Porter Case Study

Introduction

- Porter, India's largest marketplace for intra-city logistics, is revolutionizing the delivery sector with technology-driven solutions.
- This case focuses on leveraging neural networks to accurately predict delivery times, a critical aspect of customer satisfaction in logistics.
- With a dataset encompassing various aspects of orders and deliveries, Porter aims to refine its delivery time estimations.
- Analyzing this dataset can provide significant insights into delivery dynamics, efficiency bottlenecks, and optimization opportunities.
- The insights obtained can enhance Porter's operational efficiency, ensuring timely deliveries and improving driver-partner allocation.

What is expected?

- As a data scientist at Porter, your task is to analyze the dataset to accurately predict delivery times for different orders. Your primary goal is to build a regression model using neural networks, evaluate its performance, and provide insights for optimizing delivery operations.

1. Data

The analysis was done on the data located at - https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/015/039/original/dataset.csv.zip 1663710760

2. Libraries

Below are the libraries required

In [105…

```
# libraries to analyze data
import numpy as np
import pandas as pd

# libraries to visualize data
import matplotlib.pyplot as plt
```

```python
import seaborn as sns

from sklearn.neighbors import LocalOutlierFactor

from sklearn.impute import KNNImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, LeakyReLU, Dropout, BatchNor
from tensorflow.keras.optimizers import Adam, RMSprop
from sklearn.metrics import mean_squared_error
```

3. Data Loading

Loading the data into Pandas dataframe for easily handling of data

In [2]:

```python
# read the file into a pandas dataframe
df = pd.read_csv('dataset.csv')
# look at the datatypes of the columns
print('**************************************************')
print(df.info())
print('**************************************************\n')
print('**************************************************')
print(f'Shape of the dataset is {df.shape}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of nan/null values in each column: \n{df.isna().sum()}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of unique values in each column: \n{df.nunique()}')
print('**************************************************\n')
print('**************************************************')
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
```

```
************************************************
<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
None
************************************************

************************************************
Shape of the dataset is (197428, 14)
************************************************

************************************************
Number of nan/null values in each column:
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
************************************************

************************************************
Number of unique values in each column:
market_id                        6
created_at                  180985
actual_delivery_time        178110
store_id                      6743
store_primary_category          74
order_protocol                   7
```

```
total_items                    57
subtotal                     8368
num_distinct_items             20
min_item_price               2312
max_item_price               2652
total_onshift_partners        172
total_busy_partners           159
total_outstanding_orders      281
dtype: int64
*************************************************

*************************************************
Duplicate entries:
False    197428
Name: count, dtype: int64
```

In [3]:
```
# look at the top 20 rows
df.head(5)
```

Out[3]:

	market_id	created_at	actual_delivery_time	store_id	store_pı
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	

In [4]:
```
df.describe()
```

Out[4]:

	market_id	order_protocol	total_items	subtotal	num_distinct_items	m
count	196441.000000	196433.000000	197428.000000	197428.000000	197428.000000	1
mean	2.978706	2.882352	3.196391	2682.331402	2.670791	
std	1.524867	1.503771	2.666546	1823.093688	1.630255	
min	1.000000	1.000000	1.000000	0.000000	1.000000	
25%	2.000000	1.000000	2.000000	1400.000000	1.000000	
50%	3.000000	3.000000	3.000000	2200.000000	2.000000	
75%	4.000000	4.000000	4.000000	3395.000000	3.000000	
max	6.000000	7.000000	411.000000	27100.000000	20.000000	

In [5]:

```
df.describe(include='object')
```

Out[5]:

	created_at	actual_delivery_time	store_id	store_primary_
count	197428	197421	197428	
unique	180985	178110	6743	
top	2015-02-11 19:50:43	2015-02-11 20:40:45	d43ab110ab2489d6b9b2caa394bf920f	
freq	6	5	937	

In [6]:

```
min(df['created_at']), max(df['created_at'])
```

Out[6]:

```
('2014-10-19 05:24:15', '2015-02-18 06:00:44')
```

Insight

- There are **197428** entries with 14 columns
- The data is available between **19-Oct-2014 to 18-Feb-2015**, around 5 months of data
- There are null/missing values in each of the dates
- There are no **duplicates**
- The columns **market_id, total_onshift_partners, total_busy_partners** and **total_outstanding_orders** can be of type **int64**
- The columns **market_id, order_protocol** and **num_distinct_items** can be converted to categorical columns
- The columns **created_at** and **actual_delivery_time** need to be of type **datetime**
- Extract **hour** and **day** of the order placement from **created_at**
- Create **delivery_time_mins** column by subracting **created_at** from **actual_delivery_time**
- Columns **created_at** and **actual_delivery_time** can be dropped

In [7]:
```
df[['total_onshift_partners', 'total_busy_partners' , 'total_outstanding_orders']]
df[['market_id', 'order_protocol', 'num_distinct_items']] = df[['market_id', 'order
df[['created_at', 'actual_delivery_time']] = df[['created_at', 'actual_delivery_tim
df['created_hour']=df['created_at'].dt.hour
df['created_day']=df['created_at'].dt.dayofweek
df['delivery_time_mins'] = round((df['actual_delivery_time'] - df['created_at'])/pd
df.drop(columns=['created_at', 'actual_delivery_time'], inplace=True)
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  category
 1   store_id                  197428 non-null  object
 2   store_primary_category    192668 non-null  object
 3   order_protocol            196433 non-null  category
 4   total_items               197428 non-null  int64
 5   subtotal                  197428 non-null  int64
 6   num_distinct_items        197428 non-null  category
 7   min_item_price            197428 non-null  int64
 8   max_item_price            197428 non-null  int64
 9   total_onshift_partners    181166 non-null  Int64
 10  total_busy_partners       181166 non-null  Int64
 11  total_outstanding_orders  181166 non-null  Int64
 12  created_hour              197428 non-null  int32
 13  created_day               197428 non-null  int32
 14  delivery_time_mins        197421 non-null  float64
dtypes: Int64(3), category(3), float64(1), int32(2), int64(4), object(2)
memory usage: 17.7+ MB
```

In [8]:
```
df.describe()
```

Out[8]:

	total_items	subtotal	min_item_price	max_item_price	total_onshift_partners
count	197428.000000	197428.000000	197428.000000	197428.000000	181166.(
mean	3.196391	2682.331402	686.218470	1159.588630	44.80809:
std	2.666546	1823.093688	522.038648	558.411377	34.52678:
min	1.000000	0.000000	-86.000000	0.000000	-4.(
25%	2.000000	1400.000000	299.000000	800.000000	17.(
50%	3.000000	2200.000000	595.000000	1095.000000	37.(
75%	4.000000	3395.000000	949.000000	1395.000000	65.(
max	411.000000	27100.000000	14700.000000	14700.000000	171.(

