Importing the required libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import joblib
import seaborn as sns

#scoring and tuning
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

#models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

Data Collection and Preprocessing

```
df = pd.read_csv('car_data.csv')
```

```
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Tr
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	

```
df.dtypes
```

```
Car_Name          object
```

```
Year               int64
Selling_Price    float64
Present_Price    float64
Kms_Driven         int64
Fuel_Type         object
Seller_Type       object
Transmission      object
Owner              int64
dtype: object
```

```
# checking the number of rows and columns
df.shape
```

```
(301, 9)
```

```
# getting some information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Seller_Type    301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

```
df.isna().any()
```

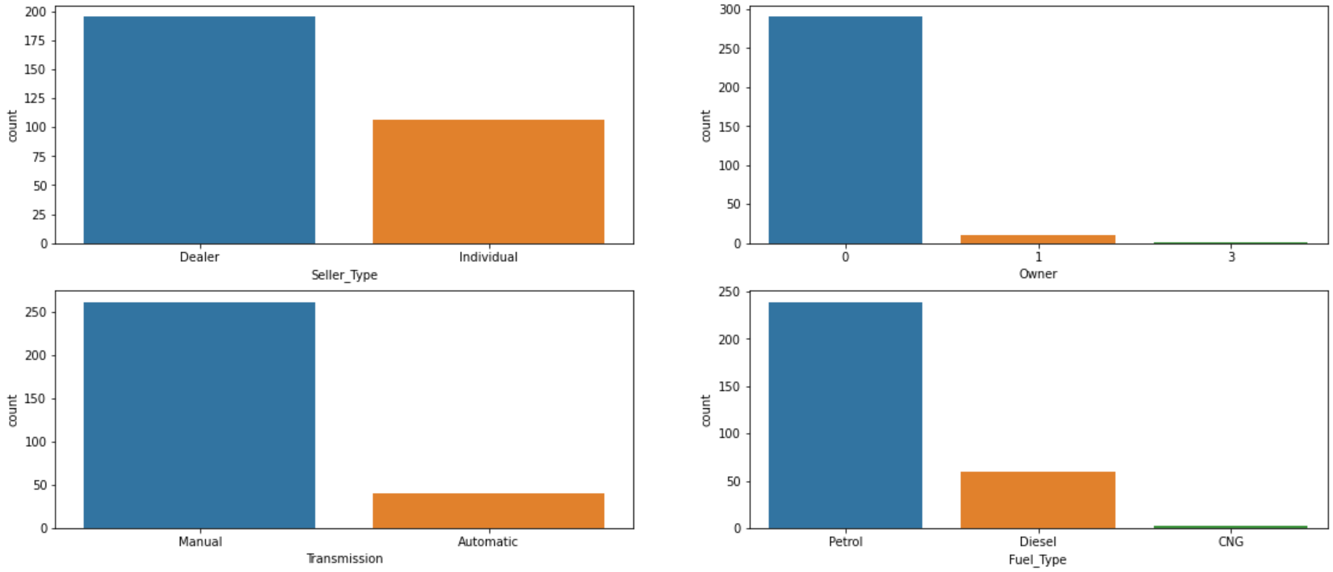
```
Car_Name         False
Year             False
Selling_Price    False
Present_Price    False
Kms_Driven       False
Fuel_Type        False
Seller_Type      False
Transmission     False
Owner            False
dtype: bool
```

This implies we do not have any missing value

```
f, axes = plt.subplots(2,2, figsize=(19,8))
sns.countplot(x='Transmission',data=df, ax=axes[1,0])
```

```
sns.countplot(x='Fuel_Type',data=df,ax=axes[1,1])
sns.countplot(x='Owner',data=df,ax=axes[0,1])
sns.countplot(x='Seller_Type',data=df,ax=axes[0,0])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f38a84d8f50>
```




```
# checking the distribution of categorical data
print(df.Fuel_Type.value_counts())
print(df.Seller_Type.value_counts())
print(df.Transmission.value_counts())
```

```
Petrol    239
Diesel     60
CNG         2
Name: Fuel_Type, dtype: int64
Dealer        195
Individual    106
Name: Seller_Type, dtype: int64
Manual       261
Automatic     40
Name: Transmission, dtype: int64
```

Encoding the Categorical Data

```
# encoding "Fuel_Type" Column
df.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)

# encoding "Seller_Type" Column
df.replace({'Seller_Type':{'Dealer':0,'Individual':1}},inplace=True)
```

```
# encoding "Transmission" Columdn
df.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

```
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Tr
0	ritz	2014	3.35	5.59	27000	0	0	
1	sx4	2013	4.75	9.54	43000	1	0	
2	ciaz	2017	7.25	9.85	6900	0	0	
3	wagon r	2011	2.85	4.15	5200	0	0	
4	swift	2014	4.60	6.87	42450	1	0	

Splitting the data and Target

```
X = df.drop(['Car_Name','Selling_Price'],axis=1)
Y = df['Selling_Price']
```

```
print(X)
```

```
     Year  Present_Price  Kms_Driven  ...  Seller_Type  Transmission  Owner
0    2014           5.59       27000  ...            0             0      0
1    2013           9.54       43000  ...            0             0      0
2    2017           9.85        6900  ...            0             0      0
3    2011           4.15        5200  ...            0             0      0
4    2014           6.87       42450  ...            0             0      0
..    ...            ...         ...  ...          ...           ...    ...
296  2016          11.60       33988  ...            0             0      0
297  2015           5.90       60000  ...            0             0      0
298  2009          11.00       87934  ...            0             0      0
299  2017          12.50        9000  ...            0             0      0
300  2016           5.90        5464  ...            0             0      0

[301 rows x 7 columns]
```

```
print(Y)
```

```
0      3.35
1      4.75
2      7.25
3      2.85
4      4.60
       ...
296    9.50
297    4.00
```

```
298     3.35
299    11.50
300     5.30
Name: Selling_Price, Length: 301, dtype: float64
```

Splitting the dataset into the Training set and Test set

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state=2)
```

Model Training

```
cv=5
r2=[]
cv_score=[]
mae=[]
mse=[]
```

```
def results(model,X_train,X_test,y_train,y_test):
    model.fit(X_train,y_train)
    predicts=model.predict(X_test)
    prediction=pd.DataFrame(predicts)
    R_2=r2_score(y_test,model.predict(X_test))
    mean_sqare_E =mean_squared_error(y_test,model.predict(X_test))
    mean_abso_E =mean_absolute_error(y_test,model.predict(X_test))
    cv_mean = -cross_val_score(model,X_train,y_train,cv=cv, scoring='neg_mean_squared_error')

    # Appending results to Lists
    r2.append(r2_score(y_test,model.predict(X_test)))
    cv_score.append(-cross_val_score(model,X,Y,cv=cv, scoring='neg_mean_squared_error').mean(
    mse.append(mean_squared_error(y_test,predicts))
    mae.append(mean_absolute_error(y_test,predicts))

    # Printing results
    print(model,"\n")
    print("r^2 value :",R_2,"\n")
    print('mean square error',mean_sqare_E,"\n")
    print('mean absolute error',mean_abso_E,"\n")
    print("CV score:",cv_mean,"\n")
    print('#'*40)
    # Plot for prediction vs originals
    plt.style.use('ggplot')
    test_index=y_test.reset_index()["Selling_Price"]
    ax=test_index.plot(label="originals",figsize=(16,8),linewidth=2,color="r",marker='o')
    ax=prediction[0].plot(label = "predictions",figsize=(16,8),linewidth=2,color="b",marker='
    plt.legend(loc='upper right')
```

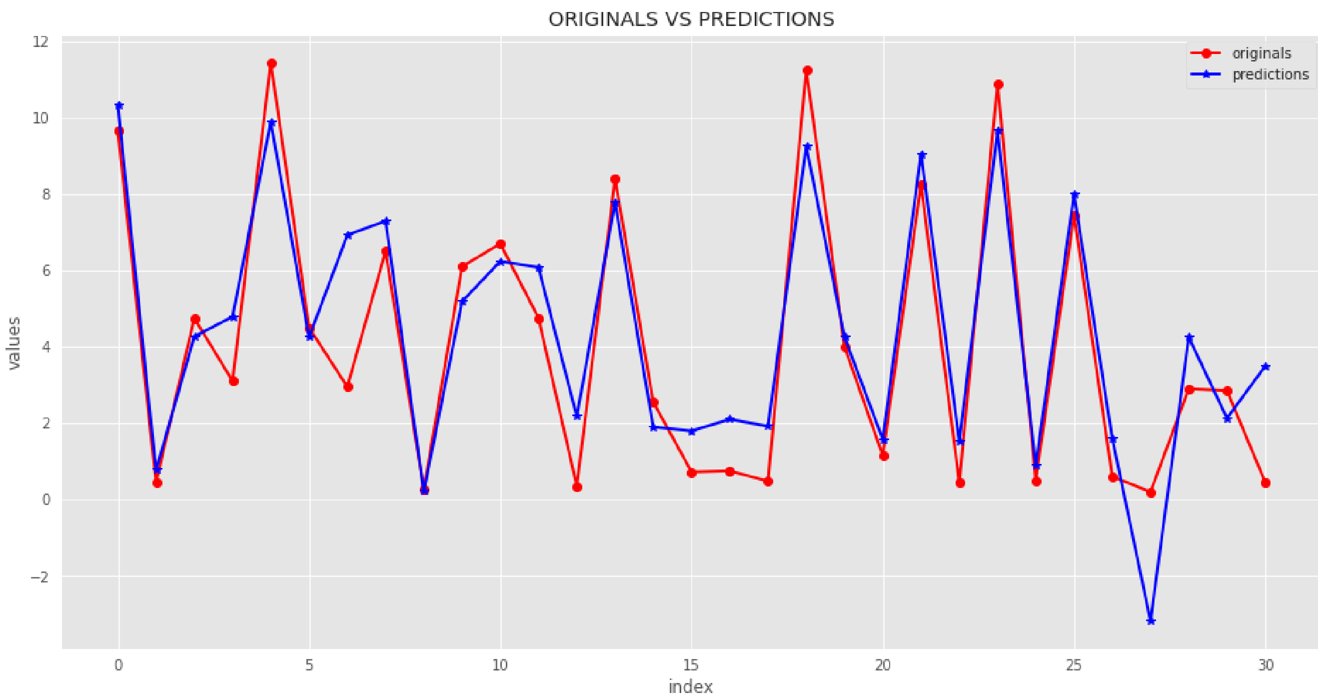
```
    plt.title("ORIGINALS VS PREDICTIONS")
    plt.xlabel("index")
    plt.ylabel("values")
    plt.show()
```

1-Linear Regression

```
# loading the linear regression model
lin_reg_model = LinearRegression()
```

```
results(lin_reg_model,X_train,X_test,Y_train,Y_test)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

r^2 value : 0.8365766715027051

mean square error 2.1501299189836294

mean absolute error 1.1516382156613783

CV score: 4.287358694010678

########################################
```

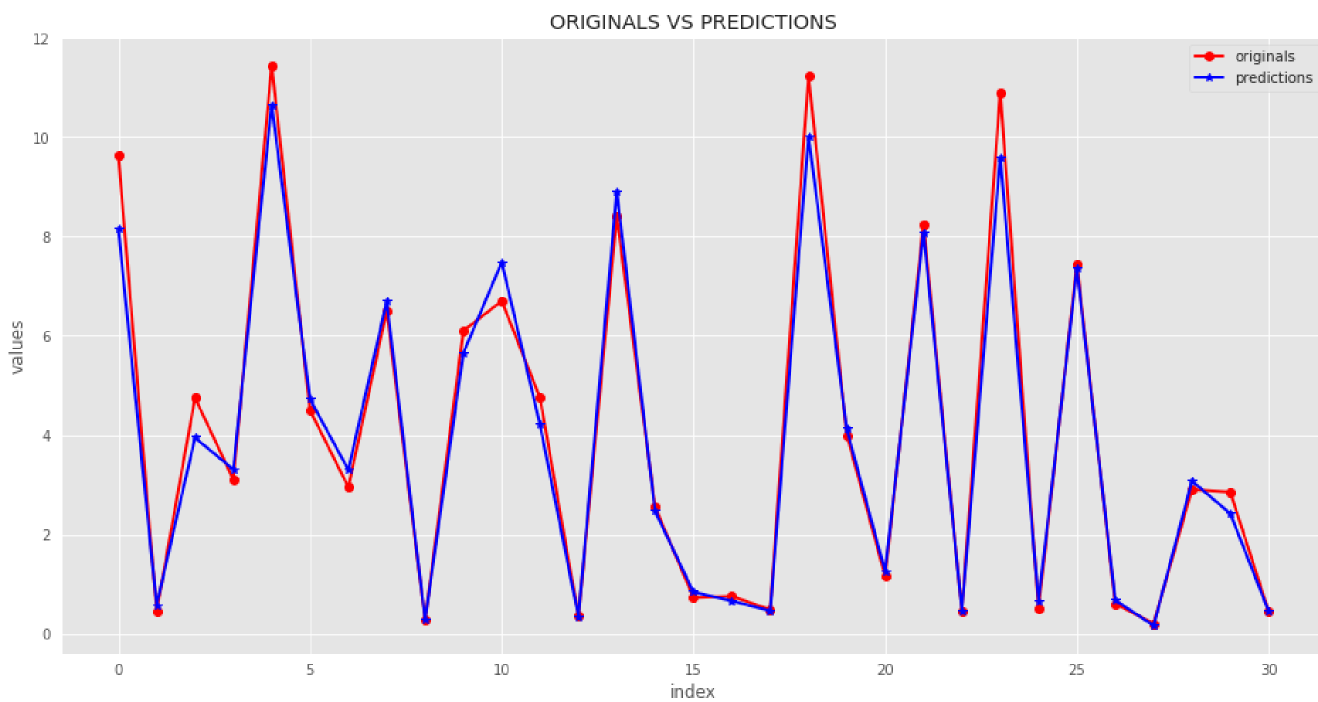


2-Random Forest

```
random_forest_model = RandomForestRegressor()
```

```
results(random_forest_model,X_train,X_test,Y_train,Y_test)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

r^2 value : 0.9787087888440023

mean square error 0.2801244506451625

mean absolute error 0.3448225806451614

CV score: 2.732727596666667

########################################
```

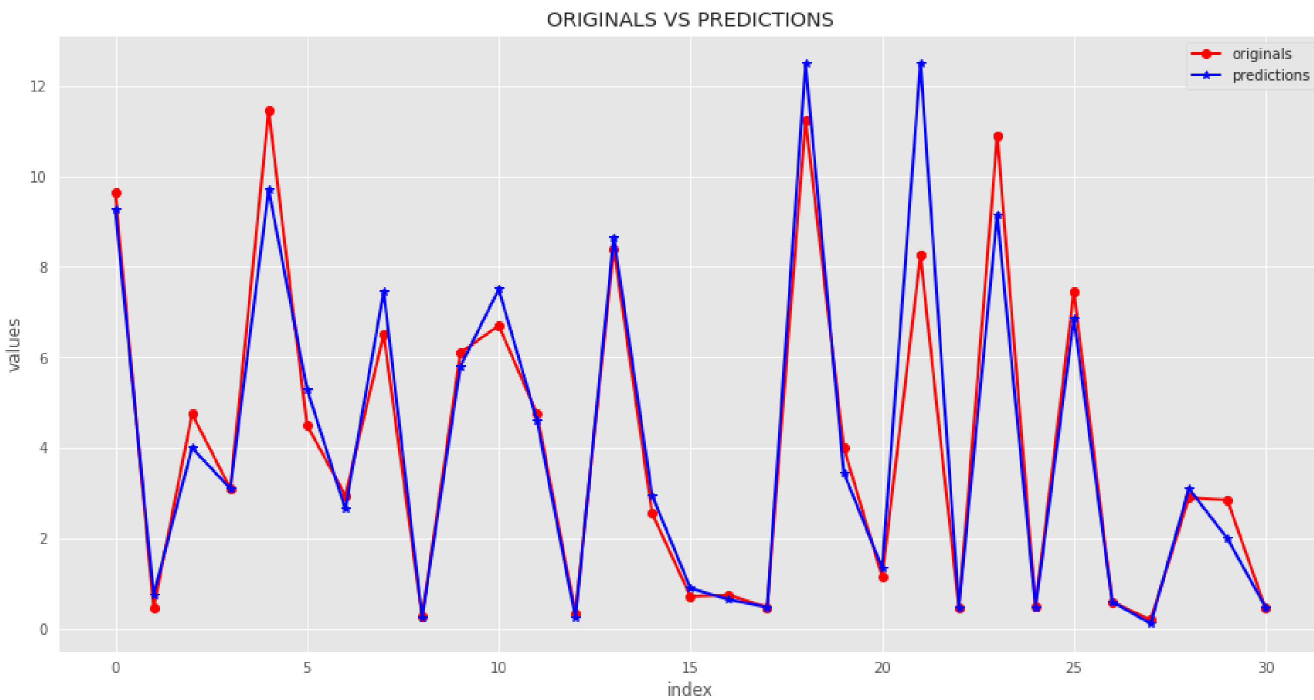


3-Decision Tree Regressor

```
decision_tree_model =DecisionTreeRegressor()
```

```
results(decision_tree_model,X_train,X_test,Y_train,Y_test)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')

r^2 value : 0.9247381133608102

mean square error 0.9902064516129032

mean absolute error 0.56

CV score: 4.100063703703704

########################################
```



Compering ALL Results

```
Results = pd.DataFrame({
    'model':['linear','random Forest','Dicision Tree'],
    'r^2':r2,
    'cv_score':cv_score,
    'mae':mae,
    'mse':mse
})
```

```
Results
```

	model	r^2	cv_score	mae	mse
0	linear	0.836577	5.249308	1.151638	2.150130
1	random Forest	0.973647	3.410647	0.375881	0.346715
2	Dicision Tree	0.949029	5.303040	0.468065	0.670610

Hence from above results we can clearly see that Random Forest Model gave the best results with least errors.