Introduction

This report presents a comprehensive analysis of a dataset containing details about used cars, including attributes such as manufacturing year, selling price, kilometers driven, fuel type, seller type, transmission type, ownership history, mileage, engine capacity, maximum power, and seating capacity. The objective is to explore and understand the factors influencing the selling price of used cars. Through data cleaning, transformation, and exploratory data analysis, key trends and correlations are identified, providing insights into the used car market. This analysis serves as a foundation for predictive modeling and further statistical analysis to better understand price determinants in the used car market.

Importing Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

Dataset Used

```
df
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0

Data Overview

The dataset contains information about used cars, with attributes such as the car's name, manufacturing year, selling price, kilometers driven, fuel type, seller type, transmission type, ownership history, mileage, engine capacity, maximum power, and number of seats.

Data Cleaning and Preparation

- **Removing Null Values**: We identified and removed null values present in the mileage, engine, max power, and seats columns.
- **Handling Duplicates**: We detected and eliminated duplicate records to ensure the dataset's integrity.

```
df.drop(columns=['torque'],inplace = True)
#checking null values
df.isnull().sum()
```

```
name               0
year               0
selling_price      0
km_driven          0
fuel               0
seller_type        0
transmission       0
owner              0
mileage          221
engine           221
max_power        215
seats            221
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           8128 non-null   object
 1   year           8128 non-null   int64
 2   selling_price  8128 non-null   int64
 3   km_driven      8128 non-null   int64
 4   fuel           8128 non-null   object
 5   seller_type    8128 non-null   object
 6   transmission   8128 non-null   object
 7   owner          8128 non-null   object
 8   mileage        7907 non-null   object
 9   engine         7907 non-null   object
 10  max_power      7913 non-null   object
 11  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(8)
memory usage: 762.1+ KB
```

```
#removing null values
df.dropna(inplace=True)
```

- **Extracting Useful Features**:
  - **No. of Years Used**: A new column, **"no_year",** was created to represent the number of years a car has been used, calculated by subtracting the manufacturing year from the current year (2024).
- **Encoding Categorical Variables**:
  - Fuel type, seller type, transmission type, owner type, and car brand names were encoded as numerical values for better analysis and model training.

```
#creating a new column No_year(Number of years car used)

current_datetime=datetime.now()
current_year=current_datetime.year
print(current_year)
df["no_year"]=current_year-df["year"]
```

```
2024
```

```
# Drop column Year

df.drop(['year'],axis=1,inplace=True)
```

```
# Fuel_Type column :'Diesel':1,'Petrol':2, 'LPG':3 ,'CNG':4

print(df['fuel'].unique())
df['fuel'].replace({'Diesel':1,'Petrol':2, 'LPG':3 ,'CNG':4},inplace=True)
```

```
['Diesel' 'Petrol' 'LPG' 'CNG']
```

```
# Selling_type column :'Individual':1,'Dealer':2,'Trustmark Dealer':3

print(df['seller_type'].unique())
df['seller_type'].replace({'Individual':1,'Dealer':2,'Trustmark Dealer':3},inplace=True)
```

```
['Individual' 'Dealer' 'Trustmark Dealer']
```

```
# Transmission column :replacing manual with 1 and automatic with 2

print(df['transmission'].unique())
df['transmission'].replace({'Manual':1,'Automatic':2},inplace=True)
```

```
['Manual' 'Automatic']
```

```
#OWNER column: 'First Owner':1 ,'Second Owner':2 ,'Third Owner':3,'Fourth & Above Owner':4,'Test Drive Car':5

print(df['owner'].unique())
df['owner'].replace({'First Owner':1 ,'Second Owner':2 ,'Third Owner':3,'Fourth & Above Owner':4,'Test Drive Car':5},inplace=True
```

```
['First Owner' 'Second Owner' 'Third Owner' 'Fourth & Above Owner'
 'Test Drive Car']
```

Data Transformation

- **Modifying Mileage, Engine, and Max Power**: These columns originally contained text values with units. We extracted the numerical part of these values for analysis.
- **Brand Name Extraction**: The car brand was extracted from the name column and encoded into numerical values.

function to extract value from mileage, engine and max_power

```
def clean_data(value):
    value=str(value)
    value = value.split(' ')[0]
    value = value.strip()
    if value == '':
        return np.nan
    else:

        return float(value)
```

```
df['mileage'] = df['mileage'].apply(clean_data)
```

```
df['engine'] = df['engine'].apply(clean_data)
```

```
df['max_power'] = df['max_power'].apply(clean_data)
```

```
df.head(3)
```

	name	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	no_year
0	1	450000	145500	1	1	1	1	23.40	1248.0	74.00	5.0	10
1	2	370000	120000	1	1	1	2	21.14	1498.0	103.52	5.0	10
2	3	158000	140000	2	1	1	3	17.70	1497.0	78.00	5.0	18

Exploratory Data Analysis

- **Distributions**: We analyzed the distribution of key numerical features (selling price, kilometers driven, mileage, engine capacity, max power, and number of years used) using distribution plots.
  - **Selling Price**: The distribution is right-skewed, indicating that most cars are sold at a lower price.
  - **Kilometers Driven**: This distribution is also right-skewed, with most cars having fewer kilometers driven.
  - **Mileage**: There is a wide range in mileage, but a significant number of cars offer moderate mileage.
  - **Engine Capacity**: The distribution shows a concentration of cars with smaller engine sizes.
  - **Max Power**: Most cars have lower maximum power values.
  - **Number of Years Used**: The distribution shows a mix of newer and older cars.
  - **Box plots**: Box plots were used to identify outliers in the dataset.Significant outliers were present in the selling price, kilometers driven, mileage, engine capacity, and max power columns.

Correlation Analysis - A correlation matrix was generated to understand the relationships between numerical variables.

- **Positive Correlations**:
  - Selling price is positively correlated with engine capacity and max power.
  - Engine capacity and max power also show a strong positive correlation.
- **Negative Correlations**:
  - Selling price is negatively correlated with the number of years used, indicating that older cars tend to sell for less. Mileage has a negative correlation with engine capacity and max power.

BEST FEATURES

