Porter is India's Largest Marketplace for Intra-City Logistics. Leader in the country's $40 billion intra-city logistics market, Porter strives to improve the lives of 1,50,000+ driver-partners by providing them with consistent earning & independence. Currently, the company has serviced 5+ million customers

Porter works with a wide range of restaurants for delivering their items directly to the people.

Porter has a number of delivery partners available for delivering the food, from various restaurants and wants to get an estimated delivery time that it can provide the customers on the basis of what they are ordering, from where and also the delivery partners.

This dataset has the required data to train a regression model that will do the delivery time estimation, based on all those features

Dataset: Porter data

Data Dictionary

Each row in this file corresponds to one unique delivery. Each column corresponds to a feature as explained below.

- market_id : integer id for the market where the restaurant lies created_at : the timestamp at which the order was placed
- actual_delivery_time : the timestamp when the order was delivered store_primary_category : category for the restaurant
- order_protocol : integer code value for order protocol(how the order was placed ie: through porter, call to restaurant, pre booked, third part etc)
- total_items subtotal : final price of the order
- num_distinct_items : the number of distinct items in the order
- min_item_price : price of the cheapest item in the order
- max_item_price : price of the costliest item in order
- total_onshift_partners : number of delivery partners on duty at the time order was placed
- total_busy_partners : number of delivery partners attending to other tasks
- total_outstanding_orders : total number of orders to be fulfilled at the moment Process

Import the data and understand the structure of the data: usual exploratory analysis steps like checking the structure & characteristics of the dataset Data preprocessing Cleaning of data Feature engineering: Creating the target column time taken in each delivery from order timestamp (created_at) and delivery timestamp (actual_delivery_time) Getting hour of day from the order time and also the day of the week Understanding pandas datetime data type and what function it provides by default Get delivery time in minutes Handling null values Encoding categorical columns Data visualization and cleaning Visualize various columns for better understanding Countplots, scatterplots Check if the data contains outliers Removing outliers by any method Plotting the data again to see if anything has improved Split the data in train and test Scaling the data for neural networks. Creating a simple neural network Trying different configurations Understanding different activation functions, optimizers and other hyperparameters. Training the neural network for required amount of epochs Plotting the losses and checking the accuracy of the model Checking its various metrics like MSE, RMSE, MAE

```
import pandas as pd
import numpy as np

# for visualizing and analyzing it
import matplotlib.pyplot as plt
import seaborn as sns

# data preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# random forest model training
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.ensemble import RandomForestRegressor

# ann training
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import
Dense,Dropout,BatchNormalization,LeakyReLU
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.losses import MeanSquaredLogarithmicError
from tensorflow.keras.losses import MeanAbsolutePercentageError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.metrics import MeanAbsoluteError
from tensorflow.keras.optimizers import SGD,Adam

from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving dataset.csv.zip to dataset.csv (1).zip
```

```
df = pd.read_csv('dataset.csv.zip')
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.tail()
```

```
{"repr_error":"0","type":"dataframe"}
```

```
df.shape      # Check the shape of the dataset: To know the number of
rows and columns.
```

```
(197428, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 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  object 
 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_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

Data preprocessing

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

df['time_taken']=df['actual_delivery_time'] - df['created_at']

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype         
---  ------                    --------------   -----         
 0   market_id                 196441 non-null  float64       
 1   created_at                197428 non-null  datetime64[ns]
 2   actual_delivery_time      197421 non-null  datetime64[ns]
 3   store_id                  197428 non-null  object        
 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_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
 14  time_taken                197421 non-null  timedelta64[ns]
dtypes: datetime64[ns](2), float64(5), int64(5), object(2),
timedelta64[ns](1)
memory usage: 22.6+ MB
```

```
df['time_taken_mins'] =
pd.to_timedelta(df['time_taken'])/pd.Timedelta('60s')

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df['hour'] = df['created_at'].dt.hour
df['day'] = df['created_at'].dt.dayofweek

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.drop(['time_taken','created_at','actual_delivery_time'],axis =
1,inplace = True)
```

Checking null values in the data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 196441 non-null  float64
 1   store_id                  197428 non-null  object
 2   store_primary_category    192668 non-null  object
 3   order_protocol            196433 non-null  float64
 4   total_items               197428 non-null  int64
 5   subtotal                  197428 non-null  int64
 6   num_distinct_items        197428 non-null  int64
 7   min_item_price            197428 non-null  int64
 8   max_item_price            197428 non-null  int64
 9   total_onshift_partners    181166 non-null  float64
 10  total_busy_partners       181166 non-null  float64
 11  total_outstanding_orders  181166 non-null  float64
 12  time_taken_mins           197421 non-null  float64
```

```
 13  hour                      197428 non-null  int32
 14  day                       197428 non-null  int32
dtypes: float64(6), int32(2), int64(5), object(2)
memory usage: 21.1+ MB
```

```
df.isna().sum()
```

```
market_id                    987
store_id                       0
store_primary_category      4760
order_protocol               995
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners     16262
total_busy_partners        16262
total_outstanding_orders   16262
time_taken_mins                7
hour                           0
day                            0
dtype: int64
```

Drop Missing Values:

Fill Missing Values:

You can fill missing values with a default value, mean/median, or using forward/backward fill:

```
df['market_id'].fillna(df['market_id'].mean(), inplace=True)
df['store_primary_category'].fillna('Unknown', inplace=True)
```

Imputation with Statistical Methods:

For numerical columns like total_onshift_partners, total_busy_partners, total_outstanding_orders, you can use mean or median imputation:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')
df[['total_onshift_partners', 'total_busy_partners',
'total_outstanding_orders']] =
imputer.fit_transform(df[['total_onshift_partners',
'total_busy_partners', 'total_outstanding_orders']])
```

```
df['order_protocol'].fillna(df['order_protocol'].median(),
inplace=True)
df['time_taken_mins'].fillna(df['time_taken_mins'].mean(),
inplace=True)

df.isna().sum()
```

```
market_id                     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_partners      0
total_busy_partners           0
total_outstanding_orders    0
time_taken_mins               0
hour                          0
day                           0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 197428 non-null  float64
 1   store_id                  197428 non-null  object
 2   store_primary_category    197428 non-null  object
 3   order_protocol            197428 non-null  float64
 4   total_items               197428 non-null  int64
 5   subtotal                  197428 non-null  int64
 6   num_distinct_items        197428 non-null  int64
 7   min_item_price            197428 non-null  int64
 8   max_item_price            197428 non-null  int64
 9   total_onshift_partners    197428 non-null  float64
 10  total_busy_partners       197428 non-null  float64
 11  total_outstanding_orders  197428 non-null  float64
 12  time_taken_mins           197428 non-null  float64
 13  hour                      197428 non-null  int32
 14  day                       197428 non-null  int32
dtypes: float64(6), int32(2), int64(5), object(2)
memory usage: 21.1+ MB
```

