# Porter: Neural Networks Regression

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.

## Problem statement

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.

In [52]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [28]:
```
import pandas as pd
import numpy as np
```

In [29]:
```
df = pd.read_csv("dataset.csv")
df
```

Out[29]:

| | market_id | created_at | actual_delivery_time | store_id | store_prir |
|---|---|---|---|---|---|
| **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 | |
| **...** | ... | ... | ... | ... | |
| **197423** | 1.0 | 2015-02-17 00:19:41 | 2015-02-17 01:24:48 | a914ecef9c12ffdb9bede64bb703d877 | |
| **197424** | 1.0 | 2015-02-13 00:01:59 | 2015-02-13 00:58:22 | a914ecef9c12ffdb9bede64bb703d877 | |
| **197425** | 1.0 | 2015-01-24 04:46:08 | 2015-01-24 05:36:16 | a914ecef9c12ffdb9bede64bb703d877 | |
| **197426** | 1.0 | 2015-02-01 18:18:15 | 2015-02-01 19:23:22 | c81e155d85dae5430a8cee6f2242e82c | |
| **197427** | 1.0 | 2015-02-08 19:24:33 | 2015-02-08 20:01:41 | c81e155d85dae5430a8cee6f2242e82c | |

197428 rows × 14 columns

In [30]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 196441 non-null  float64
 1   created_at                197428 non-null  object 
 2   actual_delivery_time      197421 non-null  object 
 3   store_id                  197428 non-null  object 
 4   store_primary_category    192668 non-null  object 
 5   order_protocol            196433 non-null  float64
 6   total_items               197428 non-null  int64  
 7   subtotal                  197428 non-null  int64  
 8   num_distinct_items        197428 non-null  int64  
 9   min_item_price            197428 non-null  int64  
 10  max_item_price            197428 non-null  int64  
 11  total_onshift_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

In [31]:

```
#Converting the date-time to datatype
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
df['created_at'] = pd.to_datetime(df['created_at'])
```

In [32]:

```
#Delivery time in minutes
df['delivery_time'] = (df['actual_delivery_time']-df['created_at']).dt.total_seconc
df['hour']=df['created_at'].dt.hour
df['day']=df['created_at'].dt.dayofweek

df.market_id = df.market_id.astype('category')
df.order_protocol = df.order_protocol.astype('category')
```

In [33]:

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

Out[33]:

```
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
delivery_time                   7
hour                            0
day                             0
dtype: int64
```

**Deleting all the rows with null values.**

In [34]:

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

Out[34]:

```
(176248, 17)
```

In [35]:
```
#Converting float columns to int datatype
float_cols = ['total_onshift_partners','total_busy_partners','total_outstanding_orc
df[float_cols] = df[float_cols].astype('int')
```

**Deleting columns which are not required**

In [36]:
```
df.drop(columns=['store_id','created_at','actual_delivery_time'], inplace=True)
```

In [37]:
```
df.describe()
```

Out[37]:

| | market_id | order_protocol | total_items | subtotal | num_distinct_items | min_item_ |
|---|---|---|---|---|---|---|
| **count** | 176248.000000 | 176248.000000 | 176248.000000 | 176248.000000 | 176248.000000 | 176248.0( |
| **mean** | 2.743747 | 2.911687 | 3.204592 | 2696.498939 | 2.674589 | 684.9: |
| **std** | 1.330911 | 1.512920 | 2.673899 | 1828.922584 | 1.625558 | 519.9 |
| **min** | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | -86.0( |
| **25%** | 2.000000 | 1.000000 | 2.000000 | 1408.000000 | 1.000000 | 299.0( |
| **50%** | 2.000000 | 3.000000 | 3.000000 | 2221.000000 | 2.000000 | 595.0( |
| **75%** | 4.000000 | 4.000000 | 4.000000 | 3407.000000 | 3.000000 | 942.0( |
| **max** | 6.000000 | 7.000000 | 411.000000 | 26800.000000 | 20.000000 | 14700.0( |

In [39]:
```
#removing negative values
df = df[~((df[['total_onshift_partners','total_busy_partners','total_outstanding_or
```

In [41]:
```
df.describe()
```

Out[41]:

