Delivery Duration Prediction

Doordash is a rideshare service where restaurants work with delivery drivers to deliver food to customers door. Regression may be used to help provide more accurate estimated delivery times. In this notebook, there will be some EDA to understand some features, Create new features, impute missing values, perform some regression and classifier models.

Time features

- market_id: A city/region in which DoorDash operates, e.g., Los Angeles, given in the data as an id
- created_at: Timestamp in UTC when the order was submitted by the consumer to DoorDash. (Note this timestamp is in UTC, but in case you need it, the actual timezone of the region was US/Pacific)
- actual_delivery_time: Timestamp in UTC when the order was delivered to the consumer

Store features

- store_id: an id representing the restaurant the order was submitted for
- store_primary_category: cuisine category of the restaurant, e.g., italian, asian
- order_protocol: a store can receive orders from DoorDash through many modes. This field represents an id denoting the protocol

Order features

- total_items: total number of items in the order
- subtotal: total value of the order submitted (in cents)
- num_distinct_items: number of distinct items included in the order
- min_item_price: price of the item with the least cost in the order (in cents)
- max_item_price: price of the item with the highest cost in the order (in cents)

Market Features

- total_onshift_dashers: Number of available dashers who are within 10 miles of the store at the time of order creation
- total_busy_dashers: Subset of above total_onshift_dashers who are currently working on an order
- total_outstanding_orders: Number of orders within 10 miles of this order that are currently being processed.

Predictions from other models

- estimated_order_place_duration: Estimated time for the restaurant to receive the order from DoorDash (in seconds)
- estimated_store_to_consumer_driving_duration: Estimated travel time between store and consumer (in seconds)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image
```

In [2]:
```
df = pd.read_csv(r'C:\Users\stanl\Downloads\Data\historical_data.csv')
```

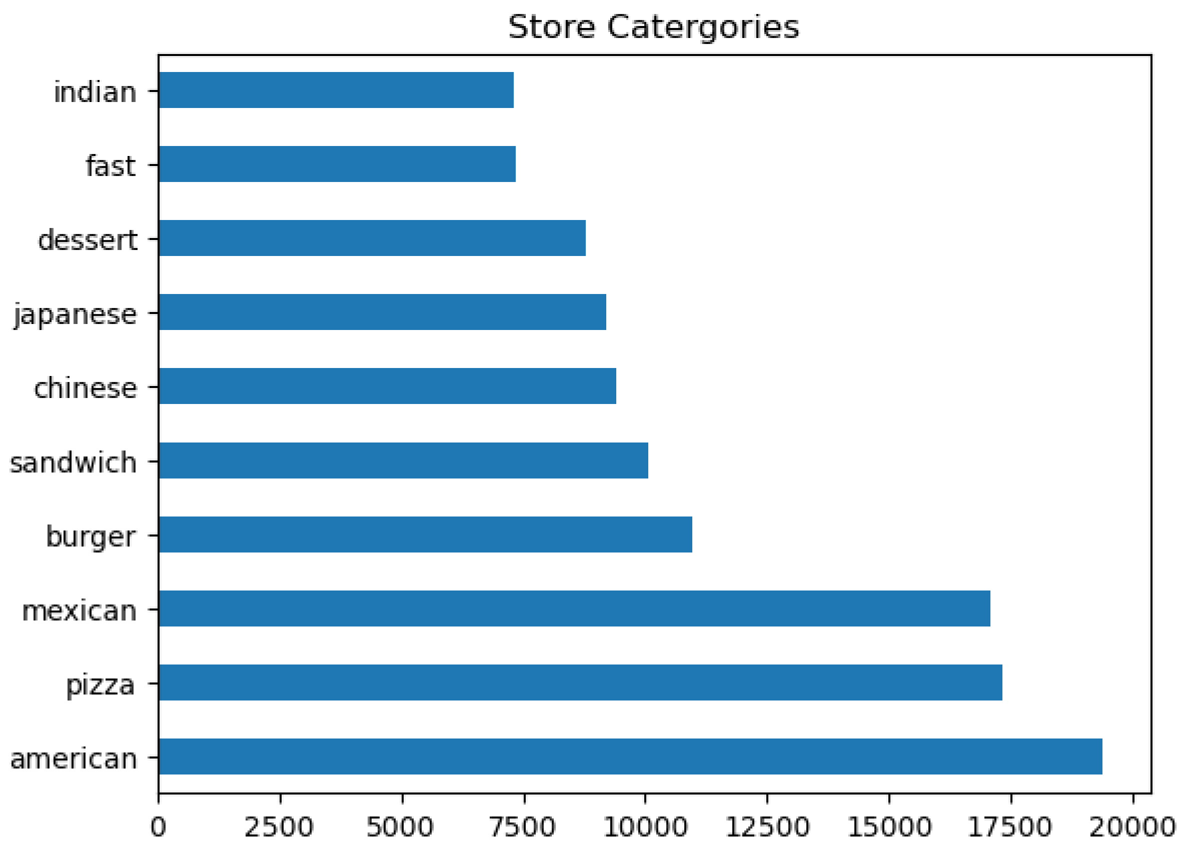
In [3]:
```
df.head(5)
```

Out[3]:

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

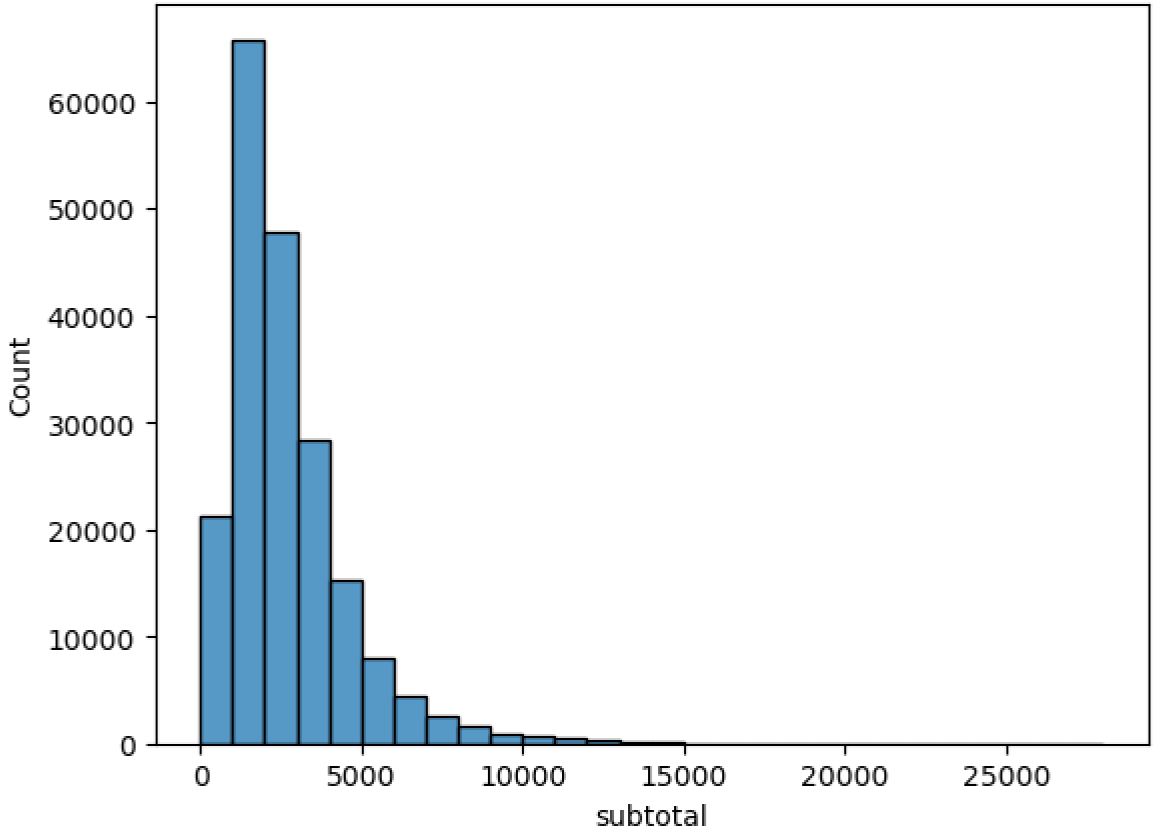
Exploratory Data Analysis

In [4]:
```
store_categories = df['store_primary_category'].value_counts().head(10)
store_categories.plot(kind='barh')
plt.title('Store Catergories')
plt.show()
```



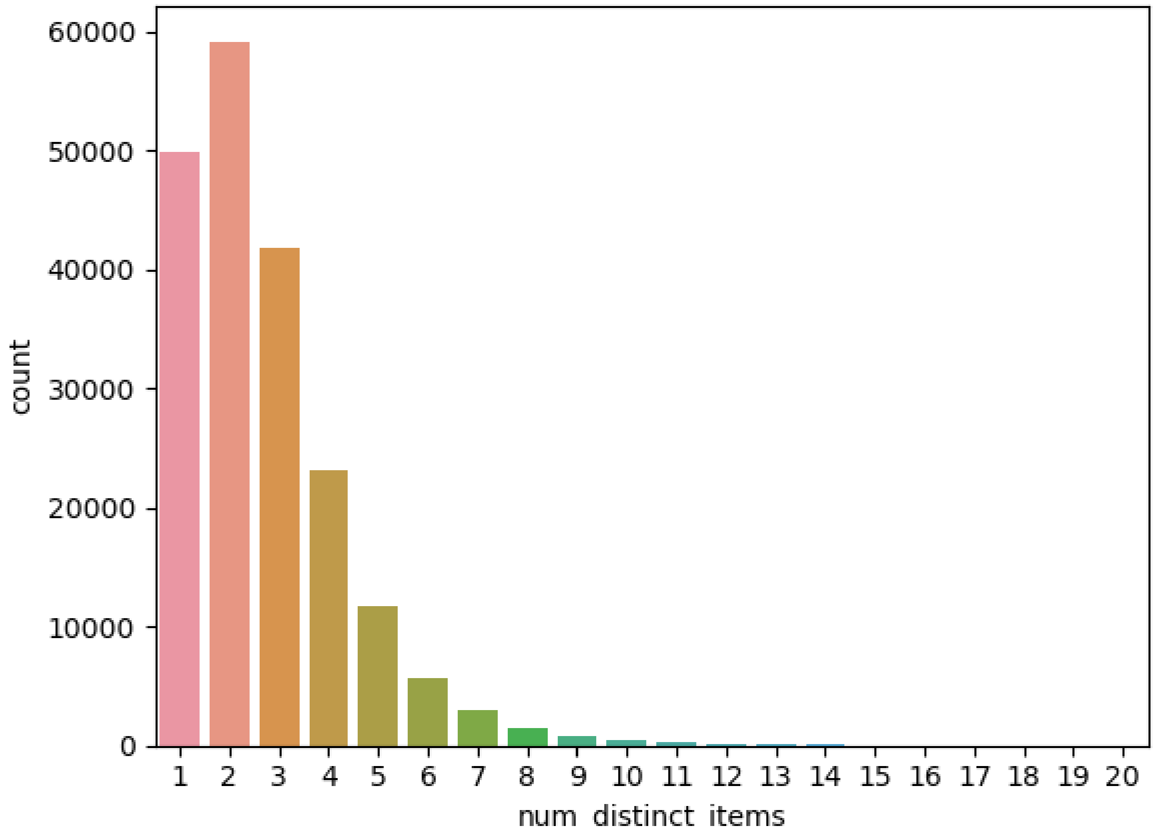
- Most popular items are american but some american can be also considered within other categories such as pizza, burger, sandwich, fast

In [5]:
```
sns.histplot(data=df, x="subtotal",binwidth=1000)
plt.show()
```



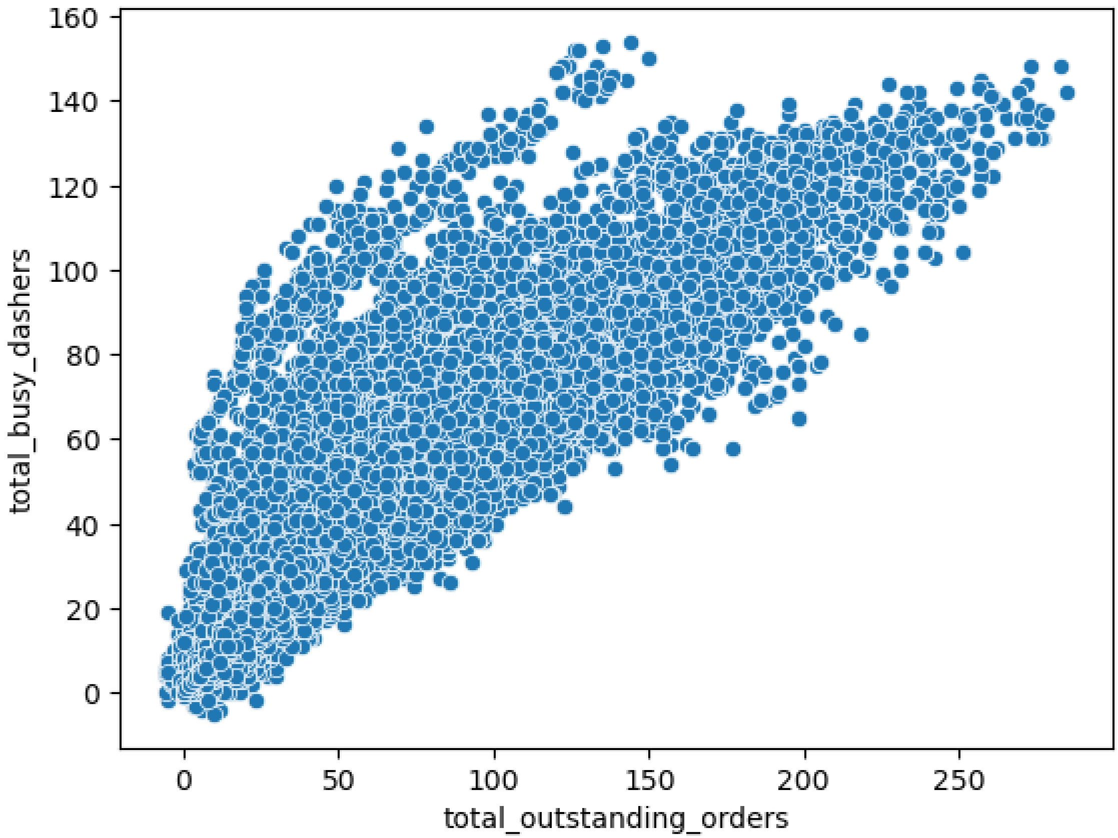
- Most orders around 10-25 dollars

In [6]:
```
sns.countplot(data=df, x="num_distinct_items")
plt.show()
```



- Looks like orders are usually 1-3 items such as main meal, drink, side

In [7]:
```
sns.scatterplot(data=df, x="total_outstanding_orders", y="total_busy_dashers")
plt.show()
```



- More Dashers online can accomodate more orders in the area

Feature Engineering

In [8]:
```
df.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
```

Duration Feature

- actual_delivery_time - created_at
- Convert datatypes from object to datetime64, find difference, and then convert to seconds

In [9]:
```
# Convert from object to datetime64
df['created_at'] = pd.to_datetime(df['created_at'])
# Convert from object to datetime64
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
# Create the Feature and convert to seconds
df['duration'] = (df['actual_delivery_time'] - df['created_at']).dt.total_seconds()
```

Busy Dasher Ratio Feature

- Higher Ratio = More occupied dasher = delivery time may increase
- A ratio greater than 1 may mean that the busy dasher is finishing their last delivery and is clocked out of their shift

In [10]:
```
df['busy_dasher_ratio'] = df['total_busy_dashers'] / df['total_onshift_dashers']
```

Total Estimated Duration Feature

- To calculate estimated duration from restaurant recieving order to dasher picking up and delivering to destination, sum the columns

In [11]:
```
df['total_estimated_duration'] = df['estimated_order_place_duration'] + df['estimated_
```

Creating Dummies for Catergorical Features

- First determine which columns are unique identifiers

In [12]:
```
df.store_id.nunique()
```
Out[12]:
```
6743
```

In [13]:
```
df.market_id.nunique()
```
Out[13]:
```
6
```

In [14]:
```
df.order_protocol.nunique()
```
Out[14]:
```
7
```

In [15]:
```
df.store_primary_category.nunique()
```
Out[15]:
```
74
```

In [16]:
```
market_id_dummies = pd.get_dummies(df.market_id)
market_id_dummies = market_id_dummies.add_prefix('market_id_')
```

In [17]:
```
order_protocol_dummies = pd.get_dummies(df.order_protocol)
order_protocol_dummies = order_protocol_dummies.add_prefix('order_protocol_')
```

In [18]:
```
store_primary_category_dummies = pd.get_dummies(df.store_primary_category)
store_primary_category_dummies = store_primary_category_dummies.add_prefix('store_prim
```

Imputing Missing Values

- for store_primary_category, imputing the mode based on store_id

In [19]:
```
df = df.dropna()
```

In [20]:
```
df.isna().sum()
```

```
Out[20]: market_id                                      0
created_at                                     0
actual_delivery_time                           0
store_id                                       0
store_primary_category                         0
order_protocol                                 0
total_items                                    0
subtotal                                       0
num_distinct_items                             0
min_item_price                                 0
max_item_price                                 0
total_onshift_dashers                          0
total_busy_dashers                             0
total_outstanding_orders                       0
estimated_order_place_duration                 0
estimated_store_to_consumer_driving_duration   0
duration                                       0
busy_dasher_ratio                              0
total_estimated_duration                       0
dtype: int64
```

```
In [21]: df.shape
```

```
Out[21]: (172274, 19)
```

```
In [22]: df.shape
```

```
Out[22]: (172274, 19)
```

```
In [23]: train_df = df.drop(columns = ['created_at','market_id','store_id','store_primary_categ
```

```
In [24]: train_df
```

Out[24]:

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	to
0	1.0	4	3441	4	557	1239	
1	2.0	1	1900	1	1400	1400	
8	3.0	4	4771	3	820	1604	
14	1.0	1	1525	1	1525	1525	
15	1.0	2	3620	2	1425	2195	
...	...	...	...	...	...	...	
197423	4.0	3	1389	3	345	649	
197424	4.0	6	3010	4	405	825	
197425	4.0	5	1836	3	300	399	
197426	1.0	1	1175	1	535	535	
197427	1.0	4	2605	4	425	750	

172274 rows × 14 columns

In [25]:
```
train_df = pd.concat([train_df,market_id_dummies,order_protocol_dummies,store_primary_
train_df = train_df.astype('float32')
train_df
```

Out[25]:

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	tot
0	1.0	4.0	3441.0	4.0	557.0	1239.0	
1	2.0	1.0	1900.0	1.0	1400.0	1400.0	
8	3.0	4.0	4771.0	3.0	820.0	1604.0	
14	1.0	1.0	1525.0	1.0	1525.0	1525.0	
15	1.0	2.0	3620.0	2.0	1425.0	2195.0	
...	...	...	...	...	...	...	
197212	NaN	NaN	NaN	NaN	NaN	NaN	
197259	NaN	NaN	NaN	NaN	NaN	NaN	
197363	NaN	NaN	NaN	NaN	NaN	NaN	
197416	NaN	NaN	NaN	NaN	NaN	NaN	
197421	NaN	NaN	NaN	NaN	NaN	NaN	

197428 rows × 101 columns

In [26]:
```
train_df.describe()
```

Out[26]:

	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_pr
count	172274.000000	172274.000000	172274.000000	172274.000000	172274.000000	172274.000(
mean	2.914659	3.202126	2700.161621	2.672905	685.989197	1162.114!
std	1.510798	2.676944	1828.341553	1.621831	520.047852	560.631ǝ
min	1.000000	1.000000	0.000000	1.000000	-86.000000	0.000(
25%	1.000000	2.000000	1419.000000	1.000000	299.000000	799.000(
50%	3.000000	3.000000	2226.000000	2.000000	595.000000	1095.000(
75%	4.000000	4.000000	3418.750000	3.000000	945.000000	1395.000(
max	7.000000	411.000000	26800.000000	20.000000	14700.000000	14700.000(

8 rows × 101 columns

In [27]:
```
train_df['busy_dasher_ratio'].describe()
```

Out[27]:
```
count    1.722740e+05
mean              NaN
std               NaN
min              -inf
25%      8.281250e-01
50%      9.629630e-01
75%      1.000000e+00
max               inf
Name: busy_dasher_ratio, dtype: float64
```

In [28]:
```
# Remove inf values

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

train_df['busy_dasher_ratio'].describe()
```

Out[28]:
```
count    172236.000000
mean          0.950529
std           0.405698
min         -13.000000
25%           0.827957
50%           0.962963
75%           1.000000
max          31.000000
Name: busy_dasher_ratio, dtype: float64
```

In [29]:
```
train_df.shape
```

Out[29]:
```
(172236, 101)
```

Scaling Data with Standard and Robust Scaler to deal with Outliers

In [30]:
```
from sklearn.preprocessing import RobustScaler
robust_scaler = RobustScaler()

robust_scaler.fit(train_df)

robust_scaled_data = robust_scaler.transform(train_df)

from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()

# combine both fit & transform into one call
standard_scaled_data = standard_scaler.fit_transform(train_df)

# dataframe with both standard and robust scaled values
scaled_values = pd.DataFrame({
    'Standard': standard_scaled_data.reshape(-1),
    'Robust': robust_scaled_data.reshape(-1)
})
```

In [31]:
```
scaled_values.describe()
```

Out[31]:

	Standard	Robust
count	1.739584e+07	1.739584e+07
mean	3.377473e-09	4.803926e-02
std	9.828860e-01	4.178860e-01
min	-3.438486e+01	-8.115971e+01
25%	-1.928992e-01	0.000000e+00
50%	-5.782543e-02	0.000000e+00
75%	-1.721026e-02	0.000000e+00
max	4.150121e+02	2.597063e+02

In [32]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = train_df.drop(columns=['total_estimated_duration'])
y = train_df['total_estimated_duration']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
scaler = StandardScaler()  # or MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 2.1432429e-07
```

Though Mean Squared Error is relatively low, it could mean overfitting. Some more means of optimization may include grid search method to pick most influential features, running random forest or gradient boosting methods. Imputing Missing Values instead of removing.