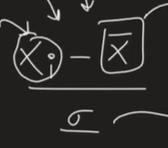
Standardization:





**import** numpy **as** np *# linear algebra*

**import** pandas **as** pd *# data processing*

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

In [43]:

df **=** pd**.**read\_csv('Social\_Network\_Ads.csv')

In [44]:

df**=**df**.**iloc[:,2:]

In [45]:

df**.**sample(5)

Out[45]:

|  | **Age** | **EstimatedSalary** | **Purchased** |
| --- | --- | --- | --- |
| **137** | 30 | 107000 | 1 |
| **251** | 37 | 52000 | 0 |
| **262** | 55 | 125000 | 1 |
| **144** | 34 | 25000 | 0 |
| **292** | 55 | 39000 | 1 |

**Train test split**

In [46]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(df**.**drop('Purchased', axis**=**1),

df['Purchased'],

test\_size**=**0.3,

random\_state**=**0)

X\_train**.**shape, X\_test**.**shape

Out[46]:

((280, 2), (120, 2))

**StandardScaler**

In [47]:

**from** sklearn.preprocessing **import** StandardScaler

scaler **=** StandardScaler()

*# fit the scaler to the train set, it will learn the parameters*

scaler**.**fit(X\_train)

*# transform train and test sets*

X\_train\_scaled **=** scaler**.**transform(X\_train)

X\_test\_scaled **=** scaler**.**transform(X\_test)

In [48]:

scaler**.**mean\_

Out[48]:

array([3.78642857e+01, 6.98071429e+04])

In [49]:

X\_train

Out[49]:

|  | **Age** | **EstimatedSalary** |
| --- | --- | --- |
| **92** | 26 | 15000 |
| **223** | 60 | 102000 |
| **234** | 38 | 112000 |
| **232** | 40 | 107000 |
| **377** | 42 | 53000 |
| **...** | ... | ... |
| **323** | 48 | 30000 |
| **192** | 29 | 43000 |
| **117** | 36 | 52000 |
| **47** | 27 | 54000 |
| **172** | 26 | 118000 |

280 rows × 2 columns

In [52]:

X\_train\_scaled

Out[52]:

|  | **Age** | **EstimatedSalary** |
| --- | --- | --- |
| **0** | -1.163172 | -1.584970 |
| **1** | 2.170181 | 0.930987 |
| **2** | 0.013305 | 1.220177 |
| **3** | 0.209385 | 1.075582 |
| **4** | 0.405465 | -0.486047 |
| **...** | ... | ... |
| **275** | 0.993704 | -1.151185 |
| **276** | -0.869053 | -0.775237 |
| **277** | -0.182774 | -0.514966 |
| **278** | -1.065133 | -0.457127 |
| **279** | -1.163172 | 1.393691 |

280 rows × 2 columns

In [51]:

X\_train\_scaled **=** pd**.**DataFrame(X\_train\_scaled, columns**=**X\_train**.**columns)

X\_test\_scaled **=** pd**.**DataFrame(X\_test\_scaled, columns**=**X\_test**.**columns)

In [9]:

np**.**round(X\_train**.**describe(), 1)

Out[9]:

|  | **Age** | **EstimatedSalary** |
| --- | --- | --- |
| **count** | 280.0 | 280.0 |
| **mean** | 37.9 | 69807.1 |
| **std** | 10.2 | 34641.2 |
| **min** | 18.0 | 15000.0 |
| **25%** | 30.0 | 43000.0 |
| **50%** | 37.0 | 70500.0 |
| **75%** | 46.0 | 88000.0 |
| **max** | 60.0 | 150000.0 |

In [10]:

np**.**round(X\_train\_scaled**.**describe(), 1)

Out[10]:

|  | **Age** | **EstimatedSalary** |
| --- | --- | --- |
| **count** | 280.0 | 280.0 |
| **mean** | 0.0 | 0.0 |
| **std** | 1.0 | 1.0 |
| **min** | -1.9 | -1.6 |
| **25%** | -0.8 | -0.8 |
| **50%** | -0.1 | 0.0 |
| **75%** | 0.8 | 0.5 |
| **max** | 2.2 | 2.3 |

**Effect of Scaling**

In [11]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

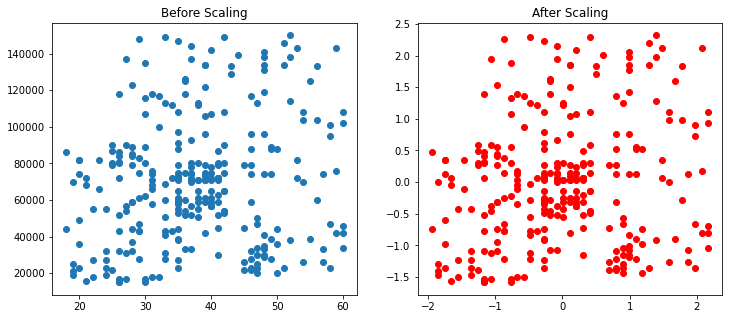
ax1**.**scatter(X\_train['Age'], X\_train['EstimatedSalary'])

ax1**.**set\_title("Before Scaling")

ax2**.**scatter(X\_train\_scaled['Age'], X\_train\_scaled['EstimatedSalary'],color**=**'red')

ax2**.**set\_title("After Scaling")

plt**.**show()



In [12]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Before Scaling')

sns**.**kdeplot(X\_train['Age'], ax**=**ax1)

sns**.**kdeplot(X\_train['EstimatedSalary'], ax**=**ax1)

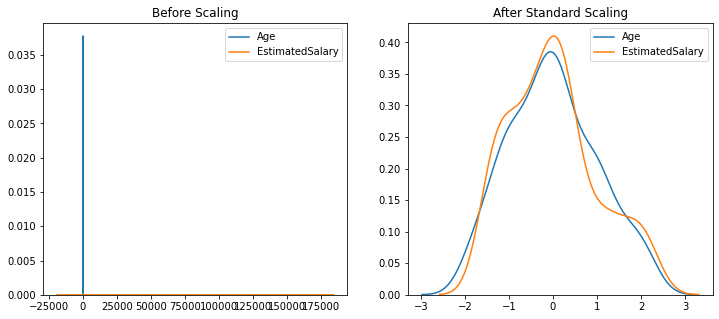
*# after scaling*

ax2**.**set\_title('After Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['Age'], ax**=**ax2)

sns**.**kdeplot(X\_train\_scaled['EstimatedSalary'], ax**=**ax2)

plt**.**show()



**Comparison of Distributions**

In [13]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Age Distribution Before Scaling')

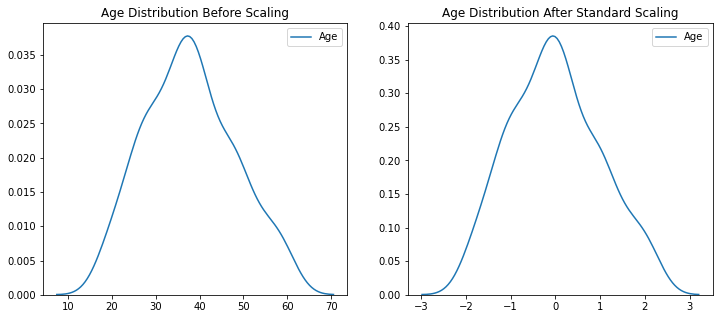
sns**.**kdeplot(X\_train['Age'], ax**=**ax1)

*# after scaling*

ax2**.**set\_title('Age Distribution After Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['Age'], ax**=**ax2)

plt**.**show()



In [14]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Salary Distribution Before Scaling')

