

Student Performance Analysis and Visualization

```
In [1]: pip install dabl

Requirement already satisfied: dabl in c:\users\vaishnavi\anaconda3\lib\site-packages (0.1.8)
Requirement already satisfied: numpy in c:\users\vaishnavi\anaconda3\lib\site-packages (from dabl) (1.18.1)
Requirement already satisfied: matplotlib in c:\users\vaishnavi\anaconda3\lib\site-packages (from dabl) (3.1.1)
Requirement already satisfied: pandas in c:\users\vaishnavi\anaconda3\lib\site-packages (from dabl) (1.0.1)
Requirement already satisfied: scikit-learn in c:\users\vaishnavi\anaconda3\lib\site-packages (from dabl) (0.22.1)
Requirement already satisfied: seaborn in c:\users\vaishnavi\anaconda3\lib\site-packages (from dabl) (0.10.1)
Requirement already satisfied: cyclo=>0.10 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib==3.1.1) (0.10.0)
Requirement already satisfied: pyarsenal=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib==3.1.1) (2.0.4)
Requirement already satisfied: python-dateutil=>2.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib==3.1.1) (2.8.1)
Requirement already satisfied: kiwisolver=>1.0.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib==3.1.1) (0.1.0)
Requirement already satisfied: joblib=>0.11 in c:\users\vaishnavi\anaconda3\lib\site-packages (from pandas=>dabl) (2019.3)
Requirement already satisfied: six in c:\users\vaishnavi\anaconda3\lib\site-packages (from cyclo=>0.10) (1.14.0)
Requirement already satisfied: setuptools in c:\users\vaishnavi\anaconda3\lib\site-packages (from kiwisolver=>1.0.1->matplotlib==3.1.1) (45.2.0.post20200210)
Note: you may need to restart the kernel to use updated packages.
```

```
In [2]: # for some basic operations

import numpy as np
import pandas as pd

# for visualizations
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import dabl
import os
print(os.listdir('../documents/input/'))

['StudentsPerformance.csv']
```

Reading the Data set

```
In [3]: # reading the data

data = pd.read_csv('../documents/input/StudentsPerformance.csv')

# getting the shape of the data
print(data.shape)

(1000, 8)
```

```
In [4]: # looking at the head of the data

data.head()
```

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 B	some college	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

Descriptive Statistics

```
In [5]: # describing the dataset

data.describe()
```

	math score	reading score	writing score
count	1000.00000	1000.00000	1000.00000
mean	68.08000	69.16000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.00000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	78.000000
max	100.00000	100.00000	100.00000

```
In [6]: # check the no. of unique items present in the categorical column

data.select_dtypes('object').nunique()
```

gender	2
race/ethnicity	5
parental level of education	6
lunch	2
test preparation course	2
dtype: int64	

```
In [7]: # check the percentage of missing data in each column present in the data

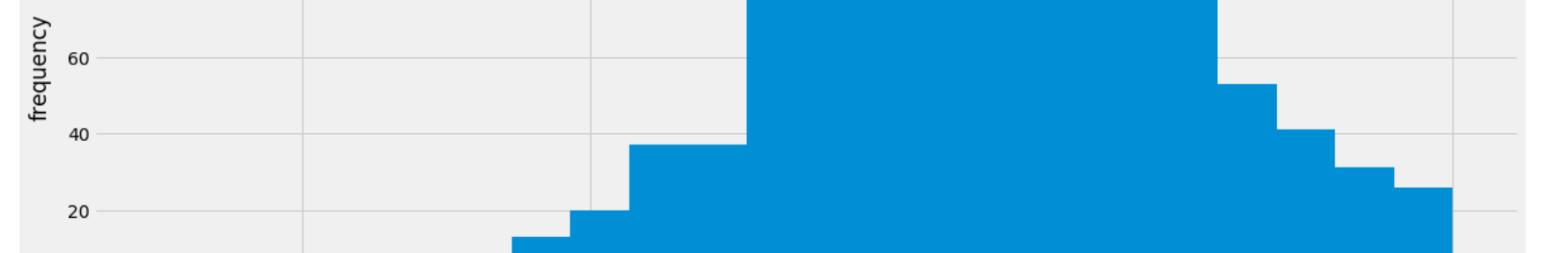
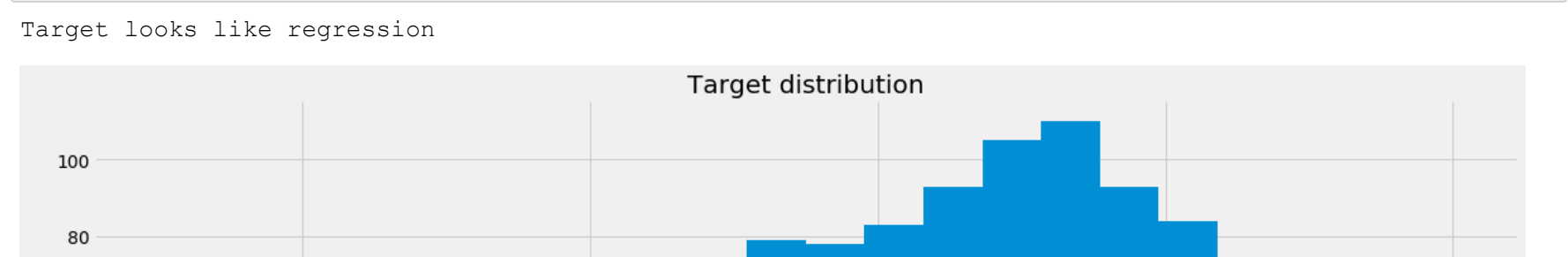
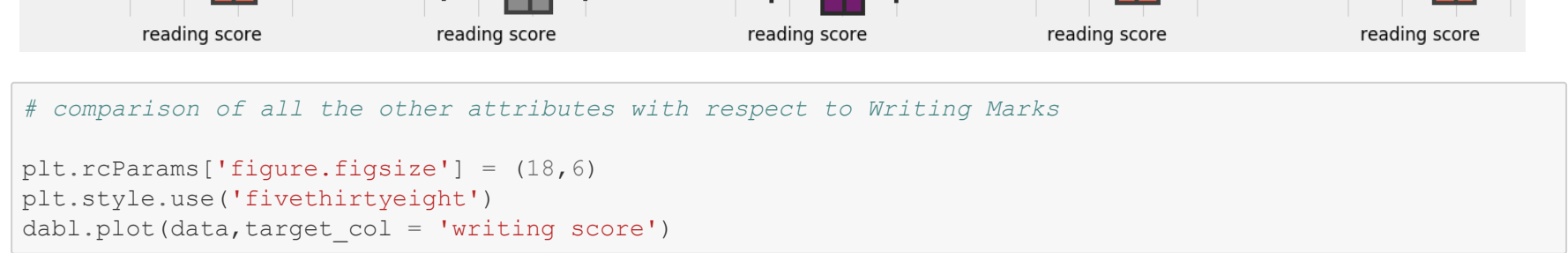
no_of_columns = data.shape[0]
percentage_of_missing_data = data.isnull().sum()/no_of_columns
print(percentage_of_missing_data)
```

