Car Purchase Decision-Making Data Analysis

1. Introduction

This analysis aims to explore factors that influence car purchase decisions based on various features provided in the dataset.

- **Dataset Overview**: The dataset contains features such as Buying Price, Maintenance Price, Number of Doors, Person Capacity, Size of Luggage, Safety, Fuel Type, Horsepower, and Car Acceptability.

2. Data Preprocessing

Loading the Data

In [20]:
```
import pandas as pd

# Load the dataset with the correct delimiter
df = pd.read_csv('car_purchase_decision.csv', sep=';')
```

Out[20]:

	Unnamed: 0	Buying_Price	Maintenance_Price	No_of_Doors	Person_Capacity	Size_of_Luggage	Safety	Fuel_Type	Horse_power	Car_Acceptability
0	0	vhigh	vhigh	2	2	small	low	petrol	238	unacc
1	1	vhigh	vhigh	2	2	small	med	hybrid	232	unacc
2	2	vhigh	vhigh	2	2	small	high	hybrid	285	unacc
3	3	vhigh	vhigh	2	2	med	low	diesel	321	unacc
4	4	vhigh	vhigh	2	2	med	med	hybrid	175	unacc

Data Cleaning

Handle missing values, check for duplicates, and standardize variable types.

In [96]:
```
# Check for missing values
df.isnull().sum()

# Remove duplicates if any
df.drop_duplicates(inplace=True)

# Rename the column 'Unnamed: 0' to 'Serial number'
df.rename(columns={'Unnamed: 0': 'Serial number'}, inplace=True)

# Convert 'No_of_Doors' and 'Person_Capacity' from object to int64
df['No_of_Doors'] = pd.to_numeric(df['No_of_Doors'], errors='coerce').fillna(0).astype('int64')
df['Person_Capacity'] = pd.to_numeric(df['Person_Capacity'], errors='coerce').fillna(0).astype('int64')

# Verify the changes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Serial number     1728 non-null   int64
 1   Buying_Price      1728 non-null   object
 2   Maintenance_Price 1728 non-null   object
 3   No_of_Doors       1728 non-null   int64
 4   Person_Capacity   1728 non-null   int64
 5   Size_of_Luggage   1728 non-null   object
 6   Safety            1728 non-null   object
 7   Fuel_Type         1728 non-null   object
 8   Horse_power       1728 non-null   int64
 9   Car_Acceptability 1728 non-null   object
dtype: int64, object: 7
memory usage: 135.1 KB
```

In [125]:
```
# Function to remove outliers based on the IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Keep only the rows that are within the bounds
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# List of numeric columns to check for outliers
numeric_columns = ['No_of_Doors', 'Person_Capacity', 'Horse_power']

# Remove outliers from each numeric column and update the DataFrame
for col in numeric_columns:
    df = remove_outliers(df, col)

# Check the total number of rows after removing outliers
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Serial number     1728 non-null   int64
 1   Buying_Price      1728 non-null   object
 2   Maintenance_Price 1728 non-null   object
 3   No_of_Doors       1728 non-null   int64
 4   Person_Capacity   1728 non-null   int64
 5   Size_of_Luggage   1728 non-null   object
 6   Safety            1728 non-null   object
 7   Fuel_Type         1728 non-null   object
 8   Horse_power       1728 non-null   int64
 9   Car_Acceptability 1728 non-null   object
dtype: int64, object: 7
memory usage: 135.1 KB
```

Encoding Categorical Variables

In [3]:
```
# Check the cetegorical columns
print(df.columns)

Index(['Serial number', 'Buying_Price', 'Maintenance_Price', 'No_of_Doors',
       'Person_Capacity', 'Size_of_Luggage', 'Safety', 'Fuel_Type',
       'Horse_power', 'Car_Acceptability'],
      dtype='object')
```

In [4]:
```
# Update the get_dummies step to exclude Maintenance_Price and Car_Acceptability
df_encoded = pd.get_dummies(df, columns=['Buying_Price',
                                         'No_of_Doors', 'Person_Capacity',
                                         'Fuel_Type', 'Safety', 'Size_of_Luggage'],
                            drop_first=True)

# Show the encoded dataset
df_encoded.head()
```

