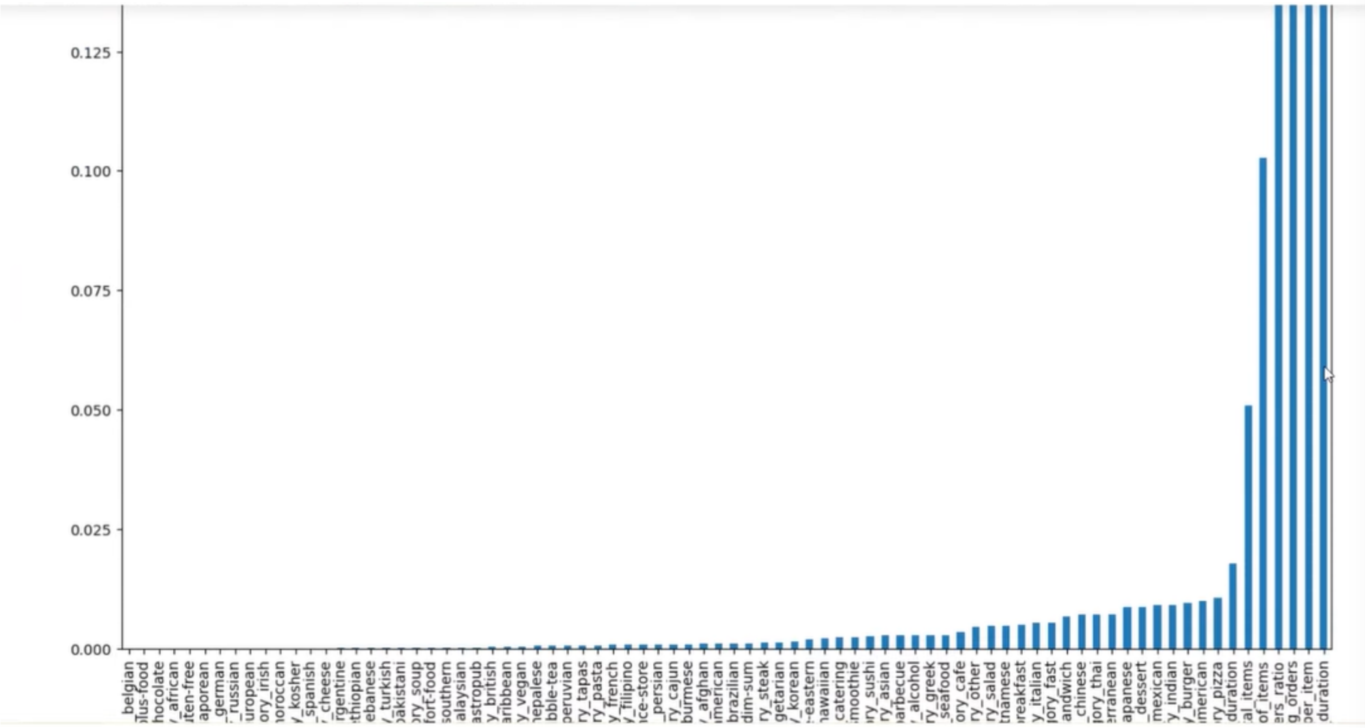
```
#Random forest with Gini importance to measure the importance of each feature. Gini import

from sklearn.ensemble import RandomForestRegressor as rf
from sklearn.model_selection import train_test_split

#selected features are selected in multicollinearity check part

x = train[selected_features]
y = train["actual_total_delivery_duration"]
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state = 4
```

```
feature_names = [f"feature {i}" for i in range ((x.shape[1]))]
forest = rf(random_state = 42)
forest.fir(x_train, y_train)
feats = {} #a dict to hold feature_name: feature_importance
for feature, importance in zip (x.columns, forest.feature_importances_):
  feats[feature]=importance # add the name/value pair
importance = pd.DataFrame.from_dict(feats, orient = 'index').rename(columns={0: 'Gini-impo
importances.sort_values(by='Gini-importance').plot(kind='bar', rot = 90, figsize = (15,12)
plt.show()
```



