```
import pandas as pd

df = pd.read_csv("dynamic_pricing.csv")

# Head & Tail
print("🔹 First 5 rows:")
print(df.head())

print("\n🔹 Last 5 rows:")
print(df.tail())

# Info
print("\n🔹 Dataset Info:")
print(df.info())

# Shape
print("\nShape (rows, cols):", df.shape)

# This checks the start, end, data types, and structure.
```

```
   Historical_Cost_of_Ride
0               284.257273
1               173.874753
2               329.795469
3               470.201232
4               579.681422

🔹 Last 5 rows:
     Number_of_Riders  Number_of_Drivers Location_Category  \
995                33                 23             Urban
996                84                 29             Urban
997                44                  6          Suburban
998                53                 27          Suburban
999                78                 63             Rural

    Customer_Loyalty_Status  Number_of_Past_Rides  Average_Ratings  \
995                    Gold                    24             4.21
996                 Regular                    92             4.55
997                    Gold                    80             4.13
998                 Regular                    78             3.63
999                    Gold                    14             4.21

    Time_of_Booking Vehicle_Type  Expected_Ride_Duration  \
995         Morning      Premium                      11
996         Morning      Premium                      94
997           Night      Premium                      40
998           Night      Premium                      58
999       Afternoon      Economy                     147

     Historical_Cost_of_Ride
995                91.389526
996               424.155987
997               157.364830
998               279.095048
999               655.065106

🔹 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Number_of_Riders         1000 non-null   int64
 1   Number_of_Drivers        1000 non-null   int64
 2   Location_Category        1000 non-null   object
 3   Customer_Loyalty_Status  1000 non-null   object
 4   Number_of_Past_Rides     1000 non-null   int64
 5   Average_Ratings          1000 non-null   float64
 6   Time_of_Booking          1000 non-null   object
 7   Vehicle_Type             1000 non-null   object
 8   Expected_Ride_Duration   1000 non-null   int64
 9   Historical_Cost_of_Ride  1000 non-null   float64
dtypes: float64(2), int64(4), object(4)
memory usage: 78.3+ KB
None

Shape (rows, cols): (1000, 10)
```

```
print("\n🔹 Column Names:")
print(df.columns.tolist())

# Sanity: strip spaces, lowercase, replace bad chars
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")
print("\n🔹 Cleaned Column Names:")
print(df.columns.tolist())


#Ensures column names are clean and consistent.
```

```
🔹 Column Names:
['Number_of_Riders', 'Number_of_Drivers', 'Location_Category', 'Customer_Loyalty_Status', 'Number_of_Past_Rides', 'Average_Ratings', 'Ti

🔹 Cleaned Column Names:
['number_of_riders', 'number_of_drivers', 'location_category', 'customer_loyalty_status', 'number_of_past_rides', 'average_ratings', 'ti
```

Double-click (or enter) to edit

```
print("\n🔹 Missing Values Count:")
print(df.isnull().sum())

print("\n🔹 Percentage of Missing Values:")
print((df.isnull().mean() * 100).round(2))

# Tells us how many missing values per column.
```

```
🔹 Missing Values Count:
number_of_riders           0
number_of_drivers          0
location_category          0
customer_loyalty_status    0
number_of_past_rides       0
average_ratings            0
time_of_booking            0
vehicle_type               0
expected_ride_duration     0
historical_cost_of_ride    0
dtype: int64

🔹 Percentage of Missing Values:
number_of_riders           0.0
number_of_drivers          0.0
location_category          0.0
customer_loyalty_status    0.0
number_of_past_rides       0.0
average_ratings            0.0
time_of_booking            0.0
vehicle_type               0.0
expected_ride_duration     0.0
historical_cost_of_ride    0.0
dtype: float64
```

```
print("\n🔹 Number of duplicate rows:", df.duplicated().sum())
# Helps remove redundancy.
```

```
🔹 Number of duplicate rows: 0
```

```
num_cols = df.select_dtypes(include=["int64","float64"]).columns.tolist()
cat_cols = df.select_dtypes(include=["object","category"]).columns.tolist()

print("\n🔹 Numerical Columns:", num_cols)
print("\n🔹 Categorical Columns:", cat_cols)
# Shows mean, std, min, max, percentiles.
```

```
🔹 Numerical Columns: ['number_of_riders', 'number_of_drivers', 'number_of_past_rides', 'average_ratings', 'expected_ride_duration', 'h

🔹 Categorical Columns: ['location_category', 'customer_loyalty_status', 'time_of_booking', 'vehicle_type']
```

```
print("\n🔹 Numerical Columns Description:")
print(df[num_cols].describe().T)
# Useful for plotting distributions
```

```
🔹 Numerical Columns Description:
                          count        mean         std        min  \
number_of_riders         1000.0   60.372000   23.701506  20.000000
number_of_drivers        1000.0   27.076000   19.068346   5.000000
number_of_past_rides     1000.0   50.031000   29.313774   0.000000
average_ratings          1000.0    4.257220    0.435781   3.500000
expected_ride_duration   1000.0   99.588000   49.165450  10.000000
historical_cost_of_ride  1000.0  372.502623  187.158756  25.993449

                               25%         50%         75%         max
number_of_riders          40.000000   60.000000   81.000000  100.000000
number_of_drivers         11.000000   22.000000   38.000000   89.000000
number_of_past_rides      25.000000   51.000000   75.000000  100.000000
average_ratings            3.870000    4.270000    4.632500    5.000000
expected_ride_duration    59.750000  102.000000  143.000000  180.000000
historical_cost_of_ride  221.365202  362.019426  510.497504  836.116419
```