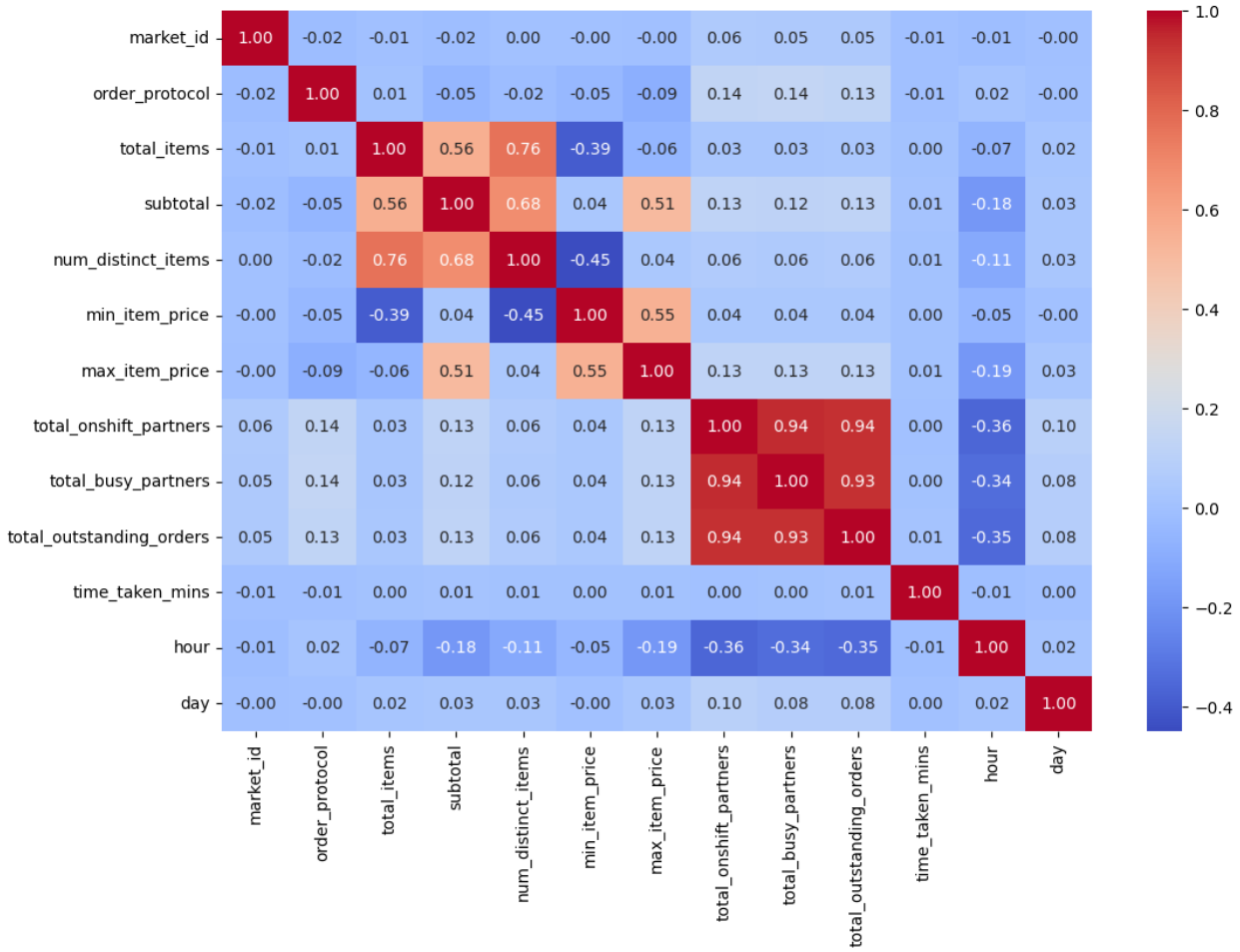
Heatmap

Plotting correlation to get an idea of the data

```
non_numeric_columns = df.select_dtypes(include=['object']).columns
print(non_numeric_columns)
```

```
Index(['store_id', 'store_primary_category'], dtype='object')
```

```
# Drop non-numeric columns
df_numeric = df.drop(columns=non_numeric_columns)

# Or convert specific columns to numeric, if they are categorical
df['store_id'] = df['store_id'].astype('category').cat.codes
df['store_primary_category'] =
df['store_primary_category'].astype('category').cat.codes

# Set the figure size
plt.figure(figsize=(12, 8))

# Generate the correlation matrix
corr_matrix = df_numeric.corr()

# Plot the heatmap with increased font size for annotations
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm',
annot_kws={"size": 10})

# Show the plot
plt.show()
```



From the correlation matrix, here are some key insights:

1. High Positive Correlations:

- total_items and num_distinct_items have a strong positive correlation (0.76), suggesting that as the number of items increases, the number of distinct items also tends to increase.
- total_busy_partners is highly correlated with total_onshift_partners (0.94) and total_outstanding_orders (0.93), indicating that when the number of on-shift partners increases, the number of busy partners and outstanding orders also tends to rise.
- subtotal shows a high correlation with total_items (0.68) and num_distinct_items (0.68), implying that higher totals and distinct items contribute to a higher subtotal.

1. Moderate Positive Correlations:

- max_item_price has a moderate correlation with total_items (0.51), which could mean that orders with more items tend to include items with higher maximum prices.
- min_item_price is moderately negatively correlated with num_distinct_items (-0.45), suggesting that orders with more distinct items may have a lower minimum price per item.

1. Low to Negligible Correlations:

- Variables like market_id, order_protocol, hour, and day generally show weak correlations with most other variables, suggesting these may have a limited or insignificant direct impact on the other metrics.

1. Negative Correlations:

- The hour variable has a moderate negative correlation with total_onshift_partners (-0.36), total_busy_partners (-0.34), and total_outstanding_orders (-0.35). This may indicate that at certain times of the day, there are fewer partners available or fewer outstanding orders.

Overall, this correlation matrix helps identify which variables are most likely to influence each other and could guide further analysis or feature selection in predictive modeling.