Inference

The graph shows 'busy_dashers_ratio', 'total_outstanding_orders', and 'avg_price_per_item' are important features for our model and also many of our features have a slight effect on our model.

```
importance.sort_values(by = 'Gini-importance')[-35].index.tolist()
```

```
['category_middle-eastern',
 'category_hawaiian',
 'category_catering',
 'category_smoothie',
 'category_sushi',
 'category_asian',
 'category_barbecue',
 'category_alcohol',
 'category_greek',
 'category_seafood',
 'category_cafe',
 'category_other',
 'category_salad',
 'category_vietnamese',
 'category_breakfast',
 'category_italian',
 'category_fast',
 'category_sandwich',
 'category_chinese',
 'category_thai',
 'category_mediterranean',
 'category_japanese',
 'category_dessert',
 'category_mexican',
 'category_indian',
 'category_burger',
 'category_american',
 'category_pizza',
 'estimated_order_place_duration',
 'total_items',
```

PCA (Principle component analysis)

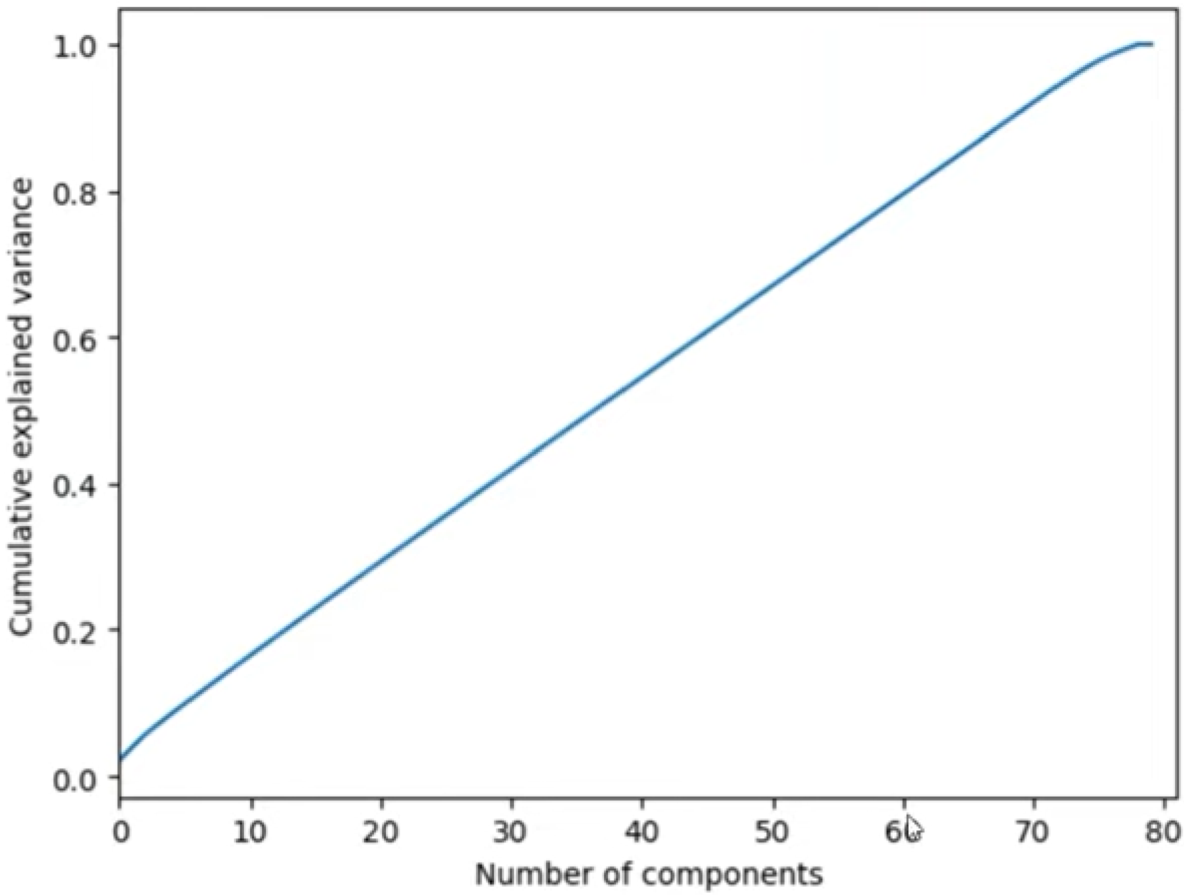
It is alos a dimension reduction technique for regression tasks. Alos it is effective to eliminate multicollinearity too. Using PCA we can analyze how many features we have to use to explain any percentage of our data

