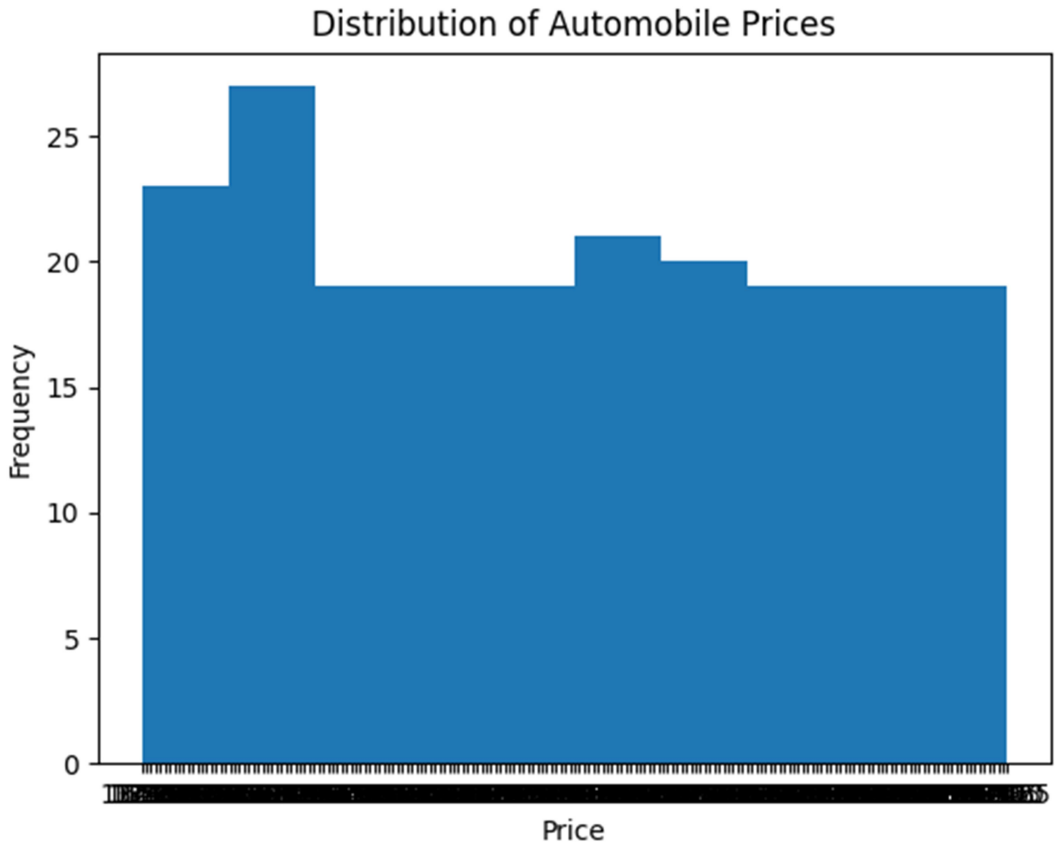
1. Load the dataset: Use the appropriate libraries or functions in your preferred programming language (such as pandas in Python) to load the dataset into memory.

```
import pandas as pd

# Load the dataset

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

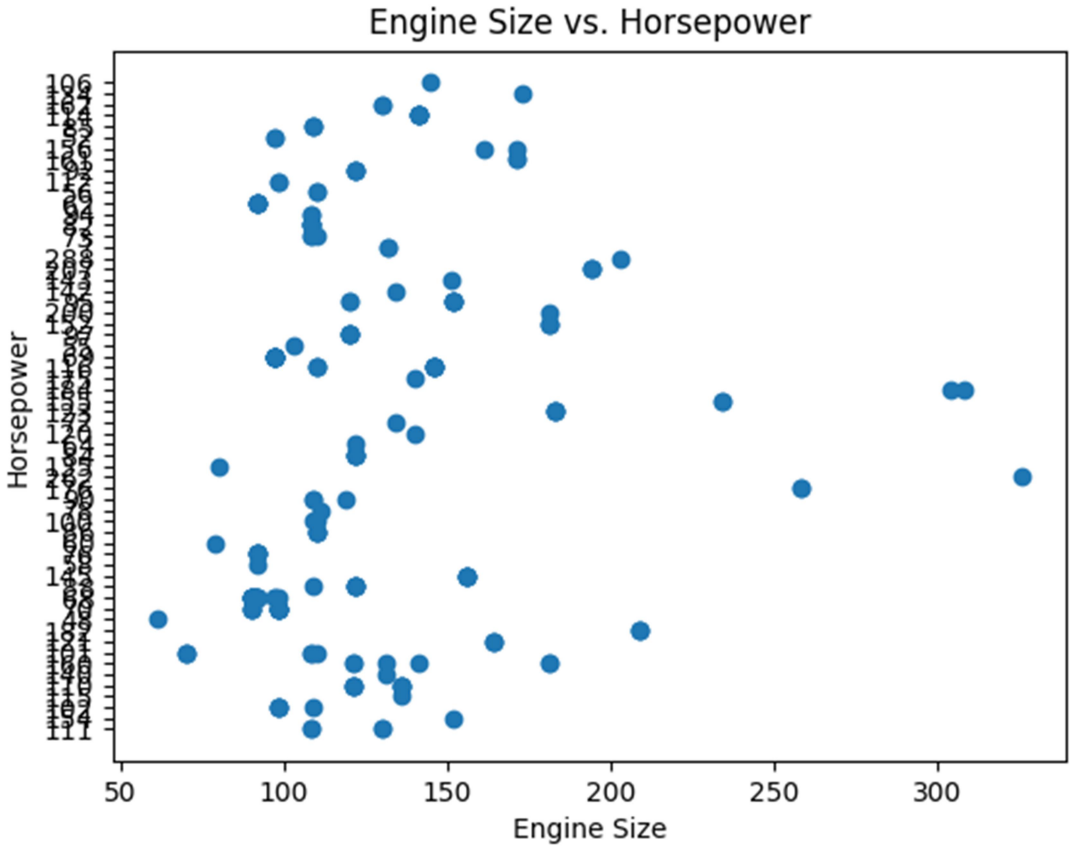
2. Perform the visualizations:
    - Univariate Analysis: Explore individual variables in the dataset. You can plot histograms, box plots, or bar charts to understand the distribution, range, and outliers of each variable.

