DMT Project Group Primrose

December 9, 2024

1 Rideshare Service Price Analysis and Prediction

1.1 Group Primrose

1.1.1 Group Members:

1. Amol Bohora
2. Ayush Shah
3. Sumit Pagar

1.1.2 Import Libraries

[187]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_text, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pd.set_option("future.no_silent_downcasting", True)
```

1.1.3 Import Data

[188]:
```
file_path="data/rideshare_kaggle.csv"
cab_rides_data=pd.read_csv(file_path)
```

[189]:
```
# Display the first few rows of the dataset
cab_rides_data.head()
```

[189]:

	id	timestamp	hour	day	month	\
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	1.544953e+09	9	16	12	
1	4bd23055-6827-41c6-b23b-3c491f24e74d	1.543284e+09	2	27	11	
2	981a3613-77af-4620-a42a-0c0866077d1e	1.543367e+09	1	28	11	

```
3  c2d88af2-d278-4bfd-a8d0-29ca77cc5512  1.543554e+09     4   30     11
4  e0126e1f-8ca9-4f2e-82b3-50505a09db9a  1.543463e+09     3   29     11

              datetime          timezone            source    destination  \
0  2018-12-16 09:30:07  America/New_York  Haymarket Square  North Station
1  2018-11-27 02:00:23  America/New_York  Haymarket Square  North Station
2  2018-11-28 01:00:22  America/New_York  Haymarket Square  North Station
3  2018-11-30 04:53:02  America/New_York  Haymarket Square  North Station
4  2018-11-29 03:49:20  America/New_York  Haymarket Square  North Station

  cab_type  … precipIntensityMax uvIndexTime  temperatureMin  \
0     Lyft  …              0.1276  1544979600           39.89
1     Lyft  …              0.1300  1543251600           40.49
2     Lyft  …              0.1064  1543338000           35.36
3     Lyft  …              0.0000  1543507200           34.67
4     Lyft  …              0.0001  1543420800           33.10

   temperatureMinTime  temperatureMax  temperatureMaxTime  \
0          1545012000           43.68          1544968800
1          1543233600           47.30          1543251600
2          1543377600           47.55          1543320000
3          1543550400           45.03          1543510800
4          1543402800           42.18          1543420800

   apparentTemperatureMin  apparentTemperatureMinTime  apparentTemperatureMax  \
0                   33.73                  1545012000                   38.07
1                   36.20                  1543291200                   43.92
2                   31.04                  1543377600                   44.12
3                   30.30                  1543550400                   38.53
4                   29.11                  1543392000                   35.75

  apparentTemperatureMaxTime
0                 1544958000
1                 1543251600
2                 1543320000
3                 1543510800
4                 1543420800

[5 rows x 57 columns]
```

[190]:
```
# Display the information about the dataset including the data types of the␣
  ↪columns
cab_rides_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 57 columns):
 #   Column                       Non-Null Count   Dtype
```

| --- | ------ | -------------- | ----- |
0	id	693071 non-null	object
1	timestamp	693071 non-null	float64
2	hour	693071 non-null	int64
3	day	693071 non-null	int64
4	month	693071 non-null	int64
5	datetime	693071 non-null	object
6	timezone	693071 non-null	object
7	source	693071 non-null	object
8	destination	693071 non-null	object
9	cab_type	693071 non-null	object
10	product_id	693071 non-null	object
11	name	693071 non-null	object
12	price	637976 non-null	float64
13	distance	693071 non-null	float64
14	surge_multiplier	693071 non-null	float64
15	latitude	693071 non-null	float64
16	longitude	693071 non-null	float64
17	temperature	693071 non-null	float64
18	apparentTemperature	693071 non-null	float64
19	short_summary	693071 non-null	object
20	long_summary	693071 non-null	object
21	precipIntensity	693071 non-null	float64
22	precipProbability	693071 non-null	float64
23	humidity	693071 non-null	float64
24	windSpeed	693071 non-null	float64
25	windGust	693071 non-null	float64
26	windGustTime	693071 non-null	int64
27	visibility	693071 non-null	float64
28	temperatureHigh	693071 non-null	float64
29	temperatureHighTime	693071 non-null	int64
30	temperatureLow	693071 non-null	float64
31	temperatureLowTime	693071 non-null	int64
32	apparentTemperatureHigh	693071 non-null	float64
33	apparentTemperatureHighTime	693071 non-null	int64
34	apparentTemperatureLow	693071 non-null	float64
35	apparentTemperatureLowTime	693071 non-null	int64
36	icon	693071 non-null	object
37	dewPoint	693071 non-null	float64
38	pressure	693071 non-null	float64
39	windBearing	693071 non-null	int64
40	cloudCover	693071 non-null	float64
41	uvIndex	693071 non-null	int64
42	visibility.1	693071 non-null	float64
43	ozone	693071 non-null	float64
44	sunriseTime	693071 non-null	int64
45	sunsetTime	693071 non-null	int64
46	moonPhase	693071 non-null	float64

```
 47  precipIntensityMax           693071 non-null  float64
 48  uvIndexTime                  693071 non-null  int64
 49  temperatureMin               693071 non-null  float64
 50  temperatureMinTime           693071 non-null  int64
 51  temperatureMax               693071 non-null  float64
 52  temperatureMaxTime           693071 non-null  int64
 53  apparentTemperatureMin       693071 non-null  float64
 54  apparentTemperatureMinTime   693071 non-null  int64
 55  apparentTemperatureMax       693071 non-null  float64
 56  apparentTemperatureMaxTime   693071 non-null  int64
dtypes: float64(29), int64(17), object(11)
memory usage: 301.4+ MB
```

1.1.4 Basic Data Validation Checks

[191]:
```
# Find the number of missing values in each column of the dataset and the␣
 ↪percentage of missing values in each column
missing_values = cab_rides_data.isnull().sum()
missing_percentage = (missing_values / len(cab_rides_data)) * 100

missing_info = pd.DataFrame({
    'Missing Values': missing_values,
    'Percentage': missing_percentage
})

print("Missing Values Report:")
print(missing_info[missing_info['Missing Values'] > 0].sort_values(by='Missing␣
 ↪Values', ascending=False))
```

```
Missing Values Report:
       Missing Values  Percentage
price           55095    7.949402
```

[192]:
```
# Find the missing values in the 'price' column for each cab company.

missing_values_per_cab_type = cab_rides_data.groupby('cab_type')['price'].
 ↪apply(lambda x: x.isnull().sum())

print("Missing Values for 'price' by Cab Type:")
print(missing_values_per_cab_type)
```

```
Missing Values for 'price' by Cab Type:
cab_type
Lyft        0
Uber    55095
Name: price, dtype: int64
```

[193]:
```
# Find the missing values in the 'price' column for each cab company.
missing_values_per_cab_type = cab_rides_data.groupby('cab_type')['price'].
 ↪apply(lambda x: x.isnull().sum())

print("Missing Values for 'price' by Cab Type:")
print(missing_values_per_cab_type)
```

```
Missing Values for 'price' by Cab Type:
cab_type
Lyft        0
Uber    55095
Name: price, dtype: int64
```

[194]:
```
# To  substitute the missing values in the 'price' column with the mean price␣
 ↪for that cab type
cab_rides_data['time_slot'] = (cab_rides_data['hour'] // 2) * 2  # e.g., 0-1␣
 ↪becomes 0, 2-3 becomes 2

# Calculate the mean price for each 2-hour time slot
slot_mean_price = cab_rides_data.groupby(['time_slot','cab_type'])['price'].
 ↪mean()

# Impute missing prices with the mean price of their respective time slot
cab_rides_data['price'] = cab_rides_data.apply(
    lambda row: slot_mean_price.get((row['time_slot'], row['cab_type']),␣
 ↪row['price'])
    if pd.isnull(row['price']) else row['price'],
    axis=1
)
```

[195]:
```
missing_values_per_cab_type=cab_rides_data.groupby('cab_type')['price'].
 ↪apply(lambda x:x.isnull().sum())
missing_values_per_cab_type
```

[195]:
```
cab_type
Lyft    0
Uber    0
Name: price, dtype: int64
```

1.1.5 Formatting Data

[196]:
```
# Create a new column 'is_rain' that indicates whether it was raining or not␣
 ↪during the ride
cab_rides_data['is_rain'] = cab_rides_data['short_summary'].str.
 ↪contains('rain', case=False).astype(int)
```

[197]:
```
#sorting by datetime column
cab_rides_data = cab_rides_data.sort_values(by='datetime')
```

[198]:
```
# Format datetime
cab_rides_data['datetime'] = pd.to_datetime(cab_rides_data['datetime'],␣
 ↪format='%Y-%m-%d %H:%M:%S')
```

[199]:
```
# Split Date and Time
cab_rides_data['date'] = cab_rides_data['datetime'].dt.date
cab_rides_data['time'] = cab_rides_data['datetime'].dt.time
cab_rides_data.head()
```

[199]:

```
                                       id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

            destination cab_type  … temperatureMax temperatureMaxTime  \
66422   Haymarket Square     Uber  …          46.15         1543154400
446073         North End     Uber  …          46.15         1543154400
184332          West End     Lyft  …          46.15         1543154400
167114       Beacon Hill     Lyft  …          46.15         1543154400
184333          West End     Lyft  …          46.15         1543154400

        apparentTemperatureMin  apparentTemperatureMinTime  \
66422                    38.23                  1543136400
446073                   38.23                  1543136400
184332                   38.23                  1543136400
167114                   38.23                  1543136400
184333                   38.23                  1543136400

        apparentTemperatureMax  apparentTemperatureMaxTime  time_slot  \
66422                    43.17                  1543186800          2
446073                   43.17                  1543186800          2
184332                   43.17                  1543186800          2
167114                   43.17                  1543186800          2
184333                   43.17                  1543186800          2
```

```
        is_rain        date      time
66422         0  2018-11-26  03:40:46
446073        0  2018-11-26  03:40:46
184332        0  2018-11-26  03:40:46
167114        0  2018-11-26  03:40:46
184333        0  2018-11-26  03:40:46

[5 rows x 61 columns]
```

[200]:
```
# Create "odd_time" column
cab_rides_data['odd_time'] = cab_rides_data['time'].apply(lambda x: 1 if x.hour␣
 ↪< 6 else 0)

# Create "peak_time" column
cab_rides_data['peak_time'] = cab_rides_data['time'].apply(lambda x: 1 if (x.
 ↪hour >= 8 and x.hour <= 10) or (x.hour >= 16 and x.hour <= 19) else 0)

# Print the updated dataframe
cab_rides_data.head()
```

[200]:
```
                                     id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

            destination cab_type  … apparentTemperatureMin  \
66422   Haymarket Square     Uber  …                  38.23
446073         North End     Uber  …                  38.23
184332          West End     Lyft  …                  38.23
167114       Beacon Hill     Lyft  …                  38.23
184333          West End     Lyft  …                  38.23

       apparentTemperatureMinTime  apparentTemperatureMax  \
66422                  1543136400                   43.17
446073                 1543136400                   43.17
184332                 1543136400                   43.17
167114                 1543136400                   43.17
184333                 1543136400                   43.17
```

```
        apparentTemperatureMaxTime  time_slot  is_rain        date      time  \
66422                   1543186800          2        0  2018-11-26  03:40:46
446073                  1543186800          2        0  2018-11-26  03:40:46
184332                  1543186800          2        0  2018-11-26  03:40:46
167114                  1543186800          2        0  2018-11-26  03:40:46
184333                  1543186800          2        0  2018-11-26  03:40:46

        odd_time peak_time
66422          1         0
446073         1         0
184332         1         0
167114         1         0
184333         1         0

[5 rows x 63 columns]
```

[201]:
```
# Add a column which stores was the ride taken in day or night
cab_rides_data['is_night'] = cab_rides_data.apply(
    lambda row: not (row['sunriseTime'] <= row['datetime'].timestamp() <=␣
 ↪row['sunsetTime']),
    axis=1
)

# Print the updated DataFrame
cab_rides_data.head()
```

[201]:
```
                                      id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

             destination cab_type  … apparentTemperatureMinTime  \
66422   Haymarket Square     Uber  …                 1543136400
446073         North End     Uber  …                 1543136400
184332          West End     Lyft  …                 1543136400
167114       Beacon Hill     Lyft  …                 1543136400
184333          West End     Lyft  …                 1543136400
```

```
       apparentTemperatureMax  apparentTemperatureMaxTime  time_slot  is_rain  \
66422                   43.17                  1543186800          2        0
446073                  43.17                  1543186800          2        0
184332                  43.17                  1543186800          2        0
167114                  43.17                  1543186800          2        0
184333                  43.17                  1543186800          2        0

              date      time  odd_time  peak_time is_night
66422   2018-11-26  03:40:46         1          0     True
446073  2018-11-26  03:40:46         1          0     True
184332  2018-11-26  03:40:46         1          0     True
167114  2018-11-26  03:40:46         1          0     True
184333  2018-11-26  03:40:46         1          0     True

[5 rows x 64 columns]
```

[202]:
```
# Rename column 'cab_type' to 'cab_company', 'name' to 'cab_type', 'odd_time'␣
 ↪to 'odd_time_of_travel' of cab_rides_data
cab_rides_data.rename(columns={
    'cab_type': 'cab_company',
    'odd_time': 'odd_time_of_travel'
}, inplace=True)

cab_rides_data.rename(columns={
    'name': 'cab_type'
}, inplace=True)

# Display the first few rows of the dataset
cab_rides_data.head()
```

[202]:
```
                                     id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

            destination cab_company  … apparentTemperatureMinTime  \
66422   Haymarket Square        Uber  …                 1543136400
```

```
446073         North End        Uber  …                1543136400
184332          West End        Lyft  …                1543136400
167114       Beacon Hill        Lyft  …                1543136400
184333          West End        Lyft  …                1543136400

       apparentTemperatureMax  apparentTemperatureMaxTime  time_slot  is_rain  \
66422                   43.17                  1543186800          2        0
446073                  43.17                  1543186800          2        0
184332                  43.17                  1543186800          2        0
167114                  43.17                  1543186800          2        0
184333                  43.17                  1543186800          2        0

              date      time  odd_time_of_travel  peak_time is_night
66422   2018-11-26  03:40:46                   1          0     True
446073  2018-11-26  03:40:46                   1          0     True
184332  2018-11-26  03:40:46                   1          0     True
167114  2018-11-26  03:40:46                   1          0     True
184333  2018-11-26  03:40:46                   1          0     True

[5 rows x 64 columns]
```

[203]:
```
# Create a new column 'day_of_week' that indicates the day of the week for each␣
 ↪ride
cab_rides_data['day_of_week'] = cab_rides_data['datetime'].dt.day_name()
```

[204]:
```
# Create "is_weekend" column that indicates whether the ride was on a weekend␣
 ↪or not
cab_rides_data['is_weekend'] = cab_rides_data['day_of_week'].apply(lambda x: 1␣
 ↪if x=="Saturday" or x=="Sunday" else 0)
cab_rides_data.head()
```

[204]:
```
                                      id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

            destination cab_company  … apparentTemperatureMaxTime  \
66422   Haymarket Square        Uber  …                  1543186800
```

```
446073         North End        Uber  …                 1543186800
184332          West End        Lyft  …                 1543186800
167114       Beacon Hill        Lyft  …                 1543186800
184333          West End        Lyft  …                 1543186800

       time_slot  is_rain        date      time  odd_time_of_travel  \
66422          2        0  2018-11-26  03:40:46                   1
446073         2        0  2018-11-26  03:40:46                   1
184332         2        0  2018-11-26  03:40:46                   1
167114         2        0  2018-11-26  03:40:46                   1
184333         2        0  2018-11-26  03:40:46                   1

        peak_time  is_night  day_of_week is_weekend
66422           0      True       Monday          0
446073          0      True       Monday          0
184332          0      True       Monday          0
167114          0      True       Monday          0
184333          0      True       Monday          0

[5 rows x 66 columns]
```

[205]:
```
# Convert 0 to False and 1 to True in the specified columns
columns_to_convert = ['peak_time', 'is_weekend', 'odd_time_of_travel',␣
 ↪'is_rain']

cab_rides_data[columns_to_convert] = cab_rides_data[columns_to_convert].apply(
    lambda col: col.replace({0: False, 1: True}).astype(bool)
)

cab_rides_data.head()
```

[205]:
```
                                      id     timestamp  hour  day  month  \
66422   a7b50600-c6c5-4e6c-bea9-4487344196d4  1.543204e+09     3   26     11
446073  9962f244-8fce-4ae9-a583-139d5d7522e1  1.543204e+09     3   26     11
184332  4aa68a5d-abc0-4fdf-a47f-0003617afbae  1.543204e+09     3   26     11
167114  ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4  1.543204e+09     3   26     11
184333  89f35ef7-7129-483d-b3e6-d89afdf6946d  1.543204e+09     3   26     11

                  datetime          timezone             source  \
66422  2018-11-26 03:40:46  America/New_York      North Station
446073 2018-11-26 03:40:46  America/New_York   Theatre District
184332 2018-11-26 03:40:46  America/New_York          North End
167114 2018-11-26 03:40:46  America/New_York  Boston University
184333 2018-11-26 03:40:46  America/New_York          North End

            destination cab_company  … apparentTemperatureMaxTime  \
66422   Haymarket Square        Uber  …                 1543186800
```

```
446073         North End        Uber  …                1543186800
184332          West End        Lyft  …                1543186800
167114       Beacon Hill        Lyft  …                1543186800
184333          West End        Lyft  …                1543186800

       time_slot  is_rain        date      time  odd_time_of_travel  \
66422          2    False  2018-11-26  03:40:46                True
446073         2    False  2018-11-26  03:40:46                True
184332         2    False  2018-11-26  03:40:46                True
167114         2    False  2018-11-26  03:40:46                True
184333         2    False  2018-11-26  03:40:46                True

        peak_time  is_night  day_of_week is_weekend
66422       False      True       Monday      False
446073      False      True       Monday      False
184332      False      True       Monday      False
167114      False      True       Monday      False
184333      False      True       Monday      False

[5 rows x 66 columns]
```

[206]:
```python
# Cleanup before selecting data
cab_rides_data['year'] = cab_rides_data['datetime'].dt.year
cab_rides_data['month'] = cab_rides_data['datetime'].dt.month
cab_rides_data['day'] = cab_rides_data['datetime'].dt.day
cab_rides_data['hour'] = cab_rides_data['datetime'].dt.hour
cab_rides_data['minute'] = cab_rides_data['datetime'].dt.minute
cab_rides_data['weekday'] = cab_rides_data['datetime'].dt.weekday

# Ensure boolean columns are explicitly cast to boolean type
cab_rides_data['is_night'] = cab_rides_data['is_night'].astype(bool)
cab_rides_data['is_rain'] = cab_rides_data['is_rain'].astype(bool)
cab_rides_data['is_weekend'] = cab_rides_data['is_weekend'].astype(bool)

cab_rides_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 693071 entries, 66422 to 166551
Data columns (total 69 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   id                           693071 non-null  object
 1   timestamp                    693071 non-null  float64
 2   hour                         693071 non-null  int32
 3   day                          693071 non-null  int32
 4   month                        693071 non-null  int32
 5   datetime                     693071 non-null  datetime64[ns]
 6   timezone                     693071 non-null  object
```

```
7   source                      693071 non-null  object
8   destination                 693071 non-null  object
9   cab_company                 693071 non-null  object
10  product_id                  693071 non-null  object
11  cab_type                    693071 non-null  object
12  price                       693071 non-null  float64
13  distance                    693071 non-null  float64
14  surge_multiplier            693071 non-null  float64
15  latitude                    693071 non-null  float64
16  longitude                   693071 non-null  float64
17  temperature                 693071 non-null  float64
18  apparentTemperature         693071 non-null  float64
19  short_summary               693071 non-null  object
20  long_summary                693071 non-null  object
21  precipIntensity             693071 non-null  float64
22  precipProbability           693071 non-null  float64
23  humidity                    693071 non-null  float64
24  windSpeed                   693071 non-null  float64
25  windGust                    693071 non-null  float64
26  windGustTime                693071 non-null  int64
27  visibility                  693071 non-null  float64
28  temperatureHigh             693071 non-null  float64
29  temperatureHighTime         693071 non-null  int64
30  temperatureLow              693071 non-null  float64
31  temperatureLowTime          693071 non-null  int64
32  apparentTemperatureHigh     693071 non-null  float64
33  apparentTemperatureHighTime 693071 non-null  int64
34  apparentTemperatureLow      693071 non-null  float64
35  apparentTemperatureLowTime  693071 non-null  int64
36  icon                        693071 non-null  object
37  dewPoint                    693071 non-null  float64
38  pressure                    693071 non-null  float64
39  windBearing                 693071 non-null  int64
40  cloudCover                  693071 non-null  float64
41  uvIndex                     693071 non-null  int64
42  visibility.1                693071 non-null  float64
43  ozone                       693071 non-null  float64
44  sunriseTime                 693071 non-null  int64
45  sunsetTime                  693071 non-null  int64
46  moonPhase                   693071 non-null  float64
47  precipIntensityMax          693071 non-null  float64
48  uvIndexTime                 693071 non-null  int64
49  temperatureMin              693071 non-null  float64
50  temperatureMinTime          693071 non-null  int64
51  temperatureMax              693071 non-null  float64
52  temperatureMaxTime          693071 non-null  int64
53  apparentTemperatureMin      693071 non-null  float64
54  apparentTemperatureMinTime  693071 non-null  int64
```

```
 55  apparentTemperatureMax       693071 non-null  float64
 56  apparentTemperatureMaxTime   693071 non-null  int64
 57  time_slot                    693071 non-null  int64
 58  is_rain                      693071 non-null  bool
 59  date                         693071 non-null  object
 60  time                         693071 non-null  object
 61  odd_time_of_travel           693071 non-null  bool
 62  peak_time                    693071 non-null  bool
 63  is_night                     693071 non-null  bool
 64  day_of_week                  693071 non-null  object
 65  is_weekend                   693071 non-null  bool
 66  year                         693071 non-null  int32
 67  minute                       693071 non-null  int32
 68  weekday                      693071 non-null  int32
dtypes: bool(5), datetime64[ns](1), float64(29), int32(6), int64(15), object(13)
memory usage: 331.1+ MB
```

[207]:
```
### Create a DF of selected features
```

[208]:
```
columns_to_include = [
    'source', 'destination', 'cab_company', 'cab_type', 'price', 'distance',
    'surge_multiplier', 'apparentTemperature', 'precipIntensity',
    'visibility.1', 'precipIntensityMax', 'day', 'month', 'hour', 'minute',
]

selected_features = cab_rides_data[columns_to_include]
selected_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 693071 entries, 66422 to 166551
Data columns (total 15 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   source               693071 non-null  object
 1   destination          693071 non-null  object
 2   cab_company          693071 non-null  object
 3   cab_type             693071 non-null  object
 4   price                693071 non-null  float64
 5   distance             693071 non-null  float64
 6   surge_multiplier     693071 non-null  float64
 7   apparentTemperature  693071 non-null  float64
 8   precipIntensity      693071 non-null  float64
 9   visibility.1         693071 non-null  float64
 10  precipIntensityMax   693071 non-null  float64
 11  day                  693071 non-null  int32
 12  month                693071 non-null  int32
 13  hour                 693071 non-null  int32
 14  minute               693071 non-null  int32
```

```
dtypes: float64(7), int32(4), object(4)
memory usage: 74.0+ MB
```

1.1.6 Visualization

[209]:
```
# Create a deep copy of the dataset for visualisation
cab_rides_data_for_visualisation =cab_rides_data.copy(deep=True)
cab_rides_data_for_visualisation.head()
```

[209]:

	id	timestamp	hour	day	month	\
66422	a7b50600-c6c5-4e6c-bea9-4487344196d4	1.543204e+09	3	26	11	
446073	9962f244-8fce-4ae9-a583-139d5d7522e1	1.543204e+09	3	26	11	
184332	4aa68a5d-abc0-4fdf-a47f-0003617afbae	1.543204e+09	3	26	11	
167114	ef8b695c-c24d-4ac1-b3fe-4aa1a7ed79f4	1.543204e+09	3	26	11	
184333	89f35ef7-7129-483d-b3e6-d89afdf6946d	1.543204e+09	3	26	11	

	datetime	timezone	source	\
66422	2018-11-26 03:40:46	America/New_York	North Station	
446073	2018-11-26 03:40:46	America/New_York	Theatre District	
184332	2018-11-26 03:40:46	America/New_York	North End	
167114	2018-11-26 03:40:46	America/New_York	Boston University	
184333	2018-11-26 03:40:46	America/New_York	North End	

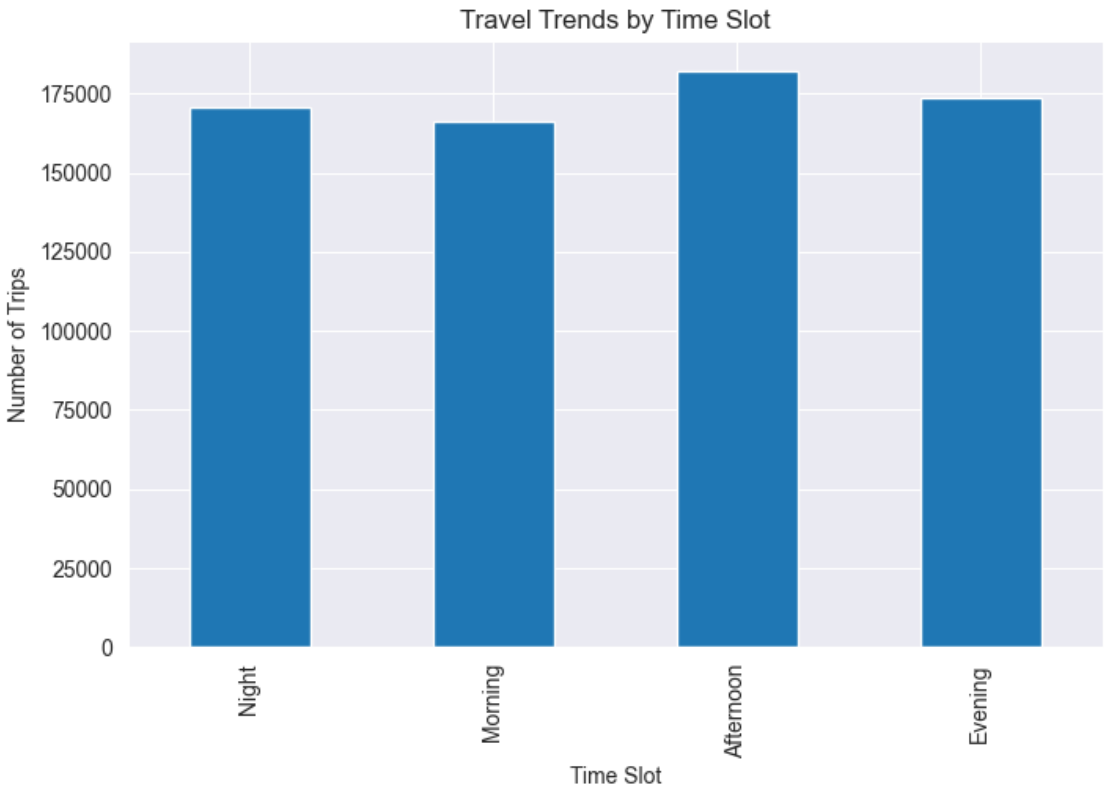
	destination	cab_company	…	date	time	\
66422	Haymarket Square	Uber	…	2018-11-26	03:40:46	
446073	North End	Uber	…	2018-11-26	03:40:46	
184332	West End	Lyft	…	2018-11-26	03:40:46	
167114	Beacon Hill	Lyft	…	2018-11-26	03:40:46	
184333	West End	Lyft	…	2018-11-26	03:40:46	

	odd_time_of_travel	peak_time	is_night	day_of_week	is_weekend	\
66422	True	False	True	Monday	False	
446073	True	False	True	Monday	False	
184332	True	False	True	Monday	False	
167114	True	False	True	Monday	False	
184333	True	False	True	Monday	False	

	year	minute	weekday
66422	2018	40	0
446073	2018	40	0
184332	2018	40	0
167114	2018	40	0
184333	2018	40	0

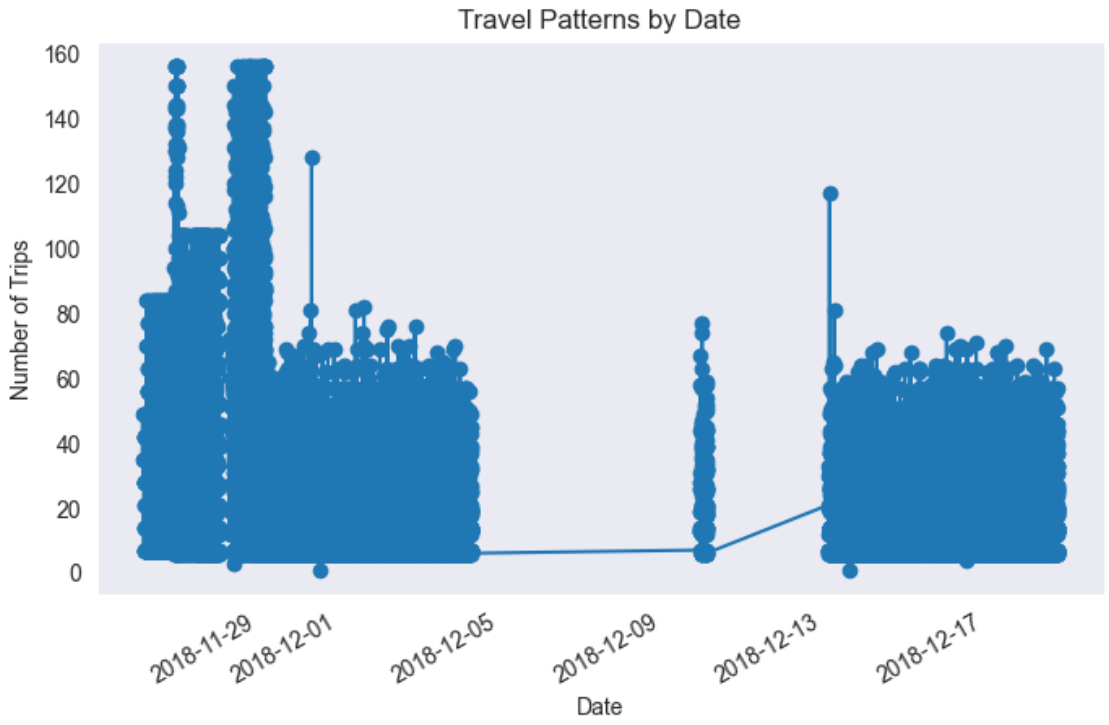
[5 rows x 69 columns]

[210]:
```
# Create a graph to visualize the distribution of rides

cab_rides_data_for_visualisation['time_slot'] = pd.
 ↪cut(cab_rides_data_for_visualisation['hour'], bins=[0, 6, 12, 18, 24],␣
 ↪labels=['Night', 'Morning', 'Afternoon', 'Evening'], right=False)
time_slot_counts = cab_rides_data_for_visualisation['time_slot'].value_counts().
 ↪sort_index()

plt.figure(figsize=(8, 5))
time_slot_counts.plot(kind='bar', title='Travel Trends by Time Slot')
plt.xlabel('Time Slot')
plt.ylabel('Number of Trips')
plt.show()
```

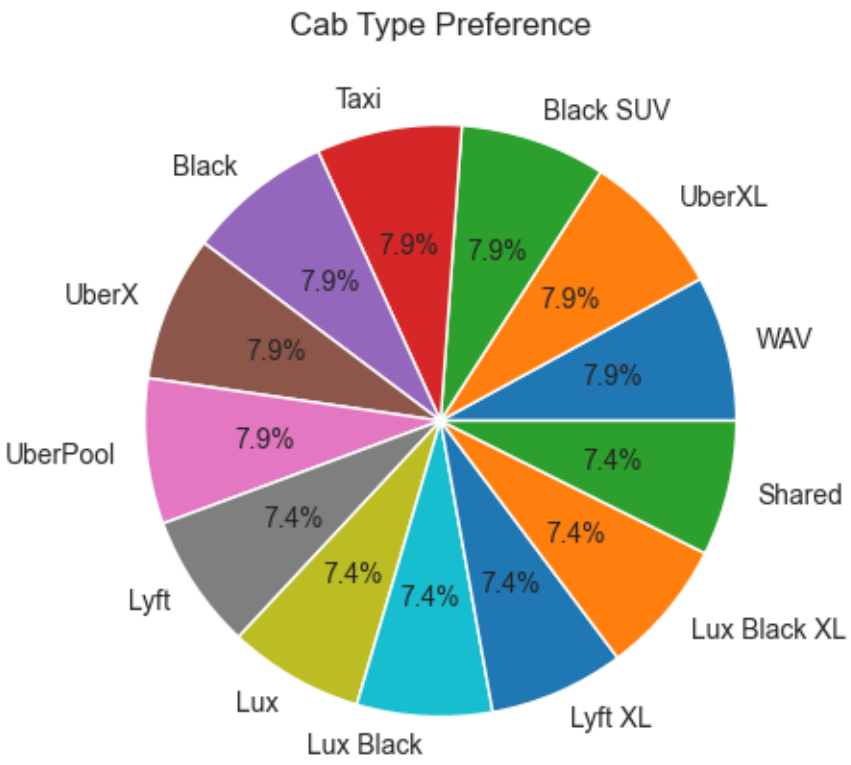


[211]:
```
# Plot Travel Patterns
date_counts = cab_rides_data_for_visualisation['datetime'].value_counts().
 ↪sort_index()

plt.figure(figsize=(8, 5))
date_counts.plot(kind='line', marker='o', title='Travel Patterns by Date')
plt.xlabel('Date')
```

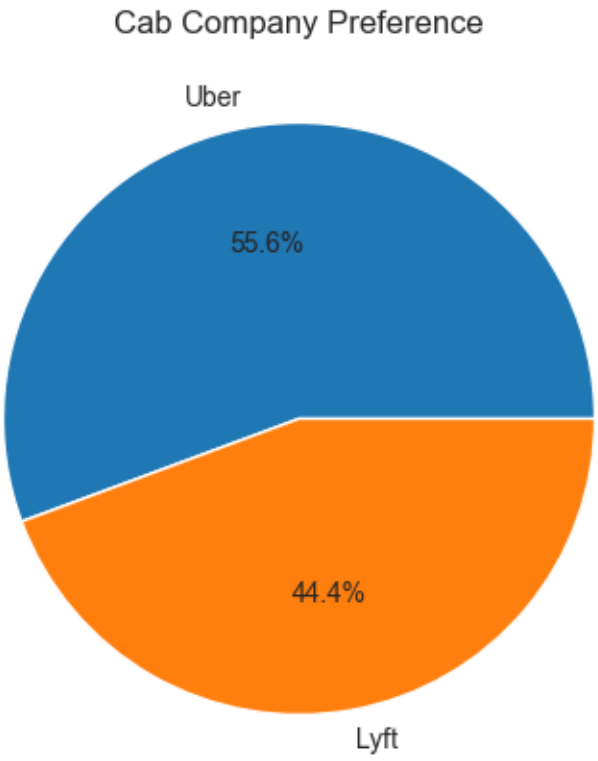
```
plt.ylabel('Number of Trips')
plt.grid()
plt.show()
```



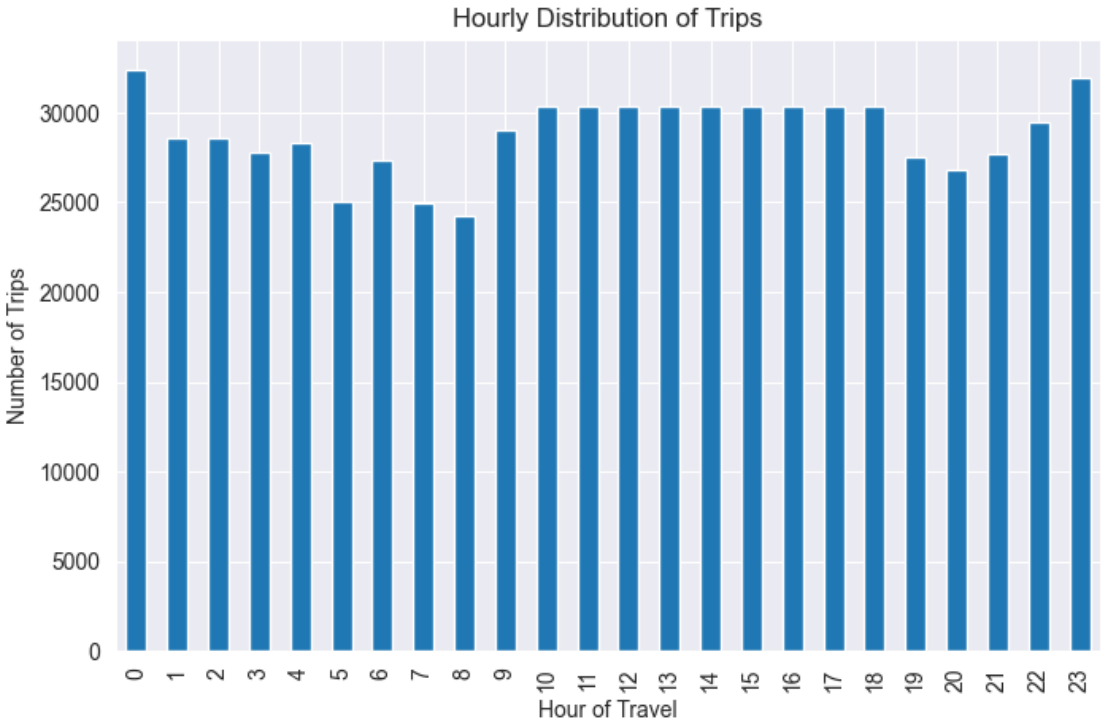
[212]:
```
# Create a graph to visualize the distribution of rides by cab type
cab_type_counts = cab_rides_data_for_visualisation['cab_type'].value_counts()

plt.figure(figsize=(8, 5))
cab_type_counts.plot(kind='pie', autopct='%1.1f%%', title='Cab Type Preference')
plt.ylabel('')
plt.show()
```



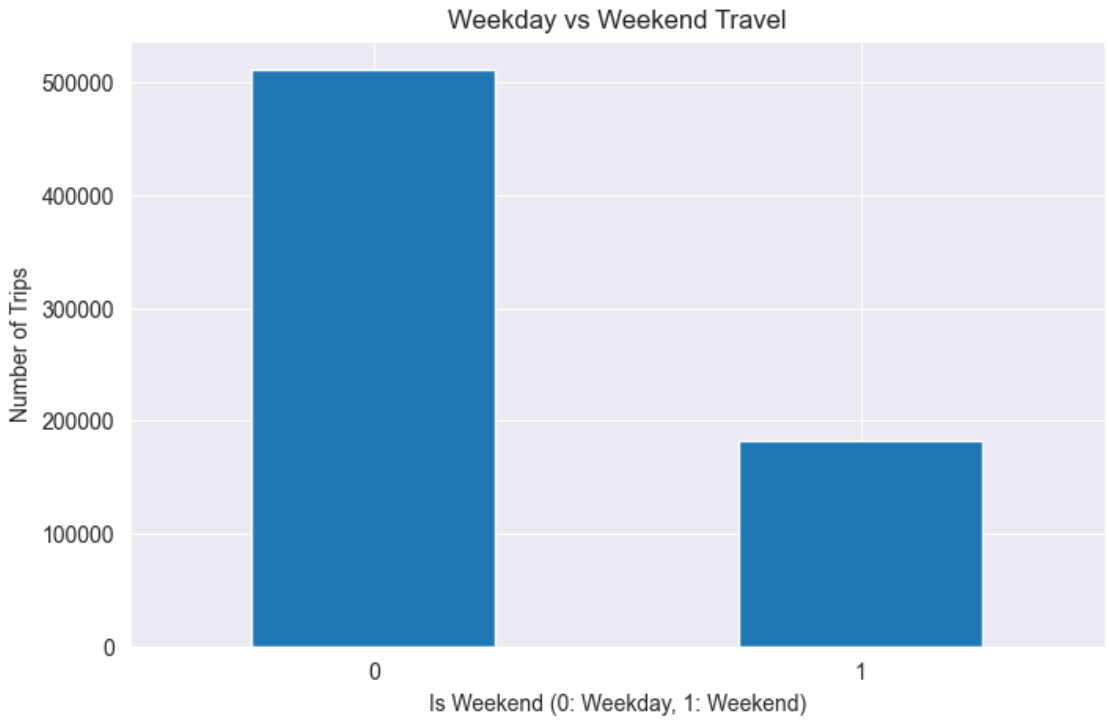
[213]:
```
# Create a graph to visualize the distribution of rides by cab company
cab_company_counts = cab_rides_data_for_visualisation['cab_company'].
 ↪value_counts()

plt.figure(figsize=(8, 5))
cab_company_counts.plot(kind='pie', autopct='%1.1f%%', title='Cab Company␣
 ↪Preference')
plt.ylabel('')
plt.show()
```



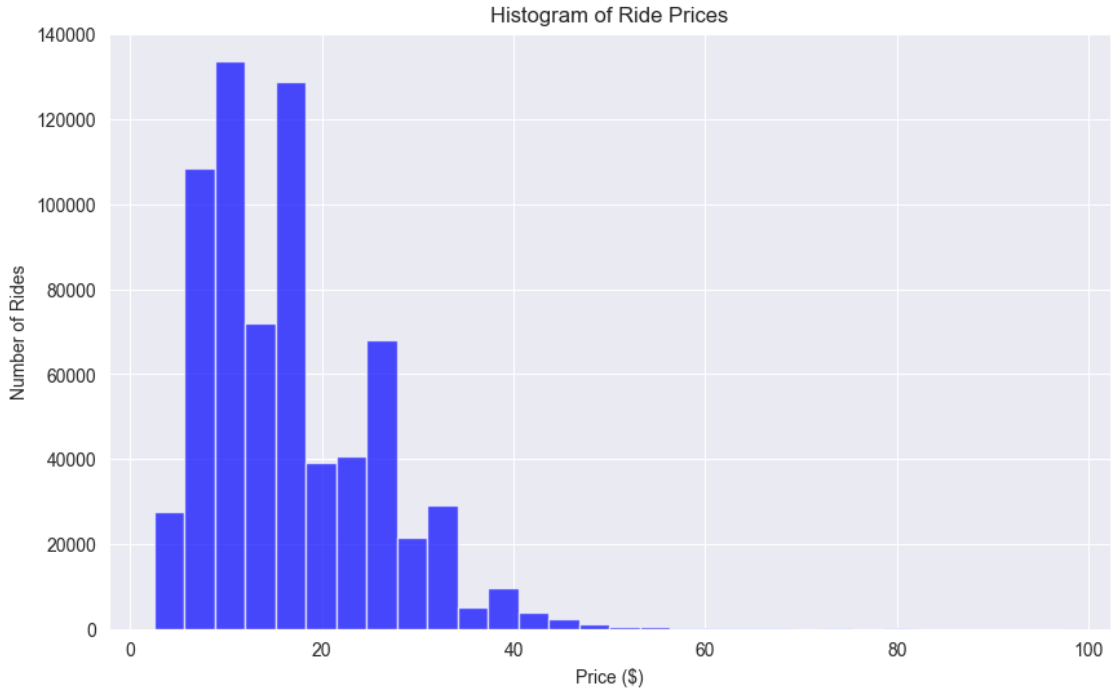
[214]:
```
# Hourly Distribution of Trips by Hour of Travel
hourly_counts = cab_rides_data_for_visualisation['hour'].value_counts().
 ↪sort_index()

plt.figure(figsize=(8, 5))
hourly_counts.plot(kind='bar', title='Hourly Distribution of Trips')
plt.xlabel('Hour of Travel')
plt.ylabel('Number of Trips')
plt.show()
```



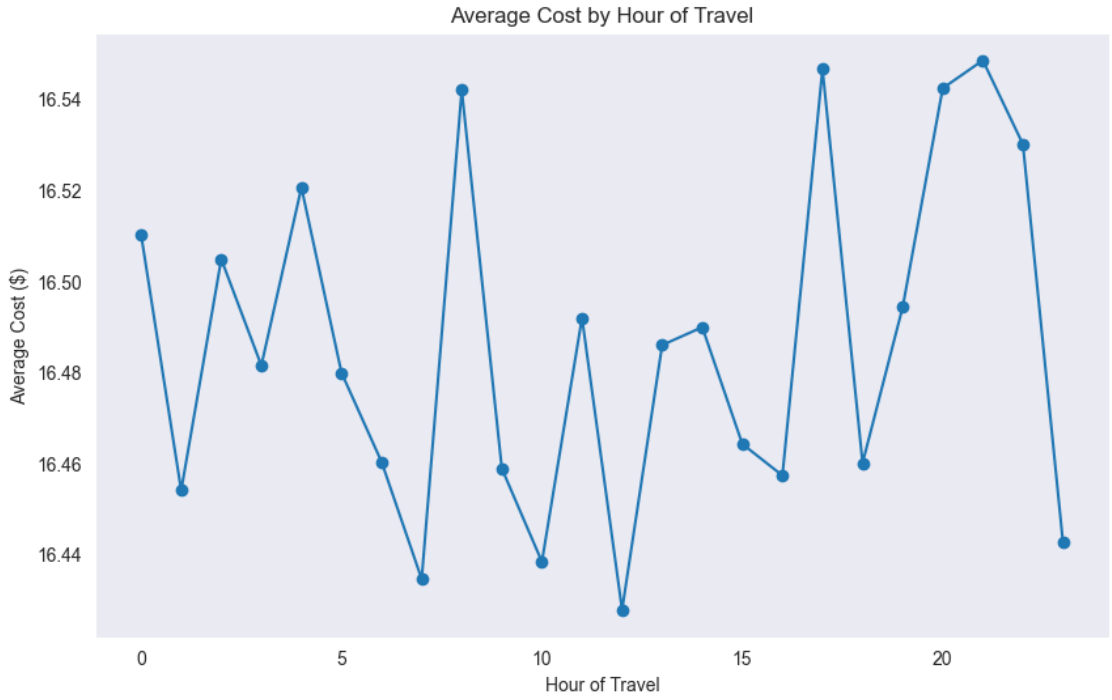
[215]:
```
# Visualize the distribution of rides by the week or weekend

cab_rides_data_for_visualisation['weekday'] = pd.
 ↪to_datetime(cab_rides_data_for_visualisation['date']).dt.dayofweek
cab_rides_data_for_visualisation['is_weekend'] =␣
 ↪cab_rides_data_for_visualisation['weekday'].apply(lambda x: 1 if x >= 5 else␣
 ↪0)
weekday_counts = cab_rides_data_for_visualisation.groupby('is_weekend')['hour'].
 ↪count()

plt.figure(figsize=(8, 5))
weekday_counts.plot(kind='bar', title='Weekday vs Weekend Travel')
plt.xlabel('Is Weekend (0: Weekday, 1: Weekend)')
plt.ylabel('Number of Trips')
plt.xticks(rotation=0)
plt.show()
```



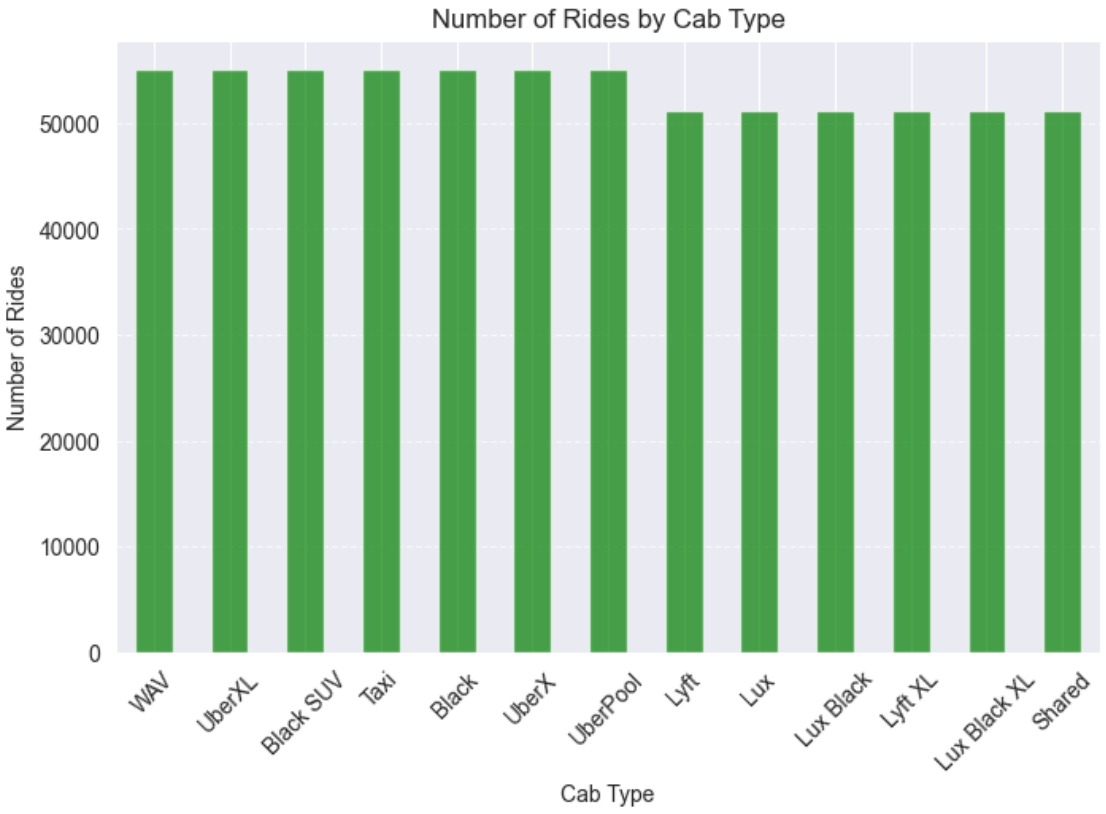
[216]:
```
# Creating a histogram of ride prices
plt.figure(figsize=(10, 6))
plt.hist(cab_rides_data_for_visualisation['price'], bins=30, color='blue',␣
 ↪alpha=0.7)
plt.title('Histogram of Ride Prices')
plt.xlabel('Price ($)')
plt.ylabel('Number of Rides')
plt.grid(True)
plt.show()
```



[217]:
```
# Avg cost of travel by the hour when the ride was taken
average_cost_by_hour = cab_rides_data_for_visualisation.
 ↪groupby('hour')['price'].mean()

# 2. Identify peak and low-cost hours
peak_cost_hour = average_cost_by_hour.idxmax()
peak_cost_value = average_cost_by_hour.max()

low_cost_hour = average_cost_by_hour.idxmin()
low_cost_value = average_cost_by_hour.min()

