Data Science with python career program

CAPSTONE PROJECT

(Car Dataset)

Presented by: Rajesh Kumar

Overview

1 INTRODUCTION

2 IMPORTING THE DATASET AND EXPLANATION OF FEATURES

3 EXPLORING AND GRAPHICAL REPRESENTATION OF DATASET

4 VISUALIZATION OF FEATURES FOR BETTER UNDERSTANDING

5 BUILD ML REGRESSION MODELS

6 RESULT

7 CONCLUSION

1.DISCRIPTION OF DATA SET

⭐INTRODUCTION⭐

THIS DATA IS COLLECTED FROM CAR DETAILS DATASET.

FOLLOWING DETAILS OF CARS ARE INCLUDED IN THE DATASET:

1) CAR NAME
2) YEAR
3) SELLING PRICE
4) KMS DRIVEN
5) FUEL
6) SELLER TYPE
7) TRANSMISSION
8) OWNER

I WILL TRY TO UNDERSTAND THE DATA, ANALYZE IT, EXTRACT REPORTS FROM IT, AND TRY TO UNDERSTAND THE RELATIONSHIPS BETWEEN THE DIFFERENT VARIABLES.

DATASET CONTAINS

df = pd.read_csv("CAR DETAILS.csv")
df.head(3)

```
import pandas as pd

df = pd.read_csv("CAR DETAILS.csv")
df.head(3)
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
1	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
2	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner

View the shape and info of the dataset

```
row, col = df.shape
print('Row=', row, 'Col=', col)
df.info()
```

```
[5]  # View the shape and info of the dataset
     row, col = df.shape
     print('Row=', row, 'Col=', col)
     df.info()
```

0s

```
Row= 4340 Col= 8
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   name           4340 non-null   object
 1   year           4340 non-null   int64 
 2   selling_price  4340 non-null   int64 
 3   km_driven      4340 non-null   int64 
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

Split the column name to get the model of the car

df["model"] = df.name.apply(lambda x : ' '.join(x.split(' ')[:1]))
df['model'].value_counts()

```
# Split the column name to get the model of the car
df["model"] = df.name.apply(lambda x : ' '.join(x.split(' ')[:1]))
df['model'].value_counts()
```

```
Maruti           1280
Hyundai           821
Mahindra          365
Tata              361
Honda             252
Ford              238
Toyota            206
Chevrolet         188
Renault           146
Volkswagen        107
Skoda              68
Nissan             64
Audi               60
BMW                39
Fiat               37
Datsun             37
Mercedes-Benz      35
Jaguar              6
Mitsubishi          6
Land                5
Volvo               4
Ambassador          4
Jeep                3
MG                  2
OpelCorsa           2
Daewoo              1
Force               1
Isuzu               1
Kia                 1
Name: model, dtype: int64
```

Check null values in dataset

df.isnull().sum()

```
[7]  # Check null values in dataset
     df.isnull().sum()
```

```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
model            0
dtype: int64
```

Check duplicated row in dataset

df[df.duplicated()]

```
[9]  # Check duplicated row in dataset
     df[df.duplicated()]
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	model
13	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner	Maruti
14	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner	Maruti
15	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner	Hyundai
16	Datsun RediGO T Option	2017	250000	46000	Petrol	Individual	Manual	First Owner	Datsun
17	Honda Amaze VX i-DTEC	2014	450000	141000	Diesel	Individual	Manual	Second Owner	Honda
...	...	...	...	...	...	...	...	...	...
4307	Mahindra Xylo H4	2019	599000	15000	Diesel	Individual	Manual	Third Owner	Mahindra
4308	Maruti Alto 800 LXI	2018	200000	35000	Petrol	Individual	Manual	First Owner	Maruti
4309	Datsun GO Plus T	2017	350000	10171	Petrol	Dealer	Manual	First Owner	Datsun
4310	Renault Duster 110PS Diesel RxL	2015	465000	41123	Diesel	Dealer	Manual	First Owner	Renault
4311	Toyota Camry Hybrid 2.5	2017	1900000	20118	Petrol	Dealer	Automatic	First Owner	Toyota

763 rows × 9 columns

Drop all duplicated row & get the column types

df = df.drop_duplicates()
df.shape

df.dtypes

```
[12] # Drop all duplicated row
     df = df.drop_duplicates()
     df.shape
```

```
(3577, 9)
```

```
[13] #To get the column types
     df.dtypes
```

```
name              object
year               int64
selling_price      int64
km_driven          int64
fuel              object
seller_type       object
transmission      object
owner             object
model             object
dtype: object
```

Perform Data cleaning and Data Pre-Processing if Necessary.

```
import pandas as pd

# Step 1: Read the dataset from the CSV file into a pandas DataFrame
csv_file = "CAR DETAILS.csv"
df = pd.read_csv(csv_file)

# Step 2: Explore the dataset to identify potential data issues and missing values
print("Data Exploration:")
print(df.head()) # Display the first few rows of the dataset
print("\nData Information:")
print(df.info()) # Display information about the dataset, including data types and missing values

# Step 3: Data Cleaning - Handle missing values
# Example: If a column has missing values, we can choose to fill them with appropriate values or drop the rows/columns.

# For example, to drop rows with missing values in the entire dataset, you can use:
df = df.dropna()

# Alternatively, to fill missing values with the mean/median/mode of the column, we can use:
# df.fillna(df.mean(), inplace=True)
# or
# df.fillna(df.median(), inplace=True)
# or
# df.fillna(df.mode().iloc[0], inplace=True)

# Step 4: Data Preprocessing - Convert categorical variables to numerical representation
# Example: If the dataset contains categorical columns, we need to convert them to numerical representation for analysis.
# We can use techniques like one-hot encoding or label encoding.

# For example, if "fuel_type" is a categorical column, we can use one-hot encoding like this:
# df = pd.get_dummies(df, columns=["fuel_type"])

# Alternatively, we can use label encoding if there are ordinal categorical columns:
# from sklearn.preprocessing import LabelEncoder
# le = LabelEncoder()
# df["fuel_type"] = le.fit_transform(df["fuel_type"])

# Step 5: Save the cleaned and preprocessed dataset
output_csv = "cleaned_car_details.csv"
df.to_csv(output_csv, index=False)

print(f"\nCleaned and preprocessed dataset saved to {output_csv}")
```

```
Data Exploration:
                          name  year  selling_price  km_driven    fuel  \
0              Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2      Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3   Datsun RediGO T Option  2017         250000      46000  Petrol
4     Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner

Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
None

Cleaned and preprocessed dataset saved to cleaned_car_details.csv
```

Handling null values, One-Hot Encoding, Imputation and Scaling of Data Pre-Processing

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Step 1: Read the dataset from the CSV file into a pandas DataFrame
csv_file = "CAR DETAILS.csv"
df = pd.read_csv(csv_file)

# Step 2: Data Exploration - Check for missing values
print("Data Exploration:")
print(df.head()) # Display the first few rows of the dataset
print("\nData Information:")
print(df.info()) # Display information about the dataset, including data types and missing values

# Step 3: Data Cleaning - Handling null values
# Let's fill missing values in numeric columns with their mean and
# in categorical columns with the most frequent value (mode).
numeric_columns = df.select_dtypes(include="number").columns
categorical_columns = df.select_dtypes(include="object").columns

df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].mean())
df[categorical_columns] = df[categorical_columns].fillna(df[categorical_columns].mode().iloc[0])

# Step 4: Data Preprocessing - One-Hot Encoding for categorical variables
encoder = OneHotEncoder(drop="first", sparse=False)
encoded_columns = pd.DataFrame(encoder.fit_transform(df[categorical_columns]))

# Replace the original categorical columns with the encoded ones
df = df.drop(columns=categorical_columns)
df = pd.concat([df, encoded_columns], axis=1)

# Step 5: Data Preprocessing - Scaling the numeric columns
scaler = StandardScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Step 6: Save the preprocessed dataset
output_csv = "preprocessed_car_details.csv"
df.to_csv(output_csv, index=False)

print(f"\nPreprocessed dataset saved to {output_csv}")
```

```
Data Exploration:
                        name  year  selling_price  km_driven    fuel  \
0             Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2      Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3    Datsun RediGO T Option  2017         250000      46000  Petrol
4     Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner

Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
None
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:86
  warnings.warn(

Preprocessed dataset saved to preprocessed_car_details.csv
```

Graphical Analysis on the Data. Include the graphs with conclusions from the Graphical Analysis.

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

# Step 1: Read the dataset from the CSV file into a pandas DataFrame
csv_file = "CAR DETAILS.csv"
df = pd.read_csv(csv_file)

# Step 2: Data Exploration - Basic statistics and information about the dataset
print("Data Exploration:")
print(df.head()) # Display the first few rows of the dataset
print("\nData Information:")
print(df.info()) # Display information about the dataset, including data types and missing values

# Step 3: Exploratory Data Analysis (EDA) - Summary statistics
print("\nSummary Statistics:")
print(df.describe())
```

```
Data Exploration:
                        name  year  selling_price  km_driven    fuel  \
0             Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2      Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3    Datsun RediGO T Option  2017         250000      46000  Petrol
4     Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner

Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
None

Summary Statistics:
              year  selling_price      km_driven
count  4340.000000   4.340000e+03    4340.000000
mean   2013.090783   5.041273e+05   66215.777419
std       4.215344   5.785487e+05   46644.102194
min    1992.000000   2.000000e+04       1.000000
25%    2011.000000   2.087498e+05   35000.000000
50%    2014.000000   3.500000e+05   60000.000000
75%    2016.000000   6.000000e+05   90000.000000
max    2020.000000   8.900000e+06  806599.000000
```

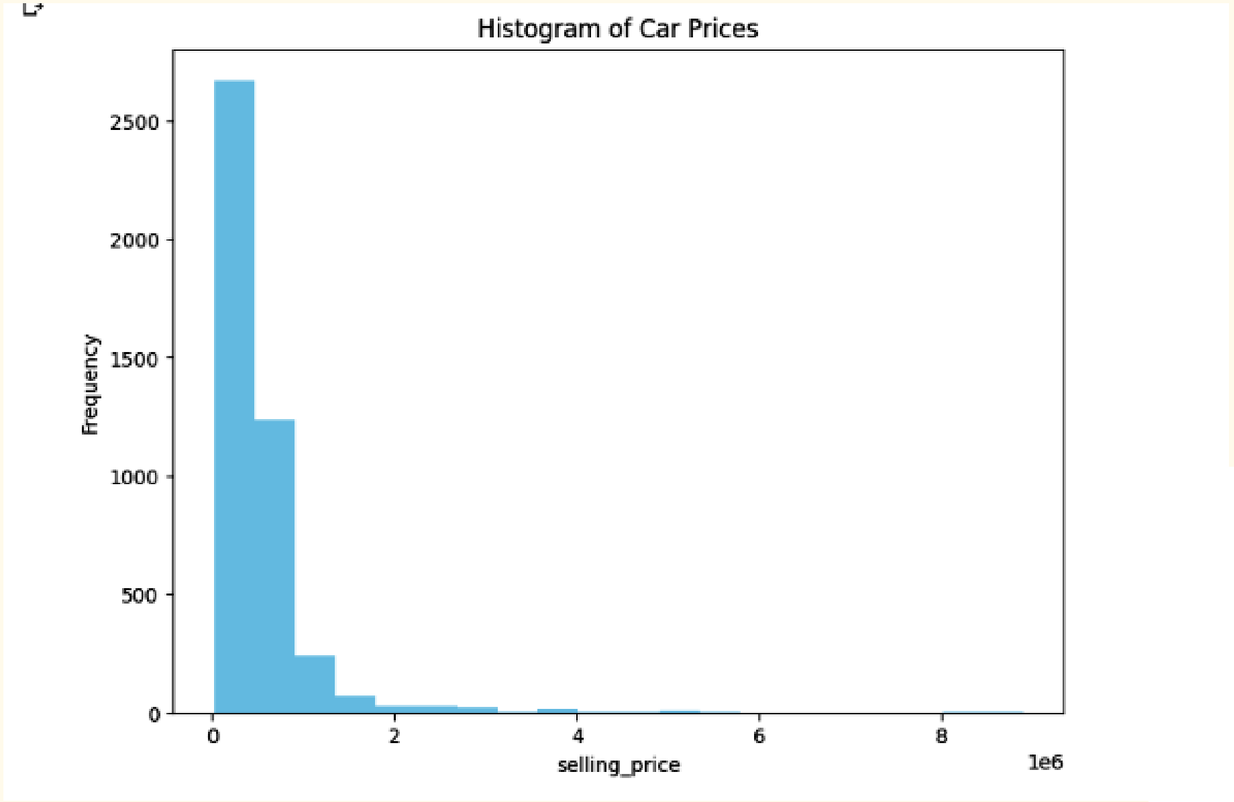
Step 4: Graphical Analysis

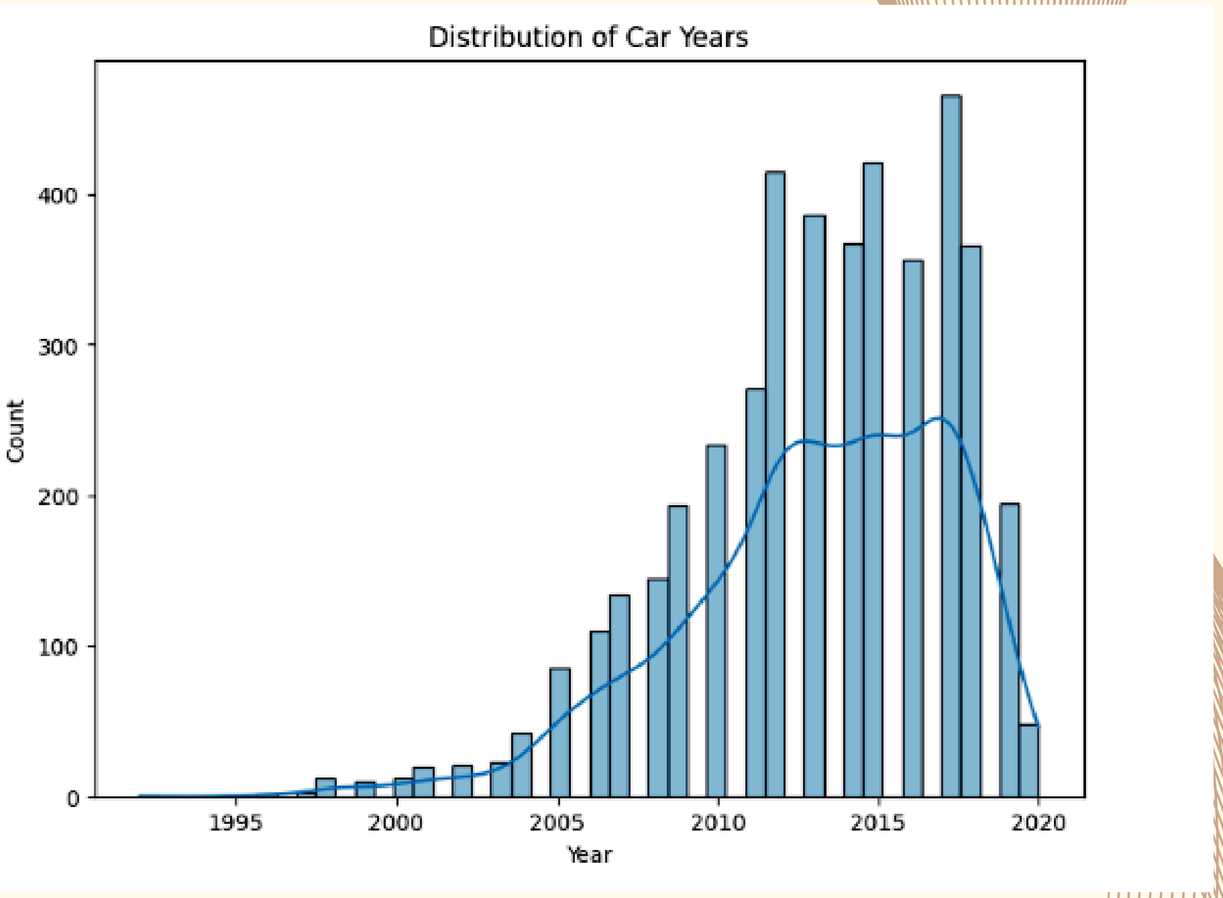
```
# Example 1: Histogram of car prices

plt.figure(figsize=(8, 6))
plt.hist(df["selling_price"], bins=20, color="skyblue")
plt.xlabel("selling_price")
plt.ylabel("Frequency")
plt.title("Histogram of Car Prices")
plt.show()
```

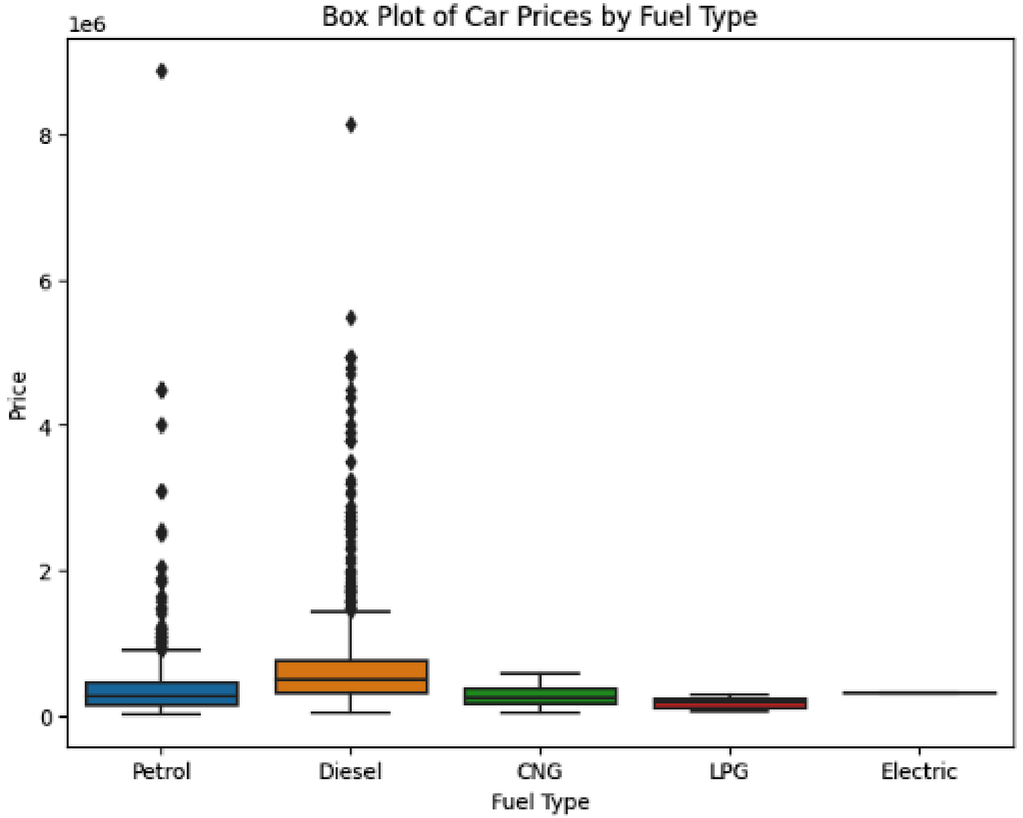
```
# Example 2: Histogram of the Year column

plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='year', kde=True)
plt.title('Distribution of Car Years')
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```