```
import matplotlib.pyplot as plt

# Example of univariate analysis - histogram of automobile prices

plt.hist(df['price'])

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.title('Distribution of Automobile Prices')

plt.show()
```



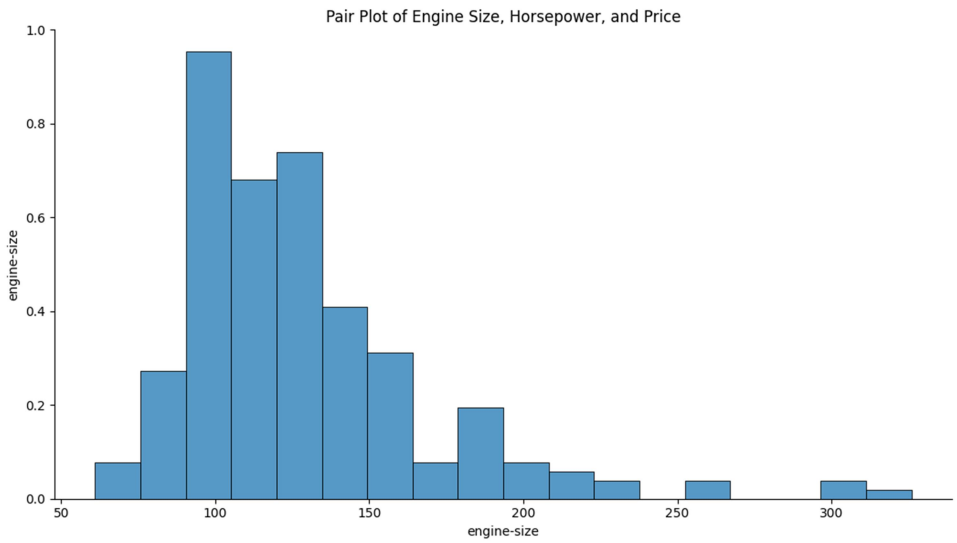
- b. Bi-Variate Analysis: Analyze the relationships between pairs of variables. Use scatter plots, line plots, or correlation matrices to identify any patterns or correlations between variables.

```
# Example of bivariate analysis - scatter plot of engine size vs. horsepower

plt.scatter(df['engine_size'], df['horsepower'])

plt.xlabel('Engine Size')

plt.ylabel('Horsepower')

plt.title('Engine Size vs. Horsepower')

plt.show()
```



- c. Multivariate Analysis: Consider the interactions between multiple variables. Utilize techniques like heatmaps, pair plots, or parallel coordinates to visualize the relationships between more than two variables.

```
import seaborn as sns

# Example of multivariate analysis - pair plot

sns.pairplot(df[['engine_size', 'horsepower', 'price']])

plt.title('Pair Plot of Engine Size, Horsepower, and Price')

plt.show()
```



3. Perform Data Preprocessing:
    - Handling missing values: Identify and handle any missing values in the dataset. You can choose to remove rows with missing values or impute them with appropriate techniques such as mean, median, or interpolation.

```
# Check for missing values
df.isnull().sum()
```

```
symboling          0
normalized-losses    0
make               0
fuel-type          0
aspiration         0
num-of-doors       0
body-style         0
drive-wheels       0
engine-location    0
wheel-base         0
```

```
length             0
width              0
height             0
curb-weight        0
engine-type        0
num-of-cylinders   0
engine-size        0
fuel-system        0
bore               0
stroke             0
compression-ratio  0
horsepower         0
peak-rpm           0
city-mpg           0
highway-mpg        0
price              0
dtype: int64
```

```
# Handle missing values by dropping rows with missing values
df.dropna(inplace=True)
```

- b. Handling categorical variables (Encoding): If your dataset contains categorical variables, you may need to encode them into numerical values for the machine learning model to process. Common encoding techniques include one-hot encoding or label encoding.

```
# Example of one-hot encoding for a categorical variable 'fuel_type'
df_encoded = pd.get_dummies(df, columns=['fuel_type'])
```

- c. Perform scaling: Scale the numerical variables to a similar range to avoid dominance by variables with larger values. Common scaling methods include standardization (subtracting mean and dividing by standard deviation) or normalization (scaling to a range between 0 and 1).

```
from sklearn.preprocessing import StandardScaler

# Example of standardization for numerical variables

scaler = StandardScaler()

df_scaled = scaler.fit_transform(df_encoded[['engine_size',
'horsepower']])
```

- d. Check correlation & descriptive statistics: Examine the correlation between variables to identify any strong relationships. Calculate descriptive statistics such as mean, median, standard deviation, etc., to gain insights into the central tendencies and distributions of the variables.

```
# Example of correlation matrix and descriptive statistics

correlation_matrix = df_encoded.corr()

descriptive_stats = df_encoded.describe()
```

4. Build Machine Learning Model: Select an appropriate machine learning algorithm based on your problem statement and dataset characteristics. Split your dataset into training and testing sets
   - . Train the model on the training set using suitable model training techniques and algorithms.

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

# Example of building a linear regression model

X = df_encoded.drop('price', axis=1)

y = df_encoded['price']

# 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)

# Train a linear regression model

model = LinearRegression()

model.fit(X_train, y_train)
```

5. Evaluate the Machine Learning model: Use the trained model to make predictions on the test set. Evaluate the model's performance using appropriate evaluation metrics such as mean squared error (MSE), R-squared score, or others relevant to your problem.

```
from sklearn.metrics import mean_squared_error, r2_score

# Make predictions on the test set

y_pred = model.predict(X_test)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared Score: {r2}")
```