Sowmyaproject

December 5, 2024

[1]:
```
# Basic Import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Modelling
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
import warnings
```

[5]:
```
df = pd.read_csv('StudentsPerformance.csv')
df
```

[5]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
..	…	…	…	…	…	…	…	…
995	female	group E	master's degree	standard				
996	male	group C	high school	free/reduced				
997	female	group C	high school	free/reduced				
998	female	group D	some college	standard				
999	female	group D	some college	free/reduced				

```
995              completed          88             99             95
996                   none          62             55             55
997              completed          59             71             65
998              completed          68             78             77
999                   none          77             86             86

[1000 rows x 8 columns]
```

[6]:
```
df.head()
```

[6]:
```
   gender race/ethnicity parental level of education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B            master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test preparation course  math score  reading score  writing score
0                    none          72             72             74
1               completed          69             90             88
2                    none          90             95             93
3                    none          47             57             44
4                    none          76             78             75
```

[15]:
```
X = df.drop(columns=['math score'],axis=1)
```

[16]:
```
X.head()
```

[16]:
```
   gender race/ethnicity parental level of education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B            master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test preparation course  reading score  writing score
0                    none             72             74
1               completed             90             88
2                    none             95             93
3                    none             57             44
4                    none             78             75
```

[17]:
```
Y = df["math score"]
Y
```

[17]:
```
0      72
1      69
```

```
2      90
3      47
4      76
       ..
995    88
996    62
997    59
998    68
999    77
Name: math score, Length: 1000, dtype: int64
```

[27]:
```
print("Categories in 'gender' variable:    ",end=" " )
print(df['gender'].unique())

print("Categories in 'race/ethnicity' variable:   ",end=" " )
print(df['race/ethnicity'].unique())

print("Categories in 'parental level of education' variable:   ",end=" ")
print(df['parental level of education'].unique())

print("Categories in 'lunch' variable:   ",end=" " )
print(df['lunch'].unique())

print("Categories in 'test preparation course' variable:    ",end="  " )
print(df['lunch'].unique())

print("Categories in 'test preparation course' variable:   ",end="  " )
print(df['test preparation course'].unique())
```

```
Categories in 'gender' variable:     ['female' 'male']
Categories in 'race/ethnicity' variable:    ['group B' 'group C' 'group A'
'group D' 'group E']
Categories in 'parental level of education' variable:    ["bachelor's degree"
'some college' "master's degree" "associate's degree"
 'high school' 'some high school']
Categories in 'lunch' variable:    ['standard' 'free/reduced']
Categories in 'test preparation course' variable:      ['standard'
'free/reduced']
Categories in 'test preparation course' variable:     ['none' 'completed']
```

[28]:
```
y = df['math score']
```

[29]:
```
y
```

[29]:
```
0      72
1      69
2      90
3      47
```

```
4      76
       ..
995    88
996    62
997    59
998    68
999    77
Name: math score, Length: 1000, dtype: int64
```

[31]:
```
#creation of transformer columns

num_cols=X.select_dtypes(exclude="object").columns
cat_cols=X.select_dtypes(include="object").columns

from sklearn.preprocessing import OneHotEncoder , StandardScaler
from sklearn.compose import ColumnTransformer

num_trans=StandardScaler()
oh_tran=OneHotEncoder()

preprocessor=ColumnTransformer(
    [
        ("OneHotEncoder",oh_tran,cat_cols),
        ("StandardScaler",num_trans,num_cols),
    ]
)
```

[32]:
```
x=preprocessor.fit_transform(X)
```

[33]:
```
x
```

[33]:
```
array([[ 1.        ,  0.        ,  0.        , …,  1.        ,
         0.19399858,  0.39149181],
       [ 1.        ,  0.        ,  0.        , …,  0.        ,
         1.42747598,  1.31326868],
       [ 1.        ,  0.        ,  0.        , …,  1.        ,
         1.77010859,  1.64247471],
       …,
       [ 1.        ,  0.        ,  0.        , …,  0.        ,
         0.12547206, -0.20107904],
       [ 1.        ,  0.        ,  0.        , …,  0.        ,
         0.60515772,  0.58901542],
       [ 1.        ,  0.        ,  0.        , …,  1.        ,
         1.15336989,  1.18158627]])
```

[34]:
```
x.shape
```

[34]:
```
(1000, 19)
```

[35]:
```
#seperating train and test for the data:

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,Y,test_size=0.
 ↪2,random_state=23)
```

[44]:
```
#create an evaluate function to give all metrics after model training:

def evaluate_model(true,predicted):
    mae=mean_absolute_error(true,predicted)
    mse=mean_squared_error(true,predicted)
    rmse=np.sqrt(mean_squared_error(true,predicted))
    r2=r2_score(true,predicted)
    return mae,mse,rmse,r2
```

