Students Performance on a Major Exam :

This Dataset is regarding Students Performance based on different factors. The independent features are Gender, Ethinicity , Parents Education , test preparation course and lunch. The dependent variables are math scores , reading scores and writing scores.(overall score) But , I created one dependent variable called 'AvgScore' which is average of math, reading and writing scores.

we need to analyse how these independent variables are effecting the 'Avgscore' of the Students.

Ethinicity has 5 unique values where students are grouped based on their ethinicity. Parent Education has 6 unique levels based on their level of education, which are ordinal.

Changing the Directory :

In [1]:
```
import os
```

In [2]:
```
os.chdir(r'C:\Users\shahe\Desktop\MachineLearning\Project\LinearRegression')
os.getcwd()
```

Out[2]:
```
'C:\\Users\\shahe\\Desktop\\MachineLearning\\Project\\LinearRegression'
```

To Supress the warnings :

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

Importing Libraries :

In [4]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Reading the Data using Pandas Dataframe :

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

Saving a Copy of the Data :

In [276]:

```python
df_original = df.copy()
```

In [277]:

```python
df.head(5)
```

Out[277]:

	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

Checking Missing Values :

In [8]:

```python
df.isna().mean()
```

Out[8]:

```
gender                         0.0
race/ethnicity                 0.0
parental level of education    0.0
lunch                          0.0
test preparation course        0.0
math score                     0.0
reading score                  0.0
writing score                  0.0
dtype: float64
```

Checking for Duplicates :

In [9]:

```python
df.duplicated().sum()
```

Out[9]:

```
0
```

Checking the Structure of the Data :

In [10]:
```python
## UNDERSTANDING THE DATA

### Checking the Shape of the Data Frame.

print ( f" The Shape of the Data Set: {df.shape} \n")
print ( f" Number of Observations: {df.shape[0]}\n " )
print ( f" Number of Columns: {df.shape[1]}\n " )
```

```
 The Shape of the Data Set: (1000, 8)

 Number of Observations: 1000

 Number of Columns: 8
```

Checking Column names :

In [11]:
```python
df.columns
```

Out[11]:
```
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
       'test preparation course', 'math score', 'reading score',
       'writing score'],
      dtype='object')
```

Changing the column names to simple ones :

In [6]:
```python
df.columns=['Gender','Ethnicity','ParentEdu','Lunch','TestPrepCourse','MathScore',
            'ReadingScore','WritingScore']
```

Creating a Dependent variable called 'AvgScore' by adding Math, Reading and Writing Scores and dividing it by 3

In [7]:
```
df['AvgScore'] = (df['MathScore'] + df['ReadingScore'] + df['WritingScore']) / 3
df.head(5)
```

Out[7]:

	Gender	Ethnicity	ParentEdu	Lunch	TestPrepCourse	MathScore	ReadingScore	WritingSco
0	female	group B	bachelor's degree	standard	none	72	72	
1	female	group C	some college	standard	completed	69	90	
2	female	group B	master's degree	standard	none	90	95	
3	male	group A	associate's degree	free/reduced	none	47	57	
4	male	group C	some college	standard	none	76	78	

Deleting the Features that are not needed for Modelling :

In [8]:
```
df = df.drop(['MathScore','ReadingScore','WritingScore'],axis=1)
```

Viewing the Description of the Data :

In [87]:
```
df.describe(include='all')
```

Out[87]:

	Gender	Ethnicity	ParentEdu	Lunch	TestPrepCourse	AvgScore
count	1000	1000	1000	1000	1000	1000.000000
unique	2	5	6	2	2	NaN
top	female	group C	some college	standard	none	NaN
freq	518	319	226	645	642	NaN
mean	NaN	NaN	NaN	NaN	NaN	67.770667
std	NaN	NaN	NaN	NaN	NaN	14.257326
min	NaN	NaN	NaN	NaN	NaN	9.000000
25%	NaN	NaN	NaN	NaN	NaN	58.333333
50%	NaN	NaN	NaN	NaN	NaN	68.333333
75%	NaN	NaN	NaN	NaN	NaN	77.666667
max	NaN	NaN	NaN	NaN	NaN	100.000000

In [88]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          1000 non-null   object 
 1   Ethnicity       1000 non-null   object 
 2   ParentEdu       1000 non-null   object 
 3   Lunch           1000 non-null   object 
 4   TestPrepCourse  1000 non-null   object 
 5   AvgScore        1000 non-null   float64
dtypes: float64(1), object(5)
memory usage: 47.0+ KB
```

UNIVARIATE ANALYSIS :

In [15]:

```
df['Gender'].value_counts()
```

Out[15]:

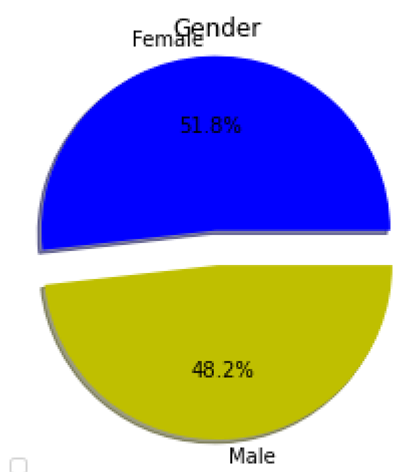
```
female    518
male      482
Name: Gender, dtype: int64
```

In [16]:

```python
Gender=df['Gender'].value_counts()
values = [Gender[0],Gender[1]]
colors = ['b', 'y']
labels = ['Female','Male']
explode = (0.2, 0)
plt.title('Gender')
plt.legend(labels,loc=3)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

Out[16]:

```
([<matplotlib.patches.Wedge at 0x1ad31c5d190>,
  <matplotlib.patches.Wedge at 0x1ad31c5dac0>],
 [Text(-0.07347412204716319, 1.2979220136007399, 'Female'),
  Text(0.06217041096298411, -1.0982417038160106, 'Male')],
 [Text(-0.04521484433671581, 0.7987212391389167, '51.8%'),
  Text(0.033911133252536786, -0.5990409293541875, '48.2%')])
```



In [17]:

```python
df['ParentEdu'].value_counts()
```

Out[17]:

```
some college          226
associate's degree    222
high school           196
some high school      179
bachelor's degree     118
master's degree        59
Name: ParentEdu, dtype: int64
```

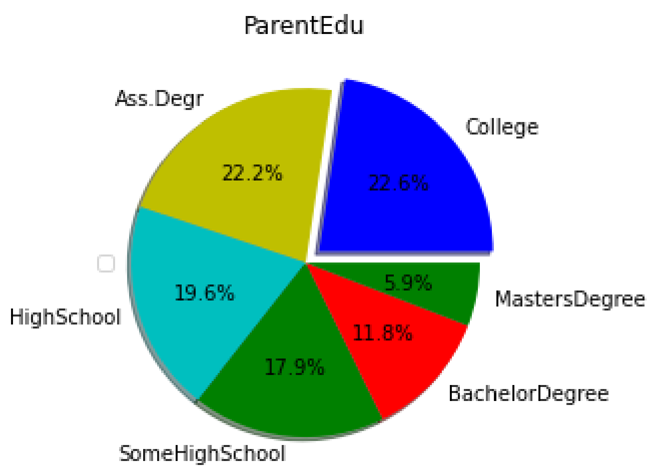
Female students outnumber the male students with percentage of 51.8 and 48.2 respectively

In [18]:

```python
ParentEdu=df['ParentEdu'].value_counts()
values = [ParentEdu[0],ParentEdu[1],ParentEdu[2],ParentEdu[3],ParentEdu[4],ParentE
colors = ['b', 'y','c','g','r','g']
labels = ['College','Ass.Degr','HighSchool','SomeHighSchool','BachelorDegree','Mas
explode = (0.1, 0,0,0,0,0)
plt.title('ParentEdu')
plt.legend(labels,loc=6)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

Out[18]:

```
([<matplotlib.patches.Wedge at 0x1ad31f7f280>,
  <matplotlib.patches.Wedge at 0x1ad31f7fa30>,
  <matplotlib.patches.Wedge at 0x1ad31f8b340>,
  <matplotlib.patches.Wedge at 0x1ad31f8bbb0>,
  <matplotlib.patches.Wedge at 0x1ad31f984c0>,
  <matplotlib.patches.Wedge at 0x1ad31f98d90>],
 [Text(0.9100343080134722, 0.7822004591141847, 'College'),
  Text(-0.5717990621018805, 0.939705183863221, 'Ass.Degr'),
  Text(-1.0543739750814827, -0.3135211646298753, 'HighSchool'),
  Text(-0.11383566476996158, -1.0940938905900084, 'SomeHighSchool'),
  Text(0.8112644257554884, -0.7428660925790178, 'BachelorDegree'),
  Text(1.0811581857178525, -0.20272389463327067, 'MastersDegree')],
 [Text(0.5308533463411921, 0.456283601149941, '22.6%'),
  Text(-0.31189039751011655, 0.5125664639253932, '22.2%'),
  Text(-0.5751130773171723, -0.1710115443435683, '19.6%'),
  Text(-0.062092180783615405, -0.5967784857763682, '17.9%'),
  Text(0.44250786859390273, -0.4051996868612824, '11.8%'),
  Text(0.5897226467551923, -0.1105766697999658, '5.9%')])
```

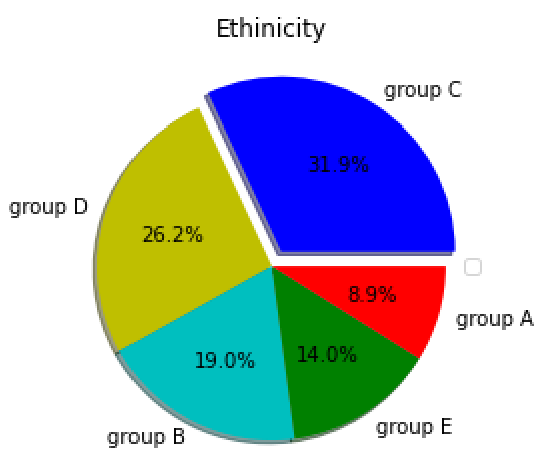


In [19]:

```
Ethnicity=df['Ethnicity'].value_counts()
values = [Ethnicity[0],Ethnicity[1],Ethnicity[2],Ethnicity[3],Ethnicity[4]]
colors = ['b', 'y','c','g','r']
labels = ['group C','group D','group B','group E','group A']
explode = (0.1, 0,0,0,0)
plt.title('Ethinicity')
plt.legend(labels,loc=5)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

Out[19]:

```
([<matplotlib.patches.Wedge at 0x1ad31fe8dc0>,
  <matplotlib.patches.Wedge at 0x1ad31ff5730>,
  <matplotlib.patches.Wedge at 0x1ad31ff5fd0>,
  <matplotlib.patches.Wedge at 0x1ad32002940>,
  <matplotlib.patches.Wedge at 0x1ad3200f250>],
 [Text(0.6461719988148862, 1.0111685062083247, 'group C'),
  Text(-1.0461621742897658, 0.3399186742226879, 'group D'),
  Text(-0.49322154359063347, -0.9832255636109514, 'group B'),
  Text(0.5952333666001212, -0.9250390474384775, 'group E'),
  Text(1.057281962489778, -0.3035701760610943, 'group A')],
 [Text(0.3769336659753503, 0.5898482952881894, '31.9%'),
  Text(-0.5706339132489631, 0.18541018593964795, '26.2%'),
  Text(-0.269029932867618, -0.536304852878700, '19.0%'),
  Text(0.32467274541824787, -0.5045667531482604, '14.0%'),
  Text(0.5766992522671516, -0.1655837323969605, '8.9%')])
```



In [20]:

```
df['Lunch'].value_counts()
```

Out[20]:

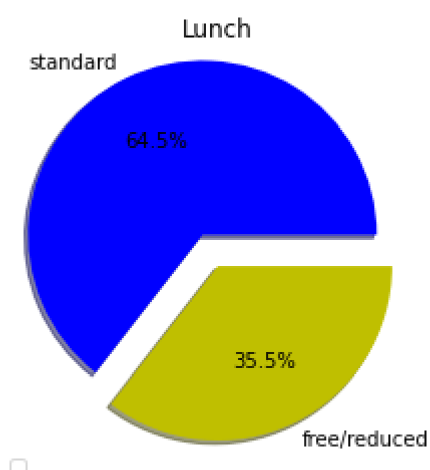
```
standard        645
free/reduced    355
Name: Lunch, dtype: int64
```

In [21]:

```
Lunch=df['Lunch'].value_counts()
values = [Lunch[0],Lunch[1]]
colors = ['b', 'y']
labels = ['standard','free/reduced']
explode = (0.2, 0)
plt.title('Lunch')
plt.legend(labels,loc=3)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

Out[21]:

```
([<matplotlib.patches.Wedge at 0x1ad32057a30>,
  <matplotlib.patches.Wedge at 0x1ad320623a0>],
 [Text(-0.5719208508586199, 1.167435882758943, 'standard'),
  Text(0.48393293516224545, -0.9878304076435662, 'free/reduced')],
 [Text(-0.35195129283607374, 0.718422081697811, '64.5%'),
  Text(0.2639634191794066, -0.5388165859873997, '35.5%')])
```



In [22]:

```
df['TestPrepCourse'].value_counts()
```

Out[22]:

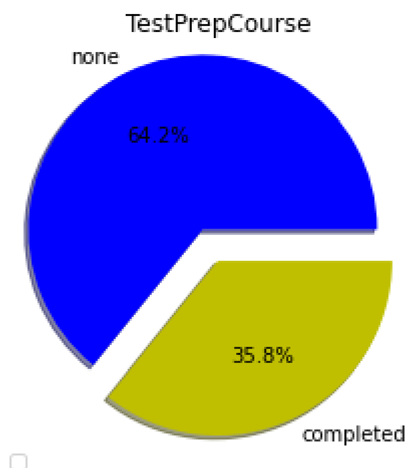
```
none         642
completed    358
Name: TestPrepCourse, dtype: int64
```

In [23]:

```python
TestPrepCourse=df['TestPrepCourse'].value_counts()
values = [TestPrepCourse[0],TestPrepCourse[1]]
colors = ['b', 'y']
labels = ['none','completed']
explode = (0.2, 0)
plt.title('TestPrepCourse')
plt.legend(labels,loc=3)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

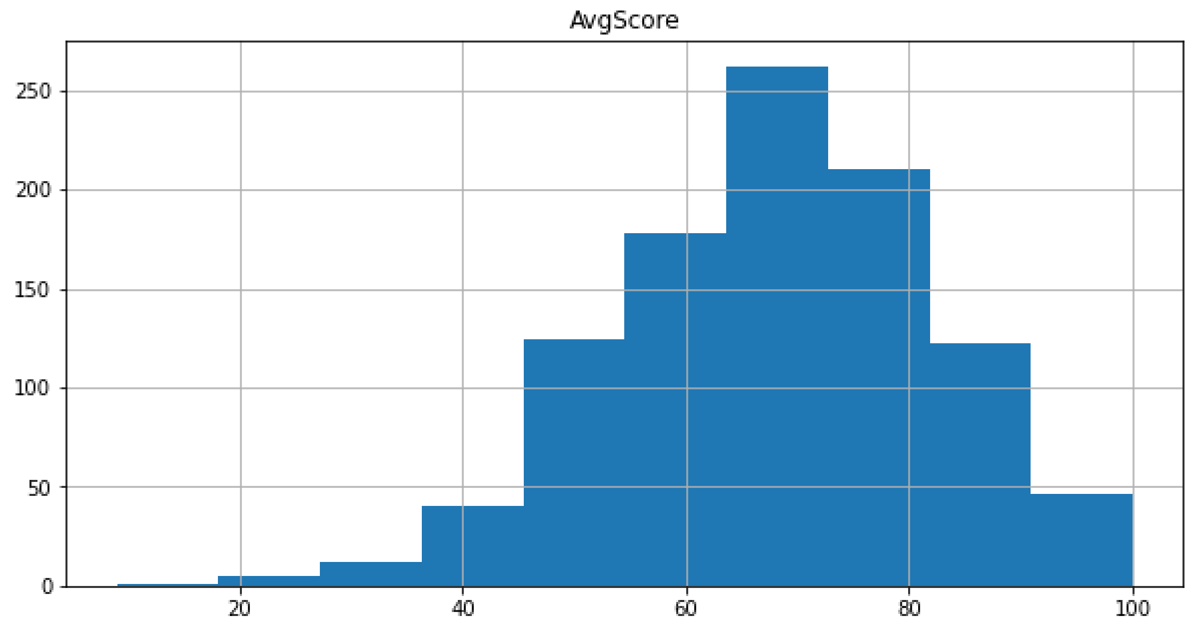
Out[23]:

```
([<matplotlib.patches.Wedge at 0x1ad320a6520>,
  <matplotlib.patches.Wedge at 0x1ad320a6e50>],
 [Text(-0.56089293141604, 1.1727741127290974, 'none'),
  Text(0.4746018041084478, -0.9923472817199666, 'completed')],
 [Text(-0.3451648808714092, 0.721707146294829, '64.2%'),
  Text(0.2588737113318806, -0.5412803354836181, '35.8%')])
```



In [106]:

```
df.hist(figsize=(10,5))
plt.show()
```




```
The Target variable shows Normal Distribution.
```

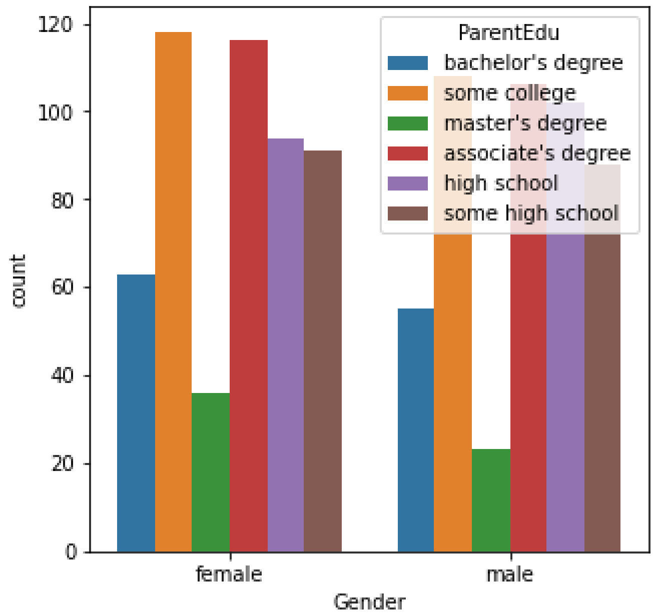
Regression models to be done:

linear regression, KNN, random forest, adaboost, SVR in sklearn and one regression model of choice in SparkML

BIVARIATE ANALYSIS :

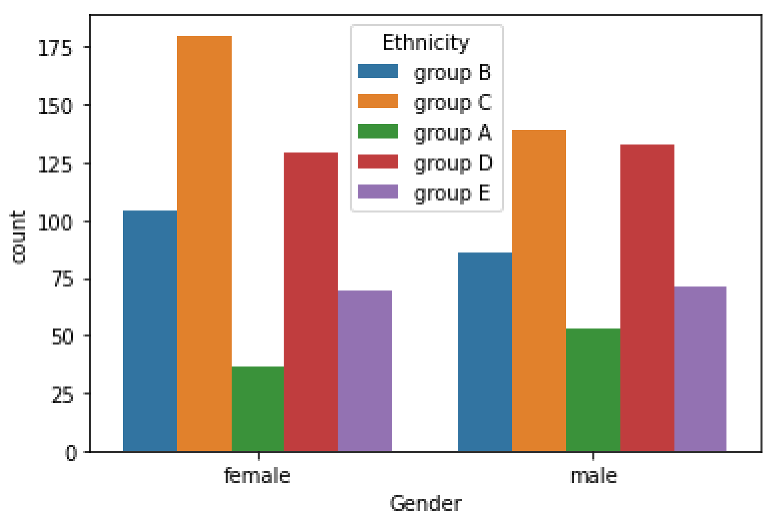
In [29]:
```
plt.figure(figsize=(5,5))
sns.countplot(x='Gender',data=df,hue='ParentEdu')
```

Out[29]: <AxesSubplot:xlabel='Gender', ylabel='count'>



In [30]:
```
sns.countplot(x='Gender',data=df,hue='Ethnicity')
```

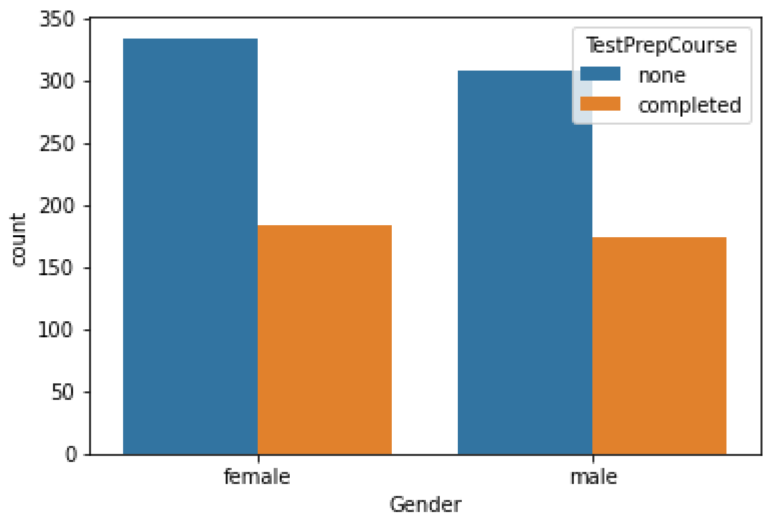
Out[30]: <AxesSubplot:xlabel='Gender', ylabel='count'>



In [31]:

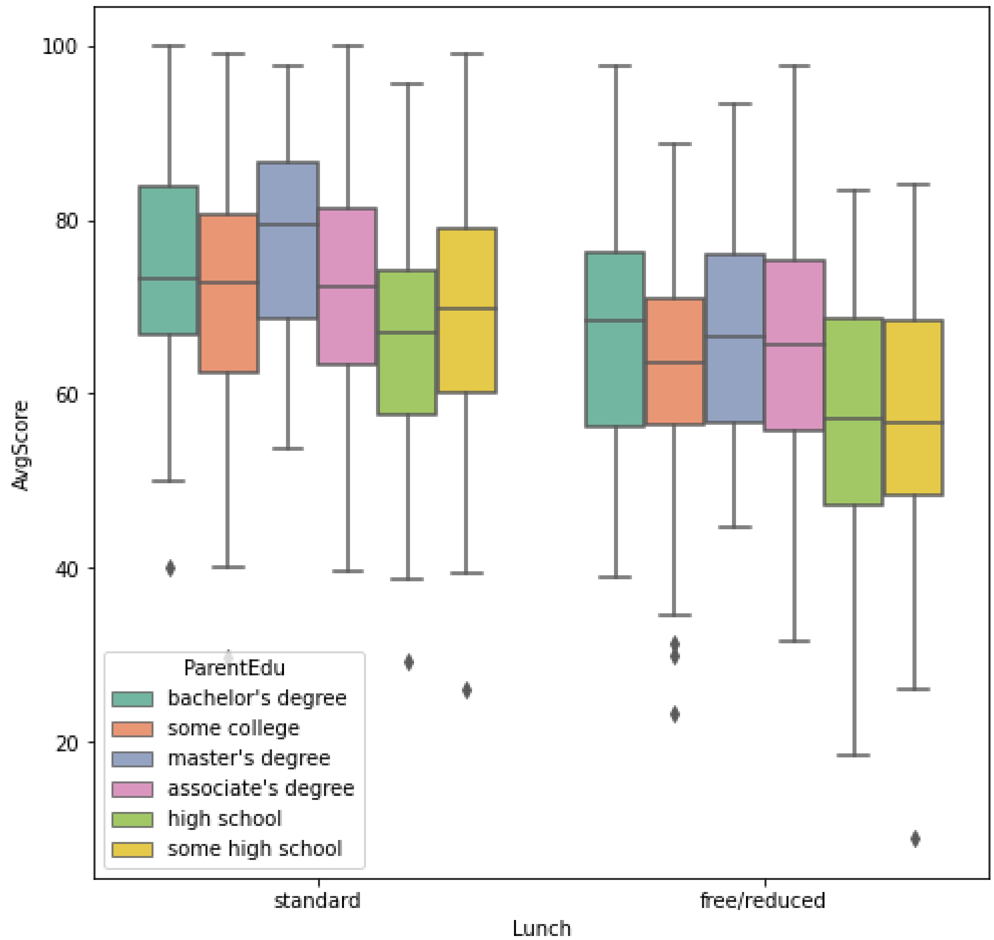
```
sns.countplot(x='Gender',data=df,hue='TestPrepCourse')
```

Out[31]: <AxesSubplot:xlabel='Gender', ylabel='count'>



In [32]:
```
plt.figure(figsize=(8,8))
sns.boxplot(x='Lunch', y='AvgScore', data=df, hue='ParentEdu',palette = 'Set2')
```

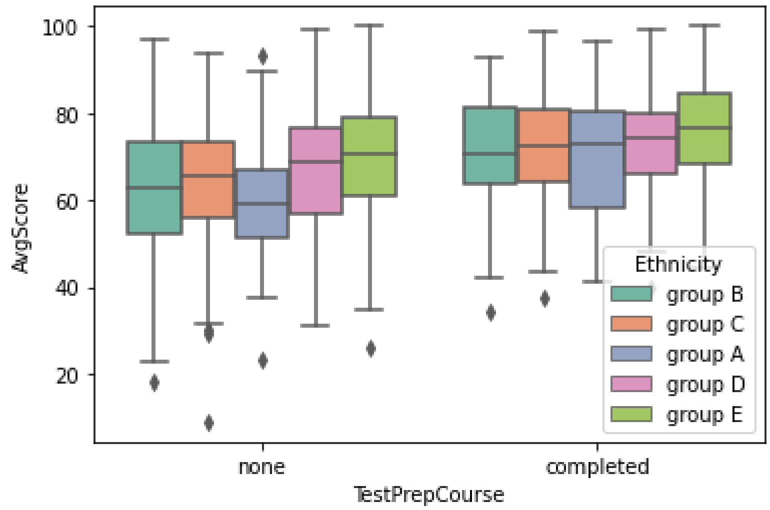
Out[32]: <AxesSubplot:xlabel='Lunch', ylabel='AvgScore'>



In [33]:

```
sns.boxplot(x = df['TestPrepCourse'],
            y = df['AvgScore'],
            hue = df['Ethnicity'],
            palette = 'Set2')
```

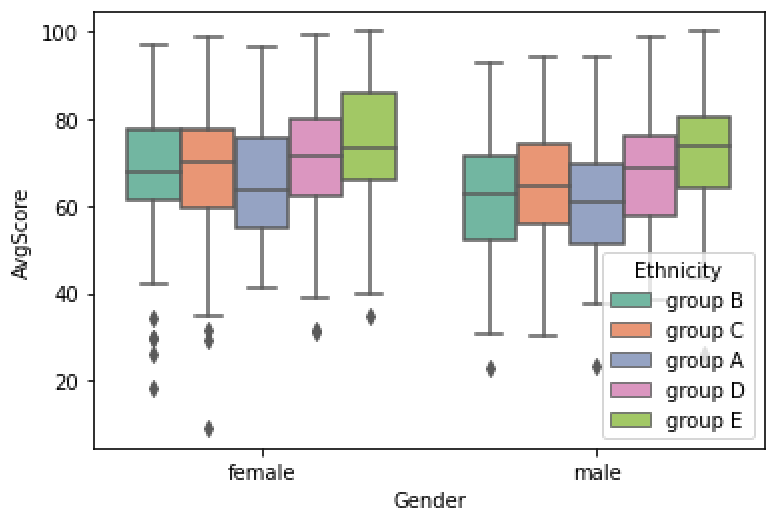
Out[33]: <AxesSubplot:xlabel='TestPrepCourse', ylabel='AvgScore'>



In [34]:

```
sns.boxplot(x = df['Gender'],
            y = df['AvgScore'],
            hue = df['Ethnicity'],
            palette = 'Set2')
```

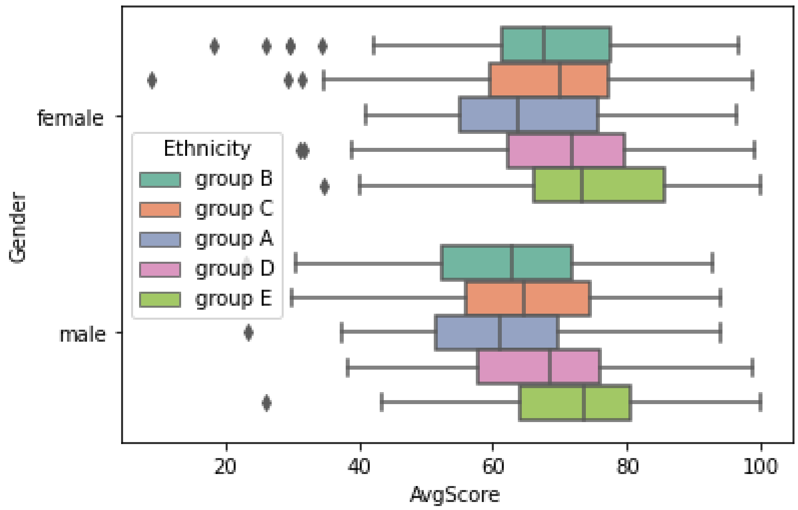
Out[34]: <AxesSubplot:xlabel='Gender', ylabel='AvgScore'>



In [35]:

```
sns.boxplot(x = df['AvgScore'],
            y = df['Gender'],
            hue = df['Ethnicity'],
            palette = 'Set2')
```

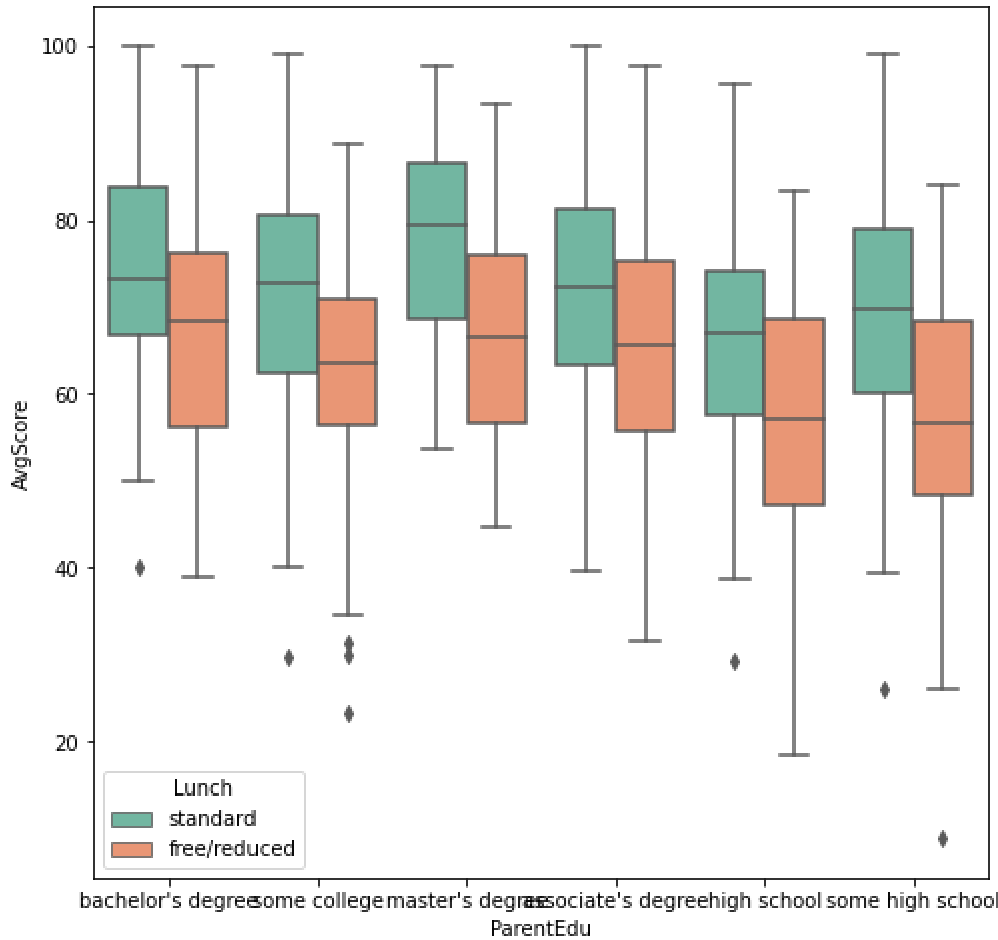
Out[35]: <AxesSubplot:xlabel='AvgScore', ylabel='Gender'>



In [36]:

```
plt.figure(figsize=(8,8))
sns.boxplot(x='ParentEdu', y='AvgScore', data=df, hue='Lunch',palette = 'Set2')
```

Out[36]: <AxesSubplot:xlabel='ParentEdu', ylabel='AvgScore'>



```
==> Male performed better than females but more 100 marks are secured by
females(strip plots)

==> Students whose parents have master's degree have performed better than others ,
all students scored above 40 (strip plots)

==> Standard Lunch's students are scoring better than other free/reduced (strip and
box plots)

==> We can't decide student's performance based on his/her race and ethnicity

==> Students who completed their course have scored better than who does not

==> As usual there always a underperformers in every class/school/college ,
definitely we also have many under performers in each category ( outliers in box
plots)

==> Math score , reading score and writing score are highly correlated if any
student is peroforming
better in anyone of subject we can say he/she will perform good in other subjects
too
```

Corerelation Analysis for Nominal Data : ChiSquare Test.

    Larger the Chisquare value , more likely the variables are related.

In [38]:

```
from scipy.stats import chi2_contingency

def chi_square(c1,c2):
    chi_2, p_val, dof, exp_val = chi2_contingency(pd.crosstab(df[c1],df[c2],margir

    print(exp_val)
    print('\nChi-square is : %f'%chi_2, '\n\np_value is : %f'%p_val, '\n\ndegree c

    if p_val < 0.05:# consider significan level is 5%
        print("\nThere is some correlation between the two variables at significar
    else:
        print("\nThere is no correlation between the two variables")
```

In [39]:

```
chi_square("Gender","ParentEdu")
```

```
[[114.996  61.124 101.528  30.562 117.068  92.722]
 [107.004  56.876  94.472  28.438 108.932  86.278]]

Chi-square is : 3.384905

p_value is : 0.640870

degree of freedom is : 5

There is no correlation between the two variables
```

In [40]:

```
chi_square("Gender","Ethnicity")
```

```
[[ 46.102  98.42  165.242 135.716  72.52 ]
 [ 42.898  91.58  153.758 126.284  67.48 ]]

Chi-square is : 9.027386

p_value is : 0.060419

degree of freedom is : 4

There is no correlation between the two variables
```

In [41]:

```
chi_square("ParentEdu","Ethnicity")
```

```
[[19.758 42.18  70.818 58.164 31.08 ]
 [10.502 22.42  37.642 30.916 16.52 ]
 [17.444 37.24  62.524 51.352 27.44 ]
 [ 5.251 11.21  18.821 15.458  8.26 ]
 [20.114 42.94  72.094 59.212 31.64 ]
 [15.931 34.01  57.101 46.898 25.06 ]]

Chi-square is : 29.458662

p_value is : 0.079113

degree of freedom is : 20

There is no correlation between the two variables
```

In [42]:

```
chi_square("ParentEdu","TestPrepCourse")
```

```
[[ 79.476 142.524]
 [ 42.244  75.756]
 [ 70.168 125.832]
 [ 21.122  37.878]
 [ 80.908 145.092]
 [ 64.082 114.918]]

Chi-square is : 9.544071

p_value is : 0.089234

degree of freedom is : 5

There is no correlation between the two variables
```

Shuffling the Data :

In [9]:

```
df = df.sample(frac = 1)
```

The independent variables Gender, Ethnicity , ParentEdu, TestPrepCourse do not show any multicollineirity.

ENCODING :

Mapping the 'ParentEdu' feature into a Ordinal numeric feature.

In [10]:

```
df.ParentEdu = df.ParentEdu.map(
    {
      "some high school" : 1,"high school"  : 2,"some college" : 3,
    "associate's degree" : 4,"bachelor's degree": 5,"master's degree": 6
    }
    )
df.Gender = df.Gender.map(
    {
      "female" :0,"male"  : 1
    }
    )
df.Lunch = df.Lunch.map(
    {
      "standard" :1,"free/reduced"  : 0
    }
    )
df.TestPrepCourse = df.TestPrepCourse.map(
    {
      "none" :0,"completed"  : 1
    }
    )
```

I am using dummies to Encode Ethinicity as it is not Ordinal. It is Nominal.

In [11]:

```
df = pd.get_dummies(df,columns=['Ethnicity'],drop_first=True)
df.head(5)
```

Out[11]:

	Gender	ParentEdu	Lunch	TestPrepCourse	AvgScore	Ethnicity_group B	Ethnicity_group C	Ethni
586	0	2	1	0	67.000000	0	0	
257	1	4	1	1	77.333333	0	1	
406	1	4	1	1	64.333333	1	0	
867	1	4	1	0	48.000000	1	0	
2	0	6	1	0	92.666667	1	0	

Standardising the Data :

In [12]:
```python
### Standardising

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
df = pd.DataFrame(sc.fit_transform(df), columns=df.columns)
```

Splitting the Data into Independent Variables (X) , Dependent Variable (y).

In [13]:
```python
#extract dependent and independent variables
X = df.drop('AvgScore',axis=1)
y = df.AvgScore
```

In [121]:
```python
X.head(5)
```

Out[121]:

	Gender	ParentEdu	Lunch	TestPrepCourse	Ethnicity_group B	Ethnicity_group C	Ethnicity_group D	Et
0	0.0	0.2	1.0	1.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
2	0.0	0.8	1.0	0.0	1.0	0.0	0.0	
3	0.0	0.2	1.0	1.0	0.0	0.0	1.0	
4	1.0	1.0	0.0	0.0	1.0	0.0	0.0	

In [122]:
```python
y.head(5)
```

Out[122]:
```
0    0.622711
1    0.498168
2    0.963370
3    0.556777
4    0.465201
Name: AvgScore, dtype: float64
```

Checking the P-values of independent variables(X) by adding constant to take care of 'bo'.

In [123]:

```
#importing OLS statsmodel to check the p-values of the X variable
import statsmodels.api as sm
X2 = sm.add_constant(X)
ols = sm.OLS(y,X2)
lr = ols.fit()
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               AvgScore   R-squared:                       0.238
Model:                            OLS   Adj. R-squared:                  0.232
Method:                 Least Squares   F-statistic:                     38.76
Date:                Thu, 27 May 2021   Prob (F-statistic):           7.46e-54
Time:                        10:00:06   Log-Likelihood:                 571.26
No. Observations:                1000   AIC:                            -1125.
Df Residuals:                     991   BIC:                            -1080.
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
=====
                         coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                  0.4931      0.018     27.993      0.000       0.459
0.528
Gender                -0.0416      0.009     -4.757      0.000      -0.059      -
0.024
ParentEdu              0.1052      0.015      7.025      0.000       0.076
0.135
Lunch                  0.0966      0.009     10.622      0.000       0.079
0.114
TestPrepCourse         0.0856      0.009      9.421      0.000       0.068
0.103
Ethnicity_group B      0.0147      0.018      0.833      0.405      -0.020
0.049
Ethnicity_group C      0.0247      0.017      1.493      0.136      -0.008
0.057
Ethnicity_group D      0.0563      0.017      3.334      0.001       0.023
0.089
Ethnicity_group E      0.0753      0.019      4.018      0.000       0.039
0.112
==============================================================================
Omnibus:                       12.060   Durbin-Watson:                   1.953
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               12.355
Skew:                          -0.265   Prob(JB):                      0.00208
Kurtosis:                       2.874   Cond. No.                         11.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Backward Elimination : Feature Elimination of variables whose p-values are

> 0.05

In [124]:
```
maxp = lr.pvalues.max()
while(maxp > 0.05):
    X2.drop(lr.pvalues.idxmax(),axis=1,inplace=True)
    ols = sm.OLS(y,X2)
    lr = ols.fit()
    maxp = lr.pvalues.max()
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               AvgScore   R-squared:                       0.236
Model:                            OLS   Adj. R-squared:                  0.232
Method:                 Least Squares   F-statistic:                     51.26
Date:                Thu, 27 May 2021   Prob (F-statistic):           4.62e-55
Time:                        10:00:09   Log-Likelihood:                 570.06
No. Observations:                1000   AIC:                            -1126.
Df Residuals:                     993   BIC:                            -1092.
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
=====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                0.5104      0.011     44.393      0.000       0.488
0.533
Gender              -0.0426      0.009     -4.890      0.000      -0.060      -
0.025
ParentEdu            0.1070      0.015      7.169      0.000       0.078
0.136
Lunch                0.0970      0.009     10.668      0.000       0.079
0.115
TestPrepCourse       0.0857      0.009      9.439      0.000       0.068
0.104
Ethnicity_group D    0.0385      0.010      3.774      0.000       0.018
0.058
Ethnicity_group E    0.0573      0.013      4.422      0.000       0.032
0.083
==============================================================================
Omnibus:                       12.740   Durbin-Watson:                   1.950
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               13.019
Skew:                          -0.269   Prob(JB):                      0.00149
Kurtosis:                       2.849   Cond. No.                         5.78
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Splitting the Data into Train(80%) and Test(20%) for Modelling :

In [14]:
```python
#For cross-validation using train-test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state =0,test_size=0.2
```

Linear Regression Model :

In [126]:
```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)# fitting the train dataset into the Linear Regression m
# X_train holds 80% of independent variables, y-train holds 80% of dependent varia
```

Out[126]: LinearRegression()

In [127]:
```python
model.score(X_test,y_test) # checking the score of the model for test dataset.
# X_test holds 20% of independent variables ,y_test holds 20% of dependent variabl
```

Out[127]: 0.2833523737357605

In [128]:
```python
y_pred = model.predict(X_test) # Predicting the y_test variable for X-test using t
```

In [129]:
```python

from sklearn.metrics import r2_score,mean_squared_error
import math

print(r2_score(y_test,y_pred)) #R^2
print(mean_squared_error(y_test,y_pred)) # MeancSquare Error (MSE)
print(math.sqrt(mean_squared_error(y_test,y_pred)))#Root Mean Square Error (RMSE)
```

```
0.2833523737357605
0.016140311270576694
0.12704452475638883
```

In [130]:
```python
#dimensions of data
n = len(X_test)
n
```

Out[130]: 200

In [131]:
```python
k = len(X_test.iloc[0])
k
```

Out[131]: 8

In [132]:
```python
# checking R2 score y-test and y-predict
R2 = r2_score(y_test,y_pred)
R2
```

Out[132]: 0.2833523737357605

In [133]:
```
#Adj R^2 is useful in multiple Linear regression
#as it accounts for number of variables in the scoring

Adj_R2 = 1 - ((n-1)*(1- R2)/(n-k-1))
print(Adj_R2)
```

```
0.2533357192325463
```

In [134]:
```
#k-fold cross validation using linear regression model

from sklearn.model_selection import cross_val_score

cross_val_score(LinearRegression(),X,y,cv=5).mean()
```

Out[134]:
```
0.2210781580927459
```

In [135]:
```
model = LinearRegression()
model.fit(X,y)
```

Out[135]:
```
LinearRegression()
```

In [136]:
```
model.intercept_
```

Out[136]:
```
0.49313409984673917
```

In [137]:
```
model.coef_
```

Out[137]:
```
array([-0.04156814,  0.1052396 ,  0.09661782,  0.08557011,  0.01473129,
        0.02472083,  0.05632688,  0.07526712])
```

KNN MODEL :

In [15]:
```
#import the knn model
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
```

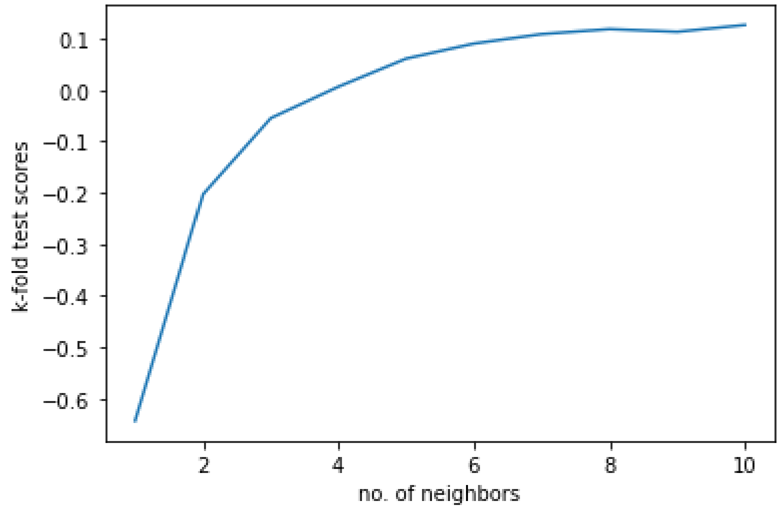
In [16]:
```
#see the cross_validated score for cv=3
from sklearn.model_selection import cross_val_score
cross_val_score(knn,X,y,cv=4).mean()
```

Out[16]:
```
0.06125887259841026
```

In [17]:
```
#for no.of neighbors from 1 - 10, graph the k-fold scores
scores = []
for i in range(1,11,1):
    knn = KNeighborsRegressor(n_neighbors=i, weights='uniform')
    scores.append(cross_val_score(knn,X,y,cv=4).mean())
```

In [18]:

```python
import matplotlib.pyplot as plt
plt.plot(range(1,11,1),scores)
plt.xlabel('no. of neighbors')
plt.ylabel('k-fold test scores')
plt.show()
```



Hyper Parameter Tuning :

In [19]:

```python
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}

knn = KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

Out[19]: {'n_neighbors': 9}

In [20]:

```python
knnmodel = KNeighborsRegressor(n_neighbors=9)
knnmodel.fit(X_train,y_train)
```

Out[20]: KNeighborsRegressor(n_neighbors=9)

In [22]:

```python
y_pred = knnmodel.predict(X_test)
```

In [23]:

```python
knnmodel.score(X_test,y_test)
```

Out[23]: 0.20178816060434024

9-NN is the Best Model.

In [24]:
```
from sklearn.metrics import r2_score,mean_squared_error
import math

print(r2_score(y_test,y_pred)) #R^2
print(mean_squared_error(y_test,y_pred)) #MSE
print(math.sqrt(mean_squared_error(y_test,y_pred)))#RMSE
```

```
0.20178816060434024
0.018434371971205513
0.1357732373157741
```

In [25]:
```
#k-fold cross validation using linear regression model

from sklearn.model_selection import cross_val_score

cross_val_score(KNeighborsRegressor(),X,y,cv=5).mean()
```

Out[25]:
```
0.0608421868841317446
```

In [26]:
```
#dimensions of data
n = len(X_test)
n
```

Out[26]:
```
200
```

In [27]:
```
k = len(X_test.iloc[0])
k
```

Out[27]:
```
8
```

In [28]:
```
# checking R2 score y-test and y-predict
R2 = r2_score(y_test,y_pred)
R2
```

Out[28]:
```
0.20178816060434024
```

In [29]:
```
#Adj R^2 is useful in multiple Linear regression
#as it accounts for number of variables in the scoring

Adj_R2 = 1 - ((n-1)*(1- R2)/(n-k-1))
print(Adj_R2)
```

```
0.16835520398043824
```

In [30]:
```
model = KNeighborsRegressor()
model.fit(X,y)
```

Out[30]:
```
KNeighborsRegressor()
```

Random Forest Regressor Model :

In [158]:

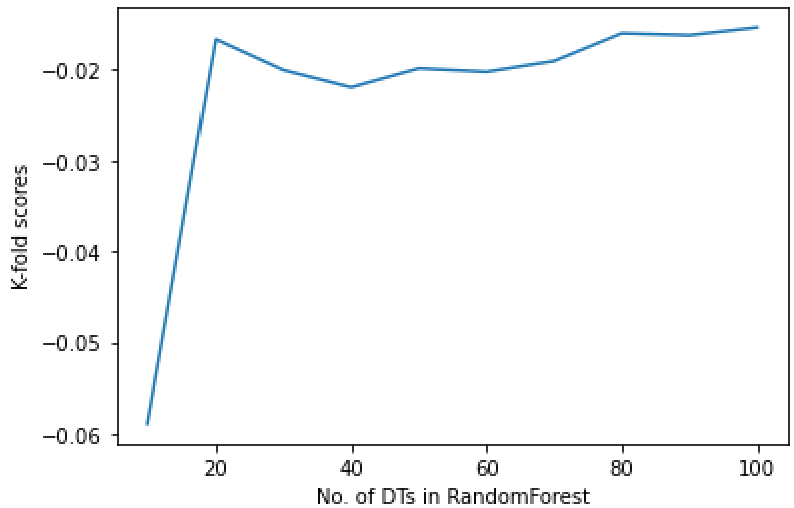
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
```

In [159]:

```
#Graph k-fold score vs no. of estimators in Random Forest
scores = []
for i in range(10,101,10):
    scores.append(cross_val_score(RandomForestRegressor(n_estimators=i,random_stat
                              X,y,cv=4).mean())
```

In [160]:

```
plt.plot(range(10,101,10),scores)
plt.xlabel('No. of DTs in RandomForest')
plt.ylabel('K-fold scores')
plt.show()
```



Hyper parameter Tuning :

In [161]:

```
params = {
            'n_estimators': [10,20,30,40,50,60,70,80],
            'max_depth': [1,2,3,4,5,6,7,8,9,10,11,12]
        }
model = GridSearchCV(RandomForestRegressor(random_state=0), params,cv=4)
model.fit(X,y)
```

Out[161]:

```
GridSearchCV(cv=4, estimator=RandomForestRegressor(random_state=0),
             param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
                         'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80]})
```

In [162]:

```
model.best_params_
```

Out[162]:

```
{'max_depth': 3, 'n_estimators': 20}
```

In [163]:
```python
model.best_score_
```
Out[163]:
```
0.19314888446684425
```

In [164]:
```python
best_model = model.best_estimator_
```

In [165]:
```python
best_model.fit(X_train,y_train)
```
Out[165]:
```
RandomForestRegressor(max_depth=3, n_estimators=20, random_state=0)
```

In [166]:
```python
best_model.score(X_test,y_test)
```
Out[166]:
```
0.23699077328682305
```

In [167]:
```python
y_pred = model.predict(X_test)
```

In [168]:
```python
cross_val_score(RandomForestRegressor(n_estimators=80,max_depth=6),X,y,cv=4).mean(
```
Out[168]:
```
0.12083134092522954
```

In [169]:
```python
from sklearn.metrics import r2_score,mean_squared_error
import math

print(r2_score(y_test,y_pred)) #R^2
print(mean_squared_error(y_test,y_pred)) #MSE
print(math.sqrt(mean_squared_error(y_test,y_pred)))#RMSE
```
```
0.23699077328682305
0.017184465517132524
0.13108953244684537
```

Support Vector Regressor (SVR) MODEL :

In [170]:
```python
#https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
from sklearn.svm import SVR
```

In [171]:
```python
from sklearn.model_selection import GridSearchCV
```

Hyper Parameter Tuning :

In [172]:
```
svr = SVR()
params = {
                'C' : [20,30,40,50,60],
               'kernel': ['linear', 'rbf','poly','sigmoid'],
               'degree': [2,3],
               'gamma' : [0.001,0.005,0.1]
         }
model = GridSearchCV(svr, params,cv=4)
```

In [173]:
```
model.fit(X,y)
```

Out[173]:
```
GridSearchCV(cv=4, estimator=SVR(),
             param_grid={'C': [20, 30, 40, 50, 60], 'degree': [2, 3],
                         'gamma': [0.001, 0.005, 0.1],
                         'kernel': ['linear', 'rbf', 'poly', 'sigmoid']})
```

In [174]:
```
model.best_params_
```

Out[174]:
```
{'C': 50, 'degree': 2, 'gamma': 0.005, 'kernel': 'sigmoid'}
```

In [175]:
```
model.best_score_
```

Out[175]:
```
0.21849914966829095
```

In [176]:
```
best_model = model.best_estimator_
```

In [177]:
```
best_model.fit(X_train,y_train)
```

Out[177]:
```
SVR(C=50, degree=2, gamma=0.005, kernel='sigmoid')
```

In [178]:
```
best_model.score(X_test,y_test)
```

Out[178]:
```
0.27883608969931306
```

In [179]:
```
y_pred = model.predict(X_test)
```

In [180]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(SVR(kernel='rbf',C=10,gamma=0.001),X,y,cv=4).mean()
```

Out[180]:
```
0.2148690346229981
```

In [181]:

```
from sklearn.metrics import r2_score,mean_squared_error
import math

print(r2_score(y_test,y_pred)) #R^2
print(mean_squared_error(y_test,y_pred)) #MSE
print(math.sqrt(mean_squared_error(y_test,y_pred)))#RMSE
```

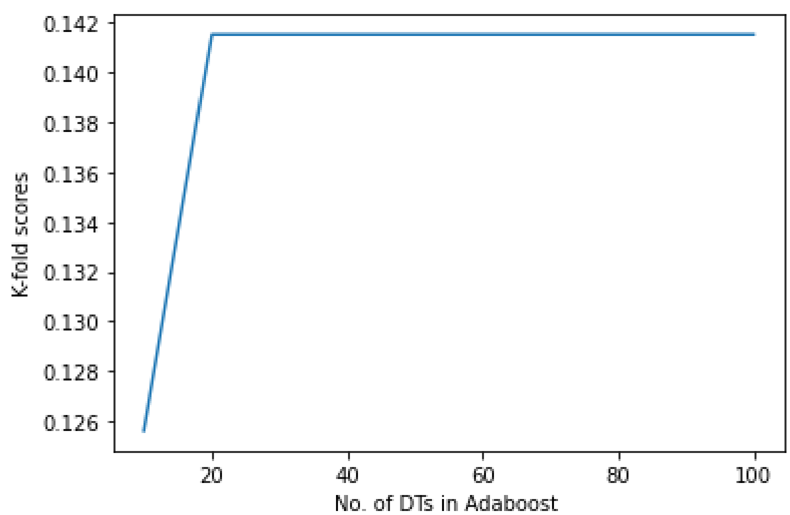
```
0.27883608969931306
0.01624202685221419
0.12744421074420836
```

AdaBoost Regressor :

In [182]:

```
from sklearn.ensemble import AdaBoostRegressor

#Graph k-fold score vs no. of estimators in Adaboost which uses DT as base estimat
scores = []
for i in range(10,101,10):
    scores.append(cross_val_score(AdaBoostRegressor(n_estimators=i,random_state=0)
                                  X_train,y_train,cv=4).mean())
plt.plot(range(10,101,10),scores)
plt.xlabel('No. of DTs in Adaboost ')
plt.ylabel('K-fold scores')
plt.show()
```



Hyper Parameter Tuning :

In [183]:
```
from sklearn.tree import DecisionTreeRegressor
#including other params like max_depth, we will apply gridsearch to fine the best
params = {
            'n_estimators': [4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100],
            'base_estimator': [  DecisionTreeRegressor(max_depth=1,random_state=0),
                                DecisionTreeRegressor(max_depth=2,random_state=0),
                                DecisionTreeRegressor(max_depth=3,random_state=0),
                                DecisionTreeRegressor(max_depth=4,random_state=0),
                                DecisionTreeRegressor(max_depth=5,random_state=0),
                                DecisionTreeRegressor(max_depth=6,random_state=0),
                                DecisionTreeRegressor(max_depth=7,random_state=0),
                                DecisionTreeRegressor(max_depth=8,random_state=0)]
        }
model = GridSearchCV(AdaBoostRegressor(random_state=0), params,cv=4)
model.fit(X_train,y_train)
```

Out[183]:
```
GridSearchCV(cv=4, estimator=AdaBoostRegressor(random_state=0),
             param_grid={'base_estimator': [DecisionTreeRegressor(max_depth=1,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=2,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=3,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=4,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=5,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=6,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=7,
                                                                     random_state=
0),
                                            DecisionTreeRegressor(max_depth=8,
                                                                     random_state=
0)],
                         'n_estimators': [4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50,
                                          60, 70, 80, 90, 100]})
```

In [184]:
```
model.best_params_
```

Out[184]:
```
{'base_estimator': DecisionTreeRegressor(max_depth=2, random_state=0),
 'n_estimators': 30}
```

In [185]:
```
model.best_score_
```

Out[185]:
```
0.15916050731306358
```

In [186]:
```python
best_model = model.best_estimator_
```

In [187]:
```python
y_pred = best_model.predict(X_test)
```

In [188]:
```python
from sklearn.metrics import r2_score,mean_squared_error
import math

print(r2_score(y_test,y_pred)) #R^2
print(mean_squared_error(y_test,y_pred)) #MSE
print(math.sqrt(mean_squared_error(y_test,y_pred)))#RMSE
```

```
0.17010611548733945
0.018690839300489794
0.1367144443739936
```

In [189]:
```python
cross_val_score(AdaBoostRegressor(n_estimators=80),X,y,cv=4).mean()
```

Out[189]: 0.1622716412705755

Inference :

```
Sklearn :

Linear Regressor :           R2 = 28.33%    MSE : 1%        crossvalidation Score :
22%
KNN Regressor :              R2 = 20.17%    MSE : 1%        crossvalidation Score :
6%
RandomForest Regressor :     R2 = 23.7%     MSE : 1%        crossvalidation Score :
12%
svm SVR                 :    R2 = 27.88%    MSE : 1%        crossvalidation Score :
21%
AdaBoost                :    R2 = 17%       MSE : 1%        crossvalidation Score :
16%


PySpark :
    RandomForest Regressor : R2 = 16.37 %


Conclusion : By looking at the above results, we cannot pick any model as the Best
model
        As, we tried the Robust models like Random Forest and Adaboost, still there
is no improvement in R2 scores.
        I would say, there is a problem with the data. and we need more instances
        and meaningful attributes to predict Students Performance.
```

PySpark :

In [ ]:

```
import os
import sys

os.environ["SPARK_HOME"] = "/usr/hdp/current/spark2-client"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
# In below two lines, use /usr/bin/python2.7 if you want to use Python 2
os.environ["PYSPARK_PYTHON"] = "/usr/local/anaconda/bin/python"
os.environ["PYSPARK_DRIVER_PYTHON"] = "/usr/local/anaconda/bin/python"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.4-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")
```

Initiating Spark Session :

In [ ]:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
```

Reading the Data in Pyspark :

In [ ]:

```
df = spark.read.csv('data/StudentsPerformance.csv',inferSchema=True,header=None)
```

In [ ]:

```
df = df.toDF('Gender','Ethnicity','ParentEdu','Lunch','TestPrepCourse','MathScore'
            'ReadingScore','WritingScore')
```

In [ ]:

```
df.printSchema()
```

In [ ]:

```
root
 |-- Gender: string (nullable = true)
 |-- Ethnicity: string (nullable = true)
 |-- ParentEdu: string (nullable = true)
 |-- Lunch: string (nullable = true)
 |-- TestPrepCourse: string (nullable = true)
 |-- MathScore: integer (nullable = true)
 |-- ReadingScore: integer (nullable = true)
 |-- WritingScore: integer (nullable = true)
```

In [ ]:

```
for col in df.columns:
    print("no. of cells in column", col, "with null values:", df.filter(df[col].is
```

In [ ]:

```

no. of cells in column Gender with null values: 0
no. of cells in column Ethnicity with null values: 0
no. of cells in column ParentEdu with null values: 0
no. of cells in column Lunch with null values: 0
no. of cells in column TestPrepCourse with null values: 0
no. of cells in column MathScore with null values: 0
no. of cells in column ReadingScore with null values: 0
no. of cells in column WritingScore with null values: 0
```

In [ ]:

```
import pyspark.sql.functions as F
from pyspark.sql.types import *
df = df.withColumn("AvgScore", (F.col("MathScore") + F.col("ReadingScore") + F.col
df.show()
```

In [ ]:

```
+------+---------+------------------+------------+--------------+---------+-------
|Gender|Ethnicity|         ParentEdu|       Lunch|TestPrepCourse|MathScore|Reading
+------+---------+------------------+------------+--------------+---------+-------
|female|  group B| bachelor's degree|    standard|          none|       72|
|female|  group C|      some college|    standard|     completed|       69|
|female|  group B|   master's degree|    standard|          none|       90|
|  male|  group A|associate's degree|free/reduced|          none|       47|
|  male|  group C|      some college|    standard|          none|       76|
|female|  group B|associate's degree|    standard|          none|       71|
|female|  group B|      some college|    standard|     completed|       88|
|  male|  group B|      some college|free/reduced|          none|       40|
|  male|  group D|       high school|free/reduced|     completed|       64|
|female|  group B|       high school|free/reduced|          none|       38|
|  male|  group C|associate's degree|    standard|          none|       58|
|  male|  group D|associate's degree|    standard|          none|       40|
|female|  group B|       high school|    standard|          none|       65|
|  male|  group A|      some college|    standard|     completed|       78|
|female|  group A|   master's degree|    standard|          none|       50|
|female|  group C|  some high school|    standard|          none|       69|
|  male|  group C|       high school|    standard|          none|       88|
|female|  group B|  some high school|free/reduced|          none|       18|
|  male|  group C|   master's degree|free/reduced|     completed|       46|
|female|  group C|associate's degree|free/reduced|          none|       54|
+------+---------+------------------+------------+--------------+---------+-------
```

Label Encoding : Using StringIndexer

In [ ]:

```
#Label encoder
from pyspark.ml.feature import StringIndexer
indexed = df

for col in df.columns:
    stringIndexer = StringIndexer(inputCol=col, outputCol=col+"_encoded")
    indexed = stringIndexer.fit(indexed).transform(indexed)
indexed.show()
```

In [ ]:

```
+------+---------+------------------+------------+--------------+---------+-------
|Gender|Ethnicity|          ParentEdu|       Lunch|TestPrepCourse|MathScore|Reading
+------+---------+------------------+------------+--------------+---------+-------
|female|  group B| bachelor's degree|    standard|          none|       72|
|female|  group C|      some college|    standard|     completed|       69|
|female|  group B|   master's degree|    standard|          none|       90|
|  male|  group A|associate's degree|free/reduced|          none|       47|
|  male|  group C|      some college|    standard|          none|       76|
|female|  group B|associate's degree|    standard|          none|       71|
|female|  group B|      some college|    standard|     completed|       88|
|  male|  group B|      some college|free/reduced|          none|       40|
|  male|  group D|       high school|free/reduced|     completed|       64|
|female|  group B|       high school|free/reduced|          none|       38|
|  male|  group C|associate's degree|    standard|          none|       58|
|  male|  group D|associate's degree|    standard|          none|       40|
|female|  group B|       high school|    standard|          none|       65|
|  male|  group A|      some college|    standard|     completed|       78|
|female|  group A|   master's degree|    standard|          none|       50|
|female|  group C|  some high school|    standard|          none|       69|
|  male|  group C|       high school|    standard|          none|       88|
|female|  group B|  some high school|free/reduced|          none|       18|
|  male|  group C|   master's degree|free/reduced|     completed|       46|
|female|  group C|associate's degree|free/reduced|          none|       54|
+------+---------+------------------+------------+--------------+---------+-------
only showing top 20 rows
```

Assembling :

In [ ]:

```python
#all the independent variables need to be packed into one column of vector type
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=["Gender_encoded","Ethnicity_encoded","Paren
                                      outputCol="features")
feature_vec=assembler.transform(indexed).select('features','AvgScore')
feature_vec.show(5)
```

In [ ]:

```
+--------------------+-----------------+
|            features|         AvgScore|
+--------------------+-----------------+
| (5,[1,2],[2.0,4.0])| 72.6666666666667|
|       (5,[4],[1.0])| 82.3333333333333|
| (5,[1,2],[2.0,5.0])| 92.6666666666667|
|[1.0,4.0,1.0,1.0,...|49.33333333333336|
|       (5,[0],[1.0])| 76.3333333333333|
+--------------------+-----------------+
only showing top 5 rows
```

In [ ]:
```
#Count of target classes
feature_vec.groupBy('AvgScore').count().show()
#there is data imbalance
```

In [ ]:
```
+------------------+-----+
|          AvgScore|count|
+------------------+-----+
|              70.0|   12|
|              67.0|    9|
| 72.33333333333333|    6|
|              69.0|   12|
|50.333333333333336|    8|
|56.666666666666664|    4|
|51.333333333333336|    6|
| 99.66666666666667|    1|
| 84.66666666666667|    4|
|46.666666666666664|    4|
| 90.33333333333333|    3|
|53.666666666666664|    7|
|55.333333333333336|    5|
| 77.66666666666667|    5|
| 83.66666666666667|    4|
|62.666666666666664|    4|
| 67.66666666666667|    7|
|47.333333333333336|    2|
|59.666666666666664|    5|
|53.333333333333336|    4|
+------------------+-----+
only showing top 20 rows
```

Splitting the data to Train and Test :

In [ ]:
```
# Split the data into train and test sets
train_data, test_data = feature_vec.randomSplit([.75,.25],seed=0)
```

RandomForest Regression :

In [ ]:
```
from pyspark.ml.regression import RandomForestRegressor
model = RandomForestRegressor(labelCol='AvgScore', featuresCol="features",
                        maxDepth=15, minInfoGain=0.001, seed=0, numTrees=110)
rfModel = model.fit(train_data)

#Evaulation of the Model
predictions = rfModel.transform(test_data)

from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol='AvgScore',metricName='r2')
evaluator.evaluate(predictions)
```

In [ ]:

```
0.16373711802764568
```

Hyper Parameter Tuning :

In [ ]:

```
#Grid Search
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
model = RandomForestRegressor(labelCol='AvgScore', featuresCol="features",
                        minInfoGain=0.001, seed=0)
paramGrid = (ParamGridBuilder()   \
             .addGrid(model.maxDepth,[12,13,14,15])    \
             .addGrid(model.numTrees,[100,110,120,130])   \
             .build())

# Create 4-fold CrossValidator
cv = CrossValidator(estimator=model, estimatorParamMaps=paramGrid, evaluator=evalu

cvModel = cv.fit(train_data)
```

Choosing Best Parameters for the Model :

In [ ]:

```
#Best Model Params
score_params_list = list(zip(cvModel.avgMetrics, cvModel.getEstimatorParamMaps()))
max(score_params_list,key=lambda item:item[0])
```

In [ ]:

```

(0.16248792271672574,
 {Param(parent='RandomForestRegressor_4b5ab7672c28b22c3f80', name='maxDepth', doc=
  Param(parent='RandomForestRegressor_4b5ab7672c28b22c3f80', name='numTrees', doc=
```

In [ ]:

```
spark.stop()
```

Inference :

```
PySpark :
    RandomForest Regressor : R2 = 16.37 %        crossvalidation Score : 16.24%
            with maxdepth of 12 and no. of trees as 120.
    There is decrease in score when compared to sklearn Random Forest

Sklearn :

Linear Regressor :            R2 = 28.33%    MSE : 1%        crossvalidation Score :
22%
KNN Regressor :               R2 = 20.17%    MSE : 1%        crossvalidation Score :
6%
RandomForest Regressor :      R2 = 23.7%     MSE : 1%        crossvalidation Score :
12%
svm SVR                 :     R2 = 27.88%    MSE : 1%        crossvalidation Score :
21%
```

```
AdaBoost                :    R2 = 17%       MSE : 1%        crossvalidation Score : 
16%




Conclusion : By looking at the above results, we cannot pick any model as the Best 
model
        As, we tried the Robust models like Random Forest and Adaboost, still there 
is no improvement in R2 scores.
        I would say, there is a problem with the data. and we need more instances
        and meaningful attributes to predict Students Performance.
```

```
                                                  THANK YOU
```