In [2]:
```
#Loading Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import statistics
```

In [3]:
```
#Load data
df = pd.read_csv('C:/Users/nneam/OneDrive/Documents/540Assignments/StudentsPerformance.csv')
```

In [4]:
```
#Show Data
df.head()
```

Out[4]:

	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

Data Cleaning

In [5]:
```
#Df info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

In [6]:
```
#Count Values
df['gender'].value_counts(ascending=False)
```

Out[6]:
```
female    518
male      482
Name: gender, dtype: int64
```

In [7]:
```
#Count Values
df['race/ethnicity'].value_counts(ascending=False)
```

Out[7]:
```
group C    319
group D    262
group B    190
group E    140
group A     89
Name: race/ethnicity, dtype: int64
```

In [8]:
```
#Count Values
df['parental level of education'].value_counts(ascending=False)
```

Out[8]:
```
some college          226
associate's degree    222
high school           196
some high school      179
bachelor's degree     118
master's degree        59
Name: parental level of education, dtype: int64
```

In [9]:
```
#Count Values
df['lunch'].value_counts(ascending=False)
```

Out[9]:
```
standard        645
free/reduced    355
Name: lunch, dtype: int64
```

In [10]:
```
#Count Values
df['test preparation course'].value_counts(ascending=False)
```

Out[10]:
```
none         642
completed    358
Name: test preparation course, dtype: int64
```

In [11]:

```
#Replcing text column value for model and combining data
df['gender'] = df['gender'].replace(['female','male'],['1','0']).astype(int)
df['race/ethnicity'] = df['race/ethnicity'].replace(['group A','group B','group C','group D','group E'],['0','1','2','
df['parental level of education'] = df['parental level of education'].replace(["some college","associate's degree","hi
df['lunch'] = df['lunch'].replace(['standard','free/reduced'],['1','0']).astype(int)
df['test preparation course'] = df['test preparation course'].replace(['none','completed'],['1','0']).astype(int)
```

In [12]:

```
#View new df
df.head()
```

Out[12]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	1	1	4	1	1	72	72	74
1	1	2	0	1	0	69	90	88
2	1	1	5	1	1	90	95	93
3	0	0	1	0	1	47	57	44
4	0	2	0	1	1	76	78	75

In [13]:

```
#View new df type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   int32
 1   race/ethnicity               1000 non-null   int32
 2   parental level of education  1000 non-null   int32
 3   lunch                        1000 non-null   int32
 4   test preparation course      1000 non-null   int32
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int32(5), int64(3)
memory usage: 43.1 KB
```

Data Exploration

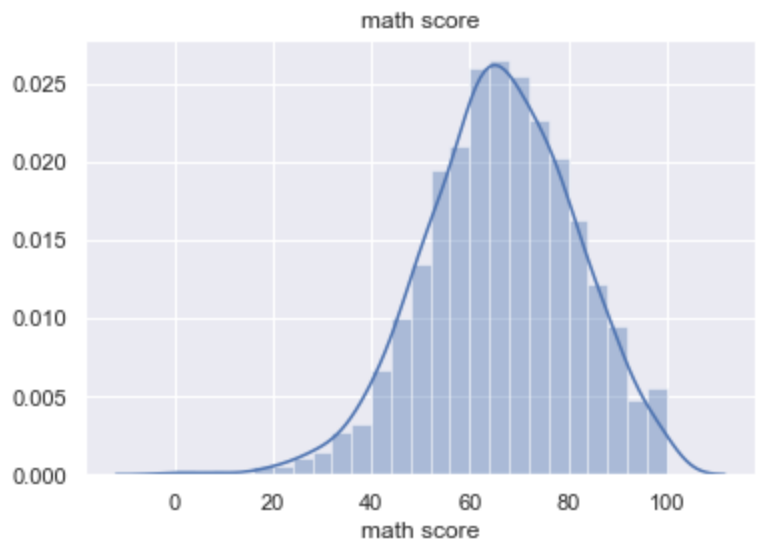
In [14]:
```
#Checking variance
df.var()
```

Out[14]:
```
gender                          0.249926
race/ethnicity                  1.339063
parental level of education     2.303580
lunch                           0.229204
test preparation course         0.230066
math score                    229.918998
reading score                 213.165605
writing score                 230.907992
dtype: float64
```

In [15]:
```
sns.set(font_scale = 1)
```

In [16]:
```
# Desnity and Histogram for math score
sns.distplot(a=df['math score']).set_title('math score')
```

Out[16]:
```
Text(0.5, 1.0, 'math score')
```



In [17]:

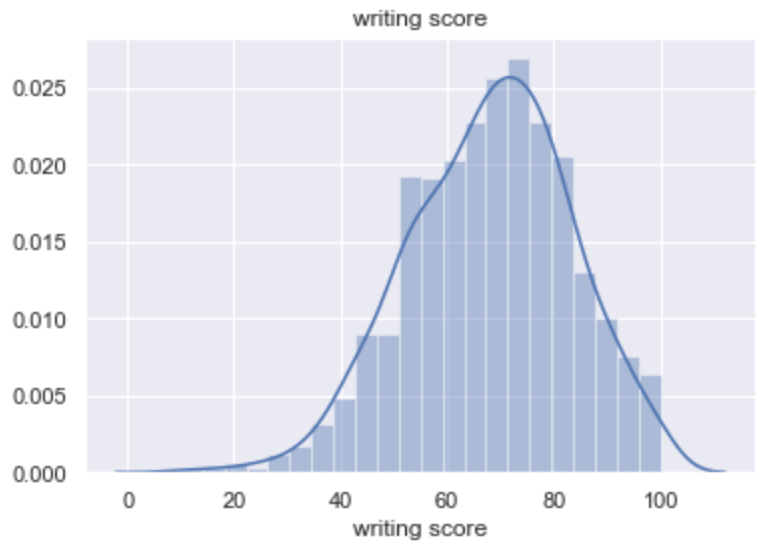
```
# Desnity and Histogram for reading score
sns.distplot(a=df['reading score']).set_title('reading score')
```

Out[17]: Text(0.5, 1.0, 'reading score')

In [18]:

```
# Desnity and Histogram for writing score
sns.distplot(a=df['writing score']).set_title('writing score')
```

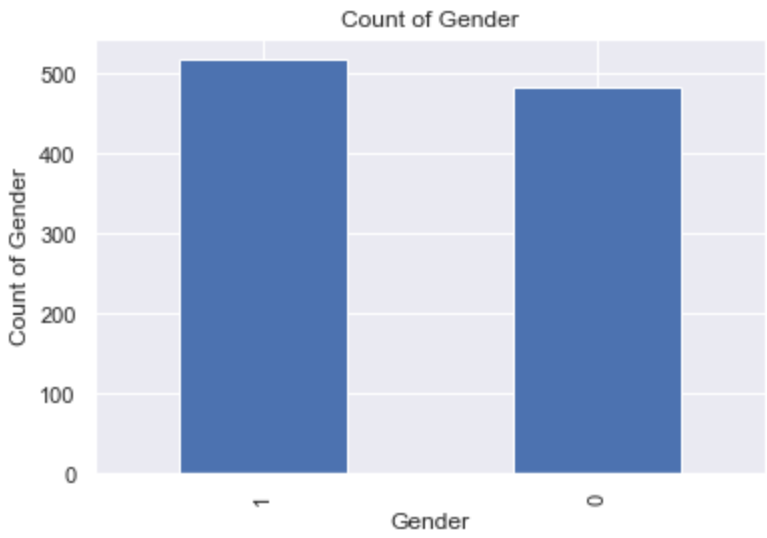
Out[18]: Text(0.5, 1.0, 'writing score')



In [19]:

```
#Count of Gender 1 = Female, 0 = Male
ax = df['gender'].value_counts().plot(kind = 'bar', title = 'Count of Gender')
ax.set_xlabel("Gender")
ax.set_ylabel("Count of Gender")
```

Out[19]: Text(0, 0.5, 'Count of Gender')



In [20]:

```
#Count of Lunch 1 = Standard, 0 = Free/Reduced
ax = df['lunch'].value_counts().plot(kind = 'bar', title = 'Count of Lunch')
ax.set_xlabel("Lunch")
ax.set_ylabel("Count of Lunch")
```

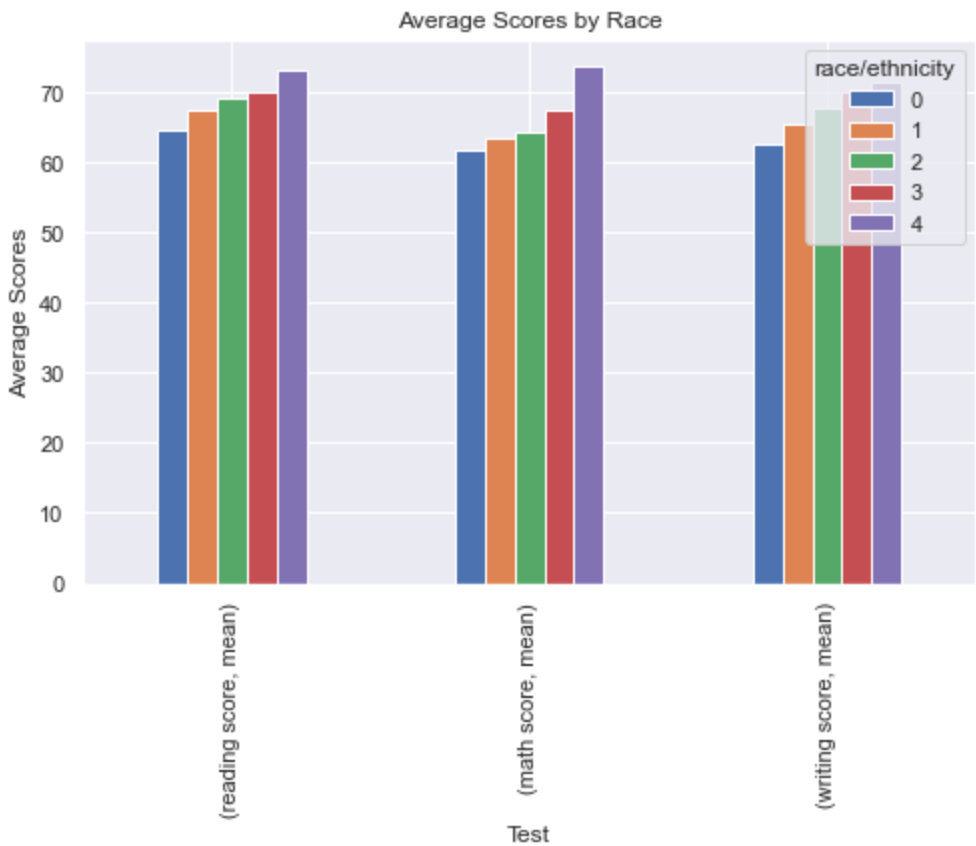
Out[20]:

```
Text(0, 0.5, 'Count of Lunch')
```

In [21]:

```
#Viewing Averages by Race group A = 0,group B = 1,group C = 2,group D = 3,group E = 4
ax = df.groupby(['race/ethnicity'])[['reading score','math score','writing score']].agg(['mean']).transpose().plot(kin
ax.set_title("Average Scores by Race")
ax.set_xlabel("Test")
ax.set_ylabel("Average Scores")
```

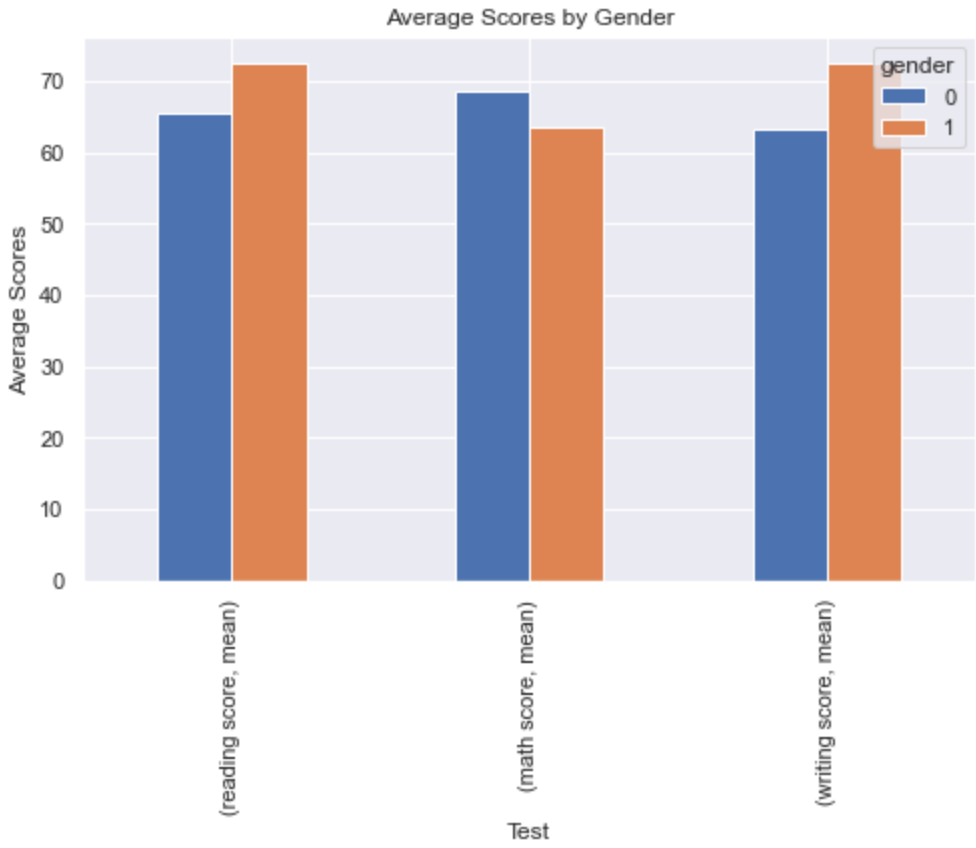
Out[21]: Text(0, 0.5, 'Average Scores')



In [22]:

```
#Viewing Averages by Gender Female = 1, Male = 0
ax = df.groupby(['gender'])[['reading score','math score','writing score']].agg(['mean']).transpose().plot(kind='bar',
ax.set_title("Average Scores by Gender")
ax.set_xlabel("Test")
ax.set_ylabel("Average Scores")
```

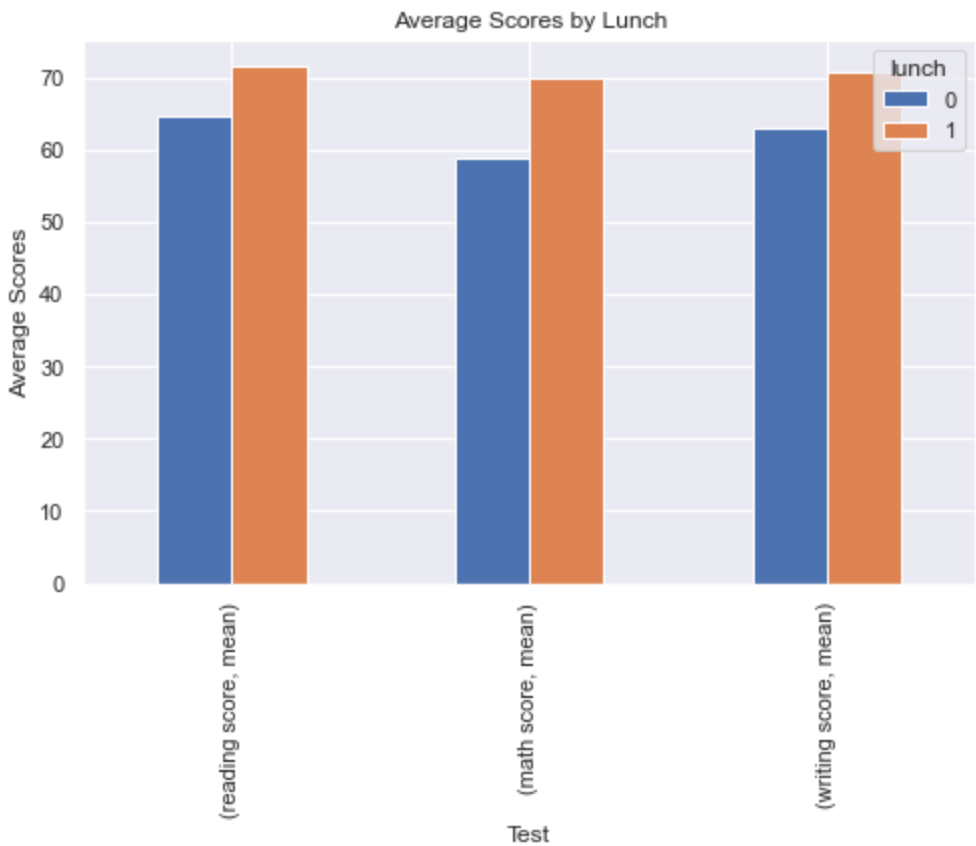
Out[22]: Text(0, 0.5, 'Average Scores')



In [23]:

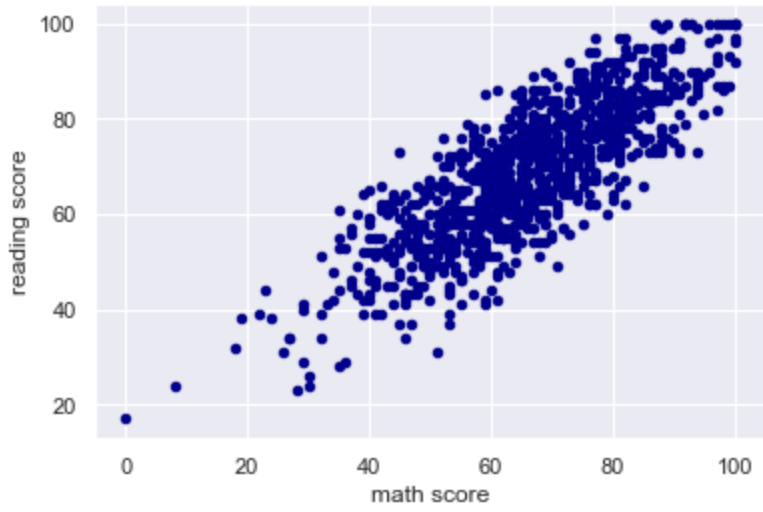
```
#Viewing Averages by Lunch 1 = Standard, 0 = Free/Reduced
ax = df.groupby(['lunch'])[['reading score','math score','writing score']].agg(['mean']).transpose().plot(kind='bar',f
ax.set_title("Average Scores by Lunch")
ax.set_xlabel("Test")
ax.set_ylabel("Average Scores")
```

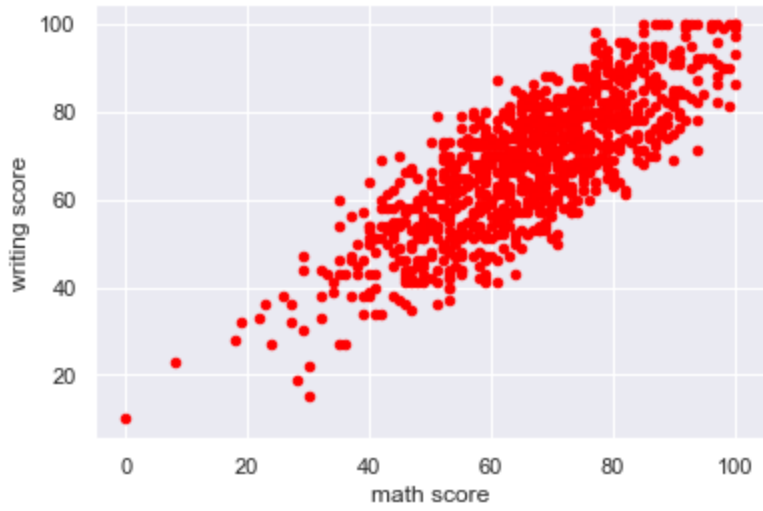
Out[23]: Text(0, 0.5, 'Average Scores')



In [24]:

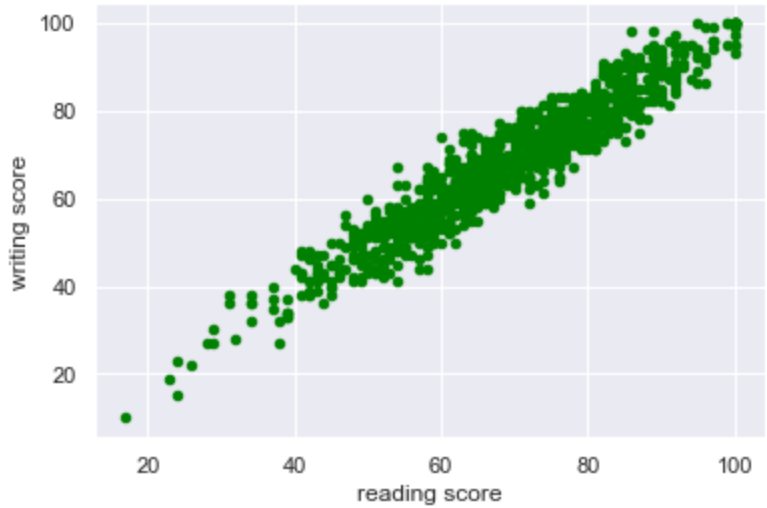
```
#Comparing Correlation Between Math Scores
ax1 = df.plot.scatter(x='math score',y='reading score', c='DarkBlue')
ax1 = df.plot.scatter(x='math score', y='writing score', c='Red')
plt.show()
```





In [25]:

```
#Comparing Correlation Between Reading Scores
ax1 = df.plot.scatter(x='reading score',y='writing score', c='Green')
plt.show()
```



In [145]:

```
#Assign pearson correlation
pearson = df.corr(method = 'pearson')
pearson
```

Out[145]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
gender	1.000000	0.001502	0.020417	-0.021372	0.006028	-0.167982	0.244313	0.301225
race/ethnicity	0.001502	1.000000	-0.029484	0.046563	-0.017508	0.216415	0.145253	0.165691
parental level of education	0.020417	-0.029484	1.000000	-0.020815	-0.030740	0.004102	0.048825	0.055357
lunch	-0.021372	0.046563	-0.020815	1.000000	0.017044	0.350877	0.229560	0.245769
test preparation course	0.006028	-0.017508	-0.030740	0.017044	1.000000	-0.177702	-0.241780	-0.312946
math score	-0.167982	0.216415	0.004102	0.350877	-0.177702	1.000000	0.817580	0.802642
reading score	0.244313	0.145253	0.048825	0.229560	-0.241780	0.817580	1.000000	0.954598
writing score	0.301225	0.165691	0.055357	0.245769	-0.312946	0.802642	0.954598	1.000000

In [146]:

```
#Initial Pearson Correlation Heatmap
fig, ax = plt.subplots(figsize=(25,25))
sns.heatmap(pearson,
            xticklabels=pearson.columns,
            yticklabels=pearson.columns,
            cmap='YlGnBu',
            annot=True,
            linewidth= 0.5,
            square = False,
            annot_kws={"size": 20},
            ax=ax)
```

Out[146]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x266136ddd00>
```

Model Creation

In [148]:

```
#Split Data
X = df.drop('math score', axis=1)
y = df['math score']
```

In [149]:

```
#Creating the Test, Train, and Split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=40)
```

In [150]:
```
#Fitting Model
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
regressor.fit(X_train, y_train)
```

Out[150]:
```
RandomForestRegressor(random_state=0)
```

In [151]:
```
#Predicting results
y_pred = regressor.predict(X_test)
```

In [157]:
```
#Analyze Results
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 5.052008333333334
Mean Squared Error: 39.177021180555556
Root Mean Squared Error: 6.259154989338062
```

KNN

In [178]:
```
#Fit model
from sklearn.model_selection import train_test_split
train , test = train_test_split(df, test_size = 0.3)
x_train = train.drop('math score', axis=1)
Y_train = train['math score']
x_test = test.drop('math score', axis = 1)
Y_test = test['math score']
```

In [179]:
```
#Scale model
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
```

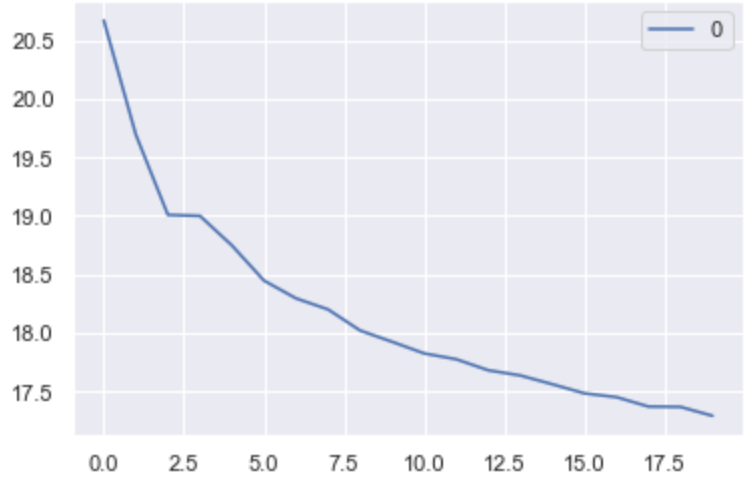
In [181]:

```
#K Analysis
rmse_val = []
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)
    model.fit(x_train, Y_train)
    pred=model.predict(x_test)
    error = sqrt(mean_squared_error(y_test,pred))
    rmse_val.append(error)
    print('RMSE value for k= ' , K , 'is:', error)
```

```
RMSE value for k=  1 is: 20.668252627318708
RMSE value for k=  2 is: 19.693674957542417
RMSE value for k=  3 is: 19.009354422377303
RMSE value for k=  4 is: 19.000745599405654
RMSE value for k=  5 is: 18.74812346165166
RMSE value for k=  6 is: 18.448446687599073
RMSE value for k=  7 is: 18.296832557375964
RMSE value for k=  8 is: 18.20257536723856
RMSE value for k=  9 is: 18.022527055923135
RMSE value for k=  10 is: 17.926291678239906
RMSE value for k=  11 is: 17.827182327896523
RMSE value for k=  12 is: 17.777482636438968
RMSE value for k=  13 is: 17.683338863822666
RMSE value for k=  14 is: 17.63910168243455
RMSE value for k=  15 is: 17.564216147864066
RMSE value for k=  16 is: 17.486358522431136
RMSE value for k=  17 is: 17.45414787292727
RMSE value for k=  18 is: 17.37364051584073
RMSE value for k=  19 is: 17.37075768439619
RMSE value for k=  20 is: 17.294606818311888
```

In [182]:
```
#plotting the rmse values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

Out[182]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26618b5b850>
```



In [195]:
```
knn_model = KNeighborsRegressor(n_neighbors=1)
```

In [196]:
```
knn_model.fit(x_train, Y_train)
```

Out[196]:
```
KNeighborsRegressor(n_neighbors=1)
```

In [198]:

```
#Analyzing results
train_preds = knn_model.predict(x_train)
mse = mean_squared_error(Y_train, train_preds)
rmse = sqrt(mse)
rmse
```

Out[198]:

```
0.4225771273642583
```