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

x_train = x_train.values
x_train = np.asarray(x_train)

#finding normalised array of x_train
x_std = StandardScaler().fit_transform(x_train)
pca = PCA().fit(x_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,81,1)
plt.xlable('Number of Components')
```

```
plt.ylable('Cumulative explained variance')
plt.show()
```



The result tells us that by using 60% of the features the dataset can be explained by 80%

Why are we using a scaler?

Developing a model to make further interpretations scaling is important since it will be hard to compare the different scaled features When the values of the features are closed that will be good for the model.

We will be using 2 different methods:

1. standard scaler : Aim is to make the mean zero, that's how our model performs best
2. min/max scaler : It will sale our model between 0 and 1

```
#Standard Scaler

from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
def scale(scaler, x, y):
  """Apply the selected scaler to features and target variables"""
  x_scaler = scaler
  x_scaler.fit(x=x, y=y)
```

```
  x_scaled = x_scaler.transform(x)
  y_scaler = scaler
  y_scaler.fit(y.values.reshape(-1, 1))
  y_scaled = y_scaler.transform(y.values.reshape(-1,1))

  return x_scaled, y_scaled, x_scaler, y_scaler
```

```
#example on how to use it
x_scaled, y_scaled, x_scaler, y_scaler = scale(MinMaxScaler(), x, y)

x_train_scaled, x_test_scaled, y_train_scaled, y_test_scaled = train_test_split(x_scaled,
```

```
from sklearn.metrics import mean_squared_error

def rmse_with_inv_transform(scaler, y_test, y_pred_scaled, model_name):
  """Convert scaled error to actual error"""
  y_predict =  scaler.inverse_transform(y_pred_scaled.reshape(-1,1))
  #Return RMSE with squared False
  rmse_error = mean_squared_error(y_test, y_predict[:,0], squared = False)
  print ("Error = "'{}'.format(rmse_error)+" in " + model_name)

  return rmse_error, y_predict
```

We have eliminated multicollinearity and performed feature selection to reduce the dimension of the dataset and we got rid of the insignificant features of the model.

Classical Machine Learning

1. We will apply 6 different models that will help us to find the best performanced model.
2. We will select 4 feature set sizes. These features are selected by the Gini's importance. That's how we will measure the effect of using sorted datasets.
3. And also we will use 3 different scalers which are Standard, Min-Max and No Scaler. So, as a result we will have many different options.

```
from xgboost import XGBRegressor
from lightgbm import LGMRegressor
from sklearn.neural_network import MLPRegressor
from sklearn import tree
from sklearn import svm
from sklearn import neighbors
from sklearn import linear_model
```

```
# create a general model which can work with multiple ML models
def make_regression(x_train, x_test, yy_train, y_test, model, model_nam, verbose=True):
```

```
  """Apply selected regression model to data and measure error"""
  model.fit(x_train, y_train)
  y_predict = model.predict(x_train)
  train_error = mean_squared_error(y_train, y_predict, squared = False)
  y_predict = model.predict(x_test)
  test_error = mean_squared_error(y_test, y_predict, squared = False)
  if verbose:
    print("Train error : "'{}'.format(train_error)+" in "+ model_name)
    print("Test error : "'{}'.format(test_error)+ " in "+ model_name)
  trained_model = model
  return trained_model, y_predict, train_error, test_error
```