 # Encoding categorical columns

```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding to 'store_id' and 'store_primary_category'
df['store_id'] = label_encoder.fit_transform(df['store_id'])
df['store_primary_category'] =
label_encoder.fit_transform(df['store_primary_category'])

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.dtypes     # Check the data types of each column: This helps in
identifying categorical and numerical features.
```

```
market_id                   float64
store_id                      int64
store_primary_category        int64
order_protocol              float64
total_items                   int64
subtotal                      int64
num_distinct_items            int64
min_item_price                int64
max_item_price                int64
total_onshift_partners      float64
total_busy_partners         float64
total_outstanding_orders    float64
time_taken_mins             float64
hour                          int32
day                           int32
dtype: object
```

Summary statistics: For numerical columns, use describe() to get a statistical summary.

```
df.describe()    # Provides summary statistics (count, mean, std, min, 25%, 50%, 75%, max) for numeric columns in a DataFrame.
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n{\n      \"column\": \"market_id\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 69800.30342154758,\n\"min\": 1.0,\n        \"max\": 197428.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n2.978706074597462,\n          3.0,\n          197428.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"store_id\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 68687.4888866806,\n        \"min\": 0.0,\n        \"max\": 197428.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n3463.4580353344004,\n          3513.0,\n          197428.0\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"store_primary_category\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 69789.0033545634,\n\"min\": 0.0,\n        \"max\": 197428.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n36.65034341633405,\n          40.0,\n          197428.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"order_protocol\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 69800.30933171313,\n        \"min\": 1.0,\n        \"max\": 197428.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\n197428.0,\n          2.882944668436088,\n          4.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"total_items\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 69779.92527927404,\n        \"min\": 1.0,\n        \"max\": 197428.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n3.196390582896043,\n          3.0,\n          197428.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"subtotal\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 68428.68300026414,\n        \"min\": 0.0,\n        \"max\": 197428.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n2682.331401827502,\n          2200.0,\n          197428.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"num_distinct_items\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 69799.75814792675,\n        \"min\": 1.0,\n        \"max\": 197428.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\n197428.0,\n          2.6707913771096297,\n          3.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"min_item_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
69088.82623113973,\n        \"min\": -86.0,\n        \"max\":
197428.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
686.2184695180015,\n          595.0,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max_item_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
68978.09320015657,\n        \"min\": 0.0,\n        \"max\": 197428.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
1159.5886297789573,\n          1095.0,\n          197428.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_onshift_partners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 69782.8279056407,\n
\"min\": -4.0,\n        \"max\": 197428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
44.808093130057514,\n          41.0,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_busy_partners\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
69784.35721831996,\n        \"min\": -5.0,\n        \"max\":
197428.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
41.739746972389966,\n          39.0,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_outstanding_orders\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
69774.44996084423,\n        \"min\": -6.0,\n        \"max\":
197428.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
58.050064581654404,\n          47.0,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"time_taken_mins\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
79899.12089873284,\n        \"min\": 1.6833333333333333,\n
\"max\": 197428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          48.470956230593515,\n
44.333333333333336,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hour\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 69798.10042036256,\n
\"min\": 0.0,\n        \"max\": 197428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
8.46721336385923,\n          3.0,\n          197428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"day\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 69800.3153008562,\n
\"min\": 0.0,\n        \"max\": 197428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.2189659014932026,\n          3.0,\n          197428.0\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

Check for missing values: This is crucial to understand the quality of the data.

```
df.isnull().sum()     # Returns the count of missing (NaN) values in
each column of the DataFrame.
```

```
market_id                     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_partners      0
total_busy_partners           0
total_outstanding_orders    0
time_taken_mins               0
hour                          0
day                           0
dtype: int64
```

Check for unique values in categorical columns: This helps in understanding the distribution of categorical features.

```
for col in df.select_dtypes(include='object').columns:    # select the
categorical columns
    print(f'{col}: {df[col].nunique()} unique values')
```

Get an overview of categorical data: Use value_counts() to see the distribution of values.

```
for col in df.select_dtypes(include='object').columns:
    print(df[col].value_counts())
```

Exploratory Data Analysis (EDA):

Correlation Matrix: To understand the relationship between numerical features.

```
# Select only the numerical features
numerical_features = df.select_dtypes(include=[np.number])

correlation_matrix = numerical_features.corr()
print(correlation_matrix)
```

```
                           market_id  store_id  store_primary_category
\
market_id                   1.000000 -0.030107                0.026736

store_id                   -0.030107  1.000000                0.021995

store_primary_category      0.026736  0.021995                1.000000

order_protocol             -0.021831  0.065247                0.084655

total_items                -0.006841  0.005976               -0.011428

subtotal                   -0.016197 -0.001586                0.035062

num_distinct_items          0.002551  0.006766               -0.005269

min_item_price             -0.000150 -0.017519                0.019053

max_item_price             -0.004910 -0.019271                0.007541

total_onshift_partners      0.058207  0.035199                0.079800

total_busy_partners         0.050480  0.037081                0.081149

total_outstanding_orders    0.053212  0.033169                0.079334

time_taken_mins            -0.005784 -0.002001                0.000674

hour                       -0.013846  0.013856               -0.028821

day                        -0.002489 -0.002393               -0.016756


                           order_protocol  total_items  subtotal  \
market_id                       -0.021831    -0.006841 -0.016197
store_id                         0.065247     0.005976 -0.001586
store_primary_category           0.084655    -0.011428  0.035062
order_protocol                   1.000000     0.008766 -0.053501
total_items                      0.008766     1.000000  0.558067
subtotal                        -0.053501     0.558067  1.000000
num_distinct_items              -0.023292     0.763912  0.681101
min_item_price                  -0.045312    -0.393149  0.037038
max_item_price                  -0.090714    -0.058233  0.505547
total_onshift_partners           0.139597     0.030471  0.125753
total_busy_partners              0.143988     0.027465  0.120692
total_outstanding_orders         0.129855     0.033014  0.125056
time_taken_mins                 -0.006662     0.004905  0.011200
hour                             0.015785    -0.066481 -0.184392
day                             -0.000957     0.021815  0.032035

                           num_distinct_items  min_item_price
```

```
max_item_price  \
market_id                          0.002551       -0.000150       -
0.004910
store_id                           0.006766       -0.017519       -
0.019271
store_primary_category            -0.005269        0.019053
0.007541
order_protocol                    -0.023292       -0.045312       -
0.090714
total_items                        0.763912       -0.393149       -
0.058233
subtotal                           0.681101        0.037038
0.505547
num_distinct_items                 1.000000       -0.448739
0.041871
min_item_price                    -0.448739        1.000000
0.545484
max_item_price                     0.041871        0.545484
1.000000
total_onshift_partners             0.063054        0.040527
0.128294
total_busy_partners                0.057981        0.041974
0.126263
total_outstanding_orders           0.064829        0.039220
0.125946
time_taken_mins                    0.006741        0.004762
0.009409
hour                              -0.114275       -0.050562       -
0.186417
day                                0.029227       -0.000168
0.029536