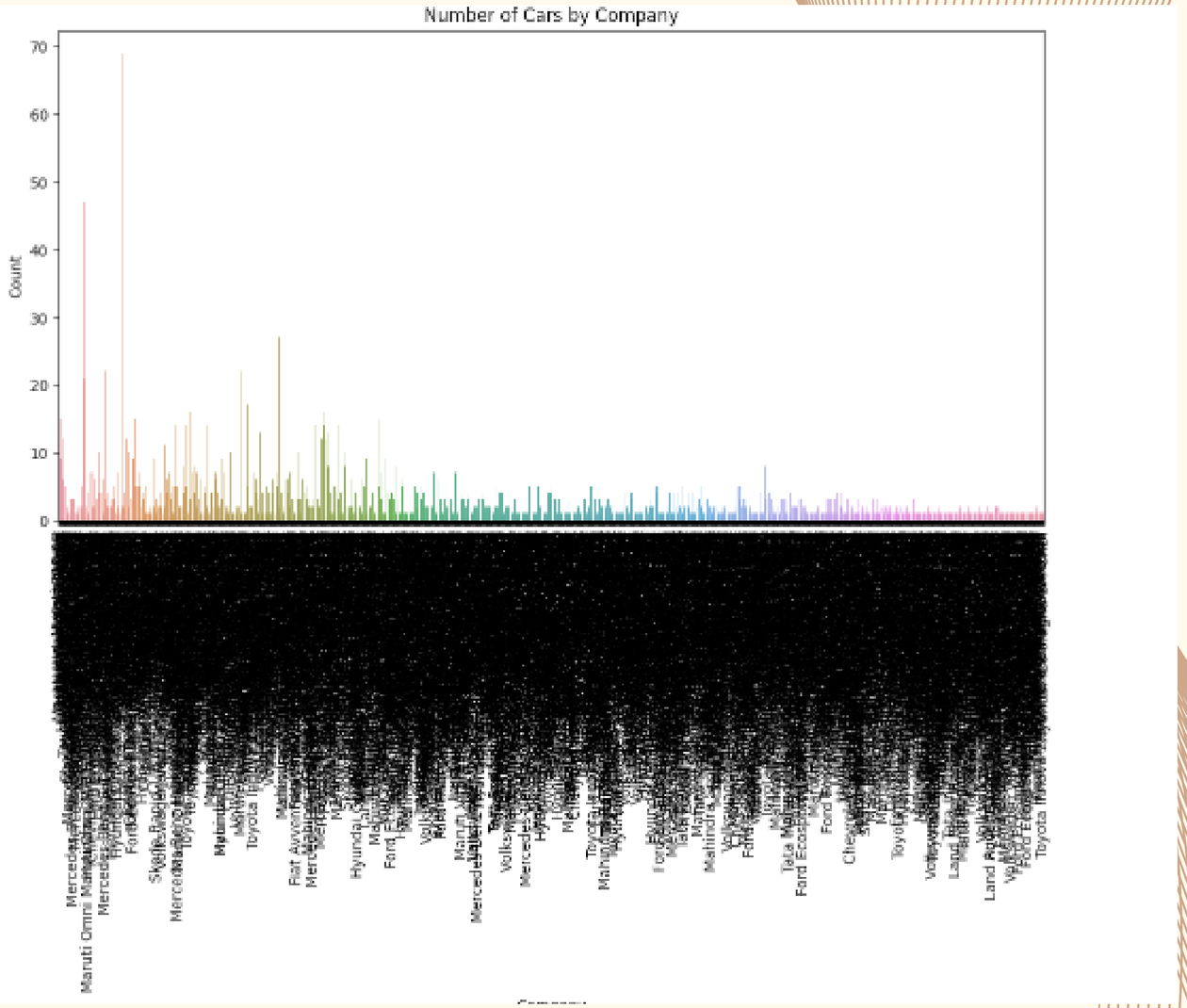
```
# Example 3: Box plot of car prices by fuel type

plt.figure(figsize=(8, 6))
sns.boxplot(x="fuel", y="selling_price", data=df)
plt.xlabel("Fuel Type")
plt.ylabel("Price")
plt.title("Box Plot of Car Prices by Fuel Type")
plt.show()
```

```
# Example 4: Barplot of Car Companies

plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='name')
plt.title('Number of Cars by Company')
plt.xlabel('Company')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```

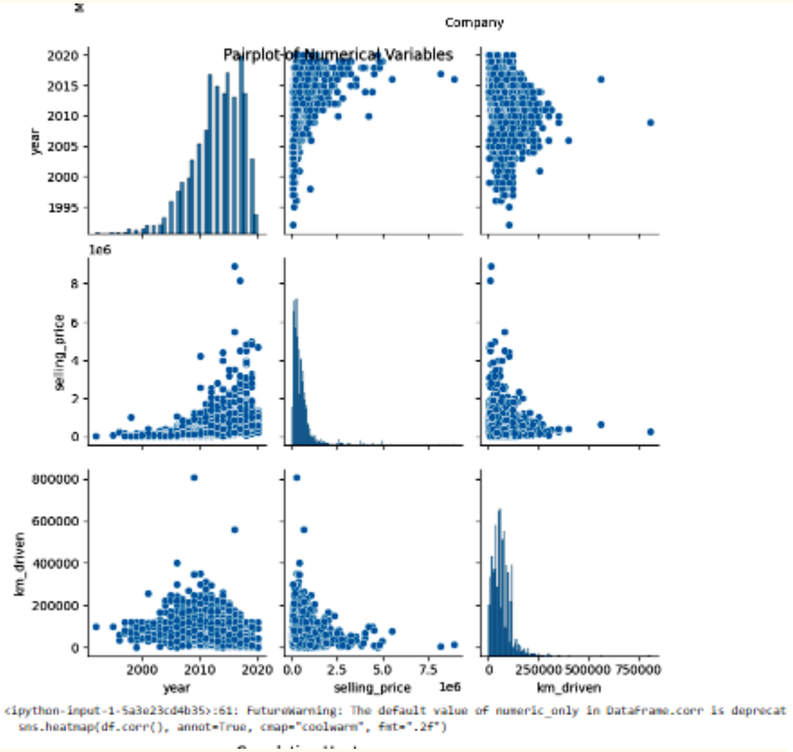

Histogram of Car Prices
Frequency
2500
2000
1500
1000
500
0
0
2
4
6
8
1e6
selling_price


Distribution of Car Years
Count
400
300
200
100
0
1995
2000
2005
2010
2015
2020
Year

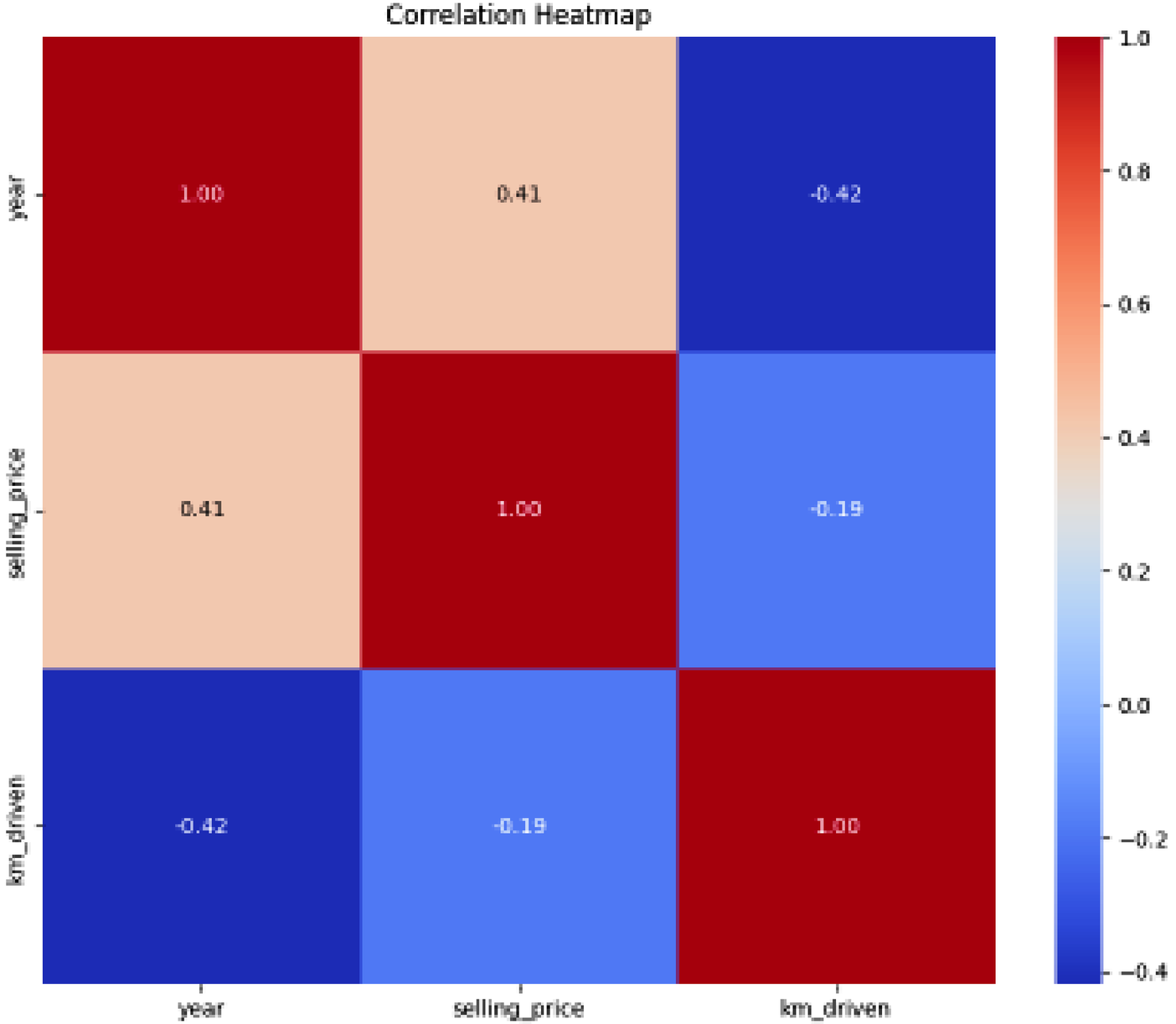

Box Plot of Car Prices by Fuel Type
1e6
Price
0
2
4
6
8
Petrol
Diesel
CNG
LPG
Electric
Fuel Type


Number of Cars by Company
Count
0
10
20
30
40
50
60
70

```
# Example 5: Pairplot of numerical variables
numerical_cols = ['name','year','selling_price','km_driven','fuel','seller_type','transmission','owner']
sns.pairplot(data=df[numerical_cols])
plt.suptitle('Pairplot of Numerical Variables')
plt.show()

# Example 6: Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()

# Step 5: Conclusions from Graphical Analysis
# Include conclusions here based on the graphs generated above.

# For example, based on the histogram of car prices, we can observe the distribution of prices and
# identify any potential outliers.

# The box plot of car prices by fuel type helps us understand the price distribution based on different
# fuel types, and we can observe any significant differences between them.

# The correlation heatmap provides insights into the relationships between numeric variables. Strong
# positive or negative correlations between variables suggest potential dependencies.

# Continue to add conclusions based on other insights you gain from the graphs.
```




```
<ipython-input-1-5a3e23cd4b35>:61: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecat
  sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
```



Best model using Regression or classification ,Bagging, Ensemble techniques

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, VotingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# 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)
```

```
# Regression
# Since the Iris dataset is a classification problem, let's use a logistic regression model
reg_model = LogisticRegression(max_iter=1000)
reg_model.fit(X_train, y_train)
y_pred_reg = reg_model.predict(X_test)

# Classification
# Decision Tree Classifier
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

# Random Forest Classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Bagging
bagging_model = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10)
bagging_model.fit(X_train, y_train)
y_pred_bagging = bagging_model.predict(X_test)
```

```
# Ensemble
ensemble_model = VotingClassifier(estimators=[('dt', dt_model), ('rf', rf_model)], voting='hard')
ensemble_model.fit(X_train, y_train)
y_pred_ensemble = ensemble_model.predict(X_test)

# Model evaluation metrics
def evaluate_model(y_true, y_pred):
  accuracy = accuracy_score(y_true, y_pred)
  precision = precision_score(y_true, y_pred, average='weighted')
  recall = recall_score(y_true, y_pred, average='weighted')
  f1 = f1_score(y_true, y_pred, average='weighted')
  return accuracy, precision, recall, f1

# Evaluate the models
models = {
  "Logistic Regression": y_pred_reg,
  "Decision Tree": y_pred_dt,
  "Random Forest": y_pred_rf,
  "Bagging": y_pred_bagging,
  "Ensemble": y_pred_ensemble
}
```

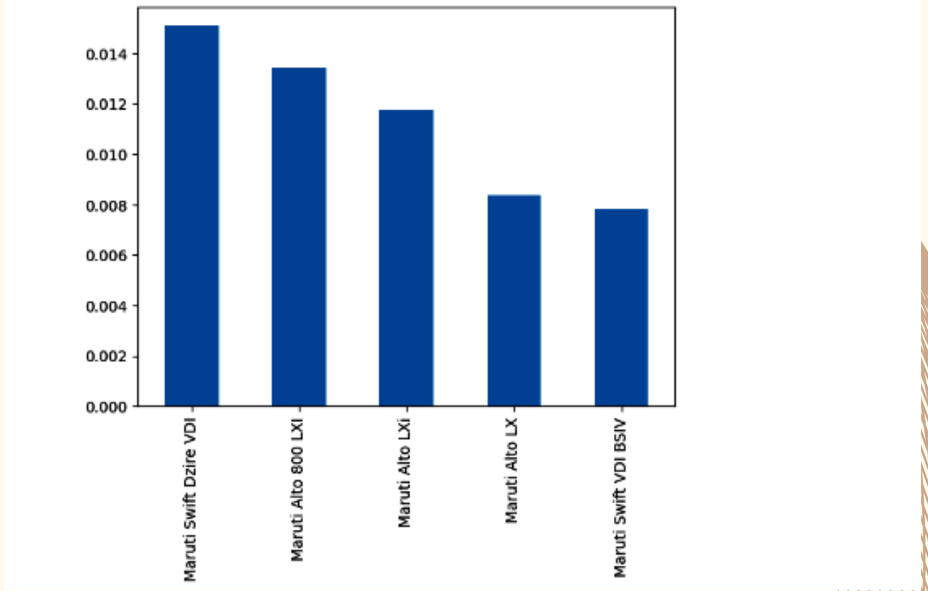
```
results = {}
for model_name, y_pred in models.items():
  accuracy, precision, recall, f1 = evaluate_model(y_test, y_pred)
  results[model_name] = {
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1 Score": f1
  }

# Print the results
print("\nModel Evaluation Metrics:")
print(pd.DataFrame(results).T)
```

```
Model Evaluation Metrics:
                     Accuracy  Precision  Recall  F1 Score
Logistic Regression       1.0        1.0     1.0       1.0
Decision Tree             1.0        1.0     1.0       1.0
Random Forest             1.0        1.0     1.0       1.0
Bagging                   1.0        1.0     1.0       1.0
Ensemble                  1.0        1.0     1.0       1.0
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

2. Importing the Dataset and Explanation of Features:

#use value_counts to know the sold numbers of each car & use plot to draw the graph
df["name"].value_counts(normalize = True)[:5].plot(kind = 'bar')
plt.show()