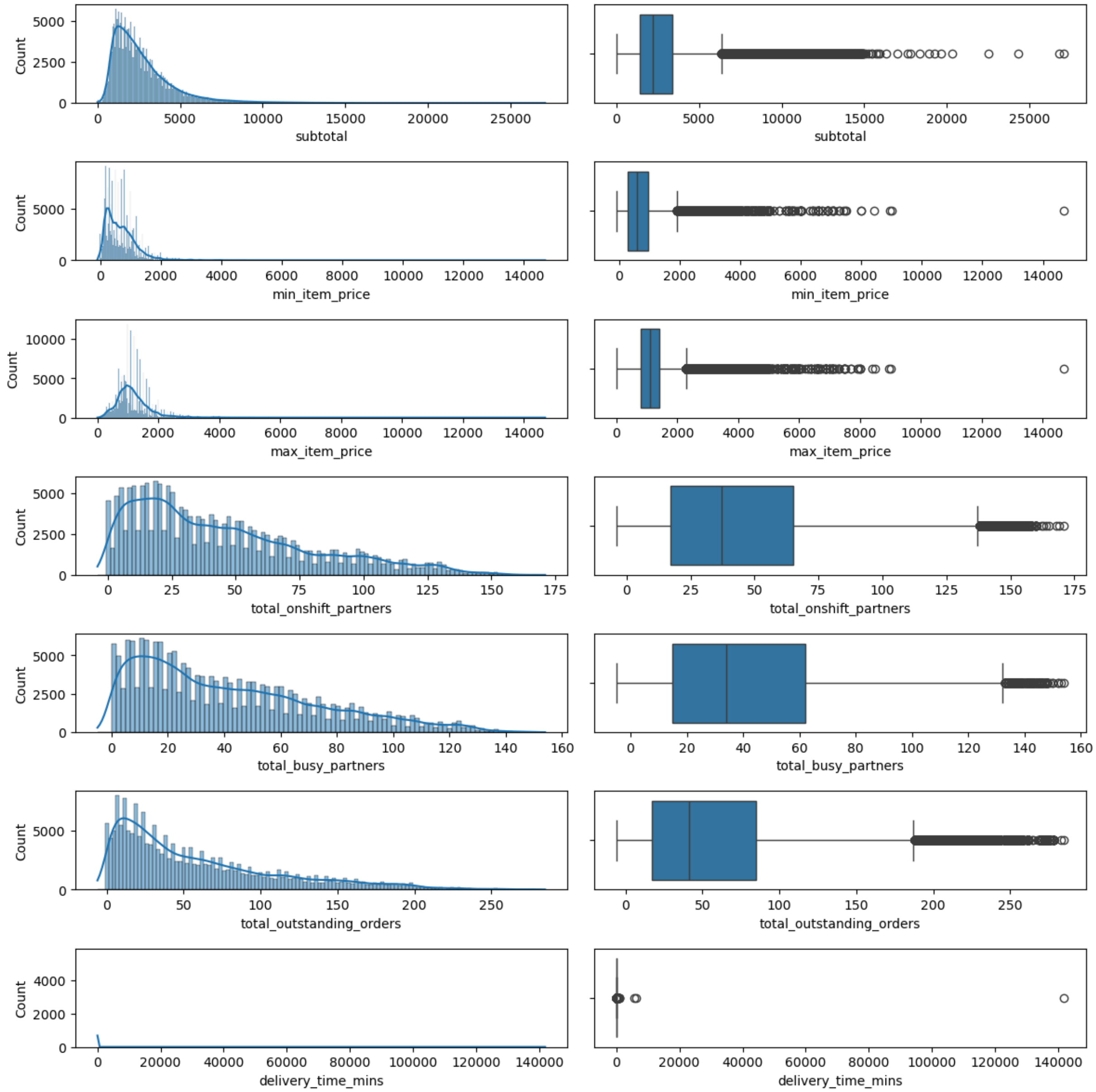
4. Exploratory Data Analysis

4.1. Univariate Analysis

4.1.1. Numerical variables

In [9]:

```
fig, axes = plt.subplots(nrows=7, ncols=2, figsize = (12, 12))
sns.histplot(data=df, x = "subtotal", kde=True, ax=axes[0,0])
sns.boxplot(data=df, x = "subtotal", ax=axes[0,1])
sns.histplot(data=df, x = "min_item_price", kde=True, ax=axes[1,0])
sns.boxplot(data=df, x = "min_item_price", ax=axes[1,1])
sns.histplot(data=df, x = "max_item_price", kde=True, ax=axes[2,0])
sns.boxplot(data=df, x = "max_item_price", ax=axes[2,1])
sns.histplot(data=df, x = "total_onshift_partners", kde=True, ax=axes[3,0])
sns.boxplot(data=df, x = "total_onshift_partners", ax=axes[3,1])
sns.histplot(data=df, x = "total_busy_partners", kde=True, ax=axes[4,0])
sns.boxplot(data=df, x = "total_busy_partners", ax=axes[4,1])
sns.histplot(data=df, x = "total_outstanding_orders", kde=True, ax=axes[5,0])
sns.boxplot(data=df, x = "total_outstanding_orders", ax=axes[5,1])
sns.histplot(data=df, x = "delivery_time_mins", kde=True, ax=axes[6,0])
sns.boxplot(data=df, x = "delivery_time_mins", ax=axes[6,1])
plt.tight_layout()
plt.show()
```



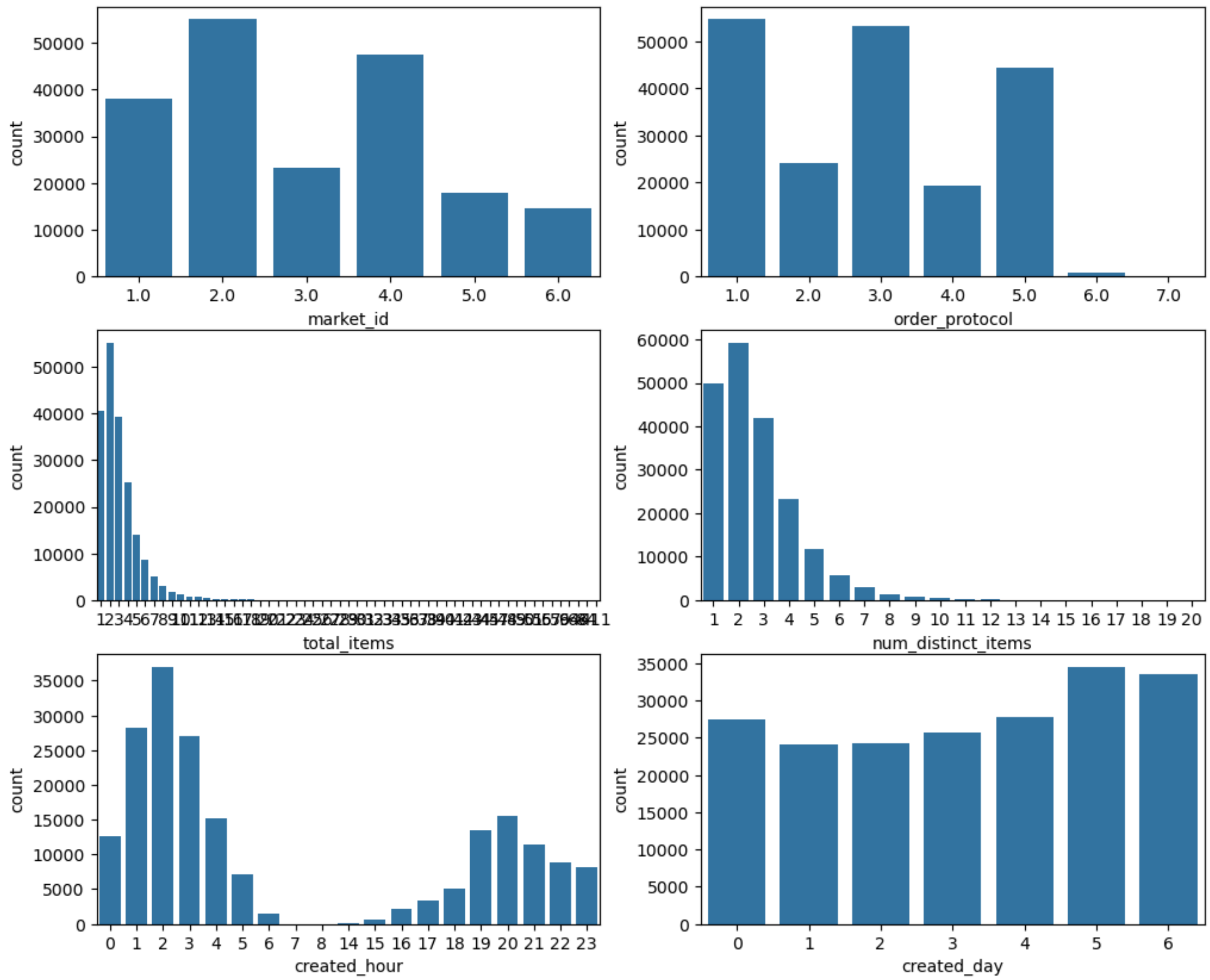
Insight

- Majority of the **subtotal** is in the range of 0 to 5000
- Majority of **min_item_price** and **max_item_price** are in the range of 0 to 2000
- **total_onshift_partners**, **total_busy_partners**, **total_outstanding_orders** and **delivery_time_mins** seem to follow similar kind of distribution - right skewed
- The boxplot clearly shows the presence of **outliers** in **subtotal**, **min_item_price**, **max_item_price** and **delivery_time_mins**