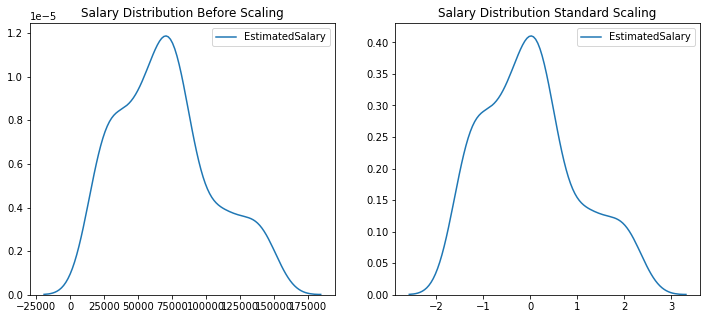
sns**.**kdeplot(X\_train['EstimatedSalary'], ax**=**ax1)

*# after scaling*

ax2**.**set\_title('Salary Distribution Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['EstimatedSalary'], ax**=**ax2)

plt**.**show()



**Why scaling is important?**

In [15]:

**from** sklearn.linear\_model **import** LogisticRegression

In [17]:

lr **=** LogisticRegression()

lr\_scaled **=** LogisticRegression()

In [18]:

lr**.**fit(X\_train,y\_train)

lr\_scaled**.**fit(X\_train\_scaled,y\_train)

Out[18]:

LogisticRegression()

In [20]:

y\_pred **=** lr**.**predict(X\_test)

y\_pred\_scaled **=** lr\_scaled**.**predict(X\_test\_scaled)

In [21]:

**from** sklearn.metrics **import** accuracy\_score

In [22]:

print("Actual",accuracy\_score(y\_test,y\_pred))

print("Scaled",accuracy\_score(y\_test,y\_pred\_scaled))

Actual 0.6583333333333333

Scaled 0.8666666666666667

In [23]:

**from** sklearn.tree **import** DecisionTreeClassifier

In [24]:

dt **=** DecisionTreeClassifier()

dt\_scaled **=** DecisionTreeClassifier()

In [25]:

dt**.**fit(X\_train,y\_train)

dt\_scaled**.**fit(X\_train\_scaled,y\_train)

Out[25]:

DecisionTreeClassifier()

In [26]:

y\_pred **=** dt**.**predict(X\_test)

y\_pred\_scaled **=** dt\_scaled**.**predict(X\_test\_scaled)

In [27]:

print("Actual",accuracy\_score(y\_test,y\_pred))

print("Scaled",accuracy\_score(y\_test,y\_pred\_scaled))

Actual 0.875

Scaled 0.875

In [29]:

df**.**describe()

Out[29]:

|  | **Age** | **EstimatedSalary** | **Purchased** |
| --- | --- | --- | --- |
| **count** | 400.000000 | 400.000000 | 400.000000 |
| **mean** | 37.655000 | 69742.500000 | 0.357500 |
| **std** | 10.482877 | 34096.960282 | 0.479864 |
| **min** | 18.000000 | 15000.000000 | 0.000000 |
| **25%** | 29.750000 | 43000.000000 | 0.000000 |
| **50%** | 37.000000 | 70000.000000 | 0.000000 |
| **75%** | 46.000000 | 88000.000000 | 1.000000 |
| **max** | 60.000000 | 150000.000000 | 1.000000 |

**Effect of Outlier**

In [34]:

df **=** df**.**append(pd**.**DataFrame({'Age':[5,90,95],'EstimatedSalary':[1000,250000,350000],'Purchased':[0,1,1]}),ignore\_index**=True**)

In [32]:

df

Out[32]:

|  | **Age** | **EstimatedSalary** | **Purchased** |
| --- | --- | --- | --- |
| **0** | 19 | 19000 | 0 |
| **1** | 35 | 20000 | 0 |
| **2** | 26 | 43000 | 0 |
| **3** | 27 | 57000 | 0 |
| **4** | 19 | 76000 | 0 |
| **...** | ... | ... | ... |
| **395** | 46 | 41000 | 1 |
| **396** | 51 | 23000 | 1 |
| **397** | 50 | 20000 | 1 |
| **398** | 36 | 33000 | 0 |
| **399** | 49 | 36000 | 1 |

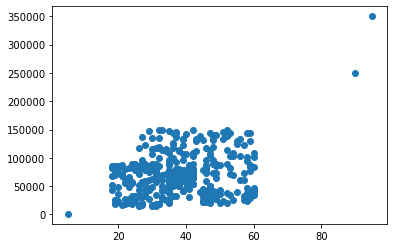
400 rows × 3 columns

In [36]:

plt**.**scatter(df['Age'], df['EstimatedSalary'])

Out[36]:

<matplotlib.collections.PathCollection at 0x1d6823eba00>



In [37]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(df**.**drop('Purchased', axis**=**1),

df['Purchased'],

test\_size**=**0.3,

random\_state**=**0)

X\_train**.**shape, X\_test**.**shape

Out[37]:

((282, 2), (121, 2))

In [38]:

**from** sklearn.preprocessing **import** StandardScaler

scaler **=** StandardScaler()

*# fit the scaler to the train set, it will learn the parameters*

scaler**.**fit(X\_train)

*# transform train and test sets*

X\_train\_scaled **=** scaler**.**transform(X\_train)

X\_test\_scaled **=** scaler**.**transform(X\_test)

In [40]:

X\_train\_scaled **=** pd**.**DataFrame(X\_train\_scaled, columns**=**X\_train**.**columns)

X\_test\_scaled **=** pd**.**DataFrame(X\_test\_scaled, columns**=**X\_test**.**columns)

In [41]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

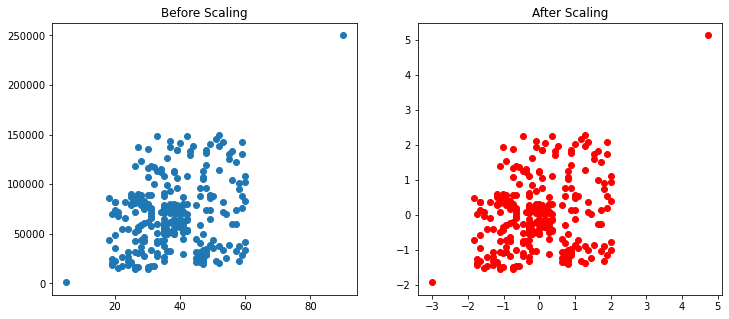
ax1**.**scatter(X\_train['Age'], X\_train['EstimatedSalary'])

ax1**.**set\_title("Before Scaling")

ax2**.**scatter(X\_train\_scaled['Age'], X\_train\_scaled['EstimatedSalary'],color**=**'red')

ax2**.**set\_title("After Scaling")

plt**.**show()



In [ ]:

Normalization:



A blackboard with white text

Description automatically generated

Min-Max scaling:

**import** numpy **as** np *# linear algebra*

**import** pandas **as** pd *# data processing*

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

In [32]:

df **=** pd**.**read\_csv('wine\_data.csv',header**=None**,usecols**=**[0,1,2])

df**.**columns**=**['Class label', 'Alcohol', 'Malic acid']

In [33]:

df

Out[33]:

|  | **Class label** | **Alcohol** | **Malic acid** |
| --- | --- | --- | --- |
| **0** | 1 | 14.23 | 1.71 |
| **1** | 1 | 13.20 | 1.78 |
| **2** | 1 | 13.16 | 2.36 |
| **3** | 1 | 14.37 | 1.95 |
| **4** | 1 | 13.24 | 2.59 |
| **...** | ... | ... | ... |
| **173** | 3 | 13.71 | 5.65 |
| **174** | 3 | 13.40 | 3.91 |
| **175** | 3 | 13.27 | 4.28 |
| **176** | 3 | 13.17 | 2.59 |
| **177** | 3 | 14.13 | 4.10 |

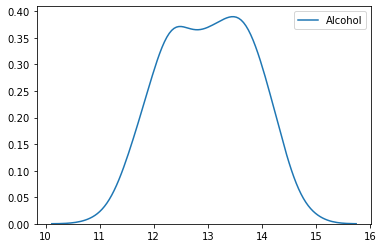
178 rows × 3 columns

In [34]:

sns**.**kdeplot(df['Alcohol'])

Out[34]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x216645559d0>

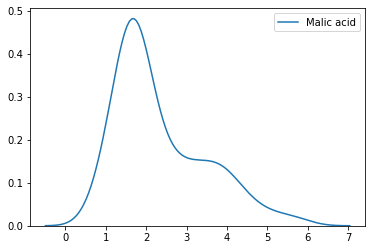


In [36]:

sns**.**kdeplot(df['Malic acid'])

Out[36]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x21666036940>



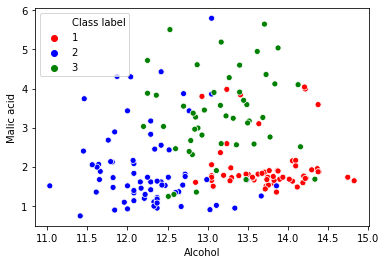
In [43]:

color\_dict**=**{1:'red',3:'green',2:'blue'}

sns**.**scatterplot(df['Alcohol'],df['Malic acid'],hue**=**df['Class label'],palette**=**color\_dict)

Out[43]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x21665b02f70>



In [44]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(df**.**drop('Class label', axis**=**1),

df['Class label'],

test\_size**=**0.3,

random\_state**=**0)

X\_train**.**shape, X\_test**.**shape

Out[44]:

((124, 2), (54, 2))

In [45]:

**from** sklearn.preprocessing **import** MinMaxScaler

scaler **=** MinMaxScaler()

*# fit the scaler to the train set, it will learn the parameters*

scaler**.**fit(X\_train)

*# transform train and test sets*

X\_train\_scaled **=** scaler**.**transform(X\_train)

X\_test\_scaled **=** scaler**.**transform(X\_test)

In [46]:

X\_train\_scaled **=** pd**.**DataFrame(X\_train\_scaled, columns**=**X\_train**.**columns)

X\_test\_scaled **=** pd**.**DataFrame(X\_test\_scaled, columns**=**X\_test**.**columns)

In [47]:

np**.**round(X\_train**.**describe(), 1)

Out[47]:

|  | **Alcohol** | **Malic acid** |
| --- | --- | --- |
| **count** | 124.0 | 124.0 |
| **mean** | 13.0 | 2.4 |
| **std** | 0.8 | 1.1 |
| **min** | 11.0 | 0.9 |
| **25%** | 12.4 | 1.6 |
| **50%** | 13.0 | 1.9 |
| **75%** | 13.6 | 3.2 |
| **max** | 14.8 | 5.6 |

In [48]:

np**.**round(X\_train\_scaled**.**describe(), 1)

Out[48]:

|  | **Alcohol** | **Malic acid** |
| --- | --- | --- |
| **count** | 124.0 | 124.0 |
| **mean** | 0.5 | 0.3 |
| **std** | 0.2 | 0.2 |
| **min** | 0.0 | 0.0 |
| **25%** | 0.4 | 0.2 |
| **50%** | 0.5 | 0.2 |
| **75%** | 0.7 | 0.5 |
| **max** | 1.0 | 1.0 |

In [52]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

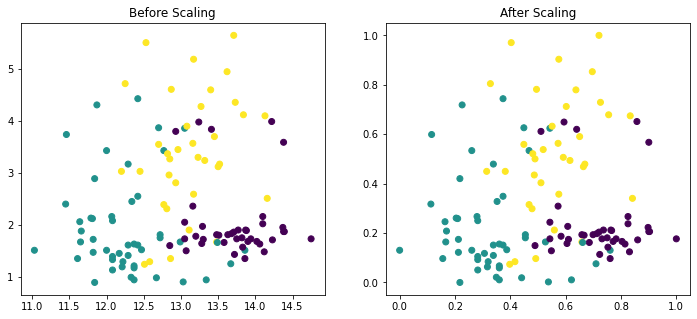
ax1**.**scatter(X\_train['Alcohol'], X\_train['Malic acid'],c**=**y\_train)

ax1**.**set\_title("Before Scaling")

ax2**.**scatter(X\_train\_scaled['Alcohol'], X\_train\_scaled['Malic acid'],c**=**y\_train)

ax2**.**set\_title("After Scaling")

plt**.**show()



In [53]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Before Scaling')

sns**.**kdeplot(X\_train['Alcohol'], ax**=**ax1)

sns**.**kdeplot(X\_train['Malic acid'], ax**=**ax1)

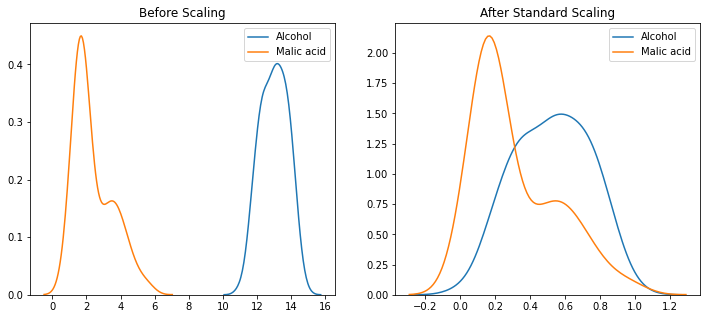
*# after scaling*

ax2**.**set\_title('After Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['Alcohol'], ax**=**ax2)

sns**.**kdeplot(X\_train\_scaled['Malic acid'], ax**=**ax2)

plt**.**show()



In [54]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Alcohol Distribution Before Scaling')

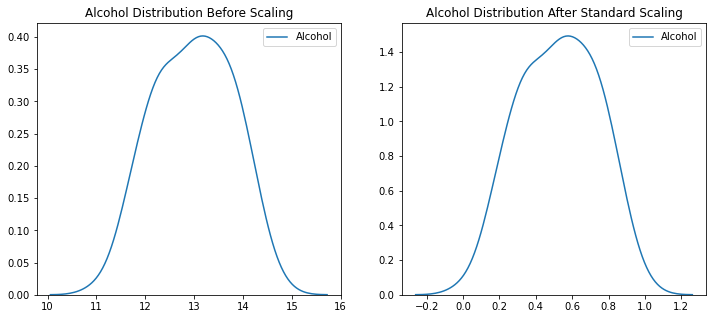
sns**.**kdeplot(X\_train['Alcohol'], ax**=**ax1)

*# after scaling*

ax2**.**set\_title('Alcohol Distribution After Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['Alcohol'], ax**=**ax2)

plt**.**show()



In [55]:

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(12, 5))

*# before scaling*

ax1**.**set\_title('Malic acid Distribution Before Scaling')

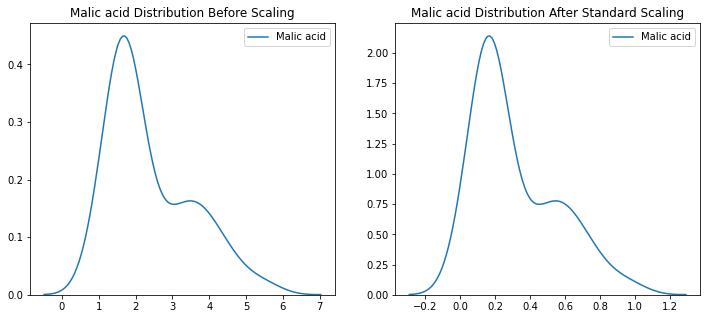
sns**.**kdeplot(X\_train['Malic acid'], ax**=**ax1)

*# after scaling*

ax2**.**set\_title('Malic acid Distribution After Standard Scaling')

sns**.**kdeplot(X\_train\_scaled['Malic acid'], ax**=**ax2)

plt**.**show()

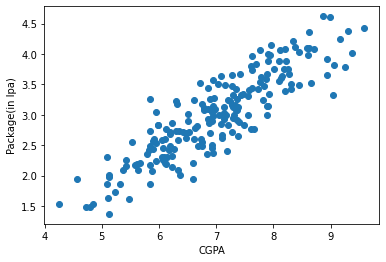


Linear Regression:

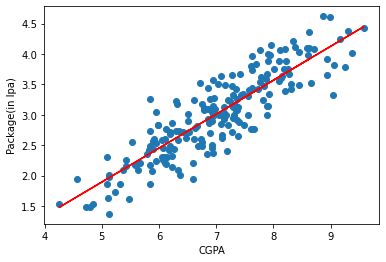


1. Simple linear regression:
2. **import** matplotlib.pyplot **as** plt
3. **import** pandas **as** pd
4. **import** numpy **as** np
5. In [74]:
6. df **=** pd**.**read\_csv('placement.csv')
7. In [75]:
8. df**.**head()
9. Out[75]:

|  | **cgpa** | **package** |
| --- | --- | --- |
| **0** | 6.89 | 3.26 |
| **1** | 5.12 | 1.98 |
| **2** | 7.82 | 3.25 |
| **3** | 7.42 | 3.67 |
| **4** | 6.94 | 3.57 |

1. In [76]:
2. plt**.**scatter(df['cgpa'],df['package'])
3. plt**.**xlabel('CGPA')
4. plt**.**ylabel('Package(in lpa)')
5. Out[76]:
6. Text(0, 0.5, 'Package(in lpa)')
7. 
8. In [77]:
9. X **=** df**.**iloc[:,0:1]
10. y **=** df**.**iloc[:,**-**1]
11. In [79]:
12. y
13. Out[79]:
14. 0 3.26
15. 1 1.98
16. 2 3.25
17. 3 3.67
18. 4 3.57
19. ...
20. 195 2.46
21. 196 2.57
22. 197 3.24
23. 198 3.96
24. 199 2.33
25. Name: package, Length: 200, dtype: float64
26. In [80]:
27. **from** sklearn.model\_selection **import** train\_test\_split
28. X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y,test\_size**=**0.2,random\_state**=**2)
29. In [81]:
30. **from** sklearn.linear\_model **import** LinearRegression
31. In [82]:
32. lr **=** LinearRegression()
33. In [83]:
34. lr**.**fit(X\_train,y\_train)
35. Out[83]:
36. LinearRegression()
37. In [84]:
38. X\_test
39. Out[84]:

|  | **cgpa** |
| --- | --- |
| **112** | 8.58 |
| **29** | 7.15 |
| **182** | 5.88 |
| **199** | 6.22 |
| **193** | 4.57 |
| **85** | 4.79 |
| **10** | 5.32 |
| **54** | 6.86 |
| **115** | 8.35 |
| **35** | 6.87 |
| **12** | 8.94 |
| **92** | 7.90 |
| **13** | 6.93 |
| **126** | 5.91 |
| **174** | 7.32 |
| **2** | 7.82 |
| **44** | 5.09 |
| **3** | 7.42 |
| **113** | 6.94 |
| **14** | 7.73 |
| **23** | 6.19 |
| **25** | 7.28 |
| **6** | 6.73 |
| **134** | 7.20 |
| **165** | 8.21 |
| **173** | 6.75 |
| **45** | 7.87 |
| **65** | 7.60 |
| **48** | 8.63 |
| **122** | 5.12 |
| **178** | 8.15 |
| **64** | 7.36 |
| **9** | 8.31 |
| **57** | 6.60 |
| **78** | 6.59 |
| **71** | 7.47 |
| **128** | 7.93 |
| **176** | 6.29 |
| **131** | 6.37 |
| **53** | 6.47 |

1. In [85]:
2. y\_test
3. Out[85]:
4. 112 4.10
5. 29 3.49
6. 182 2.08
7. 199 2.33
8. 193 1.94
9. 85 1.48
10. 10 1.86
11. 54 3.09
12. 115 4.21
13. 35 2.87
14. 12 3.65
15. 92 4.00
16. 13 2.89
17. 126 2.60
18. 174 2.99
19. 2 3.25
20. 44 1.86
21. 3 3.67
22. 113 2.37
23. 14 3.42
24. 23 2.48
25. 25 3.65
26. 6 2.60
27. 134 2.83
28. 165 4.08
29. 173 2.56
30. 45 3.58
31. 65 3.81
32. 48 4.09
33. 122 2.01
34. 178 3.63
35. 64 2.92
36. 9 3.51
37. 57 1.94
38. 78 2.21
39. 71 3.34
40. 128 3.34
41. 176 3.23
42. 131 2.01
43. 53 2.61
44. Name: package, dtype: float64
45. In [100]:
46. lr**.**predict(X\_test**.**iloc[0]**.**values**.**reshape(1,1))
47. Out[100]:
48. array([3.89111601])
49. In [95]:
50. plt**.**scatter(df['cgpa'],df['package'])
51. plt**.**plot(X\_train,lr**.**predict(X\_train),color**=**'red')
52. plt**.**xlabel('CGPA')
53. plt**.**ylabel('Package(in lpa)')
54. Out[95]:
55. Text(0, 0.5, 'Package(in lpa)')
56. 
57. In [97]:
58. m **=** lr**.**coef\_
59. In [99]:
60. b **=** lr**.**intercept\_
61. In [101]:
62. *# y = mx + b*
63. m **\*** 8.58 **+** b
64. Out[101]:
65. array([3.89111601])
66. In [102]:
67. m **\*** 9.5 **+** b
68. Out[102]:
69. array([4.40443183])
70. In [103]:
71. m **\*** 100 **+** b
72. Out[103]:
73. array([54.89908542])

**One hot encoding:**



**import** numpy **as** np

**import** pandas **as** pd

In [119]:

df **=** pd**.**read\_csv('cars.csv')

In [120]:

df**.**head()

Out[120]:

|  | **brand** | **km\_driven** | **fuel** | **owner** | **selling\_price** |
| --- | --- | --- | --- | --- | --- |
| **0** | Maruti | 145500 | Diesel | First Owner | 450000 |
| **1** | Skoda | 120000 | Diesel | Second Owner | 370000 |
| **2** | Honda | 140000 | Petrol | Third Owner | 158000 |
| **3** | Hyundai | 127000 | Diesel | First Owner | 225000 |
| **4** | Maruti | 120000 | Petrol | First Owner | 130000 |

In [121]:

df['owner']**.**value\_counts()

Out[121]:

First Owner 5289

Second Owner 2105

Third Owner 555

Fourth & Above Owner 174

Test Drive Car 5

Name: owner, dtype: int64

**1. OneHotEncoding using Pandas**

In [99]:

pd**.**get\_dummies(df,columns**=**['fuel','owner'])

Out[99]:

|  | **brand** | **km\_driven** | **selling\_price** | **fuel\_CNG** | **fuel\_Diesel** | **fuel\_LPG** | **fuel\_Petrol** | **owner\_First Owner** | **owner\_Fourth & Above Owner** | **owner\_Second Owner** | **owner\_Test Drive Car** | **owner\_Third Owner** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Maruti | 145500 | 450000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **1** | Skoda | 120000 | 370000 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | Honda | 140000 | 158000 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **3** | Hyundai | 127000 | 225000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **4** | Maruti | 120000 | 130000 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8123** | Hyundai | 110000 | 320000 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **8124** | Hyundai | 119000 | 135000 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **8125** | Maruti | 120000 | 382000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **8126** | Tata | 25000 | 290000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **8127** | Tata | 25000 | 290000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

8128 rows × 12 columns

**2. K-1 OneHotEncoding**

In [100]:

pd**.**get\_dummies(df,columns**=**['fuel','owner'],drop\_first**=True**)

Out[100]:

|  | **brand** | **km\_driven** | **selling\_price** | **fuel\_Diesel** | **fuel\_LPG** | **fuel\_Petrol** | **owner\_Fourth & Above Owner** | **owner\_Second Owner** | **owner\_Test Drive Car** | **owner\_Third Owner** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Maruti | 145500 | 450000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | Skoda | 120000 | 370000 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | Honda | 140000 | 158000 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **3** | Hyundai | 127000 | 225000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | Maruti | 120000 | 130000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8123** | Hyundai | 110000 | 320000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **8124** | Hyundai | 119000 | 135000 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **8125** | Maruti | 120000 | 382000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8126** | Tata | 25000 | 290000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8127** | Tata | 25000 | 290000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

8128 rows × 10 columns

**3. OneHotEncoding using Sklearn**

In [122]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(df**.**iloc[:,0:4],df**.**iloc[:,**-**1],test\_size**=**0.2,random\_state**=**2)

In [111]:

X\_train**.**head()

Out[111]:

|  | **brand** | **km\_driven** | **fuel** | **owner** |
| --- | --- | --- | --- | --- |
| **5571** | Hyundai | 35000 | Diesel | First Owner |
| **2038** | Jeep | 60000 | Diesel | First Owner |
| **2957** | Hyundai | 25000 | Petrol | First Owner |
| **7618** | Mahindra | 130000 | Diesel | Second Owner |
| **6684** | Hyundai | 155000 | Diesel | First Owner |

In [123]:

**from** sklearn.preprocessing **import** OneHotEncoder

In [137]:

ohe **=** OneHotEncoder(drop**=**'first',sparse**=False**,dtype**=**np**.**int32)

In [138]:

X\_train\_new **=** ohe**.**fit\_transform(X\_train[['fuel','owner']])

In [139]:

X\_test\_new **=** ohe**.**transform(X\_test[['fuel','owner']])

In [140]:

X\_train\_new**.**shape

Out[140]:

(6502, 7)

In [141]:

np**.**hstack((X\_train[['brand','km\_driven']]**.**values,X\_train\_new))

Out[141]:

array([['Hyundai', 35000, 1, ..., 0, 0, 0],

['Jeep', 60000, 1, ..., 0, 0, 0],

['Hyundai', 25000, 0, ..., 0, 0, 0],

...,

['Tata', 15000, 0, ..., 0, 0, 0],

['Maruti', 32500, 1, ..., 1, 0, 0],

['Isuzu', 121000, 1, ..., 0, 0, 0]], dtype=object)

In [ ]:

In [ ]:

**4. OneHotEncoding with Top Categories**

In [143]:

counts **=** df['brand']**.**value\_counts()

In [144]:

df['brand']**.**nunique()

threshold **=** 100

In [146]:

repl **=** counts[counts **<=** threshold]**.**index

In [150]:

pd**.**get\_dummies(df['brand']**.**replace(repl, 'uncommon'))**.**sample(5)

Out[150]:

|  | **BMW** | **Chevrolet** | **Ford** | **Honda** | **Hyundai** | **Mahindra** | **Maruti** | **Renault** | **Skoda** | **Tata** | **Toyota** | **Volkswagen** | **uncommon** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8093** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3274** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2966** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1092** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5355** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [ ]:

**Outlier removal using Z-score:**



**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

In [76]:

df **=** pd**.**read\_csv('placement.csv')

In [77]:

df**.**shape

Out[77]:

(1000, 3)

In [78]:

df**.**sample(5)

Out[78]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |
| **689** | 8.02 | 67.0 | 0 |
| **111** | 6.48 | 33.0 | 0 |
| **991** | 7.04 | 57.0 | 0 |
| **835** | 6.67 | 65.0 | 1 |
| **772** | 6.63 | 26.0 | 0 |

In [79]:

plt**.**figure(figsize**=**(16,5))

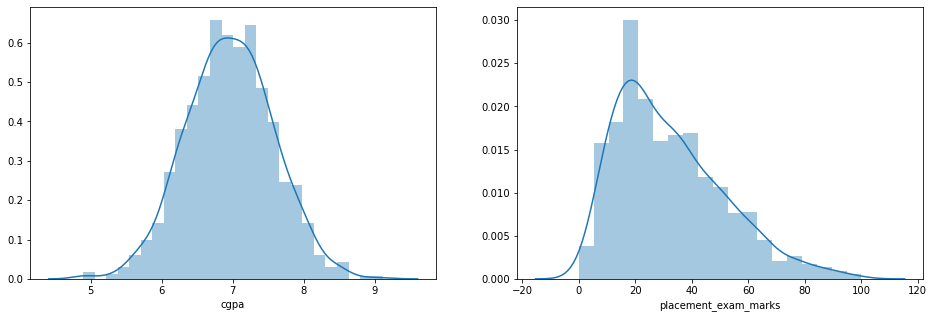
plt**.**subplot(1,2,1)

sns**.**distplot(df['cgpa'])

plt**.**subplot(1,2,2)

sns**.**distplot(df['placement\_exam\_marks'])

plt**.**show()



In [ ]:

df['placement\_exam\_marks']**.**skew()

In [57]:

print("Mean value of cgpa",df['cgpa']**.**mean())

print("Std value of cgpa",df['cgpa']**.**std())

print("Min value of cgpa",df['cgpa']**.**min())

print("Max value of cgpa",df['cgpa']**.**max())

Mean value of cgpa 6.96124000000001

Std value of cgpa 0.6158978751323894

Min value of cgpa 4.89

Max value of cgpa 9.12

In [58]:

*# Finding the boundary values*

print("Highest allowed",df['cgpa']**.**mean() **+** 3**\***df['cgpa']**.**std())

print("Lowest allowed",df['cgpa']**.**mean() **-** 3**\***df['cgpa']**.**std())

Highest allowed 8.808933625397177

Lowest allowed 5.113546374602842

In [59]:

*# Finding the outliers*

df[(df['cgpa'] **>** 8.80) **|** (df['cgpa'] **<** 5.11)]

Out[59]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |
| **485** | 4.92 | 44.0 | 1 |
| **995** | 8.87 | 44.0 | 1 |
| **996** | 9.12 | 65.0 | 1 |
| **997** | 4.89 | 34.0 | 0 |
| **999** | 4.90 | 10.0 | 1 |

**Trimming**

In [60]:

*# Trimming*

new\_df **=** df[(df['cgpa'] **<** 8.80) **&** (df['cgpa'] **>** 5.11)]

new\_df

Out[60]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |
| **0** | 7.19 | 26.0 | 1 |
| **1** | 7.46 | 38.0 | 1 |
| **2** | 7.54 | 40.0 | 1 |
| **3** | 6.42 | 8.0 | 1 |
| **4** | 7.23 | 17.0 | 0 |
| **...** | ... | ... | ... |
| **991** | 7.04 | 57.0 | 0 |
| **992** | 6.26 | 12.0 | 0 |
| **993** | 6.73 | 21.0 | 1 |
| **994** | 6.48 | 63.0 | 0 |
| **998** | 8.62 | 46.0 | 1 |

995 rows × 3 columns

In [62]:

*# Approach 2*

*# Calculating the Zscore*

df['cgpa\_zscore'] **=** (df['cgpa'] **-** df['cgpa']**.**mean())**/**df['cgpa']**.**std()

In [63]:

df**.**head()

Out[63]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** | **cgpa\_zscore** |
| --- | --- | --- | --- | --- |
| **0** | 7.19 | 26.0 | 1 | 0.371425 |
| **1** | 7.46 | 38.0 | 1 | 0.809810 |
| **2** | 7.54 | 40.0 | 1 | 0.939701 |
| **3** | 6.42 | 8.0 | 1 | -0.878782 |
| **4** | 7.23 | 17.0 | 0 | 0.436371 |

In [64]:

df[df['cgpa\_zscore'] **>** 3]

Out[64]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** | **cgpa\_zscore** |
| --- | --- | --- | --- | --- |
| **995** | 8.87 | 44.0 | 1 | 3.099150 |
| **996** | 9.12 | 65.0 | 1 | 3.505062 |

In [65]:

df[df['cgpa\_zscore'] **<** **-**3]

Out[65]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** | **cgpa\_zscore** |
| --- | --- | --- | --- | --- |
| **485** | 4.92 | 44.0 | 1 | -3.314251 |
| **997** | 4.89 | 34.0 | 0 | -3.362960 |
| **999** | 4.90 | 10.0 | 1 | -3.346724 |

In [66]:

df[(df['cgpa\_zscore'] **>** 3) **|** (df['cgpa\_zscore'] **<** **-**3)]

Out[66]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** | **cgpa\_zscore** |
| --- | --- | --- | --- | --- |
| **485** | 4.92 | 44.0 | 1 | -3.314251 |
| **995** | 8.87 | 44.0 | 1 | 3.099150 |
| **996** | 9.12 | 65.0 | 1 | 3.505062 |
| **997** | 4.89 | 34.0 | 0 | -3.362960 |
| **999** | 4.90 | 10.0 | 1 | -3.346724 |

In [67]:

*# Trimming*

new\_df **=** df[(df['cgpa\_zscore'] **<** 3) **&** (df['cgpa\_zscore'] **>** **-**3)]

In [68]:

new\_df

Out[68]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** | **cgpa\_zscore** |
| --- | --- | --- | --- | --- |
| **0** | 7.19 | 26.0 | 1 | 0.371425 |
| **1** | 7.46 | 38.0 | 1 | 0.809810 |
| **2** | 7.54 | 40.0 | 1 | 0.939701 |
| **3** | 6.42 | 8.0 | 1 | -0.878782 |
| **4** | 7.23 | 17.0 | 0 | 0.436371 |
| **...** | ... | ... | ... | ... |
| **991** | 7.04 | 57.0 | 0 | 0.127878 |
| **992** | 6.26 | 12.0 | 0 | -1.138565 |
| **993** | 6.73 | 21.0 | 1 | -0.375452 |
| **994** | 6.48 | 63.0 | 0 | -0.781363 |
| **998** | 8.62 | 46.0 | 1 | 2.693239 |

995 rows × 4 columns

**Capping**

In [69]:

upper\_limit **=** df['cgpa']**.**mean() **+** 3**\***df['cgpa']**.**std()

lower\_limit **=** df['cgpa']**.**mean() **-** 3**\***df['cgpa']**.**std()

In [71]:

lower\_limit

Out[71]:

5.113546374602842

In [72]:

df['cgpa'] **=** np**.**where(

df['cgpa']**>**upper\_limit,

upper\_limit,

np**.**where(

df['cgpa']**<**lower\_limit,

lower\_limit,

df['cgpa']

)

)

In [73]:

df**.**shape

Out[73]:

(1000, 4)

In [74]:

df['cgpa']**.**describe()

Out[74]:

count 1000.000000

mean 6.961499

std 0.612688

min 5.113546

25% 6.550000

50% 6.960000

75% 7.370000

max 8.808934

Name: cgpa, dtype: float64

In [ ]:

Outlier removal using IQR Method:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

In [32]:

df **=** pd**.**read\_csv('placement.csv')

In [33]:

df**.**head()

Out[33]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |
| **0** | 7.19 | 26.0 | 1 |
| **1** | 7.46 | 38.0 | 1 |
| **2** | 7.54 | 40.0 | 1 |
| **3** | 6.42 | 8.0 | 1 |
| **4** | 7.23 | 17.0 | 0 |

In [34]:

plt**.**figure(figsize**=**(16,5))

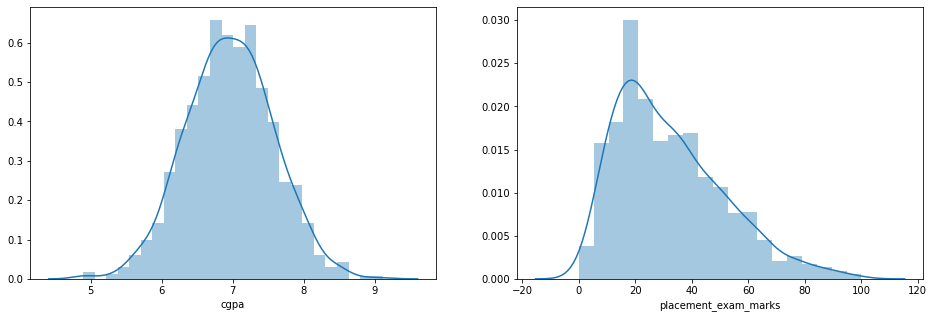
plt**.**subplot(1,2,1)

sns**.**distplot(df['cgpa'])

plt**.**subplot(1,2,2)

sns**.**distplot(df['placement\_exam\_marks'])

plt**.**show()



In [35]:

df['placement\_exam\_marks']**.**describe()

Out[35]:

count 1000.000000

mean 32.225000

std 19.130822

min 0.000000

25% 17.000000

50% 28.000000

75% 44.000000

max 100.000000

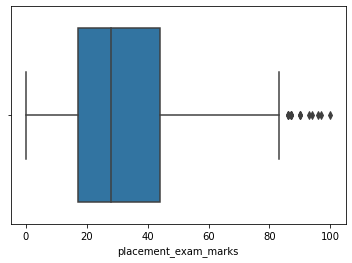
Name: placement\_exam\_marks, dtype: float64

In [36]:

sns**.**boxplot(df['placement\_exam\_marks'])

Out[36]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x177db650730>



In [37]:

*# Finding the IQR*

percentile25 **=** df['placement\_exam\_marks']**.**quantile(0.25)

percentile75 **=** df['placement\_exam\_marks']**.**quantile(0.75)

In [39]:

percentile75

Out[39]:

44.0

In [40]:

iqr **=** percentile75 **-** percentile25

In [41]:

iqr

Out[41]:

27.0

In [42]:

upper\_limit **=** percentile75 **+** 1.5 **\*** iqr

lower\_limit **=** percentile25 **-** 1.5 **\*** iqr

In [43]:

print("Upper limit",upper\_limit)

print("Lower limit",lower\_limit)

Upper limit 84.5

Lower limit -23.5

**Finding Outliers**

In [44]:

df[df['placement\_exam\_marks'] **>** upper\_limit]

Out[44]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |
| **9** | 7.75 | 94.0 | 1 |
| **40** | 6.60 | 86.0 | 1 |
| **61** | 7.51 | 86.0 | 0 |
| **134** | 6.33 | 93.0 | 0 |
| **162** | 7.80 | 90.0 | 0 |
| **283** | 7.09 | 87.0 | 0 |
| **290** | 8.38 | 87.0 | 0 |
| **311** | 6.97 | 87.0 | 1 |
| **324** | 6.64 | 90.0 | 0 |
| **630** | 6.56 | 96.0 | 1 |
| **685** | 6.05 | 87.0 | 1 |
| **730** | 6.14 | 90.0 | 1 |
| **771** | 7.31 | 86.0 | 1 |
| **846** | 6.99 | 97.0 | 0 |
| **917** | 5.95 | 100.0 | 0 |

In [45]:

df[df['placement\_exam\_marks'] **<** lower\_limit]

Out[45]:

|  | **cgpa** | **placement\_exam\_marks** | **placed** |
| --- | --- | --- | --- |

**Trimming**

In [46]:

new\_df **=** df[df['placement\_exam\_marks'] **<** upper\_limit]

In [47]:

new\_df**.**shape

Out[47]:

(985, 3)

In [48]:

*# Comparing*

plt**.**figure(figsize**=**(16,8))

plt**.**subplot(2,2,1)

sns**.**distplot(df['placement\_exam\_marks'])

plt**.**subplot(2,2,2)

sns**.**boxplot(df['placement\_exam\_marks'])

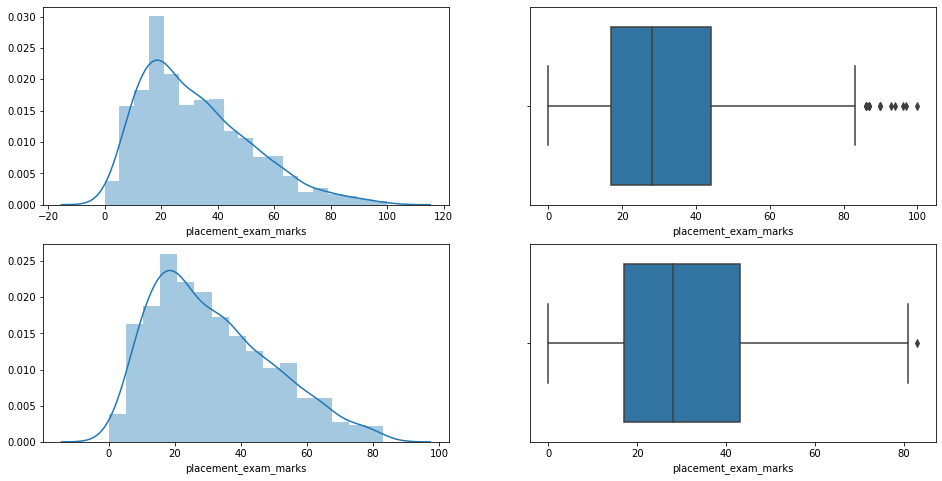
plt**.**subplot(2,2,3)

sns**.**distplot(new\_df['placement\_exam\_marks'])

plt**.**subplot(2,2,4)

sns**.**boxplot(new\_df['placement\_exam\_marks'])

plt**.**show()



**Capping**

In [49]:

new\_df\_cap **=** df**.**copy()

new\_df\_cap['placement\_exam\_marks'] **=** np**.**where(

new\_df\_cap['placement\_exam\_marks'] **>** upper\_limit,

upper\_limit,

np**.**where(

new\_df\_cap['placement\_exam\_marks'] **<** lower\_limit,

lower\_limit,

new\_df\_cap['placement\_exam\_marks']

)

)

In [ ]:

np**.**where(condtion,true,false)

In [50]:

new\_df\_cap**.**shape

Out[50]:

(1000, 3)

In [51]:

*# Comparing*

plt**.**figure(figsize**=**(16,8))

plt**.**subplot(2,2,1)

sns**.**distplot(df['placement\_exam\_marks'])

plt**.**subplot(2,2,2)

sns**.**boxplot(df['placement\_exam\_marks'])

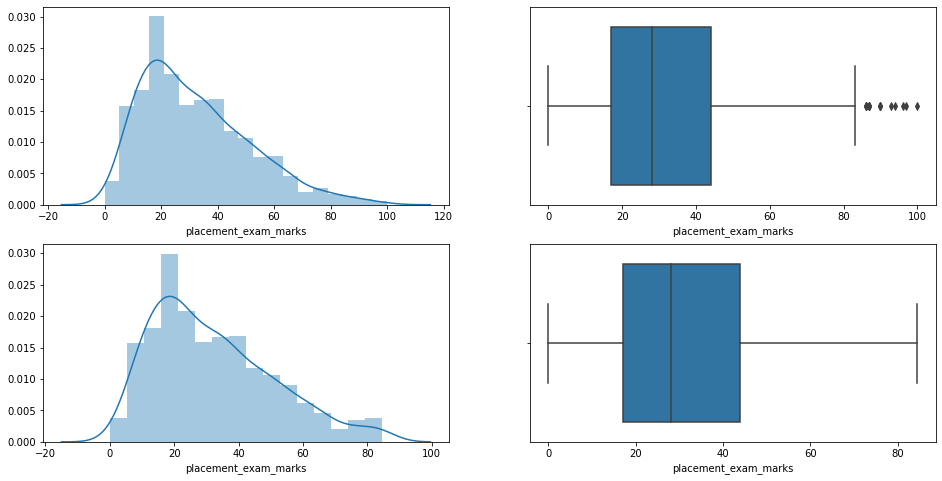
plt**.**subplot(2,2,3)

sns**.**distplot(new\_df\_cap['placement\_exam\_marks'])

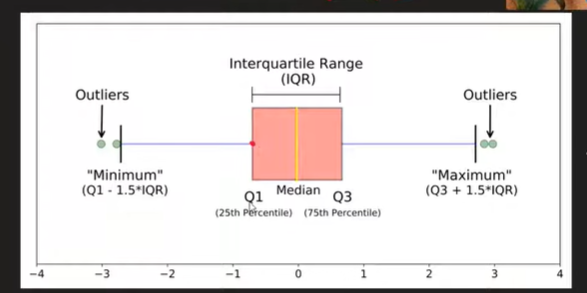
plt**.**subplot(2,2,4)

sns**.**boxplot(new\_df\_cap['placement\_exam\_marks'])

plt**.**show()



In [ ]:



**Missing value:**

KNN Imputer:

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.impute **import** KNNImputer,SimpleImputer

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** accuracy\_score

In [33]:

df **=** pd**.**read\_csv('train.csv')[['Age','Pclass','Fare','Survived']]

In [34]:

df**.**head()

Out[34]:

|  | **Age** | **Pclass** | **Fare** | **Survived** |
| --- | --- | --- | --- | --- |
| **0** | 22.0 | 3 | 7.2500 | 0 |
| **1** | 38.0 | 1 | 71.2833 | 1 |
| **2** | 26.0 | 3 | 7.9250 | 1 |
| **3** | 35.0 | 1 | 53.1000 | 1 |
| **4** | 35.0 | 3 | 8.0500 | 0 |

In [35]:

df**.**isnull()**.**mean() **\*** 100

Out[35]:

Age 19.86532

Pclass 0.00000

Fare 0.00000

Survived 0.00000

dtype: float64

In [36]:

X **=** df**.**drop(columns**=**['Survived'])

y **=** df['Survived']

In [37]:

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y,test\_size**=**0.2,random\_state**=**2)

In [38]:

X\_train**.**head()

Out[38]:

|  | **Age** | **Pclass** | **Fare** |
| --- | --- | --- | --- |
| **30** | 40.0 | 1 | 27.7208 |
| **10** | 4.0 | 3 | 16.7000 |
| **873** | 47.0 | 3 | 9.0000 |
| **182** | 9.0 | 3 | 31.3875 |
| **876** | 20.0 | 3 | 9.8458 |

In [53]:

knn **=** KNNImputer(n\_neighbors**=**3,weights**=**'distance')

X\_train\_trf **=** knn**.**fit\_transform(X\_train)

X\_test\_trf **=** knn**.**transform(X\_test)

In [54]:

lr **=** LogisticRegression()

lr**.**fit(X\_train\_trf,y\_train)

y\_pred **=** lr**.**predict(X\_test\_trf)

accuracy\_score(y\_test,y\_pred)

Out[54]:

0.7150837988826816

In [55]:

*# Comparision with Simple Imputer --> mean*

si **=** SimpleImputer()

X\_train\_trf2 **=** si**.**fit\_transform(X\_train)

X\_test\_trf2 **=** si**.**transform(X\_test)

In [56]:

lr **=** LogisticRegression()

lr**.**fit(X\_train\_trf2,y\_train)

y\_pred2 **=** lr**.**predict(X\_test\_trf2)

accuracy\_score(y\_test,y\_pred2)

Out[56]:

0.6927374301675978

In [ ]:

**PCA:**

**import numpy as np**

**import pandas as pd**

**np.random.seed(23)**

**mu\_vec1 = np.array([0,0,0])**

**cov\_mat1 = np.array([[1,0,0],[0,1,0],[0,0,1]])**

**class1\_sample = np.random.multivariate\_normal(mu\_vec1, cov\_mat1, 20)**

**df = pd.DataFrame(class1\_sample,columns=['feature1','feature2','feature3'])**

**df['target'] = 1**

**mu\_vec2 = np.array([1,1,1])**

**cov\_mat2 = np.array([[1,0,0],[0,1,0],[0,0,1]])**

**class2\_sample = np.random.multivariate\_normal(mu\_vec2, cov\_mat2, 20)**

**df1 = pd.DataFrame(class2\_sample,columns=['feature1','feature2','feature3'])**

**df1['target'] = 0**

**df = df.append(df1,ignore\_index=True)**

**df = df.sample(40)**

**In [53]:**

**df.head()**

**Out[53]:**

|  | **feature1** | **feature2** | **feature3** | **target** |
| --- | --- | --- | --- | --- |
| **2** | **-0.367548** | **-1.137460** | **-1.322148** | **1** |
| **34** | **0.177061** | **-0.598109** | **1.226512** | **0** |
| **14** | **0.420623** | **0.411620** | **-0.071324** | **1** |
| **11** | **1.968435** | **-0.547788** | **-0.679418** | **1** |
| **12** | **-2.506230** | **0.146960** | **0.606195** | **1** |

**In [54]:**

**import plotly.express as px**

***#y\_train\_trf = y\_train.astype(str)***

**fig = px.scatter\_3d(df, x=df['feature1'], y=df['feature2'], z=df['feature3'],**

**color=df['target'].astype('str'))**

**fig.update\_traces(marker=dict(size=12,**

**line=dict(width=2,**

**color='DarkSlateGrey')),**

**selector=dict(mode='markers'))**

**fig.show()**

**In [55]:**

***# Step 1 - Apply standard scaling***

**from sklearn.preprocessing import StandardScaler**

**scaler = StandardScaler()**

**df.iloc[:,0:3] = scaler.fit\_transform(df.iloc[:,0:3])**

**In [56]:**

***# Step 2 - Find Covariance Matrix***

**covariance\_matrix = np.cov([df.iloc[:,0],df.iloc[:,1],df.iloc[:,2]])**

**print('Covariance Matrix:\n', covariance\_matrix)**

**Covariance Matrix:**

**[[1.02564103 0.20478114 0.080118 ]**

**[0.20478114 1.02564103 0.19838882]**

**[0.080118 0.19838882 1.02564103]]**

**In [57]:**

***# Step 3 - Finding EV and EVs***

**eigen\_values, eigen\_vectors = np.linalg.eig(covariance\_matrix)**

**In [58]:**

**eigen\_values**

**Out[58]:**

**array([1.3536065 , 0.94557084, 0.77774573])**

**In [59]:**

**eigen\_vectors**

**Out[59]:**

**array([[-0.53875915, -0.69363291, 0.47813384],**

**[-0.65608325, -0.01057596, -0.75461442],**

**[-0.52848211, 0.72025103, 0.44938304]])**

**In [60]:**

**%pylab inline**

**from matplotlib import pyplot as plt**

**from mpl\_toolkits.mplot3d import Axes3D**

**from mpl\_toolkits.mplot3d import proj3d**

**from matplotlib.patches import FancyArrowPatch**

**class Arrow3D(FancyArrowPatch):**

**def \_\_init\_\_(self, xs, ys, zs, \*args, \*\*kwargs):**

**FancyArrowPatch.\_\_init\_\_(self, (0,0), (0,0), \*args, \*\*kwargs)**

**self.\_verts3d = xs, ys, zs**

**def draw(self, renderer):**

**xs3d, ys3d, zs3d = self.\_verts3d**

**xs, ys, zs = proj3d.proj\_transform(xs3d, ys3d, zs3d, renderer.M)**

**self.set\_positions((xs[0],ys[0]),(xs[1],ys[1]))**

**FancyArrowPatch.draw(self, renderer)**

**fig = plt.figure(figsize=(7,7))**

**ax = fig.add\_subplot(111, projection='3d')**

**ax.plot(df['feature1'], df['feature2'], df['feature3'], 'o', markersize=8, color='blue', alpha=0.2)**

**ax.plot([df['feature1'].mean()], [df['feature2'].mean()], [df['feature3'].mean()], 'o', markersize=10, color='red', alpha=0.5)**

**for v in eigen\_vectors.T:**

**a = Arrow3D([df['feature1'].mean(), v[0]], [df['feature2'].mean(), v[1]], [df['feature3'].mean(), v[2]], mutation\_scale=20, lw=3, arrowstyle="-|>", color="r")**

**ax.add\_artist(a)**

**ax.set\_xlabel('x\_values')**

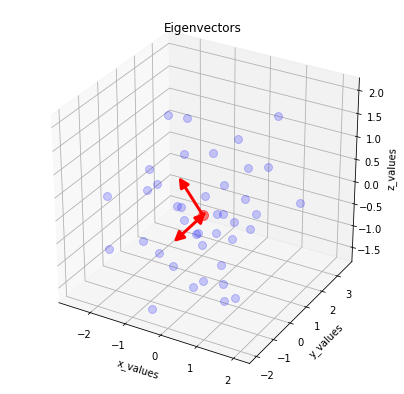
**ax.set\_ylabel('y\_values')**

**ax.set\_zlabel('z\_values')**

**plt.title('Eigenvectors')**

**plt.show()**

**Populating the interactive namespace from numpy and matplotlib**

****

**In [61]:**

**pc = eigen\_vectors[0:2]**

**pc**

**Out[61]:**

**array([[-0.53875915, -0.69363291, 0.47813384],**

**[-0.65608325, -0.01057596, -0.75461442]])**

**In [62]:**

**transformed\_df = np.dot(df.iloc[:,0:3],pc.T)**

***# 40,3 - 3,2***

**new\_df = pd.DataFrame(transformed\_df,columns=['PC1','PC2'])**

**new\_df['target'] = df['target'].values**

**new\_df.head()**

**Out[62]:**

|  | **PC1** | **PC2** | **target** |
| --- | --- | --- | --- |
| **0** | **0.599433** | **1.795862** | **1** |
| **1** | **1.056919** | **-0.212737** | **0** |
| **2** | **-0.271876** | **0.498222** | **1** |
| **3** | **-0.621586** | **0.023110** | **1** |
| **4** | **1.567286** | **1.730967** | **1** |

**In [63]:**

**new\_df['target'] = new\_df['target'].astype('str')**

**fig = px.scatter(x=new\_df['PC1'],**

**y=new\_df['PC2'],**

**color=new\_df['target'],**

**color\_discrete\_sequence=px.colors.qualitative.G10**

**)**

**fig.update\_traces(marker=dict(size=12,**

**line=dict(width=2,**

**color='DarkSlateGrey')),**

**selector=dict(mode='markers'))**

**fig.show()**

**In [ ]:**