```
from lightgbm.sklearn import LGBMRegressor
pred_dict = {
    "regression_model" : [],
    "feature_Set" : [],
    "scaler_name" : [],
    "RMSE" : []
}

regression_model = {
    "Ridge" : linear_model.Ridge(),
    "DecisionTree" : tree.DecisionTreeRegressor(max_depth = 6),
    "RandomForest" : rf(),
    "XGBOOST" : XGBRegressor(),
    "LGBM" : LGBMRegressor(),
    "MLP" : MLPRegressor()
}

feature_sets = {
    " full datasets" : x.columns.to_list(),
    "selectd_features_40" : importance.sort_values(by = "Gini Importance")[-40:].index.tol
    "selected_feature_20" : importance.sort_values(by = "Gini importance")[-20:].index.tol
    "selected_feature_10" : importance.sort_values(by = "Gini importance")[-10:].index.tol
}

scalers = {
    "Standard Scaler" : StandardScaler(),
    "MinMax Scaler" : MinMaxScaler(),
    "Notcale" : None
}

# examine the error for each combination

for feature_set_name in feature_sets.keys():
  feature_set = feature_Stes[feature_set_name]
  for scaler_name in scaler.keys():
    print(f"--------- scaled with {scaler_name}--------- incluede columns are {feature_set
    print("")
    for model_name in regression_models.keys():
      if scaler_name == "NotScale" :
        x = train[feature_set]
        y = train["actual_total_delivery_duration"]
```

```
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_
        make_regression(x_train, y_train, x_test, y_test, regression_models[model_name], m
      else:
        x_scaled, y_scaled. x_scaler, y_scaler = scale(scalers[scaler_name], x[feature_set
        x_train_scaled, x_test_scaled, y_train_scaled, y_test_scaled = train_test_split(
            x_scaled, y_scaled, test_Size = 0.2, random_state = 42
        )
        _, y_predict_scaled, _, _ = make_regression(x_train_scaled, y_train_scaled[:,0],x_
        rmse_error, y_predict = rmse_with_inv_transform(y_scaler, y_test, y_predict_scaled
        pred_dict["regression_model"].append(model_name)
        pred_dict["feature_set"].append(feature_ste_name)
        pred_dict["scaler_name"].append(scaler_name)
        pred_dict["RMSE"].append(rmse_error)
```

```
pred_df = pd.DataFrame(pred_dict)
```

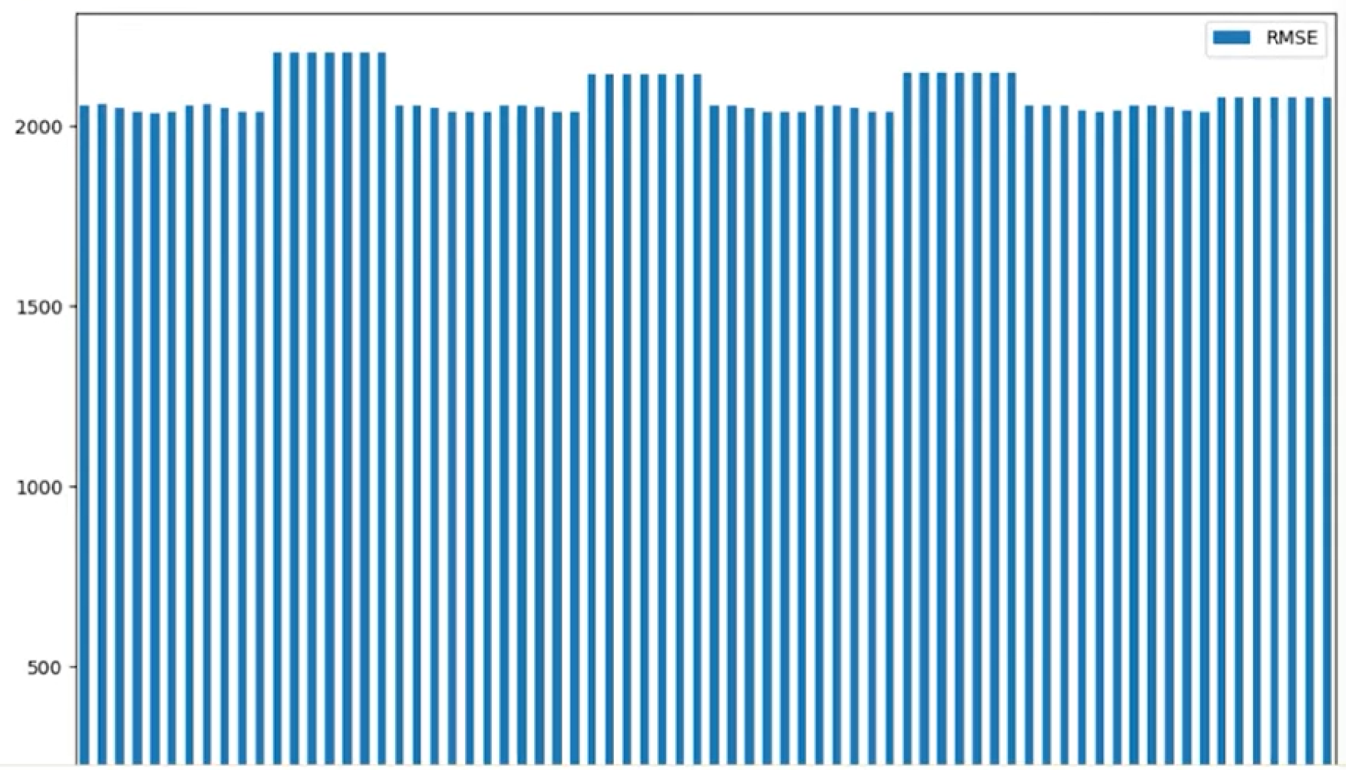
```
pred_df
```

	regression_model	feature_set	scaler_name	RMSE
0	Ridge	full dataset	Standard scaler	2053.698730
1	DecisionTree	full dataset	Standard scaler	2057.247669
2	RandomForest	full dataset	Standard scaler	2048.364098
3	XGBoost	full dataset	Standard scaler	2036.249878
4	LGBM	full dataset	Standard scaler	2033.435581
...	...	...	...	...
67	DecisionTree	selected_features_10	NotScale	2078.785889
68	RandomForest	selected_features_10	NotScale	2078.785889
69	XGBoost	selected_features_10	NotScale	2078.785889
70	LGBM	selected_features_10	NotScale	2078.785889
71	MLP	selected_features_10	NotScale	2078.785889

72 rows × 4 columns

```
pred_df.plot(kind= "bar")
```

<AxesSubplot:>



It seems that we have high errors through all the models, so there is still room for improvement.

Let us change the problem a little bit by predicting prep duration and then we will calculate the actual total delivery duration

```
train["prep_time"]= train["actual_totall_delivery_duration"]- train["estimated_store_to_co
```

```
scalers = {
    "Standard Scaler"  : StandardScaler()
}

feature_set = {
    "selected_features_40" : importance.sort_values(by = 'Gini-importance')[-40:].index.to
}
```

```
for feature_set_name in feature_sets.keys():
  feature_set = feature_sets[feature_set_name]
  for scaler_name in scaler.keys():
    print(f"-----------scaled with {scaler_name}-------------- included columns are {featu
    print("")
    for model_name in regression_models.keys():
      x = train[feature_set].drop(columns = ["estimated_store_to_customer_driving_duration
      y = train["prep_time"]

      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_st
      train_indices = x_train.index
      test_indices = x_test.index
```

```
      x_scaled, y_scaled, x_scaler, y_scaler = scale(scalers[scaler_name], x, y)

      x_train_scaled, y_train_scaled, x_test_scaler, y_test_scaler = train_test_split(x_sc
      _, y_predict_scaled, _, _ = make_regression(x_train_scaled, y_train,scaled[:,0], x_t
      rmse_error, y_predict = rmse_with_inv_transform(y_scaler, y_scaled, y_predict_scaled
      pred_dict["regression_model"].append(model_name)
      pred_dict["feature_set"].append(feature_set_name)
      pred_dict["scaler_name"].append(scaler_name)
      pred_dict["RMSE"].append(rmse_error)
```

```
-----scaled with Standard scaler-------- included columns are selected_features_40

Error = 2055.40771484375 in Ridge
Error = 2063.3402827182467 in DecisionTree
Error = 2049.9123897379004 in RandomForest
Error = 2037.8203125 in XGBoost
Error = 2035.7236370573405 in LGBM
Error = 2037.9390869140625 in MLP
```

Results: According to the above output, LGBM performs better than the rest of the regression models.