Double-click (or enter) to edit

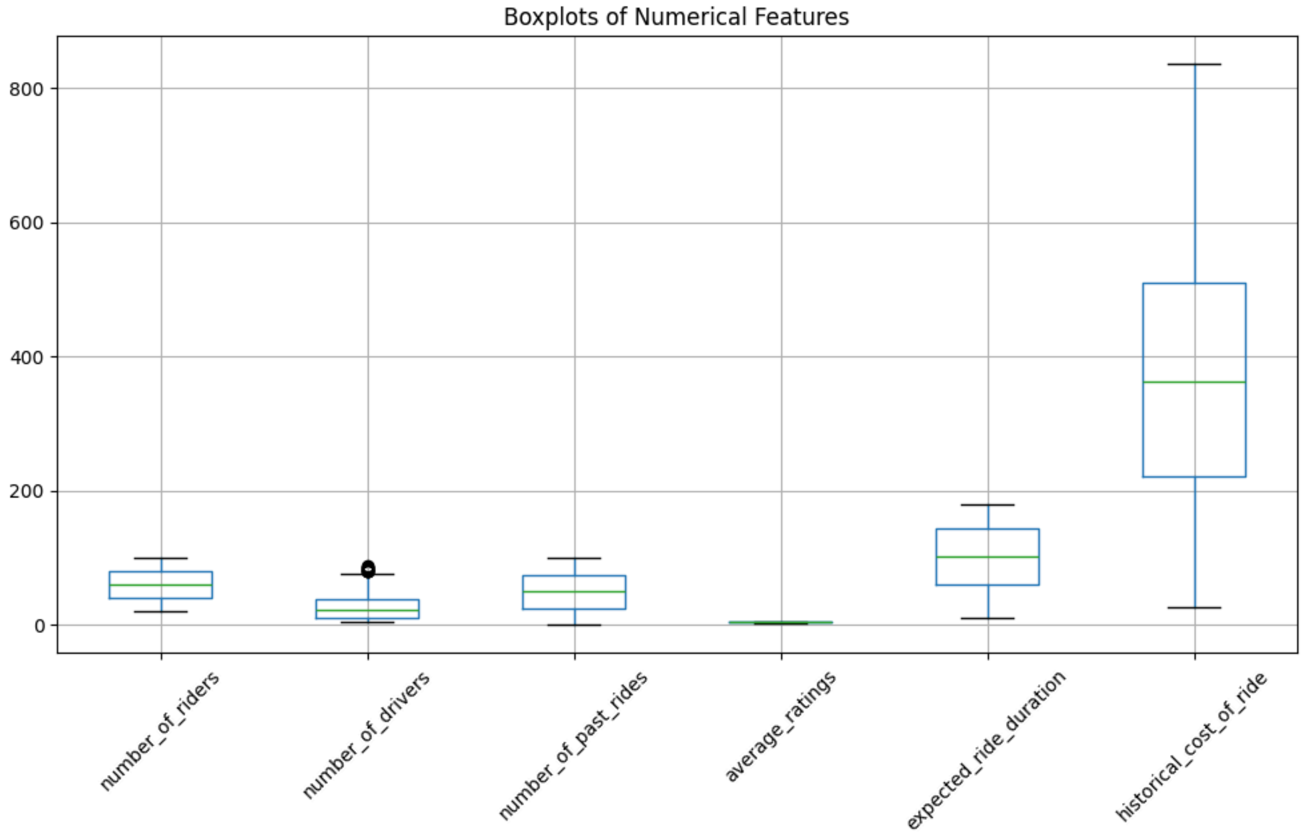
```
# Melt numerical columns into one column for visualization
num_long = df[num_cols].melt(var_name="feature", value_name="value")
print(num_long.head())
```

```
             feature  value
0  number_of_riders   90.0
1  number_of_riders   58.0
2  number_of_riders   42.0
3  number_of_riders   89.0
4  number_of_riders   78.0
```

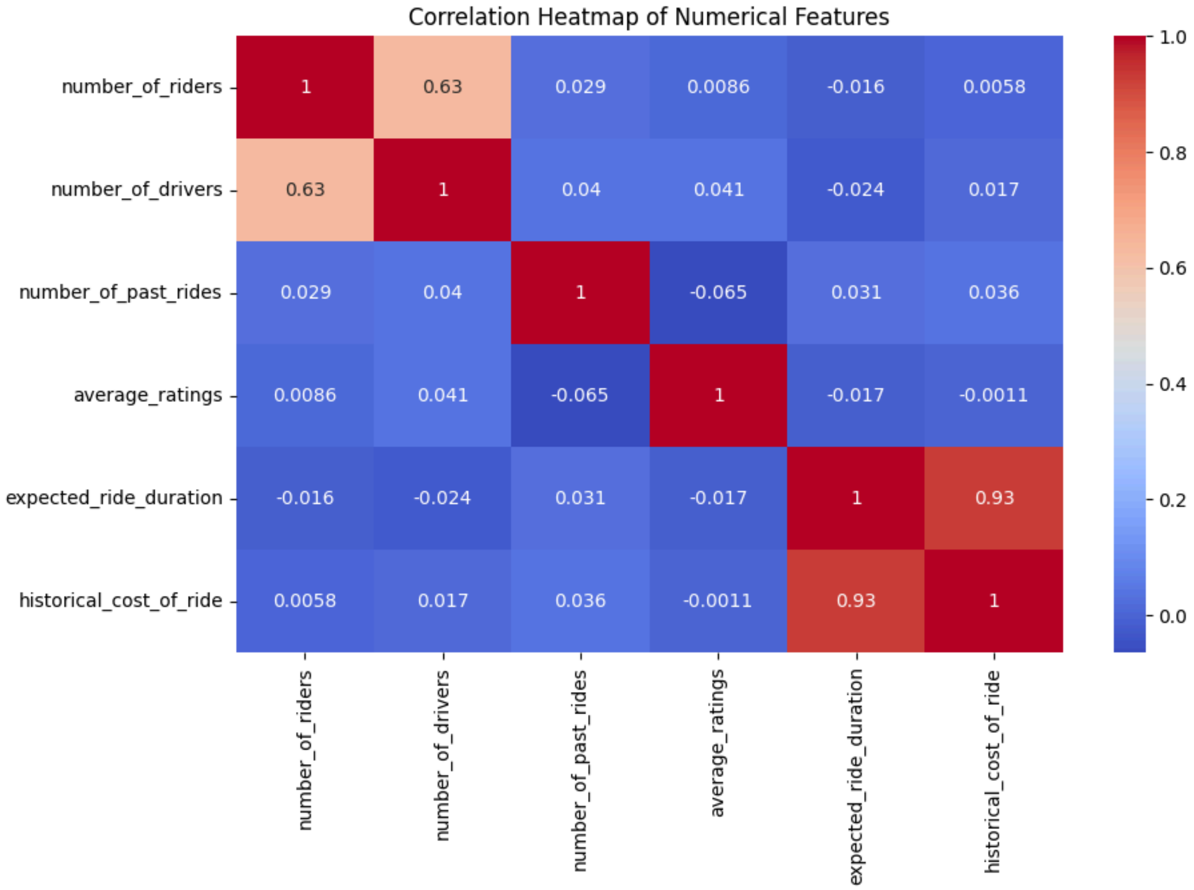
```
import numpy as np

def find_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower) | (data[column] > upper)]
    return outliers, lower, upper

for col in num_cols:
    outliers, low, high = find_outliers_iqr(df, col)
    print(f"\n🔹 {col}: {len(outliers)} outliers (lower={low:.2f}, upper={high:.2f})")
```

```

🔹 number_of_riders: 0 outliers (lower=-21.50, upper=142.50)

🔹 number_of_drivers: 10 outliers (lower=-29.50, upper=78.50)

🔹 number_of_past_rides: 0 outliers (lower=-50.00, upper=150.00)

🔹 average_ratings: 0 outliers (lower=2.73, upper=5.78)

🔹 expected_ride_duration: 0 outliers (lower=-65.12, upper=267.88)

🔹 historical_cost_of_ride: 0 outliers (lower=-212.33, upper=944.20)
```

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

# Boxplot of numerical features
plt.figure(figsize=(12,6))
df[num_cols].boxplot()
plt.xticks(rotation=45)
plt.title("Boxplots of Numerical Features")
plt.show()
```



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

plt.figure(figsize=(10,6))
sns.heatmap(df[num_cols].corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



```
for col in cat_cols:
    print(f"\n{col} value counts:")
    print(df[col].value_counts(normalize=True))
```

```
location_category value counts:
location_category
Urban       0.346
Rural       0.332
Suburban    0.322
Name: proportion, dtype: float64

customer_loyalty_status value counts:
customer_loyalty_status
Silver     0.367
Regular    0.320
Gold       0.313
Name: proportion, dtype: float64

time_of_booking value counts:
time_of_booking
Night        0.276
Afternoon    0.247
Morning      0.246
Evening      0.231
Name: proportion, dtype: float64

vehicle_type value counts:
vehicle_type
Premium    0.522
Economy    0.478
Name: proportion, dtype: float64
```

```
from scipy.stats import skew

for col in num_cols:
    print(f"{col} skewness: {skew(df[col])}")
```

```
number_of_riders skewness: 0.0021632616645425417
number_of_drivers skewness: 0.9617588924264777
number_of_past_rides skewness: -0.008444267438104912
average_ratings skewness: -0.07863604521780783
expected_ride_duration skewness: -0.13965805969802533
```

```
# ============================
# KPI Calculations for Dynamic Pricing Dataset
# ============================

# Map dataset columns to our KPI terms
df['Price'] = df['Historical_Cost_of_Ride']
df['CompletedRides'] = df['Number_of_Riders']

# Assume cost per ride = 70% of price (example, since cost column not available)
# You can adjust this value if you have actual operating cost data
df['Cost_per_ride'] = df['Price'] * 0.7

# ---------- KPI 1: Revenue ----------
df['Revenue'] = df['Price'] * df['CompletedRides']

# ---------- KPI 2: Profit ----------
df['Profit'] = (df['Price'] - df['Cost_per_ride']) * df['CompletedRides']

# ---------- KPI 3: Revenue Lift (%) ----------
# Baseline Revenue = mean revenue across dataset
baseline_revenue = df['Revenue'].mean()
df['Revenue_Lift_%'] = ((df['Revenue'] - baseline_revenue) / baseline_revenue) * 100

# ---------- KPI 4: Gross Margin (%) ----------
df['Gross_Margin_%'] = (df['Profit'] / df['Revenue']) * 100

# ---------- KPI 5: Conversion Rate (%) ----------
# Since dataset doesn’t have booking intents, assume "Number_of_Riders" = completed rides
# For now, set booking intents = riders + 10% (example assumption)
df['BookingIntents'] = df['CompletedRides'] * 1.1
df['Conversion_Rate_%'] = (df['CompletedRides'] / df['BookingIntents']) * 100

# ---------- KPI 6: Price Change Rate (%) ----------
# Price volatility: percent difference from mean price
mean_price = df['Price'].mean()
df['Price_Change_%'] = (abs(df['Price'] - mean_price) / mean_price) * 100

# ---------- KPI 7: Cancellation Rate (%) ----------
# If no "CancelledRides" column exists, assume cancellation = 5% of riders (example assumption)
df['CancelledRides'] = df['CompletedRides'] * 0.05
df['Cancellation_Rate_%'] = (df['CancelledRides'] / (df['CompletedRides'] + df['CancelledRides'])) * 100

# ============================
# Final KPI Summary
# ============================
kpi_summary = {
    "Total Revenue (₹)": df['Revenue'].sum(),
    "Total Profit (₹)": df['Profit'].sum(),
    "Avg Revenue Lift (%)": df['Revenue_Lift_%'].mean(),
    "Avg Gross Margin (%)": df['Gross_Margin_%'].mean(),
    "Avg Conversion Rate (%)": df['Conversion_Rate_%'].mean(),
    "Avg Price Change Rate (%)": df['Price_Change_%'].mean(),
    "Avg Cancellation Rate (%)": df['Cancellation_Rate_%'].mean()
}

# Show KPI summary
print("\n=== KPI Summary ===")
for k, v in kpi_summary.items():
    print(f"{k}: {v:.2f}")

# Preview dataframe with KPIs
df.head()
```

```
=== KPI Summary ===
Total Revenue (₹): 22514545.02
Total Profit (₹): 6754363.51
Avg Revenue Lift (%): 0.00
Avg Gross Margin (%): 30.00
Avg Conversion Rate (%): 90.91
Avg Price Change Rate (%): 42.13
```