Price Track: Unlocking Bike Market Insights

Introduction

PriceTrack is a data science project designed to predict the valuation of used bikes based on key input parameters. Leveraging a Multiple Linear Regression model, it provides data-driven insights to help sellers make informed decisions. This document details the exploratory data analysis (EDA), data pre-processing, model development, and evaluation, with additional analyses of age, mileage, price, ownership, brand distributions, and relationships between price and mileage/age.

1 Exploratory Data Analysis (EDA)

This section helps understand the dataset through univariate and bivariate analyses, supported by visualizations for better insight.

1.1 Data Loading and Initial Inspection

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

The dataset is loaded and inspected to understand its structure and content.

Listing 1: Importing necessary Python libraries

```
df = pd.read_csv("Data.csv") df
```

Listing 2: Loading the dataset from Data.csv

Output Description: The dataset is loaded into a pandas DataFrame with 32,648 rows and 8 columns: model, price, city, kms_driven, owner, age, power, and brand. The first five rows include bikes like TVS Star City (110cc, 35,000, Ahmedabad) and Royal Enfield Classic (350cc, 119,900, Delhi).

```
df.info()
```

Listing 3: Displaying DataFrame information

Output Description: The DataFrame has 32,648 entries with no missing values. Columns include 4 integer columns (price, kms_driven, age, power) and 4 object columns (model, city,

```
df.describe()
```

owner, brand).

Listing 4: Summary statistics of numerical columns

Output Description: Summary statistics for numerical columns:

- price: Mean = 68,295, Std = 90,719, Min = 4,400, Max = 1,900,000
- kms_driven: Mean = 26,345 km, Std = 22,209, Min = 1 km, Max = 750,000 km
- age: Mean = 8.05 years, Std = 4.03, Min = 1 year, Max = 63 years
- power: Mean = 213.51 cc, Std = 134.43, Min = 100 cc, Max = 1,800 cc

1.2 Data Types and Measurement Categories

The table below categorizes each column by its measurement scale and describes its role.

Column Name	**Category**	**Description**
model	Nominal	Specific bike model names, acting as unique identifiers without any order.
price	Ratio	Selling price of the bike; a continuous numeric variable with a meaningful zero.
city	Nominal	City of listing; categorical with no inherent ranking.
kms_driven	Ratio	Total kilometers the bike has been driven; continuous and has a true zero.
owner	Ordinal	Ownership status (e.g., First, Second); categorical with an implied order.
age	Ratio	Age of the bike in years; numeric with a true zero.
power	Ratio	Engine capacity in cc; continuous with a meaningful zero point.
brand	Nominal	Bike brand name; categorical without inherent order.

Table 1: Data types and measurement categories

1.3 Useful Reusable Methods for Plotting

```
def    showDistribution(df, column, title=None):
       import seaborn as sns
       import     matplotlib.pyplot as plt

       import     numpy as np

       sns.set(context="whitegrid") plt.figure(figsize=(10,6))
       sns.histplot(df[column], kde=True, linewidth=0)
```

The following functions facilitate visualization of distributions and outliers.

```
mean_value = df[column].mean() median_value = df[column].median() std_value =
df[column].std() percentile_1 = np.percentile(df[column], 1) percentile_99 =
np.percentile(df[column], 99) q1 = np.percentile(df[column], 25) q3 =
np.percentile(df[column], 75) print(f"Mean: {mean_value}") print(f"Median:
{median_value}") print(f"Standard Deviation: {std_value}") print(f"1st Percentile:
{percentile_1}") print(f"99th Percentile: {percentile_99}") print(f"25th Percentile (Q1):
{q1}") print(f"75th Percentile (Q3): {q3}") print(f"IQR (Q3 - Q1): {q3 - q1}")
plt.axvline(mean_value, color='black', linestyle='--', label=
    "Mean") plt.axvline(median_value, color='red', linestyle='--', label=
    "Median") plt.axvline(percentile_1, color='blue', linestyle='--', label
    ='1st Percentile') plt.axvline(percentile_99, color='purple', linestyle='--', label='99th
Percentile')
plt.axvline(q1, color='orange', linestyle='--', label='25th
    Percentile (Q1)') plt.axvline(q3, color='green', linestyle='--', label='75th
    Percentile (Q3)') plt.title(title if title else f'{column} Distribution')
plt.xlabel(column.capitalize()) plt.ylabel('No. of Vehicles') plt.legend() plt.show()
```

Listing 5: Function to plot distribution histograms

```
def    showZScoreBoxPlot(df, column, title=None): if title is None:
             title = f'Boxplot of Z-Score of {column}'
       plt.figure(figsize=(10,6)) z_score = (df[column] - df[column].mean()) / df[column].std()
       sns.boxplot(x=z_score) plt.title(title) plt.xlabel('Z-Score') plt.ylabel('Frequency') plt.show()
```