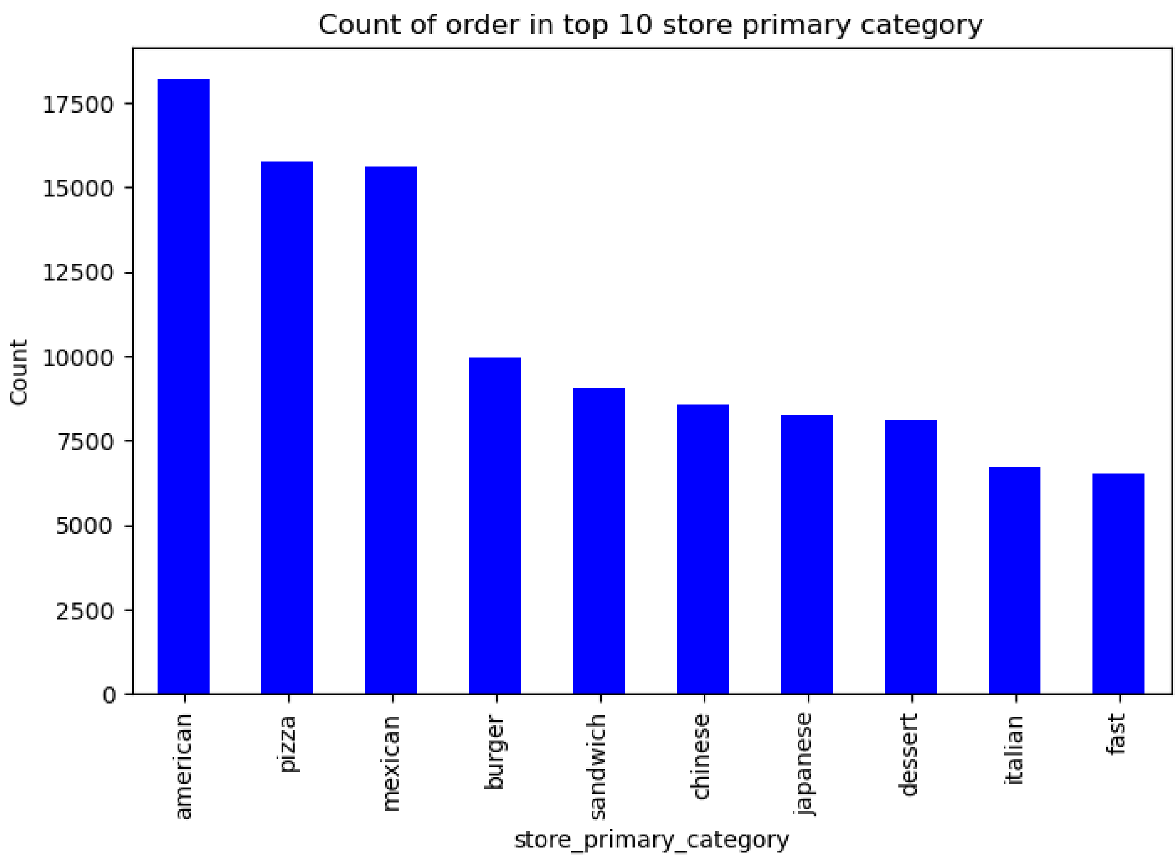
| | market_id | order_protocol | total_items | subtotal | num_distinct_items | min_item_ |
|---|---|---|---|---|---|---|
| **count** | 176157.000000 | 176157.000000 | 176157.000000 | 176157.000000 | 176157.000000 | 176157.0( |
| **mean** | 2.744109 | 2.911789 | 3.204079 | 2696.578484 | 2.674580 | 684.9! |
| **std** | 1.330883 | 1.512871 | 2.671920 | 1829.049610 | 1.625497 | 519.8: |
| **min** | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.0( |
| **25%** | 2.000000 | 1.000000 | 2.000000 | 1408.000000 | 1.000000 | 299.0( |
| **50%** | 2.000000 | 3.000000 | 3.000000 | 2221.000000 | 2.000000 | 595.0( |
| **75%** | 4.000000 | 4.000000 | 4.000000 | 3408.000000 | 3.000000 | 942.0( |
| **max** | 6.000000 | 7.000000 | 411.000000 | 26800.000000 | 20.000000 | 14700.0( |

In [40]:
```
df.isna().sum()
```

Out[40]:
```
market_id                   0
store_primary_category      0
order_protocol              0
total_items                 0
subtotal                    0
num_distinct_items          0
min_item_price              0
max_item_price              0
total_onshift_partners      0
total_busy_partners         0
total_outstanding_orders    0
delivery_time               0
hour                        0
day                         0
dtype: int64
```

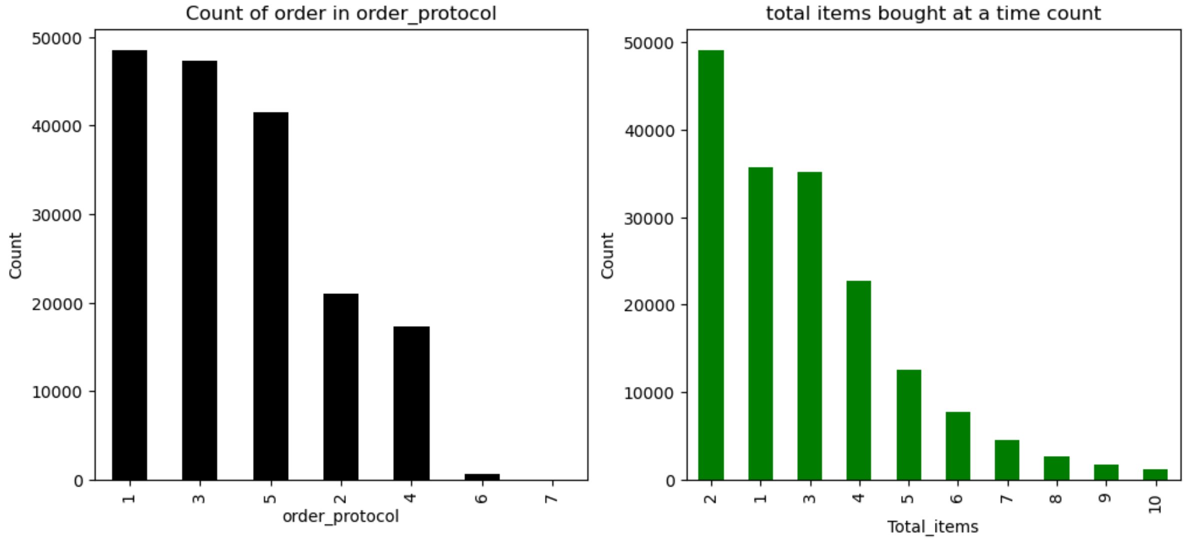
# Univariate Bivariate and Multivariate Analysis

In [43]:
```python
fig =plt.figure(figsize=(8,5))

df['store_primary_category'].value_counts().head(10).plot(kind='bar',color='blue')
plt.xlabel('store_primary_category')
plt.ylabel('Count')
plt.title('Count of order in top 10 store primary category')
plt.show()
```



