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

import tensorflow as tf
tf.config.list_physical_devices()
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Context: 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

Data Fields Description

- **market_id**:
  - Integer ID representing the market where the restaurant is located.
- **created_at**:
  - Timestamp indicating when the order was placed.
- **actual_delivery_time**:
  - Timestamp indicating when the order was delivered.
- **store_primary_category**:
  - The category of the restaurant (e.g., fast food, fine dining, etc.).
- **order_protocol**:
  - Integer code representing the order protocol, indicating how the order was placed (e.g., through a porter, direct call to restaurant, pre-booked, third-party service, etc.).
- **total_items_subtotal**:
  - The 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 the order.
- **total_onshift_partners**:

  - The number of delivery partners on duty at the time the order was placed.
- **total_busy_partners**:
  - The number of delivery partners attending to other tasks at the time the order was placed.
- **total_outstanding_orders**:
  - The total number of orders that need to be fulfilled at the time the order was placed.

```
try:
    # Disable all GPUS
    tf.config.set_visible_devices([], 'GPU')
    visible_devices = tf.config.get_visible_devices()
    for device in visible_devices:
        assert device.device_type != 'GPU'
except:
    pass

data = pd.read_csv("dataset.csv")

data.head(2)

   market_id           created_at actual_delivery_time  \
0        1.0  2015-02-06 22:24:17  2015-02-06 23:27:16
1        2.0  2015-02-10 21:49:25  2015-02-10 22:56:29

                           store_id store_primary_category
order_protocol  \
0  df263d996281d984952c07998dc54358               american
1.0
1  f0ade77b43923b38237db569b016ba25                mexican
2.0

   total_items  subtotal  num_distinct_items  min_item_price
max_item_price  \
0            4      3441                   4             557
1239
1            1      1900                   1            1400
1400

   total_onshift_partners  total_busy_partners
total_outstanding_orders
0                    33.0                 14.0
21.0
1                     1.0                  2.0
2.0

# no duplicates
data.duplicated().sum()

0
```

```python
data.shape
```

```
(197428, 14)
```

```python
data.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
```

```python
# there are some missing values
data.isna().sum()
```

```
market_id                     987
created_at                      0
actual_delivery_time            7
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
dtype: int64
```

```python
data.columns
```

```
Index(['market_id', 'created_at', 'actual_delivery_time', 'store_id',
       'store_primary_category', 'order_protocol', 'total_items',
```

```
'subtotal',
       'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders'],
      dtype='object')
```

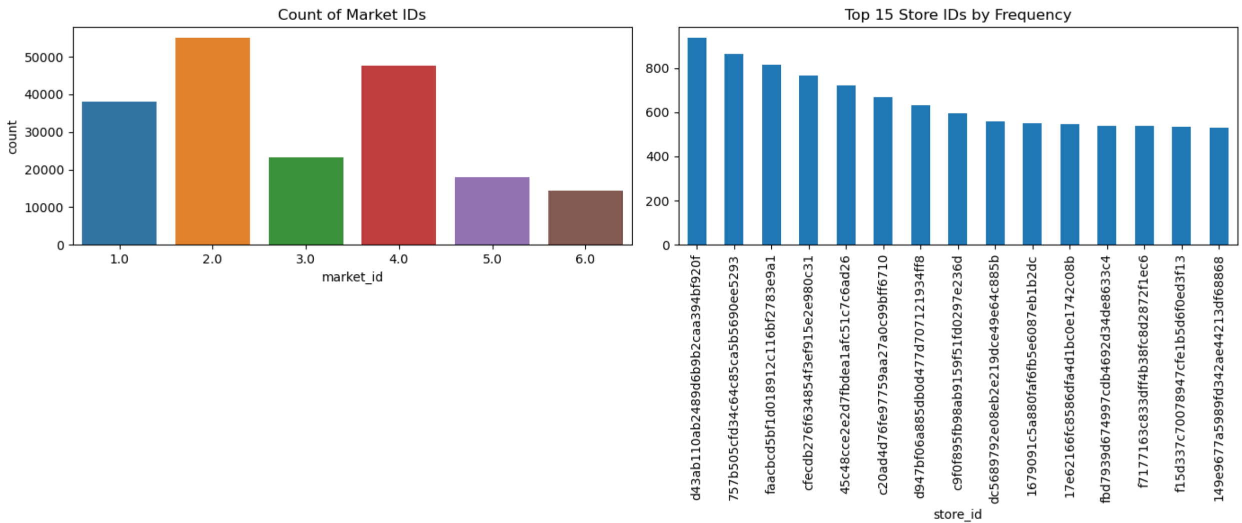
```python
category_c = ['market_id','store_id','store_primary_category']
dates_c = ['created_at' ,'actual_delivery_time']

numbers_c =['num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders','total_items', 'subtotal']
```

EDA

```python
for col in category_c:
    print(f'number of unique catgories `{col}` :
{data[col].nunique()}')
```

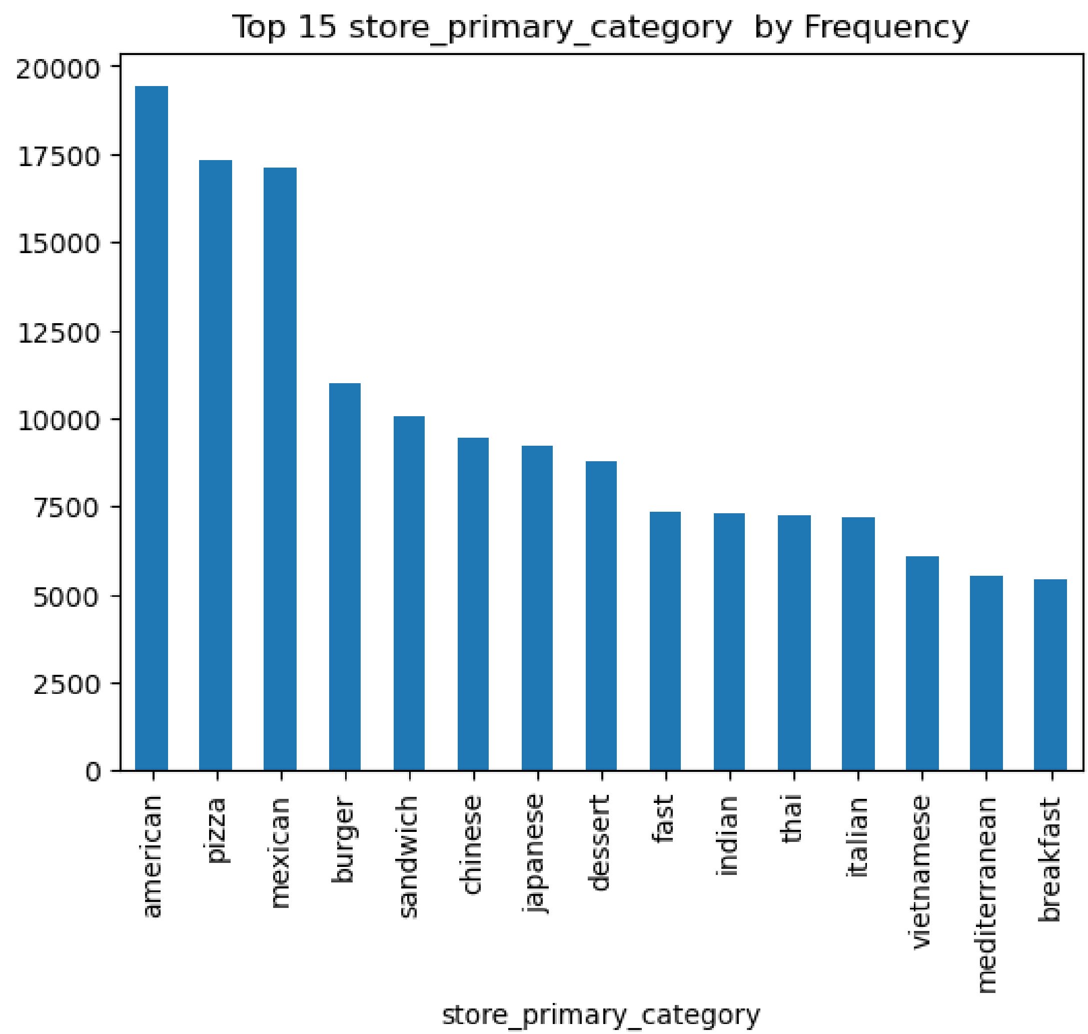
```
number of unique catgories `market_id` : 6
number of unique catgories `store_id` : 6743
number of unique catgories `store_primary_category` : 74
```

```python
import matplotlib.pyplot as plt
import seaborn as sns


fig, axes = plt.subplots(1, 2, figsize=(14, 6))

sns.countplot(data=data, x='market_id', ax=axes[0])
axes[0].set_title('Count of Market IDs')


store_ids_counts = data.groupby('store_id')
['store_id'].count().sort_values(ascending=False)


store_ids_counts.head(15).plot(kind='bar', ax=axes[1])
axes[1].set_title('Top 15 Store IDs by Frequency')


plt.tight_layout()
plt.show()
```




```
store_ids_counts = data.groupby('store_primary_category')
['store_primary_category'].count().sort_values(ascending=False)


store_ids_counts.head(15).plot(kind='bar')
plt.title('Top 15 store_primary_category  by Frequency')
```

```
Text(0.5, 1.0, 'Top 15 store_primary_category  by Frequency')
```



- market type 2 has most number of deleveris and 6 has least
- american , pizza are most frequent categories

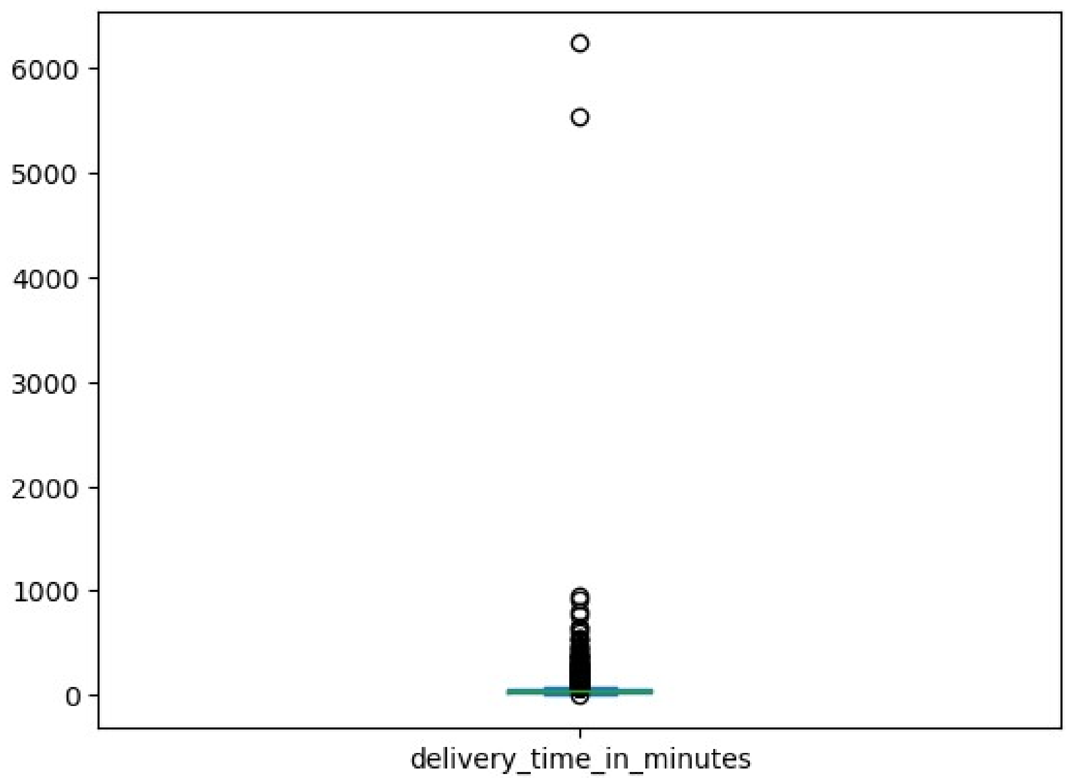
```
store_ids_counts.quantile([0.09,0.25 , 0.5 , .75 , 0.8 , 0.95])

0.09        20.71
0.25        81.00
0.50       519.50
0.75      2728.00
0.80      4562.80
0.95     10374.30
Name: store_primary_category, dtype: float64
```

- 9 or less than 9% of store has just one delivery
- 5 or less than 5% of store have more than 116 delivery
- median deliveries per store is 11
- there are 6743 unique store ids
- droping missing values

```
data.dropna(inplace=True)
data.shape

(176248, 14)
```

Creating target

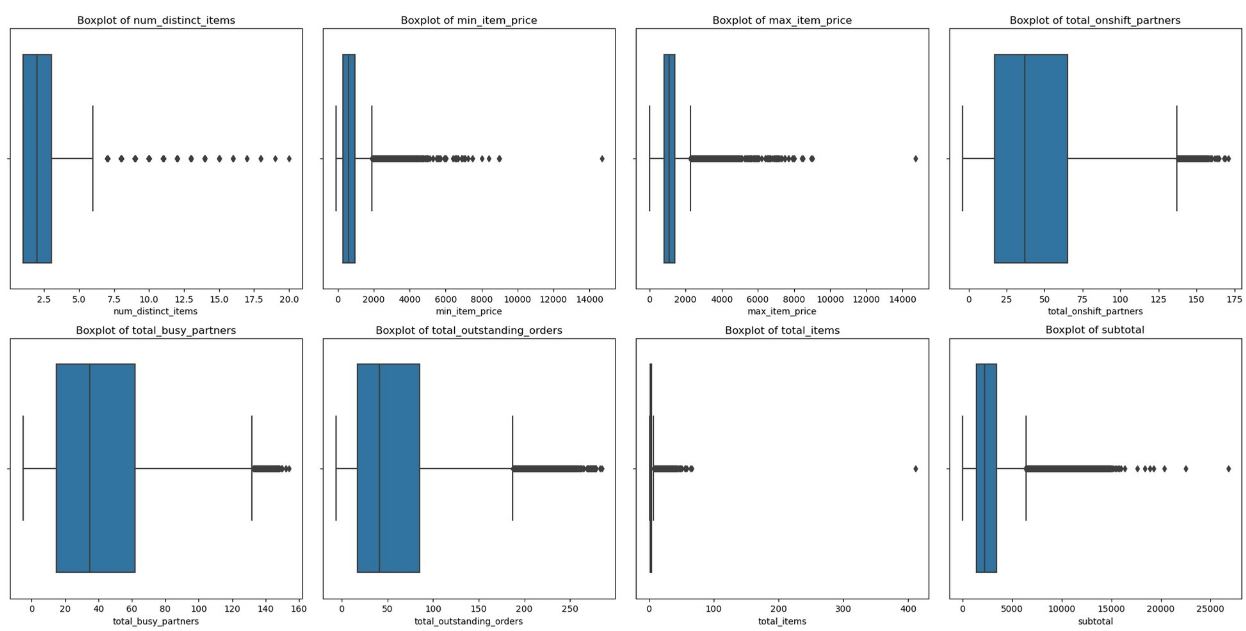
```
data['created_at']=pd.to_datetime(data['created_at'])
data['actual_delivery_time']=pd.to_datetime(data['actual_delivery_time
'])
data['delivery_time_in_minutes'] = (data['actual_delivery_time'] -
data['created_at']).dt.total_seconds() / 60

data['delivery_time_in_minutes'].plot(kind='box')
data['delivery_time_in_minutes'].agg(['min'  ,
'max' ,'mean' ,'median'])

min          1.683333
max       6231.316667
mean        47.764210
median      44.366667
Name: delivery_time_in_minutes, dtype: float64
```



- minimum time to deliver was 1.68 mins max is 103 + hours

- median time is 47 mins and mean is 44 mins
- there are lot on extream values in delivery_time_in_minutes

```
import matplotlib.pyplot as plt
import seaborn as sns

n_cols = 4
n_rows = (len(numbers_c) + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 10))


axes = axes.flatten()


for i, feature in enumerate(numbers_c):
    # Plot boxplot
    sns.boxplot(data=data, x=feature, ax=axes[i])
    axes[i].set_title(f'Boxplot of {feature}')



for i in range(len(numbers_c), len(axes)):
    fig.delaxes(axes[i])


plt.tight_layout()
plt.show()
```




```
for i, feature in enumerate(numbers_c):
    feature_stats = data[feature].describe(percentiles=[.25, .5, .75])
```

```
    print(f"Statistics for {feature}:")
    print(f"Min: {feature_stats['min']}")
    print(f"Max: {feature_stats['max']}")
    print(f"Mean: {feature_stats['mean']}")
    print(f"Median: {feature_stats['50%']}")
    print('-' * 50)
```

```
Statistics for num_distinct_items:
Min: 1.0
Max: 20.0
Mean: 2.674589215196768
Median: 2.0
--------------------------------------------------
Statistics for min_item_price:
Min: -86.0
Max: 14700.0
Mean: 684.9377297898416
Median: 595.0
--------------------------------------------------
Statistics for max_item_price:
Min: 0.0
Max: 14700.0
Mean: 1159.8869944623484
Median: 1095.0
--------------------------------------------------
Statistics for total_onshift_partners:
Min: -4.0
Max: 171.0
Mean: 44.905275520856975
Median: 37.0
--------------------------------------------------
Statistics for total_busy_partners:
Min: -5.0
Max: 154.0
Mean: 41.84543370704916
Median: 35.0
--------------------------------------------------
Statistics for total_outstanding_orders:
Min: -6.0
Max: 285.0
Mean: 58.206799509781675
Median: 41.0
--------------------------------------------------
Statistics for total_items:
Min: 1.0
Max: 411.0
Mean: 3.2045923925377875
Median: 3.0
--------------------------------------------------
Statistics for subtotal:
```

```
Min: 0.0
Max: 26800.0
Mean: 2696.4989389950524
Median: 2221.0
--------------------------------------------------
```

some anomalies in data

- min_item_price contains negative values
- max_item_price has value as zero
- total_onshift_partners has negative value ``
- total_busy_partners contains negative value ``
- total_outstanding_orders contain negative values ``
- subtotal is zero ||by defination subtotal `final price of the order` can be zero

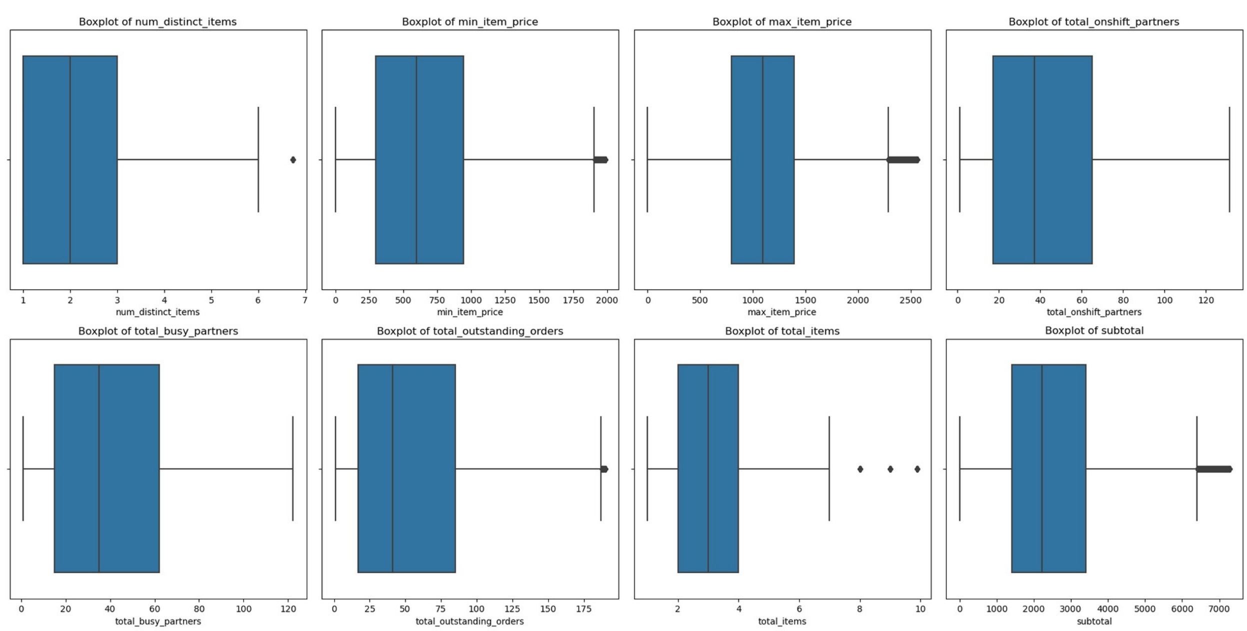
```
def outlier_treatment(column_name  , q =2.5):
    mu =data[column_name].mean()
    std =data[column_name].std()

    upper_std = mu +(q* std)
    lower_std = mu - (q* std)

    mask_u =data[column_name]>=upper_std

    data.loc[mask_u , column_name] =upper_std
    mask_l =data[column_name]<= 0

    print(column_name , upper_std , lower_std)
    data.loc[mask_l , column_name]    =1

    return data[column_name]


for col in numbers_c:
    data[col] =outlier_treatment(col)
```

```
num_distinct_items 6.738483351642369 -1.3893049212488329
min_item_price 1984.7162927261527 -614.8408331464695
max_item_price 2561.84826918368 -242.07428025898298
total_onshift_partners 131.2287596680753 -41.41820862636135
total_busy_partners 122.2318663187919 -38.54099890469359
total_outstanding_orders 189.97766003550976 -73.56406101594642
total_items 9.889340174782097 -3.4801553897065225
subtotal 7268.805398041197 -1875.807520051092
```

```
for col in numbers_c:

    print(col ,data[col].agg(['min' ,'max' ,'mean' ,'median' ,'std']))
```

```
num_distinct_items min       1.000000
max       6.738483
mean      2.635269
median    2.000000
std       1.484216
Name: num_distinct_items, dtype: float64
min_item_price min          1.000000
max       1984.716293
mean       669.351694
median     595.000000
std        451.102654
Name: min_item_price, dtype: float64
max_item_price min          1.000000
max       2561.848269
mean      1142.603869
median    1095.000000
std        486.728207
Name: max_item_price, dtype: float64
total_onshift_partners min         1.000000
max       131.228760
mean       44.787840
median     37.000000
std        34.126192
Name: total_onshift_partners, dtype: float64
total_busy_partners min         1.000000
max       122.231866
mean       41.758560
median     35.000000
std        31.825884
Name: total_busy_partners, dtype: float64
total_outstanding_orders min         1.000000
max       189.977660
mean       57.644187
median     41.000000
std        50.952133
Name: total_outstanding_orders, dtype: float64
total_items min       1.000000
max       9.889340
mean      3.117674
median    3.000000
std       2.033854
Name: total_items, dtype: float64
subtotal min          1.000000
max       7268.805398
mean      2639.114411
median    2221.000000
std       1617.397752
Name: subtotal, dtype: float64
```

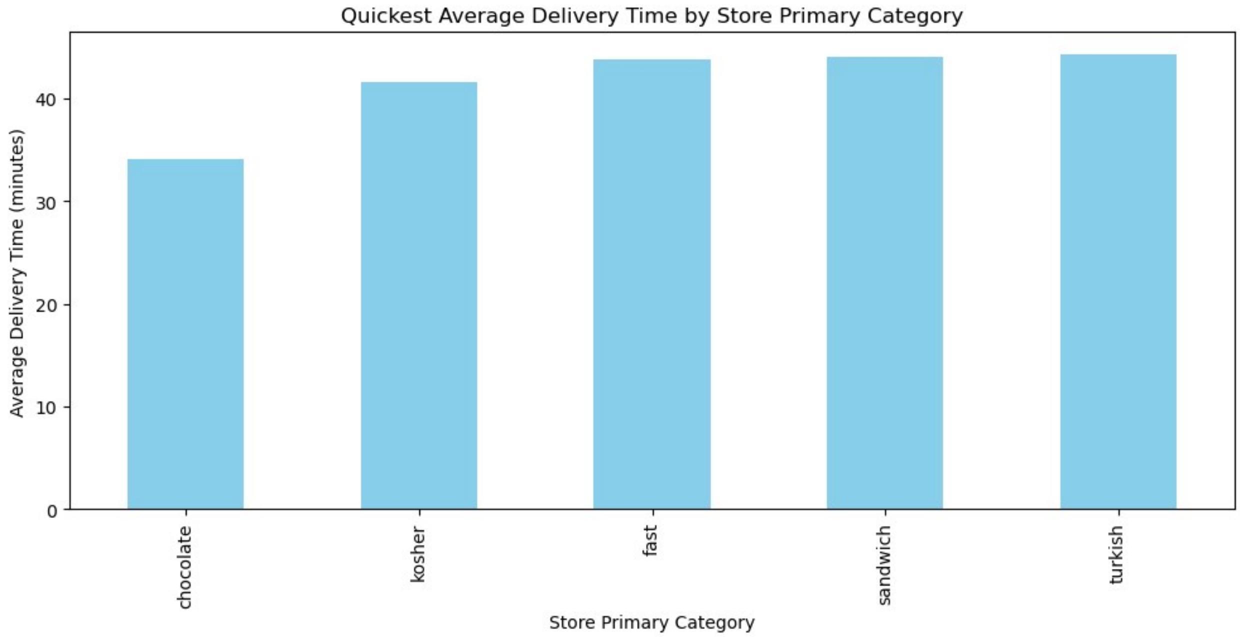
```
import matplotlib.pyplot as plt
import seaborn as sns

n_cols = 4
n_rows = (len(numbers_c) + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 10))


axes = axes.flatten()


for i, feature in enumerate(numbers_c):
    # Plot boxplot
    sns.boxplot(data=data, x=feature, ax=axes[i])
    axes[i].set_title(f'Boxplot of {feature}')



for i in range(len(numbers_c), len(axes)):
    fig.delaxes(axes[i])


plt.tight_layout()
plt.show()
```

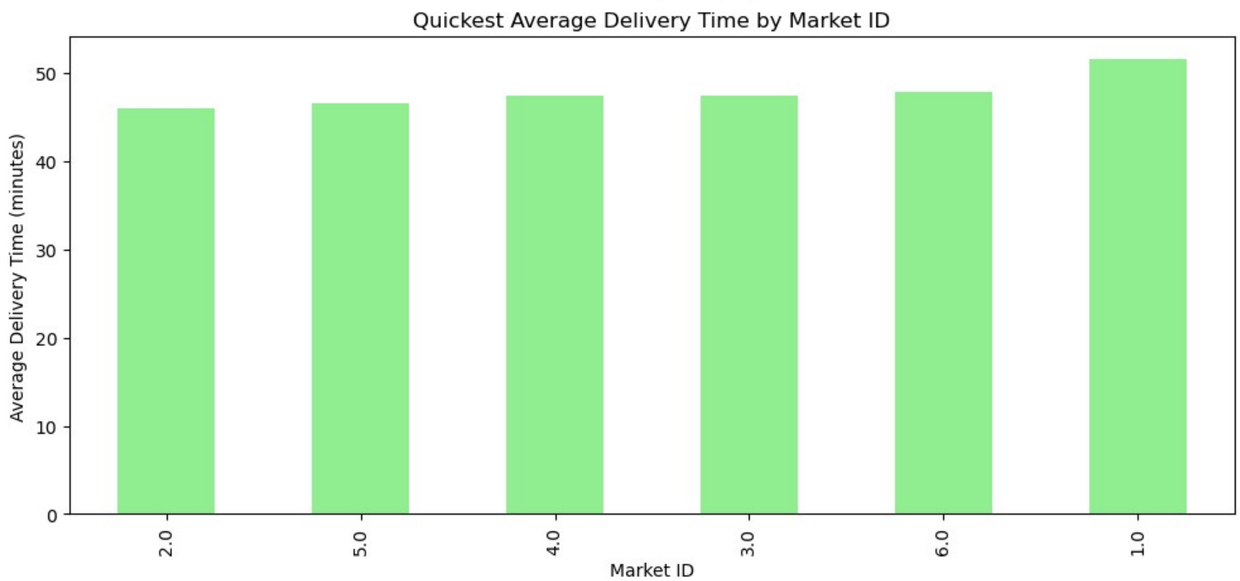


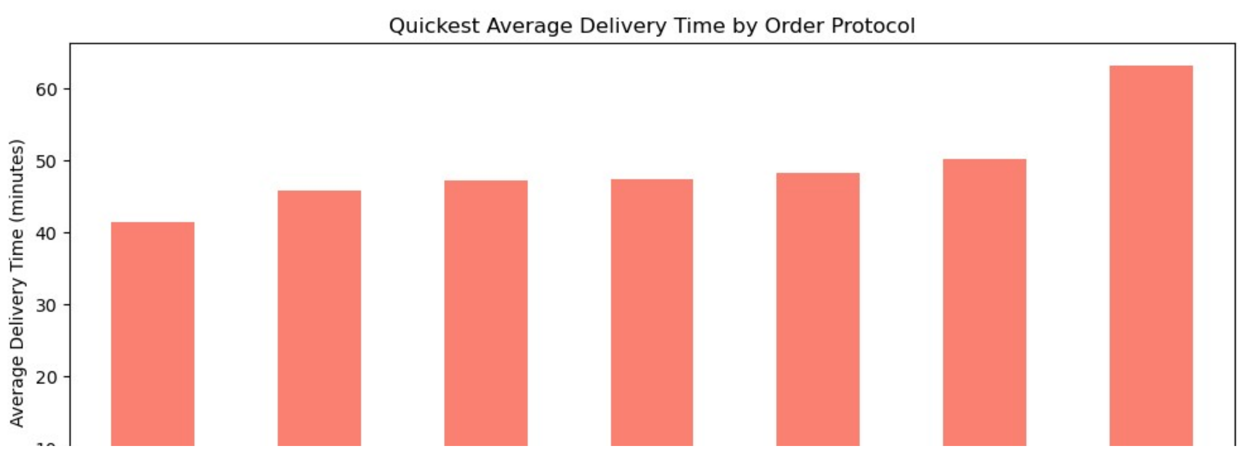

```
category_c
```

```
['market_id', 'store_id', 'store_primary_category']
```

```
import matplotlib.pyplot as plt


store_category_avg = data.groupby('store_primary_category')
['delivery_time_in_minutes'].mean().sort_values(ascending=True).head(5
)


market_avg = data.groupby('market_id')
['delivery_time_in_minutes'].mean().sort_values(ascending=True)


order_protocol_avg = data.groupby('order_protocol')
['delivery_time_in_minutes'].mean().sort_values(ascending=True)


fig, axes = plt.subplots(3, 1, figsize=(10, 15))


store_category_avg.plot(kind='bar', ax=axes[0], color='skyblue')
axes[0].set_title('Quickest Average Delivery Time by Store Primary
Category')
axes[0].set_ylabel('Average Delivery Time (minutes)')
axes[0].set_xlabel('Store Primary Category')


market_avg.plot(kind='bar', ax=axes[1], color='lightgreen')
axes[1].set_title('Quickest Average Delivery Time by Market ID')
axes[1].set_ylabel('Average Delivery Time (minutes)')
axes[1].set_xlabel('Market ID')


order_protocol_avg.plot(kind='bar', ax=axes[2], color='salmon')
axes[2].set_title('Quickest Average Delivery Time by Order Protocol')
axes[2].set_ylabel('Average Delivery Time (minutes)')
axes[2].set_xlabel('Order Protocol')

# Adjust layout
plt.tight_layout()

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

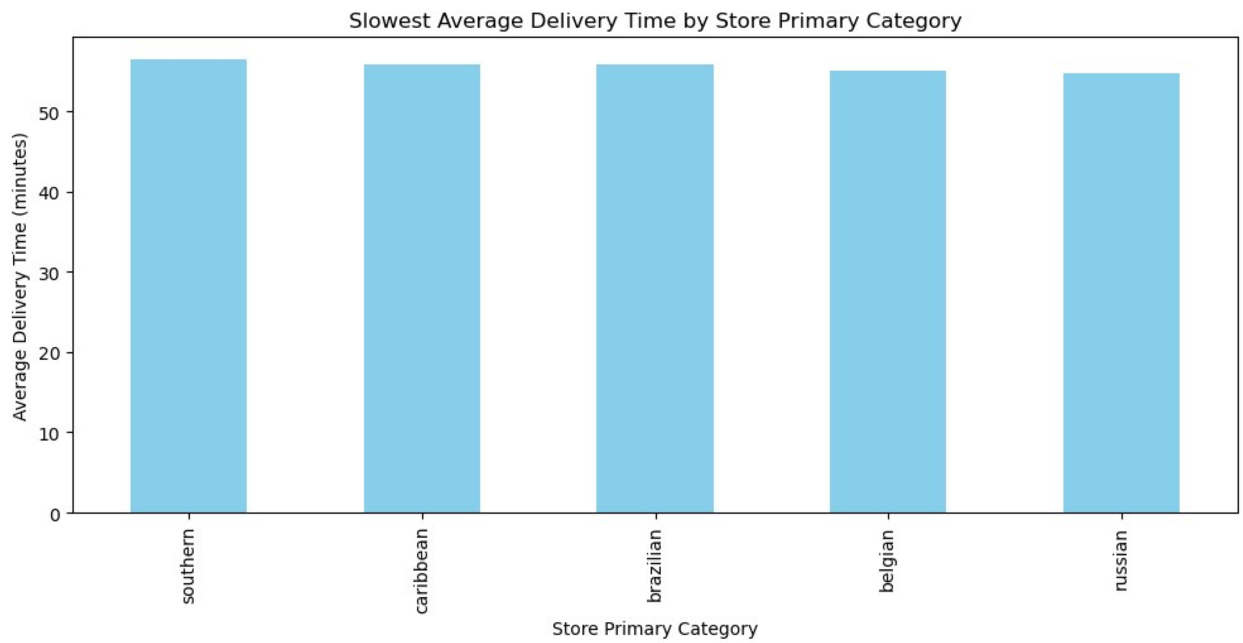

Quickest Average Delivery Time by Store Primary Category
Average Delivery Time (minutes)
0
10
20
30
40
chocolate
kosher
fast
sandwich
turkish
Store Primary Category

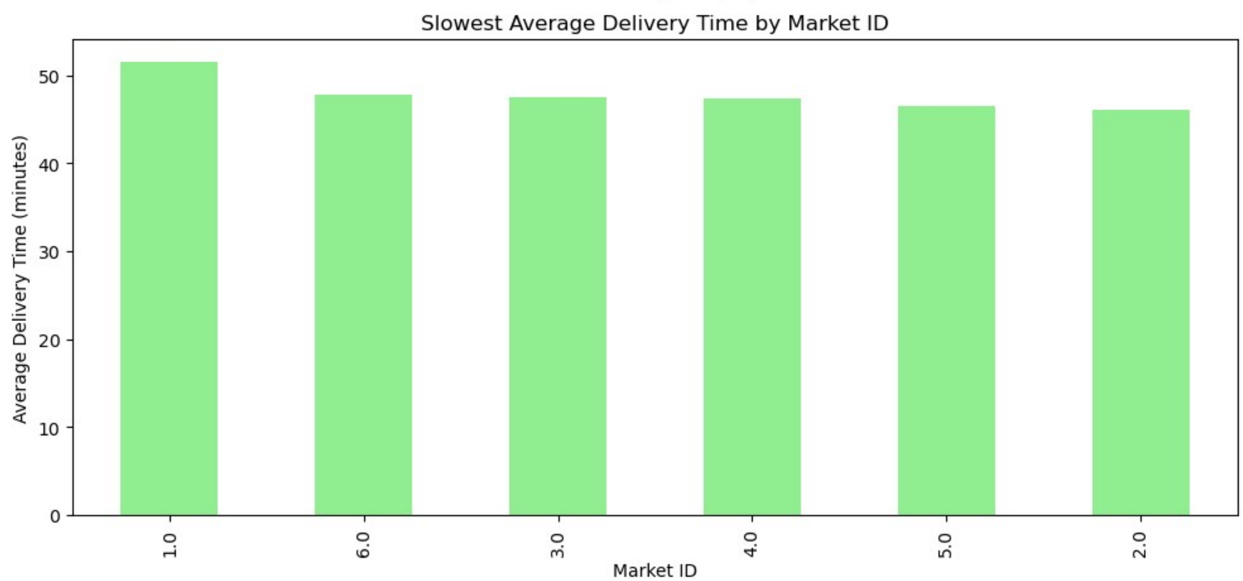

Quickest Average Delivery Time by Market ID
Average Delivery Time (minutes)
0
10
20
30
40
50
2.0
5.0
4.0
3.0
6.0
1.0
Market ID

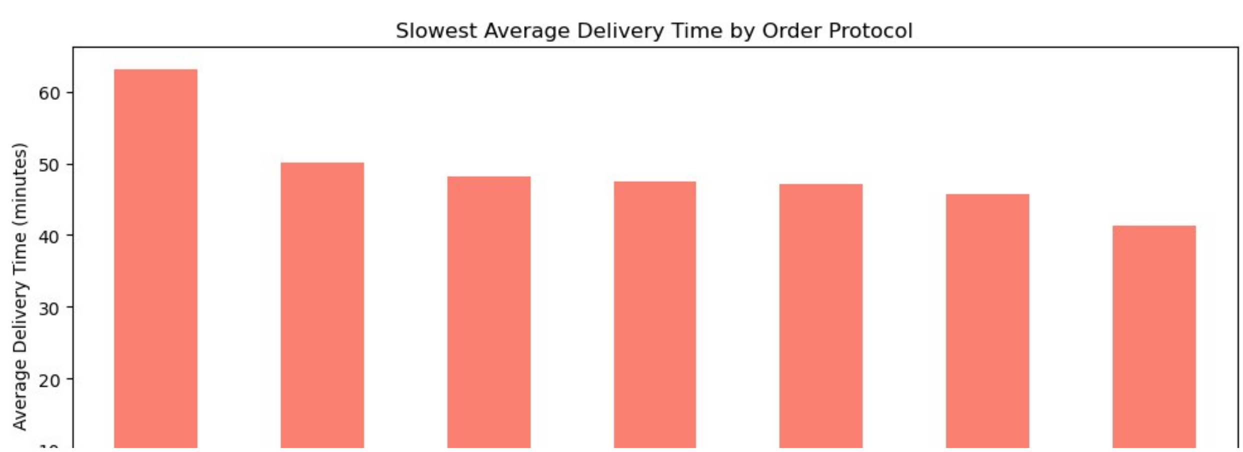

Quickest Average Delivery Time by Order Protocol
Average Delivery Time (minutes)
20
30
40
50
60

- Chocolate primary category take least avg delivery time
- market id 2 take least avg delivery time
- Order protocal 7 take least avg delivery time

```
import matplotlib.pyplot as plt


store_category_avg = data.groupby('store_primary_category')
['delivery_time_in_minutes'].mean().sort_values(ascending=False).head(
5)


market_avg = data.groupby('market_id')
['delivery_time_in_minutes'].mean().sort_values(ascending=False)


order_protocol_avg = data.groupby('order_protocol')
['delivery_time_in_minutes'].mean().sort_values(ascending=False)


fig, axes = plt.subplots(3, 1, figsize=(10, 15))


store_category_avg.plot(kind='bar', ax=axes[0], color='skyblue')
axes[0].set_title('Slowest Average Delivery Time by Store Primary
Category')
axes[0].set_ylabel('Average Delivery Time (minutes)')
axes[0].set_xlabel('Store Primary Category')


market_avg.plot(kind='bar', ax=axes[1], color='lightgreen')
axes[1].set_title('Slowest Average Delivery Time by Market ID')
axes[1].set_ylabel('Average Delivery Time (minutes)')
axes[1].set_xlabel('Market ID')


order_protocol_avg.plot(kind='bar', ax=axes[2], color='salmon')
axes[2].set_title('Slowest Average Delivery Time by Order Protocol')
axes[2].set_ylabel('Average Delivery Time (minutes)')
axes[2].set_xlabel('Order Protocol')

# Adjust layout
plt.tight_layout()

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


Slowest Average Delivery Time by Store Primary Category
Average Delivery Time (minutes)
0
10
20
30
40
50
southern
caribbean
brazilian
belgian
russian
Store Primary Category


Slowest Average Delivery Time by Market ID
Average Delivery Time (minutes)
0
10
20
30
40
50
1.0
6.0
3.0
4.0
5.0
2.0
Market ID


Slowest Average Delivery Time by Order Protocol
Average Delivery Time (minutes)
20
30
40
50
60

- southern primary category take highest avg delivery time
- market id 1 take highest avg delivery time
- Order protocal 6 take highest avg delivery time

date wise feature

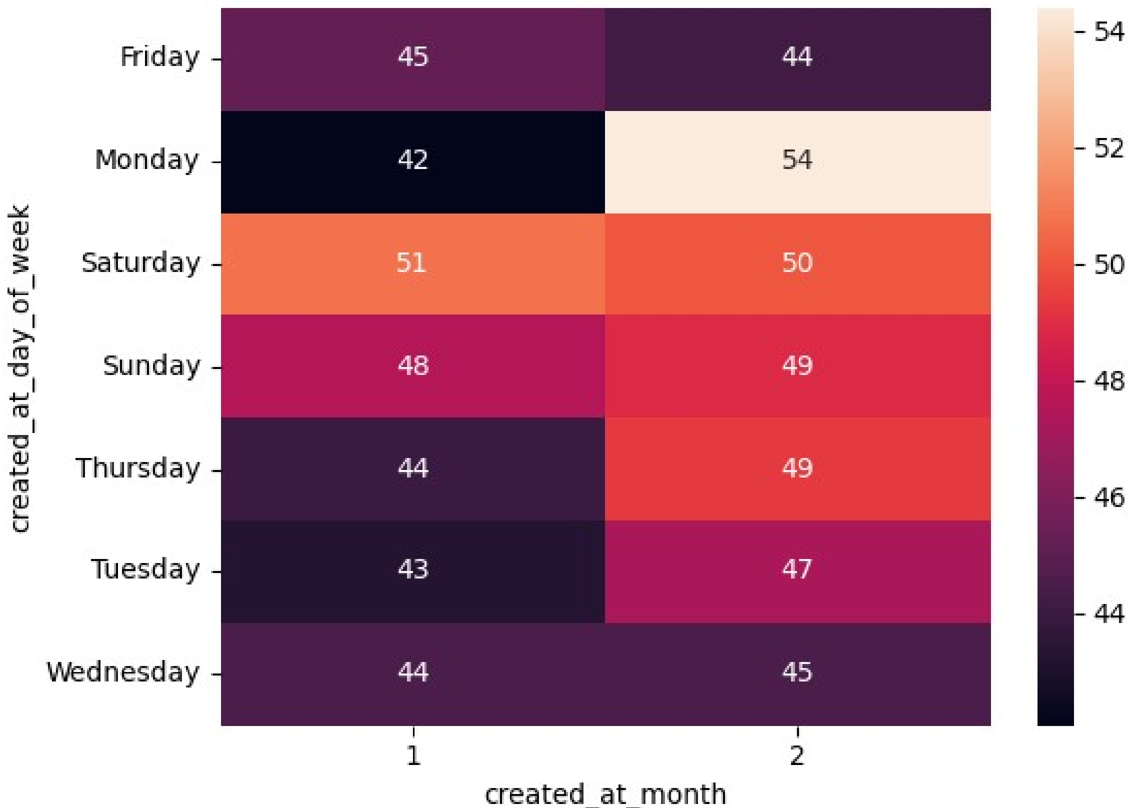
```
data['created_at_day_of_week'] = data['created_at'].dt.day_name()
data['created_at_hour_of_day'] = data['created_at'].dt.hour
data['created_at_day_of_month'] = data['created_at'].dt.day
data['created_at_month'] = data['created_at'].dt.monthr



data.drop(columns=['created_at'  ,'actual_delivery_time'    ])
```

```
        market_id                          store_id
store_primary_category  \
0             1.0  df263d996281d984952c07998dc54358
american
1             2.0  f0ade77b43923b38237db569b016ba25
mexican
8             2.0  f0ade77b43923b38237db569b016ba25
indian
14            1.0  ef1e491a766ce3127556063d49bc2f98
italian
15            1.0  ef1e491a766ce3127556063d49bc2f98
italian
...           ...                               ...
...
197423        1.0  a914ecef9c12ffdb9bede64bb703d877
fast
197424        1.0  a914ecef9c12ffdb9bede64bb703d877
fast
197425        1.0  a914ecef9c12ffdb9bede64bb703d877
fast
197426        1.0  c81e155d85dae5430a8cee6f2242e82c
sandwich
197427        1.0  c81e155d85dae5430a8cee6f2242e82c
sandwich

        order_protocol  total_items  subtotal  num_distinct_items  \
0                  1.0          4.0    3441.0                 4.0
1                  2.0          1.0    1900.0                 1.0
8                  3.0          4.0    4771.0                 3.0
14                 1.0          1.0    1525.0                 1.0
15                 1.0          2.0    3620.0                 2.0
...                ...          ...       ...                 ...
197423             4.0          3.0    1389.0                 3.0
```

```
197424             4.0           6.0    3010.0                 4.0
197425             4.0           5.0    1836.0                 3.0
197426             1.0           1.0    1175.0                 1.0
197427             1.0           4.0    2605.0                 4.0

        min_item_price  max_item_price  total_onshift_partners  \
0                557.0          1239.0                    33.0
1               1400.0          1400.0                     1.0
8                820.0          1604.0                     8.0
14              1525.0          1525.0                     5.0
15              1425.0          2195.0                     5.0
...                ...             ...                     ...
197423           345.0           649.0                    17.0
197424           405.0           825.0                    12.0
197425           300.0           399.0                    39.0
197426           535.0           535.0                     7.0
197427           425.0           750.0                    20.0

        total_busy_partners  total_outstanding_orders  \
0                      14.0                      21.0
1                       2.0                       2.0
8                       6.0                      18.0
14                      6.0                       8.0
15                      5.0                       7.0
...                     ...                       ...
197423                 17.0                      23.0
197424                 11.0                      14.0
197425                 41.0                      40.0
197426                  7.0                      12.0
197427                 20.0                      23.0

        delivery_time_in_minutes created_at_day_of_week  \
0                      62.983333                 Friday
1                      67.066667                Tuesday
8                      26.433333                 Monday
14                     37.883333               Thursday
15                     49.800000                Tuesday
...                          ...                    ...
197423                 65.116667                Tuesday
197424                 56.383333                 Friday
197425                 50.133333               Saturday
197426                 65.116667                 Sunday
197427                 37.133333                 Sunday

        created_at_hour_of_day  created_at_day_of_month
created_at_month  \
0                           22                        6
2
1                           21                       10
2
```

```
8                                   0                             16
2
14                                  3                             12
2
15                                  2                             27
1
...                               ...                            ...
...
197423                              0                             17
2
197424                              0                             13
2
197425                              4                             24
1
197426                             18                              1
2
197427                             19                              8
2

        created_at_quarter
0                        1
1                        1
8                        1
14                       1
15                       1
...                    ...
197423                   1
197424                   1
197425                   1
197426                   1
197427                   1

[176248 rows x 18 columns]


display(data['created_at_day_of_week'].value_counts())
display(data['created_at_hour_of_day'].value_counts())
pivot = pd.pivot_table(data , values='delivery_time_in_minutes' ,
index='created_at_day_of_week' , columns='created_at_month')

sns.heatmap(pivot  , annot=True)

created_at_day_of_week
Saturday     30858
Sunday       29898
Friday       25012
Monday       24202
Thursday     22997
Wednesday    21796
```

```
Tuesday      21485
Name: count, dtype: int64

created_at_hour_of_day
2     32940
1     25734
3     23719
20    14014
4     13254
19    12214
0     11466
21    10274
22     7877
23     7338
5      6079
18     4546
17     3071
16     1945
6      1223
15      504
14       39
7         9
8         2
Name: count, dtype: int64

<Axes: xlabel='created_at_month', ylabel='created_at_day_of_week'>
```



- night time most of the request is created, morning and evening very few
- Friday saturday and sunday most of the request is created
- highest avg ttd is for monday and months 2
- Request created on saturday take most amount of to deliver

```
data['created_at_month'].value_counts()

created_at_month
2    113960
1     62288
Name: count, dtype: int64

sns.heatmap(data , )
```

Feature engineering

```
data['subtotal_num_distinct_items_ratio']
=data['subtotal']/data['num_distinct_items']
data['price_range'] = data['max_item_price'] / data['min_item_price']
data['order_density'] = data['total_items'] /
data['total_outstanding_orders']
data['distinct_item_ratio'] = data['num_distinct_items'] /
```

```
data['total_items']
data['average_item_price'] = data['subtotal'] / data['total_items']
data['busy_partner_ratio'] = data['total_busy_partners'] /
data['total_onshift_partners']
data['item_price_per_partner'] = data['subtotal'] /
data['total_onshift_partners']
data['orders_per_distinct_item'] = data['total_items'] /
data['num_distinct_items']
data['item_partner_load'] = data['total_items'] /
data['total_onshift_partners']
data['log_subtotal'] = np.log(data['subtotal'] + 1)
data['log_total_items'] = np.log(data['total_items'] + 1)
data['log_total_outstanding_orders'] =
np.log(data['total_outstanding_orders'] + 1)

fef =['subtotal_num_distinct_items_ratio',

    'price_range',
    'order_density',
    'distinct_item_ratio',

    'average_item_price',

    'busy_partner_ratio',
    'item_price_per_partner',
    'orders_per_distinct_item',
    'item_partner_load',
    'log_subtotal',
    'log_total_items',
    'log_total_outstanding_orders',
]

data.drop(columns=['created_at', 'actual_delivery_time'] , inplace
=True)


# target encode
category_c =['market_id', 'store_id','store_primary_category',
'order_protocol',  'created_at_day_of_week', 'created_at_hour_of_day',
       'created_at_day_of_month', 'created_at_month',
'created_at_quarter']

# standardize
numbers_c = numbers_c+fef

target ='delivery_time_in_minutes'
```

droping row where traget values is greater than 3 std

droping extreame values

```
mu =data[target].mean()
std =data[target].std()

three_z = mu +3 *std
index = data[data[target]> three_z].index

data.drop( index = index , inplace=True)

for col in category_c:
    data[col] =data[col].astype('category')


from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# train test split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import
StandardScaler,TargetEncoder,OneHotEncoder

 from sklearn.model_selection import train_test_split

X = data.drop(columns=target)
y = data[target]
```

train test val split

```
# First split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state= 1)

# Now split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=1)
```

data pipeline

```
ct = ColumnTransformer(
    transformers=[

        # ('target_encoder',
OneHotEncoder(handle_unknown='ignore' ,sparse_output=False),
category_c),
```

```
        ('target_encoder',TargetEncoder(), category_c),
        ('scaler', StandardScaler(), numbers_c)
    ])


pipeline = Pipeline(steps=[
    ('preprocessor', ct)
])


X_train_transformed = pipeline.fit_transform(X_train, y_train)
X_val_transformed =pipeline.transform(X_val)

X_test_transformed =pipeline.transform(X_test)




import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.optimizers import RMSprop,Adam


import os

from tensorflow.keras.callbacks import EarlyStopping, TensorBoard

from tensorflow.keras import regularizers
from tensorflow.keras import layers, regularizers, callbacks
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping,
ReduceLROnPlateau

import random

# Set the seed for all relevant libraries
random.seed(42)
np.random.seed(42)
tf.random.set_seed(42)
os.environ['PYTHONHASHSEED'] = '42'
```

Model training

```


model = tf.keras.Sequential([
    layers.Input(shape=(X_train_transformed.shape[1],)),
    layers.Dense(96, kernel_regularizer=regularizers.l2(0.002)),  #
```

```
Adjust L2 regularization
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.6),  # Adjust dropout rate
    layers.Dense(64, kernel_regularizer=regularizers.l2(0.002)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.6),
    layers.Dense(48, kernel_regularizer=regularizers.l2(0.002)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.6),  # Adjust dropout rate
    layers.Dense(48, kernel_regularizer=regularizers.l2(0.002)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.3),
    layers.Dense(16, kernel_regularizer=regularizers.l2(0.002)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dense(1)
])

# Set up the TensorBoard callback
log_dir = "logs/fit"  # Directory to store logs
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

# Early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
)

# Reduce learning rate on plateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=7,
    min_lr=1e-6
)

# Use Adam optimizer
optim = Adam(learning_rate=0.001)
model.compile(optimizer=optim, loss='huber',
metrics=['mean_absolute_percentage_error'])

# Fit the model with the TensorBoard callback
history = model.fit(
    X_train_transformed,
    y_train,
```

```
    validation_data=(X_val_transformed, y_val),
    epochs=200,  # Increase number of epochs
    batch_size=256,  # Adjust batch size
    callbacks=[tensorboard_callback, reduce_lr, early_stopping]
)
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

```
Epoch 1/200
412/412 [==============================] - 3s 4ms/step - loss: 45.5019
- mean_absolute_percentage_error: 96.0499 - val_loss: 42.3696 -
val_mean_absolute_percentage_error: 88.4090 - lr: 0.0010
Epoch 2/200
412/412 [==============================] - 1s 3ms/step - loss: 38.4510
- mean_absolute_percentage_error: 78.8653 - val_loss: 31.7065 -
val_mean_absolute_percentage_error: 63.3258 - lr: 0.0010
Epoch 3/200
412/412 [==============================] - 1s 3ms/step - loss: 25.4008
- mean_absolute_percentage_error: 48.9809 - val_loss: 14.8660 -
val_mean_absolute_percentage_error: 28.2399 - lr: 0.0010
Epoch 4/200
412/412 [==============================] - 1s 3ms/step - loss: 13.8988
- mean_absolute_percentage_error: 27.9169 - val_loss: 12.1200 -
val_mean_absolute_percentage_error: 30.8193 - lr: 0.0010
Epoch 5/200
412/412 [==============================] - 2s 4ms/step - loss: 11.4458
- mean_absolute_percentage_error: 25.4722 - val_loss: 11.6204 -
val_mean_absolute_percentage_error: 24.2490 - lr: 0.0010
Epoch 6/200
412/412 [==============================] - 1s 3ms/step - loss: 11.1783
- mean_absolute_percentage_error: 25.4044 - val_loss: 12.8486 -
val_mean_absolute_percentage_error: 25.0374 - lr: 0.0010
Epoch 7/200
412/412 [==============================] - 1s 3ms/step - loss: 11.0375
- mean_absolute_percentage_error: 25.2042 - val_loss: 10.8974 -
val_mean_absolute_percentage_error: 24.0379 - lr: 0.0010
Epoch 8/200
412/412 [==============================] - 1s 3ms/step - loss: 10.9474
- mean_absolute_percentage_error: 25.0272 - val_loss: 11.3455 -
val_mean_absolute_percentage_error: 23.5839 - lr: 0.0010
Epoch 9/200
412/412 [==============================] - 1s 3ms/step - loss: 10.8369
- mean_absolute_percentage_error: 24.7629 - val_loss: 11.2973 -
val_mean_absolute_percentage_error: 23.4417 - lr: 0.0010
Epoch 10/200
412/412 [==============================] - 1s 3ms/step - loss: 10.7698
- mean_absolute_percentage_error: 24.6415 - val_loss: 14.1642 -
```

```
val_mean_absolute_percentage_error: 27.1846 - lr: 0.0010
Epoch 11/200
412/412 [==============================] - 1s 3ms/step - loss: 10.7024
- mean_absolute_percentage_error: 24.5057 - val_loss: 13.8911 -
val_mean_absolute_percentage_error: 27.1295 - lr: 0.0010
Epoch 12/200
412/412 [==============================] - 1s 3ms/step - loss: 10.6441
- mean_absolute_percentage_error: 24.4065 - val_loss: 12.9824 -
val_mean_absolute_percentage_error: 34.8116 - lr: 0.0010
Epoch 13/200
412/412 [==============================] - 1s 3ms/step - loss: 10.6640
- mean_absolute_percentage_error: 24.4650 - val_loss: 13.9963 -
val_mean_absolute_percentage_error: 26.6001 - lr: 0.0010
Epoch 14/200
412/412 [==============================] - 1s 3ms/step - loss: 10.5897
- mean_absolute_percentage_error: 24.3465 - val_loss: 11.2847 -
val_mean_absolute_percentage_error: 28.6592 - lr: 0.0010
Epoch 15/200
412/412 [==============================] - 1s 3ms/step - loss: 10.5517
- mean_absolute_percentage_error: 24.2012 - val_loss: 10.7557 -
val_mean_absolute_percentage_error: 26.0987 - lr: 5.0000e-04
Epoch 16/200
412/412 [==============================] - 1s 3ms/step - loss: 10.5164
- mean_absolute_percentage_error: 24.1760 - val_loss: 10.2968 -
val_mean_absolute_percentage_error: 23.4684 - lr: 5.0000e-04
Epoch 17/200
412/412 [==============================] - 1s 3ms/step - loss: 10.5144
- mean_absolute_percentage_error: 24.1763 - val_loss: 10.3718 -
val_mean_absolute_percentage_error: 23.4586 - lr: 5.0000e-04
Epoch 18/200
412/412 [==============================] - 1s 3ms/step - loss: 10.5099
- mean_absolute_percentage_error: 24.1745 - val_loss: 14.4205 -
val_mean_absolute_percentage_error: 39.8719 - lr: 5.0000e-04
Epoch 19/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4690
- mean_absolute_percentage_error: 24.0654 - val_loss: 10.7293 -
val_mean_absolute_percentage_error: 26.2769 - lr: 5.0000e-04
Epoch 20/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4638
- mean_absolute_percentage_error: 24.0727 - val_loss: 10.3127 -
val_mean_absolute_percentage_error: 23.3673 - lr: 5.0000e-04
Epoch 21/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4517
- mean_absolute_percentage_error: 24.0524 - val_loss: 10.6492 -
val_mean_absolute_percentage_error: 26.1102 - lr: 5.0000e-04
Epoch 22/200
412/412 [==============================] - 1s 4ms/step - loss: 10.4528
- mean_absolute_percentage_error: 24.0835 - val_loss: 10.4494 -
val_mean_absolute_percentage_error: 22.7079 - lr: 5.0000e-04
```

```
Epoch 23/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4476
- mean_absolute_percentage_error: 24.0486 - val_loss: 10.5824 -
val_mean_absolute_percentage_error: 25.5899 - lr: 5.0000e-04
Epoch 24/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4309
- mean_absolute_percentage_error: 23.9925 - val_loss: 10.3850 -
val_mean_absolute_percentage_error: 24.1302 - lr: 2.5000e-04
Epoch 25/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4152
- mean_absolute_percentage_error: 23.9843 - val_loss: 10.2955 -
val_mean_absolute_percentage_error: 23.7815 - lr: 2.5000e-04
Epoch 26/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4078
- mean_absolute_percentage_error: 23.9562 - val_loss: 10.3052 -
val_mean_absolute_percentage_error: 23.0049 - lr: 2.5000e-04
Epoch 27/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3902
- mean_absolute_percentage_error: 23.9401 - val_loss: 10.3271 -
val_mean_absolute_percentage_error: 22.5282 - lr: 2.5000e-04
Epoch 28/200
412/412 [==============================] - 1s 3ms/step - loss: 10.4020
- mean_absolute_percentage_error: 23.9906 - val_loss: 10.4179 -
val_mean_absolute_percentage_error: 22.8977 - lr: 2.5000e-04
Epoch 29/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3933
- mean_absolute_percentage_error: 23.9626 - val_loss: 10.4054 -
val_mean_absolute_percentage_error: 23.7482 - lr: 2.5000e-04
Epoch 30/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3932
- mean_absolute_percentage_error: 23.9415 - val_loss: 10.7171 -
val_mean_absolute_percentage_error: 26.2554 - lr: 2.5000e-04
Epoch 31/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3769
- mean_absolute_percentage_error: 23.9004 - val_loss: 10.2593 -
val_mean_absolute_percentage_error: 22.9929 - lr: 2.5000e-04
Epoch 32/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3696
- mean_absolute_percentage_error: 23.9387 - val_loss: 10.3862 -
val_mean_absolute_percentage_error: 22.6357 - lr: 2.5000e-04
Epoch 33/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3849
- mean_absolute_percentage_error: 23.9438 - val_loss: 10.3722 -
val_mean_absolute_percentage_error: 23.4134 - lr: 2.5000e-04
Epoch 34/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3746
- mean_absolute_percentage_error: 23.9375 - val_loss: 10.3954 -
val_mean_absolute_percentage_error: 24.5443 - lr: 2.5000e-04
Epoch 35/200
```

```
412/412 [==============================] - 1s 3ms/step - loss: 10.3439
- mean_absolute_percentage_error: 23.8636 - val_loss: 10.3322 -
val_mean_absolute_percentage_error: 23.2283 - lr: 2.5000e-04
Epoch 36/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3634
- mean_absolute_percentage_error: 23.8916 - val_loss: 10.4021 -
val_mean_absolute_percentage_error: 22.5644 - lr: 2.5000e-04
Epoch 37/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3628
- mean_absolute_percentage_error: 23.8958 - val_loss: 10.8325 -
val_mean_absolute_percentage_error: 27.1243 - lr: 2.5000e-04
Epoch 38/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3521
- mean_absolute_percentage_error: 23.8833 - val_loss: 10.6326 -
val_mean_absolute_percentage_error: 22.5032 - lr: 2.5000e-04
Epoch 39/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3327
- mean_absolute_percentage_error: 23.8854 - val_loss: 10.2064 -
val_mean_absolute_percentage_error: 23.4223 - lr: 1.2500e-04
Epoch 40/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3476
- mean_absolute_percentage_error: 23.8931 - val_loss: 10.4049 -
val_mean_absolute_percentage_error: 24.9367 - lr: 1.2500e-04
Epoch 41/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3363
- mean_absolute_percentage_error: 23.8624 - val_loss: 10.3033 -
val_mean_absolute_percentage_error: 23.5769 - lr: 1.2500e-04
Epoch 42/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3350
- mean_absolute_percentage_error: 23.8247 - val_loss: 10.3157 -
val_mean_absolute_percentage_error: 23.9910 - lr: 1.2500e-04
Epoch 43/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3425
- mean_absolute_percentage_error: 23.8696 - val_loss: 10.2901 -
val_mean_absolute_percentage_error: 24.2575 - lr: 1.2500e-04
Epoch 44/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3157
- mean_absolute_percentage_error: 23.8338 - val_loss: 10.2665 -
val_mean_absolute_percentage_error: 22.8231 - lr: 1.2500e-04
Epoch 45/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3159
- mean_absolute_percentage_error: 23.8165 - val_loss: 10.2380 -
val_mean_absolute_percentage_error: 22.8858 - lr: 1.2500e-04
Epoch 46/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3127
- mean_absolute_percentage_error: 23.7999 - val_loss: 10.5459 -
val_mean_absolute_percentage_error: 25.5876 - lr: 1.2500e-04
Epoch 47/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3197
```

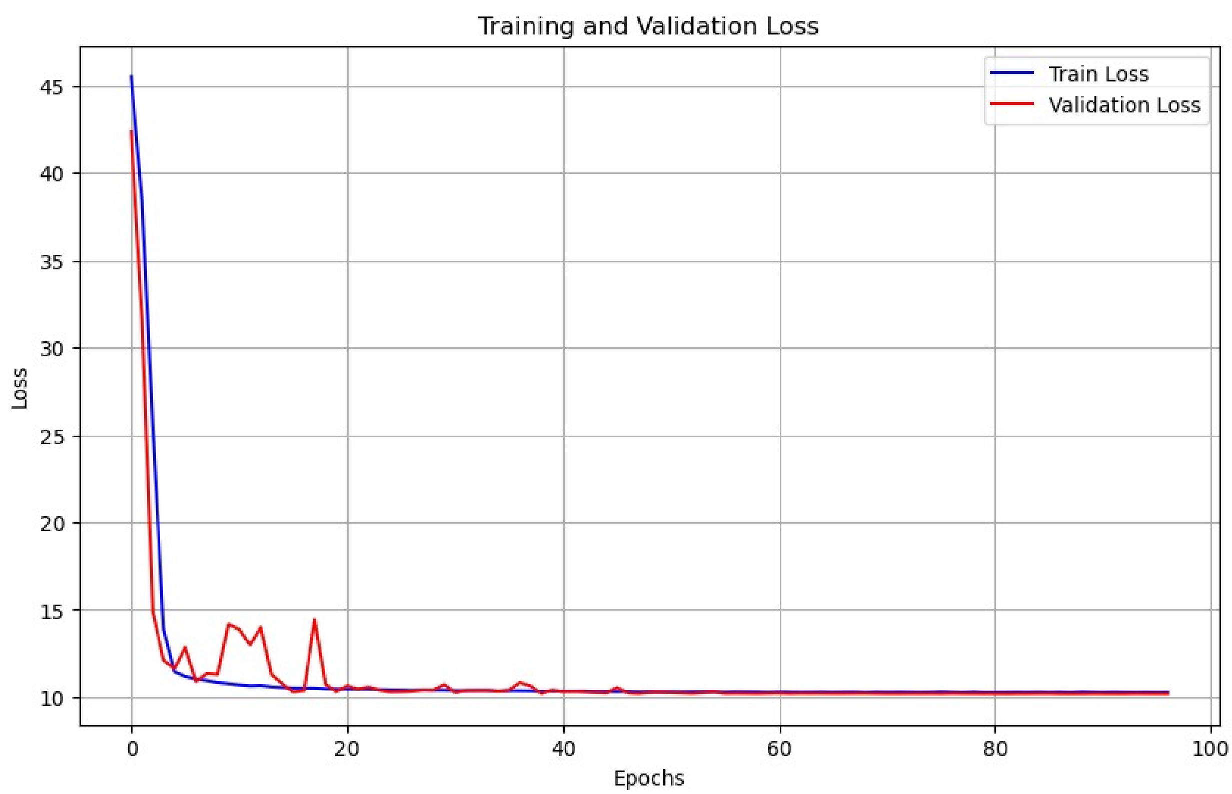
```
- mean_absolute_percentage_error: 23.8443 - val_loss: 10.2405 -
val_mean_absolute_percentage_error: 22.8213 - lr: 6.2500e-05
Epoch 48/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3011
- mean_absolute_percentage_error: 23.7773 - val_loss: 10.2014 -
val_mean_absolute_percentage_error: 23.1022 - lr: 6.2500e-05
Epoch 49/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3085
- mean_absolute_percentage_error: 23.8039 - val_loss: 10.2663 -
val_mean_absolute_percentage_error: 23.8198 - lr: 6.2500e-05
Epoch 50/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2955
- mean_absolute_percentage_error: 23.7607 - val_loss: 10.2832 -
val_mean_absolute_percentage_error: 23.7475 - lr: 6.2500e-05
Epoch 51/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3037
- mean_absolute_percentage_error: 23.7815 - val_loss: 10.2505 -
val_mean_absolute_percentage_error: 23.6372 - lr: 6.2500e-05
Epoch 52/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2994
- mean_absolute_percentage_error: 23.7935 - val_loss: 10.2358 -
val_mean_absolute_percentage_error: 23.6582 - lr: 6.2500e-05
Epoch 53/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3059
- mean_absolute_percentage_error: 23.7994 - val_loss: 10.2091 -
val_mean_absolute_percentage_error: 23.0258 - lr: 6.2500e-05
Epoch 54/200
412/412 [==============================] - 1s 4ms/step - loss: 10.3086
- mean_absolute_percentage_error: 23.8279 - val_loss: 10.2549 -
val_mean_absolute_percentage_error: 23.8807 - lr: 6.2500e-05
Epoch 55/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2995
- mean_absolute_percentage_error: 23.7622 - val_loss: 10.3027 -
val_mean_absolute_percentage_error: 24.0755 - lr: 6.2500e-05
Epoch 56/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2909
- mean_absolute_percentage_error: 23.7827 - val_loss: 10.2056 -
val_mean_absolute_percentage_error: 23.3094 - lr: 3.1250e-05
Epoch 57/200
412/412 [==============================] - 1s 3ms/step - loss: 10.3020
- mean_absolute_percentage_error: 23.7992 - val_loss: 10.2169 -
val_mean_absolute_percentage_error: 23.4723 - lr: 3.1250e-05
Epoch 58/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2975
- mean_absolute_percentage_error: 23.7825 - val_loss: 10.2089 -
val_mean_absolute_percentage_error: 23.2162 - lr: 3.1250e-05
Epoch 59/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2955
- mean_absolute_percentage_error: 23.7565 - val_loss: 10.2006 -
```

```
val_mean_absolute_percentage_error: 22.9658 - lr: 3.1250e-05
Epoch 60/200
412/412 [==============================] - 2s 4ms/step - loss: 10.2815
- mean_absolute_percentage_error: 23.7395 - val_loss: 10.2156 -
val_mean_absolute_percentage_error: 23.2778 - lr: 3.1250e-05
Epoch 61/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2979
- mean_absolute_percentage_error: 23.7910 - val_loss: 10.2275 -
val_mean_absolute_percentage_error: 22.9437 - lr: 3.1250e-05
Epoch 62/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2872
- mean_absolute_percentage_error: 23.7732 - val_loss: 10.2014 -
val_mean_absolute_percentage_error: 23.2721 - lr: 3.1250e-05
Epoch 63/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2817
- mean_absolute_percentage_error: 23.7471 - val_loss: 10.2152 -
val_mean_absolute_percentage_error: 23.0408 - lr: 3.1250e-05
Epoch 64/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2840
- mean_absolute_percentage_error: 23.7644 - val_loss: 10.2101 -
val_mean_absolute_percentage_error: 23.2229 - lr: 3.1250e-05
Epoch 65/200
412/412 [==============================] - 2s 4ms/step - loss: 10.2918
- mean_absolute_percentage_error: 23.7758 - val_loss: 10.2130 -
val_mean_absolute_percentage_error: 23.0369 - lr: 3.1250e-05
Epoch 66/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2805
- mean_absolute_percentage_error: 23.7435 - val_loss: 10.2046 -
val_mean_absolute_percentage_error: 23.0712 - lr: 3.1250e-05
Epoch 67/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2871
- mean_absolute_percentage_error: 23.7586 - val_loss: 10.2041 -
val_mean_absolute_percentage_error: 23.1700 - lr: 1.5625e-05
Epoch 68/200
412/412 [==============================] - 2s 4ms/step - loss: 10.2923
- mean_absolute_percentage_error: 23.7772 - val_loss: 10.2096 -
val_mean_absolute_percentage_error: 23.2930 - lr: 1.5625e-05
Epoch 69/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2717
- mean_absolute_percentage_error: 23.7285 - val_loss: 10.2150 -
val_mean_absolute_percentage_error: 23.0492 - lr: 1.5625e-05
Epoch 70/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2897
- mean_absolute_percentage_error: 23.7694 - val_loss: 10.2032 -
val_mean_absolute_percentage_error: 23.1127 - lr: 1.5625e-05
Epoch 71/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2823
- mean_absolute_percentage_error: 23.7401 - val_loss: 10.1981 -
val_mean_absolute_percentage_error: 23.1373 - lr: 1.5625e-05
```

```
Epoch 72/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2876
- mean_absolute_percentage_error: 23.7611 - val_loss: 10.1962 -
val_mean_absolute_percentage_error: 23.1585 - lr: 1.5625e-05
Epoch 73/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2843
- mean_absolute_percentage_error: 23.7540 - val_loss: 10.2007 -
val_mean_absolute_percentage_error: 23.0917 - lr: 1.5625e-05
Epoch 74/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2745
- mean_absolute_percentage_error: 23.7425 - val_loss: 10.1998 -
val_mean_absolute_percentage_error: 23.2571 - lr: 1.5625e-05
Epoch 75/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2815
- mean_absolute_percentage_error: 23.7537 - val_loss: 10.2037 -
val_mean_absolute_percentage_error: 23.3629 - lr: 1.5625e-05
Epoch 76/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2993
- mean_absolute_percentage_error: 23.7822 - val_loss: 10.1961 -
val_mean_absolute_percentage_error: 23.2181 - lr: 1.5625e-05
Epoch 77/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2847
- mean_absolute_percentage_error: 23.7739 - val_loss: 10.2102 -
val_mean_absolute_percentage_error: 23.2393 - lr: 1.5625e-05
Epoch 78/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2696
- mean_absolute_percentage_error: 23.7206 - val_loss: 10.2012 -
val_mean_absolute_percentage_error: 23.0784 - lr: 1.5625e-05
Epoch 79/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2916
- mean_absolute_percentage_error: 23.7967 - val_loss: 10.1984 -
val_mean_absolute_percentage_error: 23.1269 - lr: 1.5625e-05
Epoch 80/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2659
- mean_absolute_percentage_error: 23.7366 - val_loss: 10.1941 -
val_mean_absolute_percentage_error: 23.1929 - lr: 1.5625e-05
Epoch 81/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2689
- mean_absolute_percentage_error: 23.7340 - val_loss: 10.1952 -
val_mean_absolute_percentage_error: 23.3100 - lr: 1.5625e-05
Epoch 82/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2757
- mean_absolute_percentage_error: 23.7458 - val_loss: 10.1885 -
val_mean_absolute_percentage_error: 23.1655 - lr: 1.5625e-05
Epoch 83/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2810
- mean_absolute_percentage_error: 23.7721 - val_loss: 10.1955 -
val_mean_absolute_percentage_error: 23.1937 - lr: 1.5625e-05
Epoch 84/200
```

```
412/412 [==============================] - 1s 3ms/step - loss: 10.2755
- mean_absolute_percentage_error: 23.7485 - val_loss: 10.1960 -
val_mean_absolute_percentage_error: 23.0903 - lr: 1.5625e-05
Epoch 85/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2859
- mean_absolute_percentage_error: 23.7702 - val_loss: 10.2028 -
val_mean_absolute_percentage_error: 23.3552 - lr: 1.5625e-05
Epoch 86/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2730
- mean_absolute_percentage_error: 23.7398 - val_loss: 10.2020 -
val_mean_absolute_percentage_error: 23.1834 - lr: 1.5625e-05
Epoch 87/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2850
- mean_absolute_percentage_error: 23.7731 - val_loss: 10.1910 -
val_mean_absolute_percentage_error: 23.0761 - lr: 1.5625e-05
Epoch 88/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2675
- mean_absolute_percentage_error: 23.7307 - val_loss: 10.1889 -
val_mean_absolute_percentage_error: 23.1098 - lr: 1.5625e-05
Epoch 89/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2943
- mean_absolute_percentage_error: 23.7867 - val_loss: 10.1904 -
val_mean_absolute_percentage_error: 23.1857 - lr: 1.5625e-05
Epoch 90/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2805
- mean_absolute_percentage_error: 23.7585 - val_loss: 10.1961 -
val_mean_absolute_percentage_error: 23.2187 - lr: 7.8125e-06
Epoch 91/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2711
- mean_absolute_percentage_error: 23.7383 - val_loss: 10.1925 -
val_mean_absolute_percentage_error: 23.1579 - lr: 7.8125e-06
Epoch 92/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2827
- mean_absolute_percentage_error: 23.7699 - val_loss: 10.1921 -
val_mean_absolute_percentage_error: 23.0886 - lr: 7.8125e-06
Epoch 93/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2742
- mean_absolute_percentage_error: 23.7535 - val_loss: 10.1902 -
val_mean_absolute_percentage_error: 23.1771 - lr: 7.8125e-06
Epoch 94/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2702
- mean_absolute_percentage_error: 23.7341 - val_loss: 10.1985 -
val_mean_absolute_percentage_error: 23.1432 - lr: 7.8125e-06
Epoch 95/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2753
- mean_absolute_percentage_error: 23.7494 - val_loss: 10.1948 -
val_mean_absolute_percentage_error: 23.2383 - lr: 7.8125e-06
Epoch 96/200
412/412 [==============================] - 2s 4ms/step - loss: 10.2714
- mean_absolute_percentage_error: 23.7358 - val_loss: 10.1942 -
```

```
val_mean_absolute_percentage_error: 23.2145 - lr: 7.8125e-06
Epoch 97/200
412/412 [==============================] - 1s 3ms/step - loss: 10.2763
- mean_absolute_percentage_error: 23.7478 - val_loss: 10.1917 -
val_mean_absolute_percentage_error: 23.2495 - lr: 3.9063e-06


train_loss = history.history['loss']
val_loss = history.history['val_loss']


plt.figure(figsize=(10, 6))
plt.plot(train_loss, label='Train Loss', color='blue')
plt.plot(val_loss, label='Validation Loss', color='red')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```




```
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error
```

```
# Make predictions on the training set
y_train_pred = model.predict(X_train_transformed)

# Make predictions on the test set
y_test_pred = model.predict(X_test_transformed)

# Make predictions on the validation set
y_val_pred = model.predict(X_val_transformed)

# Calculate MAPE and MSE on the training set
train_mape = mean_absolute_percentage_error(y_train, y_train_pred) *
100
train_mse = mean_squared_error(y_train, y_train_pred)

# Calculate MAPE and MSE on the test set
test_mape = mean_absolute_percentage_error(y_test, y_test_pred) * 100
test_mse = mean_squared_error(y_test, y_test_pred)

# Calculate MAPE and MSE on the validation set
val_mape = mean_absolute_percentage_error(y_val, y_val_pred) * 100
val_mse = mean_squared_error(y_val, y_val_pred)

print(f"Training MAPE: {train_mape:.2f}%")
print(f"Training MSE: {train_mse:.2f}")
print(f"Test MAPE: {test_mape:.2f}%")
print(f"Test MSE: {test_mse:.2f}")
print(f"Validation MAPE: {val_mape:.2f}%")
print(f"Validation MSE: {val_mse:.2f}")

3294/3294 [==============================] - 2s 501us/step
1373/1373 [==============================] - 1s 483us/step
824/824 [==============================] - 0s 496us/step
Training MAPE: 23.22%
Training MSE: 208.76
Test MAPE: 23.12%
Test MSE: 212.75
Validation MAPE: 23.17%
Validation MSE: 210.99
```

Tuning model

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, regularizers, callbacks
from tensorflow.keras.optimizers.legacy  import Adam,RMSprop
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping,
```

```
ReduceLROnPlateau
from hyperopt import fmin, tpe, hp, Trials, STATUS_OK




space = {
    'learning_rate': hp.loguniform('learning_rate', np.log(1e-5),
np.log(1e-2)),
    'l2_reg': hp.loguniform('l2_reg', np.log(1e-5), np.log(1e-2)),
    'dropout_rate': hp.uniform('dropout_rate', 0.2, 0.6),
    'batch_size': hp.choice('batch_size', [64, 128, 256, 512]),
    'optimizer': hp.choice('optimizer', ['adam', 'rmsprop'])
}


def objective(params):
    # Create the model
    model = tf.keras.Sequential([
        layers.Input(shape=(X_train_transformed.shape[1],)),
        layers.Dense(96,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(64,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(48,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(48,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(32,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dense(16,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dense(1)
    ])
```

```
    # Set up the optimizer
    if params['optimizer'] == 'adam':
        optim = Adam(learning_rate=params['learning_rate'])
    else:
        optim = RMSprop(learning_rate=params['learning_rate'])

    # Compile the model
    model.compile(optimizer=optim, loss='huber',
metrics=['mean_absolute_percentage_error'])

    # Set up the callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=5, min_lr=1e-6)

    # Fit the model
    history = model.fit(
        X_train_transformed,
        y_train,
        validation_data=(X_val_transformed, y_val),
        epochs=200,
        batch_size=params['batch_size'],
        callbacks=[early_stopping, reduce_lr],
        verbose=0
    )
    loss  , mape=model.evaluate(X_val_transformed, y_val)




    # Return the loss and status
    return {'loss': loss, 'status': STATUS_OK}

# Run the hyperparameter optimization
trials = Trials()
best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=15,
trials=trials)

# Print the best hyperparameters
print("Best hyperparameters:", best)

  1/824 [..............................] - ETA: 10s - loss: 11.6978 -
mean_absolute_percentage_error: 24.5296
 94/824 [==>...........................] - ETA: 0s - loss: 11.0300 -
mean_absolute_percentage_error: 26.0321
187/824 [=====>........................] - ETA: 0s - loss: 11.0696 -
mean_absolute_percentage_error: 26.1392
279/824 [=========>....................] - ETA: 0s - loss: 11.0034 -
```

```
mean_absolute_percentage_error: 25.9592
371/824 [============>.................] - ETA: 0s - loss: 10.9934 -
mean_absolute_percentage_error: 25.8721
467/824 [================>.............] - ETA: 0s - loss: 10.9936 -
mean_absolute_percentage_error: 25.7877
561/824 [===================>..........] - ETA: 0s - loss: 10.9990 -
mean_absolute_percentage_error: 25.7854
655/824 [======================>.......] - ETA: 0s - loss: 10.9661 -
mean_absolute_percentage_error: 25.7832
749/824 [==========================>...] - ETA: 0s - loss: 10.9635 -
mean_absolute_percentage_error: 25.7589
824/824 [==============================] - 0s 540us/step - loss:
10.9631 - mean_absolute_percentage_error: 25.7457

  1/824 [..............................] - ETA: 10s - loss: 12.2756 -
mean_absolute_percentage_error: 26.7200
 61/824 [=>............................] - ETA: 0s - loss: 11.0654 -
mean_absolute_percentage_error: 25.9609
151/824 [====>.........................] - ETA: 0s - loss: 10.9850 -
mean_absolute_percentage_error: 25.5305
241/824 [=======>......................] - ETA: 0s - loss: 10.9490 -
mean_absolute_percentage_error: 25.4633
333/824 [===========>..................] - ETA: 0s - loss: 10.9738 -
mean_absolute_percentage_error: 25.4286
425/824 [==============>...............] - ETA: 0s - loss: 10.9362 -
mean_absolute_percentage_error: 25.3210
516/824 [=================>............] - ETA: 0s - loss: 10.9178 -
mean_absolute_percentage_error: 25.3416
603/824 [====================>.........] - ETA: 0s - loss: 10.8912 -
mean_absolute_percentage_error: 25.2526
697/824 [========================>.....] - ETA: 0s - loss: 10.8732 -
mean_absolute_percentage_error: 25.2445
789/824 [===========================>..] - ETA: 0s - loss: 10.8797 -
mean_absolute_percentage_error: 25.2658
824/824 [==============================] - 0s 577us/step - loss:
10.8746 - mean_absolute_percentage_error: 25.2546

  1/824 [..............................] - ETA: 10s - loss: 12.5919 -
mean_absolute_percentage_error: 27.7226
 63/824 [=>............................] - ETA: 0s - loss: 11.2115 -
mean_absolute_percentage_error: 26.4554
137/824 [===>..........................] - ETA: 0s - loss: 11.0310 -
mean_absolute_percentage_error: 25.8811
231/824 [=======>......................] - ETA: 0s - loss: 10.9795 -
mean_absolute_percentage_error: 25.8721
326/824 [==========>...................] - ETA: 0s - loss: 11.0056 -
mean_absolute_percentage_error: 25.7400
418/824 [==============>...............] - ETA: 0s - loss: 10.9670 -
mean_absolute_percentage_error: 25.6236
```

```
512/824 [=================>............] - ETA: 0s - loss: 10.9499 -
mean_absolute_percentage_error: 25.6616
607/824 [=====================>........] - ETA: 0s - loss: 10.9214 -
mean_absolute_percentage_error: 25.5723
702/824 [========================>.....] - ETA: 0s - loss: 10.8918 -
mean_absolute_percentage_error: 25.5663
797/824 [============================>.] - ETA: 0s - loss: 10.9058 -
mean_absolute_percentage_error: 25.5877
824/824 [==============================] - 0s 573us/step - loss:
10.9001 - mean_absolute_percentage_error: 25.5762

  1/824 [..............................] - ETA: 7s - loss: 11.3411 -
mean_absolute_percentage_error: 23.6073
 90/824 [==>...........................] - ETA: 0s - loss: 10.0630 -
mean_absolute_percentage_error: 23.4892
206/824 [======>.......................] - ETA: 0s - loss: 10.1154 -
mean_absolute_percentage_error: 23.6380
337/824 [===========>..................] - ETA: 0s - loss: 10.0833 -
mean_absolute_percentage_error: 23.4490
469/824 [================>.............] - ETA: 0s - loss: 10.0579 -
mean_absolute_percentage_error: 23.3353
602/824 [====================>.........] - ETA: 0s - loss: 10.0656 -
mean_absolute_percentage_error: 23.3675
733/824 [=========================>....] - ETA: 0s - loss: 10.0414 -
mean_absolute_percentage_error: 23.3658
824/824 [==============================] - 0s 409us/step - loss:
10.0494 - mean_absolute_percentage_error: 23.3520

  1/824 [..............................] - ETA: 8s - loss: 11.7280 -
mean_absolute_percentage_error: 24.3358
 78/824 [=>............................] - ETA: 0s - loss: 10.1664 -
mean_absolute_percentage_error: 23.6390
190/824 [=====>........................] - ETA: 0s - loss: 10.1820 -
mean_absolute_percentage_error: 23.6402
328/824 [==========>...................] - ETA: 0s - loss: 10.1753 -
mean_absolute_percentage_error: 23.5180
458/824 [===============>..............] - ETA: 0s - loss: 10.1224 -
mean_absolute_percentage_error: 23.3608
576/824 [===================>..........] - ETA: 0s - loss: 10.1455 -
mean_absolute_percentage_error: 23.4344
709/824 [========================>.....] - ETA: 0s - loss: 10.1197 -
mean_absolute_percentage_error: 23.4102
824/824 [==============================] - 0s 422us/step - loss:
10.1232 - mean_absolute_percentage_error: 23.4028

  1/824 [..............................] - ETA: 7s - loss: 11.3166 -
mean_absolute_percentage_error: 23.3333
 54/824 [>.............................] - ETA: 0s - loss: 10.2576 -
mean_absolute_percentage_error: 23.8638
167/824 [=====>........................] - ETA: 0s - loss: 10.3151 -
```

```
mean_absolute_percentage_error: 24.0145
296/824 [=========>....................] - ETA: 0s - loss: 10.2827 -
mean_absolute_percentage_error: 23.8547
425/824 [==============>...............] - ETA: 0s - loss: 10.2595 -
mean_absolute_percentage_error: 23.7323
553/824 [===================>..........] - ETA: 0s - loss: 10.3001 -
mean_absolute_percentage_error: 23.8049
677/824 [=======================>......] - ETA: 0s - loss: 10.2499 -
mean_absolute_percentage_error: 23.7412
808/824 [============================>.] - ETA: 0s - loss: 10.2385 -
mean_absolute_percentage_error: 23.7207
824/824 [==============================] - 0s 438us/step - loss:
10.2377 - mean_absolute_percentage_error: 23.7240

  1/824 [..............................] - ETA: 7s - loss: 11.4609 -
mean_absolute_percentage_error: 23.3766
105/824 [==>...........................] - ETA: 0s - loss: 10.1792 -
mean_absolute_percentage_error: 23.3062
230/824 [=======>......................] - ETA: 0s - loss: 10.3052 -
mean_absolute_percentage_error: 23.6805
347/824 [===========>..................] - ETA: 0s - loss: 10.3494 -
mean_absolute_percentage_error: 23.6687
453/824 [===============>..............] - ETA: 0s - loss: 10.3188 -
mean_absolute_percentage_error: 23.5634
580/824 [====================>.........] - ETA: 0s - loss: 10.3312 -
mean_absolute_percentage_error: 23.6048
711/824 [========================>.....] - ETA: 0s - loss: 10.3212 -
mean_absolute_percentage_error: 23.6294
824/824 [==============================] - 0s 420us/step - loss:
10.3091 - mean_absolute_percentage_error: 23.5732

  1/824 [..............................] - ETA: 7s - loss: 11.6752 -
mean_absolute_percentage_error: 24.2038
 89/824 [==>...........................] - ETA: 0s - loss: 10.1765 -
mean_absolute_percentage_error: 23.4775
212/824 [======>.......................] - ETA: 0s - loss: 10.2430 -
mean_absolute_percentage_error: 23.6242
341/824 [===========>..................] - ETA: 0s - loss: 10.2080 -
mean_absolute_percentage_error: 23.4364
467/824 [================>.............] - ETA: 0s - loss: 10.2003 -
mean_absolute_percentage_error: 23.3347
595/824 [====================>.........] - ETA: 0s - loss: 10.1988 -
mean_absolute_percentage_error: 23.3583
711/824 [========================>.....] - ETA: 0s - loss: 10.1937 -
mean_absolute_percentage_error: 23.3986
824/824 [==============================] - 0s 424us/step - loss:
10.1932 - mean_absolute_percentage_error: 23.3694

  1/824 [..............................] - ETA: 7s - loss: 11.3515 -
mean_absolute_percentage_error: 23.6702
```

```
 94/824 [==>...........................] - ETA: 0s - loss: 10.0646 -
mean_absolute_percentage_error: 23.2594
217/824 [======>.......................] - ETA: 0s - loss: 10.0782 -
mean_absolute_percentage_error: 23.2691
346/824 [===========>..................] - ETA: 0s - loss: 10.0882 -
mean_absolute_percentage_error: 23.1915
466/824 [===============>..............] - ETA: 0s - loss: 10.0698 -
mean_absolute_percentage_error: 23.0800
594/824 [====================>.........] - ETA: 0s - loss: 10.0721 -
mean_absolute_percentage_error: 23.1167
721/824 [=========================>....] - ETA: 0s - loss: 10.0718 -
mean_absolute_percentage_error: 23.1873
824/824 [==============================] - 0s 416us/step - loss:
10.0666 - mean_absolute_percentage_error: 23.1224

  1/824 [..............................] - ETA: 7s - loss: 11.0405 -
mean_absolute_percentage_error: 23.0274
 82/824 [=>............................] - ETA: 0s - loss: 9.8844 -
mean_absolute_percentage_error: 23.0488
197/824 [======>.......................] - ETA: 0s - loss: 9.9715 -
mean_absolute_percentage_error: 23.2138
328/824 [==========>...................] - ETA: 0s - loss: 9.9330 -
mean_absolute_percentage_error: 23.0388
459/824 [===============>..............] - ETA: 0s - loss: 9.8769 -
mean_absolute_percentage_error: 22.8752
591/824 [====================>.........] - ETA: 0s - loss: 9.8980 -
mean_absolute_percentage_error: 22.9555
720/824 [=========================>....] - ETA: 0s - loss: 9.8987 -
mean_absolute_percentage_error: 23.0117
824/824 [==============================] - 0s 416us/step - loss:
9.9041 - mean_absolute_percentage_error: 22.9775

  1/824 [..............................] - ETA: 6s - loss: 11.5265 -
mean_absolute_percentage_error: 23.8310
125/824 [===>..........................] - ETA: 0s - loss: 10.6441 -
mean_absolute_percentage_error: 23.9269
255/824 [========>.....................] - ETA: 0s - loss: 10.6479 -
mean_absolute_percentage_error: 24.0022
387/824 [=============>................] - ETA: 0s - loss: 10.6408 -
mean_absolute_percentage_error: 23.9102
519/824 [=================>............] - ETA: 0s - loss: 10.6179 -
mean_absolute_percentage_error: 23.8077
651/824 [======================>.......] - ETA: 0s - loss: 10.6035 -
mean_absolute_percentage_error: 23.7874
781/824 [===========================>..] - ETA: 0s - loss: 10.5943 -
mean_absolute_percentage_error: 23.7786
824/824 [==============================] - 0s 390us/step - loss:
10.5900 - mean_absolute_percentage_error: 23.7612

  1/824 [..............................] - ETA: 7s - loss: 11.1694 -
```

```
mean_absolute_percentage_error: 23.1871
 95/824 [==>...........................] - ETA: 0s - loss: 9.9110 -
mean_absolute_percentage_error: 22.9263
226/824 [=======>......................] - ETA: 0s - loss: 9.9087 -
mean_absolute_percentage_error: 22.8905
361/824 [============>.................] - ETA: 0s - loss: 9.9272 -
mean_absolute_percentage_error: 22.8069
492/824 [================>.............] - ETA: 0s - loss: 9.9514 -
mean_absolute_percentage_error: 22.7765
624/824 [=====================>........] - ETA: 0s - loss: 9.9710 -
mean_absolute_percentage_error: 22.8303
756/824 [==========================>...] - ETA: 0s - loss: 9.9600 -
mean_absolute_percentage_error: 22.8215
824/824 [==============================] - 0s 399us/step - loss:
9.9564 - mean_absolute_percentage_error: 22.8214

  1/824 [..............................] - ETA: 7s - loss: 11.9372 -
mean_absolute_percentage_error: 26.5712
125/824 [===>..........................] - ETA: 0s - loss: 10.6333 -
mean_absolute_percentage_error: 25.2855
256/824 [========>.....................] - ETA: 0s - loss: 10.6521 -
mean_absolute_percentage_error: 25.4087
388/824 [=============>................] - ETA: 0s - loss: 10.6523 -
mean_absolute_percentage_error: 25.3155
521/824 [=================>............] - ETA: 0s - loss: 10.6563 -
mean_absolute_percentage_error: 25.2719
653/824 [======================>.......] - ETA: 0s - loss: 10.6378 -
mean_absolute_percentage_error: 25.2246
785/824 [===========================>..] - ETA: 0s - loss: 10.6398 -
mean_absolute_percentage_error: 25.2156
824/824 [==============================] - 0s 386us/step - loss:
10.6411 - mean_absolute_percentage_error: 25.2136

  1/824 [..............................] - ETA: 7s - loss: 11.5246 -
mean_absolute_percentage_error: 24.0769
102/824 [==>...........................] - ETA: 0s - loss: 9.8507 -
mean_absolute_percentage_error: 22.9421
216/824 [======>.......................] - ETA: 0s - loss: 9.9785 -
mean_absolute_percentage_error: 23.1973
347/824 [===========>..................] - ETA: 0s - loss: 10.0023 -
mean_absolute_percentage_error: 23.1370
477/824 [================>.............] - ETA: 0s - loss: 10.0046 -
mean_absolute_percentage_error: 23.0641
607/824 [=====================>........] - ETA: 0s - loss: 9.9952 -
mean_absolute_percentage_error: 23.0822
735/824 [=========================>....] - ETA: 0s - loss: 9.9858 -
mean_absolute_percentage_error: 23.1176
824/824 [==============================] - 0s 416us/step - loss:
9.9833 - mean_absolute_percentage_error: 23.0866
```

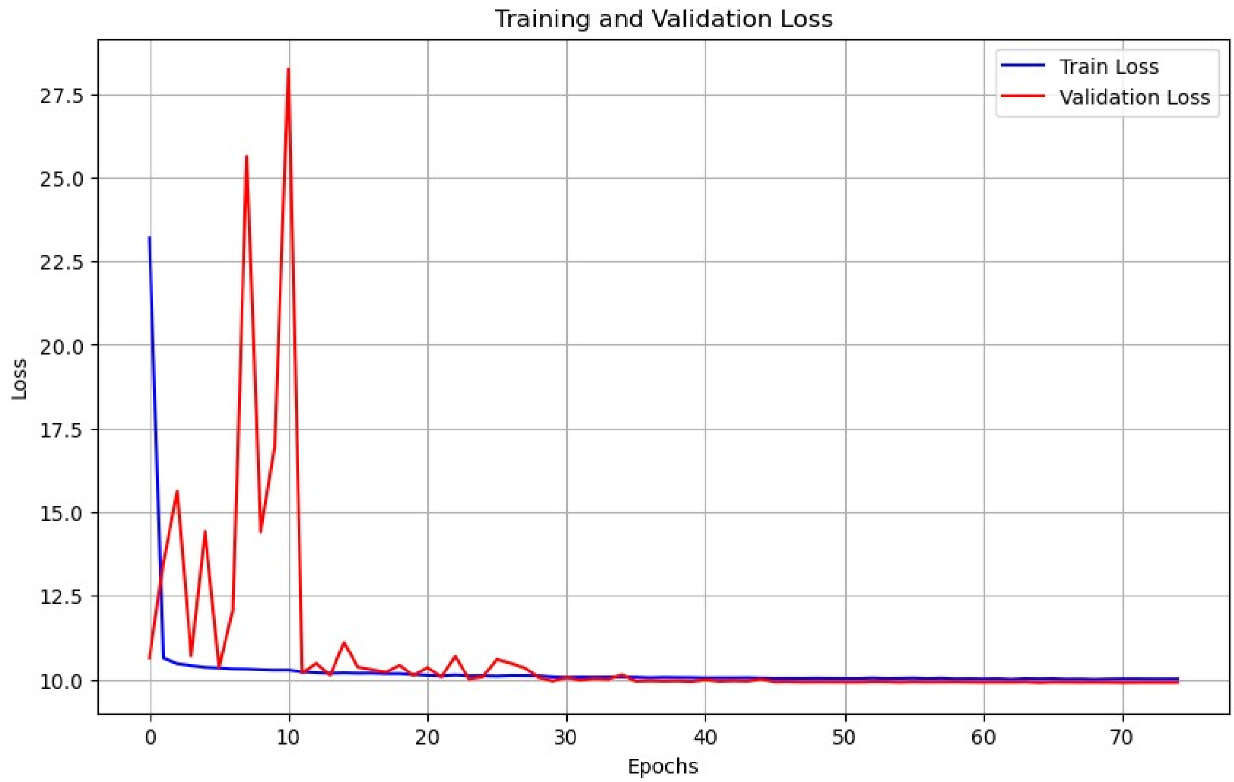
```
  1/824 [..............................] - ETA: 7s - loss: 11.8126 -
mean_absolute_percentage_error: 25.0106
103/824 [==>...........................] - ETA: 0s - loss: 10.0882 -
mean_absolute_percentage_error: 23.4991
189/824 [=====>........................] - ETA: 0s - loss: 10.2375 -
mean_absolute_percentage_error: 23.8350
317/824 [==========>...................] - ETA: 0s - loss: 10.2145 -
mean_absolute_percentage_error: 23.6532
444/824 [===============>..............] - ETA: 0s - loss: 10.1580 -
mean_absolute_percentage_error: 23.5114
572/824 [===================>..........] - ETA: 0s - loss: 10.1551 -
mean_absolute_percentage_error: 23.5254
697/824 [========================>.....] - ETA: 0s - loss: 10.1373 -
mean_absolute_percentage_error: 23.5241
819/824 [============================>.] - ETA: 0s - loss: 10.1364 -
mean_absolute_percentage_error: 23.5262
824/824 [==============================] - 0s 432us/step - loss:
10.1415 - mean_absolute_percentage_error: 23.5278

100%|██████████| 15/15 [27:44<00:00, 110.96s/trial, best loss:
9.904097557067871]
Best hyperparameters: {'batch_size': 0, 'dropout_rate':
0.22500761353444074, 'l2_reg': 0.0002961576602613677, 'learning_rate':
0.0015188747990346094, 'optimizer': 1}

best

{'batch_size': 0,
 'dropout_rate': 0.22500761353444074,
 'l2_reg': 0.0002961576602613677,
 'learning_rate': 0.0015188747990346094,
 'optimizer': 1}
```

best hyperparam

```
best={'batch_size': 0,
 'dropout_rate': 0.22500761353444074,
 'l2_reg': 0.0002961576602613677,
 'learning_rate': 0.0015188747990346094,
 'optimizer': 1}


# Map the optimizer index to the optimizer name
optimizer_choices = ['adam', 'rmsprop']
best_optimizer = optimizer_choices[best['optimizer']]
batch_choices =[64, 128, 256, 512]
best_batch_size =batch_choices[best['batch_size']]
```

```
best['optimizer'] =best_optimizer
best['batch_size'] =best_batch_size

def train_model(params):
    model = tf.keras.Sequential([
        layers.Input(shape=(X_train_transformed.shape[1],)),
        layers.Dense(96,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(64,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(48,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(48,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(params['dropout_rate']),
        layers.Dense(32,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dense(16,
kernel_regularizer=regularizers.l2(params['l2_reg'])),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dense(1)
    ])

    # Set up the optimizer
    if params['optimizer'] == 'adam':
        optim = Adam(learning_rate=params['learning_rate'])
    else:
        optim = RMSprop(learning_rate=params['learning_rate'])

    # Compile the model
    model.compile(optimizer=optim, loss='huber',
metrics=['mean_absolute_percentage_error'])

    # Set up the callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
```

```
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=5, min_lr=1e-6)

    # Fit the model
    history = model.fit(
        X_train_transformed,
        y_train,
        validation_data=(X_val_transformed, y_val),
        epochs=200,
        batch_size=params['batch_size'],
        callbacks=[early_stopping, reduce_lr],
        verbose=0
    )

    return history , model

history , model   =train_model(best)


train_loss = history.history['loss']
val_loss = history.history['val_loss']


plt.figure(figsize=(10, 6))
plt.plot(train_loss, label='Train Loss', color='blue')
plt.plot(val_loss, label='Validation Loss', color='red')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```




```
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error

# Make predictions on the training set
y_train_pred = model.predict(X_train_transformed)

# Make predictions on the test set
y_test_pred = model.predict(X_test_transformed)

# Make predictions on the validation set
y_val_pred = model.predict(X_val_transformed)

# Calculate MAPE and MSE on the training set
train_mape = mean_absolute_percentage_error(y_train, y_train_pred) *
100
train_mse = mean_squared_error(y_train, y_train_pred)

# Calculate MAPE and MSE on the test set
test_mape = mean_absolute_percentage_error(y_test, y_test_pred) * 100
test_mse = mean_squared_error(y_test, y_test_pred)

# Calculate MAPE and MSE on the validation set
val_mape = mean_absolute_percentage_error(y_val, y_val_pred) * 100
val_mse = mean_squared_error(y_val, y_val_pred)

print(f"Training MAPE: {train_mape:.2f}%")
```

```
print(f"Training MSE: {train_mse:.2f}")
print(f"Test MAPE: {test_mape:.2f}%")
print(f"Test MSE: {test_mse:.2f}")
print(f"Validation MAPE: {val_mape:.2f}%")

print(f"Validation MSE: {val_mse:.2f}")

3294/3294 [==============================] - 1s 374us/step
1373/1373 [==============================] - 1s 384us/step
824/824 [==============================] - 0s 373us/step
Training MAPE: 22.72%
Training MSE: 193.83
Test MAPE: 22.80%
Test MSE: 198.74
Validation MAPE: 22.86%
Validation MSE: 197.09
```

Model Performance Comparison

Base Model

- **Training MAPE**: 23.22%
- **Training MSE**: 208.76
- **Test MAPE**: 23.12%
- **Test MSE**: 212.75
- **Validation MAPE**: 23.17%
- **Validation MSE**: 210.99

Tuned Model

- **Training MAPE**: 22.72%
- **Training MSE**: 193.83
- **Test MAPE**: 22.80%
- **Test MSE**: 198.74
- **Validation MAPE**: 22.86%
- **Validation MSE**: 197.09

Summary of Improvements:

- **Training MAPE**: Improved by 0.50% (lower MAPE indicates better performance).
- **Training MSE**: Reduced by 14.93 (lower MSE indicates better performance).
- **Test MAPE**: Improved by 0.32% in the tuned model.
- **Test MSE**: Reduced by 14.01.
- **Validation MAPE**: Improved by 0.31% in the tuned model.
- **Validation MSE**: Reduced by 13.90.

Conclusion:

The tuned model shows improvements across all evaluation metrics, particularly in reducing both MAPE and MSE for training, testing, and validation data. This indicates better generalization and predictive performance after the tuning process.