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	

```
In [8]: # comparison of all the attributes with respect to Math Marks

plt.rcParams['figure.figsize'] = (18,6)
plt.style.use('fivethirtyeight')
dabl.plot(data,target_col = 'math score')
```

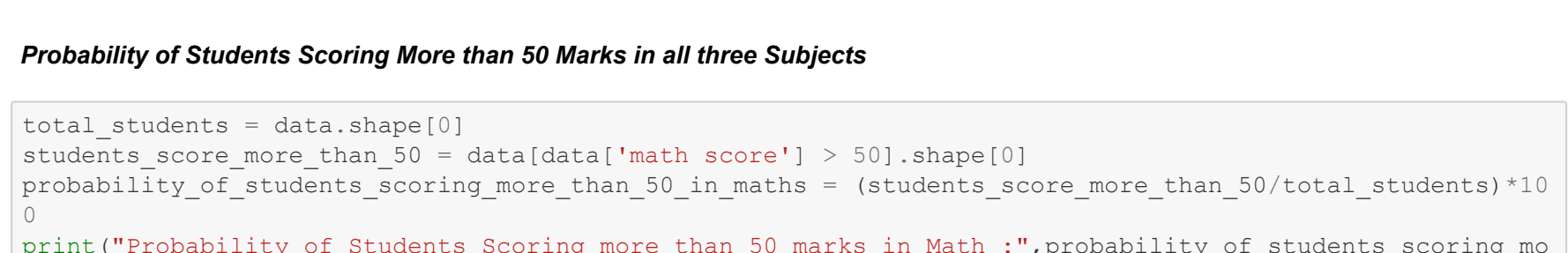
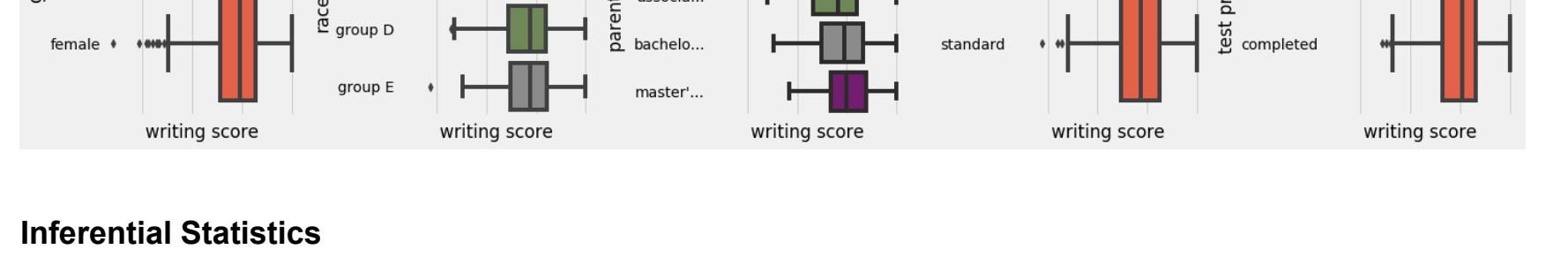
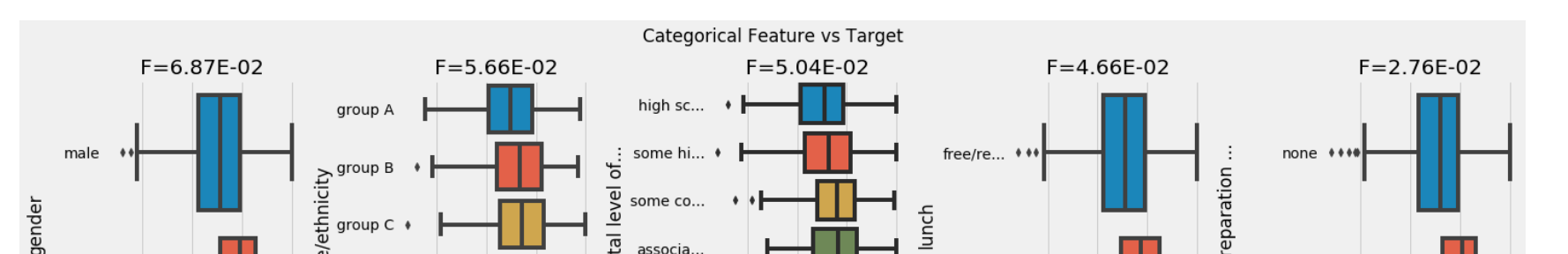
Target looks like regression



```
In [9]: # comparison of all the other attributes with respect to Reading Marks

plt.rcParams['figure.figsize'] = (18,6)
plt.style.use('fivethirtyeight')
dabl.plot(data,target_col = 'reading score')
```