```
bestfeatures = SelectKBest(score_func=f_regression, k=10)

# Fit SelectKBest to your data
fit = bestfeatures.fit(x, y)

# Create DataFrame to store feature scores
dfscores = pd.DataFrame(fit.scores_)

# Create DataFrame to store feature names
dfcolumns = pd.DataFrame(x.columns)

# Concatenate feature names and scores into a single DataFrame for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)

# Rename the columns
featureScores.columns = ['Specs', 'Score']

# Print the top 10 features with the highest scores
print(featureScores.nlargest(10, 'Score'))
```

```
           Specs        Score
8      max_power  6181.382158
4   transmission  1857.986653
7         engine  1637.476893
10       no_year  1500.224991
0           name   453.852706
3    seller_type   449.284164
2           fuel   430.886346
5          owner   307.528682
1      km_driven   179.295418
9          seats   173.112006
```

Correlation Matrix

```
correlation_matrix = df.corr()
correlation_matrix
```

	name	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	no_year
name	1.000000	0.251613	0.049784	-0.277496	0.047916	0.174458	-0.013379	-0.273961	0.369460	0.348463	0.155834	-0.033978
selling_price	0.251613	1.000000	-0.161265	-0.245558	0.250423	0.465538	-0.209265	-0.108655	0.442772	0.692323	0.158531	-0.427335
km_driven	0.049784	-0.161265	1.000000	-0.252491	-0.126336	-0.118965	0.252205	-0.196419	0.253460	0.041770	0.207890	0.387918
fuel	-0.277496	-0.245558	-0.252491	1.000000	-0.019724	0.005210	-0.012138	-0.035961	-0.510633	-0.328039	-0.343668	0.043564
seller_type	0.047916	0.250423	-0.126336	-0.019724	1.000000	0.213725	-0.151667	0.001552	0.065629	0.187339	-0.040726	-0.148137
transmission	0.174458	0.465538	-0.118965	0.005210	0.213725	1.000000	-0.076854	-0.173667	0.219526	0.441681	-0.019314	-0.143997
owner	-0.013379	-0.209265	0.252205	-0.012138	-0.151667	-0.076854	1.000000	-0.188624	0.033741	-0.052018	0.007649	0.480096
mileage	-0.273961	-0.108655	-0.196419	-0.035961	0.001552	-0.173667	-0.188624	1.000000	-0.579153	-0.378609	-0.459188	-0.366048
engine	0.369460	0.442772	0.253460	-0.510633	0.065629	0.219526	0.033741	-0.579153	1.000000	0.683506	0.658711	0.019763
max_power	0.348463	0.692323	0.041770	-0.328039	0.187339	0.441681	-0.052018	-0.378609	0.683506	1.000000	0.259028	-0.159889
seats	0.155834	0.158531	0.207890	-0.343668	-0.040726	-0.019314	0.007649	-0.459188	0.658711	0.259028	1.000000	-0.025021
no_year	-0.033978	-0.427335	0.387918	0.043564	-0.148137	-0.143997	0.480096	-0.366048	0.019763	-0.159889	-0.025021	1.000000

Insights

- **Factors Influencing Car Prices**: Engine capacity, max power, and the age of the car are significant factors affecting the selling price. Cars with higher engine capacity and max power tend to sell for more, while older cars generally fetch lower prices.

- **Model Training**: Implemented Random Forest regression model to predict the selling price and evaluate their performance using metrics like Mean Squared Error (MSE), Mean Absolute Error(MAE) and R-squared (R^2) values.

Random Forest Regressor Model

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score


# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y_scaled, test_size=0.2, random_state=42)

# Create and train the Random Forest Regression model

rf_model = RandomForestRegressor(n_estimators=10, random_state=45)
rf_model.fit(x_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(x_test)

# Evaluate the model
mae=metrics.mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MAE score:{mae}")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

```
MAE score:0.1530463136374124
Mean Squared Error: 0.08372642537719306
R^2 Score: 0.8954275160815178
```

MODEL INPUT

```
columns = ['name', 'km_driven', 'fuel', 'seller_type', 'transmission', 'owner', 'mileage', 'engine', 'max_power', 'seats', 'no_ye
dat = pd.DataFrame([[4, 90000, 1, 1, 1, 1, 19.67, 1582.0, 126.20, 5.0, 7]], columns=columns)
```

SCALE THE INPUT DATA AND INVERSEING THE OUTPUT BACK TO ORIGINAL SCALE

```
# Ensure the new data for prediction has the same columns as x
dat = dat[x.columns]

# Scale the new data
dat_scaled = scaler_x.transform(dat)

# Make predictions using the model
predictions_scaled = rf_model.predict(dat_scaled)

# Inverse transform the predictions to get them back to the original scale
predictions = scaler_y.inverse_transform(predictions_scaled.reshape(-1, 1))

# Print the predictions in the original scale
print("Predictions (original scale):", predictions)
```

```
Predictions (original scale): [[1155399.9]]
```