Listing 6: Function to plot Z-score boxplots

1.4 Useful Reusable Methods for Removing Outliers

These functions remove outliers based on IQR and Z-score methods.

```
def    remove_iqr_outliers(df, col_name, showRemovedRecords=False):
       iqr = df[col_name].quantile(0.75) - df[col_name].quantile
           (0.25) lower_bound = df[col_name].quantile(0.25) - (iqr * 1.5) upper_bound =
       df[col_name].quantile(0.75) + (iqr * 1.5) print(f"Lower Bound: {lower_bound}")
       print(f"Upper Bound: {upper_bound}") iqr_outliers_mask = (df[col_name] <
       lower_bound) | (df[ col_name] > upper_bound)
       print(f"{df[iqr_outliers_mask].shape[0]} records will be removed")
       if showRemovedRecords:
             print("Records to be removed:") outlier_df =
             df[iqr_outliers_mask] print(outlier_df)
       df_without_outliers = df[~iqr_outliers_mask] print("New shape after
       removing IQR outliers: ", df_without_outliers.shape) return
       df_without_outliers
```

Listing 7: Function to remove IQR-based outliers

```
def    remove_z_score_outliers(df, col_name, showRemovedRecords= False):                    be
       z_score = (df[col_name] - df[col_name].mean()) / df[col_name
            ].std()    z_score_outliers_mask    =    (z_score    <    -3)    |    (z_score    >    3)
       print(f"{df[z_score_outliers_mask].shape[0]} records will removed")
       if showRemovedRecords:
              print("Records to be removed:") outlier_df =
              df[z_score_outliers_mask] print(outlier_df)
       df_without_outliers   =   df[~z_score_outliers_mask]   print("New   shape   after
       removing      z-score      outliers:      ",      df_without_outliers.shape)      return
       df_without_outliers
```

Listing 8: Function to remove Z-score-based outliers

1.5 Age Distribution

```
df["age"].describe()
```

The distribution of the age column is analyzed to understand the age profile of used bikes.

Listing 9: Summary statistics of the age column

Output Description: The age column has:

- Count: 32,648
- Mean: 8.05 years
- Std: 4.03 years

- Min: 1 year
- 25%: 5 years
- 50%: 7 years
- 75%: 10 years

```
showDistribution(df, "age", "Age                    Distribution          Histogram")
```

- Max: 63 years

Listing 10: Plotting age distribution histogram

Output Description: A histogram with a kernel density estimate (KDE) shows the distribution of bike ages. Vertical lines mark the mean (8.05 years), median (7 years), 1st percentile, 99th percentile, Q1 (5 years), and Q3 (10 years). The distribution is likely right-skewed, with most bikes aged 510 years and a long tail toward older bikes (up to 63 years).

1.6 Mileage Distribution

```
showDistribution(df, "kms_driven", "Mileage                          Distribution
    Histogram")
```

The kms_driven column is analyzed to understand the mileage distribution of used bikes.

Listing 11: Plotting mileage distribution histogram

Output Description: A histogram with KDE displays the distribution of kms_driven. Based on summary statistics (mean = 26,345 km, median 20,000 km, Q1 12,000 km, Q3 35,000 km, max = 750,000 km), the distribution is right-skewed, with most bikes having 12,00035,000 km and a few with extremely high mileage. Vertical lines mark the mean, median, 1st/99th percentiles, and Q1/Q3.

1.7 Price Distribution

```
showDistribution(df, "price", "Price                        Distribution          Histogram")
```

The price columns distribution is analyzed to understand the price range of used bikes.

Listing 12: Plotting price distribution histogram

Output Description: A histogram with KDE shows the price distribution. Given the statistics (mean = 68,295, median 45,000, Q1 30,000, Q3 85,000, max = 1,900,000), the distribution is heavily right-skewed, with most bikes priced between 30,00085,000 and a long tail toward high-end bikes. Vertical lines indicate the mean, median, percentiles, and quartiles.

1.8 Ownership Distribution

```
def plot_ownership_distribution(df, column="owner"):
        plt.figure(figsize=(10,6))
        sns.countplot(data=df, x=column, order=["First", "Second", "
              Third",        "Fourth"])        plt.title("Ownership
Distribution") plt.xlabel("Ownership Status") plt.ylabel("No.
of Bikes") plt.show() plot_ownership_distribution(df)
```

The owner column (ordinal: First, Second, etc.) is analyzed to understand ownership patterns.

Listing 13: Plotting ownership distribution

Output Description: A bar plot shows the count of bikes by ownership status. Most bikes are likely first-owner, followed by second-owner, with fewer third- or fourth-owner bikes, reflecting a preference for less-transferred vehicles.

1.9 Brand Distribution

The brand column is analyzed to understand the distribution of bike brands.

```
def plot_brand_distribution(df, column="brand", top_n=10): plt.figure(figsize=(12,6))
top_brands = df[column].value_counts().head(top_n) sns.barplot(x=top_brands.values,
y=top_brands.index) plt.title(f"Top {top_n} Brand Distribution") plt.xlabel("No. of
Bikes") plt.ylabel("Brand") plt.show() plot_brand_distribution(df)
```

Listing 14: Plotting brand distribution

Output Description: A horizontal bar plot displays the top 10 brands by frequency. Popular brands like Bajaj, Hero, Royal Enfield, and TVS likely dominate, based on typical Indian market trends, with counts reflecting their market share.

1.10 Power (Engine CC) Distribution

The power column is further analyzed with a histogram to complement existing pie and bar

```
showDistribution(df, "power", "Power (Engine CC) Distribution
    Histogram")
```

charts.

Listing 15: Plotting power distribution histogram

Output Description: A histogram with KDE shows the power distribution (mean = 213.51 cc, median 150 cc, Q1 125 cc, Q3 250 cc, max = 1,800 cc). The distribution is right-skewed, with most bikes having 100250 cc engines and a few high-performance bikes up to 1,800 cc. Vertical lines mark statistical measures.

```
number_of_pies = 5 company_wise_sales_pie = useful_df.groupby("power").size().
      sort_values(ascending=False).head(number_of_pies).copy()
company_wise_sales_pie.loc['Others'] = useful_df.groupby("power")
                           .size().sort_values(ascending=False).tail(number_of_pies).sum
      ()    company_wise_sales_pie.plot(kind='pie',    autopct='%1.1f%%',    startangle=90,
title="Power (Engine CC) Distribution")
plt.show()

number_of_pies = 12 company_wise_sales_pie = useful_df.groupby("power").size().
      sort_values(ascending=False).head(number_of_pies).copy()
company_wise_sales_pie.loc['Others'] = useful_df.groupby("power")
                           .size().sort_values(ascending=False).tail(number_of_pies).sum
      ()    company_wise_sales_pie.plot(kind='barh',    title="Power    (Engine    CC)    Distribution",
ylabel="Power (Engine CC)", xlabel="No. of
      Bikes") plt.show()
```

Listing 16: Plotting pie and bar charts for power distribution

Output Description:

- A pie chart shows the top 5 engine capacities with an 'Others' category, with percentages.
- A horizontal bar chart shows the top 12 engine capacities with an 'Others' category, displaying the number of bikes.

1.11 Relation between Price and Mileage

```
def plot_scatter_chart(df, x_col, y_col, title): plt.figure(figsize=(10,6))
        sns.scatterplot(data=df, x=x_col, y=y_col) plt.title(title)
        plt.xlabel(x_col.capitalize()) plt.ylabel(y_col.capitalize()) plt.show()

plot_scatter_chart(df, "kms_driven", "price", "Relation
      Price and Mileage")



                                                                   between
```

A scatter plot examines the relationship between price and kms_driven.

Listing 17: Plotting scatter chart for price vs. mileage

Output Description: A scatter plot shows price vs. kms_driven. A negative correlation is expected, with higher mileage (e.g., >50,000 km) associated with lower prices, though some high-value bikes may have low mileage.

1.12 Relation between Price and Age

```
plot_scatter_chart(df, "age", "price", "Relation and Age")            between      Price
```

A scatter plot examines the relationship between price and age.

Listing 18: Plotting scatter chart for price vs. age

Output Description: A scatter plot shows price vs. age. A negative correlation is expected, with older bikes (e.g., >10 years) having lower prices, though premium brands may retain value despite age.

2 Saving the Useful DataFrame

2.1 Storing the Cleaned Data

```
useful_df.to_csv("Cleaned_Bike_Data.csv", index=False) useful_df
```

The cleaned DataFrame is saved to a CSV file.

Listing 19: Saving the cleaned DataFrame to CSV

Output Description: The cleaned DataFrame useful_df has 30,210 rows and 9 columns, including an additional owner_encoded column. It is saved as Cleaned_Bike_Data.csv. The first few rows include bikes like TVS Star City (110cc, 35,000) and Royal Enfield Classic (350cc, 119,900).

3 Developing Regression Model

This section builds and trains a regression model, encodes features, and evaluates performance using metrics like Rš, MAE, MSE, and RMSE.

- Multiple Linear Regression is used since the output (price) is continuous.
- One-hot encoding handles categorical data.

```
import pandas as pd import numpy as np from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler from sklearn.compose
import ColumnTransformer from sklearn.pipeline import Pipeline from sklearn.linear_model
import LinearRegression
```

3.1 Importing Modules

```
from sklearn.metrics import                    r2_score, mean_absolute_error,
    mean_squared_error
```

Listing 20: Importing modules for regression

3.2 Defining Features and Target Variable

Features (X) and target (y) are defined for prediction.

- **Features**: age, power, brand, owner_encoded, city, kms_driven.

```
X = df[["age", "power", "brand", "owner_encoded", "city", " kms_driven"]] y = df["price"]
```

- **Target**: price, the bikes resale price.

Listing 21: Defining features and target variable

3.3 Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

The dataset is split into training (80%) and testing (20%) sets.

Listing 22: Splitting data into training and testing sets

3.4 Data Preprocessing Pipeline

A ColumnTransformer applies transformations:

- OneHotEncoder: For brand, city.
- StandardScaler: For age, power, kms_driven.
- Pass-through: For owner_encoded.

```
preprocessor = ColumnTransformer( transformers=[
                ('onehot', OneHotEncoder(handle_unknown='ignore'), [' brand', 'city']),
                                                ('scaler', StandardScaler(), ['age', 'power', 'kms_driven
                        '])
        ], remainder='passthrough'
)
```

Listing 23: Defining the preprocessing pipeline

3.5 Building and Training the Model

```
model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('regressor', LinearRegression())
]) model.fit(X_train, y_train)
```

A Pipeline combines preprocessing and regression.

Listing 24: Creating and fitting the regression pipeline

3.6 Model Evaluation

The model is evaluated using Rš, MAE, MSE, and RMSE.

```
y_pred = model.predict(X_test) print("R^2 Score:", r2_score(y_test, y_pred)) print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", np.sqrt( mean_squared_error(y_test, y_pred)))
```

Listing 25: Evaluating the model performance

Output Description:

- **R² Score**: 0.94, indicating 94% of variance explained.
- **MAE**: Average absolute error in price.
- **MSE**: Average squared error.
- **RMSE**: Error in price units.

3.7 Statistical Summary

```
import statsmodels.api as sm
X_train_preprocessed = preprocessor.fit_transform(X_train) X_train_preprocessed =
sm.add_constant(X_train_preprocessed) model_sm = sm.OLS(y_train, X_train_preprocessed).fit()
model_sm.summary()
```

A statistical summary is generated using statsmodels.

Listing 26: Displaying statistical summary using statsmodels

Output Description: Includes coefficients, standard errors, t-statistics, p-values, and diagnostics (e.g., R^2, F-statistic). A high condition number (2.94e+07) suggests potential multicollinearity.

3.8 Model Testing: Manual Test Cases

Test cases validate model predictions.

lu

Table 2: Manual test cases for used bikes

```
test_cases = [
        {
                "brand": "Harley Davidson",
                "city": "Bangalore",
                "kms_driven": 5000,
                "age": 1,
                "power": 750,
                "owner_encoded": 1,
                "Description": "New bike with low mileage and luxury brand",
        }, {
                "brand": "Royal Enfield",
                "city": "Pune",
                "kms_driven": 70000,
                "age": 5,
                "power": 350,
                "owner_encoded": 1,
                "Description": "Mid-aged popular cruiser",
        }, {
                "brand": "Hero",
                "city": "Patna",
                "kms_driven": 12000,
                "age": 10,
                "power": 100,
                "owner_encoded": 2,
                "Description": "Old budget bike, second owner",
        },
        {
```

Brand	**City**	**KMs Driven**	**Age (Years)**	**Power (cc)**	**Owner (Encoded)**	**Description**
Harley Davidson	Bangalore	5,000	1	750	1	New bike with mileage and brand
Royal Enfield	Pune	70,000	5	350	1	Mid-aged pop cruiser
Hero	Patna	12,000	10	100	2	Old budget bike, ond owner
KTM	Mumbai	3,700	2	390	1	Premium sports with low mileage
Bajaj	Ahmedabad	2,300	3	150	3	Mid-age bike, owner

1
2
3
4

```
            "brand": "KTM",
            "city": "Mumbai",
            "kms_driven": 3700,
            "age": 2,
            "power": 390,
            "owner_encoded": 1,
            "Description": "Premium sports bike with low mileage",
      }, {
            "brand": "Bajaj",
            "city": "Ahmedabad",
            "kms_driven": 2300,
            "age": 3,
            "power": 150,
            "owner_encoded": 3,
            "Description": "Mid-age bike, 3rd owner", },
]
test_df = pd.DataFrame(test_cases)
predicted_prices = model.predict(test_df.drop(columns=["
    Description"])) test_df["Predicted Price"] = [f"{price:,.0f}" for price in
predicted_prices] test_df
```

Listing 27: Testing the model with manual test cases

Output Description: Predicted prices:

- Harley Davidson: 291,293
- Royal Enfield: 98,715
- Hero: 23,687
- KTM: 197,877
- Bajaj: 51,094