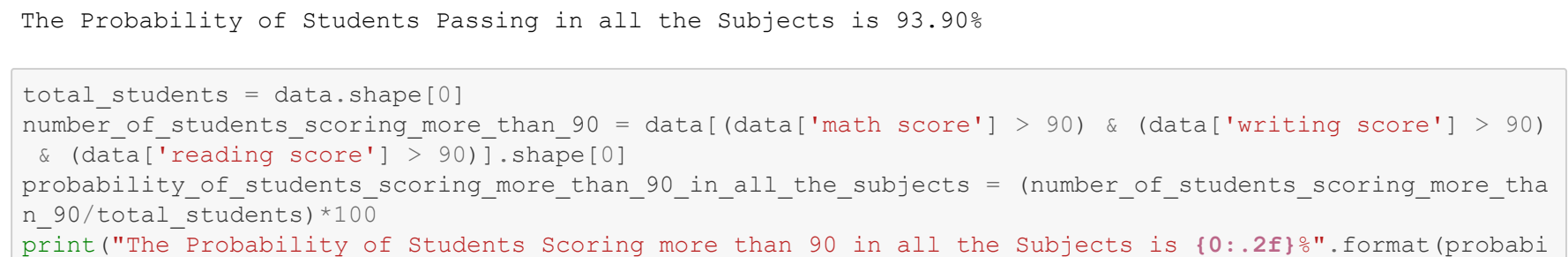
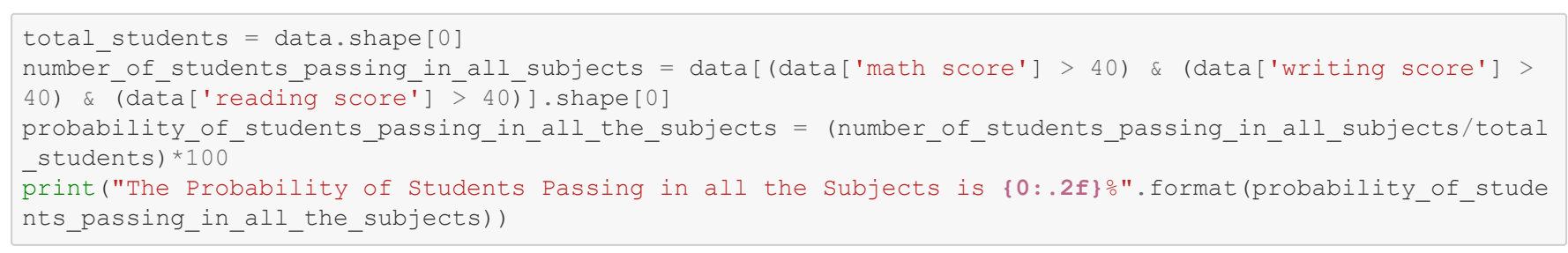
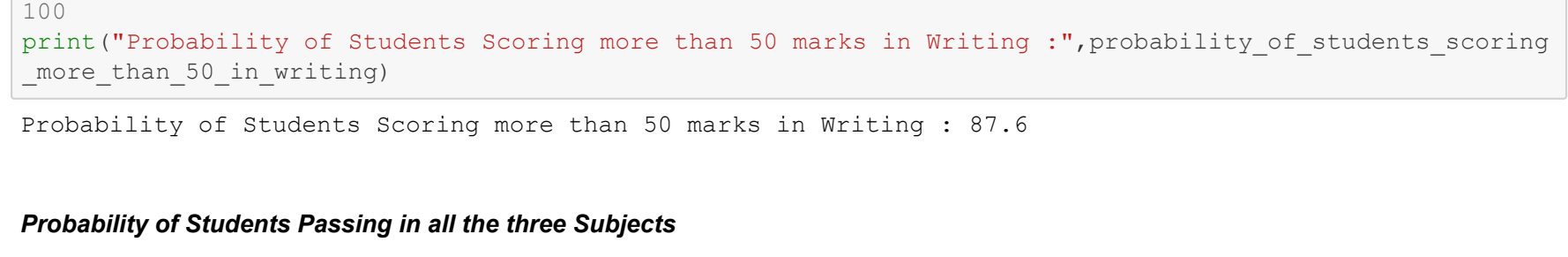
Target looks like regression



```
In [10]: # comparison of all the other attributes with respect to Writing Marks

plt.rcParams['figure.figsize'] = (18,6)
plt.style.use('fivethirtyeight')
dabl.plot(data,target_col = 'writing score')
```

Target looks like regression



The Probability of Students Scoring more than 50 marks in Math : 85.0

```
In [11]: total_students = data.shape[0]
students_score_more_than_50 = data[data['math score'] > 50].shape[0]
probability_of_students_scoring_more_than_50_in_maths = (students_score_more_than_50/total_students)*100
print("Probability of Students Scoring more than 50 marks in Math :",probability_of_students_scoring_more_than_50_in_maths)
```

The Probability of Students Scoring more than 50 marks in Reading : 90.3

```
In [12]: total_students = data.shape[0]
students_score_more_than_50 = data[data['reading score'] > 50].shape[0]
probability_of_students_scoring_more_than_50_in_reading = (students_score_more_than_50/total_students)*100
print("Probability of Students Scoring more than 50 marks in Reading :",probability_of_students_scoring_more_than_50_in_reading)
```

The Probability of Students Scoring more than 50 marks in Writing : 87.6

```
In [13]: total_students = data.shape[0]
students_score_more_than_50 = data[data['writing score'] > 50].shape[0]
probability_of_students_scoring_more_than_50_in_writing = (students_score_more_than_50/total_students)*100
print("Probability of Students Scoring more than 50 marks in Writing :",probability_of_students_scoring_more_than_50_in_writing)
```

The Probability of Students Passing in all the Subjects is 93.90%

```
In [14]: total_students = data.shape[0]
number_of_students_passing_in_all_subjects = data[(data['math score'] > 40) & (data['writing score'] > 40) & (data['reading score'] > 40)].shape[0]
probability_of_students_passing_in_all_the_subjects = (number_of_students_passing_in_all_subjects/total_students)*100
print("The Probability of Students Passing in all the Subjects is (0:.2f)%".format(probability_of_students_passing_in_all_the_subjects))
```

The Probability of Students Passing in all the Subjects is 93.90%

```
In [15]: total_students = data.shape[0]
students_score_more_than_90 = data[(data['math score'] > 90) & (data['writing score'] > 90) & (data['reading score'] > 90)].shape[0]
probability_of_students_scoring_more_than_90_in_all_the_subjects = (number_of_students_scoring_more_than_90/total_students)*100
print("The Probability of Students Scoring more than 90 in all the Subjects is (0:.2f)%".format(probability_of_students_scoring_more_than_90_in_all_the_subjects))
```

The Probability of Students Scoring more than 90 in all the Subjects is 2.30%

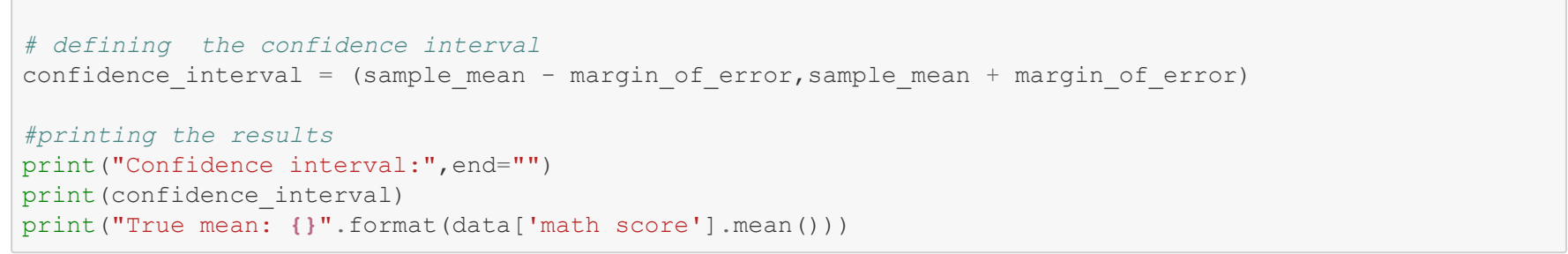
Skewness for the Maths, Reading and Writing Scores

```
In [16]: plt.subplot(1, 3, 1)
sns.distplot(data['math score'])

plt.subplot(1, 3, 2)
sns.distplot(data['reading score'])

plt.subplot(1, 3, 3)
sns.distplot(data['writing score'])

plt.suptitle('Checking for Skewness', fontsize = 18)
plt.xlabel('Degree')
```



```
In [17]: # taking seed so that everytime the random values come out to be constant
np.random.seed(6)

# taking 100 sample values from the dataset of 1000 values
sample_math_marks = np.random.choice(a = data['math score'], size = 100)

# getting the sample mean
print("Sample mean for Math Scores :",sample_math_marks.mean())

# getting the population mean
print("Population mean for Math Scores :",data['math score'].mean())

# taking 100 sample values from the dataset of 1000 values
sample_reading_marks = np.random.choice(a = data['reading score'], size = 100)

# getting the sample mean
print("Sample mean for Reading Scores :",sample_reading_marks.mean())

# getting the population mean
print("Population mean for Reading Scores :",data['reading score'].mean())

# taking 100 sample values from the dataset of 1000 values
sample_writing_marks = np.random.choice(a = data['writing score'], size = 100)

# getting the sample mean
print("Sample mean for Writing Scores :",sample_writing_marks.mean())

# getting the population mean
print("Population mean for Writing Scores :",data['writing score'].mean())
```

Sample mean for Math Scores : 63.12
Population mean for Math Scores : 68.089
Sample mean for Reading Scores : 66.5
Population mean for Reading Scores : 69.169
Sample mean for Writing Scores : 71.46
Population mean for Writing Scores : 68.054

Confidence Interval for Math Score

```
In [18]: # Import the scipy package
import scipy.stats as stats
import math

# seed the random value
np.random.seed(10)

# taking a sample size
sample_size = 1000
sample = np.random.choice(a = data['math score'], size = sample_size)
sample_mean = sample.mean()

# getting the z-critical value
z_critical = stats.norm.ppf(q = 0.95)

# checking the z-critical value
print("z-critical value: ",z_critical)

# getting the population standard deviation
pop_stdev = data['math score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining the confidence interval
confidence_interval = (sample_mean - margin_of_error,sample_mean + margin_of_error)

#printing the results
print("Confidence interval:",end="")
print(confidence_interval)
print("True mean: {}".format(data['math score'].mean()))
```

z-critical value: 1.6448536269514722
Confidence interval: (66.8096003503861, 68.40470516671672)
True mean: 66.089

Confidence Interval for Reading Score

```
In [19]: # Importing the scipy package
import scipy.stats as stats
import math

# seeding the random values
np.random.seed(10)

# taking a sample size
sample_size = 1000
sample = np.random.choice(a = data['reading score'], size = sample_size)
sample_mean = sample.mean()

# getting the z-critical value
z_critical = stats.norm.ppf(q = 0.95)

# checking the z-critical value
print("z-critical value: ",z_critical)

# checking the population standard deviation
pop_stdev = data['reading score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining confidence interval
confidence_interval = (sample_mean - margin_of_error,sample_mean + margin_of_error)

# printing the results
print("Confidence interval:",end="")
print(confidence_interval)
print("True mean: {}".format(data['reading score'].mean()))
```

z-critical value: 1.6448536269514722
Confidence interval: (67.7573737011645, 69.2746266298355)
True mean: 69.169

Confidence Interval for Writing Score

```
In [20]: # seeding the random values
np.random.seed(10)

# taking a sample size
sample_size = 1000
sample = np.random.choice(a = data['writing score'], size = sample_size)
sample_mean = sample.mean()

# getting the z-critical value
z_critical = stats.norm.ppf(q = 0.95)

# checking the z-critical value
print("z-critical value: ",z_critical)

# checking the population standard deviation
pop_stdev = data['writing score'].std()

# checking the margin of error
margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

# defining confidence variable
confidence_interval = (sample_mean - margin_of_error,sample_mean + margin_of_error)

# printing the results
print("Confidence interval:",end="")
print(confidence_interval)
print("True mean: {}".format(data['writing score'].mean()))
```

z-critical value: 1.6448536269514722
Confidence interval: (66.8096003503861, 68.39039964969137)
True mean: 68.054

Grouping Operations

Number of Students Scoring 90 in all the Subjects

```
In [21]: data[(data['gender'] == 'female') & (data['math score'] > 90) & (data['writing score'] > 90) & (data['reading score'] > 90)]
```

179	female	group D	some high school	standard	completed	97	100	100
451	female	group E	some college	standard	none	100	92	97
458	female	group E	bachelor's degree	standard	none	100	100	100
546	female	group A	some high school	standard	completed	92	100	97
566	female	group E	bachelor's degree	free/reduced	completed	92	100	100

Compare the scores secured by Boys and Girls

```
In [22]: data.groupby(['gender']).agg(['min','median','max'])
```

gender	math score	reading score	writing score
female	min 0, median 65, max 100	min 17, median 73, max 100	min 10, median 74, max 100
male	min 27, median 69, max 100	min 43, median 76, max 100	min 15, median 64, max 100

Effect of Lunch on Student's Performance

```
In [23]: data[(data['lunch', 'gender', 'math score', 'reading score', 'writing score']).groupby(['lunch', 'gender']).agg('median')]
```

lunch	gender	math score	reading score	writing score
free/reduced	female	57.0	67.0	68.0
	male	62.0	61.0	59.0
standard	female	67.0	75.0	76.0
	male	72.0	67.5	67.0

Effect of Test Preparation on Student's Performance

```
In [24]: data[(data['test preparation course', 'gender', 'math score', 'reading score', 'writing score']).groupby(['test preparation course', 'gender']).agg('median')]
```

test preparation course	gender	math score	reading score	writing score
completed	female	67	78	79
	male	63	71	70
none	female	62	71	70
	male	67	63	60

Effect of Race and Ethnicity on Student's Performance

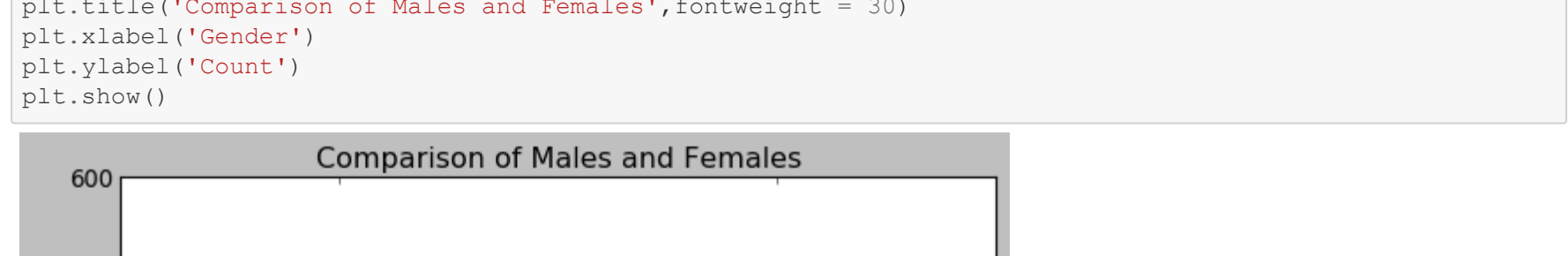
```
In [25]: data[(data['race/ethnicity', 'math score', 'reading score', 'writing score']).groupby(['race/ethnicity']).agg('mean')]
```

race/ethnicity	math score	reading score	writing score
group A	61.0	64.0	62.0
group B	63.0	67.0	67.0
group C	65.0	71.0	68.0
group D	69.0	71.0	72.0
group E	74.5	74.0	72.0

Data Visualizations

```
In [26]: # visualising the number of male and female in the data set

plt.rcParams['figure.figsize'] = (15,3)
plt.style.use('classic text')
sns.countplot(data['gender'],palette = 'bone')
plt.title('Comparison of Males and Females',fontweight = 30)
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```



```
In [27]: # visualising the different groups in the dataset

plt.rcParams['figure.figsize'] = (15,3)
plt.style.use('ggplot')
sns.countplot(data['race/ethnicity'],palette = 'pink')
plt.title('Comparison of Parental Education',fontweight = 30,fontsize = 20)
plt.xlabel('Groups')
plt.ylabel('Count')
plt.show()
```



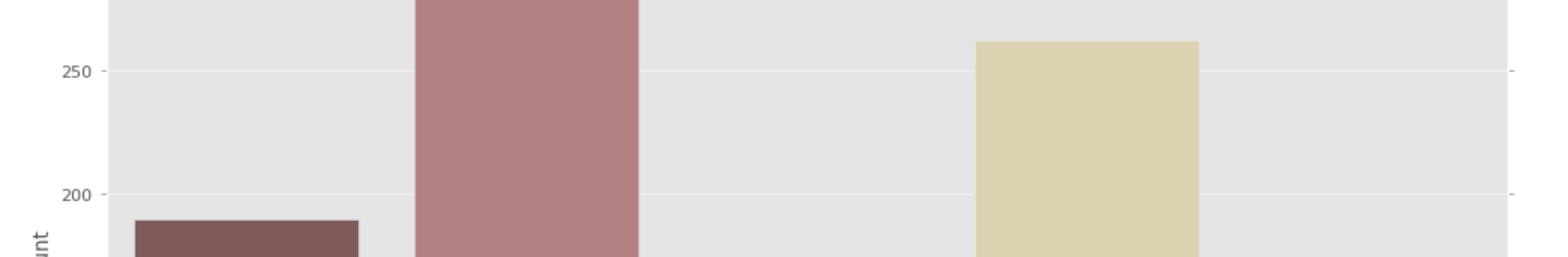
```
In [28]: # visualising the different parental education levels

plt.rcParams['figure.figsize'] = (15,3)
plt.style.use('fivethirtyeight')
sns.countplot(data['parental level of education'],palette = 'Blues')
plt.title('Comparison of Parental Education',fontweight = 30,fontsize = 20)
plt.xlabel('Degree')
plt.ylabel('Count')
plt.show()
```



```
In [29]: # visualising different types of lunch

plt.rcParams['figure.figsize'] = (15,3)
plt.style.use('seaborn-talk')
sns.countplot(data['lunch'],palette = 'PuBu')
plt.title('Comparison of different types of lunch',fontweight = 30,fontsize = 20)
plt.xlabel('types of lunch')
plt.ylabel('Count')
plt.show()
```




```
In [30]: # visualising math score
plt.rcParams['figure.figsize'] = (15,9)
plt.style.use('tableau-colorblind10')

sns.countplot(data['math score'],palette = 'Bupu')
plt.title('Comparison of math score', fontweight = 30,fontsize = 20)
plt.xlabel('score')
plt.ylabel('count')
plt.xticks(rotation = 90)
plt.show()
```



```
In [31]: # visualising reading score
plt.rcParams['figure.figsize'] = (15,9)
plt.style.use('tableau-colorblind10')

sns.countplot(data['reading score'],palette = 'RdPu')
plt.title('Comparison of reading score', fontweight = 30,fontsize = 20)
plt.xlabel('score')
plt.ylabel('count')
plt.xticks(rotation = 90)
plt.show()
```



```
In [32]: # Visualising writing score
plt.rcParams['figure.figsize'] = (15,9)
plt.style.use('tableau-colorblind10')

sns.countplot(data['writing score'],palette = 'prism')
plt.title('Comparison of writing score', fontweight = 30,fontsize = 20)
plt.xlabel('score')
plt.ylabel('count')
plt.xticks(rotation = 90)
plt.show()
```



```
In [33]: # comparison of gender and race/ethnicity
plt.rcParams['figure.figsize'] = (15,9)
pd.crosstab(data['gender'],data['race/ethnicity'])
x.div(x.sum(1).astype(float),axis = 0).plot(kind = 'bar',stacked = False)
plt.title('Gender vs Race',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [34]: # comparison of race/ethnicity and parental level of education
plt.rcParams['figure.figsize'] = (15,9)
x = pd.crosstab(data['race/ethnicity'],data['parental level of education'])
x.div(x.sum(1).astype(float),axis = 0).plot(kind = 'bar',stacked = True)
plt.title('Race vs Parental Education',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [35]: # comparison of parental degree and test preparation
plt.rcParams['figure.figsize'] = (15,9)
sns.countplot(x = 'parental level of education',data = data,hue = 'test preparation course',palette = 'dark')
plt.title('Parental Education vs Test Preparation Course',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [36]: # comparison of race/ethnicity and test preparation
plt.rcParams['figure.figsize'] = (15,9)
sns.countplot(x = 'race/ethnicity',data = data,hue = 'test preparation course',palette = 'bright')
plt.title('Race vs Test Preparation Course',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [37]: # feature engineering on the data to visualize and solve the dataset more accurately
#setting a pass on the three subjects individually
passmarks = 40

# creating a new column pass_mth telling whether the students pass or fail
data['pass_mth'] = np.where(data['math score'] < passmarks,'Fail','Pass')
data['pass_mth'].value_counts().plot.pie(colors = ['lightblue','lightgreen'])

plt.title('Pass/Fail in Maths',fontweight = 30,fontsize = 20)
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



```
In [38]: # creating new column pass_reading telling whether the students pass or fail
passmarks = 40

data['pass_reading'] = np.where(data['reading score'] < passmarks,'Fail','Pass')
data['pass_reading'].value_counts().plot.pie(colors = ['orange','yellow'])

plt.title('Pass/Fail in Reading',fontweight = 30,fontsize = 20)
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



```
In [39]: # creating new column pass_writing telling whether the students pass or fail
passmarks = 40

data['pass_writing'] = np.where(data['writing score'] < passmarks,'Fail','Pass')
data['pass_writing'].value_counts().plot.pie(colors = ['orange','grey'])

plt.title('Pass/Fail in Writing',fontweight = 30,fontsize = 20)
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



```
In [40]: # computing the total score for each student
import warnings
warnings.filterwarnings('ignore')

data['total_score'] = data['math score'] + data['reading score'] + data['writing score']
sns.distplot(data['total_score'],color = 'magenta')

plt.title('Comparison of Total Score of all the Students',fontweight = 30,fontsize = 20)
plt.xlabel('total score scored by all students')
plt.ylabel('count')
plt.show()
```



```
In [41]: # computing percentage for each of the students
from math import*
import warnings
warnings.filterwarnings('ignore')

data['percentage'] = data['total_score']/3

for i in range(0,1000):
    data['percentage'][i] = ceil(data['percentage'][i])

plt.rcParams['figure.figsize'] = (15,9)
sns.distplot(data['percentage'],color = 'orange')

plt.title('Comparison of percentage scored by all the students',fontweight = 30,fontsize = 20)
plt.xlabel('Percentage scored')
plt.ylabel('Count')
plt.show()
```



```
In [42]: # checking which student is fail overall
data['status'] = data.apply(lambda x : 'Fail' if x['pass_mth'] == 'Fail' or x['pass_reading'] == 'Fail'
or x['pass_writing'] == 'Fail' else 'pass',axis = 1)

data['status'].value_counts(dropna = False).plot.pie(colors = ['grey','crimson'])

plt.title('Overall Results',fontweight = 30,fontsize = 20)
plt.xlabel('status')
plt.ylabel('count')
plt.show()
```



```
In [43]: # assigning grades to the grades according to the following criteria
# 41 - 60 : grade D
# 61 - 70 : grade C
# 71 - 80 : grade B
# 81 - 90 : grade A
# 91 - 100 : grade O

def getgrade(percentage,status):
    if status == 'Fail':
        return 'B'
    if (percentage >= 90):
        return 'O'
    if (percentage >= 80):
        return 'A'
    if (percentage >= 70):
        return 'B'
    if (percentage >= 60):
        return 'C'
    if (percentage >= 40):
        return 'D'
    else:
        return 'G'

data['grades'] = data.apply(lambda x:getgrade(x['percentage'],x['status']),axis = 1)

data['grades'].value_counts()
```

Out[43]:

```
B    260
C    252
D    223
A    156
O     58
E     51
Name: grades, dtype: int64
```

```
In [44]: # plotting a pie chart for the distribution of various grades amongst the students
labels = ('Grade O','Grade A','Grade B','Grade C','Grade D','Grade E')
sizes = (59,156,260,252,223,51)
colors = ('yellow','gold','lightskyblue','lightcoral','pink','cyan')
explode = (0.0001,0.0001,0.0001,0.0001,0.0001,0.0001)

patches, texts = plt.pie(sizes,colors = colors,shadow = True,startangle = 90)
plt.legend(patches,labels)
plt.title('Distribution of Grades among Students',fontweight = 30,fontsize = 20)
plt.axis('equal')
plt.tight_layout()
plt.show()
```



```
In [45]: # comparing parental education and student's grade
plt.rcParams['figure.figsize'] = (15,9)

x = pd.crosstab(data['parental level of education'],data['grades'])
color = plt.cm.copper(np.linspace(0,1,8))
x.div(x.sum(1).astype(float),axis = 0).plot(kind = 'bar',stacked = True,color = color)

plt.title('Parental Education vs Student's Grades',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [46]: # plotting again using seaborn for better visualization
sns.countplot(x = data['parental level of education'],data = data,hue = data['grades'],palette = 'paste1')
plt.title('Parental Education vs Grades of Students',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [47]: # comparing the distribution of grades amont males and females
sns.countplot(x = data['grades'],data = data,hue = data['gender'],palette = 'cubehelix')
plt.title('Grades vs Gender',fontweight = 30,fontsize = 20)
plt.show()
```


Label Encoding


```
[48]: from sklearn.preprocessing import LabelEncoder

# creating an encoder
le = LabelEncoder

# label encoder for test preparation course
data['test preparation course'] = le.fit_transform(data['test preparation course'])

# label encoding for lunch
data['lunch'] = le.fit_transform(data['lunch'])

# label encoding for race/ethnicity
# mapping values to each of the categories
data['race/ethnicity'] = data['race/ethnicity'].replace('group A',1)
data['race/ethnicity'] = data['race/ethnicity'].replace('group B',2)
data['race/ethnicity'] = data['race/ethnicity'].replace('group C',3)
data['race/ethnicity'] = data['race/ethnicity'].replace('group D',4)
data['race/ethnicity'] = data['race/ethnicity'].replace('group E',5)

# label encoding for parental education
data['parental level of education'] = le.fit_transform(data['parental level of education'])

# label encoding for gender
data['gender'] = le.fit_transform(data['gender'])

# label encoding for pass_math
data['pass_math'] = le.fit_transform(data['pass_math'])

# label encoding for pass_reading
data['pass_reading'] = le.fit_transform(data['pass_reading'])

# label encoding for pass_writing
data['pass_writing'] = le.fit_transform(data['pass_writing'])

# label encoding for status
data['status'] = le.fit_transform(data['status'])
```

Data Preparation

```
In [49]: # splitting the dependent and independent variables

x = data.iloc[:,1:14]
y = data.iloc[:,14]

print(x.shape)
print(y.shape)

(1000, 14)
(1000,)
```

```
In [50]: # splitting the dataset into training and test sets

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_state = 45)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

(750, 14)
(750,)
(250, 14)
(250,)
```

```
In [51]: # importing the MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# creating a scaler
mm = MinMaxScaler()

# feeding the independent variable into the scaler
x_train = mm.fit_transform(x_train)
x_test = mm.transform(x_test)
```

Modelling

Logistic Regression

```
In [52]: from sklearn.linear_model import LogisticRegression

# creating a model
model = LogisticRegression()

model.fit(x_train,y_train)

# predicting the test set results
y_pred = model.predict(x_test)

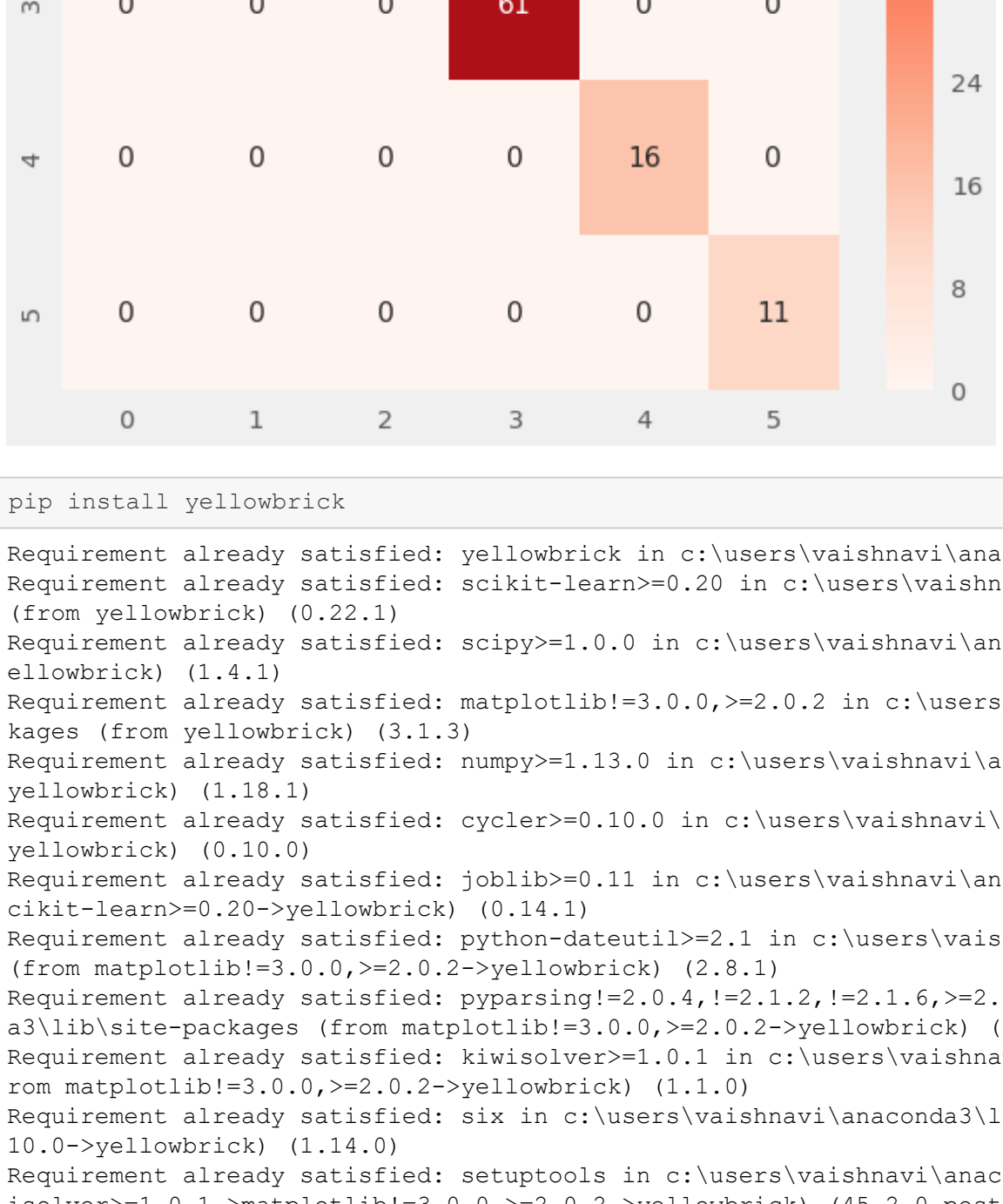
# calculating the classification accuracies
print("Training Accuracy :",model.score(x_train,y_train))
print("Testing Accuracy :",model.score(x_test,y_test))

Training Accuracy : 0.852
Testing Accuracy : 0.812
```

```
In [53]: # printing the confusion matrix
from sklearn.metrics import confusion_matrix

# creating a confusion matrix
cm = confusion_matrix(y_test,y_pred)

# printing the confusion matrix
plt.rcParams['figure.figsize'] = (8,8)
sns.heatmap(cm,annot = True,cmap = 'Greens')
plt.title('Confusion Matrix for Logistic Regression',fontweight = 30,fontsize = 20)
plt.show()
```



Random Forest

```
In [54]: from sklearn.ensemble import RandomForestClassifier

# creating a model
model = RandomForestClassifier()

# feeding the training data to the model
model.fit(x_train,y_train)

# predicting the x-test results
y_pred = model.predict(x_test)

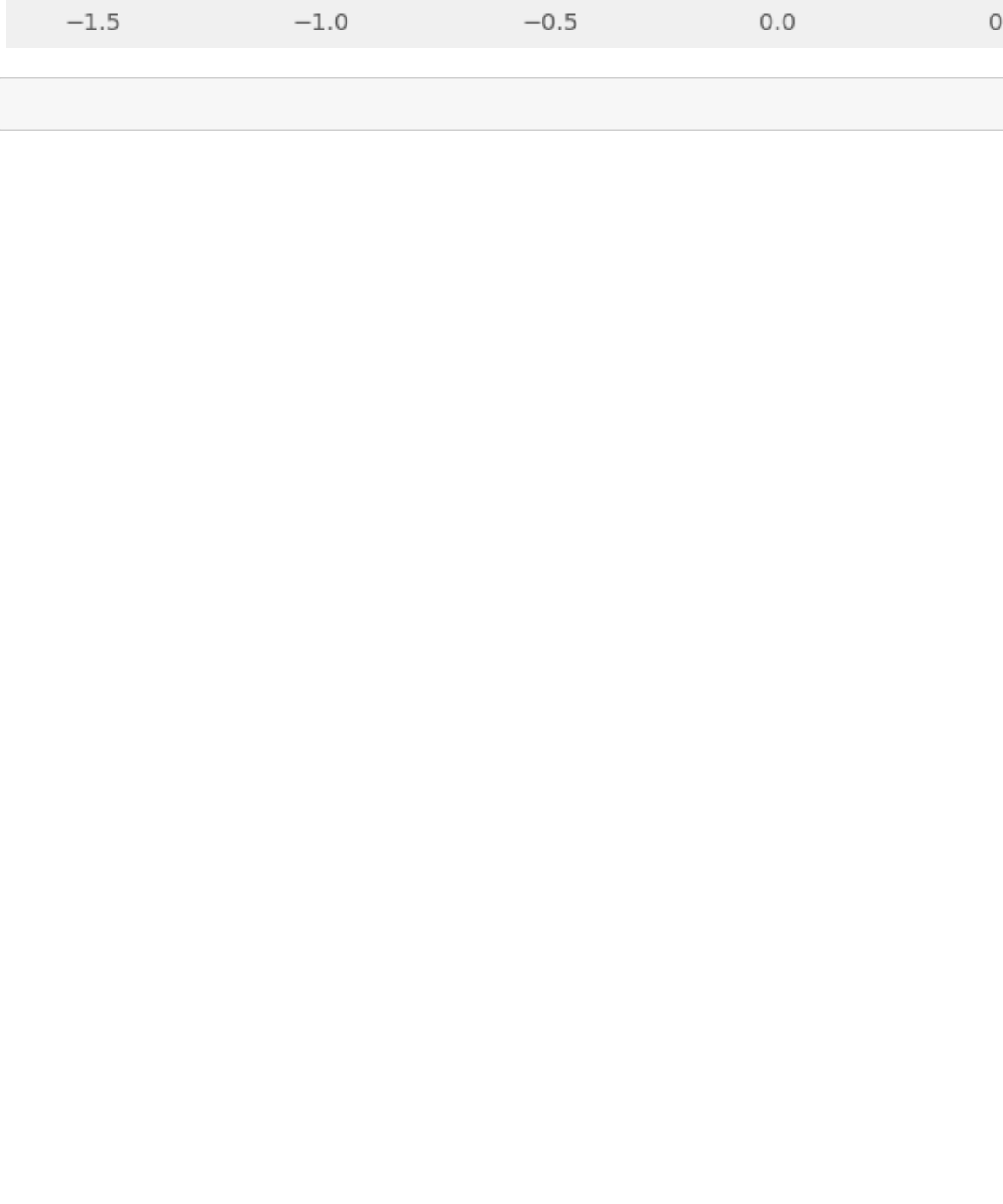
# calculating the accuracies
print("Training Accuracy :",model.score(x_train,y_train))
print("Testing Accuracy :",model.score(x_test,y_test))

Training Accuracy : 1.0
Testing Accuracy : 1.0
```

```
In [55]: # printing the confusion matrix
from sklearn.metrics import confusion_matrix

# creating a confusion matrix
cm = confusion_matrix(y_test,y_pred)

# printing the confusion matrix
plt.rcParams['figure.figsize'] = (8,8)
sns.heatmap(cm,annot = True,cmap = 'Reds')
plt.title('Confusion Matrix for Random Forest',fontweight = 30,fontsize = 20)
plt.show()
```



```
In [56]: pip install yellowbrick

Requirement already satisfied: yellowbrick in c:\users\vaishnavi\anaconda3\lib\site-packages (1.1)
Requirement already satisfied: scikit-learn>=0.20 in c:\users\vaishnavi\anaconda3\lib\site-packages (from yellowbrick) (0.22.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\vaishnavi\anaconda3\lib\site-packages (from yellowbrick) (1.4.1)
Requirement already satisfied: matplotlib>=3.0.0,>=2.0.2 in c:\users\vaishnavi\anaconda3\lib\site-packages (from yellowbrick) (3.1.3)
Requirement already satisfied: numpy>=1.13.0 in c:\users\vaishnavi\anaconda3\lib\site-packages (from yellowbrick) (1.18.1)
Requirement already satisfied: cycle>=0.10.0 in c:\users\vaishnavi\anaconda3\lib\site-packages (from yellowbrick) (0.10.0)
Requirement already satisfied: joblib>=0.11 in c:\users\vaishnavi\anaconda3\lib\site-packages (from scikit-learn>=0.20->yellowbrick) (0.14.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib>=3.0.0,>=2.0.2->yellowbrick) (2.8.1)
Requirement already satisfied: pyparsing>=2.0.4,<=2.1.2,>=2.1.6,>=2.0.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib>=3.0.0,>=2.0.2->yellowbrick) (1.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\vaishnavi\anaconda3\lib\site-packages (from matplotlib>=3.0.0,>=2.0.2->yellowbrick) (1.1.0)
Requirement already satisfied: six in c:\users\vaishnavi\anaconda3\lib\site-packages (from cycle>=0.10.0->yellowbrick) (1.14.0)
Requirement already satisfied: setuptools in c:\users\vaishnavi\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matplotlib>=3.0.0,>=2.0.2->yellowbrick) (45.2.0.post2020210)
Note: you may need to restart the kernel to use updated packages.
```

```
In [57]: from pandas.plotting import radviz
fig,ax = plt.subplots(figsize = (12,12))
new_df = x.copy()
new_df['status'] = y
radviz(new_df,"status",ax = ax,colormap = "rocket")
plt.title("Radial Visualization for Target",fontsize = 20)
plt.show()
```



```
In [ ]:
```