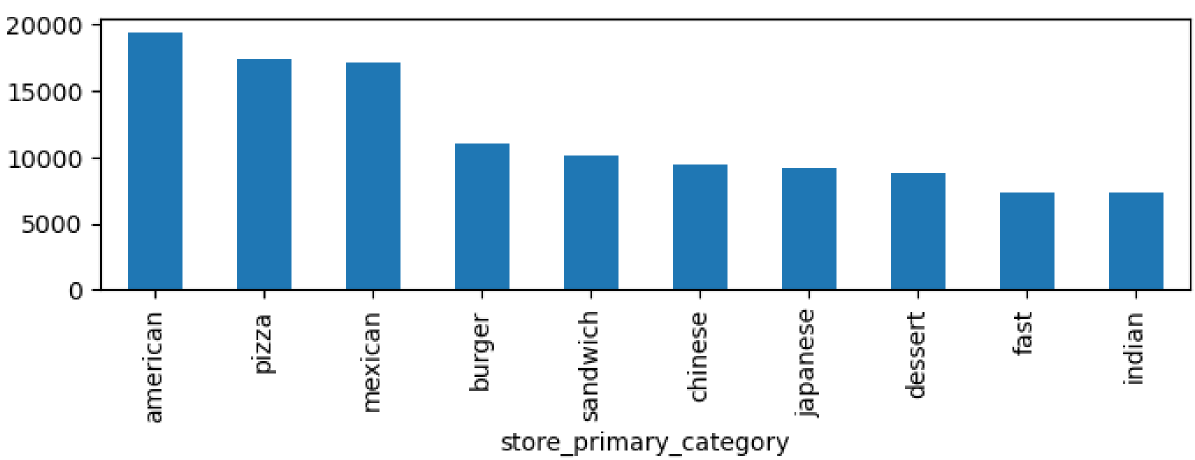
4.1.2. Categorical variables

In [10]:
```
fig, axes = plt.subplots(3,2,figsize=(12,10))
sns.countplot(ax=axes[0,0], data=df, x='market_id')
sns.countplot(ax=axes[0,1], data=df, x='order_protocol')
sns.countplot(ax=axes[1,0], data=df, x='total_items')
```

```
sns.countplot(ax=axes[1,1], data=df, x='num_distinct_items')
sns.countplot(ax=axes[2,0], data=df, x='created_hour')
sns.countplot(ax=axes[2,1], data=df, x='created_day')
plt.show()
```



In [11]:
```
df['store_primary_category'].value_counts()[:10].plot(kind='bar', figsize = (8, 2))
plt.show()
```



In [12]:
```
df['store_id'].value_counts()[:5]
```

```
Out[12]: store_id
d43ab110ab2489d6b9b2caa394bf920f    937
757b505cfd34c64c85ca5b5690ee5293    863
faacbcd5bf1d018912c116bf2783e9a1    815
cfecdb276f634854f3ef915e2e980c31    765
45c48cce2e2d7fbdea1afc51c7c6ad26    721
Name: count, dtype: int64
```

Insight

- market_id **2** is the major contributor
- Majority of order placement are through protocal **1** and **3**
- Majority of orders have **2** num_distinct_items as well as **2** total_items
- Majority of the orders are placed at around **2AM**
- Majority of the orders are placed on **weekends**
- Majority of orders delivered are from **american** restaurant
- Majority of orders delivered are from **d43ab110ab2489d6b9b2caa394bf920f** store

4.2. Missing value treatment

```
In [13]: df.isna().sum()/len(df)*100
```

```
Out[13]: market_id                   0.499929
store_id                    0.000000
store_primary_category      2.411006
order_protocol              0.503981
total_items                 0.000000
subtotal                    0.000000
num_distinct_items          0.000000
min_item_price              0.000000
max_item_price              0.000000
total_onshift_partners      8.236927
total_busy_partners         8.236927
total_outstanding_orders    8.236927
created_hour                0.000000
created_day                 0.000000
delivery_time_mins          0.003546
dtype: float64
```

Insight

- Only **0.5%** of data has missing market_id. I will **drop** all these entries
- **2.4%** of data has missing store_primary_category. I will **replace** these with 'other' category
- Only **0.5%** of data has missing order_protocol. I will **drop** all these entries
- **8.24%** of data has missing total_onshift_partners, total_busy_partners and total_outstanding_orders each. I will use **KNN imputation** to replace the missing value.
- Only **0.0035%** of data has missing delivery_timw_mins. I will **drop** these entries too

In [14]:
```
df.dropna(subset=['market_id', 'order_protocol', 'delivery_time_mins'], inplace=Tru
df.fillna({'store_primary_category':'other'}, inplace=True)
```

Remove rows which have -ve values in min_item_price, max_item_price, total_onshift_partners, total_busy_partners, total_outstanding_orders

In [15]:
```
mask = (df['min_item_price'] >= 0) & (df['max_item_price'] >= 0) & (df['total_onshi
df = df[mask]
```

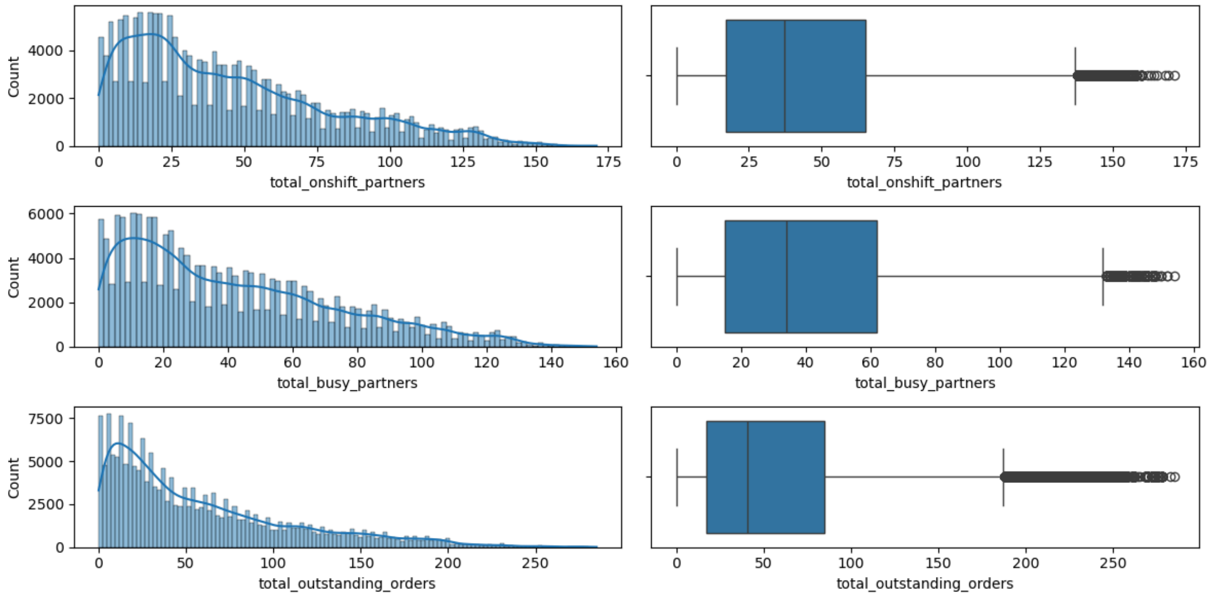
In [16]:
```
# Columns to apply KNN Imputation
columns_to_impute = ['total_onshift_partners', 'total_busy_partners', 'total_outsta
# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=2)
# Apply KNN Imputer only to specified columns
df[columns_to_impute] = imputer.fit_transform(df[columns_to_impute])
```

In [17]:
```
df.isna().sum()/len(df)*100
```

Out[17]:
```
market_id                   0.0
store_id                    0.0
store_primary_category      0.0
order_protocol              0.0
total_items                 0.0
subtotal                    0.0
num_distinct_items          0.0
min_item_price              0.0
max_item_price              0.0
total_onshift_partners      0.0
total_busy_partners         0.0
total_outstanding_orders    0.0
created_hour                0.0
created_day                 0.0
delivery_time_mins          0.0
dtype: float64
```

Let us look at the distribution again after imputation

In [18]:
```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize = (12, 6))
sns.histplot(data=df, x = "total_onshift_partners", kde=True, ax=axes[0,0])
sns.boxplot(data=df, x = "total_onshift_partners", ax=axes[0,1])
sns.histplot(data=df, x = "total_busy_partners", kde=True, ax=axes[1,0])
sns.boxplot(data=df, x = "total_busy_partners", ax=axes[1,1])
sns.histplot(data=df, x = "total_outstanding_orders", kde=True, ax=axes[2,0])
sns.boxplot(data=df, x = "total_outstanding_orders", ax=axes[2,1])
plt.tight_layout()
plt.show()
```

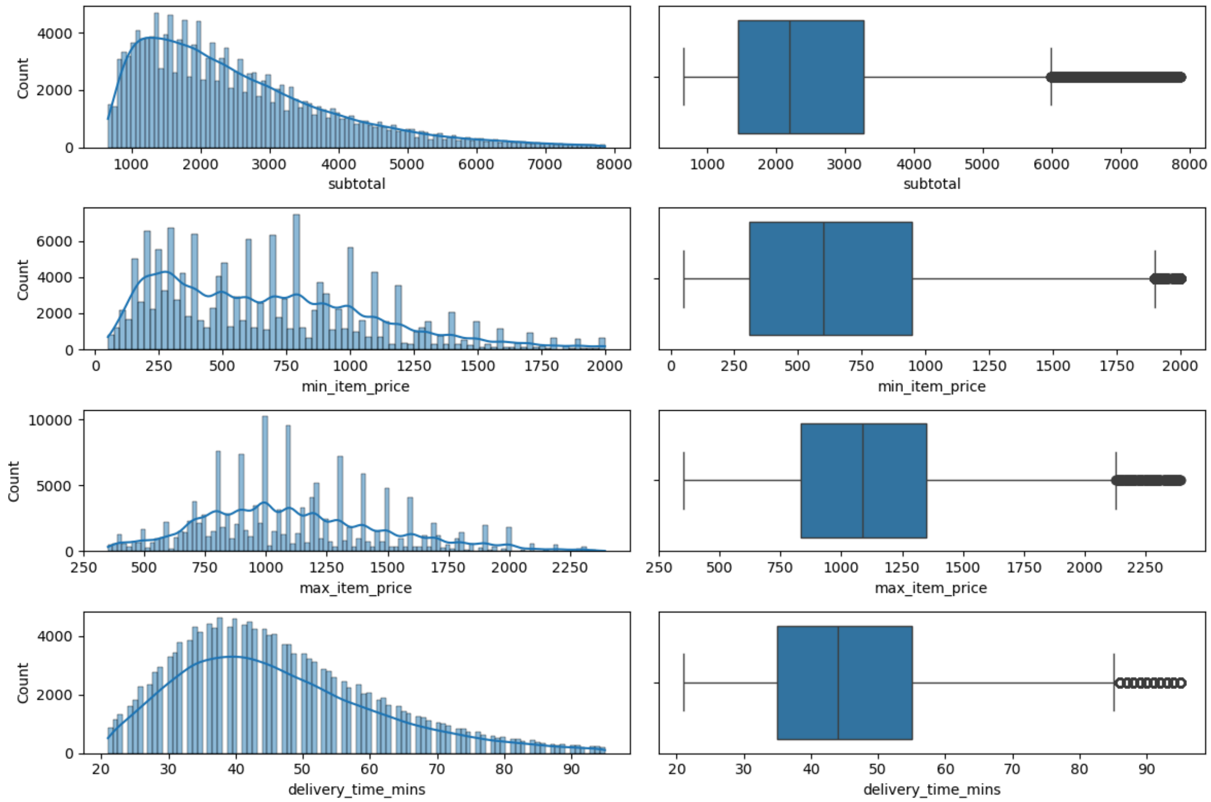


4.2. Outliers treatment

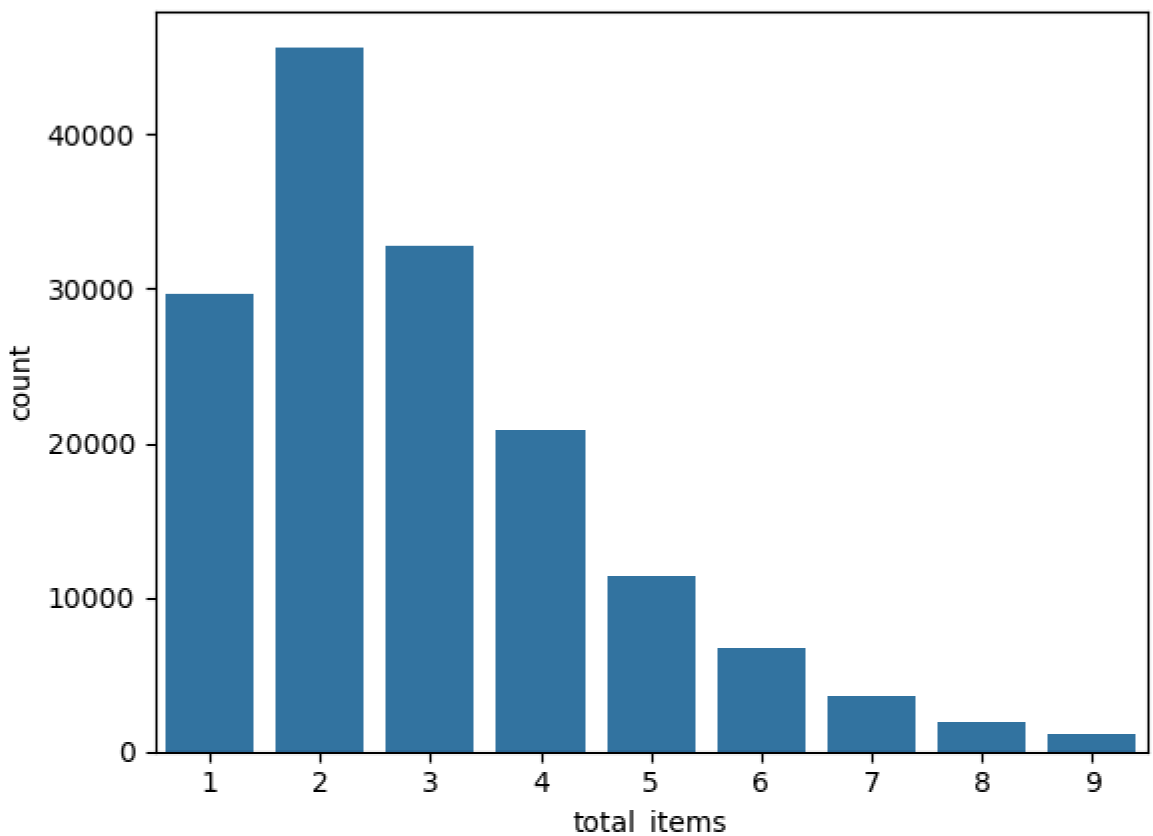
In [19]:
```
df['store_primary_category']=df['store_primary_category'].astype('category').cat.co
df['store_id']=df['store_id'].astype('category').cat.codes
```

In [26]:
```
for col in ['subtotal', 'min_item_price', 'max_item_price', 'delivery_time_mins', '
    lower_bound = df[col].quantile(0.01)
    upper_bound = df[col].quantile(0.99)
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
```

In [29]:
```
fig, axes = plt.subplots(nrows=4, ncols=2, figsize = (12, 8))
sns.histplot(data=df, x = "subtotal", kde=True, ax=axes[0,0])
sns.boxplot(data=df, x = "subtotal", ax=axes[0,1])
sns.histplot(data=df, x = "min_item_price", kde=True, ax=axes[1,0])
sns.boxplot(data=df, x = "min_item_price", ax=axes[1,1])
sns.histplot(data=df, x = "max_item_price", kde=True, ax=axes[2,0])
sns.boxplot(data=df, x = "max_item_price", ax=axes[2,1])
sns.histplot(data=df, x = "delivery_time_mins", kde=True, ax=axes[3,0])
sns.boxplot(data=df, x = "delivery_time_mins", ax=axes[3,1])
plt.tight_layout()
plt.show()
```



In [32]:
```python
sns.countplot(data=df, x='total_items')
plt.show()
```

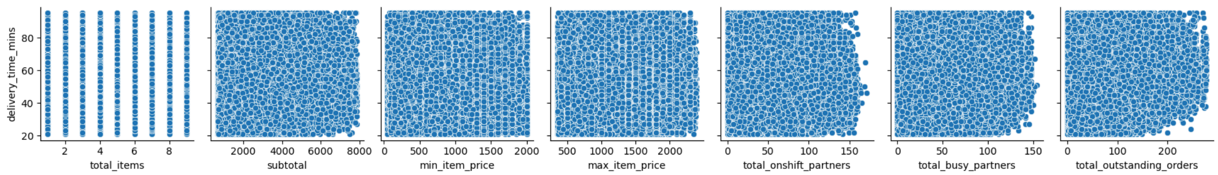


Insight

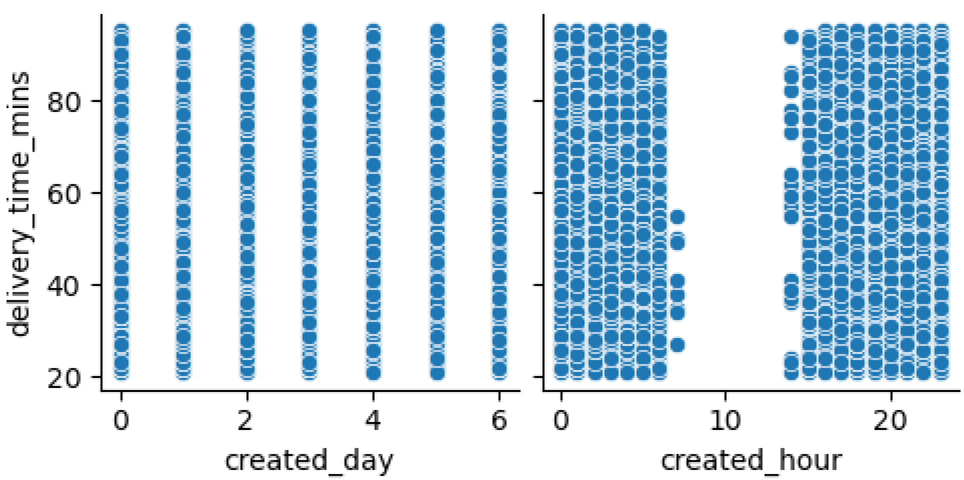
- The distribution of data looks better after removal of outliers
- It can be seen that most of the delivery is done in **40mins**

4.3. Bivariate Analysis

In [33]:
```
sns.pairplot(
    df,
    x_vars=['total_items', 'subtotal', 'min_item_price', 'max_item_price', 'total_o
    y_vars=['delivery_time_mins'],)
plt.show()
```



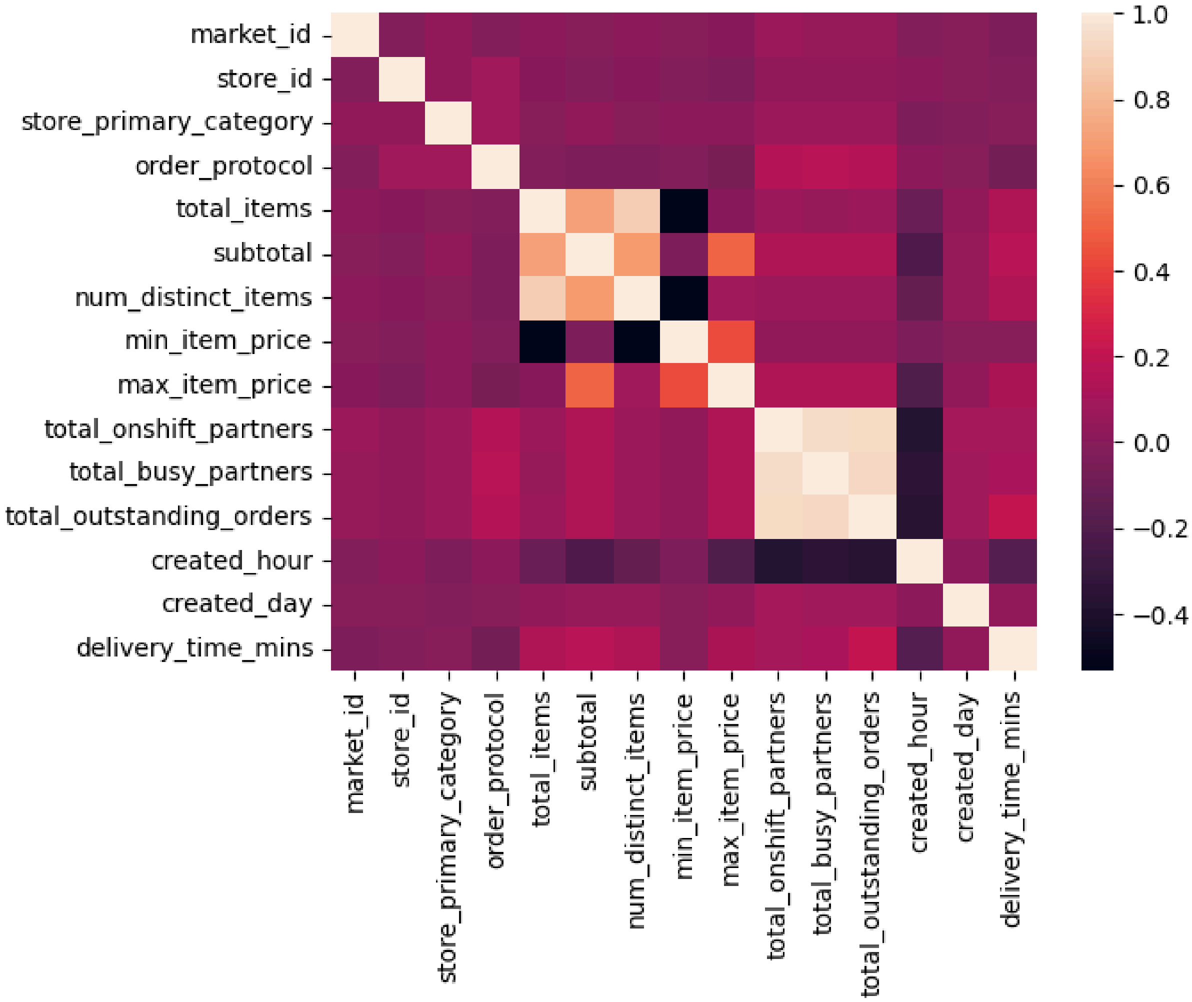
In [34]:
```
sns.pairplot(
    df,
    x_vars=['created_day', 'created_hour'],
    y_vars=['delivery_time_mins'],)
plt.show()
```



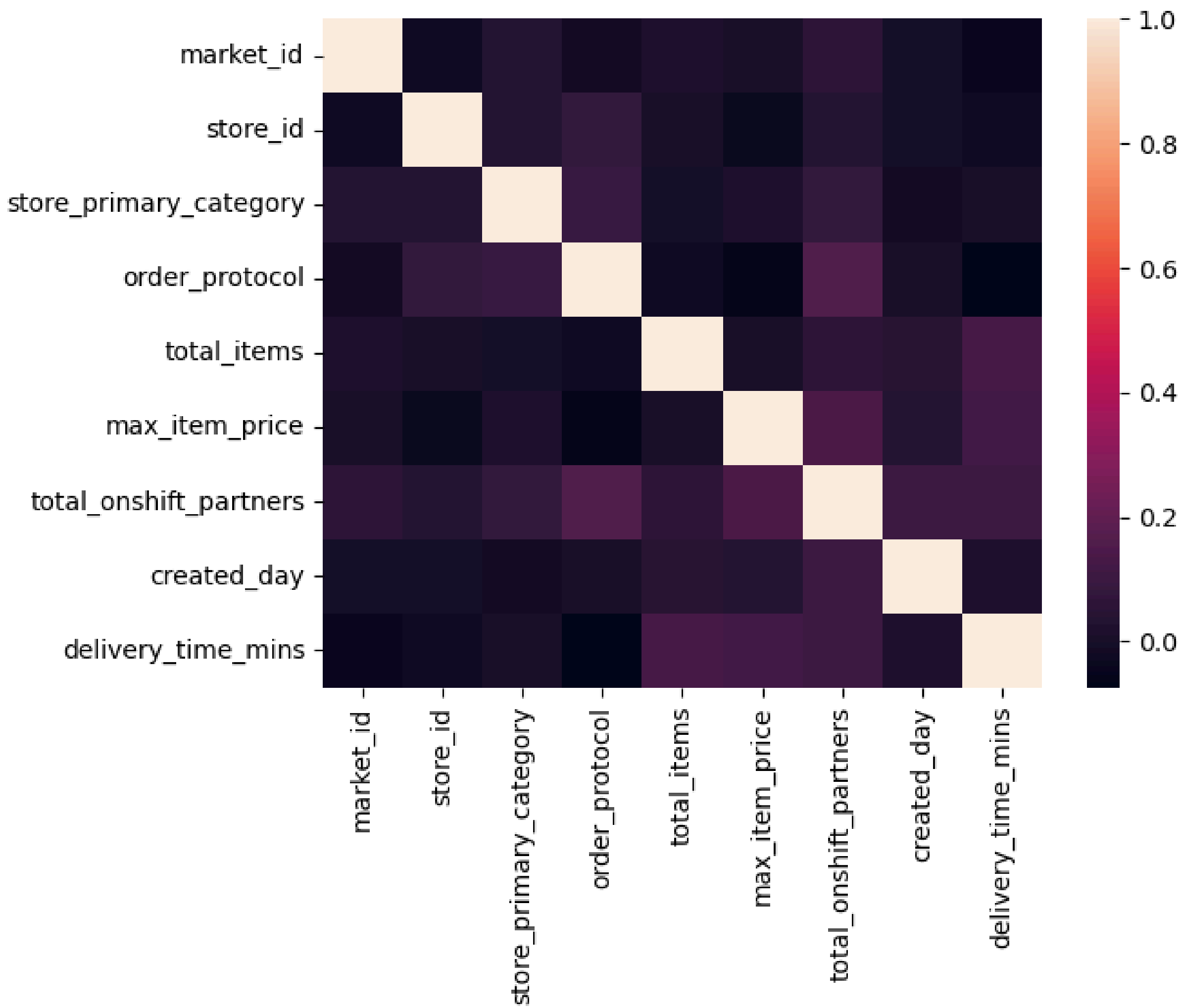
Insight

- There doesnt seem to be any relationship of delivery time with other features
- Delivery time is almost same for all days

In [35]:
```
sns.heatmap(df.corr())
plt.show()
```



In [36]:
```
sns.heatmap(df[['market_id', 'store_id', 'store_primary_category', 'order_protocol'
plt.show()
```



Insight

- Here also we see that there is no relation between delivery time and other features

5. NN Modelling

5.1. Training a NN model

I feel the store id, store primary category and order protocol will not effect delivery time, hece dropping them from feature list

In [37]:
```
df_reduced = df.drop(['store_id', 'store_primary_category', 'order_protocol'], axis
```

In [39]:
```
# Example to demonstrate categorical encoding
def encode_categorical_data(dataframe):
    # Assume some features are categorical
    categorical_columns = ['market_id']

    # Apply One-Hot Encoding
```

```
    one_hot_encoder = OneHotEncoder(sparse_output=False, drop='first')  # Drop firs
    encoded_features = pd.DataFrame(
        one_hot_encoder.fit_transform(dataframe[categorical_columns]),
        columns=one_hot_encoder.get_feature_names_out(categorical_columns)
    )

    # Drop original categorical columns and concatenate encoded columns
    dataframe = dataframe.drop(categorical_columns, axis=1)
    dataframe = pd.concat([dataframe.reset_index(drop=True), encoded_features.reset
    return dataframe
```

In [40]:
```
# Encode categorical data
df_reduced_encoded = encode_categorical_data(df_reduced)
```

In [41]:
```
df_reduced_encoded.head()
```

Out[41]:

	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_
0	4	3441	4	557	1239	
1	1	1900	1	1400	1400	
2	1	1900	1	1900	1900	
3	6	6900	5	600	1800	
4	3	3900	3	1100	1600	

In [42]:
```
# Split data into features and target
y = df_reduced_encoded['delivery_time_mins']
X = df_reduced_encoded.drop('delivery_time_mins', axis=1)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Scale features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [132…
```
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.layers import LeakyReLU
from tensorflow.keras.losses import LogCosh, Huber
from tensorflow.keras.callbacks import Callback
import tensorflow as tf
from tensorflow.keras.losses import MeanSquaredError
```

In [159…
```
# Model definition function
def create_and_train_model(X_train, y_train, X_test, y_test, epochs=50):

    # Model definition
    def create_model(input_shape):
        model = Sequential([
            Dense(32, input_shape=(input_shape,), kernel_initializer='he_normal'),
            BatchNormalization(),
```

```
        LeakyReLU(alpha=0.01),
        Dense(64),
        BatchNormalization(),
        LeakyReLU(alpha=0.01),
        Dense(128),
        BatchNormalization(),
        LeakyReLU(alpha=0.01),
        Dense(64),
        BatchNormalization(),
        LeakyReLU(alpha=0.01),
        Dense(1, activation='linear')
    ])

    model.compile(optimizer=Adam(learning_rate=0.001, global_clipnorm=1.0), los
    return model

class GradientLogger(Callback):
    def __init__(self, train_data):
        super().__init__()
        self.train_data = train_data  # Training data as (X, y)

    def on_epoch_end(self, epoch, logs=None):
        # Get the training data
        X, y = self.train_data
        with tf.GradientTape() as tape:
            # Forward pass
            y_pred = self.model(X, training=True)  # Access self.model directly
            # Compute the loss
            loss = self.model.compiled_loss(y, y_pred)

        # Compute gradients
        gradients = tape.gradient(loss, self.model.trainable_variables)

        # Log gradient statistics
        grad_norms = [tf.norm(g).numpy() for g in gradients if g is not None]
        print(f"\nEpoch {epoch + 1}: Gradient Norms:")
        print(f"  Mean: {np.mean(grad_norms):.4f}")
        print(f"  Min: {np.min(grad_norms):.4f}")
        print(f"  Max: {np.max(grad_norms):.4f}")

X_train_sample, y_train_sample = X_train[:100], y_train[:100]
gradient_logger = GradientLogger(train_data=(X_train_sample, y_train_sample))

# Create and train the model
input_shape = X_train.shape[1]
model = create_model(input_shape)
history = model.fit(X_train, y_train, epochs=epochs, verbose=1, validation_spli

# Predictions and final MSE calculation
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)

# Plot training and validation loss
plt.figure(figsize=(10, 4))
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
```

```python
    plt.xlabel('Epochs')
    plt.ylabel('Loss (MSE)')
    plt.title('Training and Validation Loss Across Epochs')
    plt.legend()
    plt.grid(True)
    plt.show()

    return mse
```

In [160…

```python
# Train model with all features
mse_all_features = create_and_train_model(X_train_scaled, y_train, X_test_scaled, y
```

```
Epoch 1/50
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\activations\leaky_relu.py:41: UserWarning: Argument `alpha` is deprecated. Use `negative_slope` instead.
  warnings.warn(
205/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 2249.5393 - mae: 45.0351
Epoch 1: Gradient Norms:
  Mean: 44.9948
  Min: 0.0000
  Max: 345.6995
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 2238.4680 - mae: 44.9097 - val_loss: 1803.5947 - val_mae: 39.8345
Epoch 2/50
 26/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1605.2399 - mae: 37.4779
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\backend\tensorflow\trainer.py:664: UserWarning: `model.compiled_loss()` is deprecated. Instead, use `model.compute_loss(x, y, y_pred, sample_weight, training)`.
  warnings.warn(
```

```
203/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1440.5004 - mae: 35.1261
Epoch 2: Gradient Norms:
  Mean: 56.2747
  Min: 0.0000
  Max: 322.0479
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1426.9658 - mae: 34.9186 - val_los
s: 871.9685 - val_mae: 26.0319
Epoch 3/50
215/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 613.4295 - mae: 20.5820
Epoch 3: Gradient Norms:
  Mean: 54.2441
  Min: 0.0000
  Max: 196.7765
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 612.0215 - mae: 20.5487 - val_loss:
250.4935 - val_mae: 11.8398
Epoch 4/50
212/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 200.9866 - mae: 10.7713
Epoch 4: Gradient Norms:
  Mean: 49.0397
  Min: 0.0000
  Max: 246.7629
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 200.6886 - mae: 10.7671 - val_loss:
190.4771 - val_mae: 10.5099
Epoch 5/50
214/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 178.1042 - mae: 10.4803
Epoch 5: Gradient Norms:
  Mean: 43.3286
  Min: 0.0000
  Max: 220.3572
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 178.0918 - mae: 10.4800 - val_loss:
183.3919 - val_mae: 10.3654
Epoch 6/50
213/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 174.6668 - mae: 10.3751
Epoch 6: Gradient Norms:
  Mean: 42.3577
  Min: 0.0000
  Max: 232.2325
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 174.6826 - mae: 10.3756 - val_loss:
181.3274 - val_mae: 10.3360
Epoch 7/50
216/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 174.4046 - mae: 10.3652
Epoch 7: Gradient Norms:
  Mean: 41.7513
  Min: 0.0000
  Max: 227.0512
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 174.4054 - mae: 10.3653 - val_loss:
179.9899 - val_mae: 10.3920
Epoch 8/50
211/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 173.5091 - mae: 10.3272
Epoch 8: Gradient Norms:
  Mean: 39.6308
  Min: 0.0000
  Max: 195.9327
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 173.5175 - mae: 10.3279 - val_loss:
178.9618 - val_mae: 10.3840
Epoch 9/50
```

```
213/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 172.7879 - mae: 10.3252
Epoch 9: Gradient Norms:
  Mean: 37.7135
  Min: 0.0000
  Max: 194.6149
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 172.7924 - mae: 10.3252 - val_loss:
177.3396 - val_mae: 10.3103
Epoch 10/50
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 172.7053 - mae: 10.3070
Epoch 10: Gradient Norms:
  Mean: 36.7855
  Min: 0.0000
  Max: 201.8718
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 172.6942 - mae: 10.3072 - val_loss:
176.8672 - val_mae: 10.3415
Epoch 11/50
212/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.8870 - mae: 10.1976
Epoch 11: Gradient Norms:
  Mean: 38.6563
  Min: 0.0000
  Max: 217.1819
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.9298 - mae: 10.1994 - val_loss:
175.9690 - val_mae: 10.3145
Epoch 12/50
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 170.0631 - mae: 10.2443
Epoch 12: Gradient Norms:
  Mean: 38.4658
  Min: 0.0000
  Max: 211.4359
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 170.1384 - mae: 10.2463 - val_loss:
175.5651 - val_mae: 10.3350
Epoch 13/50
212/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 171.0560 - mae: 10.2680
Epoch 13: Gradient Norms:
  Mean: 36.0993
  Min: 0.0000
  Max: 200.1110
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 171.0535 - mae: 10.2677 - val_loss:
176.2726 - val_mae: 10.4024
Epoch 14/50
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.7702 - mae: 10.2290
Epoch 14: Gradient Norms:
  Mean: 38.5769
  Min: 0.0000
  Max: 215.0470
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.8285 - mae: 10.2305 - val_loss:
176.0461 - val_mae: 10.2768
Epoch 15/50
212/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.8562 - mae: 10.2204
Epoch 15: Gradient Norms:
  Mean: 37.8294
  Min: 0.0000
  Max: 226.3993
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.8674 - mae: 10.2209 - val_loss:
175.1677 - val_mae: 10.3324
Epoch 16/50
```

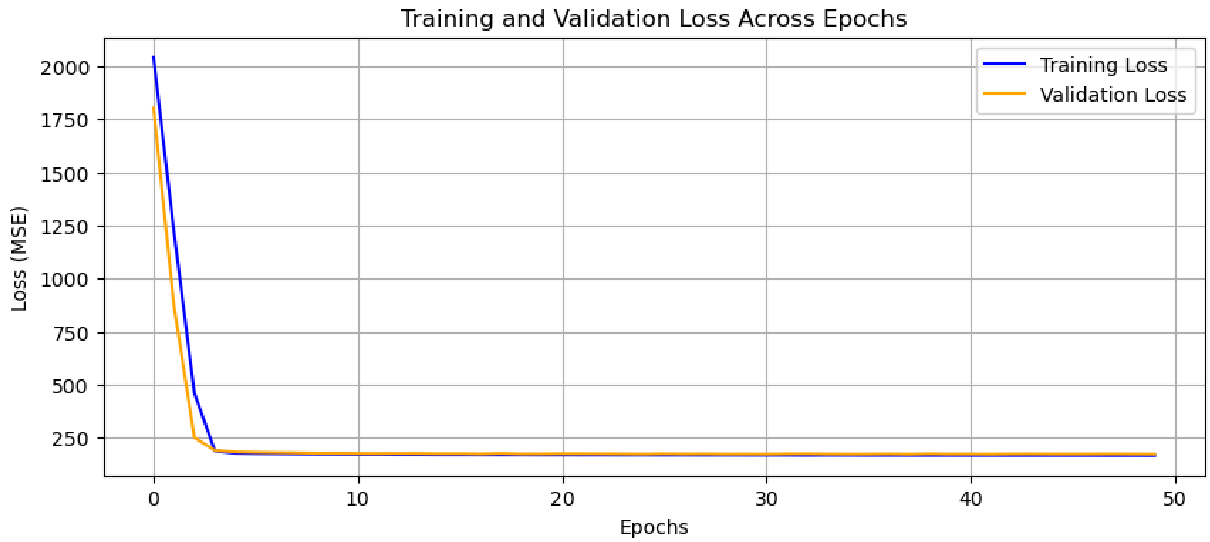
```
209/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.8015 - mae: 10.2303
Epoch 16: Gradient Norms:
  Mean: 35.2032
  Min: 0.0000
  Max: 197.5538
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.8246 - mae: 10.2307 - val_loss:
175.3601 - val_mae: 10.3170
Epoch 17/50
210/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 168.2390 - mae: 10.1704
Epoch 17: Gradient Norms:
  Mean: 35.2649
  Min: 0.0000
  Max: 199.6725
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.2965 - mae: 10.1725 - val_loss:
173.9087 - val_mae: 10.3053
Epoch 18/50
211/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 169.8642 - mae: 10.2347
Epoch 18: Gradient Norms:
  Mean: 37.6104
  Min: 0.0000
  Max: 215.0655
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.8629 - mae: 10.2344 - val_loss:
176.5241 - val_mae: 10.4087
Epoch 19/50
207/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 170.5861 - mae: 10.2477
Epoch 19: Gradient Norms:
  Mean: 36.8726
  Min: 0.0000
  Max: 219.9571
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 170.5537 - mae: 10.2464 - val_loss:
173.9826 - val_mae: 10.2936
Epoch 20/50
204/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.0736 - mae: 10.2095
Epoch 20: Gradient Norms:
  Mean: 36.7550
  Min: 0.0000
  Max: 212.5412
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.0885 - mae: 10.2091 - val_loss:
174.1143 - val_mae: 10.3169
Epoch 21/50
201/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 169.0007 - mae: 10.2054
Epoch 21: Gradient Norms:
  Mean: 42.3597
  Min: 0.0000
  Max: 248.8876
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.0352 - mae: 10.2057 - val_loss:
175.1497 - val_mae: 10.4070
Epoch 22/50
209/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 170.0843 - mae: 10.2112
Epoch 22: Gradient Norms:
  Mean: 37.9089
  Min: 0.0000
  Max: 232.0540
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 170.0564 - mae: 10.2109 - val_loss:
174.8790 - val_mae: 10.3625
Epoch 23/50
```

```
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 168.7414 - mae: 10.1781
Epoch 23: Gradient Norms:
  Mean: 37.8884
  Min: 0.0000
  Max: 223.5155
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.7738 - mae: 10.1795 - val_loss:
174.5758 - val_mae: 10.3420
Epoch 24/50
213/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 170.0041 - mae: 10.2206
Epoch 24: Gradient Norms:
  Mean: 35.8177
  Min: 0.0000
  Max: 204.6021
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.9845 - mae: 10.2201 - val_loss:
173.8655 - val_mae: 10.3372
Epoch 25/50
209/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.3510 - mae: 10.1334
Epoch 25: Gradient Norms:
  Mean: 36.8118
  Min: 0.0000
  Max: 213.9263
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.4093 - mae: 10.1354 - val_loss:
173.1388 - val_mae: 10.2854
Epoch 26/50
214/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.1989 - mae: 10.1519
Epoch 26: Gradient Norms:
  Mean: 38.1846
  Min: 0.0000
  Max: 226.2578
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.2194 - mae: 10.1523 - val_loss:
174.7395 - val_mae: 10.3924
Epoch 27/50
213/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 168.4286 - mae: 10.1805
Epoch 27: Gradient Norms:
  Mean: 39.3817
  Min: 0.0000
  Max: 229.7686
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.4355 - mae: 10.1807 - val_loss:
173.5632 - val_mae: 10.2985
Epoch 28/50
213/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 169.3706 - mae: 10.1954
Epoch 28: Gradient Norms:
  Mean: 34.7444
  Min: 0.0000
  Max: 193.5687
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 169.3543 - mae: 10.1949 - val_loss:
174.1017 - val_mae: 10.2638
Epoch 29/50
215/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 168.0120 - mae: 10.1593
Epoch 29: Gradient Norms:
  Mean: 35.0953
  Min: 0.0000
  Max: 203.3445
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.0181 - mae: 10.1596 - val_loss:
173.0428 - val_mae: 10.3166
Epoch 30/50
```

```
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 168.3662 - mae: 10.1597
Epoch 30: Gradient Norms:
  Mean: 32.5813
  Min: 0.0000
  Max: 188.6980
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.3691 - mae: 10.1603 - val_loss:
173.2359 - val_mae: 10.2668
Epoch 31/50
211/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 168.0315 - mae: 10.1484
Epoch 31: Gradient Norms:
  Mean: 36.5598
  Min: 0.0000
  Max: 203.8364
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 168.0353 - mae: 10.1487 - val_loss:
172.8335 - val_mae: 10.2614
Epoch 32/50
208/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.6252 - mae: 10.1483
Epoch 32: Gradient Norms:
  Mean: 37.2033
  Min: 0.0000
  Max: 215.8843
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.6480 - mae: 10.1491 - val_loss:
174.2439 - val_mae: 10.2597
Epoch 33/50
207/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.5430 - mae: 10.1612
Epoch 33: Gradient Norms:
  Mean: 36.2382
  Min: 0.0000
  Max: 200.4900
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.5444 - mae: 10.1605 - val_loss:
174.8259 - val_mae: 10.3340
Epoch 34/50
211/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.3783 - mae: 10.1375
Epoch 34: Gradient Norms:
  Mean: 35.8597
  Min: 0.0000
  Max: 198.7228
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.3972 - mae: 10.1382 - val_loss:
173.4129 - val_mae: 10.3532
Epoch 35/50
208/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 167.4493 - mae: 10.1517
Epoch 35: Gradient Norms:
  Mean: 33.2744
  Min: 0.0000
  Max: 198.6350
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.4669 - mae: 10.1519 - val_loss:
172.8176 - val_mae: 10.3464
Epoch 36/50
204/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.0669 - mae: 10.1092
Epoch 36: Gradient Norms:
  Mean: 35.4396
  Min: 0.0000
  Max: 210.2507
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.1475 - mae: 10.1111 - val_loss:
173.1565 - val_mae: 10.3404
Epoch 37/50
```

```
214/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.8537 - mae: 10.1367
Epoch 37: Gradient Norms:
  Mean: 35.3704
  Min: 0.0000
  Max: 202.6881
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.8489 - mae: 10.1368 - val_loss:
174.0972 - val_mae: 10.1805
Epoch 38/50
207/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 165.7655 - mae: 10.0874
Epoch 38: Gradient Norms:
  Mean: 34.7899
  Min: 0.0000
  Max: 203.2329
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 165.8356 - mae: 10.0894 - val_loss:
172.6210 - val_mae: 10.2326
Epoch 39/50
216/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.5712 - mae: 10.1160
Epoch 39: Gradient Norms:
  Mean: 33.4628
  Min: 0.0000
  Max: 193.4768
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.5741 - mae: 10.1161 - val_loss:
174.6069 - val_mae: 10.2312
Epoch 40/50
215/216 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 167.2635 - mae: 10.1377
Epoch 40: Gradient Norms:
  Mean: 36.1214
  Min: 0.0000
  Max: 199.0746
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 167.2632 - mae: 10.1377 - val_loss:
173.2289 - val_mae: 10.2637
Epoch 41/50
214/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 165.9865 - mae: 10.1016
Epoch 41: Gradient Norms:
  Mean: 36.1178
  Min: 0.0000
  Max: 187.8597
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 165.9996 - mae: 10.1020 - val_loss:
173.3917 - val_mae: 10.2460
Epoch 42/50
208/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.4256 - mae: 10.1377
Epoch 42: Gradient Norms:
  Mean: 34.2584
  Min: 0.0000
  Max: 183.8495
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.4053 - mae: 10.1371 - val_loss:
172.6636 - val_mae: 10.2706
Epoch 43/50
208/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.2100 - mae: 10.0899
Epoch 43: Gradient Norms:
  Mean: 38.2909
  Min: 0.0000
  Max: 221.6710
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.2339 - mae: 10.0910 - val_loss:
173.8541 - val_mae: 10.3175
Epoch 44/50
```

```
215/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.2875 - mae: 10.0918
Epoch 44: Gradient Norms:
  Mean: 38.1552
  Min: 0.0000
  Max: 209.5925
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.2926 - mae: 10.0920 - val_loss:
174.0124 - val_mae: 10.3583
Epoch 45/50
216/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 167.4233 - mae: 10.1576
Epoch 45: Gradient Norms:
  Mean: 35.2712
  Min: 0.0000
  Max: 206.3295
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 167.4204 - mae: 10.1574 - val_loss:
173.3062 - val_mae: 10.3218
Epoch 46/50
207/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 164.5789 - mae: 10.0513
Epoch 46: Gradient Norms:
  Mean: 31.8549
  Min: 0.0000
  Max: 164.0808
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 164.6734 - mae: 10.0542 - val_loss:
173.2081 - val_mae: 10.3170
Epoch 47/50
209/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.5354 - mae: 10.1125
Epoch 47: Gradient Norms:
  Mean: 35.1648
  Min: 0.0000
  Max: 205.3537
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.5397 - mae: 10.1125 - val_loss:
173.4931 - val_mae: 10.3266
Epoch 48/50
206/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.5779 - mae: 10.1161
Epoch 48: Gradient Norms:
  Mean: 32.0254
  Min: 0.0000
  Max: 173.7895
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.5529 - mae: 10.1153 - val_loss:
173.8577 - val_mae: 10.3471
Epoch 49/50
212/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 164.1559 - mae: 10.0640
Epoch 49: Gradient Norms:
  Mean: 36.6529
  Min: 0.0000
  Max: 195.7887
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 164.2043 - mae: 10.0651 - val_loss:
173.3613 - val_mae: 10.2497
Epoch 50/50
211/216 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 166.1796 - mae: 10.0880
Epoch 50: Gradient Norms:
  Mean: 32.6656
  Min: 0.0000
  Max: 177.2888
216/216 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 166.1765 - mae: 10.0882 - val_loss:
172.7631 - val_mae: 10.2411
960/960 ━━━━━━━━━━━━━━━━━━━━ 1s 983us/step
```



In [161…
```
mse_all_features
```

Out[161… 168.559646042214

- Increasing the number of layers, number of neurons or changing the optmizers did not give better performance than the above model