Import of Packages

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import statsmodels.api as sm
```

Import and clean data

In [2]:

```
df = pd.read_csv('ElectricCarData_Clean.csv')
df.head()
```

Out[2]:

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	Rapid
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	
2	Polestar	2	4.7	210	400	181	620	
3	BMW	iX3	6.8	180	360	206	560	
4	Honda	e	9.5	145	170	168	190	

In [3]:

```
df.columns.to_list()
```

Out[3]:

```
['Brand',
 'Model',
 'AccelSec',
 'TopSpeed_KmH',
 'Range_Km',
 'Efficiency_WhKm',
 'FastCharge_KmH',
 'RapidCharge',
 'PowerTrain',
 'PlugType',
 'BodyStyle',
 'Segment',
 'Seats',
 'PriceEuro']
```

In [4]:

```
#Print a Specific Row of a Pandas Dataframe
df.loc[[77]]
```

Out[4]:

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCha
77	Smart	EQ forfour	12.7	130	95	176	-	

After examining the data dictionary, it seems all variables are relevant.

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
Brand              0
Model              0
AccelSec           0
TopSpeed_KmH       0
Range_Km           0
Efficiency_WhKm    0
FastCharge_KmH     0
RapidCharge        0
PowerTrain         0
PlugType           0
BodyStyle          0
Segment            0
Seats              0
PriceEuro          0
dtype: int64
```

There exists no null value

Descriptive Statistics of the dataset

In [6]:

```
df.describe()
```

Out[6]:

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro
count	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000
mean	7.396117	179.194175	338.786408	189.165049	4.883495	55811.563107
std	3.017430	43.573030	126.014444	29.566839	0.795834	34134.665280
min	2.100000	123.000000	95.000000	104.000000	2.000000	20129.000000
25%	5.100000	150.000000	250.000000	168.000000	5.000000	34429.500000
50%	7.300000	160.000000	340.000000	180.000000	5.000000	45000.000000
75%	9.000000	200.000000	400.000000	203.000000	5.000000	65000.000000
max	22.400000	410.000000	970.000000	273.000000	7.000000	215000.000000

Information of the type of data in seach column

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Brand            103 non-null    object 
 1   Model            103 non-null    object 
 2   AccelSec         103 non-null    float64
 3   TopSpeed_KmH     103 non-null    int64  
 4   Range_Km         103 non-null    int64  
 5   Efficiency_WhKm  103 non-null    int64  
 6   FastCharge_KmH   103 non-null    object 
 7   RapidCharge      103 non-null    object 
 8   PowerTrain       103 non-null    object 
 9   PlugType         103 non-null    object 
 10  BodyStyle        103 non-null    object 
 11  Segment          103 non-null    object 
 12  Seats            103 non-null    int64  
 13  PriceEuro        103 non-null    int64  
dtypes: float64(1), int64(5), object(8)
memory usage: 11.4+ KB
```

Number of vehicles produced by each brand

In [8]:

```
companies = df.groupby('Brand').count()
print(companies['Model'].sort_values(ascending = False))
```

```
Brand
Tesla         13
Audi           9
Nissan         8
Volkswagen     8
Skoda          6
Kia            5
Porsche        5
Renault        5
BMW            4
Ford           4
Smart          3
Mercedes       3
Opel           3
Hyundai        3
Byton          3
Peugeot        2
Honda          2
Fiat           2
SEAT           1
Sono           1
Polestar       1
Aiways         1
MG             1
Mini           1
Mazda          1
Lucid          1
Lightyear      1
Lexus          1
Jaguar         1
DS             1
Citroen        1
```

```
CUPRA            1
Volvo            1
Name: Model, dtype: int64
```

In [9]:

```
df.corr()
```

Out[9]:

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro
AccelSec	1.000000	-0.786195	-0.677062	-0.382904	-0.175335	-0.627174
TopSpeed_KmH	-0.786195	1.000000	0.746662	0.355675	0.126470	0.829057
Range_Km	-0.677062	0.746662	1.000000	0.313077	0.300163	0.674844
Efficiency_WhKm	-0.382904	0.355675	0.313077	1.000000	0.301230	0.396705
Seats	-0.175335	0.126470	0.300163	0.301230	1.000000	0.020920
PriceEuro	-0.627174	0.829057	0.674844	0.396705	0.020920	1.000000

The closer to 1, the stronger the correlation between these variables.

A minus sign means that these 2 variables are negatively correlated, i.e. one decreases with increasing the other and vice versa.

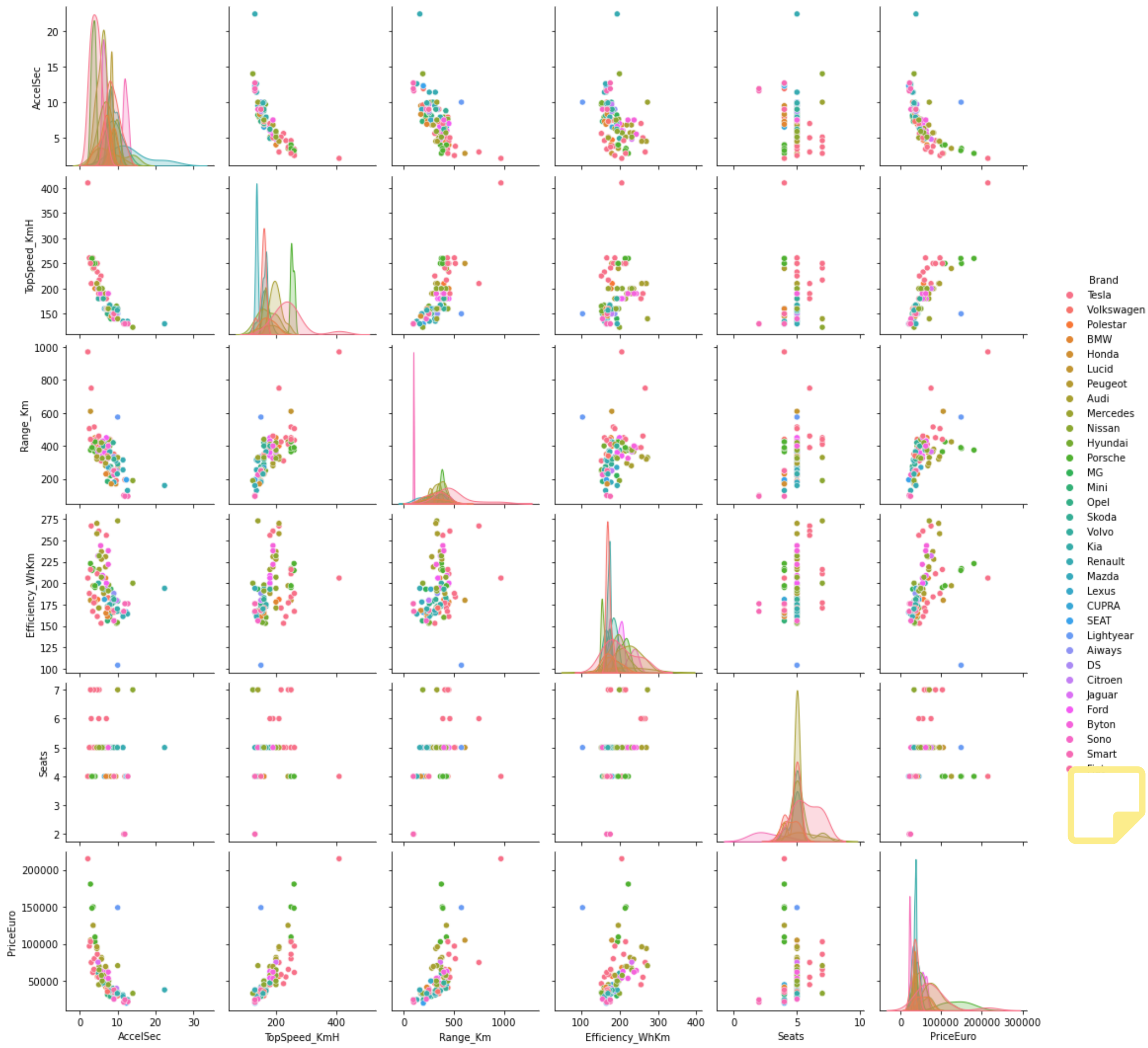
Pairplot of all the columns based on Brand presence

In [10]:

```
sb.pairplot(df,hue='Brand')
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x2212bba9ee0>
```



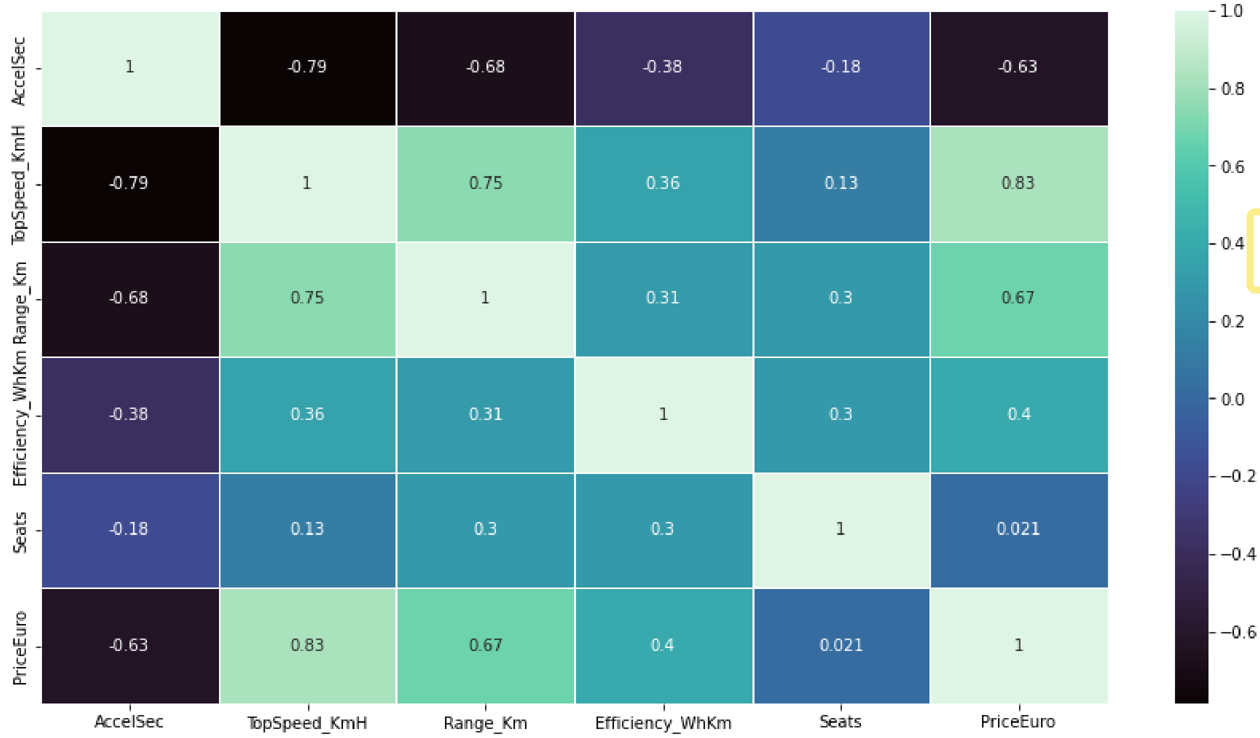
Heatmap to show the correlation of the data

In [11]:

```python
# Generating correlation matrix using Seaborn library
ax= plt.figure(figsize=(15,8))
sb.heatmap(df.corr(),linewidths=1,linecolor='white',annot=True, cmap='mako')
```

Out[11]:

```
<AxesSubplot:>
```



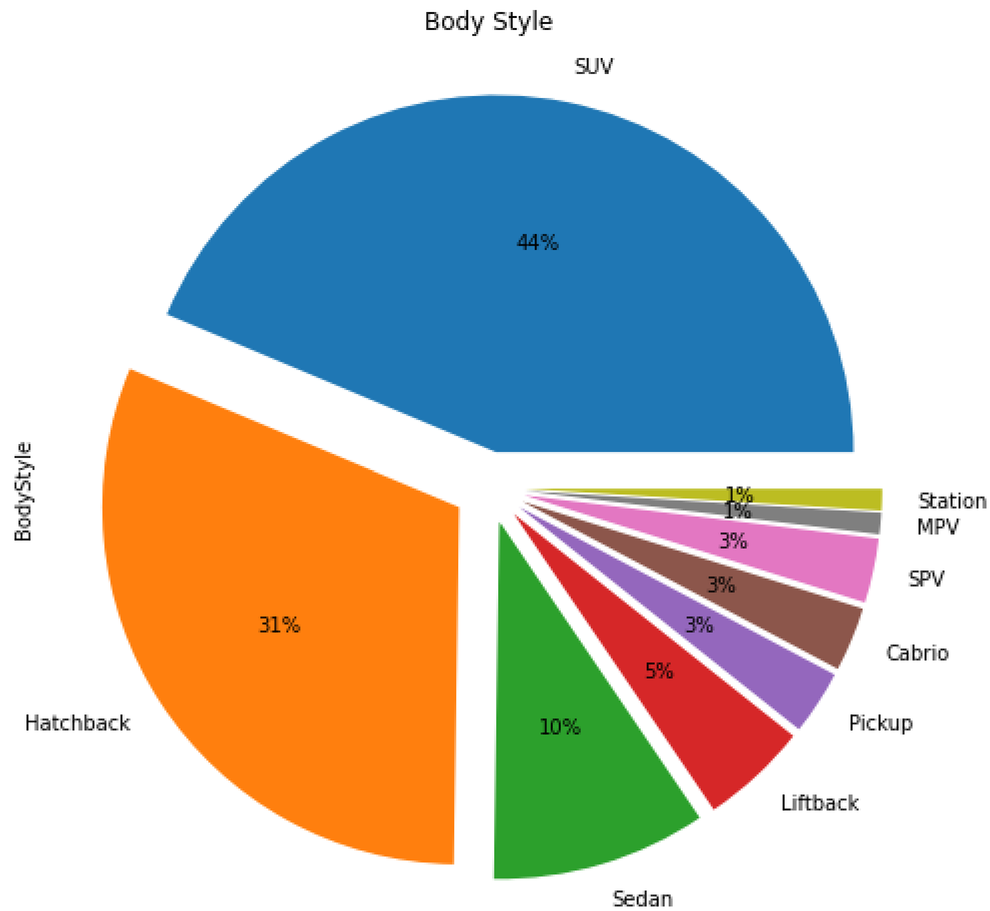
Cars and their body style

In [12]:

```
# display the body style
df['BodyStyle'].value_counts().plot.pie(figsize=(8,15),autopct='%.0f%%',explode=(0.1,0.
plt.title('Body Style')
```

Out[12]:

```
Text(0.5, 1.0, 'Body Style')
```

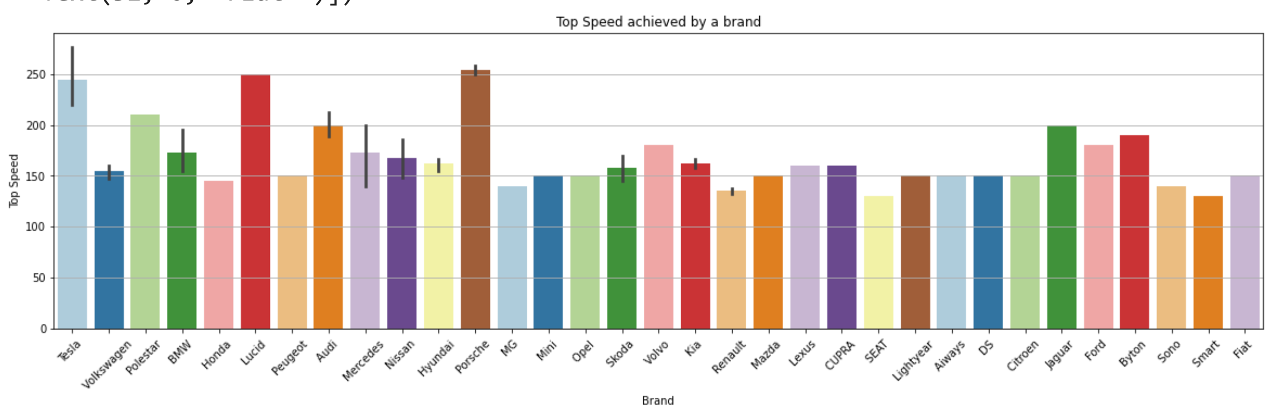


Top speeds achieved by the cars of a brand

In [13]:
```
ax= plt.figure(figsize=(20,5))
sb.barplot(x='Brand',y='TopSpeed_KmH',data=df,palette='Paired')
plt.grid(axis='y')
plt.title('Top Speed achieved by a brand')
plt.xlabel('Brand')
plt.ylabel('Top Speed')
plt.xticks(rotation=45)
```

Out[13]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 [Text(0, 0, 'Tesla '),
  Text(1, 0, 'Volkswagen '),
  Text(2, 0, 'Polestar '),
  Text(3, 0, 'BMW '),
  Text(4, 0, 'Honda '),
  Text(5, 0, 'Lucid '),
  Text(6, 0, 'Peugeot '),
  Text(7, 0, 'Audi '),
  Text(8, 0, 'Mercedes '),
  Text(9, 0, 'Nissan '),
  Text(10, 0, 'Hyundai '),
  Text(11, 0, 'Porsche '),
  Text(12, 0, 'MG '),
  Text(13, 0, 'Mini '),
  Text(14, 0, 'Opel '),
  Text(15, 0, 'Skoda '),
```

```
 Text(16, 0, 'Volvo '),
 Text(17, 0, 'Kia '),
 Text(18, 0, 'Renault '),
 Text(19, 0, 'Mazda '),
 Text(20, 0, 'Lexus '),
 Text(21, 0, 'CUPRA '),
 Text(22, 0, 'SEAT '),
 Text(23, 0, 'Lightyear '),
 Text(24, 0, 'Aiways '),
 Text(25, 0, 'DS '),
 Text(26, 0, 'Citroen '),
 Text(27, 0, 'Jaguar '),
 Text(28, 0, 'Ford '),
 Text(29, 0, 'Byton '),
 Text(30, 0, 'Sono '),
 Text(31, 0, 'Smart '),
 Text(32, 0, 'Fiat ')])
```



Porsche, Lucid and Tesla produce the fastest cars and Smart the lowest

Car Efficiency

Byton , Jaguar and Audi are the most efficient and Lightyear the least

Price of cars (in Euro)

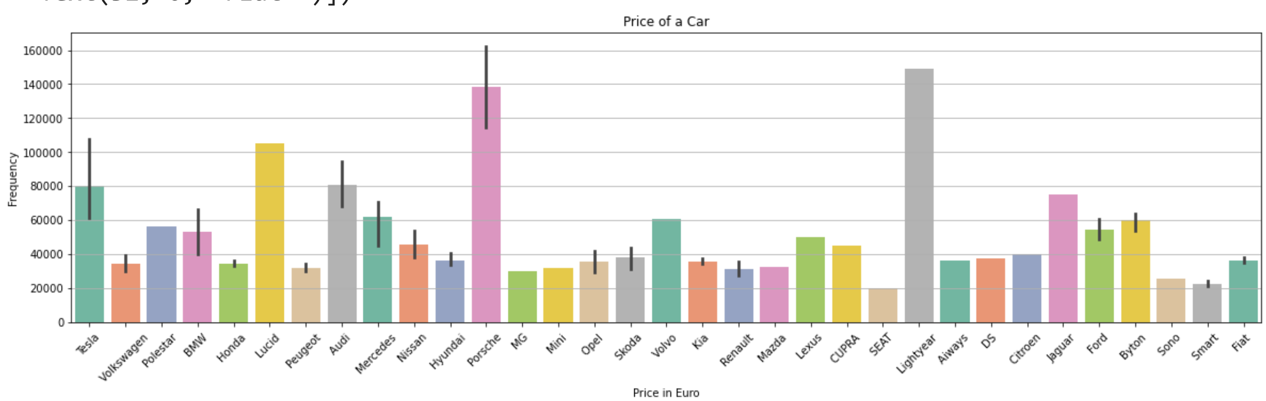
In [14]:

```
ax= plt.figure(figsize=(20,5))
sb.barplot(x='Brand',y='PriceEuro',data=df,palette='Set2')
plt.title('Price of a Car')
plt.xlabel('Price in Euro')
plt.grid(axis='y')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
```

Out[14]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 [Text(0, 0, 'Tesla '),
  Text(1, 0, 'Volkswagen '),
  Text(2, 0, 'Polestar '),
  Text(3, 0, 'BMW '),
  Text(4, 0, 'Honda '),
  Text(5, 0, 'Lucid '),
  Text(6, 0, 'Peugeot '),
  Text(7, 0, 'Audi '),
  Text(8, 0, 'Mercedes '),
```

```
 Text(9, 0, 'Nissan '),
 Text(10, 0, 'Hyundai '),
 Text(11, 0, 'Porsche '),
 Text(12, 0, 'MG '),
 Text(13, 0, 'Mini '),
 Text(14, 0, 'Opel '),
 Text(15, 0, 'Skoda '),
 Text(16, 0, 'Volvo '),
 Text(17, 0, 'Kia '),
 Text(18, 0, 'Renault '),
 Text(19, 0, 'Mazda '),
 Text(20, 0, 'Lexus '),
 Text(21, 0, 'CUPRA '),
 Text(22, 0, 'SEAT '),
 Text(23, 0, 'Lightyear '),
 Text(24, 0, 'Aiways '),
 Text(25, 0, 'DS '),
 Text(26, 0, 'Citroen '),
 Text(27, 0, 'Jaguar '),
 Text(28, 0, 'Ford '),
 Text(29, 0, 'Byton '),
 Text(30, 0, 'Sono '),
 Text(31, 0, 'Smart '),
 Text(32, 0, 'Fiat ')])
```



Lightyear, Porsche and Lucid are the most expensive and SEAT and Smart the least

Car efficiency

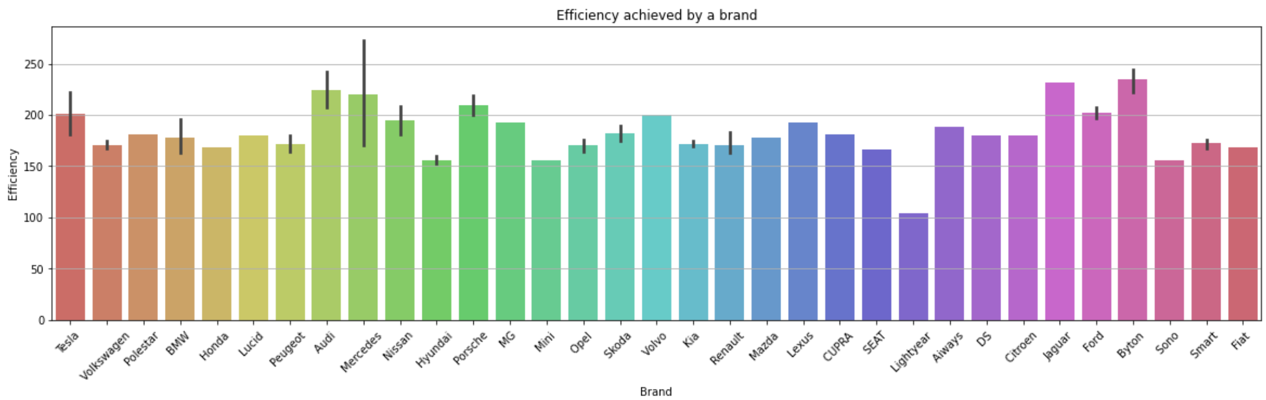
In [15]:

```
ax= plt.figure(figsize=(20,5))
sb.barplot(x='Brand',y='Efficiency_WhKm',data=df,palette='hls')
plt.grid(axis='y')
plt.title('Efficiency achieved by a brand')
plt.xlabel('Brand')
plt.ylabel('Efficiency')
plt.xticks(rotation=45)
```

Out[15]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 [Text(0, 0, 'Tesla '),
  Text(1, 0, 'Volkswagen '),
  Text(2, 0, 'Polestar '),
  Text(3, 0, 'BMW '),
  Text(4, 0, 'Honda '),
  Text(5, 0, 'Lucid '),
```

```
 Text(6, 0, 'Peugeot '),
 Text(7, 0, 'Audi '),
 Text(8, 0, 'Mercedes '),
 Text(9, 0, 'Nissan '),
 Text(10, 0, 'Hyundai '),
 Text(11, 0, 'Porsche '),
 Text(12, 0, 'MG '),
 Text(13, 0, 'Mini '),
 Text(14, 0, 'Opel '),
 Text(15, 0, 'Skoda '),
 Text(16, 0, 'Volvo '),
 Text(17, 0, 'Kia '),
 Text(18, 0, 'Renault '),
 Text(19, 0, 'Mazda '),
 Text(20, 0, 'Lexus '),
 Text(21, 0, 'CUPRA '),
 Text(22, 0, 'SEAT '),
 Text(23, 0, 'Lightyear '),
 Text(24, 0, 'Aiways '),
 Text(25, 0, 'DS '),
 Text(26, 0, 'Citroen '),
 Text(27, 0, 'Jaguar '),
 Text(28, 0, 'Ford '),
 Text(29, 0, 'Byton '),
 Text(30, 0, 'Sono '),
 Text(31, 0, 'Smart '),
 Text(32, 0, 'Fiat ')])
```



Byton , Jaguar and Audi are the most efficient and Lightyear the least

Build and evaluate the models

- linear Regression using OLS method

In [16]:
```
#Putting independent variables as x and dependent variable as y
x=df[['AccelSec','Range_Km','TopSpeed_KmH','Efficiency_WhKm']]
y=df['PriceEuro']
```

In [17]:
```
#Finding out the linear regression using OLS method
x= sm.add_constant(x)
results = sm.OLS(y,x)
```

In [18]:

```python
#Fitting the model and summarizing
model=results.fit()
model.summary()
```

Out[18]:

OLS Regression Results

Dep. Variable:	PriceEuro	**R-squared:**	0.711
Model:	OLS	**Adj. R-squared:**	0.699
Method:	Least Squares	**F-statistic:**	60.28
Date:	Sun, 30 Oct 2022	**Prob (F-statistic):**	1.37e-25
Time:	10:54:35	**Log-Likelihood:**	-1156.8
No. Observations:	103	**AIC:**	2324.
Df Residuals:	98	**BIC:**	2337.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.051e+05	2.3e+04	-4.578	0.000	-1.51e+05	-5.96e+04
AccelSec	1482.2127	1033.219	1.435	0.155	-568.178	3532.603
Range_Km	37.7714	22.680	1.665	0.099	-7.236	82.779
TopSpeed_KmH	613.9243	78.224	7.848	0.000	458.691	769.157
Efficiency_WhKm	143.7166	68.228	2.106	0.038	8.320	279.113

Omnibus:	94.859	**Durbin-Watson:**	2.071
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1049.593
Skew:	2.978	**Prob(JB):**	1.21e-228
Kurtosis:	17.460	**Cond. No.**	5.53e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.53e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Importing train test split from Scikit Learn

In [19]:

```python
from sklearn.model_selection import train_test_split
```

In [20]:

```python
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=36
```

Importing Linear regression

In [21]:

```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

In [22]:

```
lr.fit(X_train, y_train)
pred = lr.predict(X_test)
```

In [23]:

```
#Finding out the R-squared value
from sklearn.metrics import r2_score
r2=(r2_score(y_test,pred))
print(r2*100)
```

```
78.35225979903609
```

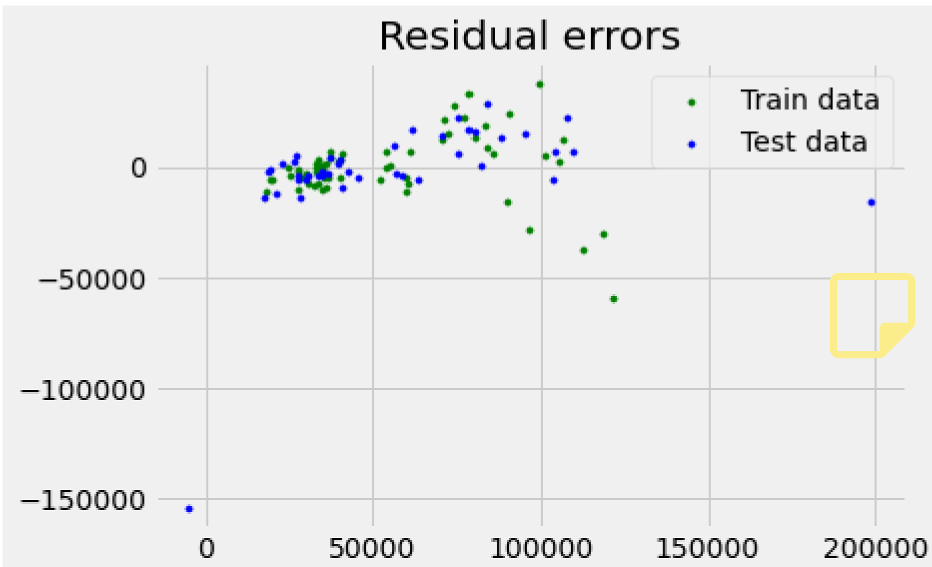
- Around 78% of the dependant variable has been explained by the independant variables

In [24]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics


# defining feature matrix(X) and response vector(y)
X=df[['AccelSec','Range_Km','TopSpeed_KmH','Efficiency_WhKm']]
y=df['PriceEuro']


# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)


# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))

# plot for residual error

# setting plot style
plt.style.use('fivethirtyeight')

# plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
            color = "green", s = 10, label = 'Train data')

# plotting residual errors in test data
```

```
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
            color = "blue", s = 10, label = 'Test data')

# plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

# plotting legend
plt.legend(loc = 'upper right')

# plot title
plt.title("Residual errors")

# method call for showing the plot
plt.show()
```

```
Coefficients:  [427.4297895  -38.60326281 711.40780336 371.28905011]
Variance score: 0.4885529030998087
```



- In the above plot, I determine the accuracy score using Explained Variance Score.
- Variance score is around .5
- The best possible score is 1.0, lower values are worse

Logistic Regression

In [25]:

```
# Import the necessary packages to perform Logistic regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import matplotlib.pyplot as plt
```

In [26]:

```
#Putting Yes value as 1 and No value as 0 for Logistic Regression

df['RapidCharge'].replace(to_replace=['No','Yes'],value=[0, 1],inplace=True)
```

In [27]:

```
y1=df[['RapidCharge']]
x1=df[['PriceEuro']]
```

In [28]:
```python
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2,random_st
```

In [29]:
```python
#Importing Logistic Regression
from sklearn.linear_model import LogisticRegression
log= LogisticRegression()
```

In [30]:
```python
log.fit(X1_train, y1_train)
pred1 = log.predict(X1_test)
pred1
```

```
C:\Users\Yousof\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[30]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int64)
```

Confusion Matrix of the regression

In [31]:
```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y1_test, pred1)
cm
```

Out[31]:
```
ray([[ 0,  1],
       [ 0, 20]],     int64)
```

In [33]:
```python
#Finding out the accuracy score
from sklearn.metrics import accuracy_score
score=accuracy_score(y1_test,pred1)
score*100
```

Out[33]:
```
95.23809523809523
```

- Data is accurate upto 95%