# 3. Visualize average cost across hours
plt.figure(figsize=(10, 6))
average_cost_by_hour.plot(kind='line', marker='o', title='Average Cost by Hour␣
 ↪of Travel')
plt.xlabel('Hour of Travel')
plt.ylabel('Average Cost ($)')
plt.grid()
plt.show()
```

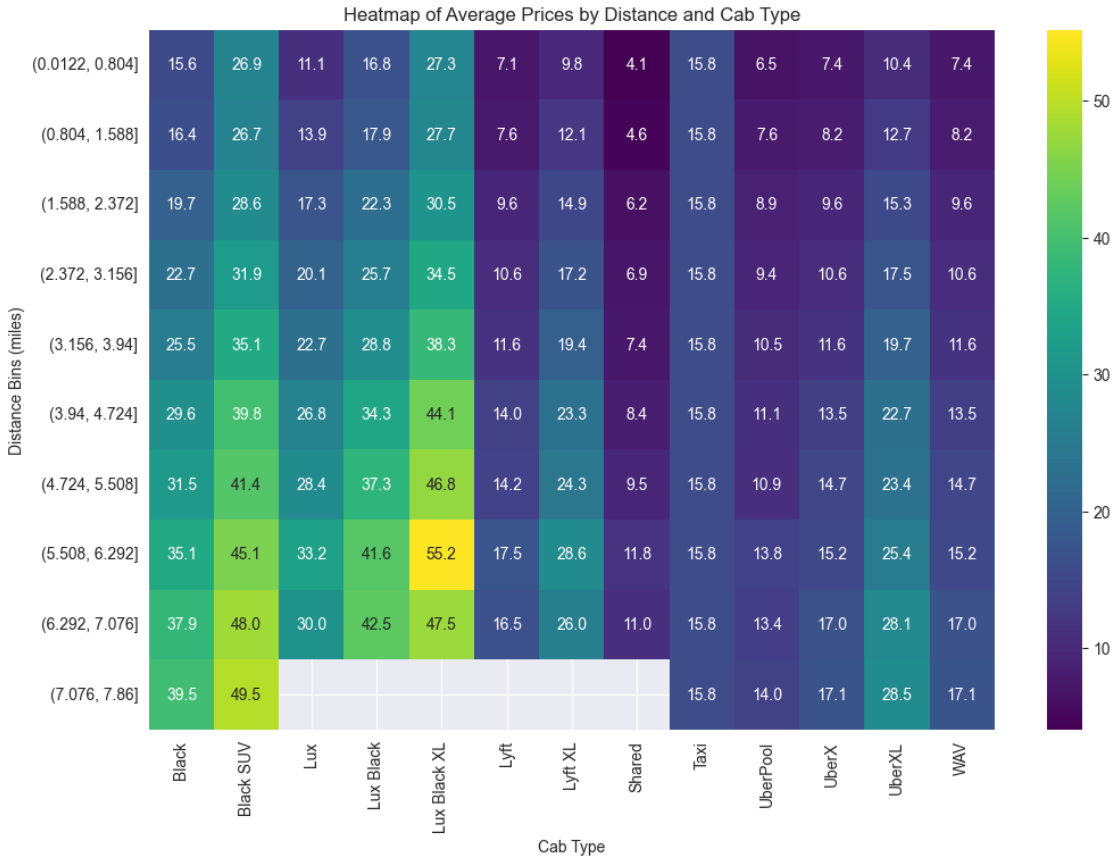


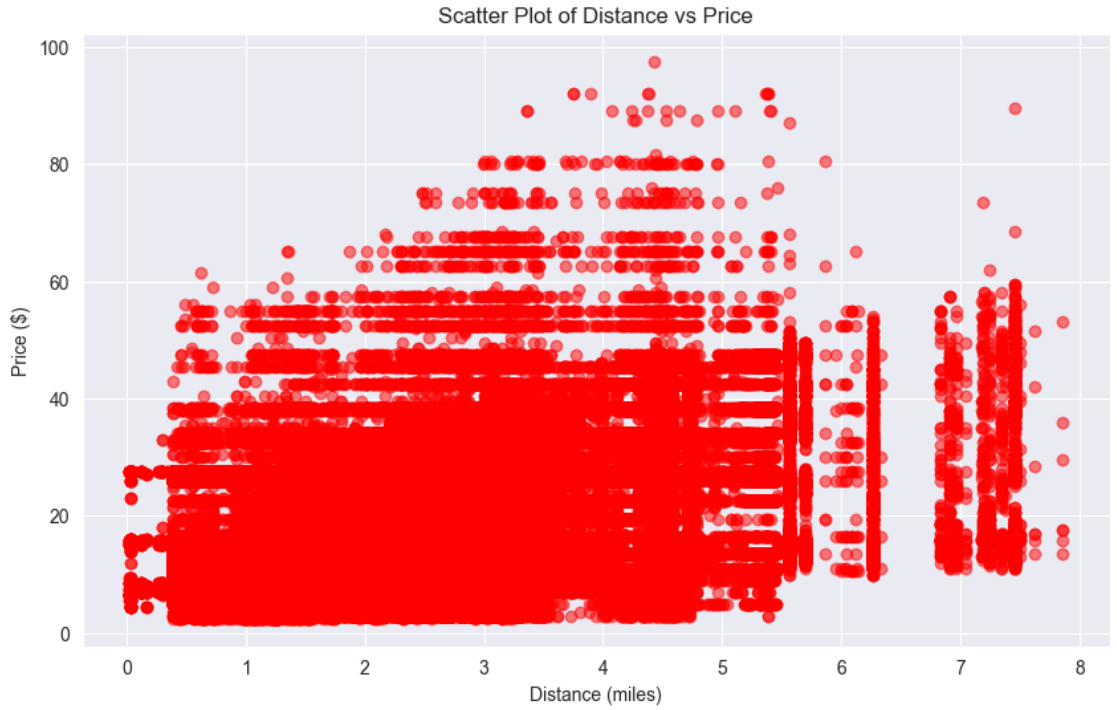
[218]:
```
# Creating a bar chart of cab types
cab_type_counts = cab_rides_data_for_visualisation['cab_type'].value_counts()

plt.figure(figsize=(8, 5))
cab_type_counts.plot(kind='bar', color='green', alpha=0.7)
plt.title('Number of Rides by Cab Type')
plt.xlabel('Cab Type')
plt.ylabel('Number of Rides')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

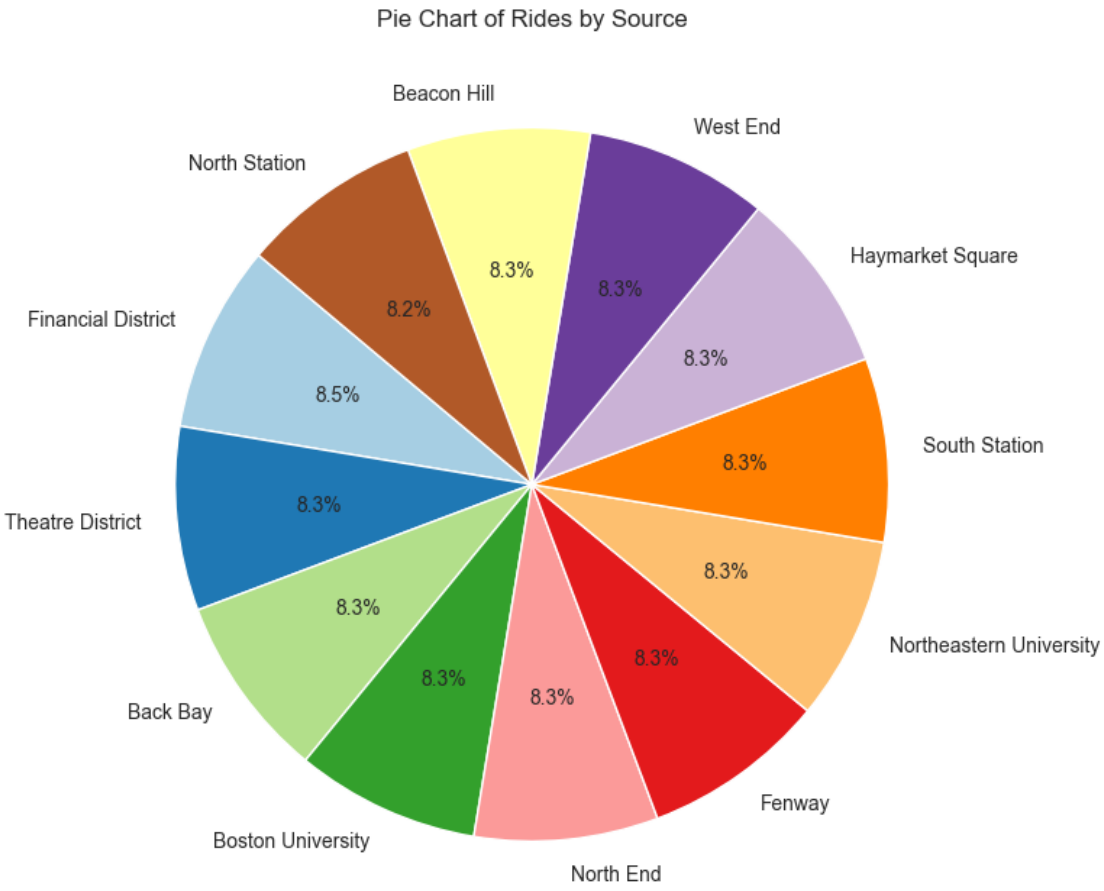


[219]:
```
# Creating Heatmap of Average Prices by Distance and Cab Type

# Creating distance bins for better visualization
cab_rides_data_for_visualisation['distance_bins'] = pd.
 ↪cut(cab_rides_data_for_visualisation['distance'], bins=10)

# Creating a pivot table to analyze prices across distances and cab types
price_heatmap_data = cab_rides_data_for_visualisation.pivot_table(
    values='price',
    index='distance_bins',
    columns='cab_type',
    aggfunc="mean",
    observed=False
)

# Plotting the heatmap
plt.figure(figsize=(12, 8))
plt.title('Heatmap of Average Prices by Distance and Cab Type')
sns.heatmap(price_heatmap_data, annot=True, fmt=".1f", cmap='viridis')
plt.xlabel('Cab Type')
plt.ylabel('Distance Bins (miles)')
```

```
plt.show()
# Creating a scatter plot of distance vs price
plt.figure(figsize=(10, 6))
plt.scatter(cab_rides_data_for_visualisation['distance'],␣
 ↪cab_rides_data_for_visualisation['price'], alpha=0.5, color='red')
plt.title('Scatter Plot of Distance vs Price')
plt.xlabel('Distance (miles)')
plt.ylabel('Price ($)')
plt.grid(True)
plt.show()

# Creating a pie chart of rides by source
source_counts = cab_rides_data_for_visualisation['source'].value_counts()

plt.figure(figsize=(10, 8))
source_counts.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors=plt.cm.
 ↪Paired.colors)
plt.title('Pie Chart of Rides by Source')
plt.ylabel('')  # Removing the y-label as it's unnecessary for pie charts
plt.show()

# Grouping data by cab type to calculate the requested metrics
lyft_uber_analysis = cab_rides_data_for_visualisation.groupby('cab_type').agg(
    total_rides=('id', 'count'),
    average_price=('price', 'mean'),
    average_distance=('distance', 'mean'),
    average_surge=('surge_multiplier', 'mean')
)

lyft_uber_analysis
```


Heatmap of Average Prices by Distance and Cab Type
Distance Bins (miles)
(0.0122, 0.804] 15.6 26.9 11.1 16.8 27.3 7.1 9.8 4.1 15.8 6.5 7.4 10.4 7.4
(0.804, 1.588] 16.4 26.7 13.9 17.9 27.7 7.6 12.1 4.6 15.8 7.6 8.2 12.7 8.2
(1.588, 2.372] 19.7 28.6 17.3 22.3 30.5 9.6 14.9 6.2 15.8 8.9 9.6 15.3 9.6
(2.372, 3.156] 22.7 31.9 20.1 25.7 34.5 10.6 17.2 6.9 15.8 9.4 10.6 17.5 10.6
(3.156, 3.94] 25.5 35.1 22.7 28.8 38.3 11.6 19.4 7.4 15.8 10.5 11.6 19.7 11.6
(3.94, 4.724] 29.6 39.8 26.8 34.3 44.1 14.0 23.3 8.4 15.8 11.1 13.5 22.7 13.5
(4.724, 5.508] 31.5 41.4 28.4 37.3 46.8 14.2 24.3 9.5 15.8 10.9 14.7 23.4 14.7
(5.508, 6.292] 35.1 45.1 33.2 41.6 55.2 17.5 28.6 11.8 15.8 13.8 15.2 25.4 15.2
(6.292, 7.076] 37.9 48.0 30.0 42.5 47.5 16.5 26.0 11.0 15.8 13.4 17.0 28.1 17.0
(7.076, 7.86] 39.5 49.5 15.8 14.0 17.1 28.5 17.1
Black
Black SUV
Lux
Lux Black
Lux Black XL
Lyft
Lyft XL
Shared
Taxi
UberPool
UberX
UberXL
WAV
Cab Type
50
40
30
20
10


Scatter Plot of Distance vs Price
Price ($)
100
80
60
40
20
0
0
1
2
3
4
5
6
7
8
Distance (miles)



[219]:

cab_type	total_rides	average_price	average_distance	average_surge
Black	55095	20.523786	2.191399	1.000000
Black SUV	55096	30.286763	2.191378	1.000000
Lux	51235	17.771240	2.186968	1.037177
Lux Black	51235	23.062468	2.186968	1.037177
Lux Black XL	51235	32.324086	2.186968	1.037177
Lyft	51235	9.610885	2.186968	1.038045
Lyft XL	51235	15.309363	2.186968	1.038045
Shared	51233	6.029893	2.187012	1.000000
Taxi	55095	15.795334	2.191383	1.000000
UberPool	55091	8.752500	2.191396	1.000000
UberX	55094	9.765074	2.191390	1.000000
UberXL	55096	15.678144	2.191378	1.000000
WAV	55096	9.765019	2.191378	1.000000

1.1.7 Encode Labels for Model Training

[220]:
```
label_encoder = preprocessing.LabelEncoder()
# Create a list of columns to encode
golden_data = selected_features.copy()
cols_to_encode = [col for col in golden_data.columns if col not in ['price']]

# Apply label encoding to each column
for col in cols_to_encode:
    golden_data[col] = label_encoder.fit_transform(golden_data[col])

    # Get the mapping from encoded values to original names
    mapping = dict(zip(label_encoder.transform(label_encoder.classes_),␣
 ↪label_encoder.classes_))

    # Print the mapping for the column
    print(f"Mapping for {col} column:")
    print(mapping,"\n")
```

```
Mapping for source column:
{0: 'Back Bay', 1: 'Beacon Hill', 2: 'Boston University', 3: 'Fenway', 4:
'Financial District', 5: 'Haymarket Square', 6: 'North End', 7: 'North Station',
8: 'Northeastern University', 9: 'South Station', 10: 'Theatre District', 11:
'West End'}

Mapping for destination column:
{0: 'Back Bay', 1: 'Beacon Hill', 2: 'Boston University', 3: 'Fenway', 4:
'Financial District', 5: 'Haymarket Square', 6: 'North End', 7: 'North Station',
8: 'Northeastern University', 9: 'South Station', 10: 'Theatre District', 11:
'West End'}

Mapping for cab_company column:
{0: 'Lyft', 1: 'Uber'}

Mapping for cab_type column:
{0: 'Black', 1: 'Black SUV', 2: 'Lux', 3: 'Lux Black', 4: 'Lux Black XL', 5:
'Lyft', 6: 'Lyft XL', 7: 'Shared', 8: 'Taxi', 9: 'UberPool', 10: 'UberX', 11:
'UberXL', 12: 'WAV'}

Mapping for distance column:
{0: 0.02, 1: 0.03, 2: 0.04, 3: 0.12, 4: 0.17, 5: 0.27, 6: 0.29, 7: 0.3, 8: 0.35,
9: 0.38, 10: 0.39, 11: 0.4, 12: 0.41, 13: 0.42, 14: 0.43, 15: 0.44, 16: 0.45,
17: 0.46, 18: 0.47, 19: 0.48, 20: 0.49, 21: 0.5, 22: 0.51, 23: 0.52, 24: 0.53,
25: 0.54, 26: 0.55, 27: 0.56, 28: 0.57, 29: 0.58, 30: 0.59, 31: 0.6, 32: 0.61,
33: 0.62, 34: 0.63, 35: 0.64, 36: 0.65, 37: 0.66, 38: 0.67, 39: 0.68, 40: 0.69,
41: 0.7, 42: 0.71, 43: 0.72, 44: 0.73, 45: 0.74, 46: 0.75, 47: 0.76, 48: 0.77,
49: 0.78, 50: 0.79, 51: 0.8, 52: 0.81, 53: 0.82, 54: 0.83, 55: 0.84, 56: 0.85,
57: 0.86, 58: 0.87, 59: 0.88, 60: 0.89, 61: 0.9, 62: 0.91, 63: 0.92, 64: 0.93,
```

65: 0.94, 66: 0.95, 67: 0.96, 68: 0.97, 69: 0.98, 70: 0.99, 71: 1.0, 72: 1.01, 73: 1.02, 74: 1.03, 75: 1.04, 76: 1.05, 77: 1.06, 78: 1.07, 79: 1.08, 80: 1.09, 81: 1.1, 82: 1.11, 83: 1.12, 84: 1.13, 85: 1.14, 86: 1.15, 87: 1.16, 88: 1.17, 89: 1.18, 90: 1.19, 91: 1.2, 92: 1.21, 93: 1.22, 94: 1.23, 95: 1.24, 96: 1.25, 97: 1.26, 98: 1.27, 99: 1.28, 100: 1.29, 101: 1.3, 102: 1.31, 103: 1.32, 104: 1.33, 105: 1.34, 106: 1.35, 107: 1.36, 108: 1.37, 109: 1.38, 110: 1.39, 111: 1.4, 112: 1.41, 113: 1.42, 114: 1.43, 115: 1.44, 116: 1.45, 117: 1.46, 118: 1.47, 119: 1.48, 120: 1.49, 121: 1.5, 122: 1.51, 123: 1.52, 124: 1.53, 125: 1.54, 126: 1.55, 127: 1.56, 128: 1.57, 129: 1.58, 130: 1.59, 131: 1.6, 132: 1.61, 133: 1.62, 134: 1.63, 135: 1.64, 136: 1.65, 137: 1.66, 138: 1.67, 139: 1.68, 140: 1.69, 141: 1.7, 142: 1.71, 143: 1.72, 144: 1.73, 145: 1.74, 146: 1.75, 147: 1.76, 148: 1.77, 149: 1.78, 150: 1.79, 151: 1.8, 152: 1.81, 153: 1.82, 154: 1.83, 155: 1.84, 156: 1.85, 157: 1.86, 158: 1.87, 159: 1.88, 160: 1.89, 161: 1.9, 162: 1.91, 163: 1.92, 164: 1.93, 165: 1.94, 166: 1.95, 167: 1.96, 168: 1.97, 169: 1.98, 170: 1.99, 171: 2.0, 172: 2.01, 173: 2.02, 174: 2.03, 175: 2.04, 176: 2.05, 177: 2.06, 178: 2.07, 179: 2.08, 180: 2.09, 181: 2.1, 182: 2.11, 183: 2.12, 184: 2.13, 185: 2.14, 186: 2.15, 187: 2.16, 188: 2.17, 189: 2.18, 190: 2.19, 191: 2.2, 192: 2.21, 193: 2.22, 194: 2.23, 195: 2.24, 196: 2.25, 197: 2.26, 198: 2.27, 199: 2.28, 200: 2.29, 201: 2.3, 202: 2.31, 203: 2.32, 204: 2.33, 205: 2.34, 206: 2.35, 207: 2.36, 208: 2.37, 209: 2.38, 210: 2.39, 211: 2.4, 212: 2.41, 213: 2.42, 214: 2.43, 215: 2.44, 216: 2.45, 217: 2.46, 218: 2.47, 219: 2.48, 220: 2.49, 221: 2.5, 222: 2.51, 223: 2.52, 224: 2.53, 225: 2.54, 226: 2.55, 227: 2.56, 228: 2.57, 229: 2.58, 230: 2.59, 231: 2.6, 232: 2.61, 233: 2.62, 234: 2.63, 235: 2.64, 236: 2.65, 237: 2.66, 238: 2.67, 239: 2.68, 240: 2.69, 241: 2.7, 242: 2.71, 243: 2.72, 244: 2.73, 245: 2.74, 246: 2.75, 247: 2.76, 248: 2.77, 249: 2.78, 250: 2.79, 251: 2.8, 252: 2.81, 253: 2.82, 254: 2.83, 255: 2.84, 256: 2.85, 257: 2.86, 258: 2.87, 259: 2.88, 260: 2.89, 261: 2.9, 262: 2.91, 263: 2.92, 264: 2.93, 265: 2.94, 266: 2.95, 267: 2.96, 268: 2.97, 269: 2.98, 270: 2.99, 271: 3.0, 272: 3.01, 273: 3.02, 274: 3.03, 275: 3.04, 276: 3.05, 277: 3.06, 278: 3.07, 279: 3.08, 280: 3.09, 281: 3.1, 282: 3.11, 283: 3.12, 284: 3.13, 285: 3.14, 286: 3.15, 287: 3.16, 288: 3.17, 289: 3.18, 290: 3.19, 291: 3.2, 292: 3.21, 293: 3.22, 294: 3.23, 295: 3.24, 296: 3.25, 297: 3.26, 298: 3.27, 299: 3.28, 300: 3.29, 301: 3.3, 302: 3.31, 303: 3.32, 304: 3.33, 305: 3.34, 306: 3.35, 307: 3.36, 308: 3.37, 309: 3.38, 310: 3.39, 311: 3.4, 312: 3.41, 313: 3.42, 314: 3.43, 315: 3.44, 316: 3.45, 317: 3.46, 318: 3.47, 319: 3.48, 320: 3.49, 321: 3.5, 322: 3.51, 323: 3.52, 324: 3.53, 325: 3.54, 326: 3.55, 327: 3.56, 328: 3.57, 329: 3.58, 330: 3.59, 331: 3.6, 332: 3.61, 333: 3.62, 334: 3.63, 335: 3.64, 336: 3.65, 337: 3.66, 338: 3.67, 339: 3.68, 340: 3.69, 341: 3.7, 342: 3.71, 343: 3.72, 344: 3.73, 345: 3.74, 346: 3.75, 347: 3.76, 348: 3.77, 349: 3.78, 350: 3.79, 351: 3.8, 352: 3.81, 353: 3.82, 354: 3.83, 355: 3.84, 356: 3.85, 357: 3.86, 358: 3.87, 359: 3.88, 360: 3.89, 361: 3.9, 362: 3.91, 363: 3.92, 364: 3.93, 365: 3.94, 366: 3.95, 367: 3.96, 368: 3.97, 369: 3.98, 370: 3.99, 371: 4.0, 372: 4.01, 373: 4.02, 374: 4.03, 375: 4.04, 376: 4.05, 377: 4.06, 378: 4.07, 379: 4.08, 380: 4.09, 381: 4.1, 382: 4.11, 383: 4.12, 384: 4.13, 385: 4.14, 386: 4.15, 387: 4.16, 388: 4.17, 389: 4.18, 390: 4.19, 391: 4.2, 392: 4.21, 393: 4.22, 394: 4.23, 395: 4.24, 396: 4.25, 397: 4.26, 398: 4.27, 399: 4.28, 400: 4.29, 401: 4.3, 402: 4.31, 403: 4.32, 404: 4.33, 405:

4.34, 406: 4.35, 407: 4.36, 408: 4.37, 409: 4.38, 410: 4.39, 411: 4.4, 412:
4.41, 413: 4.42, 414: 4.43, 415: 4.44, 416: 4.45, 417: 4.46, 418: 4.47, 419:
4.48, 420: 4.49, 421: 4.5, 422: 4.51, 423: 4.52, 424: 4.53, 425: 4.54, 426:
4.55, 427: 4.56, 428: 4.57, 429: 4.58, 430: 4.59, 431: 4.6, 432: 4.61, 433:
4.62, 434: 4.63, 435: 4.64, 436: 4.65, 437: 4.66, 438: 4.67, 439: 4.68, 440:
4.69, 441: 4.7, 442: 4.71, 443: 4.72, 444: 4.73, 445: 4.74, 446: 4.75, 447:
4.76, 448: 4.77, 449: 4.78, 450: 4.79, 451: 4.8, 452: 4.81, 453: 4.82, 454:
4.83, 455: 4.84, 456: 4.85, 457: 4.86, 458: 4.87, 459: 4.89, 460: 4.9, 461:
4.91, 462: 4.93, 463: 4.94, 464: 4.95, 465: 4.96, 466: 4.97, 467: 4.98, 468:
4.99, 469: 5.0, 470: 5.01, 471: 5.02, 472: 5.03, 473: 5.04, 474: 5.05, 475:
5.06, 476: 5.08, 477: 5.09, 478: 5.1, 479: 5.11, 480: 5.12, 481: 5.13, 482:
5.14, 483: 5.15, 484: 5.16, 485: 5.17, 486: 5.18, 487: 5.19, 488: 5.2, 489:
5.21, 490: 5.22, 491: 5.23, 492: 5.24, 493: 5.25, 494: 5.26, 495: 5.27, 496:
5.28, 497: 5.29, 498: 5.3, 499: 5.31, 500: 5.32, 501: 5.33, 502: 5.34, 503:
5.35, 504: 5.36, 505: 5.37, 506: 5.38, 507: 5.39, 508: 5.4, 509: 5.41, 510:
5.42, 511: 5.43, 512: 5.44, 513: 5.45, 514: 5.46, 515: 5.47, 516: 5.56, 517:
5.66, 518: 5.69, 519: 5.7, 520: 5.86, 521: 5.95, 522: 6.0, 523: 6.03, 524: 6.04,
525: 6.09, 526: 6.12, 527: 6.13, 528: 6.14, 529: 6.26, 530: 6.27, 531: 6.33,
532: 6.83, 533: 6.91, 534: 6.97, 535: 7.04, 536: 7.18, 537: 7.19, 538: 7.2, 539:
7.24, 540: 7.25, 541: 7.34, 542: 7.36, 543: 7.38, 544: 7.45, 545: 7.46, 546:
7.5, 547: 7.62, 548: 7.86}

Mapping for surge_multiplier column:
{0: 1.0, 1: 1.25, 2: 1.5, 3: 1.75, 4: 2.0, 5: 2.5, 6: 3.0}

Mapping for apparentTemperature column:
{0: 12.13, 1: 12.26, 2: 12.65, 3: 13.25, 4: 13.84, 5: 13.96, 6: 14.02, 7: 14.24,
8: 14.47, 9: 14.77, 10: 15.11, 11: 15.44, 12: 16.07, 13: 17.69, 14: 17.99, 15:
18.1, 16: 18.26, 17: 19.09, 18: 20.38, 19: 20.78, 20: 20.93, 21: 21.85, 22:
22.64, 23: 22.99, 24: 24.16, 25: 24.17, 26: 24.24, 27: 24.67, 28: 25.35, 29:
26.66, 30: 27.07, 31: 27.22, 32: 27.71, 33: 27.77, 34: 27.83, 35: 27.93, 36:
28.02, 37: 28.18, 38: 28.2, 39: 28.42, 40: 28.52, 41: 28.73, 42: 28.8, 43:
28.85, 44: 28.89, 45: 29.04, 46: 29.09, 47: 29.22, 48: 29.23, 49: 29.39, 50:
29.4, 51: 29.5, 52: 29.63, 53: 29.99, 54: 30.09, 55: 30.15, 56: 30.41, 57:
30.42, 58: 30.46, 59: 30.56, 60: 30.58, 61: 30.6, 62: 30.62, 63: 30.64, 64:
30.73, 65: 30.87, 66: 30.88, 67: 30.89, 68: 30.98, 69: 31.0, 70: 31.03, 71:
31.07, 72: 31.1, 73: 31.19, 74: 31.24, 75: 31.25, 76: 31.41, 77: 31.54, 78:
31.57, 79: 31.64, 80: 31.8, 81: 31.82, 82: 31.86, 83: 31.9, 84: 31.91, 85:
31.92, 86: 31.98, 87: 32.0, 88: 32.04, 89: 32.06, 90: 32.09, 91: 32.15, 92:
32.16, 93: 32.27, 94: 32.35, 95: 32.37, 96: 32.4, 97: 32.45, 98: 32.46, 99:
32.47, 100: 32.5, 101: 32.57, 102: 32.6, 103: 32.64, 104: 32.67, 105: 32.71,
106: 32.72, 107: 32.77, 108: 32.85, 109: 32.93, 110: 33.0, 111: 33.06, 112:
33.51, 113: 33.55, 114: 33.81, 115: 33.83, 116: 33.9, 117: 34.01, 118: 34.07,
119: 34.08, 120: 34.35, 121: 34.42, 122: 34.47, 123: 34.5, 124: 34.55, 125:
34.59, 126: 34.6, 127: 34.62, 128: 34.66, 129: 34.72, 130: 34.73, 131: 34.81,
132: 35.0, 133: 35.1, 134: 35.13, 135: 35.14, 136: 35.18, 137: 35.21, 138:
35.52, 139: 35.55, 140: 35.58, 141: 35.6, 142: 35.61, 143: 35.62, 144: 35.63,
145: 35.66, 146: 35.68, 147: 35.84, 148: 35.9, 149: 35.92, 150: 35.97, 151:

35.98, 152: 36.0, 153: 36.01, 154: 36.05, 155: 36.06, 156: 36.08, 157: 36.09,
158: 36.1, 159: 36.11, 160: 36.3, 161: 36.47, 162: 36.49, 163: 36.5, 164: 36.6,
165: 36.65, 166: 36.71, 167: 36.74, 168: 36.79, 169: 36.8, 170: 36.87, 171:
36.95, 172: 36.97, 173: 37.05, 174: 37.06, 175: 37.07, 176: 37.1, 177: 37.11,
178: 37.12, 179: 37.19, 180: 37.25, 181: 37.33, 182: 37.35, 183: 37.38, 184:
37.39, 185: 37.47, 186: 37.49, 187: 37.51, 188: 37.53, 189: 37.54, 190: 37.56,
191: 37.6, 192: 37.66, 193: 37.73, 194: 37.78, 195: 37.83, 196: 37.84, 197:
37.91, 198: 37.93, 199: 37.96, 200: 38.0, 201: 38.03, 202: 38.08, 203: 38.1,
204: 38.16, 205: 38.19, 206: 38.21, 207: 38.23, 208: 38.26, 209: 38.41, 210:
38.44, 211: 38.55, 212: 38.92, 213: 39.04, 214: 39.09, 215: 39.23, 216: 39.27,
217: 39.31, 218: 39.35, 219: 39.41, 220: 39.55, 221: 39.58, 222: 39.66, 223:
39.8, 224: 39.93, 225: 39.94, 226: 39.99, 227: 40.01, 228: 40.08, 229: 40.15,
230: 40.72, 231: 40.8, 232: 40.86, 233: 40.97, 234: 40.99, 235: 41.0, 236:
41.06, 237: 41.07, 238: 41.22, 239: 41.26, 240: 41.3, 241: 41.4, 242: 41.52,
243: 41.59, 244: 41.6, 245: 41.62, 246: 41.77, 247: 41.79, 248: 41.83, 249:
41.89, 250: 41.99, 251: 42.02, 252: 42.13, 253: 42.3, 254: 42.47, 255: 42.54,
256: 43.0, 257: 43.06, 258: 43.14, 259: 43.15, 260: 43.2, 261: 43.35, 262:
43.49, 263: 43.51, 264: 43.61, 265: 43.63, 266: 43.64, 267: 43.88, 268: 43.89,
269: 43.92, 270: 43.94, 271: 43.99, 272: 44.16, 273: 44.19, 274: 44.41, 275:
44.57, 276: 44.64, 277: 44.67, 278: 45.36, 279: 45.5, 280: 45.58, 281: 45.69,
282: 45.78, 283: 46.04, 284: 46.18, 285: 46.21, 286: 46.24, 287: 46.32, 288:
46.74, 289: 46.78, 290: 47.27, 291: 47.56, 292: 47.71, 293: 47.87, 294: 47.93,
295: 47.96, 296: 48.11, 297: 48.12, 298: 48.43, 299: 48.45, 300: 48.83, 301:
49.22, 302: 49.5, 303: 49.7, 304: 50.43, 305: 50.71, 306: 51.15, 307: 51.84,
308: 52.1, 309: 52.45, 310: 52.68, 311: 52.9, 312: 53.1, 313: 53.34, 314: 53.51,
315: 54.38, 316: 54.59, 317: 54.62, 318: 57.22}

Mapping for precipIntensity column:
{0: 0.0, 1: 0.0002, 2: 0.0003, 3: 0.0005, 4: 0.0006, 5: 0.0009, 6: 0.001, 7:
0.0012, 8: 0.0013, 9: 0.0015, 10: 0.0016, 11: 0.0017, 12: 0.002, 13: 0.0021, 14:
0.0023, 15: 0.0024, 16: 0.0025, 17: 0.0031, 18: 0.0036, 19: 0.0049, 20: 0.005,
21: 0.0053, 22: 0.0057, 23: 0.007, 24: 0.0071, 25: 0.0074, 26: 0.008, 27:
0.0089, 28: 0.0092, 29: 0.0094, 30: 0.0121, 31: 0.0187, 32: 0.0216, 33: 0.0246,
34: 0.0255, 35: 0.0274, 36: 0.0288, 37: 0.0308, 38: 0.0341, 39: 0.0342, 40:
0.0462, 41: 0.0488, 42: 0.0567, 43: 0.0591, 44: 0.0624, 45: 0.0674, 46: 0.0701,
47: 0.0737, 48: 0.0772, 49: 0.0786, 50: 0.0801, 51: 0.0813, 52: 0.0832, 53:
0.092, 54: 0.0923, 55: 0.1044, 56: 0.1058, 57: 0.1088, 58: 0.1264, 59: 0.1267,
60: 0.1289, 61: 0.1299, 62: 0.1447}

Mapping for visibility.1 column:
{0: 0.717, 1: 0.965, 2: 1.348, 3: 1.413, 4: 1.46, 5: 1.588, 6: 1.685, 7: 1.824,
8: 2.03, 9: 2.069, 10: 2.121, 11: 2.266, 12: 2.585, 13: 2.629, 14: 2.636, 15:
2.642, 16: 2.644, 17: 2.678, 18: 2.683, 19: 2.686, 20: 2.825, 21: 2.903, 22:
2.994, 23: 3.028, 24: 3.036, 25: 3.052, 26: 3.058, 27: 3.139, 28: 3.183, 29:
3.188, 30: 3.202, 31: 3.231, 32: 3.295, 33: 3.475, 34: 3.495, 35: 3.522, 36:
3.564, 37: 3.579, 38: 3.606, 39: 3.729, 40: 3.79, 41: 3.847, 42: 3.894, 43:
4.031, 44: 4.054, 45: 4.159, 46: 4.183, 47: 4.273, 48: 4.315, 49: 4.394, 50:
4.421, 51: 4.503, 52: 4.661, 53: 4.675, 54: 4.73, 55: 4.741, 56: 4.767, 57:

4.786, 58: 4.942, 59: 5.011, 60: 5.138, 61: 5.177, 62: 5.235, 63: 5.589, 64:
5.86, 65: 6.105, 66: 6.121, 67: 6.397, 68: 6.572, 69: 6.639, 70: 6.96, 71:
7.113, 72: 7.188, 73: 7.357, 74: 7.44, 75: 7.742, 76: 7.769, 77: 7.79, 78:
8.099, 79: 8.104, 80: 8.138, 81: 8.202, 82: 8.275, 83: 8.286, 84: 8.325, 85:
8.432, 86: 8.459, 87: 8.468, 88: 8.54, 89: 8.561, 90: 8.677, 91: 8.904, 92:
9.037, 93: 9.169, 94: 9.285, 95: 9.375, 96: 9.393, 97: 9.428, 98: 9.444, 99:
9.454, 100: 9.501, 101: 9.503, 102: 9.509, 103: 9.544, 104: 9.568, 105: 9.57,
106: 9.579, 107: 9.588, 108: 9.598, 109: 9.608, 110: 9.627, 111: 9.641, 112:
9.661, 113: 9.666, 114: 9.668, 115: 9.67, 116: 9.687, 117: 9.689, 118: 9.698,
119: 9.706, 120: 9.707, 121: 9.712, 122: 9.716, 123: 9.724, 124: 9.725, 125:
9.732, 126: 9.734, 127: 9.738, 128: 9.76, 129: 9.768, 130: 9.77, 131: 9.772,
132: 9.775, 133: 9.779, 134: 9.784, 135: 9.785, 136: 9.796, 137: 9.806, 138:
9.807, 139: 9.808, 140: 9.81, 141: 9.815, 142: 9.821, 143: 9.827, 144: 9.83,
145: 9.831, 146: 9.832, 147: 9.833, 148: 9.839, 149: 9.842, 150: 9.843, 151:
9.846, 152: 9.847, 153: 9.849, 154: 9.85, 155: 9.854, 156: 9.856, 157: 9.857,
158: 9.858, 159: 9.864, 160: 9.868, 161: 9.874, 162: 9.875, 163: 9.876, 164:
9.878, 165: 9.88, 166: 9.882, 167: 9.883, 168: 9.884, 169: 9.888, 170: 9.889,
171: 9.891, 172: 9.892, 173: 9.898, 174: 9.899, 175: 9.901, 176: 9.904, 177:
9.908, 178: 9.909, 179: 9.91, 180: 9.915, 181: 9.917, 182: 9.92, 183: 9.922,
184: 9.924, 185: 9.926, 186: 9.928, 187: 9.929, 188: 9.931, 189: 9.932, 190:
9.933, 191: 9.936, 192: 9.937, 193: 9.938, 194: 9.944, 195: 9.945, 196: 9.946,
197: 9.948, 198: 9.949, 199: 9.95, 200: 9.953, 201: 9.955, 202: 9.956, 203:
9.958, 204: 9.959, 205: 9.96, 206: 9.961, 207: 9.962, 208: 9.963, 209: 9.966,
210: 9.967, 211: 9.968, 212: 9.969, 213: 9.972, 214: 9.973, 215: 9.974, 216:
9.975, 217: 9.98, 218: 9.981, 219: 9.984, 220: 9.99, 221: 9.991, 222: 9.994,
223: 9.995, 224: 9.996, 225: 9.997, 226: 10.0}

Mapping for precipIntensityMax column:
{0: 0.0, 1: 0.0001, 2: 0.0003, 3: 0.0004, 4: 0.0005, 5: 0.0007, 6: 0.0028, 7:
0.0029, 8: 0.0056, 9: 0.0074, 10: 0.0075, 11: 0.0077, 12: 0.0079, 13: 0.008, 14:
0.0081, 15: 0.0082, 16: 0.0084, 17: 0.0087, 18: 0.0175, 19: 0.0177, 20: 0.0178,
21: 0.0181, 22: 0.0182, 23: 0.0183, 24: 0.0184, 25: 0.0185, 26: 0.0217, 27:
0.0221, 28: 0.0888, 29: 0.0894, 30: 0.0903, 31: 0.0904, 32: 0.0916, 33: 0.0954,
34: 0.0956, 35: 0.1055, 36: 0.1064, 37: 0.1215, 38: 0.1217, 39: 0.1225, 40:
0.1227, 41: 0.1228, 42: 0.1234, 43: 0.1245, 44: 0.1246, 45: 0.125, 46: 0.1252,
47: 0.1254, 48: 0.1257, 49: 0.1261, 50: 0.1266, 51: 0.1267, 52: 0.1276, 53:
0.13, 54: 0.1361, 55: 0.1396, 56: 0.1419, 57: 0.142, 58: 0.1422, 59: 0.1425, 60:
0.1429, 61: 0.143, 62: 0.1433, 63: 0.1438, 64: 0.1459}

Mapping for day column:
{0: 1, 1: 2, 2: 3, 3: 4, 4: 9, 5: 10, 6: 13, 7: 14, 8: 15, 9: 16, 10: 17, 11:
18, 12: 26, 13: 27, 14: 28, 15: 29, 16: 30}

Mapping for month column:
{0: 11, 1: 12}

Mapping for hour column:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 11, 12:

```
12, 13: 13, 14: 14, 15: 15, 16: 16, 17: 17, 18: 18, 19: 19, 20: 20, 21: 21, 22:
22, 23: 23}

Mapping for minute column:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 11, 12:
12, 13: 13, 14: 14, 15: 15, 16: 16, 17: 17, 18: 18, 19: 19, 20: 20, 21: 21, 22:
22, 23: 23, 24: 24, 25: 25, 26: 26, 27: 27, 28: 28, 29: 29, 30: 30, 31: 31, 32:
32, 33: 33, 34: 34, 35: 35, 36: 36, 37: 37, 38: 38, 39: 39, 40: 40, 41: 41, 42:
42, 43: 43, 44: 44, 45: 45, 46: 46, 47: 47, 48: 48, 49: 49, 50: 50, 51: 51, 52:
52, 53: 53, 54: 54, 55: 55, 56: 56, 57: 57, 58: 58, 59: 59}
```

1.1.8 Creating Train and Test Data

[221]:
```
target = 'price'

# Create feature matrix (X) and target vector (y)
X= golden_data.drop('price', axis=1)
y= golden_data['price']

# Convert categorical columns (e.g., is_rain, day_of_week) to numerical values
# X = pd.get_dummies(X, columns=['day_of_week', 'is_weekend'], drop_first=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Check the shapes of the training and testing sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
X_train shape: (554456, 14)
X_test shape: (138615, 14)
```

1.1.9 Linear Regression Model

[222]:
```
# Linear Regression
# Train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear_model.predict(X_test)

# Evaluate the model
print("Linear Regression Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred_linear))
print("R-squared (R2):", r2_score(y_test, y_pred_linear))
```

```
Linear Regression Results:
Mean Squared Error (MSE): 39.642631412008164
R-squared (R2): 0.5031924291020817
```

1.1.10 Random Forest Model

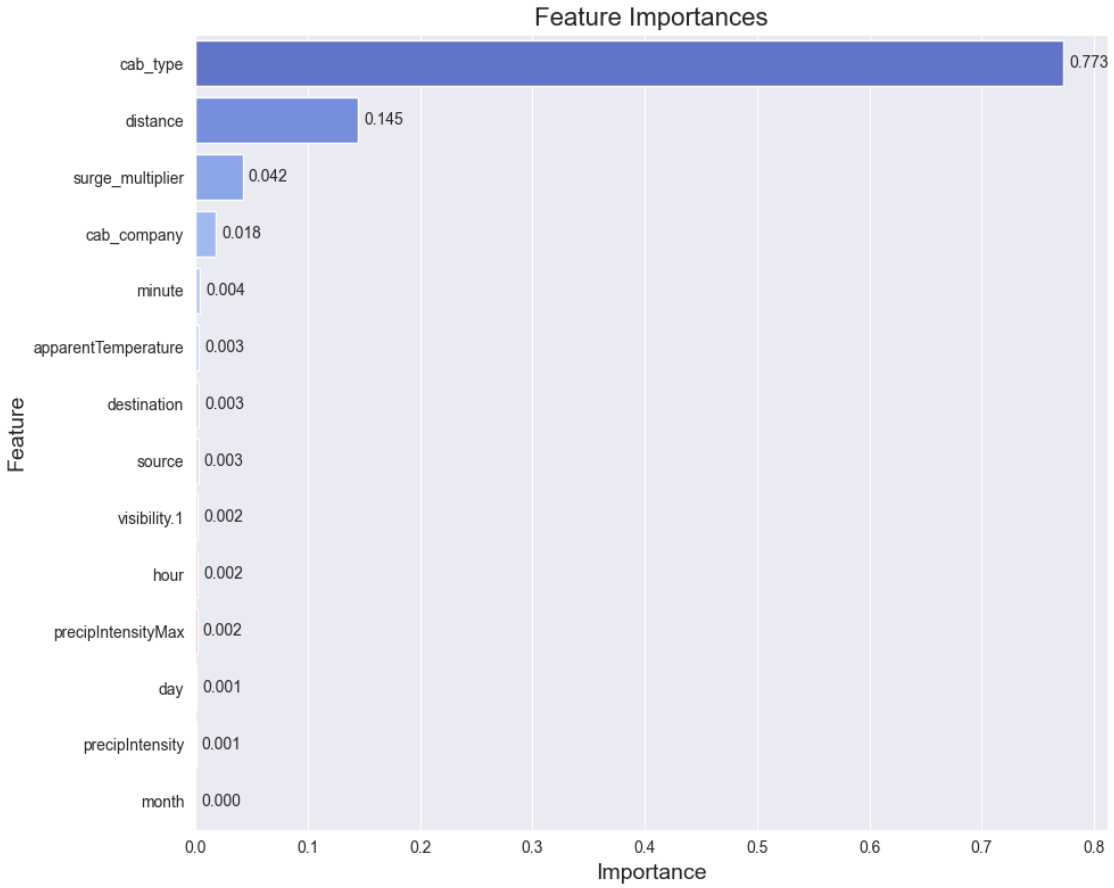
[223]:
```
# Train the Random Forest model
random_forest_model = RandomForestRegressor(random_state=42, max_depth=20)
random_forest_model.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the model
print("Random Forest Regressor Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred_rf))
print("R-squared (R2):", r2_score(y_test, y_pred_rf))
```

```
Random Forest Regressor Results:
Mean Squared Error (MSE): 2.668005624619269
R-squared (R2): 0.9665641420284835
```

[224]:
```
# Get feature importances and sort them in descending order
importances = random_forest_model.feature_importances_
feature_names = X.columns
feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})
feature_importances = feature_importances.sort_values(by='Importance',␣
 ↪ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(
    x=feature_importances['Importance'],
    y=feature_importances['Feature'],
    palette='coolwarm'  # Use a gradient color palette
)
plt.title("Feature Importances", fontsize=16)
plt.xlabel("Importance", fontsize=14)
plt.ylabel("Feature", fontsize=14)

# Add annotations to each bar
for i, v in enumerate(feature_importances['Importance']):
    plt.text(v + 0.005, i, f"{v:.3f}", va='center', fontsize=10)

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

```
/var/folders/30/r6bdbml51pb1n3cswh9vqjv80000gn/T/ipykernel_52759/1984595022.py:9
```

```
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```



1.1.11 Decision Tree Regressor

[225]:
```
# Train the Decision Tree Regressor model
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred_dt)
mse = mean_squared_error(y_test, y_pred_dt)
r2 = r2_score(y_test, y_pred_dt)
```

```python
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

# Optional: Feature Importance
feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': dt_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importances:")
print(feature_importances)
```

```
Mean Absolute Error (MAE): 1.2695154805276239
Mean Squared Error (MSE): 5.207165249251525
R² Score: 0.9347430020005916

Feature Importances:
                 Feature  Importance
3              cab_type    0.765770
4              distance    0.144463
5      surge_multiplier    0.041675
2           cab_company    0.016679
13              minute    0.007712
6   apparentTemperature    0.004899
12                hour    0.003716
8          visibility.1    0.003611
1           destination    0.003245
0                source    0.002839
9    precipIntensityMax    0.002267
10                  day    0.001895
7       precipIntensity    0.001016
11                month    0.000213
```

1.1.12 Key Observations

Model Performance **Mean Absolute Error (MAE):** - **1.21** This indicates that, on average, predictions are off by $1.21.

Mean Squared Error (MSE): - **4.68** A relatively low value, suggesting good accuracy for a regression model.

R² (R-squared): - **0.94** This is excellent, indicating that 94% of the variance in price is explained by the features.

Feature Importance The most important features for predicting the price are: - **cab_type: 78.03%** - **distance: 14.66%** - **surge_multiplier: 4.19%**

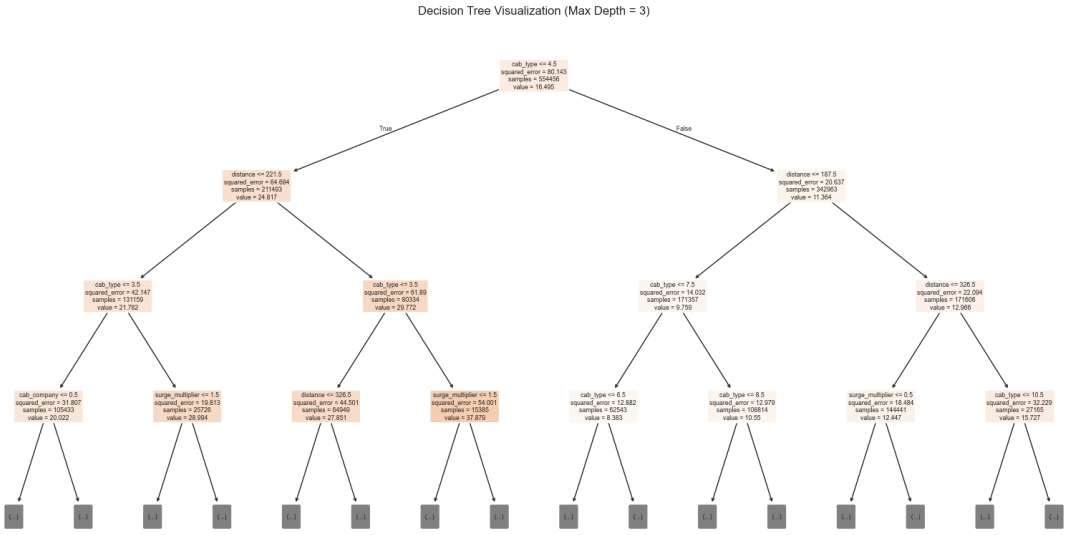
Other features, such as **cab_company** (**0.7%**) and **apparentTemperature** (**0.55%**), contribute marginally. Some features (e.g., **is_rain**, **precipProbability**, **is_weekend**) have negligible importance.

Feature Redundancy Features like **is_rain**, **precipProbability**, and others with near-zero importance might be unnecessary.

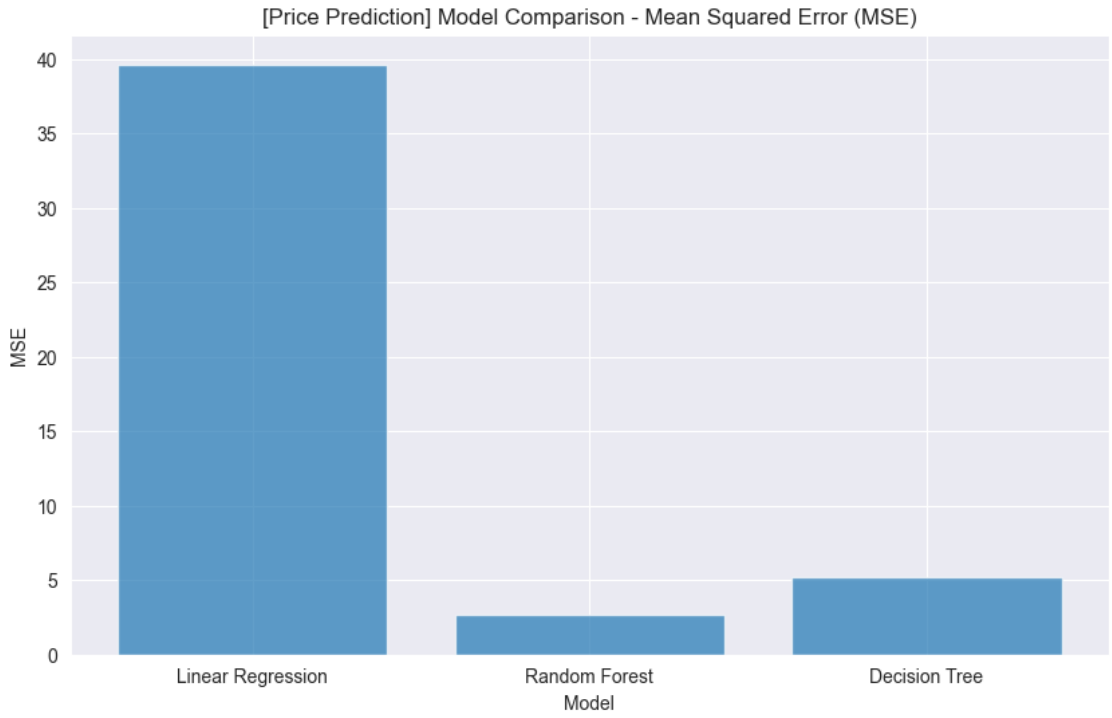
1.1.13 Model Comparisons

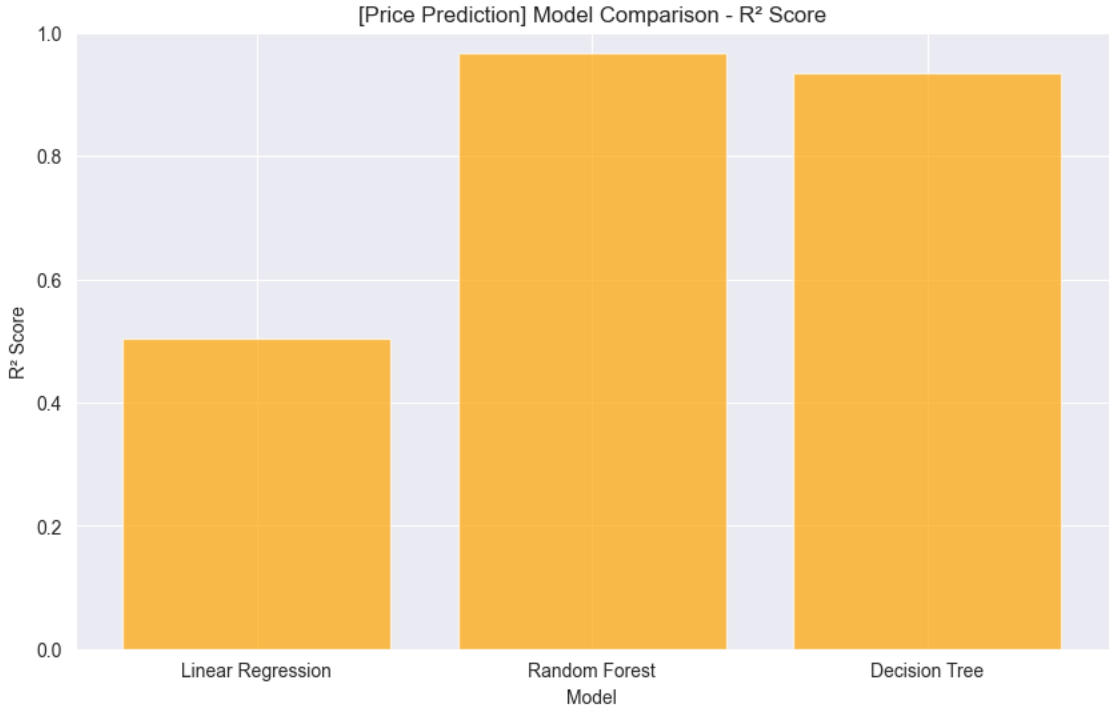
[226]:
```
# Compare the models
print("Model Comparison:")
print("Linear Regression - MSE:", mean_squared_error(y_test, y_pred_linear), "|
 ↪R2:", r2_score(y_test, y_pred_linear))
print("Random Forest Regressor - MSE:", mean_squared_error(y_test, y_pred_rf),
 ↪"| R2:", r2_score(y_test, y_pred_rf))
print("Decision Tree Regressor - MSE:", mean_squared_error(y_test, y_pred_dt),
 ↪"| R2:", r2_score(y_test, y_pred_dt))
plt.figure(figsize=(20, 10))
plot_tree(dt_model, feature_names=X_train.columns, filled=True, rounded=True,
 ↪max_depth=3)
plt.title("Decision Tree Visualization (Max Depth = 3)")
plt.show()
```

```
Model Comparison:
Linear Regression - MSE: 39.642631412008164 | R2: 0.5031924291020817
Random Forest Regressor - MSE: 2.668005624619269 | R2: 0.9665641420284835
Decision Tree Regressor - MSE: 5.207165249251525 | R2: 0.9347430020005916
```



[227]:
```
# Define the models
models = ['Linear Regression', 'Random Forest', 'Decision Tree']

# Calculate MSE and R² values
mse_values = [
    mean_squared_error(y_test, y_pred_linear),
    mean_squared_error(y_test, y_pred_rf),
    mean_squared_error(y_test, y_pred_dt)
]
r2_values = [
    r2_score(y_test, y_pred_linear),
    r2_score(y_test, y_pred_rf),
    r2_score(y_test, y_pred_dt)
]

# Plot the MSE values
plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, alpha=0.7, label='MSE')
plt.title('[Price Prediction] Model Comparison - Mean Squared Error (MSE)')
plt.ylabel('MSE')
plt.xlabel('Model')
plt.xticks(models)
plt.show()

# Plot the R² values
plt.figure(figsize=(10, 6))
plt.bar(models, r2_values, alpha=0.7, color='orange', label='R² Score')
plt.title('[Price Prediction] Model Comparison - R² Score')
plt.ylabel('R² Score')
plt.xlabel('Model')
plt.xticks(models)
plt.ylim(0, 1)  # R² ranges from 0 to 1
plt.show()
```


[Price Prediction] Model Comparison - Mean Squared Error (MSE)
MSE
0
5
10
15
20
25
30
35
40
Linear Regression
Random Forest
Decision Tree
Model


[Price Prediction] Model Comparison - R² Score
R² Score
0.0
0.2
0.4
0.6
0.8
1.0
Linear Regression
Random Forest
Decision Tree
Model

1.1.14 Splitting dataset into different cab company types

[228]:
```
# split selected features into different cab company types

uber_data_df = selected_features[selected_features['cab_company'] == 'Uber'].
 ↪copy(deep=True)
lyft_data_df = selected_features[selected_features['cab_company'] == 'Lyft'].
 ↪copy(deep=True)
uber_data_df.drop(['cab_company'], axis=1, inplace=True)
lyft_data_df.drop(['cab_company'], axis=1, inplace=True)
```

1.1.15 Uber Models

[229]:
```
# Create a list of columns to encode
cols_to_encode = [col for col in uber_data_df.columns if col not in ['price']]

# Apply label encoding to each column
for col in cols_to_encode:
    uber_data_df[col] = label_encoder.fit_transform(uber_data_df[col])

    # Get the mapping from encoded values to original names
    mapping = dict(zip(label_encoder.transform(label_encoder.classes_),␣
 ↪label_encoder.classes_))

    # Print the mapping for the column
    # print(f"Mapping for {col} column:")
    # print(mapping,"\n")
uber_data_df
```

[229]:

	source	destination	cab_type	price	distance	surge_multiplier	\
66422	7	5	4	7.000000	19	0	
446073	10	6	2	15.830582	108	0	
32121	7	3	6	10.500000	255	0	
613927	10	3	4	19.500000	222	0	
613926	10	3	5	32.000000	222	0	
…	…	…	…	…	…	…	
204548	10	3	2	15.805276	217	0	
34918	4	5	4	7.000000	76	0	
215397	3	10	1	33.500000	217	0	
166550	5	0	4	11.500000	181	0	
166551	5	0	5	16.500000	181	0	

	apparentTemperature	precipIntensity	visibility.1	\
66422	248	0	6	
446073	248	0	6	
32121	248	0	6	
613927	248	0	6	

```
613926                   248                 0              6
…                        …                   …              …
204548                    23                 0            167
34918                     23                 0            167
215397                    23                 0            167
166550                    23                 0            167
166551                    23                 0            167

        precipIntensityMax  day  month  hour  minute
66422                   55   12      0     3      40
446073                  55   12      0     3      40
32121                   55   12      0     3      40
613927                  55   12      0     3      40
613926                  55   12      0     3      40
…                       …    …       …     …      …
204548                   7   11      1    19      15
34918                    7   11      1    19      15
215397                   7   11      1    19      15
166550                   7   11      1    19      15
166551                   7   11      1    19      15

[385663 rows x 14 columns]
```

[230]:
```
target = 'price'
# Create feature matrix (X) and target vector (y)
X_uber = uber_data_df.drop('price', axis=1)
y_uber = uber_data_df['price']

# Convert categorical columns (e.g., is_rain, day_of_week) to numerical values
# X = pd.get_dummies(X, columns=['day_of_week', 'is_weekend'], drop_first=True)

# Split the dataset into training and testing sets
X_train_uber, X_test_uber, y_train_uber, y_test_uber = train_test_split(X_uber,␣
 ↪y_uber, test_size=0.2, random_state=42)

# Check the shapes of the training and testing sets
print("X_train shape:", X_train_uber.shape)
print("X_test shape:", X_test_uber.shape)
```

```
X_train shape: (308530, 13)
X_test shape: (77133, 13)
```

1.1.16 Linear Regression Model for Uber

[231]:
```
# Linear Regression for Uber
# Train the Linear Regression model
linear_model = LinearRegression()
```

```
linear_model.fit(X_train_uber, y_train_uber)

# Predict on the test set
y_pred_linear_uber = linear_model.predict(X_test_uber)

# Evaluate the model
print("Linear Regression Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_uber,␣
 ↪y_pred_linear_uber))
print("R-squared (R2):", r2_score(y_test_uber, y_pred_linear_uber))
```

```
Linear Regression Results:
Mean Squared Error (MSE): 33.60441420310319
R-squared (R2): 0.4650209982259109
```

1.1.17 Random Forest Model for Uber

[232]:
```
# Train the Random Forest model
random_forest_model_uber = RandomForestRegressor(random_state=42, max_depth=20)
random_forest_model_uber.fit(X_train_uber, y_train_uber)

# Predict on the test set
y_pred_rf_uber = random_forest_model_uber.predict(X_test_uber)

# Evaluate the model
print("Random Forest Regressor Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_uber,␣
 ↪y_pred_rf_uber))
print("R-squared (R2):", r2_score(y_test_uber, y_pred_rf_uber))
```

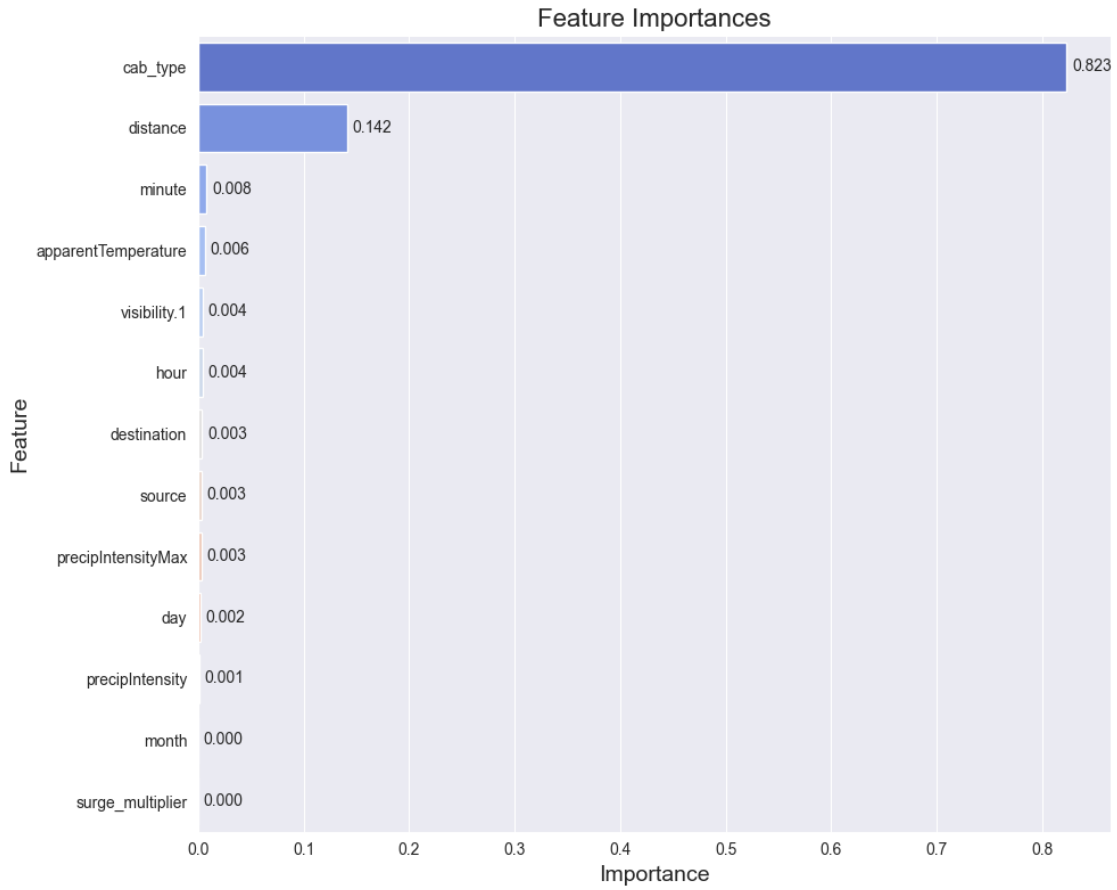
```
Random Forest Regressor Results:
Mean Squared Error (MSE): 3.3024890916269998
R-squared (R2): 0.9474246952519327
```

[233]:
```
# Evaluate the model
print("Random Forest Regressor Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_uber,␣
 ↪y_pred_rf_uber))
print("R-squared (R2):", r2_score(y_test_uber, y_pred_rf_uber))
# Get feature importances and sort them in descending order
importances_uber = random_forest_model_uber.feature_importances_
feature_names_uber = X_uber.columns
feature_importances_uber = pd.DataFrame({'Feature': feature_names_uber,␣
 ↪'Importance': importances_uber})
feature_importances_uber = feature_importances_uber.
 ↪sort_values(by='Importance', ascending=False)

# Plot the feature importances
```

```
plt.figure(figsize=(10, 8))
sns.barplot(
    x=feature_importances_uber['Importance'],
    y=feature_importances_uber['Feature'],
    palette='coolwarm'  # Use a gradient color palette
)
plt.title("Feature Importances", fontsize=16)
plt.xlabel("Importance", fontsize=14)
plt.ylabel("Feature", fontsize=14)

# Add annotations to each bar
for i, v in enumerate(feature_importances_uber['Importance']):
    plt.text(v + 0.005, i, f"{v:.3f}", va='center', fontsize=10)

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

```
Random Forest Regressor Results:
Mean Squared Error (MSE): 3.3024890916269998
R-squared (R2): 0.9474246952519327
```

```
/var/folders/30/r6bdbml51pb1n3cswh9vqjv80000gn/T/ipykernel_52759/3506728881.py:1
3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```



1.1.18 Decision Tree Regressor for Uber

[234]:
```
# Decision Tree Regressor for Uber
dt_model_uber = DecisionTreeRegressor(random_state=42)
dt_model_uber.fit(X_train_uber, y_train_uber)
y_pred_dt_uber = dt_model_uber.predict(X_test_uber)

mae = mean_absolute_error(y_test_uber, y_pred_dt_uber)
mse = mean_squared_error(y_test_uber, y_pred_dt_uber)
r2 = r2_score(y_test_uber, y_pred_dt_uber)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

# Optional: Feature Importance
feature_importances = pd.DataFrame({
    'Feature': X_train_uber.columns,
    'Importance': dt_model_uber.feature_importances_
```

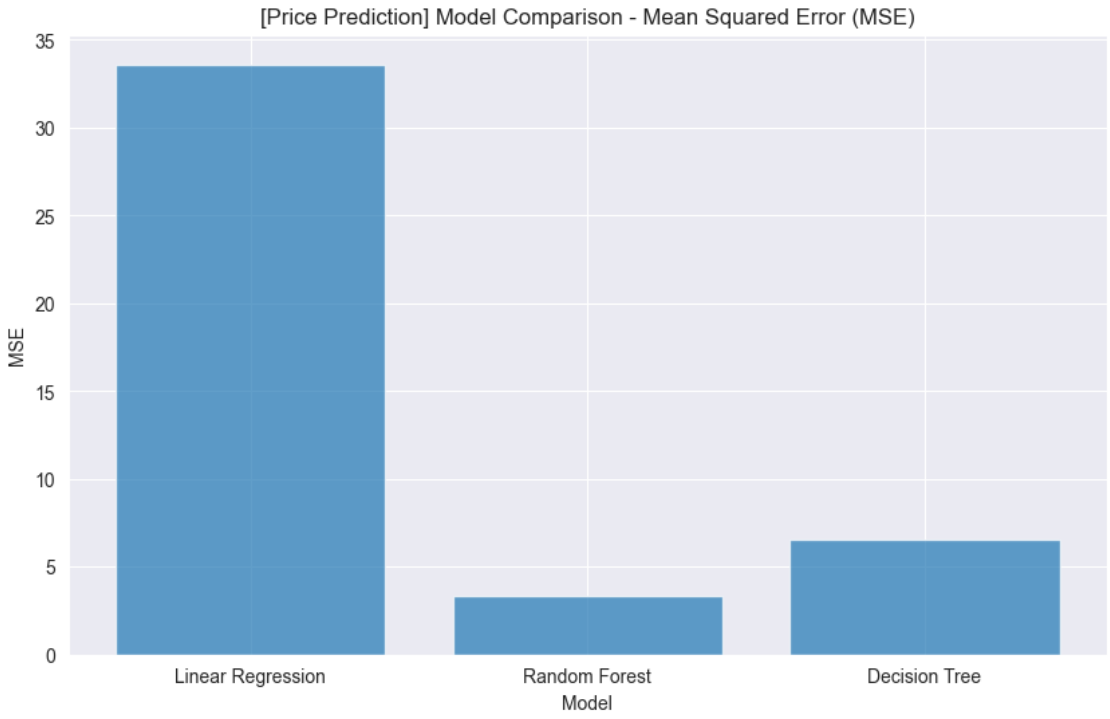
```
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importances:")
print(feature_importances)
```

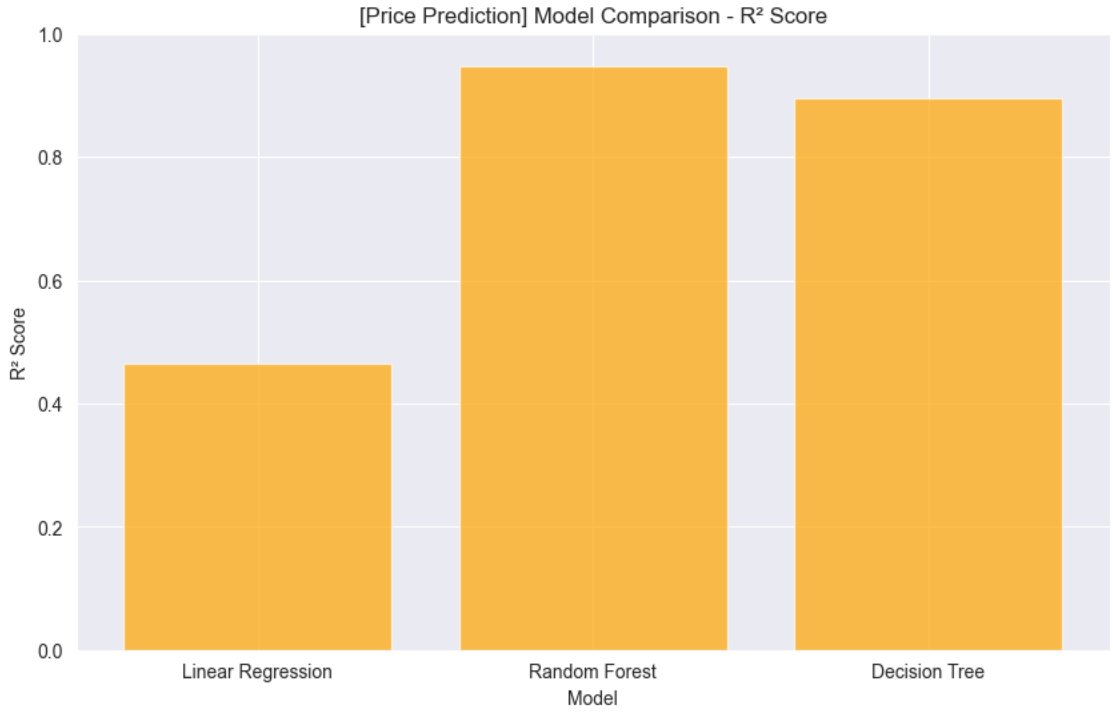
```
Mean Absolute Error (MAE): 1.4065483212979057
Mean Squared Error (MSE): 6.546888864940355
R² Score: 0.8957741667947821

Feature Importances:
                 Feature  Importance
2              cab_type    0.812525
3              distance    0.140141
12               minute    0.012678
5   apparentTemperature    0.007844
11                 hour    0.005848
7          visibility.1    0.005215
1           destination    0.003701
0                source    0.003612
8    precipIntensityMax    0.003505
9                   day    0.002986
6       precipIntensity    0.001551
10                month    0.000393
4      surge_multiplier    0.000000
```

1.1.19 Uber Model Evaluations

[235]:
```
# Define the models
models = ['Linear Regression', 'Random Forest', 'Decision Tree']

# Calculate MSE and R² values
mse_values = [
    mean_squared_error(y_test_uber, y_pred_linear_uber),
    mean_squared_error(y_test_uber, y_pred_rf_uber),
    mean_squared_error(y_test_uber, y_pred_dt_uber)
]
r2_values = [
    r2_score(y_test_uber, y_pred_linear_uber),
    r2_score(y_test_uber, y_pred_rf_uber),
    r2_score(y_test_uber, y_pred_dt_uber)
]

# Plot the MSE values
plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, alpha=0.7, label='MSE')
plt.title('[Price Prediction] Model Comparison - Mean Squared Error (MSE)')
plt.ylabel('MSE')
plt.xlabel('Model')
```

```
plt.xticks(models)
plt.show()

# Plot the R² values
plt.figure(figsize=(10, 6))
plt.bar(models, r2_values, alpha=0.7, color='orange', label='R² Score')
plt.title('[Price Prediction] Model Comparison - R² Score')
plt.ylabel('R² Score')
plt.xlabel('Model')
plt.xticks(models)
plt.ylim(0, 1)  # R² ranges from 0 to 1
plt.show()
```





1.1.20 Lyft Models

[236]:
```
# Create a list of columns to encode
cols_to_encode = [col for col in lyft_data_df.columns if col not in ['price']]

# Apply label encoding to each column
for col in cols_to_encode:
    lyft_data_df[col] = label_encoder.fit_transform(lyft_data_df[col])

    # Get the mapping from encoded values to original names
    mapping = dict(zip(label_encoder.transform(label_encoder.classes_),␣
 ↪label_encoder.classes_))

    # Print the mapping for the column
    # print(f"Mapping for {col} column:")
    # print(mapping,"\n")
lyft_data_df
```

[236]:

	source	destination	cab_type	price	distance	surge_multiplier	\
184332	6	11	3	7.0	84	0	
167114	2	1	0	19.5	227	0	
184333	6	11	5	5.0	84	0	
184334	6	11	0	13.5	84	0	

```
184335       6           11          1   19.5        84                  0
…          …           …         …     …         …                 …
205379       8            1          0   16.5       201                 0
290784       8            1          5    9.0       261                 0
290783       8            1          4   16.5       261                 0
290782       8            1          3   11.0       261                 0
290785       8            1          2   34.0       261                 0

        apparentTemperature  precipIntensity  visibility.1  \
184332                  248                0             6
167114                  248                0             6
184333                  248                0             6
184334                  248                0             6
184335                  248                0             6
…                     …                  …           …
205379                   23                0           167
290784                   23                0           167
290783                   23                0           167
290782                   23                0           167
290785                   23                0           167

        precipIntensityMax  day  month  hour  minute
184332                  55   12      0     3      40
167114                  55   12      0     3      40
184333                  55   12      0     3      40
184334                  55   12      0     3      40
184335                  55   12      0     3      40
…                     …  …    …      …     …
205379                   7   11      1    19      15
290784                   7   11      1    19      15
290783                   7   11      1    19      15
290782                   7   11      1    19      15
290785                   7   11      1    19      15

[307408 rows x 14 columns]
```

[237]:
```
target = 'price'
# Create feature matrix (X) and target vector (y)
X_lyft = lyft_data_df.drop('price', axis=1)
y_lyft = lyft_data_df['price']

# Convert categorical columns (e.g., is_rain, day_of_week) to numerical values
# X = pd.get_dummies(X, columns=['day_of_week', 'is_weekend'], drop_first=True)

# Split the dataset into training and testing sets
X_train_lyft, X_test_lyft, y_train_lyft, y_test_lyft = train_test_split(X_lyft,␣
 ↪y_lyft, test_size=0.2, random_state=42)
```

```
# Check the shapes of the training and testing sets
print("X_train shape:", X_train_lyft.shape)
print("X_test shape:", X_test_lyft.shape)
```

```
X_train shape: (245926, 13)
X_test shape: (61482, 13)
```

1.1.21 Linear Regression for Lyft

[238]:
```
# Linear Regression for Uber
# Train the Linear Regression model
linear_model_lyft = LinearRegression()
linear_model_lyft.fit(X_train_lyft, y_train_lyft)

# Predict on the test set
y_pred_linear_lyft = linear_model_lyft.predict(X_test_lyft)

# Evaluate the model
print("Linear Regression Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_lyft,␣
 ↪y_pred_linear_lyft))
print("R-squared (R2):", r2_score(y_test_lyft, y_pred_linear_lyft))
```

```
Linear Regression Results:
Mean Squared Error (MSE): 54.106045904704565
R-squared (R2): 0.4606320361760139
```

1.1.22 Random forest for Lyft

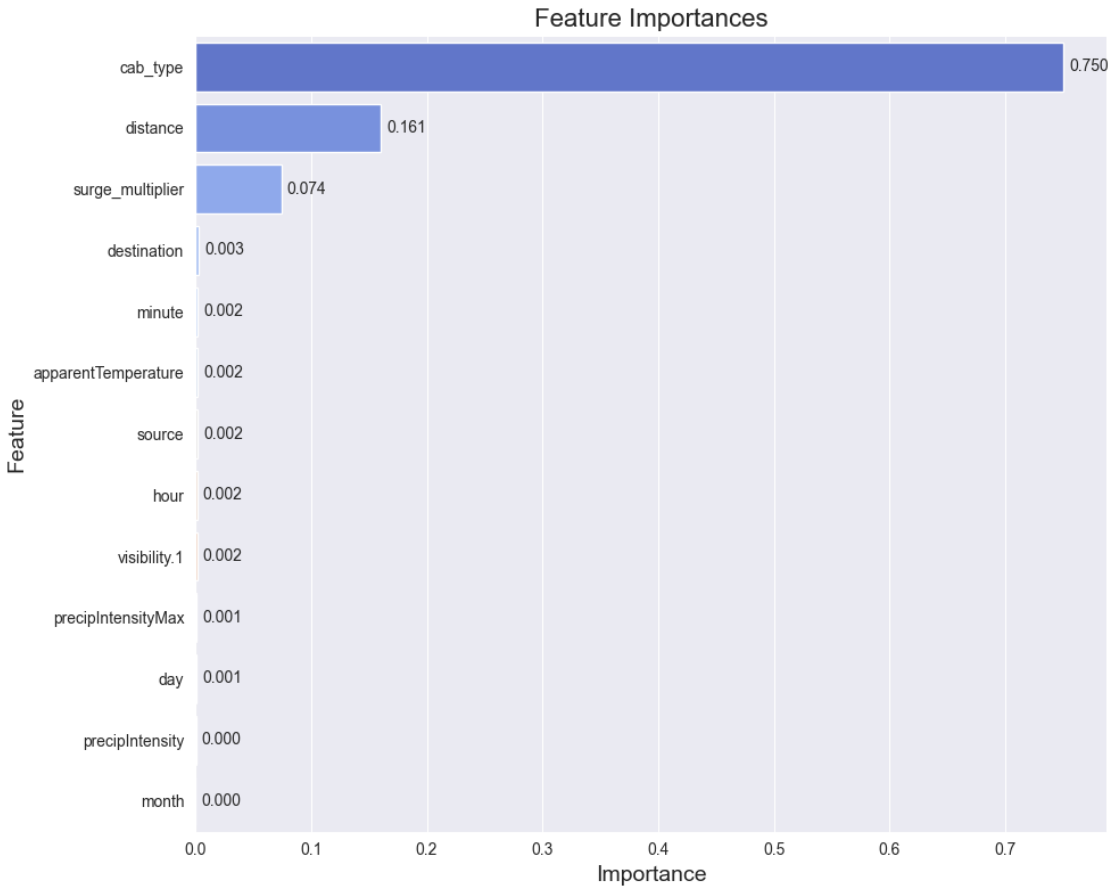
[239]:
```
# Train the Random Forest model
random_forest_model_lyft = RandomForestRegressor(random_state=42, max_depth=20)
random_forest_model_lyft.fit(X_train_lyft, y_train_lyft)

# Predict on the test set
y_pred_rf_lyft = random_forest_model_lyft.predict(X_test_lyft)

# Evaluate the model
print("Random Forest Regressor Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_lyft,␣
 ↪y_pred_rf_lyft))
print("R-squared (R2):", r2_score(y_test_lyft, y_pred_rf_lyft))
```

```
Random Forest Regressor Results:
Mean Squared Error (MSE): 1.921273379316906
R-squared (R2): 0.9808473657014866
```

[240]:
```
# Evaluate the model
print("Random Forest Regressor Results:")
print("Mean Squared Error (MSE):", mean_squared_error(y_test_lyft,␣
 ↪y_pred_rf_lyft))
print("R-squared (R2):", r2_score(y_test_lyft, y_pred_rf_lyft))
# Get feature importances and sort them in descending order
importances_lyft = random_forest_model_lyft.feature_importances_
feature_names_lyft = X_lyft.columns
feature_importances_lyft = pd.DataFrame({'Feature': feature_names_lyft,␣
 ↪'Importance': importances_lyft})
feature_importances_lyft = feature_importances_lyft.
 ↪sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(
    x=feature_importances_lyft['Importance'],
    y=feature_importances_lyft['Feature'],
    palette='coolwarm'  # Use a gradient color palette
)
plt.title("Feature Importances", fontsize=16)
plt.xlabel("Importance", fontsize=14)
plt.ylabel("Feature", fontsize=14)

# Add annotations to each bar
for i, v in enumerate(feature_importances_lyft['Importance']):
    plt.text(v + 0.005, i, f"{v:.3f}", va='center', fontsize=10)

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

```
Random Forest Regressor Results:
Mean Squared Error (MSE): 1.921273379316906
R-squared (R2): 0.9808473657014866
```

```
/var/folders/30/r6bdbml51pb1n3cswh9vqjv80000gn/T/ipykernel_52759/1200634200.py:1
3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```



1.1.23 Decision Regressor for Lyft

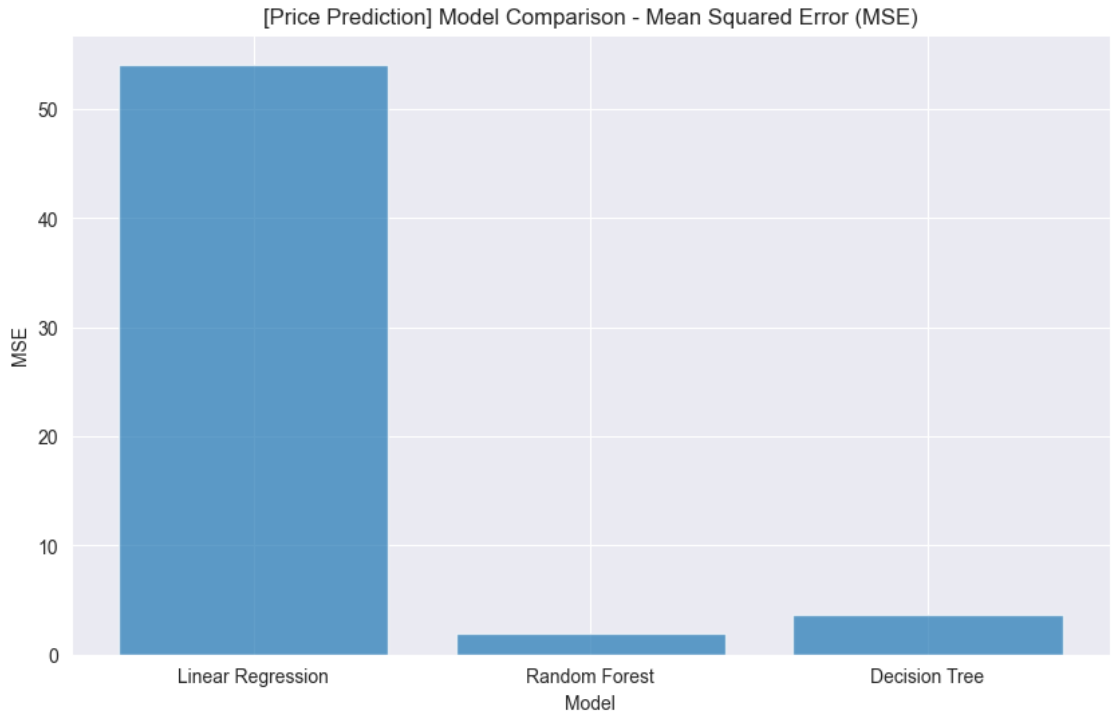
[241]:
```
# Decision Regressor Model for Lyft
dt_model_lyft = DecisionTreeRegressor(random_state=42)
dt_model_lyft.fit(X_train_lyft, y_train_lyft)
y_pred_dt_lyft = dt_model_lyft.predict(X_test_lyft)

mae = mean_absolute_error(y_test_lyft, y_pred_dt_lyft)
mse = mean_squared_error(y_test_lyft, y_pred_dt_lyft)
r2 = r2_score(y_test_lyft, y_pred_dt_lyft)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

# Optional: Feature Importance
feature_importances = pd.DataFrame({
    'Feature': X_train_lyft.columns,
    'Importance': dt_model_lyft.feature_importances_
```

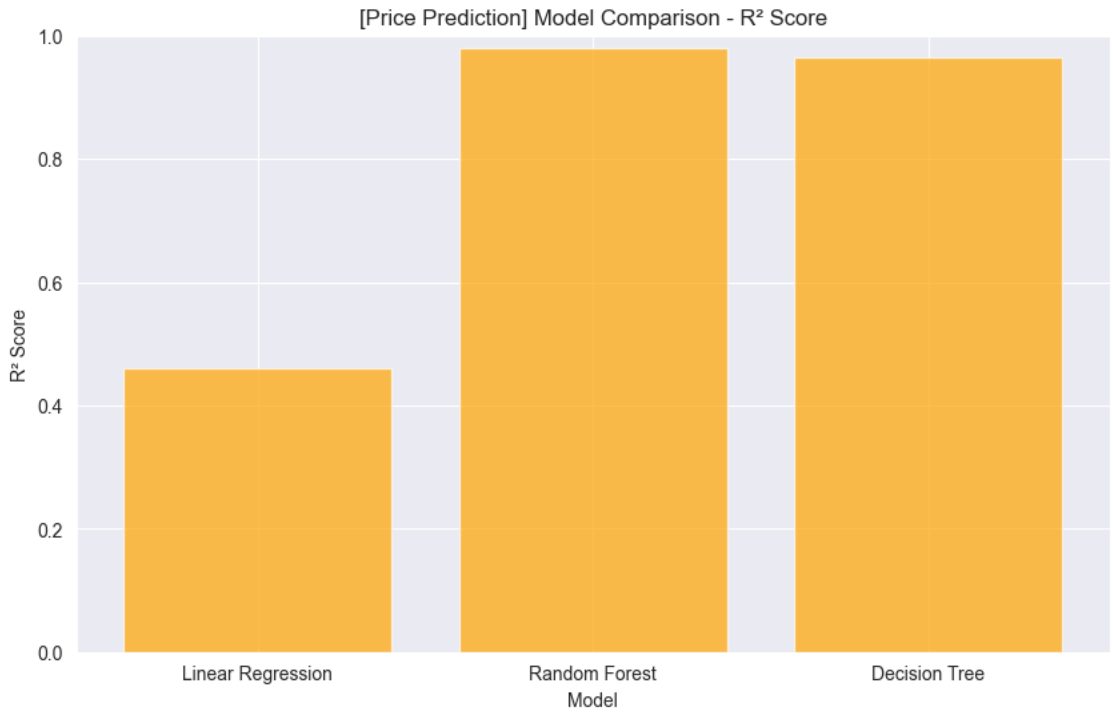
```
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importances:")
print(feature_importances)
```

```
Mean Absolute Error (MAE): 1.0933652125825444
Mean Squared Error (MSE): 3.6335568018101054
R² Score: 0.963778093540916

Feature Importances:
                Feature  Importance
2              cab_type    0.746423
3              distance    0.160794
4      surge_multiplier    0.073585
12               minute    0.003466
5   apparentTemperature    0.002969
1           destination    0.002909
0                source    0.002287
11                 hour    0.002174
7          visibility.1    0.002118
8    precipIntensityMax    0.001402
9                    day    0.001229
6       precipIntensity    0.000530
10                 month    0.000115
```

[242]:
```
# Model Comparison for Lyft
# Define the models
models = ['Linear Regression', 'Random Forest', 'Decision Tree']

# Calculate MSE and R² values
mse_values = [
    mean_squared_error(y_test_lyft, y_pred_linear_lyft),
    mean_squared_error(y_test_lyft, y_pred_rf_lyft),
    mean_squared_error(y_test_lyft, y_pred_dt_lyft)
]
r2_values = [
    r2_score(y_test_lyft, y_pred_linear_lyft),
    r2_score(y_test_lyft, y_pred_rf_lyft),
    r2_score(y_test_lyft, y_pred_dt_lyft)
]

# Plot the MSE values
plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, alpha=0.7, label='MSE')
plt.title('[Price Prediction] Model Comparison - Mean Squared Error (MSE)')
plt.ylabel('MSE')
plt.xlabel('Model')
```

```
plt.xticks(models)
plt.show()

# Plot the R² values
plt.figure(figsize=(10, 6))
plt.bar(models, r2_values, alpha=0.7, color='orange', label='R² Score')
plt.title('[Price Prediction] Model Comparison - R² Score')
plt.ylabel('R² Score')
plt.xlabel('Model')
plt.xticks(models)
plt.ylim(0, 1)  # R² ranges from 0 to 1
plt.show()
```





1.2 Model Usage Helper Functions

[ ]: