In [120]:
```
1 pip install certifi==2020.11.8
```

```
Requirement already satisfied: certifi==2020.11.8 in c:\users\16146\anacond
a3\lib\site-packages (2020.11.8)
Note: you may need to restart the kernel to use updated packages.
```

In [122]:
```
1 pip install contextlib2==0.6.0.post1
```

```
Requirement already satisfied: contextlib2==0.6.0.post1 in c:\users\16146\a
naconda3\lib\site-packages (0.6.0.post1)
Note: you may need to restart the kernel to use updated packages.
```

In [123]:
```
1 pip install Cython==0.29.20
```

```
Requirement already satisfied: Cython==0.29.20 in c:\users\16146\anaconda3
\lib\site-packages (0.29.20)
Note: you may need to restart the kernel to use updated packages.
```

In [124]:
```
1 pip install joblib==0.17.0
```

```
Requirement already satisfied: joblib==0.17.0 in c:\users\16146\anaconda3\l
ib\site-packages (0.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [127]:
```
1 pip install lxml==4.5.1
```

```
Requirement already satisfied: lxml==4.5.1 in c:\users\16146\anaconda3\lib
\site-packages (4.5.1)
Note: you may need to restart the kernel to use updated packages.
```

In [128]:
```
1 pip install numpy==1.19.4
```

```
Requirement already satisfied: numpy==1.19.4 in c:\users\16146\anaconda3\li
b\site-packages (1.19.4)
Note: you may need to restart the kernel to use updated packages.
```

In [129]:
```
pip install pandas==1.1.4
```

```
Requirement already satisfied: pandas==1.1.4 in c:\users\16146\anaconda3\li
b\site-packages (1.1.4)
Requirement already satisfied: numpy>=1.15.4 in c:\users\16146\anaconda3\li
b\site-packages (from pandas==1.1.4) (1.19.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\16146\ana
conda3\lib\site-packages (from pandas==1.1.4) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\16146\anaconda3\lib
\site-packages (from pandas==1.1.4) (2020.4)
Requirement already satisfied: six>=1.5 in c:\users\16146\appdata\roaming\p
ython\python38\site-packages (from python-dateutil>=2.7.3->pandas==1.1.4)
(1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
pip install Pillow==7.1.2
```

```
Requirement already satisfied: Pillow==7.1.2 in c:\users\16146\anaconda3\li
b\site-packages (7.1.2)
Note: you may need to restart the kernel to use updated packages.
```

In [130]:
```
pip install python-dateutil==2.8.1
```

```
Requirement already satisfied: python-dateutil==2.8.1 in c:\users\16146\ana
conda3\lib\site-packages (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\16146\appdata\roaming\p
ython\python38\site-packages (from python-dateutil==2.8.1) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [134]:
```
pip install pytz==2020.4
```

```
Requirement already satisfied: pytz==2020.4 in c:\users\16146\anaconda3\lib
\site-packages (2020.4)
Note: you may need to restart the kernel to use updated packages.
```

In [135]:
```
1 pip install scikit-learn==0.23.2
```

```
Requirement already satisfied: scikit-learn==0.23.2 in c:\users\16146\anaco
nda3\lib\site-packages (0.23.2)
Requirement already satisfied: numpy>=1.13.3 in c:\users\16146\anaconda3\li
b\site-packages (from scikit-learn==0.23.2) (1.19.4)
Requirement already satisfied: joblib>=0.11 in c:\users\16146\anaconda3\lib
\site-packages (from scikit-learn==0.23.2) (0.17.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\16146\anaconda3\li
b\site-packages (from scikit-learn==0.23.2) (1.5.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\16146\anaco
nda3\lib\site-packages (from scikit-learn==0.23.2) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [136]:
```
1 pip install scipy==1.5.4
```

```
Requirement already satisfied: scipy==1.5.4 in c:\users\16146\anaconda3\lib
\site-packages (1.5.4)
Requirement already satisfied: numpy>=1.14.5 in c:\users\16146\anaconda3\li
b\site-packages (from scipy==1.5.4) (1.19.4)
Note: you may need to restart the kernel to use updated packages.
```

In [17]:
```
1 pip install six==1.15.0
```

```
Requirement already satisfied: six==1.15.0 in c:\users\16146\appdata\roamin
g\python\python38\site-packages (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [18]:
```
1 pip install tf-slim==1.1.0
```

```
Requirement already satisfied: tf-slim==1.1.0 in c:\users\16146\anaconda3\l
ib\site-packages (1.1.0)
Requirement already satisfied: absl-py>=0.2.2 in c:\users\16146\anaconda3\l
ib\site-packages (from tf-slim==1.1.0) (0.11.0)
Requirement already satisfied: six in c:\users\16146\appdata\roaming\python
\python38\site-packages (from absl-py>=0.2.2->tf-slim==1.1.0) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```
1 pip install threadpoolctl==2.1.0
```

```
Requirement already satisfied: threadpoolctl==2.1.0 in c:\users\16146\anaco
nda3\lib\site-packages (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```
pip install wincertstore==0.2
```

```
Requirement already satisfied: wincertstore==0.2 in c:\users\16146\anaconda
3\lib\site-packages (0.2)Note: you may need to restart the kernel to use up
dated packages.
```

In [143]:
```
#import the libraries
import pandas as pd
import numpy as np
```

In [144]:
```
#read our Excel file using pandas
df = pd.read_excel(r'C:\Users\16146\Desktop\File_running.xlsx')
df['Level '] = df['Level '].str.lower()
df['Level '] = df['Level '].str.replace('-', ' ')
df.head(5)

```

...

In [145]:
```
#lets het a pivot table of first time second time third time and the fou
df1 = df.pivot_table('Scores', ['SI'], 'Level ').reset_index()
df1 = df1.fillna(0)
df1
```

...

In [195]:
```
#create a new column called final_score to store the final score of all 
df1['grad_GPA'] = df1.mean(axis=1)
df1.head(10)
```

...

In [147]:
```
#we predict the results only using first two years so lets remove the th
final_dataset = df1.drop(columns=['third term', 'fourth term'])
final_dataset
```

...

In [199]:

```
#lest create a linear regression machine learning model to get the output
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
#extact the feature values
features = ['first term']

X = final_dataset.loc[:, features].values
#extract the label values
y = final_dataset['grad_GPA']

#define train and test dataset of X and y respectively
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

print(y_pred)


```

. . .

In [149]:

```
new_feature1 = df1[['third term']]
new_feature2 = df1[['fourth term']]
new_pred1 = regr.predict(new_feature1)
new_pred2 = regr.predict(new_feature2)
print(new_pred1)
print(new_pred2)

#plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.plot(new_pred1, '--')
plt.xlabel('number of students')
plt.ylabel('Third Term GPA')
plt.title('third term Results')


plt.figure(figsize=(10, 10))
plt.plot(new_pred2, '--')
plt.xlabel('number of students')
plt.ylabel('Fourth Term GPA')
plt.title('fourth term Results')


plt.show()
```

...

In [150]:

```
#lest create a linear regression machine learning model to get the output
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
#extact the feature values
features = ['second term']

X = final_dataset.loc[:, features].values
#extract the label values
y = final_dataset['grad_GPA']

#define train and test dataset of X and y respectively
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
regr1 = linear_model.LinearRegression()

# Train the model using the training sets
regr1.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr1.predict(X_test)

# The coefficients
print('Coefficients: \n', regr1.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

print(y_pred)
```

. . .

In [151]:

```
new_feature3 = df1[['third term']]
new_feature4 = df1[['fourth term']]
new_pred3 = regr1.predict(new_feature3)
new_pred4 = regr1.predict(new_feature4)
print(new_pred3)
print(new_pred3)

#plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.plot(new_pred3)
plt.xlabel('number of students')
plt.ylabel('Third Term GPA')
plt.title('predicted third term GPA')


plt.figure(figsize=(10, 10))
plt.plot(new_pred4)
plt.xlabel('number of students')
plt.ylabel('Fourth Term GPA')
plt.title('predicted fourth term GPA')


plt.show()

```

...

In [ ]:

```

```

In [152]:

```
#lest create a linear regression machine learning model to get the output
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
#extact the feature values
features = ['first term']

X = final_dataset.loc[:, features].values
#extract the label values
y = final_dataset['grad_GPA']

#define train and test dataset of X and y respectively
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
regr2 = linear_model.LinearRegression()

# Train the model using the training sets
regr2.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr2.predict(X_test)

# The coefficients
print('Coefficients: \n', regr2.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

print(y_pred)
```

...

In [153]:

```
new_feature3 = df1[['third term']]
new_feature4 = df1[['fourth term']]
new_pred3 = regr1.predict(new_feature3)
new_pred4 = regr1.predict(new_feature4)
print(new_pred4)

#plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.plot(new_pred3)
plt.xlabel('number of students')
plt.ylabel('some numbers')
plt.title('predicted final score using third time results')


plt.figure(figsize=(10, 10))
plt.plot(new_pred4)
plt.xlabel('number of students')
plt.ylabel('some numbers')
plt.title('predicted final score using fourth time results')


plt.show()
```

...

In [192]:

```
#lest create a support vector regression machine learning model to get th
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
#split feature set from the data frame
features = ['second term']
#set above columns as the feature
X = final_dataset.loc[:, features].values
#set final_score column as the laber
y = final_dataset['grad_GPA']

#add train_test_split to split train and test data
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
svr = SVR(kernel = 'rbf')

# Train the model using the training sets
svr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = svr.predict(X_test)



print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

# Plot output
```

```
Mean squared error: 0.03
Coefficient of determination: -0.58
```

In [193]:

```
new_feature5 = df1[['third term']]
new_feature6 = df1[['fourth term']]
new_pred5 = svr.predict(new_feature5)
new_pred6 = svr.predict(new_feature6)
print(new_pred5)
print(new_pred6)

#plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.plot(new_pred5)
plt.xlabel('number of students')
plt.ylabel('GPA')
plt.title('predicted third term results')


plt.figure(figsize=(10, 10))
plt.plot(new_pred6)
plt.xlabel('number of students')
plt.ylabel('GPA')
plt.title('predicted fourth term results')


plt.show()
```

. . .

In [ ]:

```
1
```

In [156]:

```
# lets find the cross validation
from sklearn.model_selection import KFold
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error as mse
from math import sqrt

## Define variables for the for loop

kf = KFold(n_splits=10)
RMSE_sum=0
RMSE_length=10
X = final_dataset.loc[:, features].values
y = final_dataset['grad_GPA']

for loop_number, (train, test) in enumerate(kf.split(X)):

    ## Get Training Matrix and Vector

    training_X_array = X[train]
    training_y_array = y[train]

    ## Get Testing Matrix Values

    X_test_array = X[test]
    y_actual_values = y[test]

    ## Fit the Linear Regression Model

    svr_model = SVR().fit(training_X_array, training_y_array)
#svr = SVR(kernel = 'rbf')
    ## Compute the predictions for the test data

    prediction = svr_model.predict(X_test_array)
    prediction = np.array(prediction)

    ## Calculate the RMSE

    RMSE_cross_fold = sqrt(mse(prediction, y_actual_values))

    ## Add each RMSE_cross_fold value to the sum

    RMSE_sum=RMSE_cross_fold+RMSE_sum

## Calculate the average and print

RMSE_cross_fold_avg=RMSE_sum/RMSE_length

print('The Mean RMSE across all folds is',RMSE_cross_fold_avg)

```

```
The Mean RMSE across all folds is 0.19764159165326972
```

In [ ]:

```
1
```

In [208]:

```
#lets label the final_score before feed to the classifires
label = []

for num in final_dataset['grad_GPA']:
  if num < 2.00:
    label.append("Minimally accepted")
  elif num >= 2.00 and num < 3.00:
    label.append("Satisfactory")
  else:
    label.append("Excellent")


```

In [209]:

```
#add that array to a dataframe of final_dataset
final_dataset['label'] = pd.DataFrame(label)

```

In [210]:

```
final_dataset

```

...

In [223]:

```
# import k nearest neighbor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
#split feature set from the data frame
features = ['second term']
#split the train and test data
X = final_dataset.loc[:, features].values
y = final_dataset['label']


X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
clf = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
clf.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = clf.predict(X_test)



%time
from sklearn.metrics import classification_report
y_pred = clf.predict(X_test)

print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test, y_pred))

print(final_dataset)
```

...

In [224]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np

features = ['second term']

X = final_dataset.loc[:, features].values
y = final_dataset['label']


X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
nbc = GaussianNB()

# Train the model using the training sets
nbc.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = nbc.predict(X_test)

%time
from sklearn.metrics import classification_report
y_pred = nbc.predict(X_test)
print(y_pred)

print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test, y_pred))

print(final_dataset)

```

...

In [225]:
```
new_feature9 = df1[['third term']]
new_feature10 = df1[['fourth term']]
new_pred9 = nbc.predict(new_feature9)
new_pred10 = nbc.predict(new_feature10)
print(new_pred9)
print(new_pred10)


#plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.plot(new_pred9)
plt.xlabel('number of students')
plt.ylabel('Student Performance')
plt.title('predicted third term results')


plt.figure(figsize=(10, 10))
plt.plot(new_pred10)
plt.xlabel('no. of students')
plt.ylabel('Student Performance')
plt.title('predicted fourth time results')


plt.show()
```

...

In [ ]:
```

```

In [ ]:
```

```

In [214]:
```
#lets create another pivot table with SI and Code
df2 = df.pivot_table('Scores', ['SI','CPrograms'], 'Level ').reset_index
#fill any null values with the 0
df2 = df2.fillna(0)
df2
```

...

In [ ]:
```

```

In [216]:

```
#now create a new column called final_socre_sub to store the mean values
df2['final_score_sub'] = df2[['first term', 'second term', 'third term',
df2
```

. . .

In [220]:

```
#lets merge df2 with df1 SI and final_score
dataset_final = df2.merge(df1[[ 'SI','grad_GPA']], on='SI', how='left')

```

In [ ]:

```
# Lets find the cross validation
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse
from math import sqrt

## Define variables for the for loop

kf = KFold(n_splits=10)
RMSE_sum=0
RMSE_length=10
features = ['first time', 'second time', 'final_score_sub', 'final_score

X = dataset_final.loc[:, features].values
y = dataset_final['Code']

for loop_number, (train, test) in enumerate(kf.split(X)):

    ## Get Training Matrix and Vector

    training_X_array = X[train]
    training_y_array = y[train]

    ## Get Testing Matrix Values

    X_test_array = X[test]
    y_actual_values = y[test]

    ## Fit the Linear Regression Model

    lr_model = LinearRegression().fit(training_X_array, training_y_array
#svr = SVR(kernel = 'rbf')
    ## Compute the predictions for the test data

    prediction = lr_model.predict(X_test_array)
    prediction = np.array(prediction)

    ## Calculate the RMSE

    RMSE_cross_fold = sqrt(mse(prediction, y_actual_values))

    ## Add each RMSE_cross_fold value to the sum

    RMSE_sum=RMSE_cross_fold+RMSE_sum

## Calculate the average and print

RMSE_cross_fold_avg=RMSE_sum/RMSE_length

print('The Mean RMSE across all folds is',RMSE_cross_fold_avg)
```

```
The Mean RMSE across all folds is 1331.6325263356082
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
x =
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

In [ ]:
```
#lest create a support vector regression machine learning model to get th
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
#split feature set from the data frame
features = ['first time', 'second time', 'final_score_sub', 'final_score
#set above columns as the feature
X = dataset_final.loc[:, features].values
#set final_score column as the laber
y = dataset_final['Code']


#add train_test_split to split train and test data
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2

# Create linear regression object
svr = SVR(kernel = 'rbf')

# Train the model using the training sets
svr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)



print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

# Plot output
```

```
Mean squared error: 1741116.55
Coefficient of determination: 0.43
```

In [ ]:

```
new_input = [[1.12309797, 1.41131072,0.652500, 2.044]]

new_pred = svr.predict(new_input)
print(new_pred)
```

In [ ]:

```
#lets merge the dataset_final with df to get the Sub-Programs names. it w
dataset_final_to_plot = dataset_final.merge(df[[ 'Code','Sub-Programs']],
dataset_final_to_plot
```

In [ ]:

```

x = dataset_final_to_plot['Sub-Programs']
y = dataset_final_to_plot['final_score']
#colors = np.random.rand(N)
#area = (30 * np.random.rand(N))**2  # 0 to 15 point radii
plt.figure(figsize=(20,10))
plt.scatter(x, y,  alpha=0.5)
plt.xticks(rotation = 90)
plt.show()
```

. . .