[96]:
```
models={
    "LR":LinearRegression(),
    "Lasso":Lasso(),
    "Ridge":Ridge(),
    "KNN":KNeighborsRegressor(),
    "DT":DecisionTreeRegressor(),
    "RF":RandomForestRegressor()
}

model_list=[]
r2_list=[]

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(x_train, y_train)

    #MAke predictions
    y_train_predicted = model.predict(x_train)
    y_test_predicted = model.predict(x_test)

    #Evaluate train and test dataset
    model_train_mae=evaluate_model(y_train, y_train_predicted)
    model_train_mse=evaluate_model(y_train, y_train_predicted)
    model_train_rmse=evaluate_model(y_train, y_train_predicted)
    model_train_r2=evaluate_model(y_train, y_train_predicted)

    model_test_mae=evaluate_model(y_test, y_test_predicted)
    model_test_rmse=evaluate_model(y_test, y_test_predicted)
    model_test_r2=evaluate_model(y_test, y_test_predicted)
    print(list(models.keys())[i])
```

```
    model_list.append(list(models.keys())[i])

    print('Model performance for training set')
    print("-Root mean squared error: (:.4f)".format(model_train_rmse))
    print("-Mean absolute error: (:.4f)".format(model_train_mae))
    print("-R2 score: (:.4f)".format(model_train_r2))

    print('----------------------------------------')

    print('Model performance for test set')
    print("-Root mean squared error: (:.4f)".format(model_test_rmse))
    print("-Mean absolute error: (:.4f)".format(model_test_mae))
    print("-R2 score: (:.4f)".format(model_test_r2))
    r2_list.append(model_test_r2)

    print('='*35)
    print('\n')
```

```
LR
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
===================================


Lasso
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
===================================


Ridge
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
```

```
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
==================================


KNN
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
==================================


DT
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
==================================


RF
Model performance for training set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
----------------------------------------
Model performance for test set
-Root mean squared error: (:.4f)
-Mean absolute error: (:.4f)
-R2 score: (:.4f)
==================================
```

[83]:
```
pd.DataFrame(list(zip(model_list, r2_list)),columns=['Model Name','R2_Score']).
↪sort_values(by=["R2_Score"],ascending=False)
```

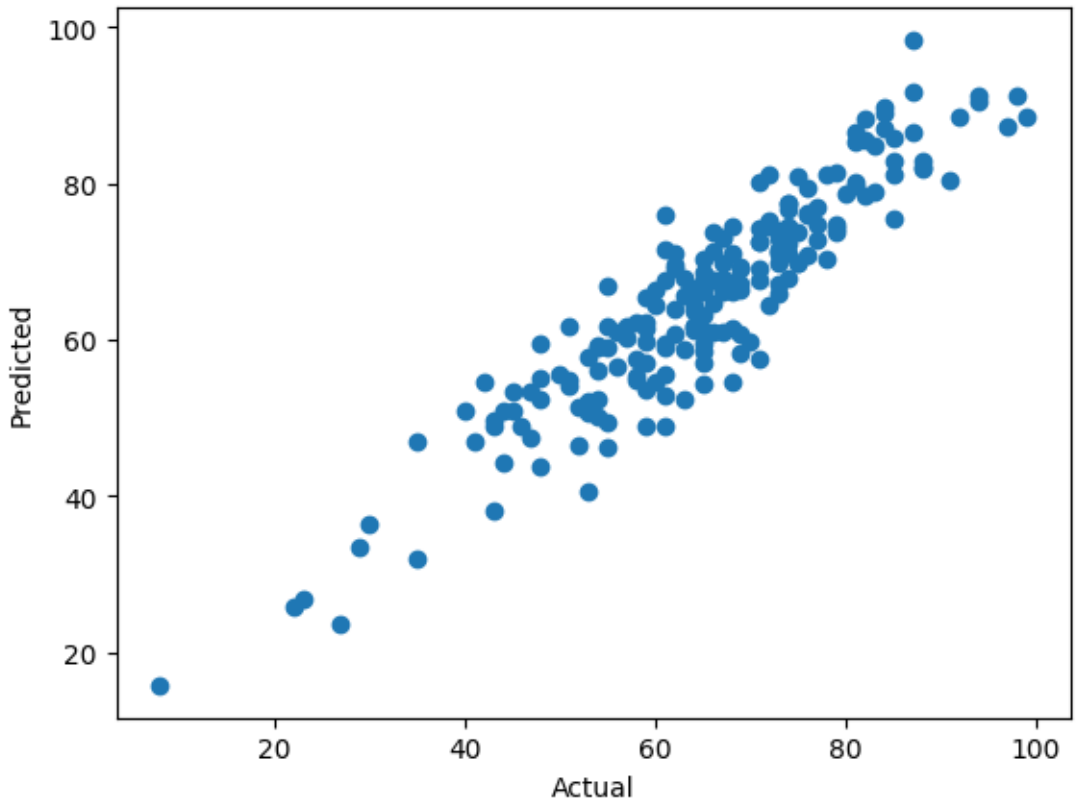
[83]:

	Model Name	R2_Score
4	DT	(6.95, 74.23, 8.615683373940804, 0.65327196267…
3	KNN	(5.730999999999999, 50.6826, 7.119171300088234…
5	RF	(5.1141, 41.370517, 6.431991682208552, 0.80675…
1	Lasso	(5.100486061635123, 43.51661877787512, 6.59671…
0	LR	(4.57875, 31.616044921875, 5.6228146796666705,…
2	Ridge	(4.567206455474207, 31.57418254990426, 5.61909…

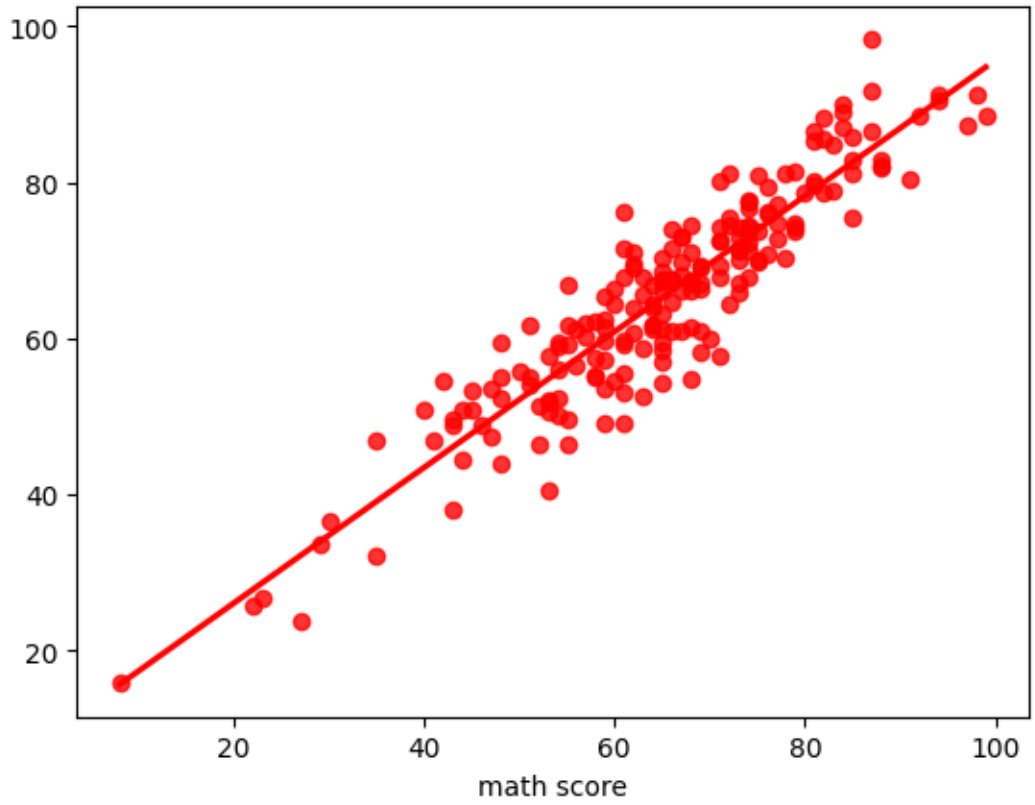
[86]:
```
lin_model = LinearRegression(fit_intercept=True)
lin_model = lin_model.fit(x_train, y_train)
y_prediction = lin_model.predict(x_test)
score = r2_score(y_test, y_prediction)*100
print("Accuracy of the model is %.2f" %score)
```

```
Accuracy of the model is 85.23
```

[88]:
```
plt.scatter(y_test,y_prediction);
plt.xlabel('Actual');
plt.ylabel('Predicted');
```



[91]:
```
sns.regplot(x=y_test,y=y_prediction,ci=None,color = 'red');
```



[92]:
```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':
  ↪y_prediction,'Difference':y_test-y_prediction})
pred_df
```

[92]:

	Actual Value	Predicted value	Difference
519	67	66.03125	0.96875
837	75	69.87500	5.12500
208	74	67.93750	6.06250
525	68	70.96875	-2.96875
978	55	49.53125	5.46875
..	…	…	…
647	64	61.65625	2.34375
481	52	46.50000	5.50000
134	74	73.43750	0.56250
366	69	58.34375	10.65625
879	64	66.75000	-2.75000

[200 rows x 3 columns]

[ ]: