All Labs

M2: Python Practice

In [2]:
```
# Lab 1 Exercise 1
for i in [1,2]:
    print(i)
    for j in ['A','B']:
        print(j)
```

```
1
A
B
2
A
B
```

In [8]:
```
# Lab 1 Exercise 2
def absolute_value(num):
    """ This function returns the absolute
      value of the entered number.
      It takes one numeric argument"""

    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(6))
print(absolute_value(-6))
```

```
6
6
```

In [11]:
```
# Lab 2 Exercise 1
set_s = {-2,-1,2,3,4}

def count_positives_and_negatives(s):
    """ This function returns the number of
    positive and negative values in a set.
    Input: a numeric set of which zero is not a part. """
    positive_count = 0
    negative_count = 0

    for num in s:
        if num > 0:
            positive_count = positive_count + 1
        elif num < 0:
            negative_count = negative_count + 1

    return positive_count, negative_count

print("The set consists of:", set_s)
p, n = count_positives_and_negatives(set_s)

print("The number of positive values in S is:", p)
print("The number of negative values in S is:", n)
```

```
The set consists of: {2, 3, 4, -1, -2}
The number of positive values in S is: 3
The number of negative values in S is: 2
```

In [2]:
```
# Lab 3 Exercise 1

def product_of_list(l):
    """ This function returns the product
    of the list of numbers given as input """
    product = 1

    for x in l:
        product *= x

    return product

list_of_numbers=[1,2,3,4,5]
list_of_numbers.append(product_of_list(list_of_numbers))

print(list_of_numbers)
```

```
[1, 2, 3, 4, 5, 120]
```

In [3]:
```
# Lab 3 Exercise 2

def factorial(integer):
    if not isinstance(integer, int):
        print("ERROR: The given value is not an integer.")
        return;

    if integer < 0:
        print ("ERROR: The given value is negative.")
        return;

    def inner_factorial(integer):
        product = 1
        for j in range(1,integer + 1):
            product *= j
        return product

    return inner_factorial(integer)

print("Factorial of 6 is:", factorial(6))
```

```
Factorial of 6 is: 720
```

In [8]:
```
# Lab 4 Exercise 1
# Complete the code

import numpy as np
x = np.eye(3)
print(x)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [9]:
```
# Lab 4 Exercise 2
# Complete the code

import pandas as pd
d = {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0], 'col3': [7, 5, 8, 12, 1,11]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
print("\nAfter removing first three rows of the said DataFrame:")
df1 = df.iloc[3:]
print(df1)
```

```
Original DataFrame
   col1  col2  col3
0     1     4     7
1     2     5     5
2     3     6     8
3     4     9    12
4     7     5     1
5    11     0    11

After removing first three rows of the said DataFrame:
   col1  col2  col3
3     4     9    12
4     7     5     1
5    11     0    11
```

M3: Linear Regression

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split

print('Libraries have been imported.')

Read in the dataset

data = pd.read_csv('advertising.csv')

Check the number of features and observations

data.shape
(200, 5)
Display the first 5 rows

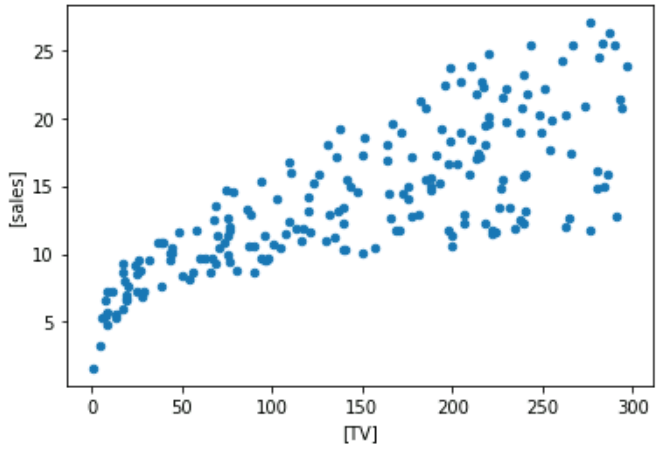
data.head()

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4

```
2        3        17.2    45.9    69.3    9.3
3        4        151.5   41.3    58.5    18.5
4        5        180.8   10.8    58.4    12.9
```

Plot a scatter plot of sales vs. TV ad spending budget

```
data.plot.scatter(x=['TV'],y=['sales'])
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e2e697d30>
```



Define the variables

```
feature_cols = ['newspaper']      # Extract the feature of interest
X = data[feature_cols]            # Select the predictor for regression
y = data['sales']                 # Select the outcome variable for regression

feature_cols2 = ['radio']
X2 = data[feature_cols2]
```

Split the data to test and training data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 101)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y, test_size = 0.3, random_state =
101)
```

test_size parameter defines the fraction of data that will be used as test data

Run a linear regression model

```
lr = LinearRegression()          # Select the estimator
lr.fit(X_train, y_train)          # Fit the model
```

```
lr2 = LinearRegression()
lr2.fit(X_train2, y_train2)

# Print the coefficients

print ("intercept : ",lr.intercept_)
print ("coefficient : ",lr.coef_)

print ("intercept2 : ",lr2.intercept_)
print ("coefficient2 : ",lr2.coef_)
intercept :  12.413083366368108
coefficient :  [0.06102883]
intercept2 :  9.494380889222633
coefficient2 :  [0.20574997]

# Import statsmodels

import statsmodels.api as sm
import statsmodels.formula.api as smf

# Estimate the simple linear regression

est = smf.ols('sales ~ newspaper', data).fit()    # Regresses sales (y) on newspaper (X)
est.summary()   # Print results
Dep. Variable:          sales    R-squared:      0.052
Model:          OLS    Adj. R-squared:         0.047
Method:         Least Squares  F-statistic:      10.89
Date:   Sun, 17 Jul 2022        Prob (F-statistic):       0.00115
Time:   00:48:24        Log-Likelihood:         -608.34
No. Observations:       200     AIC:    1221.
Df Residuals:  198      BIC:    1227.
Df Model:       1
Covariance Type:        nonrobust
             coef    std err  t         P>|t|    [0.025  0.975]
Intercept         12.3514         0.621   19.876          0.000   11.126 13.577
newspaper       0.0547          0.017   3.300   0.001   0.022   0.087
Omnibus:        6.231   Durbin-Watson:          1.983
Prob(Omnibus):          0.044   Jarque-Bera (JB):       5.483
Skew:  0.330   Prob(JB):       0.0645
Kurtosis:       2.527   Cond. No.       64.7


Notes:
```

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

```
# Perform predictions using the test data

predictions_1 = lr.predict(X_test)
predictions_1
```

```
array([15.20210069, 12.74874191, 14.76879603, 15.54996499, 12.54734678,
       13.77402617, 13.38344169, 13.21256098, 12.96234279, 14.72607585,
       13.13932639, 12.55955255, 13.21256098, 16.07481289, 17.02686256,
       13.67027717, 13.60314546, 12.86469667, 13.53601375, 14.36600578,
       13.4505734 , 14.71387008, 14.96408827, 15.75136011, 12.46800931,
       14.52468072, 12.52293525, 15.04342574, 13.48719069, 16.64238096,
       16.43488296, 15.16548339, 12.98065144, 15.4401131 , 14.77489891,
       14.24394813, 13.99372994, 12.90741685, 15.17158627, 12.91962262,
       19.37036946, 14.03645012, 14.78710467, 15.48893616, 13.77402617,
       13.52991087, 12.98675432, 15.45842175, 15.97716677, 15.64150823,
       15.16548339, 14.02424436, 13.71910023, 12.74263902, 13.70689446,
       12.76094767, 12.71822749, 12.76705055, 12.43139201, 14.33549137])
```

```
# Obtain the MSE

metrics.mean_squared_error(y_test, lr.predict(X_test))
```

```
28.43936183400341
```

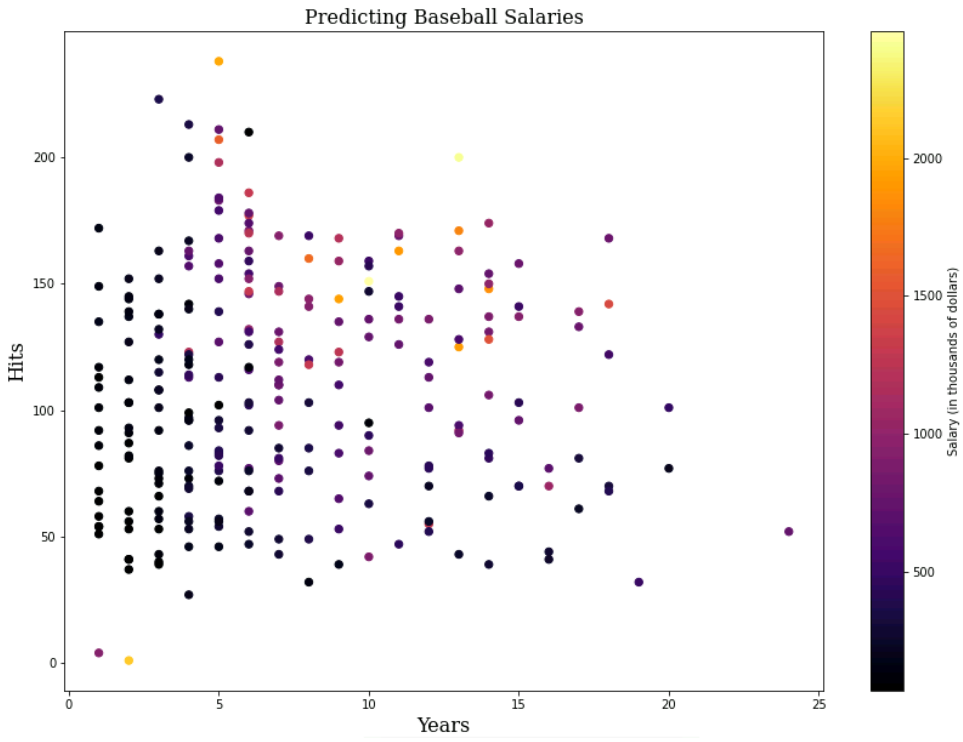
M3: Regression Trees

```
# Import relevant libraries and functions
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pydotplus
#from sklearn.externals.six import StringIO
from sklearn.tree import DecisionTreeRegressor, export_graphviz, DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split

print('Libraries have been imported.')

# Import data from Hitters.csv

df = pd.read_csv('hitters.csv', index_col=0)

# Display 10 random rows of data to inspect and ensure data is imported correctly
```

```
df.sample(n=10)

# Check the data types
df.dtypes
AtBat         int64
Hits          int64
HmRun         int64
Runs          int64
RBI           int64
Walks         int64
Years         int64
CAtBat        int64
CHits         int64
CHmRun        int64
CRuns         int64
CRBI          int64
CWalks        int64
League       object
Division     object
PutOuts       int64
Assists       int64
Errors        int64
Salary      float64
NewLeague    object
dtype: object

# Drop rows with missing data

df = df.dropna()
# Note: To replace the NaN values, use the fillna() pandas function, for example, fillna(0)
will replace all NaN values with zero.

# Specify the figure size

fig_size = plt.rcParams["figure.figsize"]
plt.rcParams["figure.figsize"] = (14,10)

# Define font characteristics which will be used later for plot title

font = {'family': 'serif',
        'color':  'k',
        'weight': 'normal',
        'size': 16,
        }
```

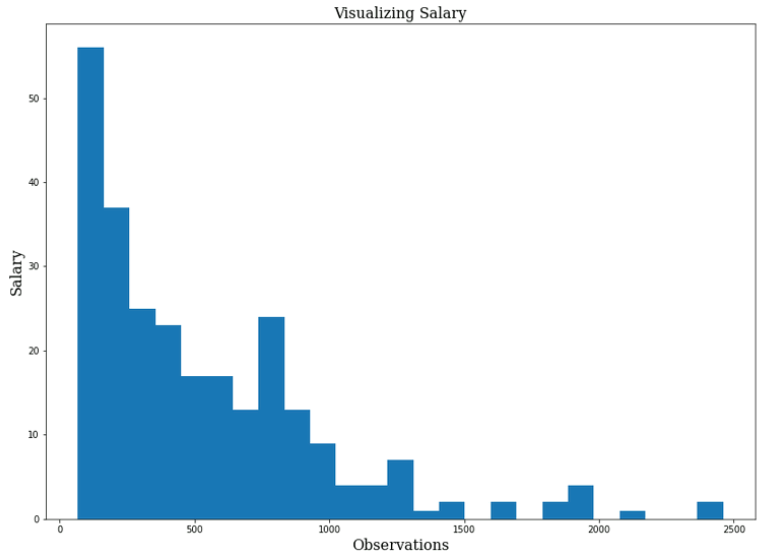
Generate a plot

```
plt.scatter(x=df['Years'], y=df['Hits'], c=df['Salary'], s=40, cmap='inferno')
plt.title("Predicting Baseball Salaries",
          fontdict=font)
plt.xlabel("Years", fontdict=font)
plt.ylabel("Hits", fontdict=font)
color_bar = plt.colorbar()
color_bar.ax.set_ylabel('Salary (in thousands of dollars)')
plt.show()
```

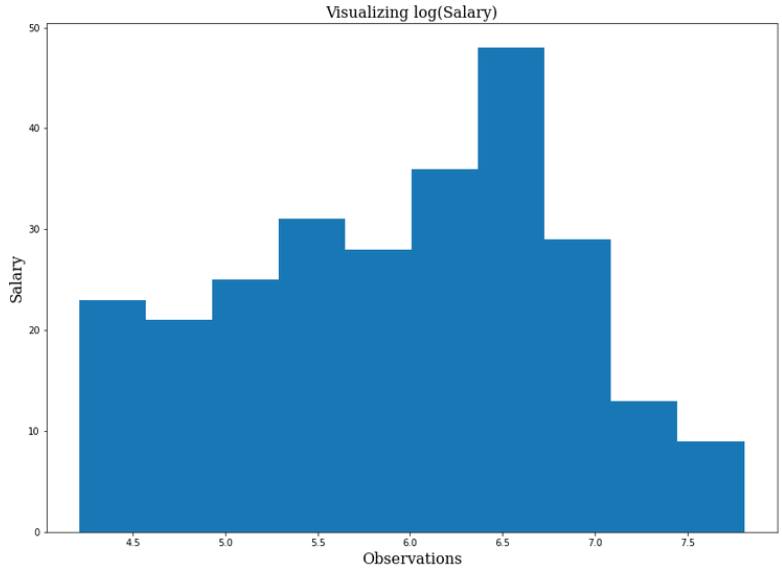


Histogram of Salary with 25 bins

```
plt.hist(df['Salary'],bins=25)
plt.title("Visualizing Salary",
          fontdict=font)
plt.xlabel("Observations", fontdict=font)
plt.ylabel("Salary", fontdict=font)

plt.show()
```




```
# Histogram of log(Salary)

plt.hist(np.log(df['Salary']))

plt.title("Visualizing log(Salary)",
          fontdict=font)
plt.xlabel("Observations", fontdict=font)
plt.ylabel("Salary", fontdict=font)

plt.show()
```




```
X = df[['Years', 'Hits']]
y = np.log(df['Salary'])

# Split the data to test and training data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 101)

# test_size parameter defines the fraction of data that will be used as test data

# Regression tree on the training set

tree_1 = DecisionTreeRegressor(random_state=0)
tree_1.fit(X_train, y_train)
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
# Perform the prediction using the test set

predictions_1=tree_1.predict(X_test)

# Compute the MSE

metrics.mean_squared_error(y_test, predictions_1)
0.5169494587029303

# Regression tree on training set using a defined leaf node value

tree_2 = DecisionTreeRegressor(max_leaf_nodes=4, random_state=0)
tree_2.fit(X_train, y_train)

# Perform the prediction using the test set

predictions_2=tree_2.predict(X_test)

# Compute the MSE

metrics.mean_squared_error(y_test, predictions_2)
0.41629010874269995
# Notice that the MSE value is now around 0.43, which means model tree_2 performs better
than model tree_1

dot_data = export_graphviz(tree_2,
                           feature_names=['Years', 'Hits'],
                           out_file=None,
                           filled=True,
                           rounded=True,
```

```
                    special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
nodes = graph.get_node_list()

# Save the plot as a png file in the current notebook folder

graph.write_png('python_decision_tree.png')
```

M3: Logistic Regression

```
# Import relevant libraries

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import (confusion_matrix, plot_confusion_matrix, classification_report)
from sklearn.linear_model import LogisticRegression

print('Libraries have been imported.')

# Import the breastcancer.csv dataset

cancer = pd.read_csv('breastcancer.csv')

# Display last 10 rows

cancer.tail(10)
```

	id	name	radius	texture	perimeter	area	smoothness	compactness	concavity	symmetry	fractal_dimension	age	diagnosis
559	ID925291	Aimee Rioux	11.51	23.93	74.52	403.5	0.09261	0.10210	0.11120	0.1388	0.06570	39	0
560	ID925292	Natala Sheppard	14.05	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.1537	0.06171	45	0
561	ID925311	Leanora Arrizubieta	11.20	29.37	70.67	386.0	0.07449	0.03558	0.00000	0.1060	0.05502	47	0
562	ID925622	Kattie Lima	NaN	30.62	103.40		716.9	0.10480	0.20870	0.25500	0.2128	0.07152	51	1
563	ID926125	Sheelagh Gjurasic	20.92	25.09	143.00		1347.0	0.10990	0.22360	0.31740	0.2149	0.06879	59	1

```
564     ID926424        Liz Babb        21.56   22.39   142.00          1479.0          0.11100
        0.11590         0.24390         0.1726          0.05623         18      1
565     ID926682        Silvana Shen    20.13   28.25   131.20          1261.0          0.09780
        0.10340         0.14400         0.1752          0.05533         24      1
566     ID926954        Sunny Hiorns    16.60   28.08   108.30          858.1   0.08455
0.10230         0.09251         0.1590          0.05648         21      1
567     ID927241        Colline Beade   20.60   29.33   140.10          1265.0          0.11780
        0.27700         0.35140         0.2397          0.07016         41      1
568     ID92751         Rita Ryan blanco        7.76    24.54   47.92   181.0   0.05263
0.04362         0.00000         0.1587          0.05884         37      0
```

Display a summary of statistics

cancer.describe().T

```
                count   mean    std     min     25%     50%     75%     max
radius          498.0   14.326635       3.506881        7.76000         11.81750        13.46500
16.15500        28.11000
texture         569.0   19.289649       4.301036        9.71000         16.17000        18.84000
        21.80000        39.28000
perimeter       569.0   91.969033       24.298981       43.79000        75.17000        86.24000
        104.10000       188.50000
area    569.0   654.889104      351.914129      143.50000       420.30000       551.10000
782.70000       2501.00000
smoothness      569.0   0.096360        0.014064        0.05263         0.08637         0.09587
        0.10530         0.16340
compactness     569.0   0.104341        0.052813        0.01938         0.06492         0.09263
        0.13040         0.34540
concavity       569.0   0.088799        0.079720        0.00000         0.02956         0.06154
        0.13070         0.42680
symmetry        569.0   0.181162        0.027414        0.10600         0.16190         0.17920
        0.19570         0.30400
fractal_dimension       569.0   0.062798        0.007060        0.04996         0.05770
0.06154         0.06612         0.09744
age     569.0   39.467487       13.604683       16.00000        27.00000        40.00000
52.00000        62.00000
diagnosis       569.0   0.372583        0.483918        0.00000         0.00000         0.00000
        1.00000         1.00000
```

Inspect the various columns (features) in the dataset

cancer.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
```

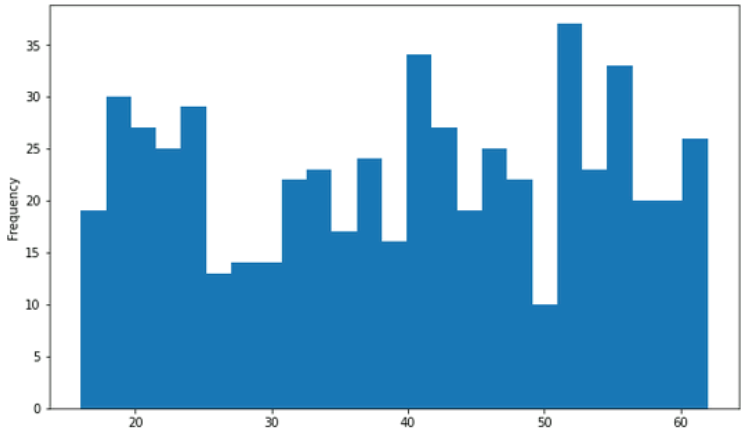
```
---  ------           --------------  -----
 0   id               569 non-null    object
 1   name             569 non-null    object
 2   radius           498 non-null    float64
 3   texture          569 non-null    float64
 4   perimeter        569 non-null    float64
 5   area             569 non-null    float64
 6   smoothness       569 non-null    float64
 7   compactness      569 non-null    float64
 8   concavity        569 non-null    float64
 9   symmetry         569 non-null    float64
 10  fractal_dimension  569 non-null  float64
 11  age              569 non-null    int64
 12  diagnosis        569 non-null    int64
dtypes: float64(9), int64(2), object(2)
memory usage: 57.9+ KB
```

```
# Drop id and name columns

cancer.drop(['id', 'name'], axis = 1, inplace = True)
# axis informs whether to drop labels from rows (0) or columns (1).
# if inplace is set to False, it returns a copy, otherwise it performs the operation
# in place.

# View head of data to confirm columns are dropped

cancer.head()
```

	radius	texture	perimeter	area	smoothness	compactness	concavity
	symmetry	fractal_dimension	age	diagnosis			
0	NaN	10.38	122.80	1001.0	0.11840	0.27760	0.3001
	0.2419	0.07871	35	1			
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
	0.1812	0.05667	27	1			
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
	0.2069	0.05999	31	1			
3	NaN	20.38	77.58	386.1	0.14250	0.28390	0.2414
	0.09744	49	1				
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980
	0.1809	0.05883	20	1			

```
# Visualize the ages of the individuals using a histogram

cancer['age'].plot.hist(bins = 25, figsize = (10,6))
```

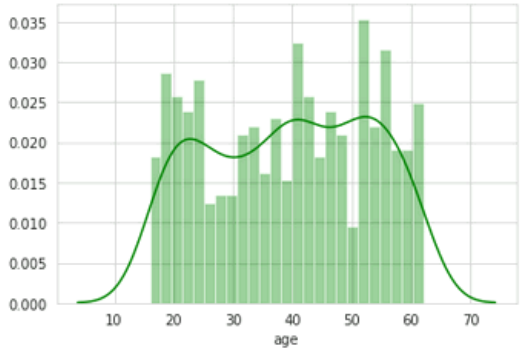
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c23e01fd0>
```



Visualize the ages of the individuals using a Seaborn distribution plot

```
sns.set_style(style='whitegrid')
sns.distplot(cancer['age'], color = 'green', bins = 25)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c21dad5f8>
```



Drop rows having NaN values

```
cancer = cancer.dropna()
```

Make sure NaN values were removed

```
cancer.isna().any()
radius              False
texture             False
perimeter           False
area                False
smoothness          False
compactness         False
concavity           False
symmetry            False
fractal_dimension   False
age                 False
diagnosis           False
```

```
dtype: bool
```

```
# Define the variables

X = cancer.drop('diagnosis', axis = 1)
y = cancer['diagnosis']

# Create test and training data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 101)

# test_size parameter defines the fraction of data that will be used as test data
# random_state initializes the pseudo-random number generator, allowing the splits to be
randomly generated

# Fit the model using the training data

logreg = LogisticRegression(solver='liblinear', max_iter=1000)
logreg.fit(X_train, y_train)

# Obtain the predictions for the test data

y_pred = logreg.predict(X_test)

# Inspect the predictions

y_pred, y_pred.shape
(array([0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]),
 (150,))

# Inspect the actual data

y_test.shape, X_test.shape
((150,), (150, 10))

# Combine the actual dataset target with the predictions
```

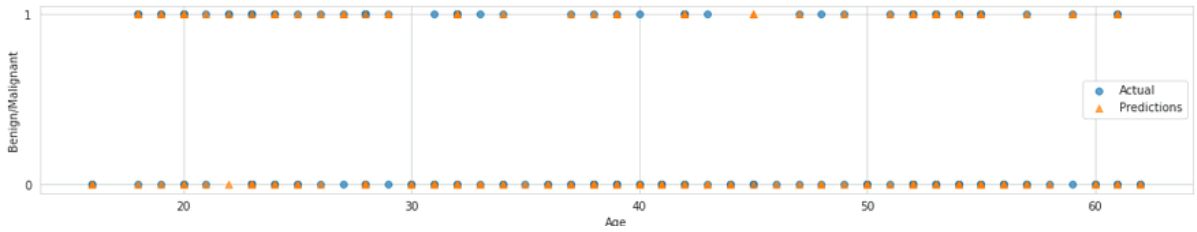
```
cancer_test_data = pd.DataFrame({'Actual':y_test, 'Predictions':y_pred})

# Inspect the dataframe

cancer_test_data.sample(n=10)
```

	Actual	Predictions
267	0	0
373	1	1
270	0	0
187	0	0
22	1	0
548	0	0
215	1	0
565	1	1
168	1	1
398	0	0

```
X_test.head()
```

	radius	texture	perimeter	area	smoothness	compactness	concavity
	symmetry	fractal_dimension	age				
179	12.81	13.06	81.29	508.8	0.08739	0.03774	0.009193	0.1466
	0.06133	52					
29	17.57	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.1739
	0.06149	38					
516	18.31	20.58	120.80	1052.0	0.10680	0.12480	0.156900	
	0.1860	0.05941	54				
411	11.04	16.83	70.92	373.2	0.10770	0.07804	0.030460	0.1714
	0.06340	32					
503	23.09	19.83	152.10	1682.0	0.09342	0.12750	0.167600	
	0.1505	0.05484	42				

```
# Plot the predictions with the actual target

plt.figure(figsize=(18,3))
plt.scatter(X_test['age'],cancer_test_data['Actual'],label='Actual', alpha=0.7)
plt.scatter(X_test['age'],cancer_test_data['Predictions'],label='Predictions',
marker='^',alpha=0.7)
plt.legend(loc=7)
plt.yticks([0,1])
plt.xlabel('Age')
plt.ylabel('Benign/Malignant')
plt.show()
```



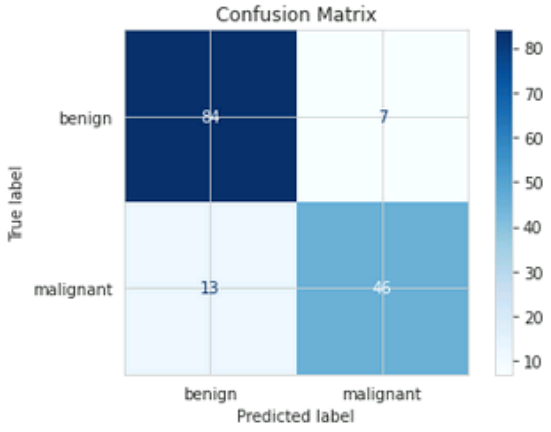
Obtain the confusion matrix

```
conf = confusion_matrix(y_test, y_pred)
print(conf)
```

```
[[84  7]
 [13 46]]
```

Plot the confusion matrix

```
disp = plot_confusion_matrix(
logreg,
X_test,
y_test,
display_labels=["benign","malignant"],
cmap = plt.cm.Blues)
disp.ax_.set_title("Confusion Matrix")

plt.show()
```



Obtain the accuracy score

```
accuracy = (conf[0,0]+conf[1,1])/(conf[0,0]+conf[0,1]+conf[1,0]+conf[1,1])
print(accuracy)
```

```
0.8666666666666667
```

```
logreg_s = logreg.score(X_test, y_test)
print("Accuracy:", int(logreg_s *100), "%")
```

Accuracy: 86 %

```
# Obtain the classification report

print(classification_report(y_test, y_pred, labels=[0,1], target_names=['benign','malignant']))
```

```
              precision    recall  f1-score   support

      benign       0.87      0.92      0.89        91
   malignant       0.87      0.78      0.82        59

    accuracy                           0.87       150
   macro avg       0.87      0.85      0.86       150
weighted avg       0.87      0.87      0.87       150
```

M3: Support Vector Machines

```
# Import relevant libraries

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (confusion_matrix, plot_confusion_matrix, classification_report)

print('Libraries have been imported.')

# Import the breastcancer.csv dataset

cancer = pd.read_csv('breastcancer.csv')

# Display first 5 rows

cancer.head()
```

	id	name	radius	texture	perimeter	area	smoothness	compactness	concavity	symmetry	fractal_dimension	age	diagnosis
0	ID842302	Glynnis Munson	NaN	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.2419	0.07871	35	1
1	ID842517	Lana Behrer	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.1812	0.05667	27	1
2	ID84300903	Devondra Vanvalkenburgh	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.2069	0.05999	31	1
3	ID84348301	Glory Maravalle	NaN	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.2597	0.09744	49	1

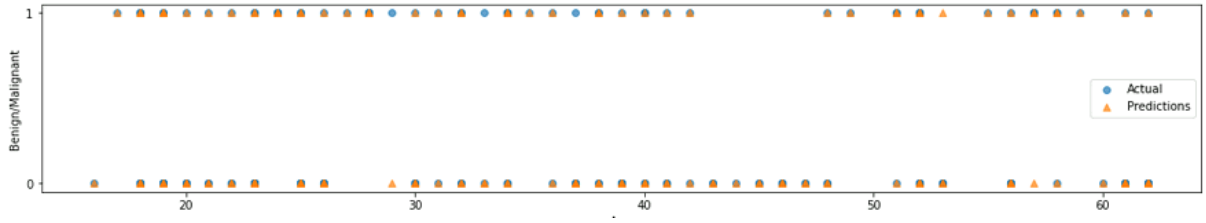
```
4       ID84358402   Mellie Mccurdy        20.29   14.34   135.10          1297.0
0.10030         0.13280         0.1980          0.1809          0.05883         20      1
```

```
# Drop id and name columns

cancer.drop(['id', 'name'], axis = 1, inplace = True)

# Drop rows having NaN values

cancer = cancer.dropna()

X = cancer.drop('diagnosis', axis = 1)
y = cancer['diagnosis']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 109)
# note if you change the random_state value, you may get slightly different results

# Import svm model
from sklearn import svm

# Create a svm Classifier
svm_model = svm.SVC(kernel='linear') # Linear Kernel

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

# Predict the response for test dataset
y_pred = svm_model.predict(X_test)

# Combine the actual dataset target with the predictions

cancer_test_data = pd.DataFrame({'Actual':y_test, 'Predictions':y_pred})

# Inspect the dataframe

cancer_test_data.sample(n=10)
```

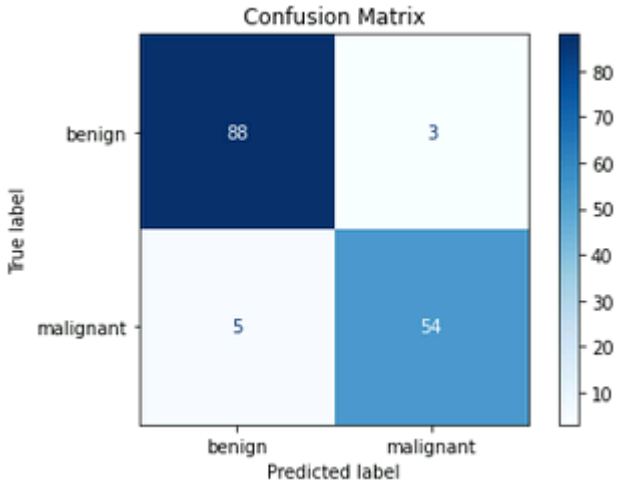
```
        Actual Predictions
70      1       1
561     0       0
97      0       0
398     0       0
292     0       0
```

```
488    0       0
418    0       0
63     0       0
174    0       0
393    1       1
```

```
# Plot the predictions with the actual target

plt.figure(figsize=(18,3))
plt.scatter(X_test['age'],cancer_test_data['Actual'],label='Actual',alpha =0.7)
plt.scatter(X_test['age'],cancer_test_data['Predictions'],label='Predictions', marker='^',alpha
=0.7)
plt.legend(loc=7)
plt.yticks([0,1])
plt.xlabel('Age')
plt.ylabel('Benign/Malignant')
plt.show()
```




```
# Obtain the confusion matrix

conf = confusion_matrix(y_test, y_pred)
print(conf)
[[88  3]
 [ 5 54]]

# Plot the confusion matrix

from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(svm_model,
                             X_test,
                             y_test,
                             display_labels=["benign","malignant"],
                             cmap=plt.cm.Blues)
disp.ax_.set_title("Confusion Matrix")

plt.show()
```




```
accuracy = (conf[0,0]+conf[1,1])/(conf[0,0]+conf[0,1]+conf[1,0]+conf[1,1])
print(accuracy)
print("\nAccuracy:", int(accuracy*100), "%")
```

```
0.9466666666666667

Accuracy: 94 %
```

```
# Obtain the accuracy score

svm_model_s = svm_model.score(X_test, y_test)
print("Accuracy:", int(svm_model_s *100), "%")
```

```
Accuracy: 94 %
```

```
# Obtain the classification report

print(classification_report(y_test, y_pred, labels=[0,1], target_names=['benign','malignant']))
```

```
              precision    recall  f1-score   support

      benign       0.95      0.97      0.96        91
   malignant       0.95      0.92      0.93        59

    accuracy                           0.95       150
   macro avg       0.95      0.94      0.94       150
weighted avg       0.95      0.95      0.95       150
```

```
## Convert age to integer type
# print(records.dtypes) # checking type before
# records['age']=records['age'].astype('int64')
#
```

M3: Naive Bayes

```
# Import libraries

import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import metrics
from sklearn.metrics import (confusion_matrix, plot_confusion_matrix, classification_report)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB   # Import Gaussian Naive Bayes model

# Load dataset

wine = datasets.load_wine()

# Inspect the dataset

wine
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]),
 'target_names': array(['class_0', 'class_1', 'class_2'], dtype='<U7'),
```