                          total_onshift_partners  total_busy_partners
\
market_id                                0.058207             0.050480

store_id                                 0.035199             0.037081

store_primary_category                   0.079800             0.081149

order_protocol                           0.139597             0.143988

total_items                              0.030471             0.027465

subtotal                                 0.125753             0.120692

num_distinct_items                       0.063054             0.057981

min_item_price                           0.040527             0.041974

max_item_price                           0.128294             0.126263
```

```
total_onshift_partners                  1.000000             0.943789
total_busy_partners                     0.943789             1.000000
total_outstanding_orders                0.936122             0.932913
time_taken_mins                         0.003855             0.004976
hour                                   -0.359284            -0.335679
day                                     0.097687             0.081655


                          total_outstanding_orders  time_taken_mins
hour  \
market_id                                 0.053212        -0.005784 -
0.013846
store_id                                  0.033169        -0.002001
0.013856
store_primary_category                    0.079334         0.000674 -
0.028821
order_protocol                            0.129855        -0.006662
0.015785
total_items                               0.033014         0.004905 -
0.066481
subtotal                                  0.125056         0.011200 -
0.184392
num_distinct_items                        0.064829         0.006741 -
0.114275
min_item_price                            0.039220         0.004762 -
0.050562
max_item_price                            0.125946         0.009409 -
0.186417
total_onshift_partners                    0.936122         0.003855 -
0.359284
total_busy_partners                       0.932913         0.004976 -
0.335679
total_outstanding_orders                  1.000000         0.010036 -
0.347452
time_taken_mins                           0.010036         1.000000 -
0.010526
hour                                     -0.347452        -0.010526
1.000000
day                                       0.083147         0.002857
0.018919

                                day
market_id                 -0.002489
store_id                  -0.002393
```

```
store_primary_category   -0.016756
order_protocol           -0.000957
total_items               0.021815
subtotal                  0.032035
num_distinct_items        0.029227
min_item_price           -0.000168
max_item_price            0.029536
total_onshift_partners    0.097687
total_busy_partners       0.081655
total_outstanding_orders  0.083147
time_taken_mins           0.002857
hour                      0.018919
day                       1.000000
```

```
# Setting plot style
sns.set(style="whitegrid")

# Visualizations
fig, axes = plt.subplots(3, 2, figsize=(14, 16))
fig.suptitle('Data Visualization', fontsize=16)

# Countplot for 'market_id'
sns.countplot(x='market_id', data=df, ax=axes[0, 0])
axes[0, 0].set_title('Countplot of Market ID')

# Countplot for 'store_primary_category'
sns.countplot(x='store_primary_category', data=df, ax=axes[0, 1])
axes[0, 1].set_title('Countplot of Store Primary Category')

# Scatterplot for 'total_items' vs 'subtotal'
sns.scatterplot(x='total_items', y='subtotal', data=df, ax=axes[1, 0])
axes[1, 0].set_title('Total Items vs Subtotal')

# Scatterplot for 'min_item_price' vs 'max_item_price'
sns.scatterplot(x='min_item_price', y='max_item_price', data=df,
ax=axes[1, 1])
axes[1, 1].set_title('Min Item Price vs Max Item Price')

# Scatterplot for 'total_onshift_partners' vs
'total_outstanding_orders'
sns.scatterplot(x='total_onshift_partners',
y='total_outstanding_orders', data=df, ax=axes[2, 0])
axes[2, 0].set_title('Total Onshift Partners vs Total Outstanding
Orders')

# Scatterplot for 'time_taken_mins' vs 'hour'
sns.scatterplot(x='hour', y='time_taken_mins', data=df, ax=axes[2, 1])
axes[2, 1].set_title('Time Taken vs Hour')
```

```
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Countplot of Market ID:

Market ID 2.0 has the highest count, indicating it has the most records in the dataset. Market IDs 4.0, and 6.0 have relatively lower counts, suggesting fewer records or activities associated with these markets. Countplot of Store Primary Category:

There are many different store categories, but only a few have significantly higher counts. Categories like 0, 2, and 8 have the highest number of stores, while many other categories have very low counts, indicating they are less common.

Total Items vs. Subtotal Scatter Plot:

There's a dense cluster of points towards the lower left of the graph, suggesting that most orders have a smaller number of total items and a lower subtotal. A few outliers exist with high subtotals but a varying number of items, indicating some expensive items or bulk purchases.

Min Item Price vs. Max Item Price Scatter Plot:

The plot shows a positive linear relationship between the minimum and maximum item prices, suggesting that stores with higher minimum prices also tend to have higher maximum prices. There are a few outliers where the maximum price is significantly higher than the minimum, indicating a diverse price range within those stores.

Total Onshift Partners vs. Total Outstanding Orders Scatter Plot:

There's a strong positive correlation between the total number of onshift partners and the total number of outstanding orders. This indicates that as the number of available partners increases, the number of outstanding orders also increases, possibly due to higher demand or busier times.

Time Taken vs. Hour Scatter Plot:

Most data points are concentrated at lower hours, suggesting the majority of orders occur within a certain time range. However, there are some extreme outliers in the "Time Taken" axis, indicating a few instances where the time taken is unusually high, which could be due to delays or other issues.

General Insights: The visualizations suggest variations in market activities, store types, item pricing, and operational metrics like the number of onshift partners and outstanding orders. Outliers in the data could indicate unique behaviors or anomalies that may require further investigation. Strong correlations in certain graphs (e.g., Total Onshift Partners vs. Total Outstanding Orders) suggest potential patterns that could be predictive of operational metrics.