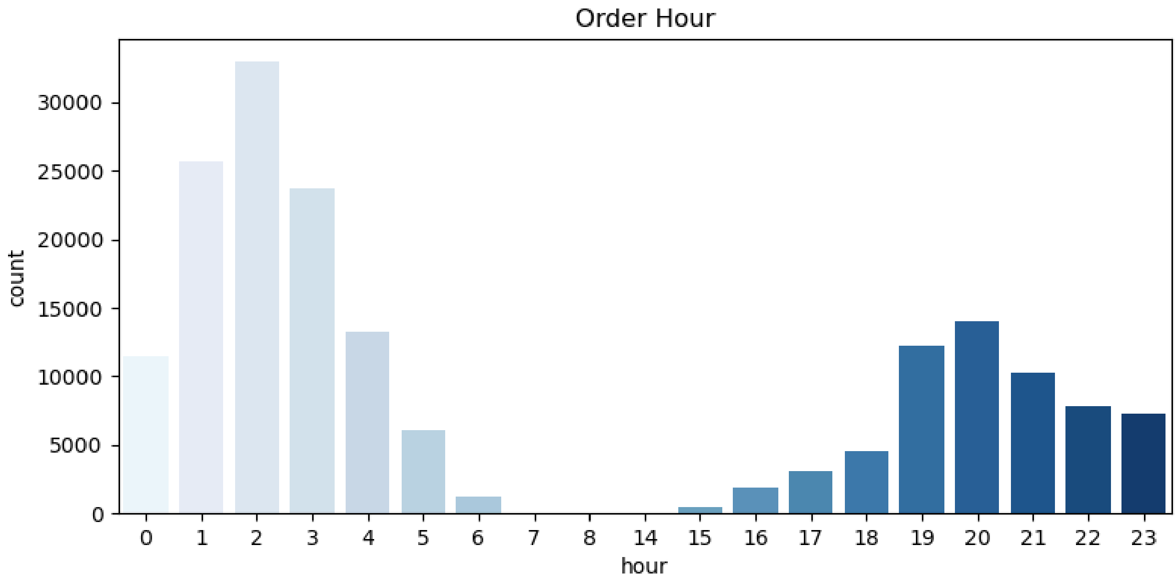

In [46]:
```python
fig =plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
df['order_protocol'].value_counts().plot(kind='bar',color='black')
plt.xlabel('order_protocol')
plt.ylabel('Count')
plt.title('Count of order in order_protocol')

plt.subplot(1,2,2)
df['total_items'].value_counts().head(10).plot(kind='bar',color='green')
```

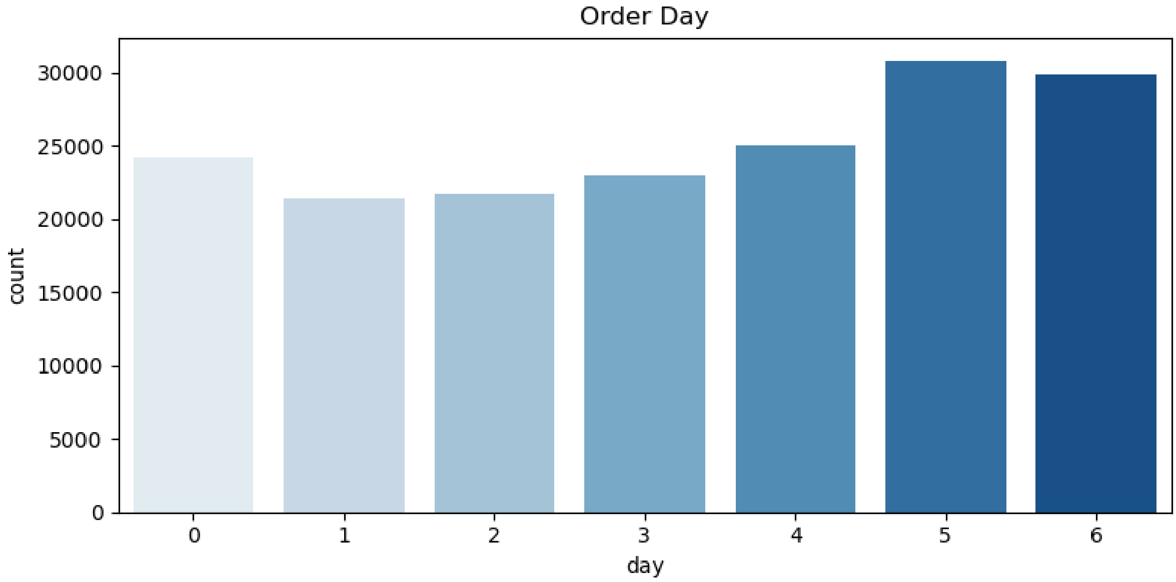
```
plt.xlabel('Total_items')
plt.ylabel('Count')
plt.title('total items bought at a time count')
plt.show()
```