'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n------------------------\n\n**Data Set Characteristics:**\n\n   :Number of Instances: 178 (50 in each of three classes)\n   :Number of Attributes: 13 numeric, predictive attributes and the class\n   :Attribute Information:\n \t\t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of ash \n \t\t- Magnesium\n\t\t- Total phenols\n \t\t- Flavanoids\n \t\t- Nonflavanoid phenols\n \t\t- Proanthocyanins\n\t\t- Color intensity\n \t\t- Hue\n \t\t- OD280/OD315 of diluted wines\n \t\t- Proline\n\n   - class:\n - class_0\n   - class_1\n   - class_2\n\t\t\n   :Summary Statistics:\n   \n ============================== ==== ===== ======= =====\n Min   Max   Mean   SD\n ============================== ==== ===== ======= =====\n   Alcohol: 11.0 14.8   13.0   0.8\n   Malic Acid: 0.74 5.80   2.34 1.12\n   Ash: 1.36 3.23   2.36 0.27\n   Alcalinity of Ash: 10.6 30.0   19.5   3.3\n   Magnesium: 70.0 162.0   99.7 14.3\n   Total Phenols: 0.98 3.88   2.29 0.63\n   Flavanoids: 0.34 5.08   2.03 1.00\n   Nonflavanoid Phenols: 0.13 0.66   0.36 0.12\n   Proanthocyanins: 0.41 3.58   1.59 0.57\n   Colour Intensity: 1.3 13.0   5.1   2.3\n   Hue: 0.48 1.71   0.96 0.23\n   OD280/OD315 of diluted wines: 1.27 4.00   2.61 0.71\n   Proline: 278 1680   746   315\n ============================== ==== ===== ======= =====\n\n   :Missing Attribute Values: None\n   :Class Distribution: class_0 (59), class_1 (71), class_2 (48)\n   :Creator: R.A. Fisher\n   :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n   :Date: July, 1988\n\nThis is a copy of UCI ML Wine recognition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data\n\nThe data is the results of a chemical analysis of wines grown in the same\nregion in Italy by three different cultivators. There are thirteen different\nmeasurements taken for different constituents found in the three types of\nwine.\n\nOriginal Owners: \n\nForina, M. et al, PARVUS - \nAn Extendible Package for Data Exploration, Classification and Correlation. \nInstitute of Pharmaceutical and Food Analysis and Technologies,\nVia Brigata Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013). UCI Machine Learning Repository\n[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,\nSchool of Information and Computer Science. \n\n.. topic:: References\n\n (1) S. Aeberhard, D. Coomans and O. de Vel, \n Comparison of Classifiers in High Dimensional Settings, \n Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of \n Mathematics and Statistics, James Cook University of North Queensland. \n (Also submitted to Technometrics). \n\n The data was used with many others for comparing various \n classifiers. The classes are separable, though only RDA \n has achieved 100% correct classification. \n (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data)) \n (All results using the leave-one-out technique) \n\n (2) S. Aeberhard, D. Coomans and O. de Vel, \n "THE CLASSIFICATION PERFORMANCE OF RDA" \n Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of \n Mathematics and Statistics, James Cook University of North Queensland. \n (Also submitted to Journal of Chemometrics).\n',
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',

```
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']}

# Print the names of the features

print("Features: ", wine.feature_names)

# Print the label type of wine(class_0, class_1, class_2)

print("Labels: ", wine.target_names)
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols',
'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']
Labels:  ['class_0' 'class_1' 'class_2']

# Print data(feature)shape

wine.data.shape
(178, 13)

# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target,
test_size=0.3,random_state=109) # 70% training and 30% test

# Create a Gaussian Classifier
gnb_model = GaussianNB()

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

# Predict the response for test dataset
y_pred = gnb_model.predict(X_test)

# Combine the actual dataset target with the predictions

wine_test_data = pd.DataFrame({'Actual':y_test, 'Predictions':y_pred})
```

Inspect the dataframe

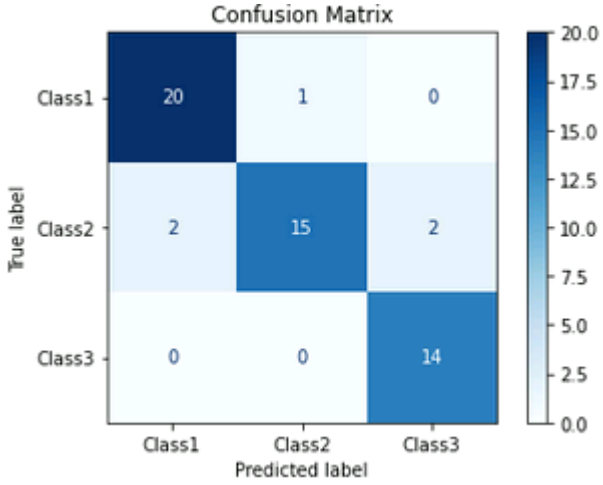
```
wine_test_data.head(10)
```

	Actual	Predictions
0	0	0
1	0	0
2	1	1
3	2	2
4	0	0
5	1	1
6	0	0
7	1	0
8	1	1
9	0	0

```
conf = confusion_matrix(y_test, y_pred)
print(conf)
[[20  1  0]
 [ 2 15  2]
 [ 0  0 14]]
```

Plot the confusion matrix

```
disp = plot_confusion_matrix(gnb_model,X_test, y_test,
display_labels=["Class1", "Class2", "Class3"],
cmap = plt.cm.Blues
)
disp.ax_.set_title("Confusion Matrix")

plt.show()
```



Obtain model accuracy

```
metrics.accuracy_score(y_test, y_pred)
0.9074074074074074

s = gnb_model.score (X_test,y_test)
print("Accuracy:", int(s*100), "%")
Accuracy: 90 %

# Obtain the classification report

print(classification_report(y_test, y_pred, labels=[0,1,2],
target_names=['Class1','Class2','Class3']))
              precision    recall  f1-score   support

      Class1       0.91      0.95      0.93        21
      Class2       0.94      0.79      0.86        19
      Class3       0.88      1.00      0.93        14

    accuracy                           0.91        54
   macro avg       0.91      0.91      0.91        54
weighted avg       0.91      0.91      0.91        54
```

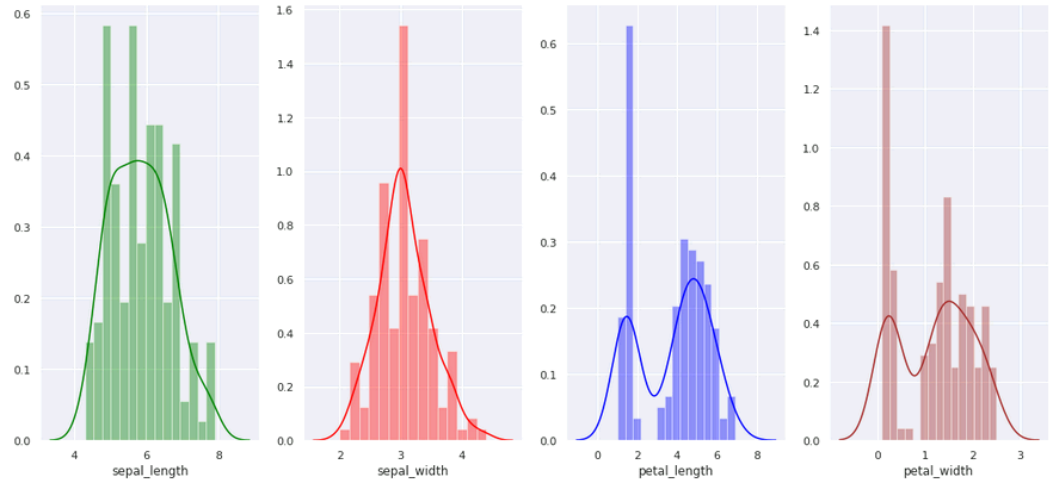
M4: K-Means Clustering

```
# Import all relevant libraries

%matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Import the dataset

df = pd.read_csv('iris.csv')

# Display ten random entries of the dataset

df.sample(n=10)
     sepal.length  sepal.width  petal.length  petal.width     variety
128           6.4          2.8           5.6          2.1   Virginica
36            5.5          3.5           1.3          0.2      Setosa
```

```
92     5.8     2.6     4.0     1.2     Versicolor
16     5.4     3.9     1.3     0.4     Setosa
123    6.3     2.7     4.9     1.8     Virginica
25     5.0     3.0     1.6     0.2     Setosa
35     5.0     3.2     1.2     0.2     Setosa
69     5.6     2.5     3.9     1.1     Versicolor
59     5.2     2.7     3.9     1.4     Versicolor
```

```
# Rename the column by replacing the dot by an underscore

df= df.rename({c:c.replace('.', '_') for c in df.columns}, axis=1) # Using a lambda function

# Use describe function to show no outliers, no negative values - data looks good
# Display the number of occurrences of each target value

df['variety'].value_counts()
Setosa       50
Virginica    50
Versicolor   50
Name: variety, dtype: int64
# The number of occurrences of target values is the same, this indicates a balanced dataset.

# Plot the distribution of features

sns.set(rc={'figure.figsize':(18,8)})
fig, axs = plt.subplots(ncols=4)  # plot four plots in one figure
sns.set_style(style='whitegrid')

sns.distplot(df['sepal_length'], color = 'green', bins = 15,ax=axs[0])
sns.distplot(df['sepal_width'], color = 'red', bins = 15,ax=axs[1])
sns.distplot(df['petal_length'], color = 'blue', bins = 15,ax=axs[2])
sns.distplot(df['petal_width'], color = 'brown', bins = 15,ax=axs[3])
```

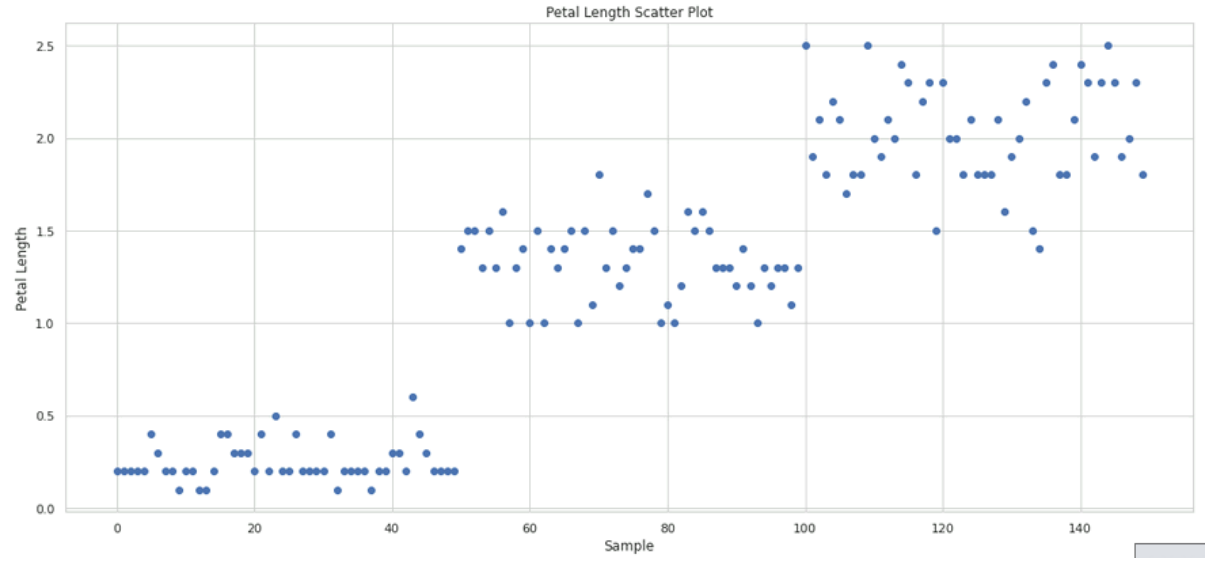
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb6f480898>
```



Plot the distribution of petal width

```
plt.scatter(df.index, df.petal_width)
plt.xlabel('Sample')
plt.ylabel('Petal Length')
plt.title('Petal Length Scatter Plot')
```

```
Text(0.5, 1.0, 'Petal Length Scatter Plot')
```



Extract the four features indexed by the columns (0,1,2,3)

```
X = df.iloc[:, [0,1,2,3]].values
# We can also use pandas drop() function, X= df.drop(columns=['variety'])

kmeans3 = KMeans(n_clusters=3)      # K-Means with 3 clusters
y_pred = kmeans3.fit_predict(X)  # Calling K-Means with 3 clusters on the data
```

```
print(y_pred)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]

# Add the predictions to the dataframe for comparison

df['precited_label'] = y_pred
df.sample(n=10)

# Allow for changing display of number of columns and rows displayed in Jupyter
# notebooks. If the number of rows is set too low, it will lead to a truncated
# display of the data.

