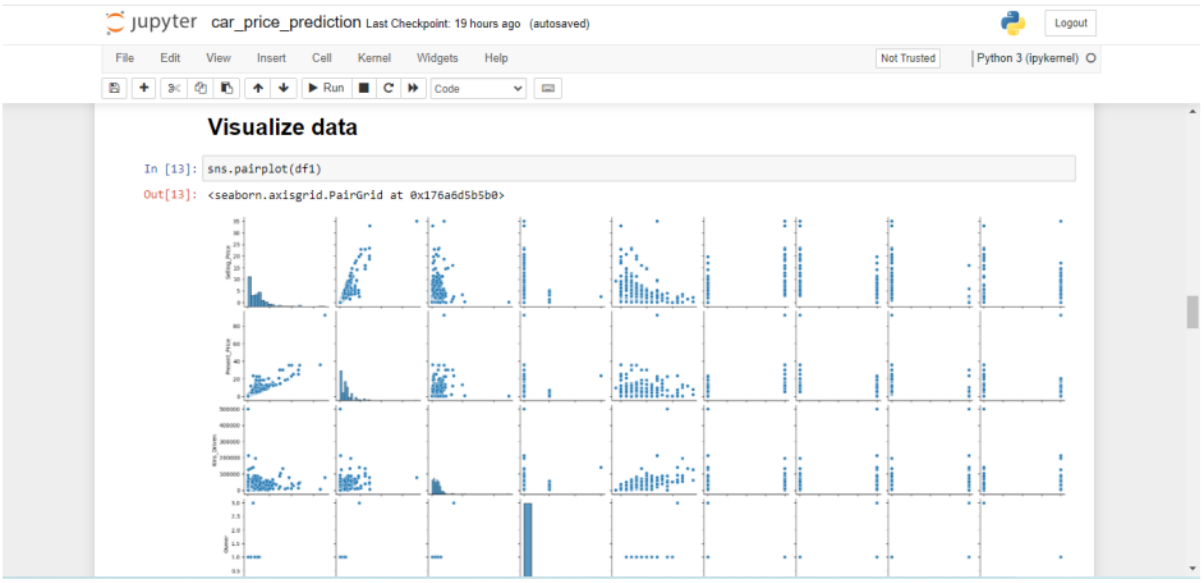
Model building ,

Model testing

Model evaluation

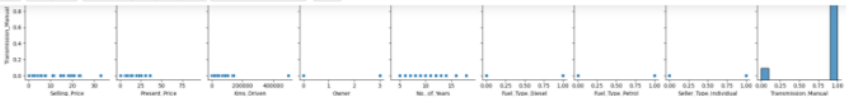


jupyter car_price_prediction Last Checkpoint: 19 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Code



Visualize the correlation using heatmap

In [14]:
```
corrmat = df1.corr()
top_corr_features = corrmat.index
plt.figure(figsize = (25,25))
h = sns.heatmap(df1[top_corr_features].corr() , annot=True , cmap='Accent')
```

divide data into dependent and independent features

In [15]:
```
X = df1.drop(columns='Selling_Price')
y = df1['Selling_Price']
```

Split data into training and testing

In [16]:
```
shuffle = StratifiedShuffleSplit(random_state=51 , test_size=0.2 , n_splits=1)

for train_index , test_index in shuffle.split(df1 , df1['Fuel_Type_Diesel'] , df1['Fuel_Type_Petrol']):
```

jupyter car_price_prediction Last Checkpoint: 19 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Code

```
for train_index , test_index in shuffle.split(df1 , df1['Fuel_Type_Diesel'] , df1['Fuel_Type_Petrol']):
    X_train_shuffle = df1.iloc[train_index]
    X_test_shuffle =  df1.iloc[test_index]
```

In [17]:
```
X_train_shuffle.shape ,  X_test_shuffle.shape
```

Out[17]: ((240, 9), (61, 9))

In [18]:
```
X_train = X_train_shuffle.drop(columns = 'Selling_Price')
y_train = X_train_shuffle['Selling_Price']
X_test = X_test_shuffle.drop(columns = 'Selling_Price')
y_test = X_test_shuffle['Selling_Price']
X_train.shape , X_test.shape , y_train.shape , y_test.shape
```

Out[18]: ((240, 8), (61, 8), (240,), (61,))

In [ ]:

Apply Hyperparameter Tuning on RandomForestRegressor

In [21]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
```

jupyter car_price_prediction Last Checkpoint: 19 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run Code

In [21]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [22]:
```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'max_dept
h': [5, 10, 15, 20, 25, 30], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [23]:
```
rfr = RandomForestRegressor()
```

jupyter car_price_prediction Last Checkpoint: 19 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [24]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rfr = RandomizedSearchCV(estimator = rfr, param_distributions = random_grid,scoring='neg_mean_squared_error',
                         n_iter = 10, cv = 5, verbose=2, random_state=51, n_jobs = 1)
```

In [ ]:
```
rfr.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=2, n_estimators=600; total time=   1.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=2, n_estimators=600; total time=   1.5s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=2, n_estimators=600; total time=   1.5s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=2, n_estimators=600; total time=   1.4s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=2, n_estimators=600; total time=   1.4s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   2.1s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   1.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=900; total time=   2.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=900; total time=   2.1s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=900; total time=   1.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=900; total time=   1.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=900; total time=   1.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=600; total time=   1.1s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=600; total time=   1.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=600; total time=   1.7s
```

Predict data

In [ ]:
```
pred = rfr.predict(X_test)
```

Check the r2_score

In [33]:
```
r2_score(pred , y_test)
```

Out[33]:
```
0.9656759933799154
```

Visualize the actual and predicted data

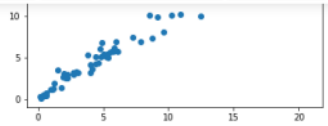
In [25]:
```
plt.scatter(y_test,pred)
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x27310569c18>
```

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code



Display the actual and predicted data

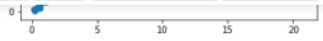
In [26]:
```
pd.DataFrame(np.c_[y_test , pred] , columns =['Actual' , 'Predicted'])
```

Out[26]:

	Actual	Predicted
0	12.50	10.006827
1	0.50	0.455760
2	0.65	0.705512
3	0.35	0.392024
4	14.90	16.263583
...	...	...
56	9.15	9.904631
57	4.75	6.072011
58	10.25	10.055072
59	0.38	0.313276

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code



Display the actual and predicted data

In [26]:
```
pd.DataFrame(np.c_[y_test , pred] , columns =['Actual' , 'Predicted'])
```

Out[26]:

	Actual	Predicted
0	12.50	10.006827
1	0.50	0.455760
2	0.65	0.705512
3	0.35	0.392024
4	14.90	16.263583
...	...	...
56	9.15	9.904631
57	4.75	6.072011
58	10.25	10.055072
59	0.38	0.313276
60	2.75	3.184455

61 rows × 2 columns