```
scalers = {
    "Standard Scaler"  : StandardScaler()
}

feature_set = {
    "selected_features_40" : importance.sort_values(by = 'Gini-importance')[-40:].index.to
}

regression_models ={
    "LGBM" : LGBMRegressor()
}

for feature_set_name in feature_sets.keys():
  feature_set = feature_sets[feature_set_name]
  for scaler_name in scaler.keys():
    print(f"-----------scaled with {scaler_name}-------------- included columns are {featu
    print("")
    for model_name in regression_models.keys():
      x = train[feature_set].drop(columns = ["estimated_store_to_customer_driving_duration
      y = train["prep_time"]

      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_st
      train_indices = x_train.index
      test_indices = x_test.index

      x_scaled, y_scaled, x_scaler, y_scaler = scale(scalers[scaler_name], x, y)

      x_train_scaled, y_train_scaled, x_test_scaler, y_test_scaler = train_test_split(x_sc
      _, y_predict_scaled, _, _ = make_regression(x_train_scaled, y_train,scaled[:,0], x_t
```

```
        rmse_error, y_predict = rmse_with_inv_transform(y_scaler, y_scaled, y_predict_scaled
        pred_dict["regression_model"].append(model_name)
        pred_dict["feature_set"].append(feature_set_name)
        pred_dict["scaler_name"].append(scaler_name)
        pred_dict["RMSE"].append(rmse_error)
```

```
-----scaled with Standard scaler-------- included columns are selected_features_40

Error = 2035.7236370573405 in LGBM
```

Now, let's define a dictionary to choose the best performance model and extract the prep duration prediction

```
pred_values_dict = {
    "actual_total_delivery_duration": train["actual_total_delivery_duration"][test_indice
    "prep_duration_prediction": y_predict[:,0].tolist(),
    "estimated_Store_to_customer_driving_duration": train["estimated_store_to_customer_dr
    "estimated_order_place_duration": train["estimated_order_place_duration"][test_indice
}
```

```
values_df = pd.DataFrame.from_dict(pred_values_dict)
```

```
values_df["sum_total_delivery_duration"] = values_df["prep_duration_prediction"] + values_
```

```
mean_squared_error(values_df["actual_total_delivery_duration"], values_df["sum_total_deliv
```

2035.7236370573405

The error is stil high. Let us try another approach

```
x = values_df[["prep_duration_prediction","estimated_Store_to_customer_driving_duration","
y = values_df[["actual_total_delivery_duration"]]
```

```
regression_model = {
    "Ridge" : linear_model.Ridge(),
    "DecisionTree" : tree.DecisionTreeRegressor(max_depth = 6),
    "RandomForest" : rf(),
    "XGBOOST" : XGBRegressor(),
    "LGBM" : LGBMRegressor(),
    "MLP" : MLPRegressor()
}

for model_name in regression_model.keys():
  _, y_predict, _, _ = make_regression(
      x_train, y_train, x_test, y_test, regression_models[model_name], model_name, verbode
  print("RMSE of:", model_name, mean_Squared_error(y_test, y_predict, squared = False))
```

```
RMSE of: LinearReg 986.6912510303843
RMSE of: Ridge 986.6912510344928
RMSE of: DecisionTree 1235.578088153976
RMSE of: RandomForest 1182.7132515140177
C:\Users\fouzi\anaconda3\lib\site-packages\xgboost\data.py:250: FutureWarning: pandas.Int64Index is deprecated and will be r
emoved from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
RMSE of: XGBoost 1370.412425918564
RMSE of: LGBM 1079.2949179771774
RMSE of: MLP 987.7683741837485
```

As we can see this approach gives better solution as the rates dropped more than half from the previous approach, this can be opt as the official solution of the problem.

Deep Learning for Prediction

```
#dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
tf.random.set_seed(42)
```

```
#Neural Network
def create_model(feature_set_size):

  model = Sequential()
  model.add(Dense(16, input_dim = feature_set_size,activation = "relu"))
  model.add(Dense(1, activation = 'linear'))

  model.compile(optimizer = 'sgd', loss = "mse",
                metrics = [tf.keras.metrics.RootMeanSquaredError()])

  return model
```

```
print(f"-----------scaled with {scaler_name}-------------- included columns are {feature_s
print("")
```