Out[4]:

	Serial number	Maintenance_Price	Horse_power	Car_Acceptability	Buying_Price_low	Buying_Price_med	Buying_Price_high	No_of_Doors_3	No_of_Doors_4	No_o
0	0	vhigh	238	unacc	False	False	True	True	False	False
1	1	vhigh	232	unacc	False	False	True	True	True	False
2	2	vhigh	285	unacc	False	False	True	True	False	True
3	3	vhigh	321	unacc	False	False	True	True	True	False
4	4	vhigh	175	unacc	False	False	True	True	False	False

3. Exploratory Data Analysis (EDA)

Descriptive Statistics

In [5]:
```
# Summary statistics
df.describe()
```

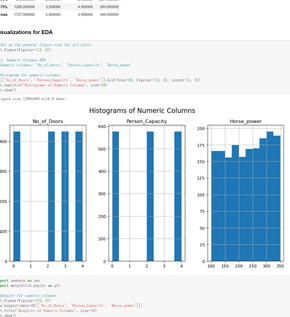
Out[5]:

	Serial number	No_of_Doors	Person_Capacity	Horse_power
count	1728.000000	1728.000000	1728.000000	1728.000000
mean	863.500000	2.250000	2.000000	226.416088
std	498.979949	1.479449	1.413466	72.945798
min	0.000000	0.000000	0.000000	100.000000
25%	431.750000	1.500000	0.500000	198.000000
50%	863.500000	2.500000	2.000000	231.000000
75%	1295.250000	3.250000	4.000000	255.000000
max	1727.000000	4.000000	4.000000	349.000000

Visualizations for EDA

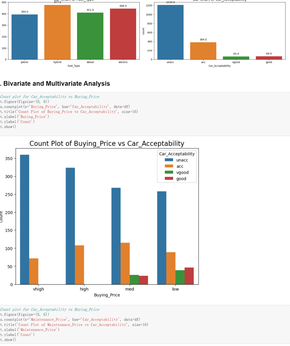
In [23]:
```
# Set up the general figure size for all plots
plt.figure(figsize=(12, 6))

# 1. Numeric Columns Histograms
# Numeric columns: 'No_of_Doors', 'Person_Capacity', 'Horse_power'
numeric_columns = ['No_of_Doors', 'Person_Capacity', 'Horse_power']
df[numeric_columns].hist(bins=10, figsize=(12, 6), layout=(1, 3))
plt.suptitle('Histograms of Numeric Columns', size=16)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```

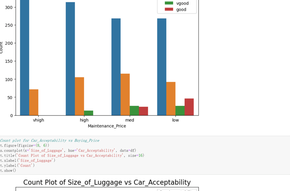


In [24]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# Boxplot for numeric columns
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['No_of_Doors', 'Person_Capacity', 'Horse_power']])
plt.title('Boxplots of Numeric Columns', size=16)
plt.show()
```

In [25]:
```
# Boxplot "Horse_power" in numeric
df['Horse_power'] = pd.to_numeric(df['Horse_power'], errors='coerce')

# Create a histogram to visualize the distribution of "Horse_power"
plt.figure(figsize=(10, 6))
sns.histplot(df['Horse_power'], kde=True, bins=30)
plt.title('Histogram of Horse Power')
plt.xlabel('Horse Power')
plt.ylabel('Frequency')
plt.show()
```



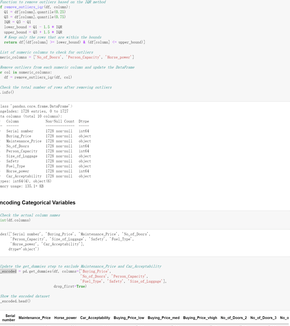
In [26]:
```
# 2. Categorical Column EDA
# Categorical columns: 'Buying_Price', 'Maintenance_Price', 'Size_of_Luggage', 'Safety', 'Fuel_Type', 'Car_Acceptability'

# Bar charts for categorical columns
# Define your categorical columns
categorical_columns = ['Buying_Price', 'Maintenance_Price', 'Size_of_Luggage', 'Safety', 'Fuel_Type', 'Car_Acceptability']

# Plot count plots for each categorical column and add annotations
plt.figure(figsize=(16, 12))
for i, column in enumerate(categorical_columns):
    plt.subplot(3, 2, i + 1)
    ax = sns.countplot(x=column, data=df)  # Draw the countplot
    plt.title('Bar Chart of {}', size=14)

    # Add annotations on top of each bar
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width()/ 2., p.get_height()),
                    ha='center', va='center', xytext=(0, 10), textcoords='offset points')

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



4. Bivariate and Multivariate Analysis

In [62]:
```
# Count plot for Car_Acceptability vs Buying_Price
plt.figure(figsize=(9, 6))
sns.countplot(x='Buying_Price', hue='Car_Acceptability', data=df)
plt.title('Count Plot of Buying_Price vs Car_Acceptability', size=18)
plt.xlabel('Buying_Price')
plt.ylabel('Count')
plt.show()
```

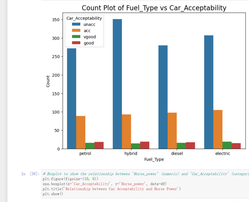

In [64]:
```
# Count plot for Car_Acceptability vs Maintenance_Price
plt.figure(figsize=(9, 6))
sns.countplot(x='Maintenance_Price', hue='Car_Acceptability', data=df)
plt.title('Count Plot of Maintenance_Price vs Car_Acceptability', size=18)
plt.xlabel('Maintenance_Price')
plt.ylabel('Count')
plt.show()
```



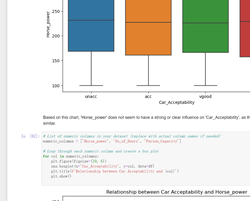
In [66]:
```
# Count plot for Car_Acceptability vs Buying_Price
plt.figure(figsize=(9, 6))
sns.countplot(x='Size_of_Luggage', hue='Car_Acceptability', data=df)
plt.title('Count Plot of Size_of_Luggage vs Car_Acceptability', size=18)
plt.xlabel('Size_of_Luggage')
plt.ylabel('Count')
plt.show()
```

```
# Count plot for Car_Acceptability vs Boot_Space
plt.figure(figsize=(6, 6))
sns.countplot(x='Boot_Space', hue='Car_Acceptability', data=df)
plt.title('Count Plot of Safety vs Car_Acceptability', size=18)
plt.xlabel('Safety')
plt.ylabel('Count')
plt.show()
```

```
# Count plot for Car_Acceptability vs Buying_Price
plt.figure(figsize=(6, 6))
sns.countplot(x='Fuel_Type', hue='Car_Acceptability', data=df)
plt.title('Count Plot of Fuel_Type vs Car_Acceptability', size=18)
plt.xlabel('Fuel_Type')
plt.ylabel('Count')
plt.show()
```

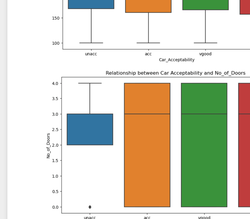


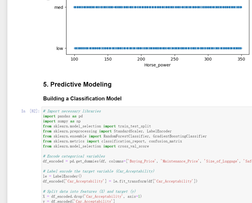

```
# Bivariate analysis
plt.figure(figsize=(6, 6))
sns.boxplot(x='Car_Acceptability', y='Horse_power', data=df)
plt.title('Relationship between Car Acceptability and Horse Power')
plt.show()
```



Based on this chart, 'Horse_power' does not seem to have a strong or clear influence on 'Car_Acceptability', as the distributions across the categories are quite similar.

```
# List of numeric columns in your dataset (replace with actual column names if needed)
numeric_columns = ['Horse_power', 'No_of_Doors', 'Person_Capacity']

# Loop through each numeric column and create a box plot
for col in numeric_columns:
    plt.figure(figsize=(6, 6))
    sns.boxplot(x='Car_Acceptability', y=col, data=df)
    plt.title('Relationship between Car Acceptability and {}'.
    plt.show()
```

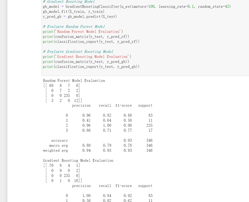




Cars that are classified as unacceptable tend to have fewer doors (median around 2-3), while cars classified as acceptable, very good, or good almost universally have 4 doors.

Specifically, cars that are classified as unacceptable tend to have fewer doors and a lower person capacity, while cars in the acceptable, good, and very good categories tend to have 4 doors and a person capacity of 4.

```
# Scatter plot
plt.figure(figsize=(6, 6))
sns.scatterplot(x='Horse_power', y='Buying_Price', data=df)
plt.title('Scatterplot of Horsepower and Buying Price')
plt.show()
```



5. Predictive Modeling

Building a Classification Model

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score

# Encode categorical variables
df_encoded = pd.get_dummies(df, columns=['Buying_Price', 'Maintenance_Price', 'Size_of_Luggage', 'Safety', 'Fuel_Type'], drop_first=True)

# Label encode the target variable (Car_Acceptability)
le = LabelEncoder()
df_encoded['Car_Acceptability'] = le.fit_transform(df['Car_Acceptability'])

# Split data into features (X) and target (y)
X = df_encoded.drop('Car_Acceptability', axis=1)
y = df_encoded['Car_Acceptability']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale numeric variables (optional if tree models are used)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Gradient Boosting Model
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)

# Evaluate Random Forest Model
print("Random Forest Model Evaluation:")
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))

# Evaluate Gradient Boosting Model
print("Gradient Boosting Model Evaluation:")
print(confusion_matrix(y_test, y_pred_gb))
print(classification_report(y_test, y_pred_gb))
```

```
Random Forest Model Evaluation
[[ 68   0   0   0]
 [  0   7   2   2]
 [  0   2 200   4]
 [  0   2   8 121]]
              precision    recall  f1-score   support

           0       0.96      0.92      0.93        68
           1       0.64      0.64      0.64        11
           2       0.94      0.98      0.96       206
           3       0.96      0.71      0.77       131

    accuracy                           0.93       416
   macro avg       0.90      0.79      0.78       416
weighted avg       0.94      0.93      0.93       416

Gradient Boosting Model Evaluation
[[ 70   0   0   0]
 [  0   9   2   2]
 [  0   2 200   4]
 [  0   0   8 121]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        70
           1       0.50      0.82      0.62        11
           2       0.95      0.97      0.96       206
           3       0.94      0.94      0.94       129

    accuracy                           0.96       416
   macro avg       0.85      0.90      0.88       416
weighted avg       0.96      0.95      0.96       416
```

```
from sklearn.model_selection import cross_val_score

# Cross-validation scores for Random Forest
rf_cv_scores = cross_val_score(rf_model, X, y, cv=5)
print("Random Forest Cross-Validation Scores:", rf_cv_scores)
print("Mean Cross-Validation Score (Random Forest):", np.mean(rf_cv_scores))

# Cross-validation scores for Gradient Boosting
gb_cv_scores = cross_val_score(gb_model, X, y, cv=5)
print("Gradient Boosting Cross-Validation Scores:", gb_cv_scores)
print("Mean Cross-Validation Score (Gradient Boosting):", np.mean(gb_cv_scores))
```

```
Random Forest Cross-Validation Scores: [0.99038461 0.93509615 0.98557692 0.98076923 0.98461538]
Mean Cross-Validation Score (Random Forest): 0.9792788461538462
Gradient Boosting Cross-Validation Scores: [0.99519231 0.98076923 0.98557692 0.98076923 0.98269231]
Mean Cross-Validation Score (Gradient Boosting): 0.9849759615384738
```

```
# Feature importance from Random Forest
feature_importances = rf_model.feature_importances_
feature_names = X.columns

# Display feature importance
print("Feature ranking:")
for i, feature in enumerate(X.columns):
    print(f"{i+1}. {feature} ({feature_names[indices[i]]: {importances[indices[i]]:.4f}")

# Plot feature importance
import matplotlib.pyplot as plt
importances = rf_model.feature_importances_
plt.title("Feature Importance")
plt.bar(range(X.shape[1]), importances[indices], color="r", align="center")
plt.xticks(range(X.shape[1]), feature_names[indices], rotation=90)
plt.xlim(-1, X.shape[1])
plt.show()
```

It suggests that the ability of a car to accommodate more people and having lower safety ratings are major factors that influence the classification of 'Car_Acceptability'.

'No_of_Doors' and 'Safety_med' are the next most influential features, which makes sense considering that these are fundamental characteristics that could affect a consumer's perception of car quality. Surprisingly, 'Maintenance_Price' and 'Buying_Price' (at different levels) have relatively low importance. This suggests that, in the dataset, price might not be as critical in determining the acceptability of a car compared to other features like safety and capacity.

Conclusions: The key factors driving car acceptability seem to be person capacity, safety ratings, and horse power. Price (both buying and maintenance) and fuel type do not play a significant role in the acceptability of the cars in this dataset, which could be counterintuitive but is likely dependent on the context of the data.

Model Further Improvement

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Assume the data frame 'df' is already in memory and the target variable (y) is in 'df'
X = df_encoded.drop(columns=['
    'Unit_number',                  # Exclude identifier
    'Car_Acceptability',            # Exclude the target variable from the features
    'Fuel_Type_petrol',             # Low-importance Fuel Type feature
    'Fuel_Type_electric',           # Low-importance Fuel Type feature
    'Fuel_Type_hybrid',             # Low-importance Fuel Type feature
    'Buying_Price_low',             # Low-importance Buying Price feature
    'Buying_Price_med',             # Low-importance Buying Price feature
    'Buying_Price_vhigh',           # Low-importance Buying Price feature
    'Maintenance_Price_med',        # Low-importance Maintenance Price feature
    'Maintenance_Price_low',        # Low-importance Maintenance Price feature
    'Maintenance_Price_vhigh'       # Low-importance Maintenance Price feature
], )

# Target variable (Car_Acceptability)
y = df_encoded['Car_Acceptability']

# Encode the target variable if necessary
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))

# Check the new feature importance after removing the low-importance variables
feature_importance = rf.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance})

# Sort and display feature importance
print(feature_importance_df.sort_values(by='Importance', ascending=False))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.94       195
           1       0.56      0.52      0.54        23
           2       0.96      0.92      0.94       596
           3       0.24      0.27      0.25        29

    accuracy                           0.89       843
   macro avg       0.67      0.66      0.67       843
weighted avg       0.90      0.89      0.89       843

                 Feature  Importance
0          Lower_Boot_Space    0.151
1          Person_Capacity    0.147
2                Horse_power    0.142
3                 Safety_med    0.120
4               No_of_Doors    0.117
5  Size_of_Luggage_small    0.080
6  Maintenance_Price_high    0.037
7   Maintenance_Price_low    0.036
8     Buying_Price_med    0.034
9     Buying_Price_low    0.031
10  Maintenance_Price_med    0.031
11   Maintenance_Price_vhigh    0.030
12          Safety_high    0.029
13     Fuel_Type_diesel    0.028
14     Fuel_Type_electric    0.027
15     Buying_Price_vhigh    0.026
16     Fuel_Type_petrol    0.025
17     Fuel_Type_hybrid    0.024
```

```
   macro avg       0.42      0.40      0.41       519
weighted avg       0.74      0.76      0.75       519

                 Feature  Importance
2            Horse_power    0.432168
1         Person_Capacity    0.210405
5             Safety_low    0.173481
0            No_of_Doors    0.088748
6             Safety_med    0.036776
4  Size_of_Luggage_small    0.034610
3    Size_of_Luggage_med    0.023811
```

This indicates that the model's performance has decreased after removing the low-importance variables. This could mean that some of the variables that were removed (like price or fuel type) may still have contributed to improving the classification, even if their individual importance was low.

Horse_power has become the most important feature with an importance of 0.43. This suggests that engine power plays a significant role in determining car acceptability.

Person_Capacity and Safety_low are also highly important, contributing 0.21 and 0.17, respectively.

Other variables, such as No_of_Doors, Safety_med, and Size_of_Luggage, have lower importance but still contribute to the model.

In [ ]: