Riphah International University

Assignment 1

Submitted By: Sokaina Kanwal shah, Shafiq Rasool

SAP ID: 25582,25583

Submitted To: Sir Aleem Raheem

Semester: 6A BSCS

Subject: AI(LAB)

Errors:

The error was about scaler word, so before using it in data scaling I declare it above.

76]:
```
#Split the data into training and testing sets
X = df.drop("Maximum_price", axis=1)  # Input features
y = df["Maximum_price"]  # Target variable

# Step 7: Data preprocessing for target variable
y = label_encoder.transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 8: Data scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 9: Train the linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Step 10: Make predictions on the testing set
y_pred = model.predict(X_test_scaled)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12468\2069513073.py in <module>
      4
      5 # Step 7: Data preprocessing for target variable
----> 6 y = label_encoder.transform(y)
      7
      8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

~\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py in transform(self, y)
    136             return np.array([])
    137
--> 138         return _encode(y, uniques=self.classes_)
    139
    140     def inverse_transform(self, y):

~\anaconda3\lib\site-packages\sklearn\utils\_encode.py in _encode(values, uniques, check_unknown)
    185     else:
    186         if check_unknown:
--> 187             diff = _check_unknown(values, uniques)
```

There is an exception ,

```
# Convert categorical variables to numerical representation for new data
new_data_encoded = new_data.copy()
if 'Loyalty_customer' in label_encoder.classes_:
    new_data_encoded["Loyalty_customer"] = label_encoder.transform(new_data["Loyalty_customer"])
if 'Product_Category' in label_encoder.classes_:
    new_data_encoded["Product_Category"] = label_encoder.transform(new_data["Product_Category"])

# Scale numerical variables in new data
numerical_columns = ['Stall_no', 'Market_Category', 'Grade', 'Demand', 'Discount_avail', 'charges_1', 'charges_2 (%)', 'Minir
new_data_scaled = scaler.transform(new_data_encoded[numerical_columns])

predicted_prices = model.predict(new_data_scaled)
print("Predicted prices:", predicted_prices)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The feature names should match those that were
passed during fit. Starting version 1.2, an error will be raised.
Feature names seen at fit time, yet now missing:
- Loyalty_customer
- Product_Category

  warnings.warn(message, FutureWarning)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12468\905507186.py in <module>
      8 # Scale numerical variables in new data
      9 numerical_columns = ['Stall_no', 'Market_Category', 'Grade', 'Demand', 'Discount_avail', 'charges_1', 'charges_2
(%)', 'Minimum_price']
---> 10 new_data_scaled = scaler.transform(new_data_encoded[numerical_columns])
     11
     12 predicted_prices = model.predict(new_data_scaled)

~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in transform(self, X, copy)
    971
    972         copy = copy if copy is not None else self.copy
--> 973         X = self._validate_data(
    974             X,
    975             reset=False,

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    583
    584         if not no_val_X and check_params.get("ensure_2d", True):
--> 585             self._check_n_features(X, reset=reset)
    586
    587         return out
```

```
# Convert categorical variables to numerical representation for new data
new_data["Loyalty_customer"] = label_encoder.transform(new_data["Loyalty_customer"])
new_data["Product_Category"] = label_encoder.transform(new_data["Product_Category"])

# Handle unseen labels with error handling
try:
    new_data_scaled = scaler.transform(new_data)
except ValueError as e:
    print("Unseen labels in the new data. Handle them accordingly.")
    # Handle the unseen labels in the new data (e.g., replace them, discard the rows, etc.)

predicted_prices = model.predict(new_data_scaled)
print("Predicted prices:", predicted_prices)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\sklearn\utils\_encode.py in _encode(values, uniques, check_unknown)
    181         try:
--> 182             return _map_to_integer(values, uniques)
    183         except KeyError as e:

~\anaconda3\lib\site-packages\sklearn\utils\_encode.py in _map_to_integer(values, uniques)
    125     table = _nandict({val: i for i, val in enumerate(uniques)})
--> 126     return np.array([table[v] for v in values])
    127

~\anaconda3\lib\site-packages\sklearn\utils\_encode.py in <listcomp>(.0)
    125     table = _nandict({val: i for i, val in enumerate(uniques)})
--> 126     return np.array([table[v] for v in values])
    127

~\anaconda3\lib\site-packages\sklearn\utils\_encode.py in __missing__(self, key)
    119             return self.nan_value
--> 120         raise KeyError(key)
    121

KeyError: 'Yes'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12468\2605202119.py in <module>
      1 # Convert categorical variables to numerical representation for new data
----> 2 new_data["Loyalty_customer"] = label_encoder.transform(new_data["Loyalty_customer"])
      3 new_data["Product_Category"] = label_encoder.transform(new_data["Product_Category"])
      4
```

```
# Convert categorical variables to numerical representation for new data
new_data_encoded = new_data.copy()
if 'Loyalty_customer' in label_encoder.classes_:
    new_data_encoded["Loyalty_customer"] = label_encoder.transform(new_data["Loyalty_customer"])
if 'Product_Category' in label_encoder.classes_:
    new_data_encoded["Product_Category"] = label_encoder.transform(new_data["Product_Category"])

new_data_scaled = scaler.transform(new_data_encoded)
predicted_prices = model.predict(new_data_scaled)
print("Predicted prices:", predicted_prices)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12468\2992698804.py in <module>
      6     new_data_encoded["Product_Category"] = label_encoder.transform(new_data["Product_Category"])
      7
----> 8 new_data_scaled = scaler.transform(new_data_encoded)
      9 predicted_prices = model.predict(new_data_scaled)
     10 print("Predicted prices:", predicted_prices)

NameError: name 'scaler' is not defined
```

Here again same exception happens but we tried different method but still it’s not solving.

Solution:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np
```

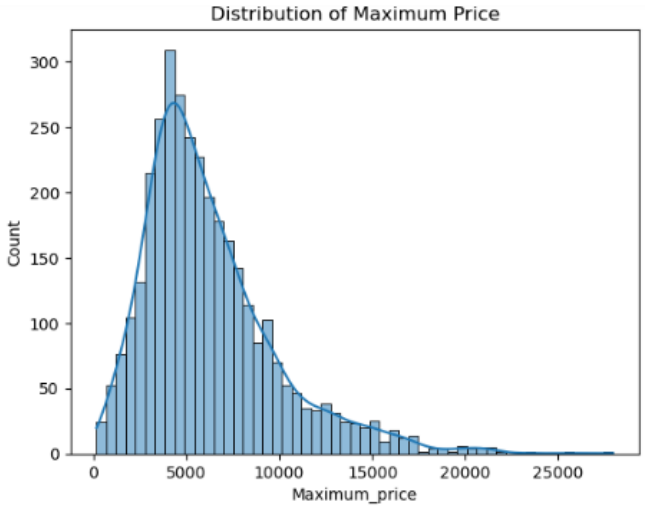
In [2]:
```
# Step 1: Load the dataset
df = pd.read_csv("test.csv")
```

In [3]:
```
# Step 2: Data preprocessing
# Drop irrelevant columns
df = df.drop(["Product_id", "Customer_name", "instock_date"], axis=1)
```

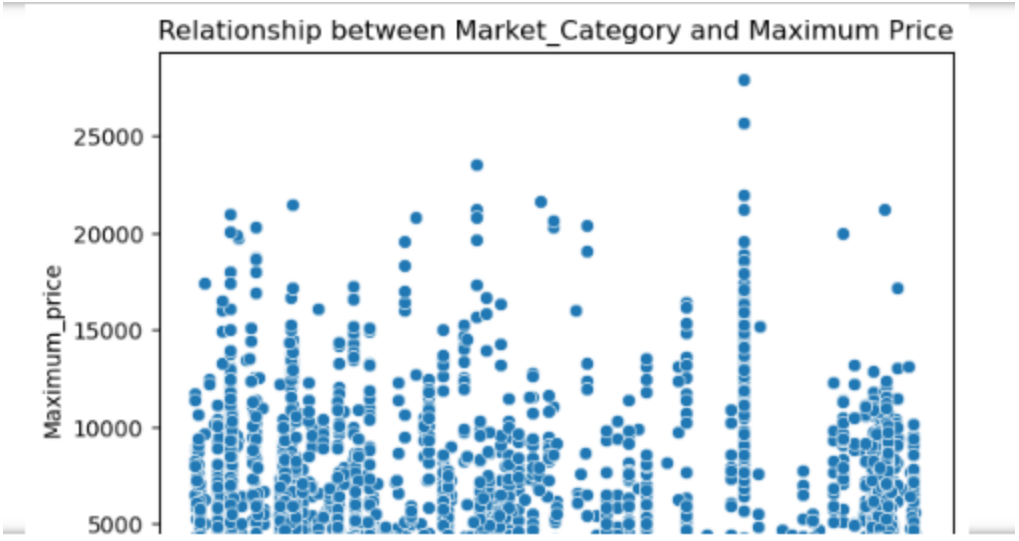
In [4]:
```
# Handle missing values (if any)
df = df.dropna()
```

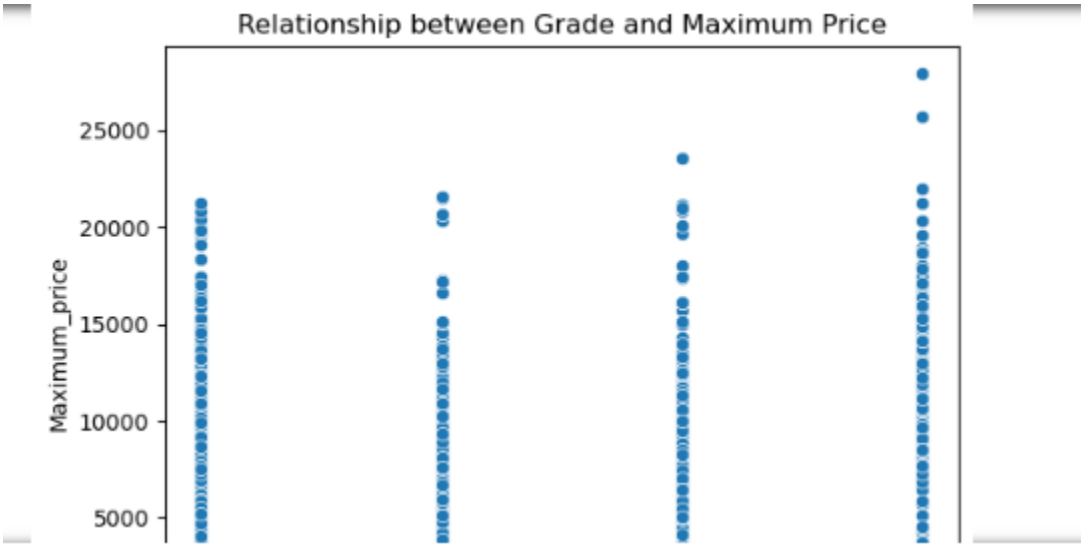
In [5]:
```
# Convert categorical variables to numerical representation
label_encoder = LabelEncoder()
df["Loyalty_customer"] = label_encoder.fit_transform(df["Loyalty_customer"])
df["Product_Category"] = label_encoder.fit_transform(df["Product_Category"])
```

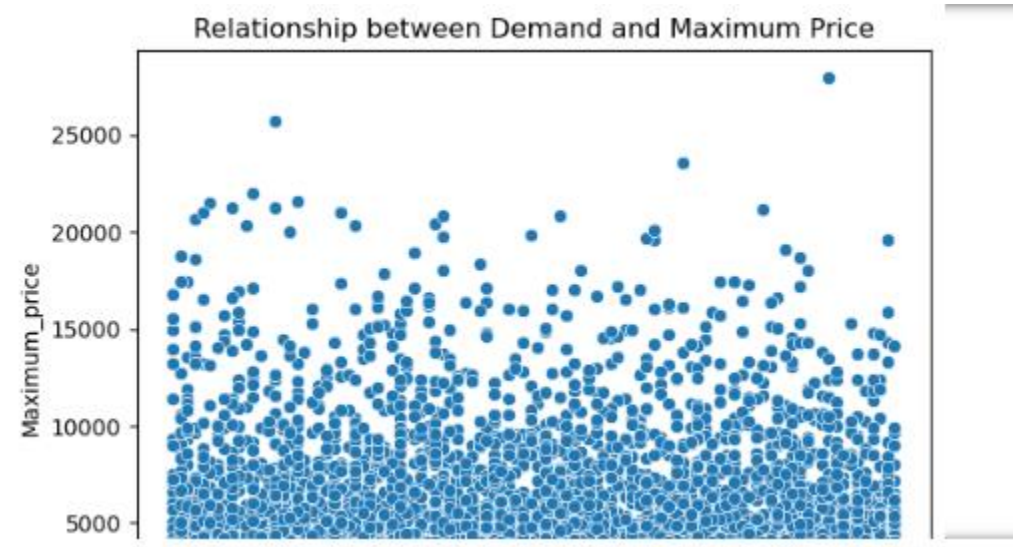
In [6]:
```
# Explore and visualize the data
# Visualize the distribution of the target variable (Maximum_price)
sns.histplot(data=df, x="Maximum_price", kde=True)
plt.title("Distribution of Maximum Price")
plt.show()
```



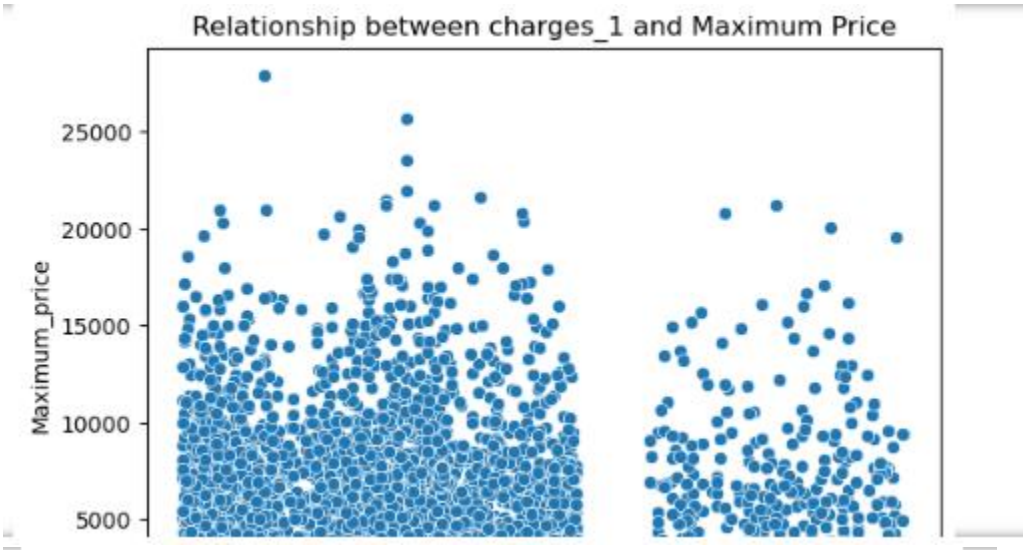
[7]:
```
# Visualize the relationship between numerical features and the target variable
numerical_features = ["Market_Category", "Grade", "Demand", "Discount_avail", "charges_1", "charges_2 (%)", "Minimum_price"]
for feature in numerical_features:
    sns.scatterplot(data=df, x=feature, y="Maximum_price")
    plt.title(f"Relationship between {feature} and Maximum Price")
    plt.show()
```

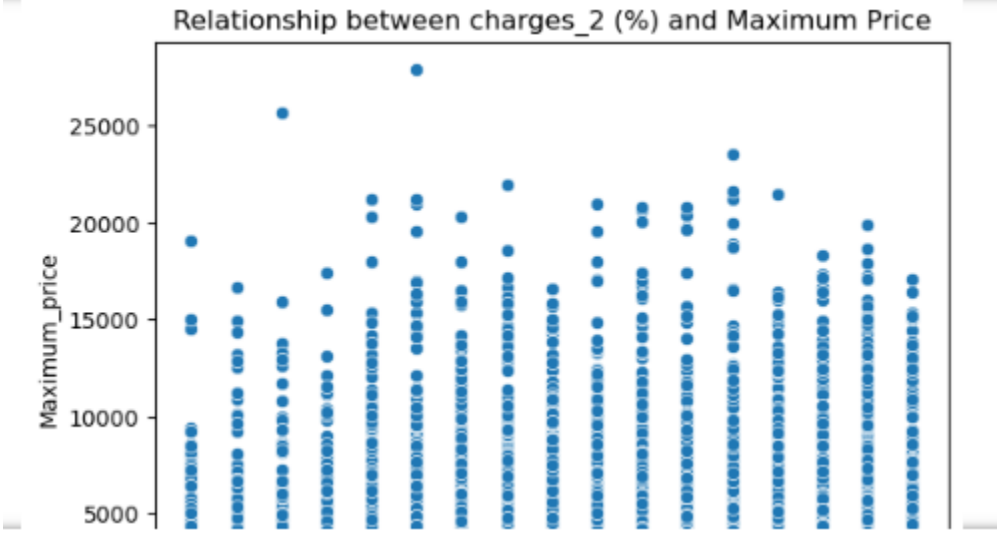

Relationship between Market_Category and Maximum Price
Maximum_price
25000
20000
15000
10000
5000

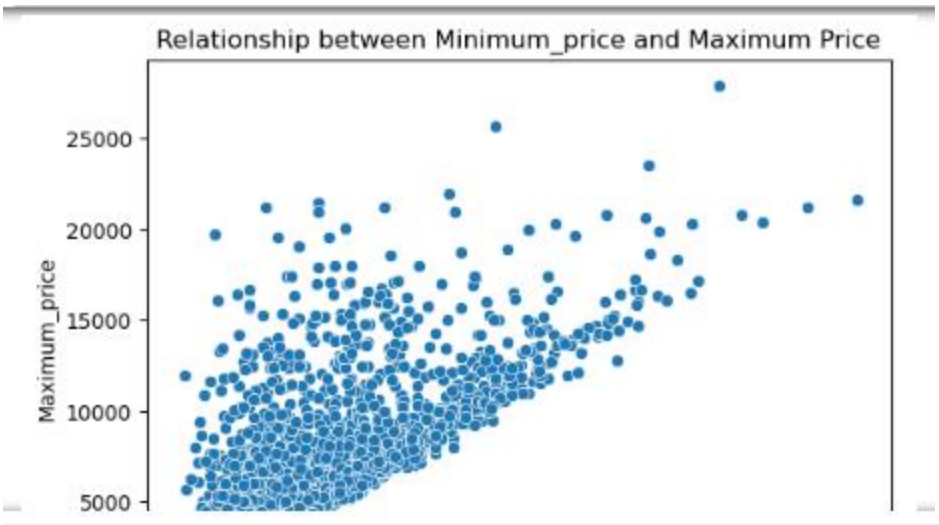

Relationship between Grade and Maximum Price
Maximum_price
25000
20000
15000
10000
5000


Relationship between Demand and Maximum Price
Maximum_price
25000
20000
15000
10000
5000


Relationship between Discount_avail and Maximum Price
Maximum_price
25000
20000
15000
10000


Relationship between charges_1 and Maximum Price
Maximum_price
25000
20000
15000
10000
5000


Relationship between charges_2 (%) and Maximum Price
Maximum_price
25000
20000
15000
10000
5000



In [9]:
```
# Split the data into training and testing sets
X = df.drop("Maximum_price", axis=1)  # Input features
y = df["Maximum_price"]  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [10]:
```
# Data scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [11]:
```
# Train the model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

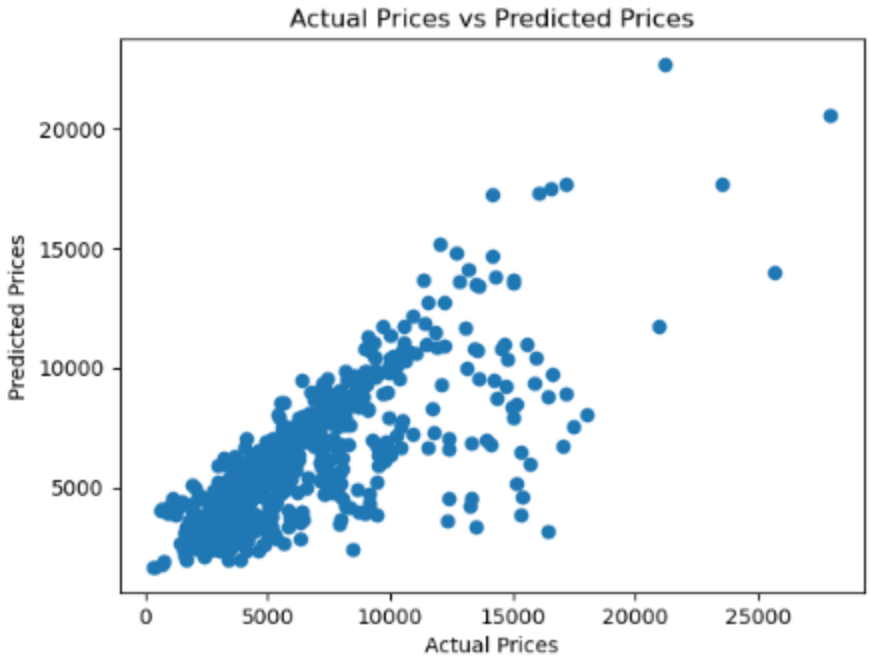
Out[11]:
```
LinearRegression()
```

In [12]:
```
# Make predictions on the testing set
y_pred = model.predict(X_test_scaled)
```

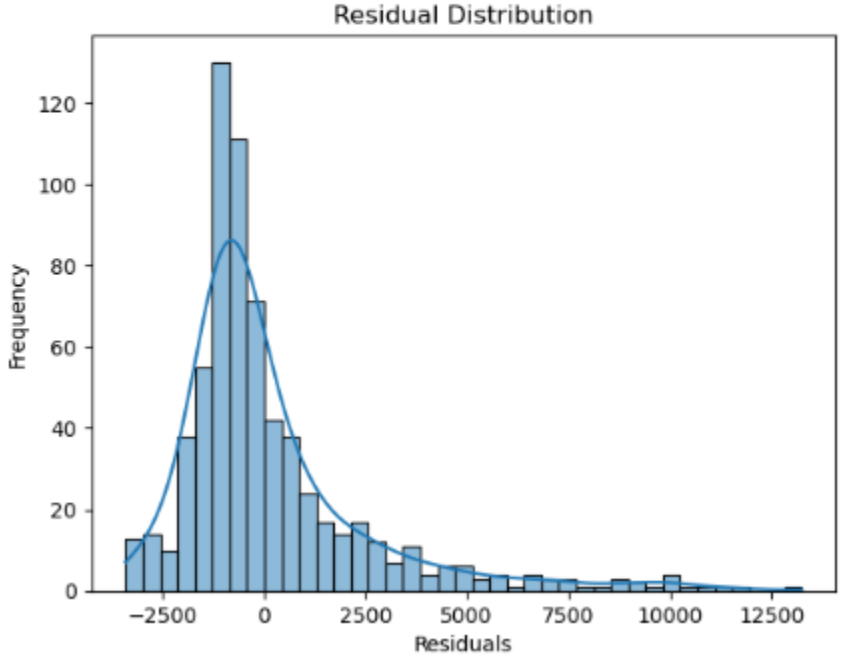
In [13]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r_squared = r2_score(y_test, y_pred)

print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r_squared)
```

```
Root Mean Squared Error (RMSE): 2445.927402023207
R-squared: 0.5733060508631739
```

In [14]:
```
# Additional graphs
# Visualize the predicted prices against the actual prices in the testing set
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs Predicted Prices")
plt.show()
```




```
# Visualize the residuals
residuals = y_test - y_pred
sns.histplot(data=residuals, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residual Distribution")
plt.show()
```



```python
# Visualize feature importance (coefficients)
feature_importance = model.coef_
feature_names = X.columns
plt.bar(feature_names, feature_importance)
plt.xlabel("Features")
plt.ylabel("Coefficient Magnitude")
plt.title("Feature Importance")
plt.xticks(rotation=90)
plt.show()
```