In [49]:

```
# Distribution of Categorical Features

fig, ax = plt.subplots(2,1, figsize=(8,8))

sns.countplot(data=df, x='hour', palette='Blues', ax=ax[0])
ax[0].set_title('Order Hour')

sns.countplot(data=df, x='day', palette='Blues', ax=ax[1])
ax[1].set_title('Order Day')

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

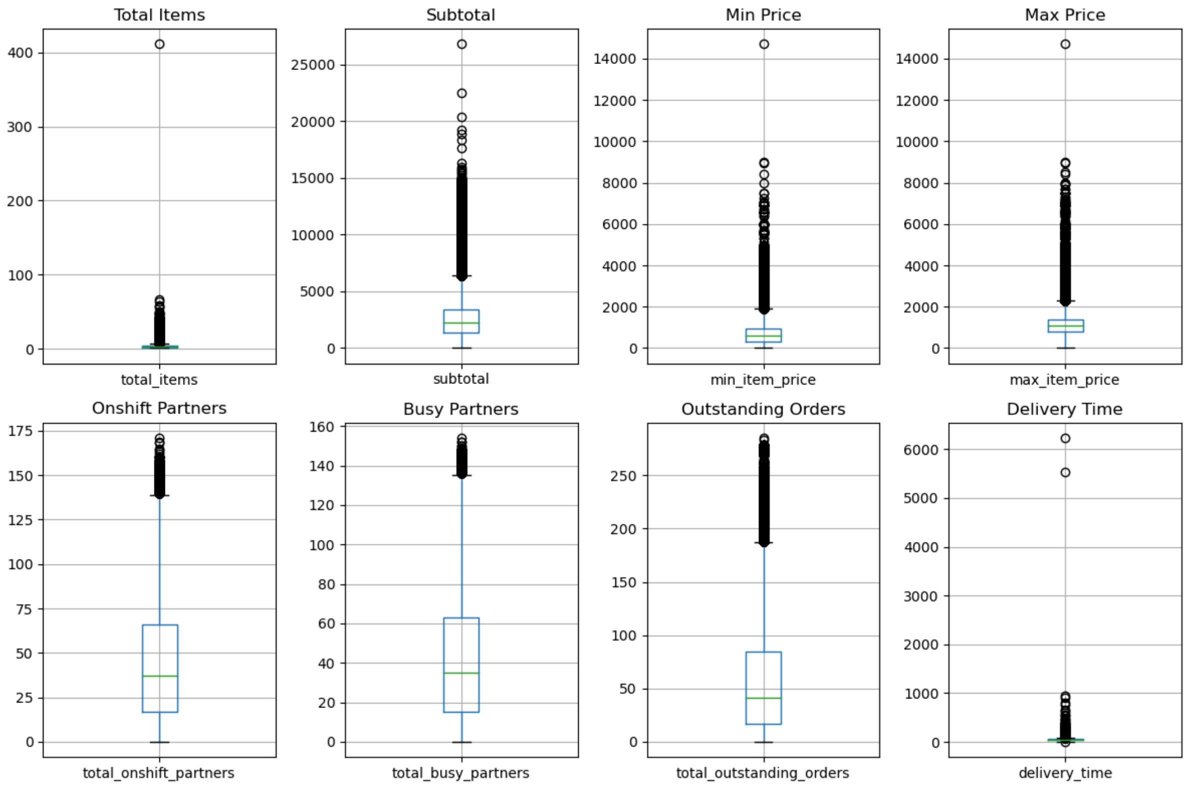



In [50]:

```
fig, ax = plt.subplots(2,4, figsize=(12,8))

df.boxplot(column='total_items', ax=ax[0,0])
ax[0,0].set_title('Total Items')

df.boxplot(column='subtotal', ax=ax[0,1])
ax[0,1].set_title('Subtotal')

df.boxplot(column='min_item_price', ax=ax[0,2])
ax[0,2].set_title('Min Price')

df.boxplot(column='max_item_price', ax=ax[0,3])
ax[0,3].set_title('Max Price')

df.boxplot(column='total_onshift_partners', ax=ax[1,0])
ax[1,0].set_title('Onshift Partners')

df.boxplot(column='total_busy_partners', ax=ax[1,1])
ax[1,1].set_title('Busy Partners')

df.boxplot(column='total_outstanding_orders', ax=ax[1,2])
ax[1,2].set_title('Outstanding Orders')

df.boxplot(column='delivery_time', ax=ax[1,3])
ax[1,3].set_title('Delivery Time')
```

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

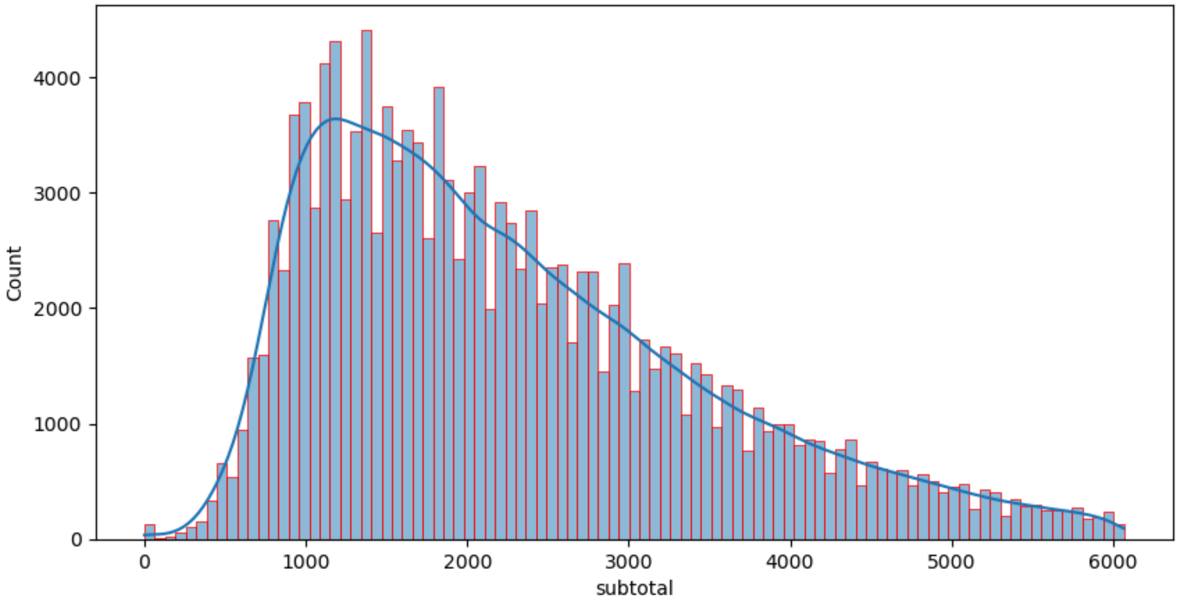



## Outlier Treatment

In [53]:
```
# Remove outliers through IQR method
outlier_cols = ['total_items', 'subtotal', 'min_item_price',
                'max_item_price', 'delivery_time', 'total_onshift_partners',
                'total_busy_partners', 'total_outstanding_orders']

for col in outlier_cols:
  # Calculate Q1 and Q3
  q1 = np.percentile(df[col], 25)
  q3 = np.percentile(df[col], 75)

  # Calculate IQR
  iqr = q3 - q1

  # Define lower and upper bounds
  lower_bound = q1 - (1.5 * iqr)
  upper_bound = q3 + (1.5 * iqr)

  # Remove outliers
  df = df.loc[~((df[col] < lower_bound) | (df[col] > upper_bound))]

df.shape
```

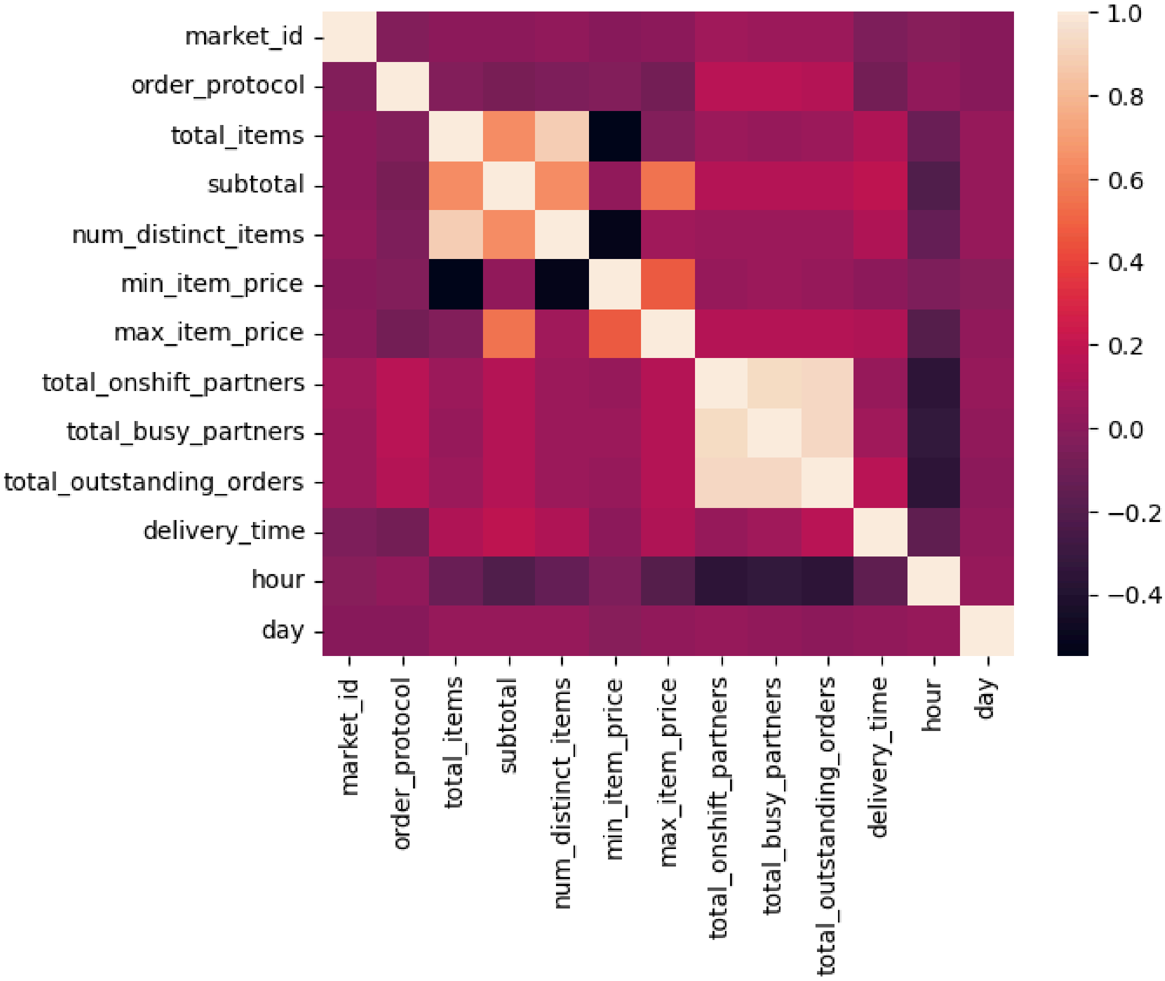
Out[53]:
```
(142433, 14)
```

In [55]:
```
plt.figure(figsize=(10,5))
sns.histplot(x='subtotal',data=df,kde=True, edgecolor='red')
plt.show()
```




In [56]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df.corr())
```

Out[56]: <Axes: >




1. Most orders were for restaurants in the American, Pizza, Mexican, and Burger categories, in that order.

2. The majority of orders followed protocol 1, with protocols 3 and 5 following. Protocol 7 saw very few orders.

3. Most orders were for 2 items at a time, followed by 1, 3, 4, and so on.

4. No orders were placed between 9 and 13 hours, and very few orders were placed at 6, 7, 8, 14, 15, and 16 hours. It appears that the hours between 7 and 16 don't see many orders.

5. Weekends had a higher order count.

6. There was almost no difference between the count of orders based on the time of delivery and the time of order creation, suggesting that active hours are between 0-5 and 17-23.

7. No clear trend was observed based on the hour of the day or day of the week, aside from some off-peak times.

8. No correlation could be found between the total time taken to deliver an item and other numerical features.

9. There is a strong correlation between the following feature pairs: (total_onshift_partners, total_busy_partners, total_outstanding_orders), (hour_created, hour_delivered), and (day_created, day_delivered).

## Data Preparation

In [57]:
```
X = df.drop(columns=['delivery_time'])
y = df['delivery_time']

print(X.shape, y.shape)
```

```
(142433, 13) (142433,)
```

In [61]:
```
from sklearn.preprocessing import OneHotEncoder

cat_cols = ['market_id', 'order_protocol', 'hour', 'day']
enc = OneHotEncoder(handle_unknown='ignore')
X_encoded = pd.DataFrame(enc.fit_transform(X[cat_cols]).toarray(), columns=enc.get_
X_num = X.drop(cat_cols, axis=1)
X = pd.concat([X_num, X_encoded], axis=1)
```

In [63]:
```
#split the data for training, validation and test

from sklearn.model_selection import train_test_split

X_train_val, X_test, y_train_val, y_test = train_test_split(X,y,test_size=0.1, ranc
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_si

print('Train: ', X_train.shape, y_train.shape)
print('Val: ', X_val.shape, y_val.shape)
print('Test: ', X_test.shape, y_test.shape)
```

```
Train:  (115370, 47) (115370,)
Val:  (12819, 47) (12819,)
Test:  (14244, 47) (14244,)
```

### Encoding

In [64]:
```
from category_encoders import TargetEncoder

encoder = TargetEncoder(cols=['store_primary_category'])

X_train = encoder.fit_transform(X_train, y_train)
X_val = encoder.transform(X_val)
X_test = encoder.transform(X_test)
```

In [65]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train_scaled, columns=X_train.columns)

X_val_scaled = scaler.transform(X_val)
X_val=pd.DataFrame(X_val_scaled, columns=X_val.columns)

X_test_scaled = scaler.transform(X_test)
X_test=pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

# Neural Network

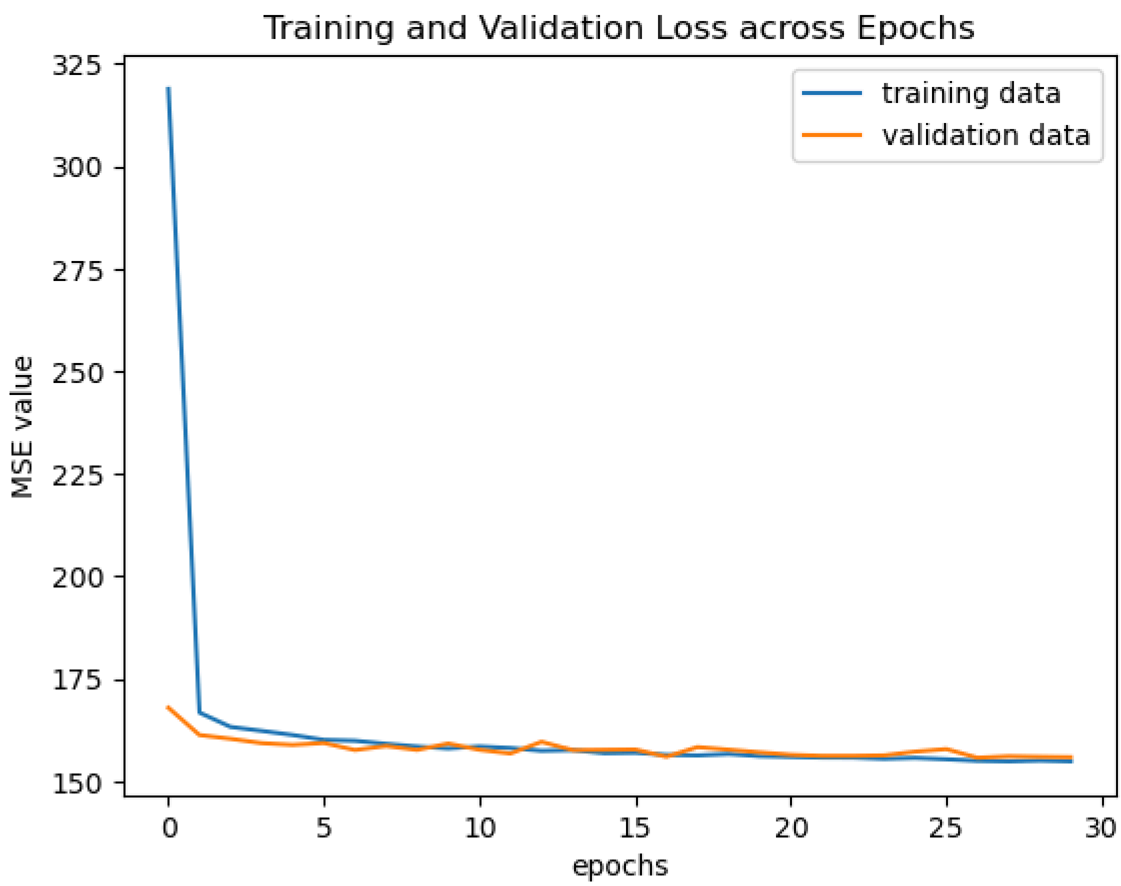
In [69]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Model definition
model = Sequential()
model.add(Dense(64, kernel_initializer='normal', activation='relu', input_shape=(X_
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='linear'))

# Model compilation
model.compile(loss='mse', optimizer=Adam(), metrics=['mae'])

# Model training
history = model.fit(X_train, y_train, epochs=30, batch_size=512, verbose=1, validat
```

```
Epoch 1/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - loss: 658.9433 - mae: 19.4586 - val_lo
ss: 167.9703 - val_mae: 10.2440
Epoch 2/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 168.4960 - mae: 10.2641 - val_lo
ss: 161.3014 - val_mae: 10.1014
Epoch 3/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 163.6679 - mae: 10.1344 - val_lo
ss: 160.3808 - val_mae: 10.0292
Epoch 4/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 161.0559 - mae: 10.0285 - val_lo
ss: 159.3210 - val_mae: 9.9441
Epoch 5/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 160.6059 - mae: 10.0424 - val_lo
ss: 158.8936 - val_mae: 9.8771
Epoch 6/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 159.1536 - mae: 9.9747 - val_los
s: 159.3620 - val_mae: 9.8863
Epoch 7/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 160.3427 - mae: 9.9939 - val_los
s: 157.6386 - val_mae: 9.9454
Epoch 8/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 160.7506 - mae: 10.0244 - val_lo
ss: 158.6249 - val_mae: 10.1010
Epoch 9/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 158.5397 - mae: 9.9619 - val_los
s: 157.6664 - val_mae: 9.8620
Epoch 10/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 157.7243 - mae: 9.9233 - val_los
s: 159.2165 - val_mae: 9.8683
Epoch 11/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - loss: 158.1557 - mae: 9.9198 - val_los
s: 157.7001 - val_mae: 9.9731
Epoch 12/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 159.4295 - mae: 9.9711 - val_los
s: 156.7899 - val_mae: 9.8632
Epoch 13/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 156.5410 - mae: 9.8854 - val_los
s: 159.6482 - val_mae: 9.8230
Epoch 14/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 156.9104 - mae: 9.8990 - val_los
s: 157.5926 - val_mae: 9.8704
Epoch 15/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 157.3654 - mae: 9.8923 - val_los
s: 157.6852 - val_mae: 9.8252
Epoch 16/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.3760 - mae: 9.8367 - val_los
s: 157.7569 - val_mae: 9.8277
Epoch 17/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 156.4272 - mae: 9.8699 - val_los
s: 156.0047 - val_mae: 9.9212
Epoch 18/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 156.7515 - mae: 9.8912 - val_los
s: 158.3524 - val_mae: 9.7995
Epoch 19/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 155.1786 - mae: 9.8318 - val_los
s: 157.7239 - val_mae: 9.7945
Epoch 20/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 156.1209 - mae: 9.8580 - val_los
s: 157.0641 - val_mae: 9.8153
Epoch 21/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 156.5166 - mae: 9.8723 - val_los
s: 156.4756 - val_mae: 9.9386
Epoch 22/30
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.9741 - mae: 9.8615 - val_los
s: 156.2014 - val_mae: 9.8319
Epoch 23/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.3581 - mae: 9.8586 - val_los
s: 156.1990 - val_mae: 9.8898
Epoch 24/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.4746 - mae: 9.8463 - val_los
s: 156.3251 - val_mae: 9.8419
Epoch 25/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.5638 - mae: 9.8596 - val_los
s: 157.2416 - val_mae: 10.0491
Epoch 26/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.8516 - mae: 9.8623 - val_los
s: 157.8698 - val_mae: 9.8325
Epoch 27/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 154.6075 - mae: 9.7978 - val_los
s: 155.7472 - val_mae: 9.8931
Epoch 28/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 154.1745 - mae: 9.8131 - val_los
s: 156.1086 - val_mae: 9.9201
Epoch 29/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 155.7748 - mae: 9.8496 - val_los
s: 155.9931 - val_mae: 9.8565
Epoch 30/30
226/226 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 154.5389 - mae: 9.8037 - val_los
s: 155.8954 - val_mae: 9.9424
```

In [70]:

```python
plt.plot(history.history['loss'], label='training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Training and Validation Loss across Epochs')
plt.ylabel('MSE value')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show()
```



In [74]:
```
#Evaluating the Model

val_loss, val_mae = model.evaluate(X_val, y_val, verbose=1)
print(f'Validation Loss: {val_loss}, Validation MAE: {val_mae}')
```

```
401/401 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 156.0142 - mae: 9.9293
Validation Loss: 155.89535522460938, Validation MAE: 9.94243335723877
```

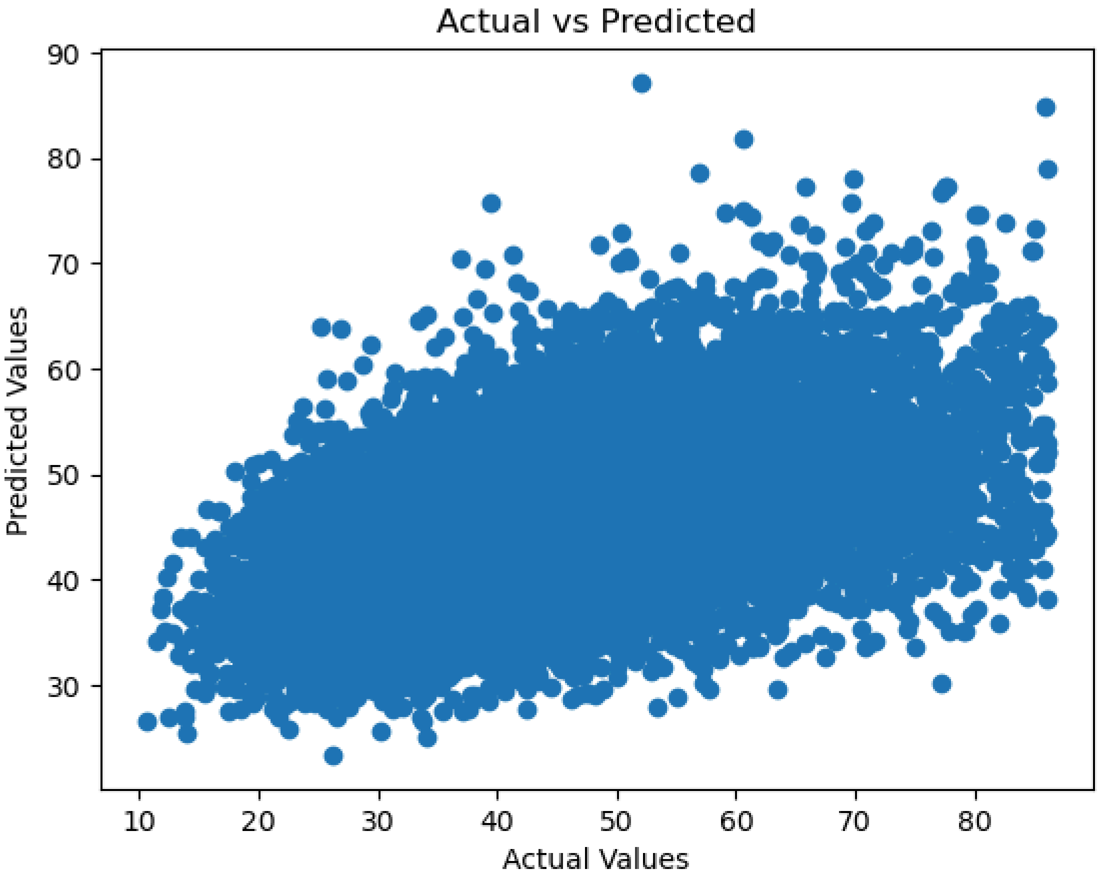
- the model is not overfitting and is performing consistently on both the training and validation datasets.
- the model generalizes well, with similar prediction errors on unseen data

In [76]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

y_pred = model.predict(X_val)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_val, y_pred)
print(f'Mean Squared Error (MSE): {mse}')

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_val, y_pred)
print(f'Mean Absolute Error (MAE): {mae}')
```

```
401/401 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Mean Squared Error (MSE): 155.89536218754068
Root Mean Squared Error (RMSE): 12.48580642920355
Mean Absolute Error (MAE): 9.942436836844385
```

In [75]:
```
#Predicting and Analyzing Results

plt.scatter(y_val, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted')
plt.show()
```

```
401/401 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```




**Recommendations:**

1. **Incentives for Night Shifts**: Given the high delivery demand between midnight and 5 AM and the limited number of delivery personnel available at night, Porter should offer incentives and bonuses for night shifts. This can enhance customer satisfaction by meeting the substantial late-night demand.

1. **Weekend Staffing**: Since weekends see larger order volumes, Porter should consider hiring employees specifically for weekend shifts. This approach allows scaling up operations without the need for permanent staff.

2. **Strategic Placement of Delivery Personnel**: To improve delivery times and customer satisfaction, Porter should position delivery personnel near popular restaurants, such as those serving American cuisine and pizza, which receive the majority of orders.

3. **Promote Underutilized Portals**: Portal 1 is receiving most of the orders, while Portal 7 remains underutilized. Porter should increase awareness and encourage customers to use Portal 7 to balance order distribution across platforms.

# Leading Questions:

#### 1. Defining the problem statements and where can this and modifications of this be used?

**Problem Statement:** Porter aims to develop a regression model to estimate delivery times for food orders based on various factors such as the type of order, the restaurant, and the delivery partner. This involves predicting how long it will take for an order to be delivered from the restaurant to the customer.

Use Cases and Modifications:

- Customer Communication: Accurate delivery time predictions can improve customer satisfaction by setting realistic expectations.
- Operational Efficiency: Helps optimize delivery schedules and resource allocation, potentially reducing costs and enhancing efficiency.
- Dynamic Pricing: Delivery times can be factored into pricing models, adjusting costs based on expected delivery duration and urgency.

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

1. **pd.to_datetime():** Converts a scalar, array-like, or Series into a pandas datetime object.
2. **pd.DatetimeIndex():** Creates a DatetimeIndex object from an array-like or iterable object of date-time strings.
3. **pd.date_range():** Generates a fixed frequency date range, which is useful for creating sequences of dates.

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

**Datetime:** Represents a specific point in time, including both date and time components. In pandas, it is used for time-series data, enabling easy manipulation and analysis of temporal data.

**Timedelta:** Represents a duration or difference between two datetime values. It is used to perform operations like adding or subtracting time intervals from datetime objects.

**Time Span (Period):** Represents a specific length of time or a range between two dates. In pandas, the Period class can be used to represent and work with periods of time, such as months or years, allowing for easy manipulation of time periods.

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

- Due to outliers, scaling techniques such as standardisation or MinMaxScaler are impacted and result in wrong representation of the data for training
- Outliers can significantly impact descriptive statistics of the data like mean, standard deviation and correlation. This can cause the data to be misrepresented
- Outliers disproportionately influence the weights and biases while model training leading to overfitting. Outliers in training data make it difficult for the model to generalise well on test data.

**5. Name 3 outlier removal methods?**

Z-Score Method, IQR (Interquartile Range) Method and standard deviation method.

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

For estimating delivery times, classical machine learning methods like **Linear Regression** provide a straightforward approach, while **Decision Trees** and **Random Forests** handle complex, non-linear relationships effectively. **Gradient Boosting Machines** (GBM) and

**Support Vector Regression** (SVR) offer advanced techniques for capturing intricate patterns and improving prediction accuracy.

**7. Why is scaling required for neural networks?**

Scaling is required for neural networks to ensure that input features are on a similar scale, which helps in achieving faster convergence during training. It also prevents issues with gradient descent where features with larger magnitudes can dominate, leading to inefficient learning or instability in the model's training process.

**8. Briefly explain your choice of optimizer.**

Adam (Adaptive Moment Estimation): Combines the advantages of two other optimizers, Momentum and RMSprop. It adjusts the learning rate for each parameter individually based on estimates of first and second moments of the gradients, which helps in converging faster and handling sparse gradients effectively. It is widely used due to its adaptive learning rate and robustness to hyperparameter settings.

**9. Which activation function did you use and why?**

**ReLU (Rectified Linear Unit)** was used as the activation function because it introduces non-linearity into the model while being computationally efficient. ReLU helps in mitigating the vanishing gradient problem by allowing gradients to flow through the network more effectively during training, which accelerates convergence and improves performance.

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

A neural network performs well on a large dataset because:

1. **Better Generalization**: Large datasets provide diverse examples, helping the model learn more generalized patterns and reduce overfitting, leading to improved performance on unseen data.
2. **Complexity Handling**: Neural networks have numerous parameters and layers, and large datasets help in training these complex models effectively by providing sufficient data to capture intricate relationships and patterns.