Check if the data contains outliers

Removing outliers by any method

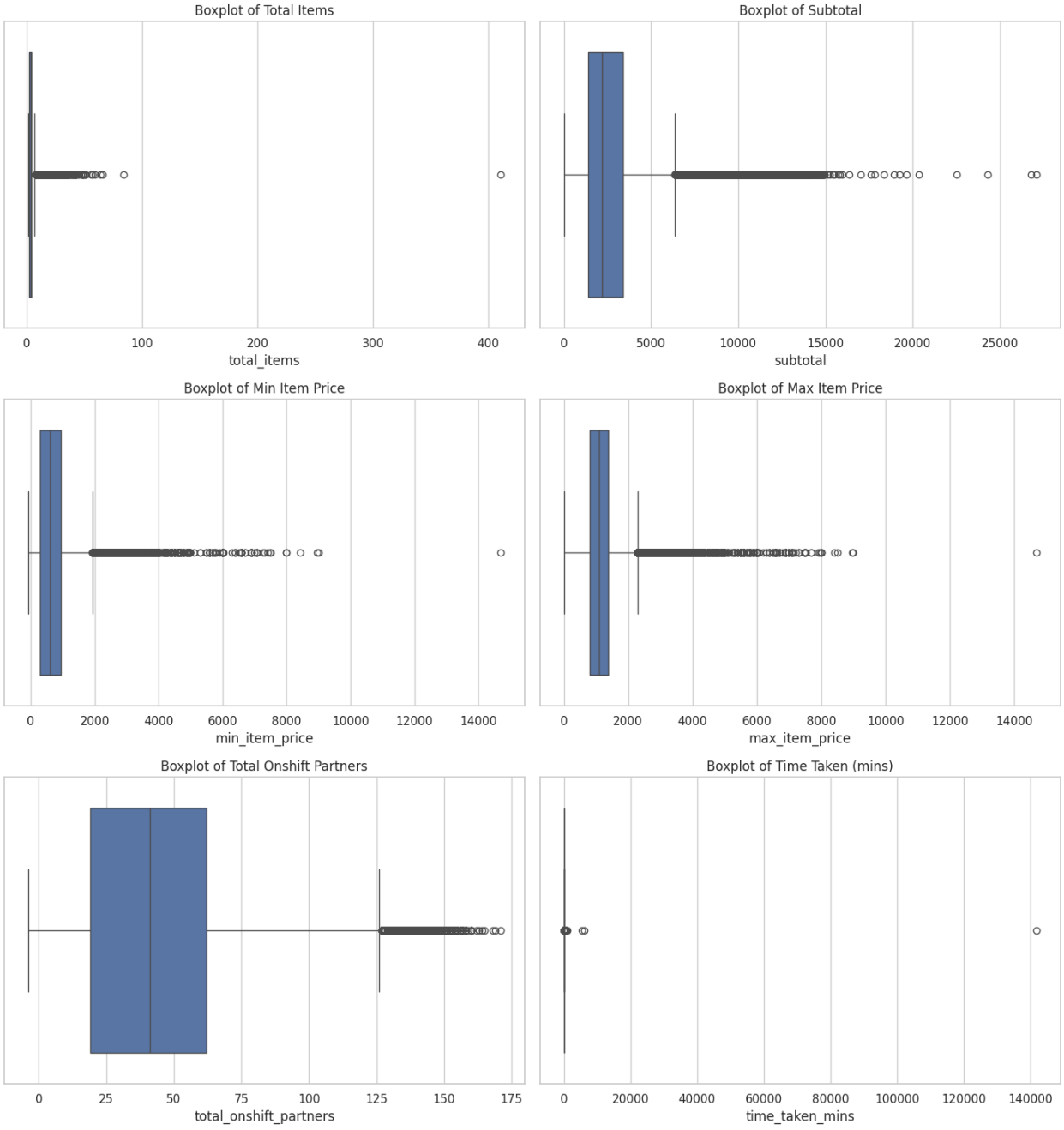
Plotting the data again to see if anything has improved

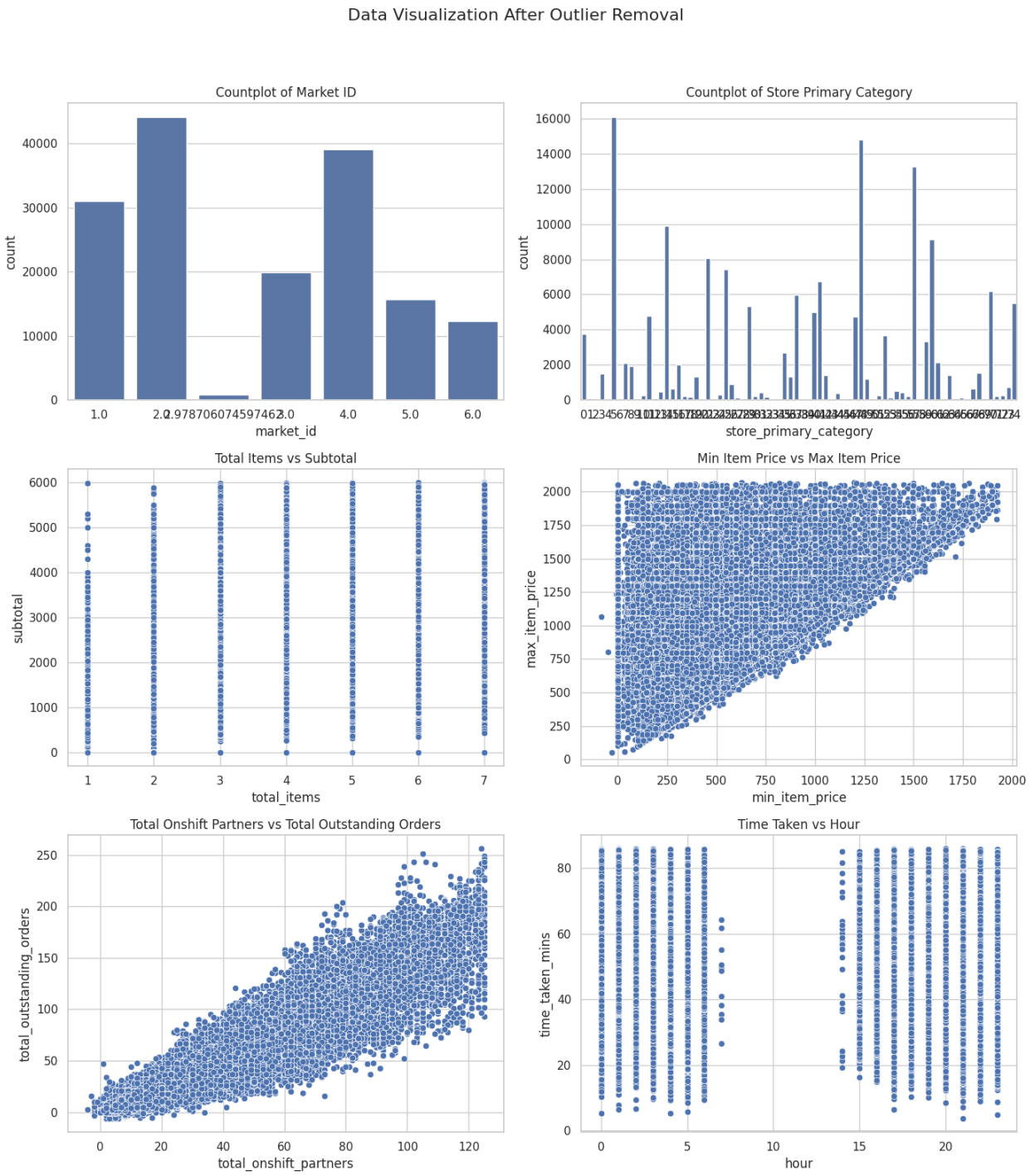
```
# Checking for outliers using boxplots
fig, axes = plt.subplots(3, 2, figsize=(14, 16))
fig.suptitle('Outlier Detection Using Boxplots', fontsize=16)
```

```
# Boxplot for 'total_items'
sns.boxplot(x=df['total_items'], ax=axes[0, 0])
axes[0, 0].set_title('Boxplot of Total Items')

# Boxplot for 'subtotal'
sns.boxplot(x=df['subtotal'], ax=axes[0, 1])
axes[0, 1].set_title('Boxplot of Subtotal')

# Boxplot for 'min_item_price'
sns.boxplot(x=df['min_item_price'], ax=axes[1, 0])
axes[1, 0].set_title('Boxplot of Min Item Price')

# Boxplot for 'max_item_price'
sns.boxplot(x=df['max_item_price'], ax=axes[1, 1])
axes[1, 1].set_title('Boxplot of Max Item Price')

# Boxplot for 'total_onshift_partners'
sns.boxplot(x=df['total_onshift_partners'], ax=axes[2, 0])
axes[2, 0].set_title('Boxplot of Total Onshift Partners')

# Boxplot for 'time_taken_mins'
sns.boxplot(x=df['time_taken_mins'], ax=axes[2, 1])
axes[2, 1].set_title('Boxplot of Time Taken (mins)')

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

# Removing outliers using the IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <=
upper_bound)]

# Removing outliers from relevant columns
columns_to_clean = ['total_items', 'subtotal', 'min_item_price',
'max_item_price', 'total_onshift_partners', 'time_taken_mins']
for col in columns_to_clean:
    df = remove_outliers(df, col)

# Plotting data again to check improvement
fig, axes = plt.subplots(3, 2, figsize=(14, 16))
fig.suptitle('Data Visualization After Outlier Removal', fontsize=16)

# Countplot for 'market_id'
sns.countplot(x='market_id', data=df, ax=axes[0, 0])
axes[0, 0].set_title('Countplot of Market ID')
```

```
# Countplot for 'store_primary_category'
sns.countplot(x='store_primary_category', data=df, ax=axes[0, 1])
axes[0, 1].set_title('Countplot of Store Primary Category')

# Scatterplot for 'total_items' vs 'subtotal'
sns.scatterplot(x='total_items', y='subtotal', data=df, ax=axes[1, 0])
axes[1, 0].set_title('Total Items vs Subtotal')

# Scatterplot for 'min_item_price' vs 'max_item_price'
sns.scatterplot(x='min_item_price', y='max_item_price', data=df,
ax=axes[1, 1])
axes[1, 1].set_title('Min Item Price vs Max Item Price')

# Scatterplot for 'total_onshift_partners' vs
'total_outstanding_orders'
sns.scatterplot(x='total_onshift_partners',
y='total_outstanding_orders', data=df, ax=axes[2, 0])
axes[2, 0].set_title('Total Onshift Partners vs Total Outstanding
Orders')

# Scatterplot for 'time_taken_mins' vs 'hour'
sns.scatterplot(x='hour', y='time_taken_mins', data=df, ax=axes[2, 1])
axes[2, 1].set_title('Time Taken vs Hour')

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Outlier Detection Using Boxplots





The figure shows a series of boxplots for different variables, which are used to detect outliers in the dataset. Here are the insights from each boxplot:

1. Total Items (total_items):

- The majority of the data points are below 100, but there are some significant outliers, with a few exceeding 400.
- The data is highly skewed to the right, indicating that most orders contain a relatively low number of items, with a few orders containing a much higher number.

1. Subtotal (subtotal):
- Most data points lie below 10,000, with several outliers extending up to 25,000.
- The distribution is right-skewed, suggesting that while most subtotals are lower, there are some extremely high subtotals in the data.

1. Min Item Price (min_item_price):
- The majority of minimum item prices are clustered close to 0, with some outliers extending beyond 2,000, and a few extreme cases up to 14,000.
- This indicates that while most items have a low minimum price, there are some with unusually high minimum prices.

1. Max Item Price (max_item_price):
- Similar to the minimum item price, most maximum item prices are clustered close to 0, with outliers extending up to 14,000.
- The presence of outliers on the higher end indicates variability in the maximum price of items in the dataset.

1. Total Onshift Partners (total_onshift_partners):
- The majority of data points range between 0 and 75, with a few outliers extending up to 175.
- The distribution is right-skewed, indicating that most shifts have a relatively low number of partners, but a few have a significantly higher number.

1. Time Taken (minutes) (time_taken_mins):
- Most data points are below 20,000 minutes, but there are some extreme outliers that extend up to 140,000 minutes.
- This suggests that while most deliveries or tasks are completed in a relatively short time, there are some cases that took an unusually long amount of time. **Overall Insights:**
- All boxplots show right-skewed distributions with several outliers, indicating that while the majority of data points fall within a certain range, there are some extreme values that deviate significantly. The presence of these outliers may suggest data entry errors, exceptional cases, or special conditions that might need further investigation or cleansing before analysis.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
y=df['time_taken_mins']
x=df.drop(['time_taken_mins'],axis=1)
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=42)
regressor=RandomForestRegressor()
regressor.fit(X_train,y_train)
```

```
RandomForestRegressor()
```

```
from sklearn.metrics import
mean_squared_error,mean_absolute_error,r2_score
prediction=regressor.predict(X_test)
mse=mean_squared_error(y_test,prediction)
rmse=mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
```

```
mae=mean_absolute_error(y_test,prediction)
print("mase : ",mae)
mape=np.mean(np.abs((y_test - prediction)/y_test))*100
print("mape : ",mape)

r2_score(y_test,prediction)
```

```
mse :  156.51890983840116
rmse :  12.510751769514139
mase :  9.910262973624647
mape :  24.850390136487626

0.23299146990942032
```

#Regression with neural networks

- Data scaling
- Defining NN architecture
- Trying different combinations and hyperparameters
- Model training
- Comparing results with random forest

Scaling the data for neural networks.

```
#Scalling the data to feed before neural network

from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
x_scaled=scaler.fit_transform(x)
X_train,X_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.
2,random_state=42)
```

We will build a simple neural network to train our regression model it is a sequential model with two layers,

we have kept the number of nodes in the first layers equal to the number of input columns, and for the subsequent layers 32, 32, which can we changed or experimented with

the activation for the layers is kept as relu because it is a great non linear activation function that works for most cases, we could have used leaky relu if we see gradient vanishing.

the last layer has one node because it will give the single result that is our delivery time and the activation function for that should be linear

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model=Sequential()
model.add(Dense(11,kernel_initializer='normal',activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1,activation='linear'))
model.compile(loss='mse',optimizer='Adam',metrics=['mse','mae'])
history=model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,v
alidation_split=0.2)
```

```
Epoch 1/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 1799.0333 - mae:
38.8017 - mse: 1799.0333 - val_loss: 236.4522 - val_mae: 12.0857 -
val_mse: 236.4522
Epoch 2/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 235.0303 - mae:
12.1226 - mse: 235.0303 - val_loss: 227.3011 - val_mae: 11.9449 -
val_mse: 227.3011
Epoch 3/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 225.1225 - mae:
11.8886 - mse: 225.1225 - val_loss: 216.8840 - val_mae: 11.6645 -
val_mse: 216.8840
Epoch 4/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 215.5042 - mae:
11.6622 - mse: 215.5042 - val_loss: 207.7526 - val_mae: 11.4381 -
val_mse: 207.7526
Epoch 5/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 206.7768 - mae:
11.4611 - mse: 206.7768 - val_loss: 200.6391 - val_mae: 11.2267 -
val_mse: 200.6391
Epoch 6/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 198.1921 - mae:
11.2322 - mse: 198.1921 - val_loss: 194.9805 - val_mae: 11.1056 -
val_mse: 194.9805
Epoch 7/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 193.3450 - mae:
11.0972 - mse: 193.3450 - val_loss: 185.5117 - val_mae: 10.8801 -
val_mse: 185.5117
Epoch 8/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 182.4257 - mae:
10.8183 - mse: 182.4257 - val_loss: 178.7505 - val_mae: 10.7198 -
val_mse: 178.7505
Epoch 9/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 176.3884 - mae:
10.6130 - mse: 176.3884 - val_loss: 175.4544 - val_mae: 10.6048 -
val_mse: 175.4544
Epoch 10/10
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 173.6138 - mae:
```

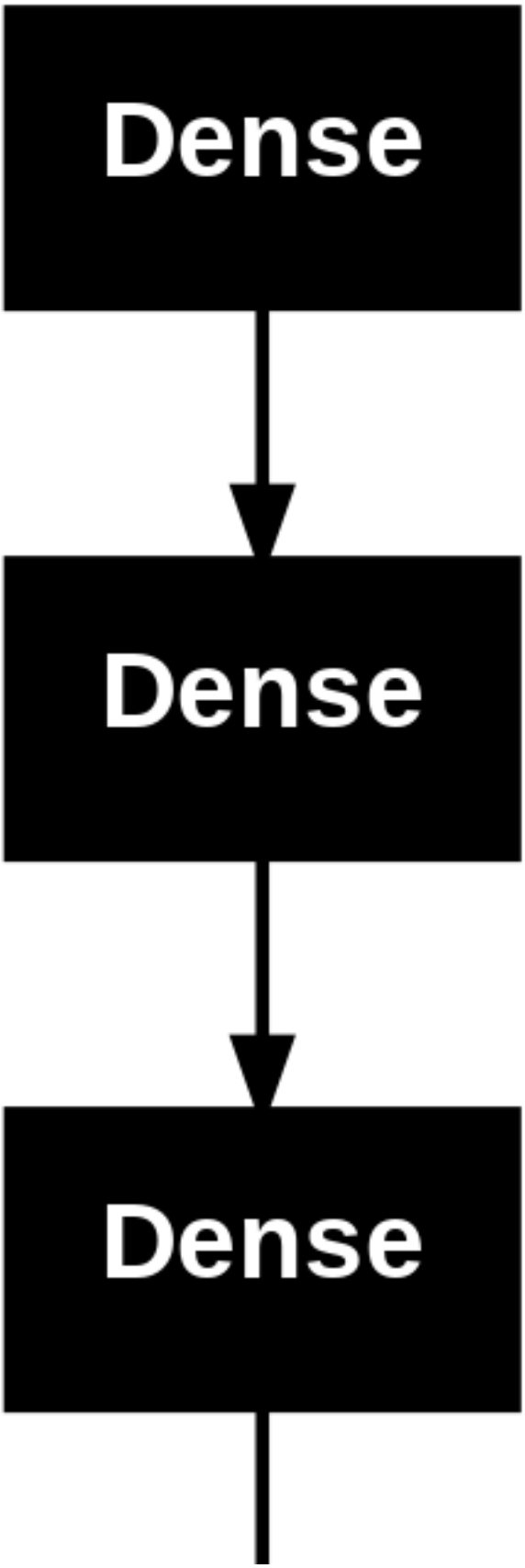
```
10.5217 - mse: 173.6138 - val_loss: 173.8786 - val_mae: 10.4782 -
val_mse: 173.8786
```

```
model.summary()
from tensorflow.keras.utils import plot_model
plot_model(model)
```

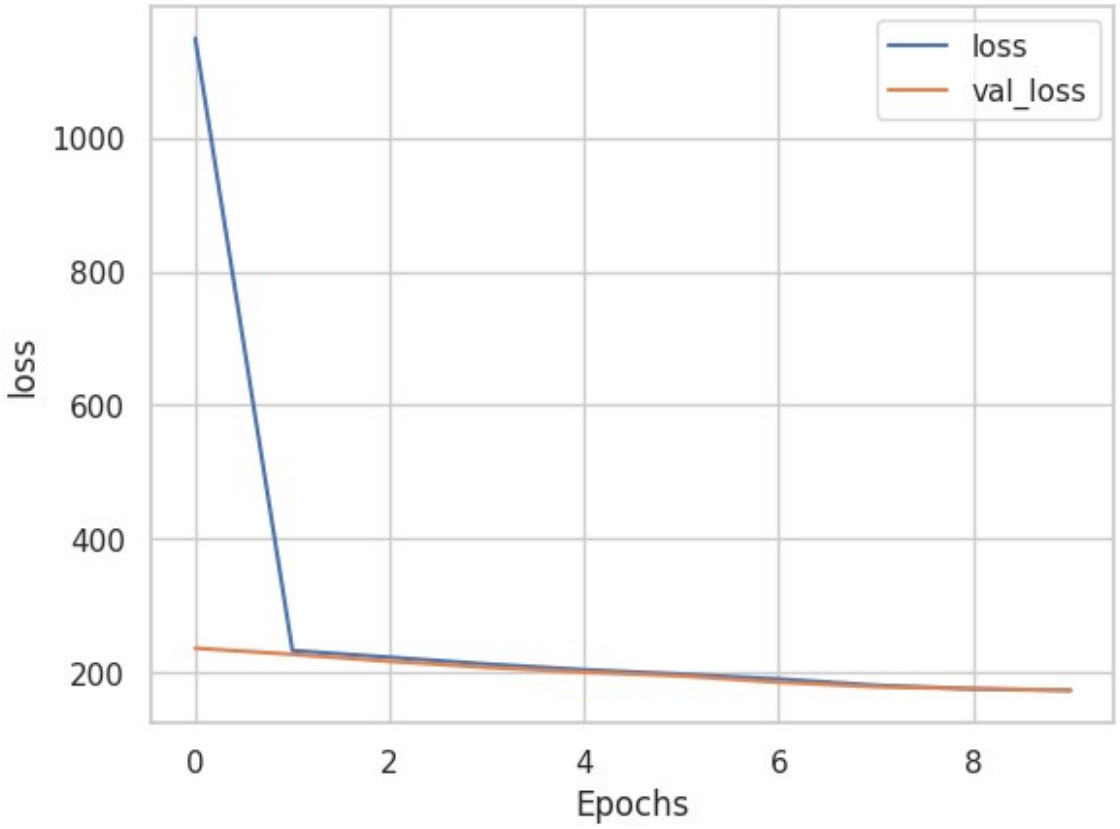
```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃ Param #         ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense (Dense)                        │ (None, 11)                  │ 165             │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 32)                  │ 384             │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ (None, 32)                  │ 1,056           │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_3 (Dense)                      │ (None, 1)                   │ 33              │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 4,916 (19.21 KB)

 Trainable params: 1,638 (6.40 KB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 3,278 (12.81 KB)
```


Dense
Dense
Dense

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel("Epochs")
plt.ylabel('loss')
plt.legend(['loss','val_loss'])
plt.show()
```




```
print('r2_score:',r2_score(y_test, model.predict(X_test)))
mse = mean_squared_error(y_test, model.predict(X_test))
rmse = mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
print("errors for neural net")
mae = mean_absolute_error(y_test, model.predict(X_test))
print("mae : ",mae)
```

```
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
r2_score: 0.1599950357913913
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
mse :  171.4148618937131
rmse :  13.092549862181663
errors for neural net
```

```
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
mae :  10.38658438795292

from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, model.predict(X_test))

1018/1018 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step

0.2593568884722927
```

Leading Questions:

Q1 Defining the problem statements and where can this and modifications of this be used?

Problem Statements:

1. **Predicting Delivery Time:** The primary problem this code addresses is predicting the time it takes for a delivery to be completed (time_taken_mins) based on various features like the number of items, subtotal, store category, etc.

2. **Optimizing Delivery Operations:** By accurately predicting delivery times, businesses can optimize their operations, allocate resources efficiently, and improve customer satisfaction by providing more accurate delivery estimates.

3. **Identifying Factors Affecting Delivery Time:** The analysis helps identify key factors that significantly influence delivery times, allowing businesses to focus on improving those areas for faster deliveries.

Where Can This Be Used (and Modifications):

1. **Food Delivery Platforms:** This model can be directly applied to food delivery platforms like Uber Eats, DoorDash, or Grubhub to predict delivery times and improve their logistics.

2. **E-commerce and Retail:** Online retailers can use a similar approach to predict shipping times and optimize their delivery networks.

3. **Logistics and Supply Chain:** Logistics companies can leverage this model to predict transportation times and optimize routes for efficient delivery of goods.

Modifications:

1. **Feature Engineering:** Adding more relevant features like traffic conditions, distance between store and customer, and delivery personnel experience can improve the model's accuracy.

2. **Advanced Models:** Exploring more complex machine learning models like gradient boosting or deep neural networks could potentially lead to better predictions.

3. **Real-time Predictions:** Integrating the model with real-time data feeds can enable dynamic delivery time predictions based on current conditions.

4. **Customer-Specific Predictions:** Personalizing predictions based on individual customer preferences and historical data can further enhance the user experience.

Q2 List 3 functions the pandas datetime provides with one line explanation.

Here are three useful functions provided by pandas for handling datetime:

1. pd.to_datetime() - Converts an argument (e.g., string, list, or Series) to a datetime object.

2. pd.date_range() - Generates a range of datetime values based on a specified frequency, start, and end date.

3. pd.DatetimeIndex() - Creates an index with datetime elements, enabling datetime-based indexing for pandas objects.

3. Short note on datetime, timedelta, time span (period)

`datetime` is a Python module that provides classes for manipulating date and time.

The `datetime` class represents a point in time and supports date and time arithmetic.

`timedelta` represents the difference between two dates or times, allowing for date and time calculations like adding or subtracting days.

`time span` or `period` is often used in data analysis to describe the length of time between two points,

for example, the duration between two events.

4. Why do we need to check for outliers in our data?

Outliers can significantly impact the performance of machine learning models,

especially models sensitive to variations in data, like linear regression.

They can distort statistical analyses, resulting in misleading results,

and may indicate data errors, variability in measurements, or a novel phenomenon.

5. Name 3 outlier removal methods?

(i) Z-score method: Identifies outliers by calculating the Z-score for each data point.

(ii) IQR (Interquartile Range) method: Identifies outliers based on the range between the first and third quartiles.

(iii) Isolation Forest: A machine learning algorithm specifically designed to detect anomalies.

6. What classical machine learning methods can we use for this problem?

Classical machine learning methods include:

(i) Linear Regression

(ii) Logistic Regression

(iii) Decision Trees

(iv) Random Forest

(v) K-Nearest Neighbors (KNN)

7. Why is scaling required for neural networks?

Scaling is required for neural networks because it ensures that the input features have similar ranges,

which helps the model converge faster during training. It prevents the gradient descent algorithm

from getting stuck in local minima and ensures that the model learns efficiently.

8. Briefly explain your choice of optimizer.

An optimizer adjusts the weights of a neural network to minimize the loss function.

For example, Adam (Adaptive Moment Estimation) is often preferred due to its adaptive learning rate and

momentum, which speeds up convergence and improves performance on complex problems.

9. Which activation function did you use and why?

The ReLU (Rectified Linear Unit) activation function is commonly used because it is computationally efficient

and helps the network learn faster by alleviating the vanishing gradient problem.

For classification problems, a softmax function may be used in the output layer to output probability scores.

10. Why does a neural network perform well on a large dataset?

Neural networks perform well on large datasets because they can learn complex patterns and relationships

within the data. A large dataset provides more information, reducing the risk of overfitting,

and helps the model generalize better to unseen data.