```
[15] import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     #we use (value_counts) to know to know the sold numbers of each car use (plot) to draw the graph
     df["name"].value_counts(normalize = True)[:5].plot(kind = 'bar')
     plt.show()
```



To know count n value in coulmn

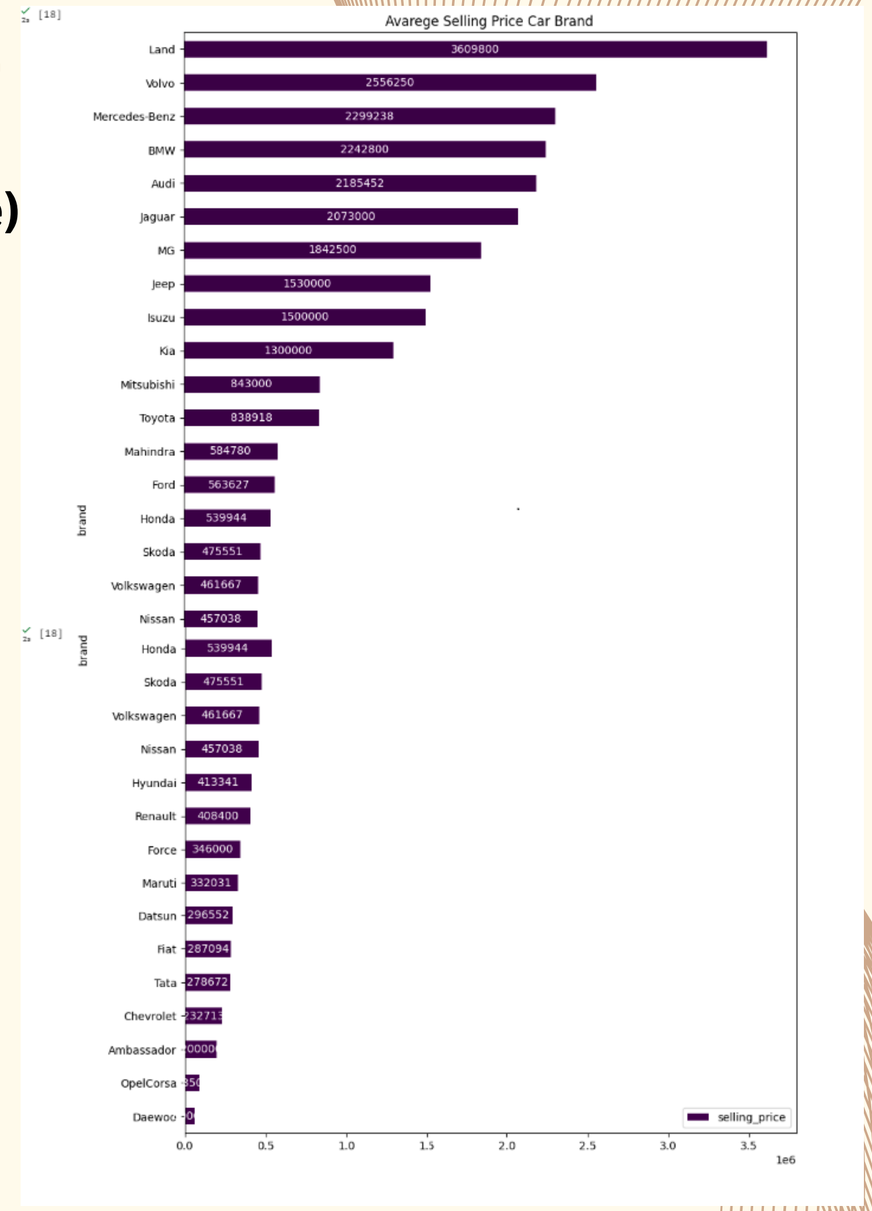
df.brand.value_counts()

```
[17] #To know countnvalue in coulmn
     df.brand.value_counts()
```

```
Maruti           1072
Hyundai           637
Mahindra          328
Tata              308
Ford              220
Honda             216
Toyota            170
Chevrolet         151
Renault           110
Volkswagen         93
Nissan             52
Skoda              49
Fiat               32
Audi               31
Datsun             29
BMW                25
Mercedes-Benz      21
Jaguar              5
Mitsubishi          5
Land                5
Volvo               4
Jeep                3
Ambassador          3
MG                  2
OpelCorsa           2
Daewoo              1
Force               1
Isuzu               1
Kia                 1
Name: brand, dtype: int64
```

use groupby and mean to extract values and plot to draw the graph

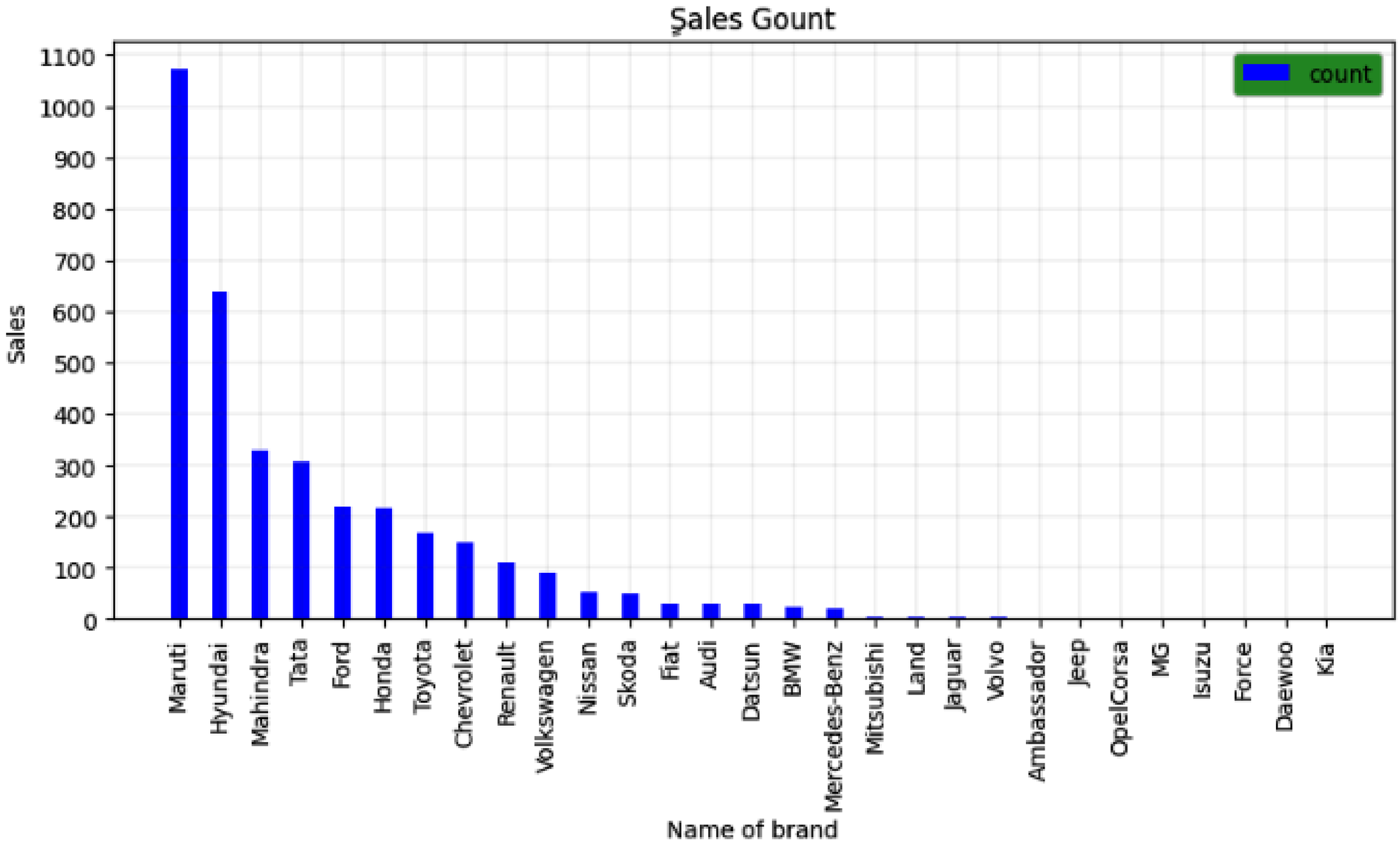
```
price = df.groupby(['brand'])[['selling_price']].mean()
price.sort_values(by='selling_price', ascending=True, inplace=True)
ax = price.plot(kind='barh', cmap='PRGn' , figsize=(10,16) ,
title= 'Avarege Selling Price Car Brand')
for c in ax.containers:

  # set the bar label
  ax.bar_label(c, fmt='%.0f',label_type='center', color='w',
  rotation=0)
```



Now trying to find out the sales count classified by brand

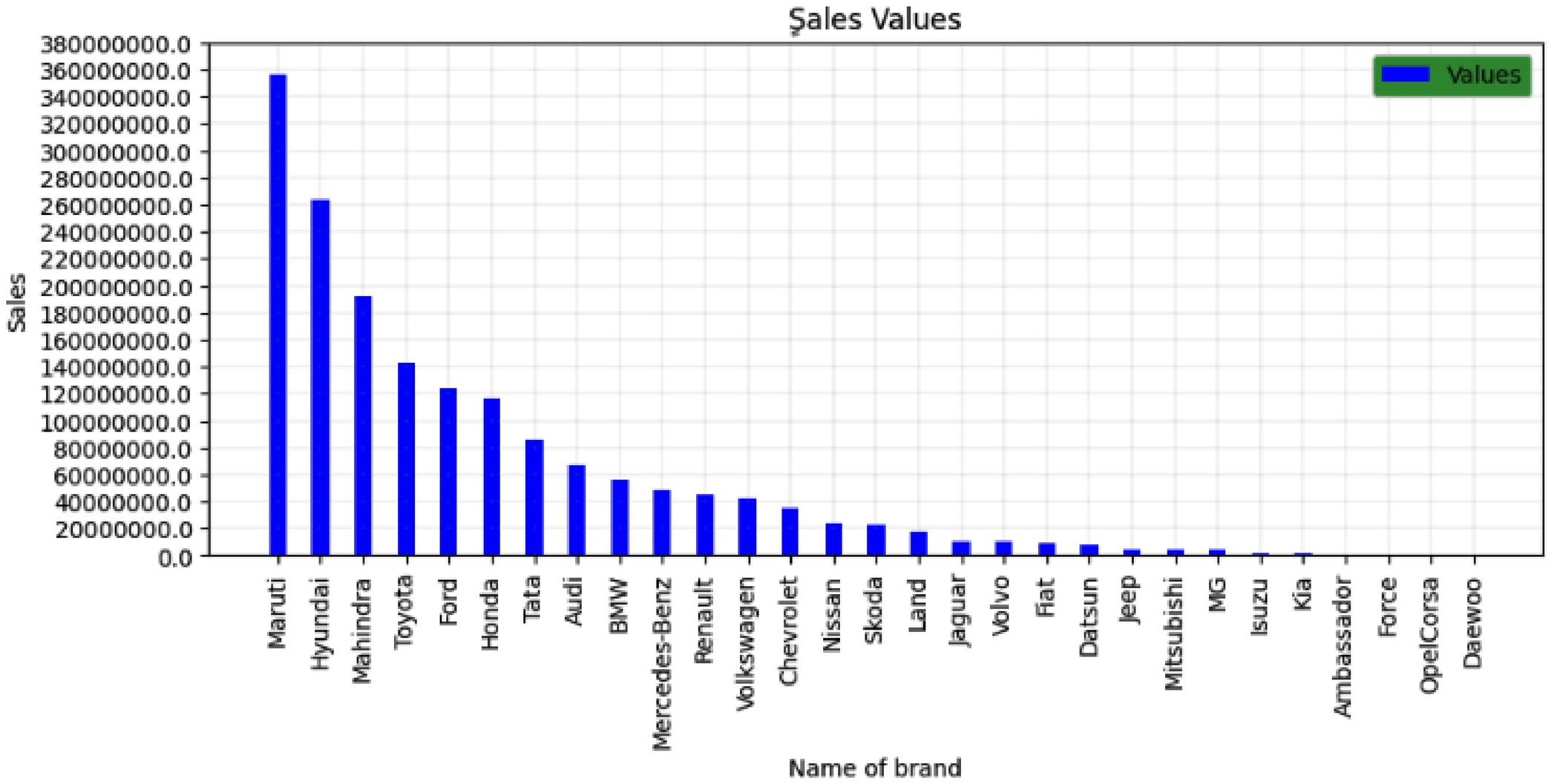
```
#plot 1:
data = df.groupby(['brand'])['brand'].count().sort_values(ascending=False) #to extract the count
x = data.index # to extract the brand name
y = data.values # to extract the count to brand
plt.subplot(2, 1, 1)#The location of the first graph 2 1 1
#the figure has 2 row, 1 columns, and this plot is the first plot.
plt.bar(x, y, color ='blue',width = 0.4)#Fomat to plt.bar
plt.rcParams['axes.facecolor'] = '#FFFFFF'#background color
plt.xticks(rotation=90) #Make the text of the label Make the text of the label at angle 90
plt.xlabel("Name of brand",fontsize=10,color="black")#Fomat and name to x
plt.ylabel("Sales",fontsize=10,color="black")#Fomat and name to y
plt.title("Sales Gount",color="black")#Fomat and name to title
plt.legend(["count"], loc ="upper right" ,facecolor='green', labelcolor='black')#Fomat and name to legend
plt.rcParams['figure.figsize'] = [10, 10]#Determine the size of the graph
plt.grid(color='grey', linestyle='-', linewidth=.1)#Fomat grid network that appears in graph background
#plt.xlim([0, 1]) #
#plt.ylim([0, 2000])#
#plt.locator_params(axis='x', nbins=20)#
plt.locator_params(axis='y', nbins=20)# to make y texts 100-200-300-- like that
```

[39]


Sales Gount
count
Sales
1100
1000
900
800
700
600
500
400
300
200
100
0
Maruti
Hyundai
Mahindra
Tata
Ford
Honda
Toyota
Chevrolet
Renault
Volkswagen
Nissan
Skoda
Fiat
Audi
Datsun
BMW
Mercedes-Benz
Mitsubishi
Land
Jaguar
Volvo
Ambassador
Jeep
OpelCorsa
MG
Isuzu
Force
Daewoo
Kia
Name of brand

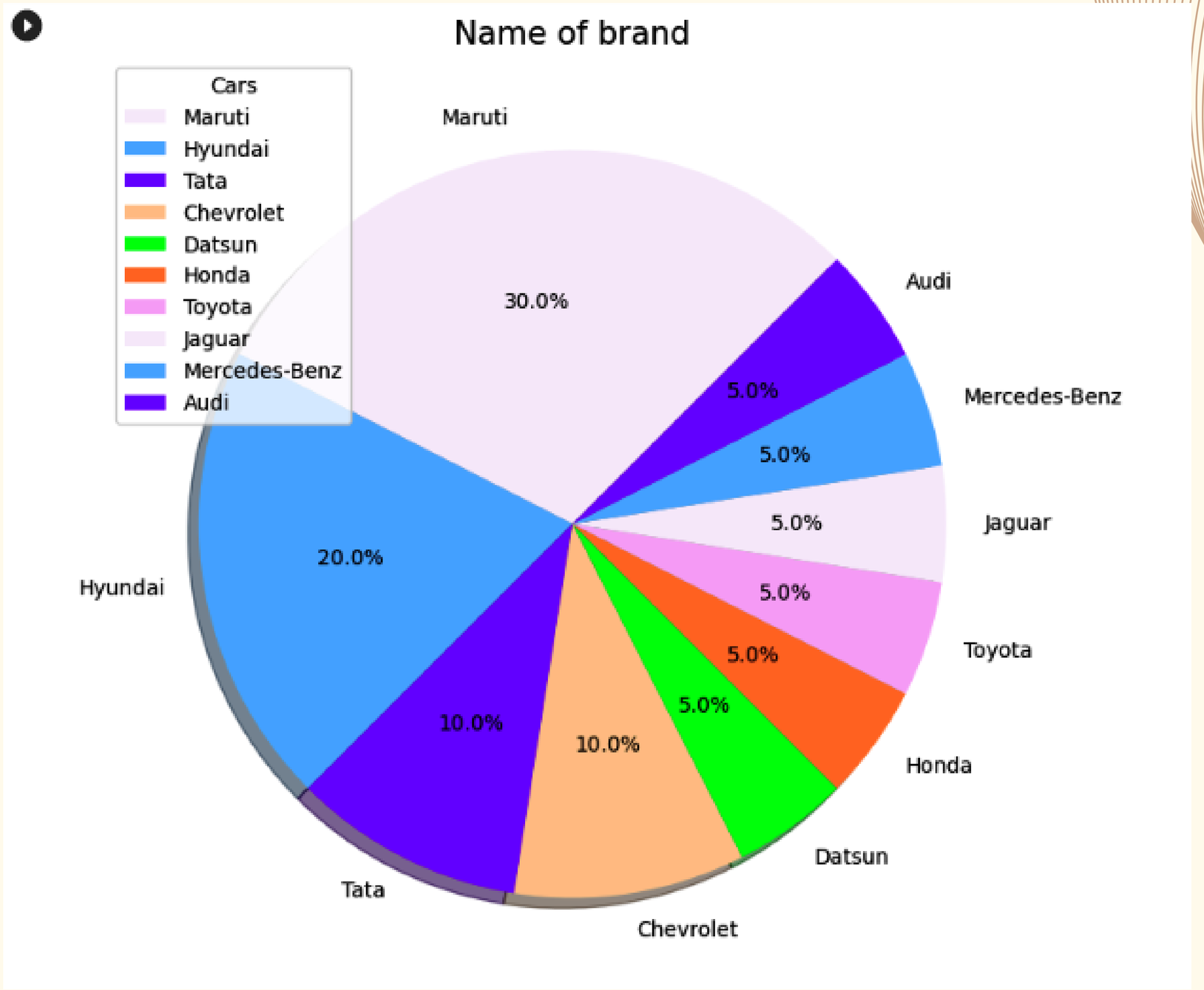
Now trying to find out the sales value classified by brand

```
#plot 2:
plt.subplot(2, 1, 2)#The location of the second graph 2 1 2
#the figure has 2 row, 1 columns, and this plot is the second plot.
data = df.groupby(['brand'])['selling_price'].sum().sort_values(ascending=False)
x = data.index # to extract the brand name
y = data.values #to extract the sum to brand
plt.bar(x, y, color ='blue',width = 0.4)
plt.rcParams['axes.facecolor'] = '#FFFFFF'
plt.xticks(rotation=90)
plt.xlabel("Name of brand",fontsize=10,color="black")
plt.ylabel("Sales",fontsize=10,color="black")
plt.title("Sales Values",color="black")
plt.legend(["Values"], loc ="upper right" ,facecolor='green', labelcolor='black')
plt.rcParams['figure.figsize'] = [10, 10]
plt.grid(color='grey', linestyle='-', linewidth=.1)
#plt.xlim([0, 1])
#plt.ylim([0, 2000])
#plt.locator_params(axis='x', nbins=20)
plt.locator_params(axis='y', nbins=20)
#plt.margins(x=0, y=0)
plt.yticks(ticks=plt.yticks()[0], labels=plt.yticks()[0])# Show real values,numbers big without it appear short
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9,top=0.9,wspace=0.5, hspace=0.6)# set the spacing between subplots
plt.suptitle("Sales")#Name for the whole graph
plt.show()#view
```


Şales Values
Values
Sales
380000000.0
360000000.0
340000000.0
320000000.0
300000000.0
280000000.0
260000000.0
240000000.0
220000000.0
200000000.0
180000000.0
160000000.0
140000000.0
120000000.0
100000000.0
80000000.0
60000000.0
40000000.0
20000000.0
0.0
Maruti
Hyundai
Mahindra
Toyota
Ford
Honda
Tata
Audi
BMW
Mercedes-Benz
Renault
Volkswagen
Chevrolet
Nissan
Skoda
Land
Jaguar
Volvo
Fiat
Datsun
Jeep
Mitsubishi
MG
Isuzu
Kia
Ambassador
Force
OpelCorsa
Daewoo
Name of brand

Display sales by count in pie graphs

```
labels = df["brand"][:20].value_counts().index #We chose only twenty
sizes = df["brand"][:20].value_counts() # We chose only twenty
data = df.groupby(['brand'])['brand'].count().sort_values(ascending=False)#to extract the count
x = data.index #to extract the brand name
y = data.values#to extract the count to brand
colors = ['#F8EEFB','#66b3ff','#8000FF','#ffcc99',"#00FF1B","#FF8040","#F8AEF8"]#color choice
plt.figure(figsize = (8,8))#Determine the size of the graph
# Creating explode data
#explode = (0.1, 0.0, 0.2, 0.3, 0.0, 0.0)
plt.pie(sizes, labels=labels, rotatelabels=False, autopct='%1.1f%%',colors=colors,shadow=True,
startangle=45)#Fomat pie
plt.title('Name of brand',color = 'black',fontsize = 15)#Fomat title
#plt.legend()#
plt.legend(title = "Cars")#title legend
#plt.legend(wedges, cars, title ="Cars",loc ="center left",bbox_to_anchor =(1, 0, 0.5, 1))
#myexplode = [0.2, 0, 0, 0]
plt.show()#view
```


Name of brand
Cars
Maruti
Hyundai
Tata
Chevrolet
Datsun
Honda
Toyota
Jaguar
Mercedes-Benz
Audi
Maruti
30.0%
Audi
5.0%
Mercedes-Benz
5.0%
Jaguar
5.0%
Toyota
5.0%
Honda
5.0%
Datsun
5.0%
Chevrolet
10.0%
Tata
10.0%
Hyundai
20.0%

Analysis by year

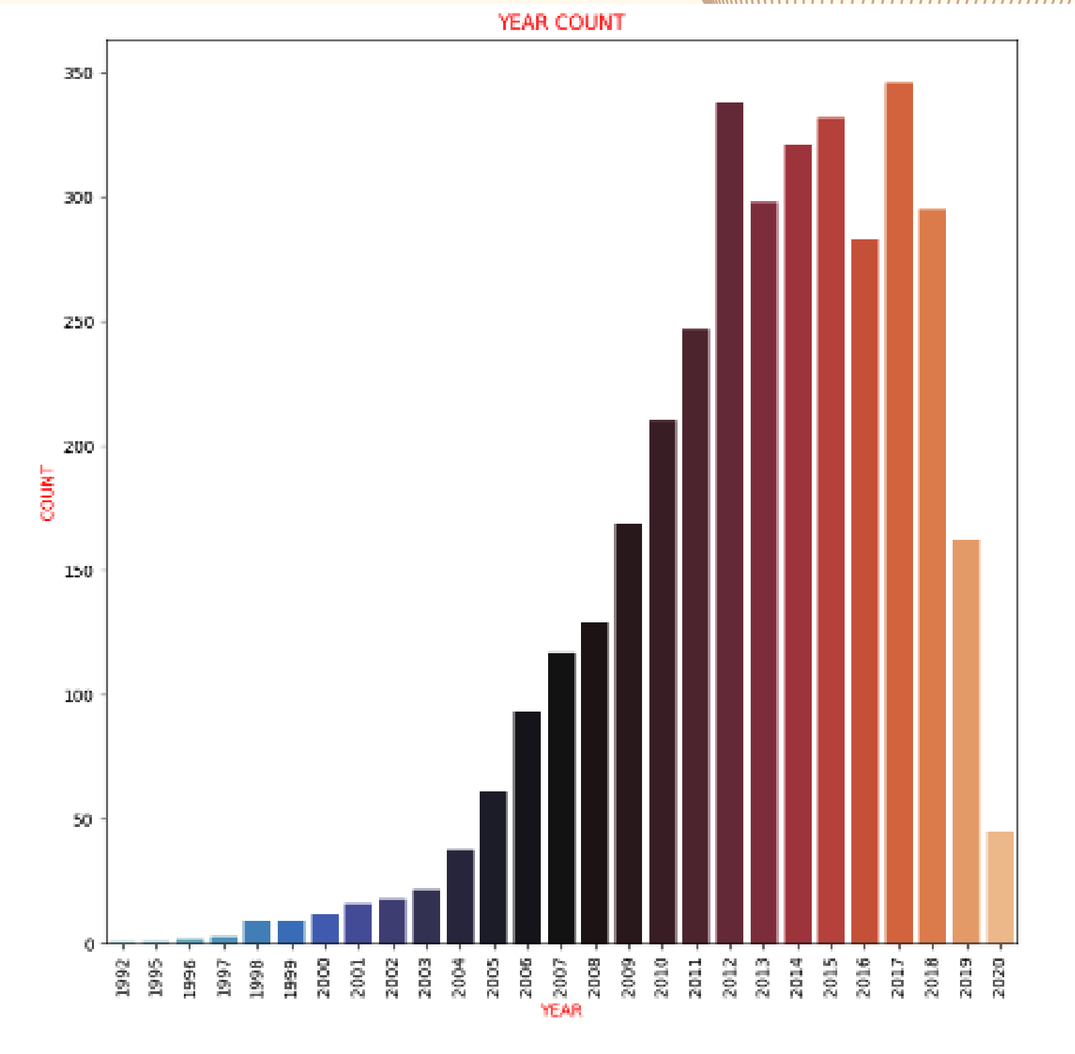
#To know countnvalue in year coulmn
df.year.value_counts()

```
[28] #To know countnvalue in year coulmn
     df.year.value_counts()
```

```
2017    346
2012    338
2015    332
2014    321
2013    298
2018    295
2016    283
2011    247
2010    210
2009    169
2019    162
2008    129
2007    117
2006     93
2005     61
2020     45
2004     38
2003     22
2002     18
2001     16
2000     12
1998      9
1999      9
1997      3
1996      2
1995      1
1992      1
Name: year, dtype: int64
```

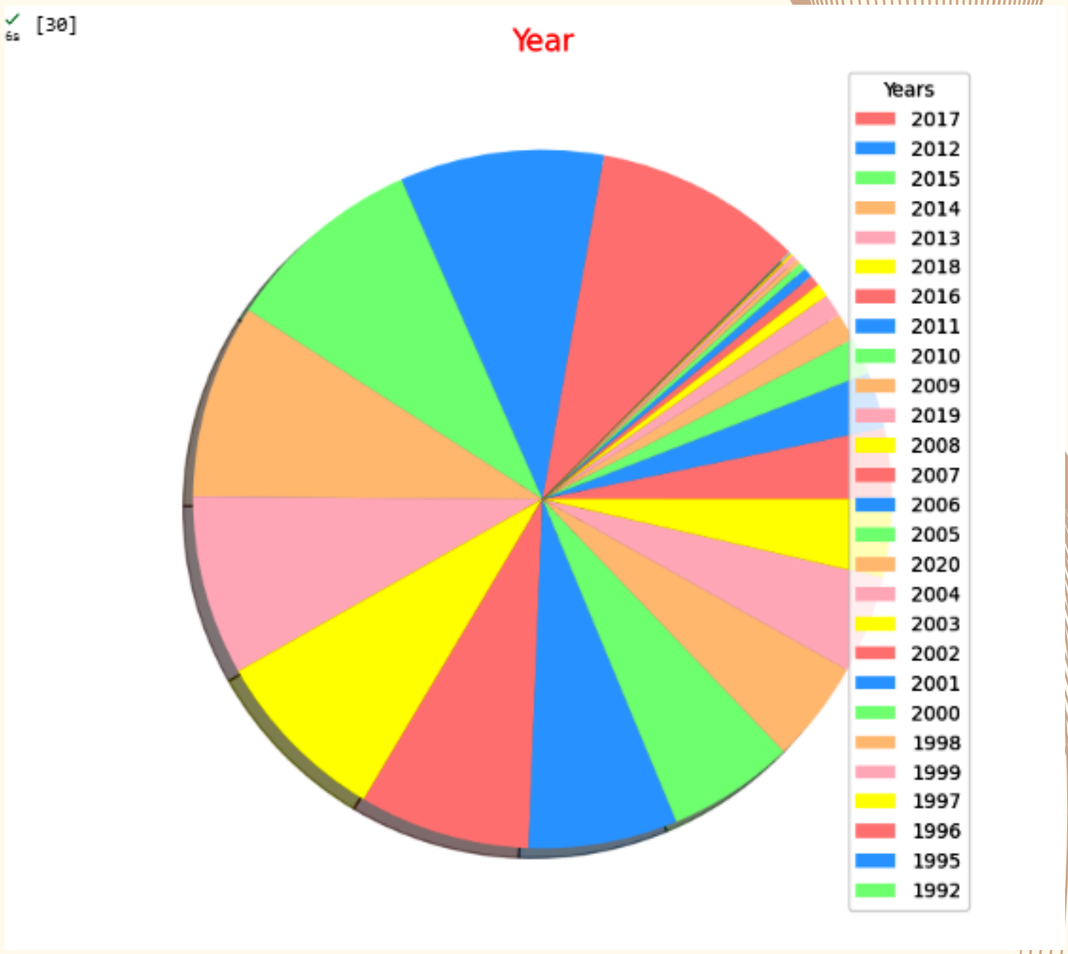
Analysis by year Display the graph by seaborn

```
sns.countplot(data=df,x="year",palette="icefire")
plt.xticks(rotation=90)
plt.xlabel("YEAR",fontsize=10,color="RED")
plt.ylabel("COUNT",fontsize=10,color="RED")
plt.title("YEAR COUNT",color="RED")
plt.show()
```



Variance between sales over the years by pie graph

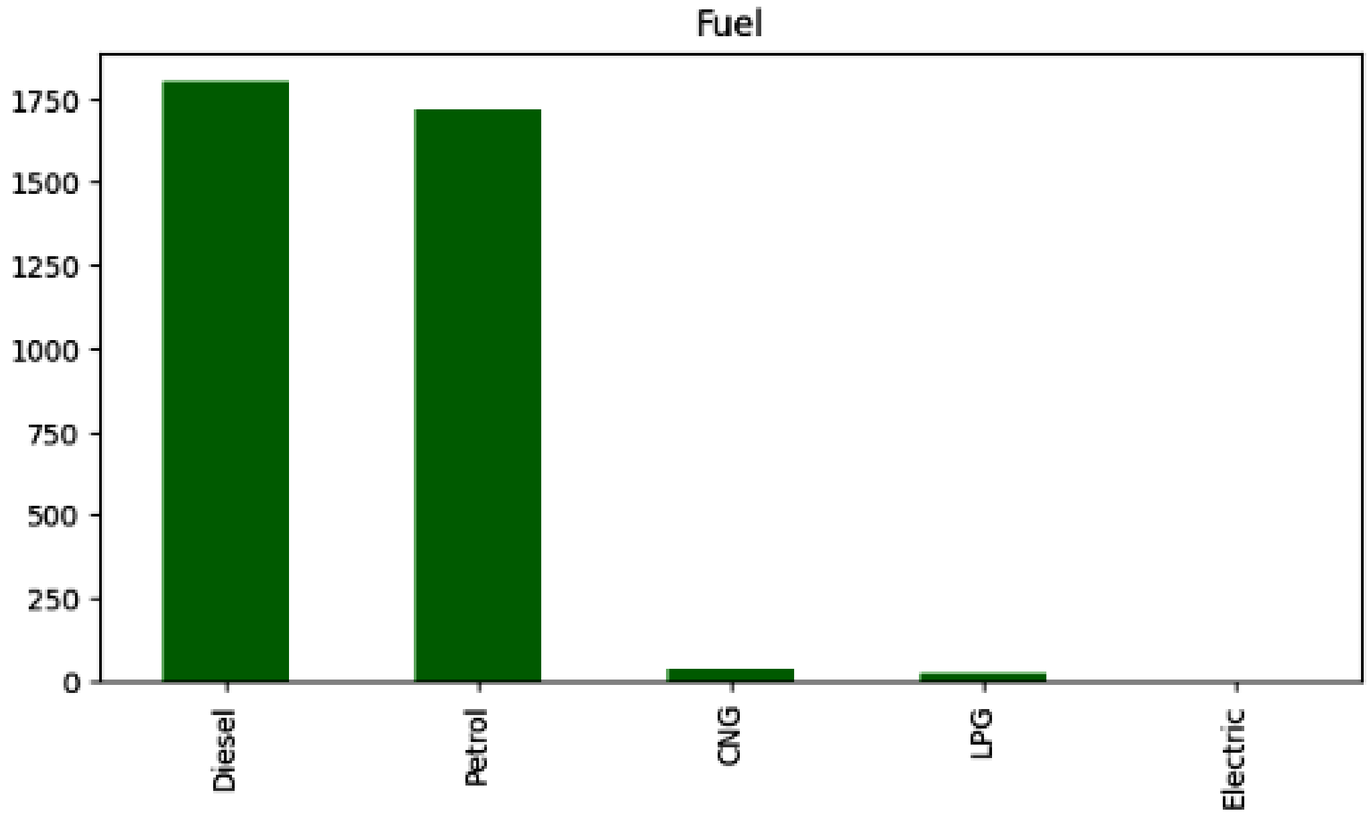
```
labels = df["year"].value_counts().index
sizes = df["year"].value_counts()
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99',"pink","yellow"]
plt.figure(figsize = (8,8))
plt.pie(sizes, labels=labels , rotatelabels=False, autopct=None,colors=colors,shadow=True, startangle=45,
labeldistance=None)
plt.title('Year',color = 'red',fontsize = 15)
plt.legend(title = "Years", loc='upper right')#title legend
plt.show()
```



Bar graph to view sales classified by fuel

df["fuel"].value_counts(sort =True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Fuel');

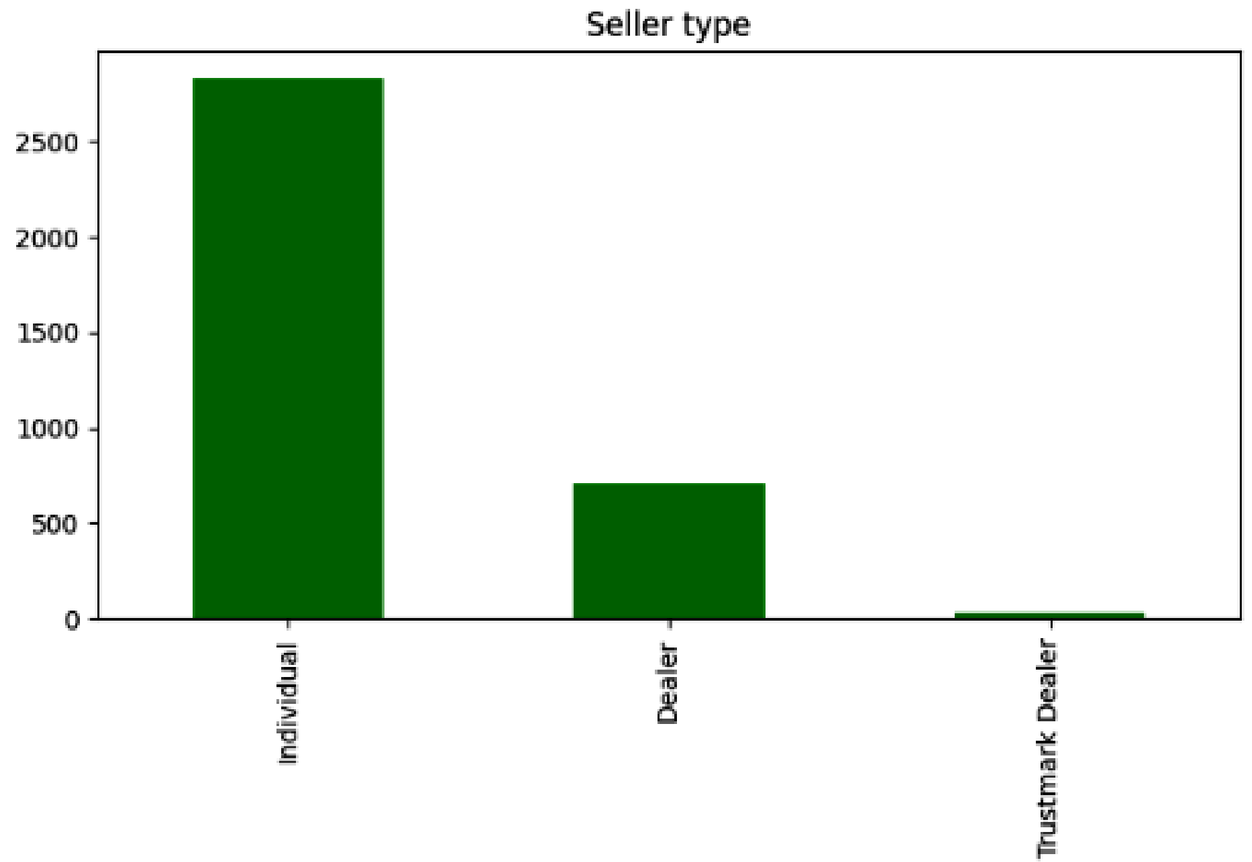
```
[31] #We will display bar graphs
     df["fuel"].value_counts(sort =True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Fuel');
```



Bar_graph to view sales classified by seller type

df["seller_type"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Seller type');

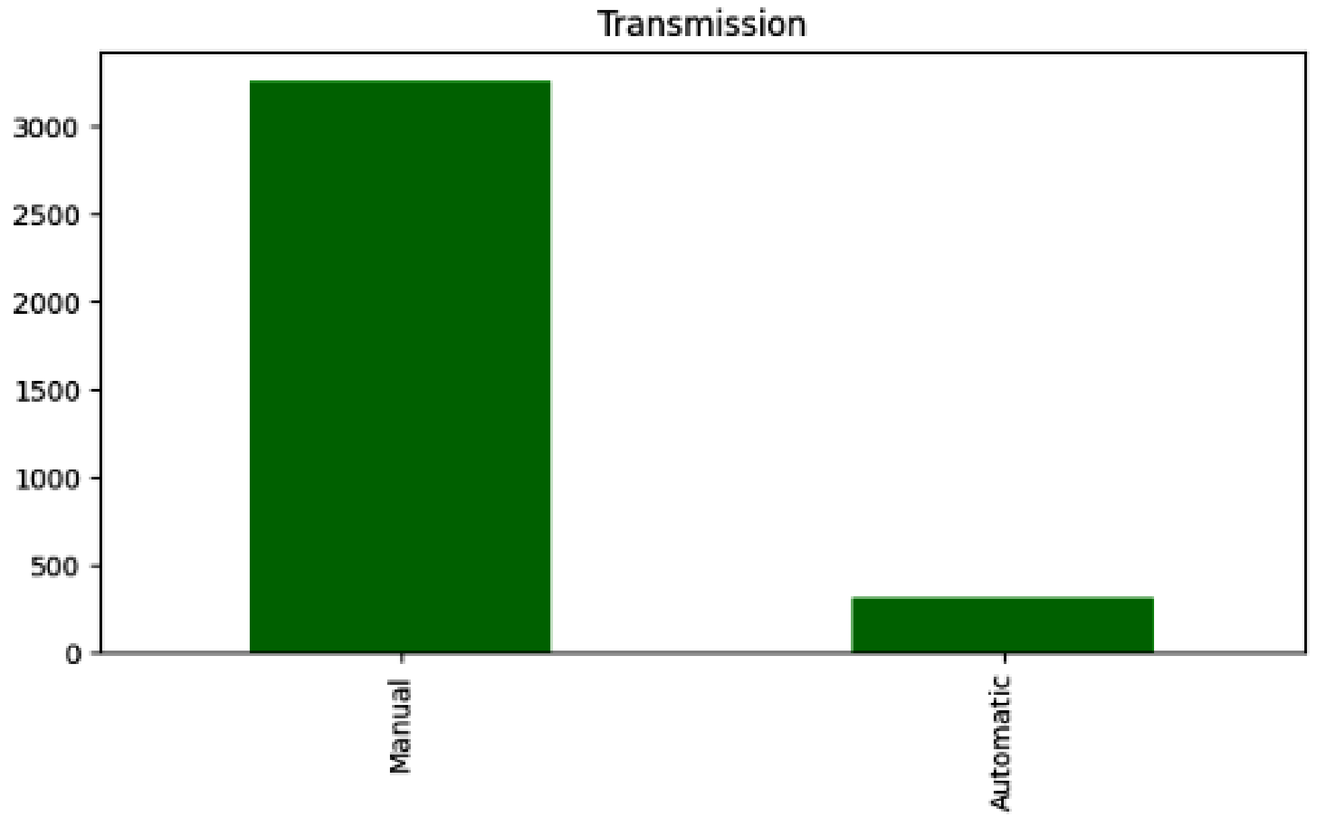
```
[32] #We will display bar graphs
     df["seller_type"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Seller type');
```



Bar graph to view sales classified by transmission

df["transmission"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Transmission');

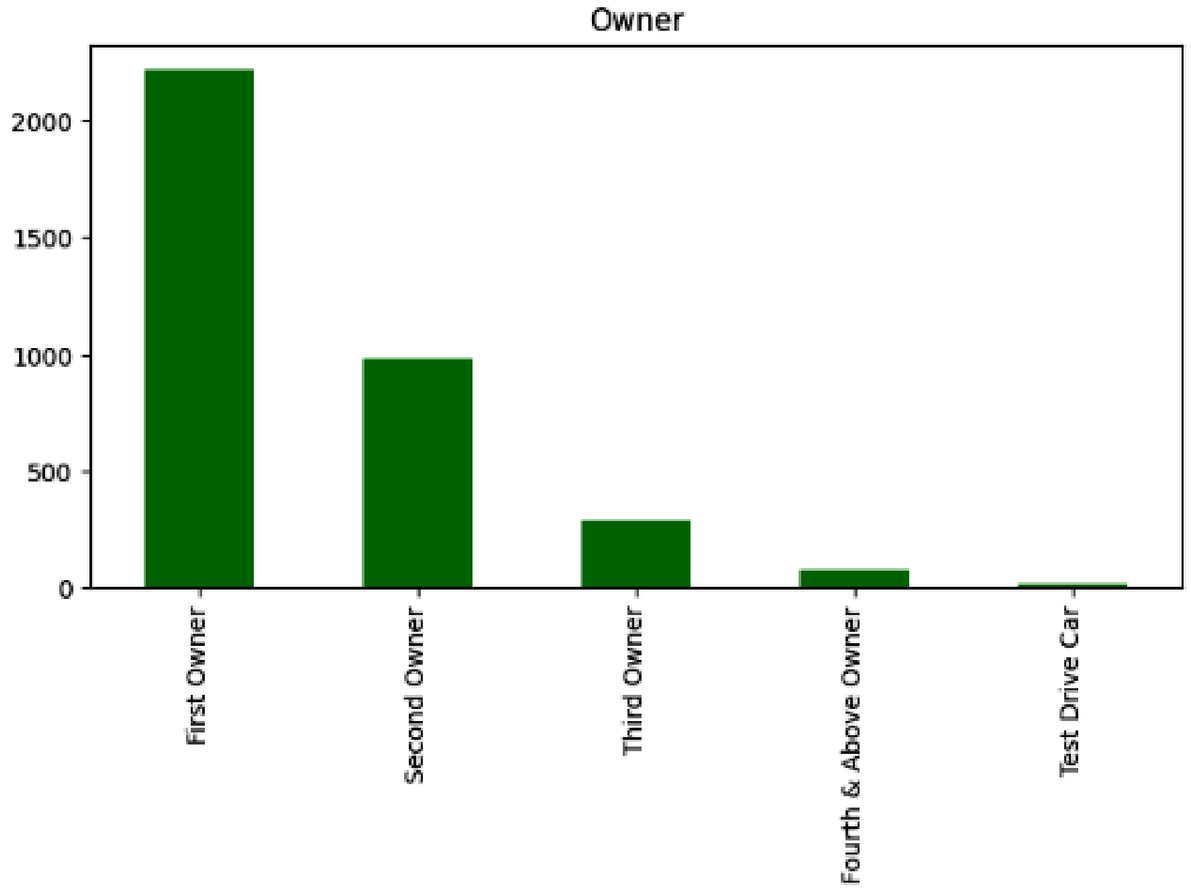
```
[33] #We will display bar graphs
     df["transmission"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Transmission');
```



Bar graph to view sales classified by owner

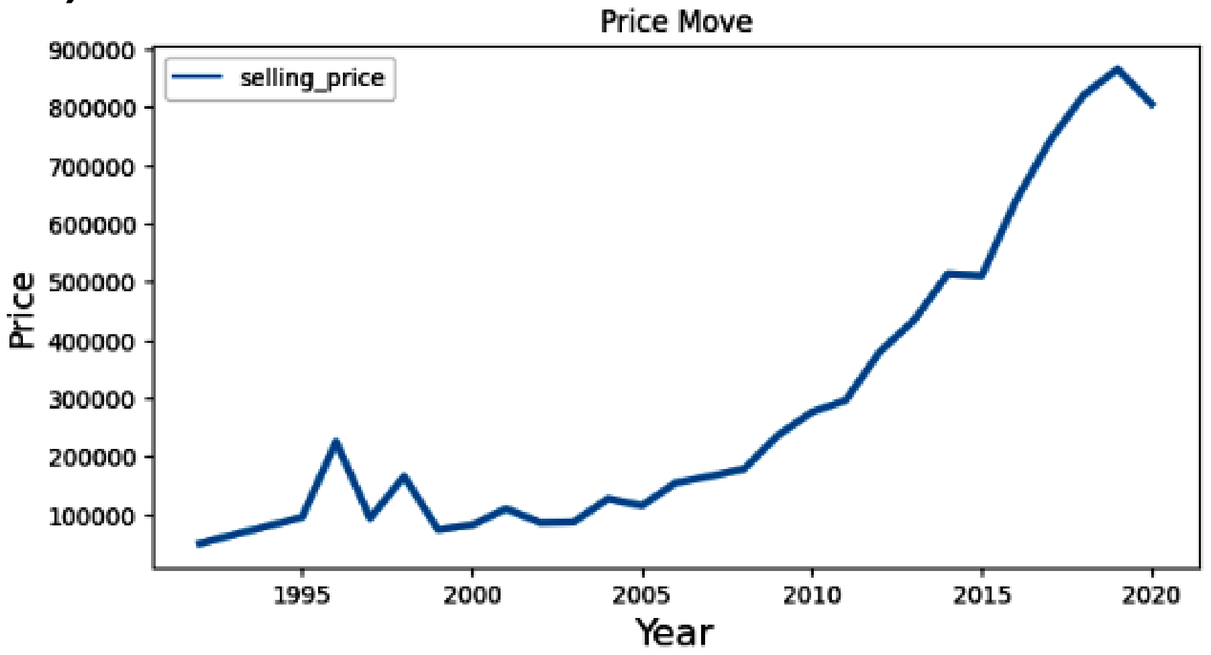
df["owner"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Owner');

```
[34] #We will display bar graphs
     df["owner"].value_counts(sort = True).plot(kind="bar", color=["green"], figsize=(8, 4) , title='Owner');
```



Graph linking the year to the selling price

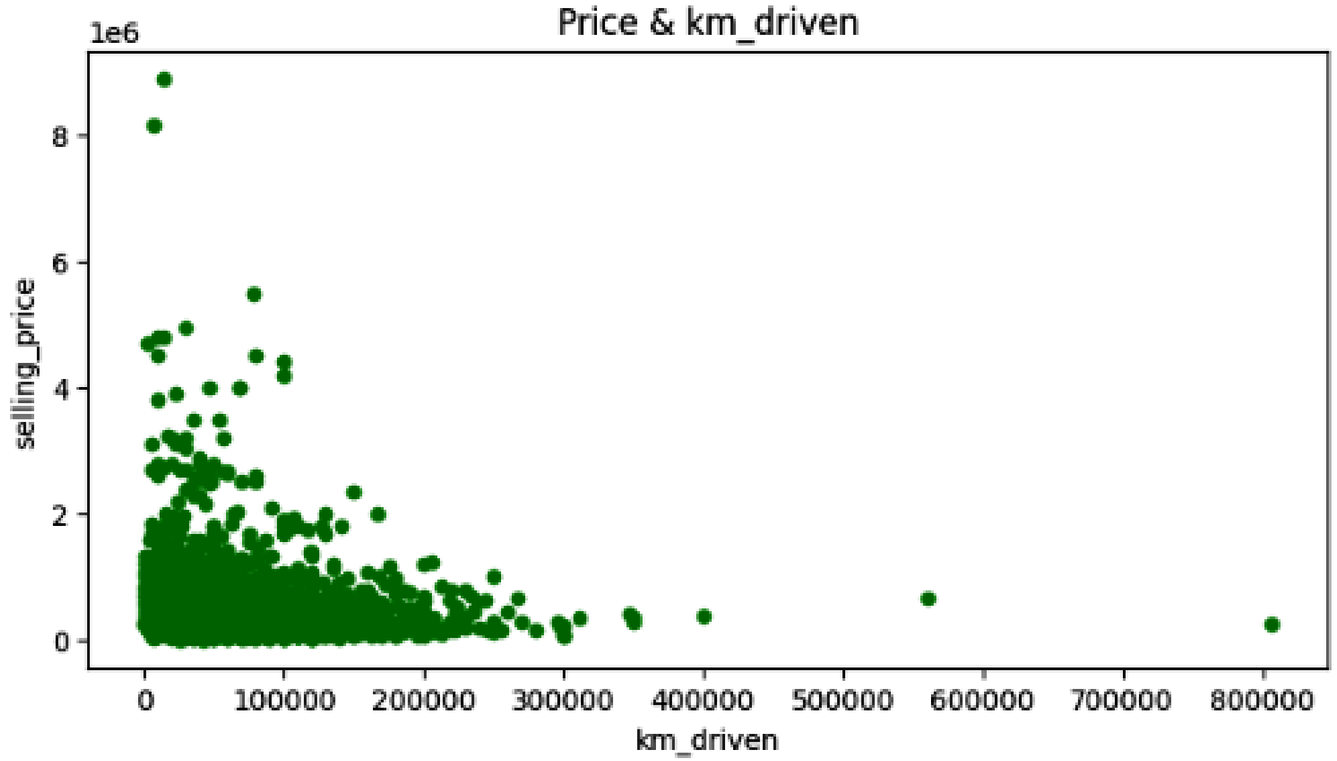
```
def line_plot(data, title ,xlabel, ylabel):
  plt.figure(figsize=(8, 4))
  sns.lineplot(data=data , palette="tab10", linewidth=3.0)
  plt.title(title, fontsize=12)
  plt.ylabel(ylabel, size=14)
  plt.xlabel(xlabel, size=16)
df_price_move = df.groupby(['year'])[['selling_price']].mean()
line_plot(df_price_move,'Price Move', 'Year', "Price")
```



Scatter plot between km_driven and price

df.plot(x="km_driven", y="selling_price", kind="scatter", figsize=(8, 4), title="Price & km_driven", color="green");

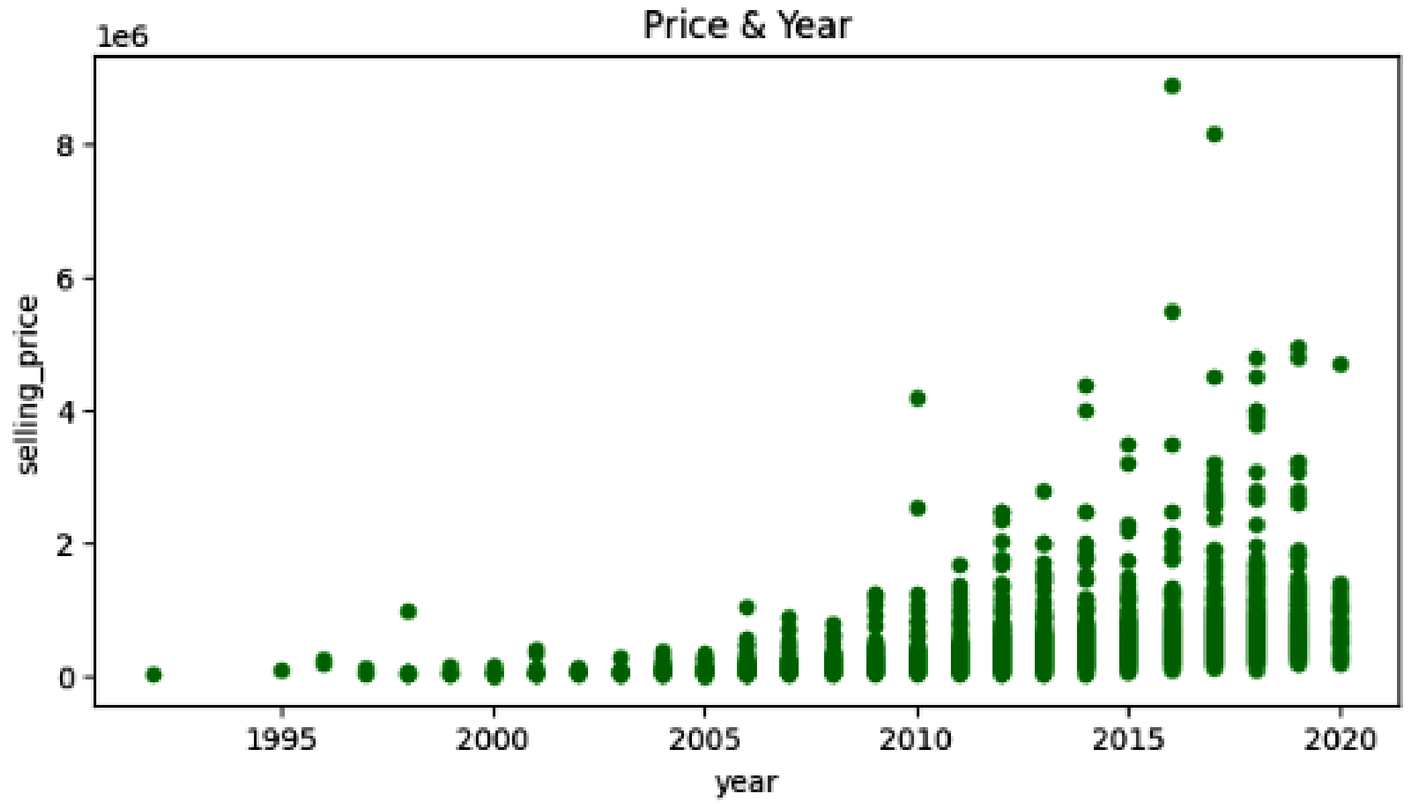
```
[36] #We will display bar graphs scatter
     df.plot(x="km_driven", y="selling_price", kind="scatter", figsize=(8, 4), title="Price & km_driven", color="green");
```



Scatter plot between km_driven and year

df.plot(x="year", y="selling_price", kind="scatter", figsize=(8, 4), title="Price & Year", color="green");

```
[37] #We will display bar graphs scatter
     df.plot(x="year", y="selling_price", kind="scatter", figsize=(8, 4), title="Price & Year", color="green");
```



Results

During our data analysis journey, we discovered that I can make things concise.

1. Maruti suzuki is best selling brand than Hyundai
2. Manual cars have higher sales compared to automatics.
3. The newer the car, the higher its sales performance.
4. Individual sellers have higher sales numbers.
5. Petrol and diesel cars are the top-selling fuel types.
6. The majority of sales occurred in the year 2017.

There are factors that affect on car price.

1. The newer the year of the car, the higher its price tends to be.
2. As the KM driving increases, the price of the car decreases.
3. The brand and model of the car play a significant role in determining its value.

Conclusion

While all the models showed promising performance, the Random Forest Regressor stood out with the highest score. By utilizing the Random Forest Regressor, we can significantly enhance the accuracy of Selling_Price prediction.

Rajesh Kumar


Thank
you