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 200)
print(df)

# Compute the mean of features per cluster (flower species)

group_class = df.groupby('precited_label')  # groups dataframe by the predicted clusters

group_class.mean() # Compute mean for dataframe columns
```

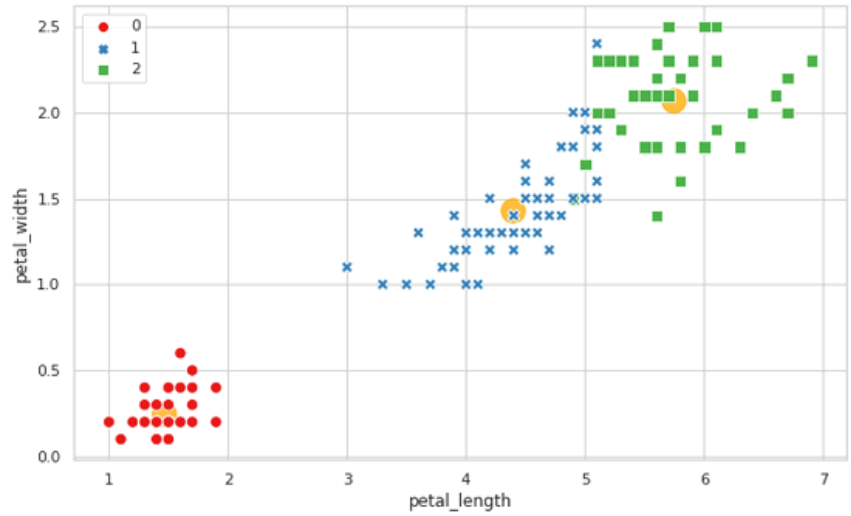
	sepal_length	sepal_width	petal_length	petal_width
precited_label				
0	5.006000	3.428000	1.462000	0.246000
1	5.901613	2.748387	4.393548	1.433871
2	6.850000	3.073684	5.742105	2.071053

```
# Plot the predicted clusters based on petal length

plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x=df.index, y=df.petal_length, hue= y_pred, palette = 'Set1',
style=y_pred, s=70)

plt.xlabel('Sample')
plt.ylabel('Petal Length')
plt.title('Predited Clusters')
```

```
Text(0.5, 1.0, 'Predited Clusters')
```


```
kmeans3.cluster_centers_ # cluster_centers_ contains the value of centroids
array([[5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])

# Plot the predicted clusters with the centroids

plt.figure(figsize=(10,6))

# Plot the centroids
plt.scatter(x = kmeans3.cluster_centers_[: , 2] , # cluster_centers_ contains the value of centroids
            y = kmeans3.cluster_centers_[: , 3] ,
            s = 300 ,      # Define marker size
            c = 'orange' ,
            alpha = 0.75)

# Plot the clusters
sns.scatterplot(data=df, x="petal_length", y="petal_width", hue= y_pred, palette = 'Set1', style=y_pred, s=70)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb6ec57390>
```



M4: K-Means Clustering for Image Compression

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()  # Plotting library
import numpy as np
from sklearn.cluster import MiniBatchKMeans
from sklearn.datasets import load_sample_image # datasets contains sample images in the Sklearn library

# Load a sample image flower.jpg

flower = load_sample_image("flower.jpg")
flower.shape
```

(427, 640, 3)

```
# Plot the image using Matplotlib

ax = plt.axes(xticks=[], yticks=[])
ax.imshow(flower);
```

View portion of the dataset

flower

```
array([[[ 2, 19, 13],
        [ 3, 18, 13],
        [ 7, 20, 13],
        ...,
        [ 1, 77, 64],
        [ 0, 76, 64],
        [ 0, 75, 63]],

       [[ 1, 18, 12],
        [ 3, 18, 13],
        [ 7, 20, 13],
        ...,
        [ 0, 76, 64],
        [ 1, 74, 65],
        [ 1, 74, 65]],

       [[ 2, 17, 12],
        [ 6, 19, 12],
        [ 7, 20, 13],
        ...,
        [ 1, 74, 65],
        [ 1, 74, 67],
        [ 1, 74, 67]],

       ...,

       [[ 0, 46, 40],
        [ 1, 48, 40],
        [ 1, 47, 37],
        ...,
        [ 5, 44, 26],
        [ 6, 43, 26],
        [ 7, 44, 27]],

       [[ 0, 47, 41],
        [ 1, 48, 40],
        [ 1, 47, 37],
        ...,
        [ 6, 45, 27],
        [ 7, 44, 27],
```

```
        [ 7, 44, 27]],

       [[ 0, 47, 41],
        [ 1, 48, 40],
        [ 0, 46, 36],
        ...,
        [ 7, 46, 28],
        [ 8, 45, 28],
        [ 9, 43, 27]]], dtype=uint8)
```

```
data = flower / 255.0              # Convert RGB intensity to a [0,1] scale
```

```
# The image is stored in a 3-D array of height, width, RGB intensity. We will
# reshape it into a 2-D array indexed by the pixel number and the RGB intensity.
```

```
data = data.reshape(427 * 640, 3)
data.shape
(273280, 3)
```

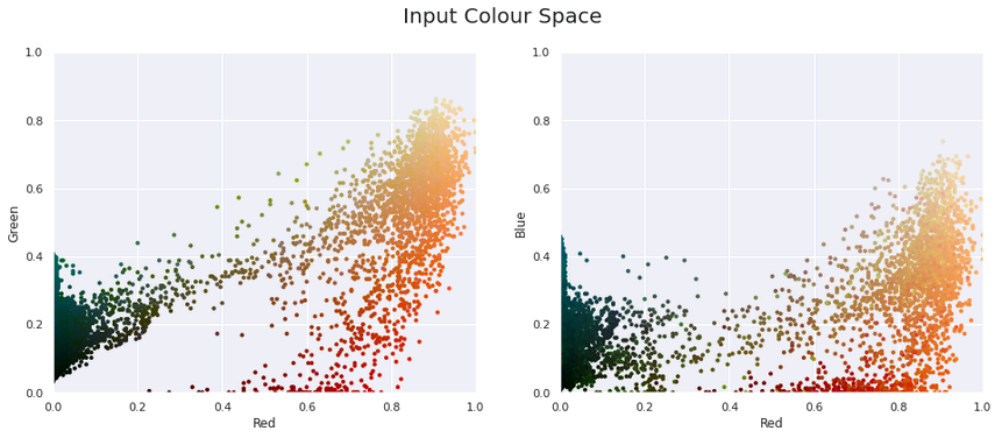
```
# View the reshaped dataset
```

```
data
array([[0.00784314, 0.0745098 , 0.05098039],
       [0.01176471, 0.07058824, 0.05098039],
       [0.02745098, 0.07843137, 0.05098039],
       ...,
       [0.02745098, 0.18039216, 0.10980392],
       [0.03137255, 0.17647059, 0.10980392],
       [0.03529412, 0.16862745, 0.10588235]])
```

```
# Define a plot_pixels function
```

```
def plot_pixels(data, title, colours=None, N=10000):
    if colours is None:
        colours = data # Save a copy of this sequence to be inputted the scatter plot

    # Choose a random subset of pixels

    rng = np.random.RandomState(0)
    i = rng.permutation(data.shape[0])[:N] # Return a randomly permuted sequence of N
values range;data.shape[0]=number of rows)
    colours = colours[i]  # Return an array of a 10,000 rows each with three colours intesnities
(10000x3 array)
    R, G, B = data[i].T # Transfom the dataset to the required format (3x10000 array)
```

```
    # data[i] will be an array of 3 arrays each has 10,000 columns
    # each of R, G, and B will be one of these arrays.


    # Define the axes and markers on the figure

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))      # Define figure size
    ax[0].scatter(R, G, color=colours, marker='.')  # Scatter plot of R as x and G as y intensities
    ax[0].set(xlabel='Red', ylabel='Green', xlim=(0, 1), ylim=(0, 1))  # Set x & y-axis labels
and limits

    ax[1].scatter(R, B, color=colours, marker='.')  # Scatter plot of R and B intensities
    ax[1].set(xlabel='Red', ylabel='Blue', xlim=(0, 1), ylim=(0, 1))

    fig.suptitle(title, size=20);

plot_pixels(data, title='Input Colour Space')
```



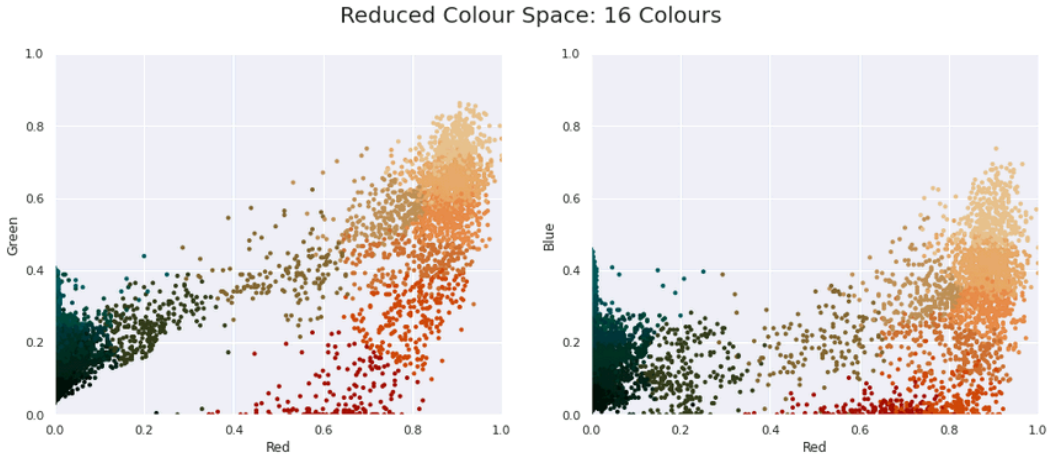

```
# Run the minibatchkmeans function from Sklearn

# Run K-Means with K = 16

kmeans = MiniBatchKMeans(16)
kmeans.fit(data)    # Fit the model
predictions = kmeans.predict(data)  # Return 16 clusters
new_colours = kmeans.cluster_centers_[predictions] # Return compressed colour intensities

print("shape",new_colours.shape)

print("\nnew_colours:\n",new_colours)
print("\n original_colours:\n",data)
```

shape (273280, 3)

```
new_colours:
 [[0.0115298  0.06949825 0.04708946]
 [0.0115298  0.06949825 0.04708946]
 [0.0115298  0.06949825 0.04708946]
 ...
 [0.02288888 0.18947147 0.1531567 ]
 [0.03161677 0.1341272  0.09769261]
 [0.03161677 0.1341272  0.09769261]]

 original_colours:
 [[0.00784314 0.0745098  0.05098039]
 [0.01176471 0.07058824 0.05098039]
 [0.02745098 0.07843137 0.05098039]
 ...
 [0.02745098 0.18039216 0.10980392]
 [0.03137255 0.17647059 0.10980392]
 [0.03529412 0.16862745 0.10588235]]
```

Plot the reduced colour space using the 16 colours

```
plot_pixels(data, colours=new_colours,
            title="Reduced Colour Space: 16 Colours")
```



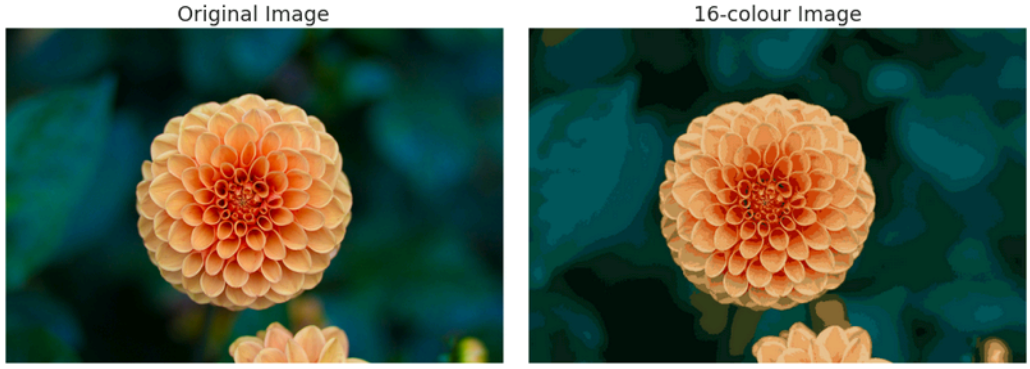
Display the size of the original image

```
flower.shape
(427, 640, 3)
```

Define the recoloured image

```
o_colour = flower.shape
```

```
flower_recoloured = new_colours.reshape(o_colour) # Define the resized image and convert
it to 3-D array

# Define the figure settings, including number of plots

fig, ax = plt.subplots(1, 2, figsize=(16, 6),  # 1: 1 rows; 2: 2 columns;
                       subplot_kw=dict(xticks=[], yticks=[])) # dict with keywords to be used to
create
                                                  # each subplot.
fig.subplots_adjust(wspace=0.05)  # Set space between two subplots

# Display the original flower

ax[0].imshow(flower)          #  Display the orginal flower image on first subplot
ax[0].set_title('Original Image', size=18)  # Specify the title and the font size

# Display the resized flower

ax[1].imshow(flower_recoloured)
ax[1].set_title('16-colour Image', size=18);
```

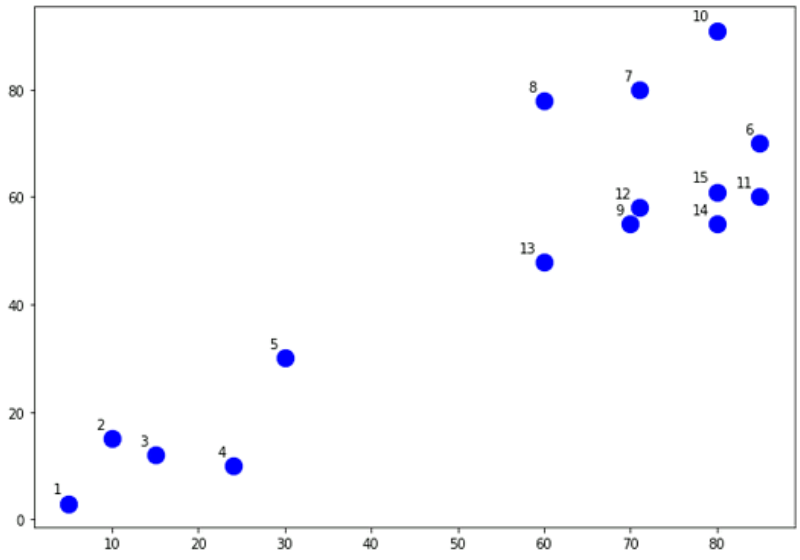



```
# Save the above figure in the current directory

fig.savefig('flower_comparison.png')
```

M4: Hierarchical Clustering

```
# Import libraries

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
# Define an array of fifteen observations, each with two feature values

X = np.array([[5,3],
    [10,15],
    [15,12],
    [24,10],
    [30,30],
    [85,70],
    [71,80],
    [60,78],
    [70,55],
    [80,91],
    [85,60],
    [71,58],
    [60,48],
    [80,55],
    [80,61],])

X.shape
(15, 2)

# Define figure characteristics

labels = range(1, 16) # Define the labels
plt.figure(figsize=(10, 7))    # figure size
plt.scatter(X[:,0],X[:,1], s = 150, label='True Position', color = 'blue')

# Label the observations from 1 to 15 then print the plot

for label, x, y in zip(labels, X[:, 0], X[:, 1]):   # zip() combines the ith element in each list
    plt.annotate(              # Annotate the point (observation) *xy* with text *text*
        label,
        xy=(x, y),        # The observation to annotate
        xytext=(-5, 5),      # Specify the position of the label wrt to the observation
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```



Explaining the previous code

```
X1=X[:, 0]
print("X1",X1)

X2=X[:, 1]
print("X2",X2)

print("labels:")
for i in labels:
    print(i)

w = zip(labels, X[:, 0], X[:, 1])
print("\nzip() Output\n",tuple(w))
```

```
X1 [ 5 10 15 24 30 85 71 60 70 80 85 71 60 80 80]
X2 [ 3 15 12 10 30 70 80 78 55 91 60 58 48 55 61]
labels:
1
2
3
4
5
6
7
8
9
10
11
12
```

13
14
15

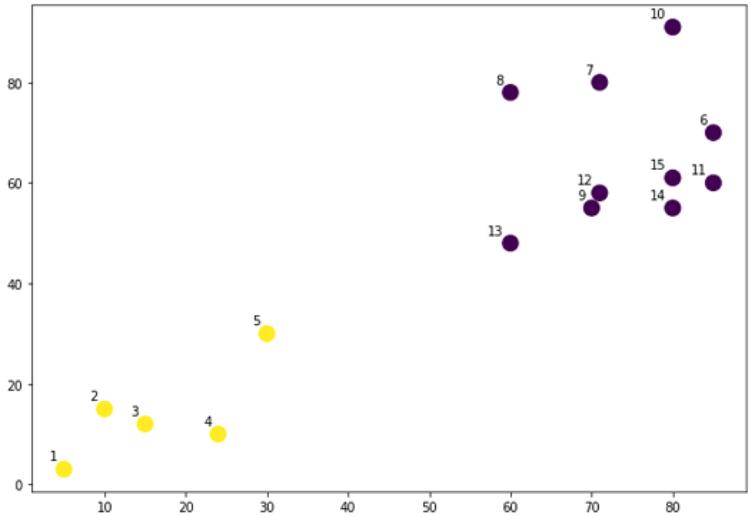
zip() Output
((1, 5, 3), (2, 10, 15), (3, 15, 12), (4, 24, 10), (5, 30, 30), (6, 85, 70), (7, 71, 80), (8, 60, 78), (9, 70, 55), (10, 80, 91), (11, 85, 60), (12, 71, 58), (13, 60, 48), (14, 80, 55), (15, 80, 61))

Fit the clustering model to the data

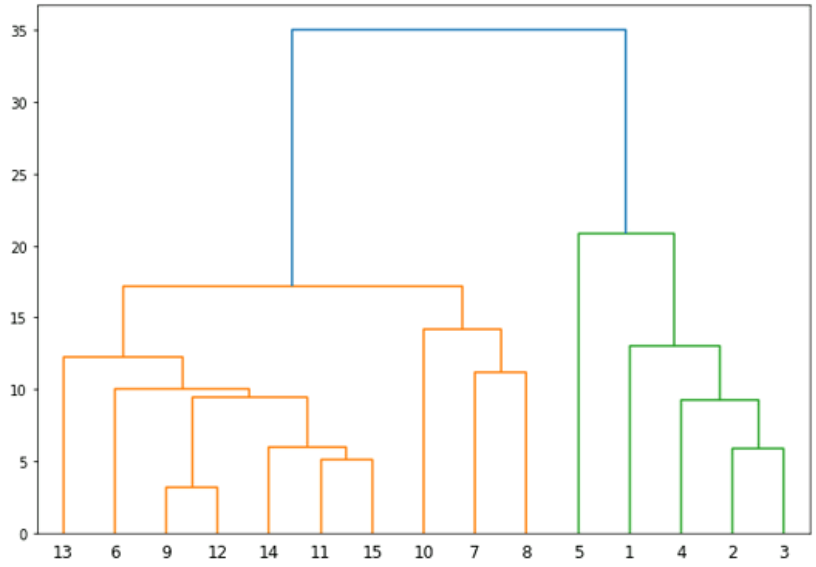
```
cluster = AgglomerativeClustering(affinity='euclidean', linkage='ward')
cluster.fit_predict(X)
# The Euclidean affinity argument tells the algorithm to compute the distance
# between two observations.
# The linkage argument tells the algorithm to merge the two nearest clusters.
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Plot the two clusters using different colors

```
plt.figure(figsize=(10, 7))
plt.scatter(X[:,0],X[:,1], s=150, c=cluster.labels_, cmap='viridis')

for label, x, y in zip(labels, X[:, 0], X[:, 1]):   # zip() combines the ith element in each list
    plt.annotate(              # Annotate the point (observation) *xy* with text *text*
        label,
        xy=(x, y),        # The observation to annotate
        xytext=(-5, 5),      # Specify the position of the label wrt to the observation
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```

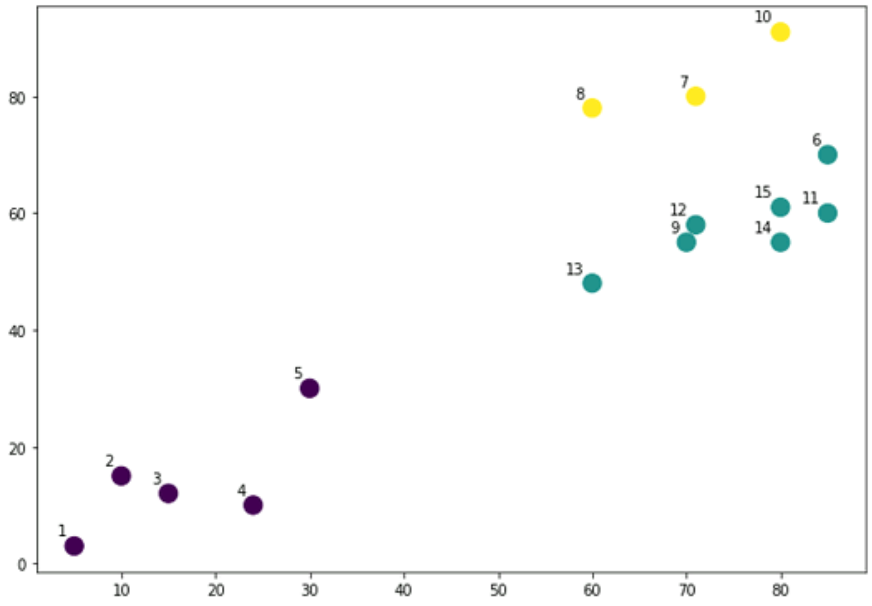


```
# Define the linkage matrix encoding the hierarchical clustering to render as a dendrogram.

linked = linkage(X, 'single')

# Label on the plot

labelList = range(1, 16) # Return a sequence of 15 numbers (1 to 15)

# Display the plot

plt.figure(figsize=(10, 7))

# Specify the orientation; whether the tree is displayed top-down or bottom-up

dendrogram(linked,
           orientation='top',
           labels=labelList,
           show_leaf_counts=True)

plt.show()
```




```
# Run the hierarchical clustering model with three clusters

cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
cluster.fit_predict(X)
# Use Shift + 4 Tabs shortcut to pull the documentation of AgglomerativeClustering()
array([0, 0, 0, 0, 0, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1])

# Plot the clusters
plt.figure(figsize=(10, 7))
```

```
plt.scatter(X[:,0],X[:,1], s=150, c=cluster.labels_, cmap='viridis')

for label, x, y in zip(labels, X[:, 0], X[:, 1]):   # zip() combines the ith element in each list
    plt.annotate(              # Annotate the point (observation) *xy* with text *text*
        label,
        xy=(x, y),        # The observation to annotate
        xytext=(-5, 5),      # Specify the position of the label wrt to the observation
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```



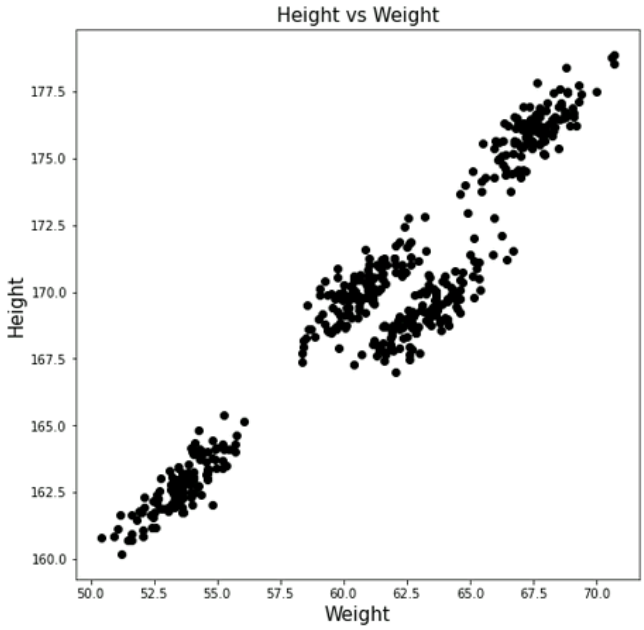
M4: Mixture Models

```
# Import libraries

import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Read in the clustering_gmm.csv dataset using pandas

data = pd.read_csv('clustering_gmm.csv')

# Inspect the head of the data to confirm data has been imported correctly

data.head()
```

	Weight	Height
0	67.062924	176.086355

```
1       68.804094    178.388669
2       60.930863    170.284496
3       59.733843    168.691992
4       65.431230    173.763679
```

```
# Generate a scatter plot of the data

plt.figure(figsize=(8,8))   # figure size
plt.scatter(data["Weight"],data["Height"], c = 'black')

plt.xlabel('Weight', fontsize = 15)
plt.ylabel('Height', fontsize = 15)

plt.title('Height vs Weight', fontsize = 15)
plt.show()
```




```
# Import KMeans algorithm and train it on the data for K = 4

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(data)

# Store the predictions from K-means

pred = kmeans.predict(data)
print("Predictions:\n",pred)


data['Cluster'] = pred   # Add a new column having the predictions to the dataset
```

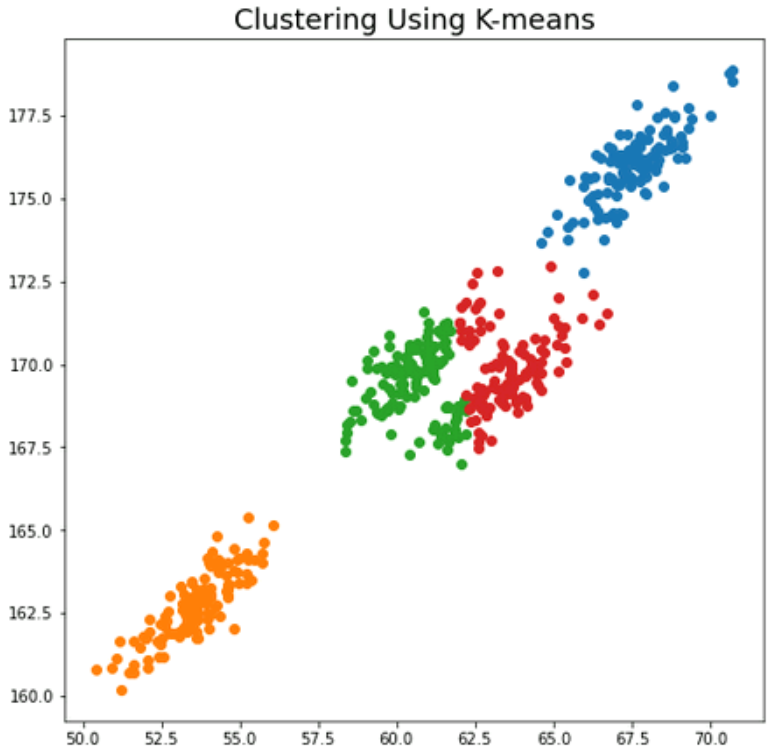
Predictions:

```
[0 0 2 2 0 2 3 2 3 1 3 0 3 1 2 2 0 2 1 2 2 0 3 1 2 1 0 3 1 3 3 3 0 3 3 0 1
 3 1 0 1 3 2 0 3 0 3 2 0 0 1 1 3 3 2 3 3 1 1 0 0 1 0 0 0 3 2 0 2 3 1 2 1 0
 1 0 3 3 2 2 1 2 1 3 2 3 1 1 0 1 1 1 0 1 2 2 0 0 0 2 1 1 0 1 2 2 2 0 2 1 1
 3 1 1 0 1 0 2 3 0 3 1 0 0 1 0 0 0 0 1 2 1 3 1 1 1 2 0 2 3 0 0 0 0 3 1 3 2
 3 1 1 0 2 2 2 3 3 0 2 2 0 2 0 0 3 3 0 2 1 3 0 1 0 3 3 1 3 1 1 1 0 0 3 2 2
 1 3 0 3 1 0 3 1 3 3 1 2 3 2 1 1 2 2 3 2 3 3 2 3 3 2 0 0 0 2 0 3 1 1 1 3 0
 3 1 3 0 1 2 1 2 3 3 2 3 2 3 2 3 1 0 0 3 2 0 1 1 2 0 2 3 2 3 0 1 2 1 1 3 1
 3 2 2 0 1 0 0 2 2 1 0 2 1 0 2 0 1 3 3 2 0 0 2 0 2 0 2 3 0 2 2 2 0 1 1 0 1
 1 1 2 0 0 1 3 1 0 0 3 1 1 2 1 3 0 3 1 3 1 2 3 2 2 2 3 3 1 0 2 3 1 0 2 2 0
 0 3 0 3 2 2 3 1 0 0 3 0 2 0 0 0 0 2 0 2 0 2 2 1 0 2 1 1 2 2 0 0 3 3 0 2 1
 2 2 2 1 2 2 2 1 1 2 1 1 2 1 1 3 2 3 3 1 1 1 1 0 1 0 2 0 1 2 0 2 0 2 2 3 1
 2 1 3 3 0 1 3 2 0 1 3 3 2 0 2 0 3 0 3 0 1 2 0 1 0 2 3 3 3 0 1 2 2 3 0 0 2
 2 3 2 2 1 0 3 0 1 2 3 3 1 3 2 1 3 3 3 3 3 2 1 0 3 0 2 0 2 1 2 1 3 1 2 1 2
 3 3 1 3 3 2 1 0 3 3 1 3 2 2 2 0 1 1 2]
```

```
# Inspect cluster assignments using sample() functions

data.sample(n=8)
```

	Weight	Height	Cluster
318	62.683789	168.655935	3
181	67.773328	176.911080	0
436	67.308444	175.667912	0
305	67.860729	175.896424	0
377	53.372145	162.916702	1
94	59.259593	168.818003	2
278	59.935944	169.973733	2
435	63.178255	169.252909	3

```
# Plot results

plt.figure(figsize=(8,8))     # Set the size of the figure

for k in range(0,4):
    data_cluster = data[data["Cluster"]==k]  # data[true]; Select the observaions that belong to cluster 'k'
    plt.scatter(data_cluster["Weight"],data_cluster["Height"],cmap = 'viridis')   # Scatter plot per cluster
                                                # cmap assigns different colors to each cluster

plt.title('Clustering Using K-means', fontsize = 18)            # Set the title of the plot
plt.show()
```




```
# Import libraries

import pandas as pd
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

# Read in the data

data = pd.read_csv('clustering_gmm.csv')
data.sample(n=3)
```

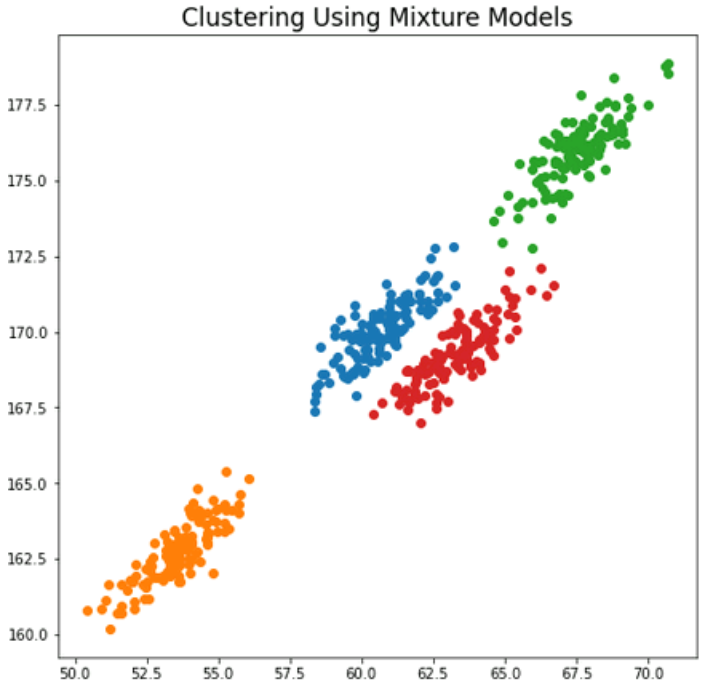
	Weight	Height
87	54.454723	163.984927
242	62.088906	168.715370
65	63.026581	167.723608

```
# Train the GaussianMixture on the data

from sklearn.mixture import GaussianMixture
gm_model = GaussianMixture(n_components=4)
gm_model.fit(data)

# Store the predictions of the mixture model
```

```
predictions = gm_model.predict(data)

data['Cluster'] = predictions

# Display random 5 rows

data.sample(n=10)
```

	Weight	Height	Cluster
356	53.669741	161.763262	1
135	54.349137	162.386507	1
31	64.514773	169.349350	3
248	61.090496	170.457400	0
494	62.200362	167.889268	3
404	61.159511	168.025080	3
448	53.936043	162.958416	1
349	67.180573	174.524007	2
48	67.792162	175.615118	2
415	68.820349	176.557909	2

```
# Visualize the clusters

plt.figure(figsize=(8,8))
for k in range(0,4):
    data_cluster = data[data["Cluster"]==k]
    plt.scatter(data_cluster["Weight"],data_cluster["Height"],cmap = 'viridis')

plt.title('Clustering Using Mixture Models', fontsize = 17)
plt.show()
```



M4: Apriori Algorithm

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules

df = pd.read_excel('Online Retail.xlsx')

# Note that describe() only summarizes the numerical columns in the dataset. To force it to also summarize other columns, one can pass the following argument, include = 'all'

# Format the Dataset

basket = (df[df['Country'] =="France"]                    # Pick entries with 'Country' being 'France'
          .groupby(['InvoiceNo', 'Description'])['Quantity']  # Group data based on InvoiceNo. (transaction) and  Description (product)
          .sum()    # Compute the total quantities of each product per invoice
          .unstack()
          .reset_index()
          .fillna(0)           # Enter 0 for any items that are not purchased
          .set_index('InvoiceNo')
         )

# Define function to convert one-hot encoding: if entry is one or greater, return 1
```

```
# if entry is 0 or smaller, return 0

def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

# Apply the above defined function to basket_sets

basket_sets = basket.applymap(encode_units)

# Obtain associations using the Apriori algorithm

frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)

frequent_itemsets.shape
(129, 2)

frequent_itemsets.dtypes
support     float64
itemsets     object
dtype: object

# Display all learned Associations

pd.options.display.max_rows = 129 # Display all rows
print(frequent_itemsets.tail(100))
     support                                itemsets
29   0.143167          (PLASTERS IN TIN CIRCUS PARADE )
30   0.117137             (PLASTERS IN TIN SPACEBOY)
31   0.069414             (PLASTERS IN TIN STRONGMAN)
32   0.145336       (PLASTERS IN TIN WOODLAND ANIMALS)
33   0.052061             (POPPY'S PLAYHOUSE KITCHEN)
34   0.650759                          (POSTAGE)
35   0.160521                (RABBIT NIGHT LIGHT)
…

# Remove 'POSTAGE' item

basket_sets.drop('POSTAGE', inplace=True, axis=1)


# Rerun the algorithm
```

```
frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)

print(frequent_itemsets)
    support                         itemsets
0   0.060738              (4 TRADITIONAL SPINNING TOPS)
1   0.082430               (ALARM CLOCK BAKELIKE GREEN)
2   0.086768                (ALARM CLOCK BAKELIKE PINK)
3   0.080260                (ALARM CLOCK BAKELIKE RED )
4   0.058568               (ASSORTED COLOUR MINI CASES)
5   0.069414             (BAKING SET 9 PIECE RETROSPOT )
6   0.058568              (CHARLOTTE BAG APPLES DESIGN)
7   0.056399            (CHARLOTTE BAG DOLLY GIRL DESIGN)
8   0.056399            (CHILDRENS APRON SPACEBOY DESIGN)
…

frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)

frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x)) # Define
an additional column 'length'

frequent_itemsets
        support         itemsets        length
0       0.060738        (4 TRADITIONAL SPINNING TOPS)       1
1       0.082430        (ALARM CLOCK BAKELIKE GREEN)    1
2       0.086768        (ALARM CLOCK BAKELIKE PINK)     1
3       0.080260        (ALARM CLOCK BAKELIKE RED )     1
4       0.058568        (ASSORTED COLOUR MINI CASES)    1
…
# Allow the entire column to be displayed

pd.set_option('display.max_colwidth', None)

# Display associations of length > 1

frequent_itemsets[(frequent_itemsets['length'] > 1)]
        support         itemsets        length
58      0.062907        (ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK
BAKELIKE PINK)  2
59      0.067245        (ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK
BAKELIKE RED )  2
60      0.062907        (ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELIKE
RED ) 2
```

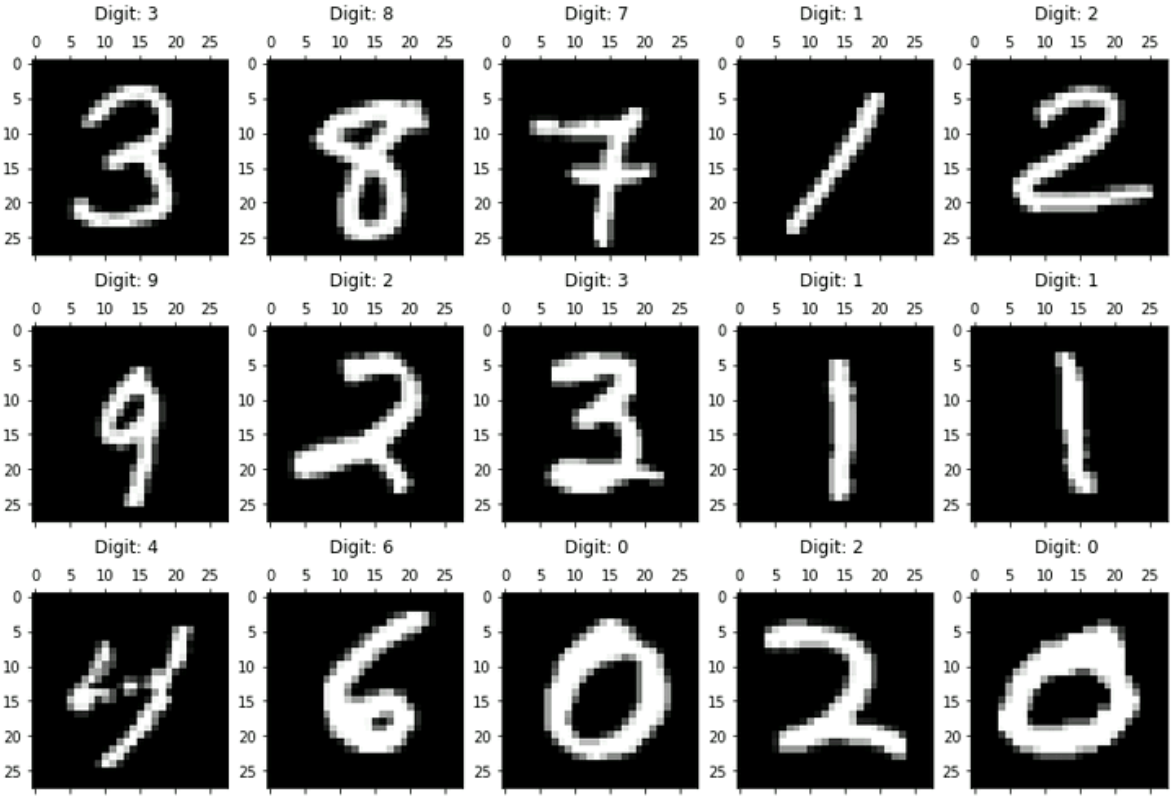
```
61      0.054230        (CHILDRENS CUTLERY SPACEBOY , CHILDRENS CUTLERY
DOLLY GIRL )        2
62      0.060738        (SPACEBOY LUNCH BOX , DOLLY GIRL LUNCH BOX)         2
63      0.056399        (LUNCH BAG APPLE DESIGN, LUNCH BAG RED RETROSPOT)
        2
…
```

M5: PCA and t-SNE

```
# Import relevant libraries

from __future__ import print_function
import time
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# Import algorithms from Sklearn

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Load MNIST dataset using pandas

df = pd.read_csv('mnist_784_subset.csv')

print('Size of the dataframe: {}'.format(df.shape))
Size of the dataframe: (10000, 785)

df.dtypes
pixel1      int64
pixel2      int64
pixel3      int64
pixel4      int64
pixel5      int64
           ...
pixel781    int64
pixel782    int64
pixel783    int64
pixel784    int64
class       int64
```

```
Length: 785, dtype: object

# Define variables

X = df.drop('class',axis=1)
y = df['class']

# Inspect the shape of the data

print(X.shape, y.shape)
(10000, 784) (10000,)

# Define an array with names features

feat_cols = X.columns

# Set a random seed for reproducibility of code using NumPy

np.random.seed(10)
rndperm = np.random.permutation(df.shape[0])

# Plot 15 images to get a sense of how the images look like

plt.gray()
fig = plt.figure( figsize=(16,12) )
for i in range(0,15):
    ax = fig.add_subplot(3,5,  # 3 rows, 5 columns
                         i+1,
                         title="Digit: {} \n"  # Set a title for each image with its label
                         .format(str(df.loc[rndperm[i],'class'])) )  #

    ax.matshow(df.loc[rndperm[i],feat_cols]
               .values
               .reshape((28,28))  # Reformat each observation (image) to its original 28x28 shape
               .astype(float)
              )
plt.subplots_adjust(bottom=0.1, right=0.8, top=0.7)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Run PCA with three components

pca = PCA(n_components=3)
pca_result = pca.fit_transform(df[feat_cols].values)

Add the PCA components as columns (features) to the dataframe

df['First Principal Component'] = pca_result[:,0] # Select the first column
df['Second Principal Component'] = pca_result[:,1] # Select the second column
df['Third Principal Component'] = pca_result[:,2]

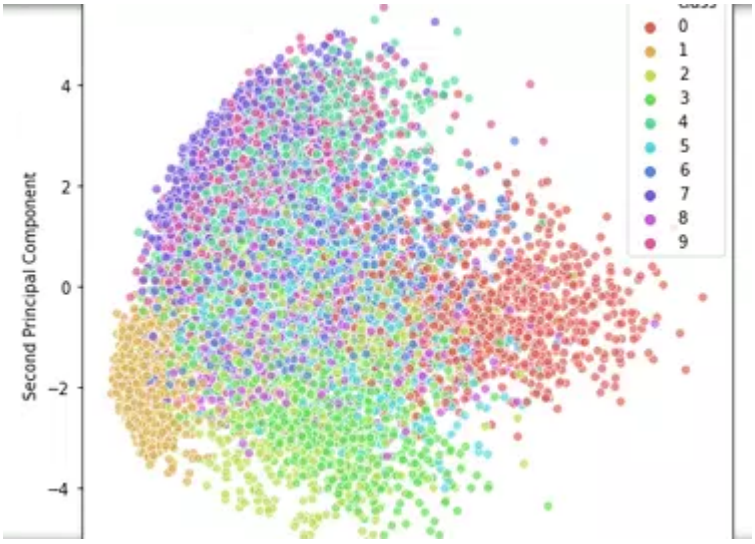
Compute the amount of variation

print('Explained variation per principal component:
{}'.format(pca.explained_variance_ratio_))
Explained variation per principal component: [0.09862132 0.07236824 0.06289581]

View the PCA components

pca_result
array([[3.88198345, 1.75235064, 0.07212854],
       [-2.34986151, 2.2061296 , -2.12294161],

```
       [-1.17082709, -1.1273351 ,  2.04783921],
       ...,
       [-0.40073054,  1.23909288,  2.51718872],
       [-0.43855931, -1.0072638 , -0.0450079 ],
       [ 1.69850537, -0.88357368, -2.31765729]])

# View first 5 rows in data
# Note the three PCA components added as columns to the dataset

df.head().T
                                0         1         2         3         4
pixel1                   0.000000  0.000000  0.000000  0.000000  0.000000
pixel2                   0.000000  0.000000  0.000000  0.000000  0.000000
pixel3                   0.000000  0.000000  0.000000  0.000000  0.000000
pixel4                   0.000000  0.000000  0.000000  0.000000  0.000000
pixel5                   0.000000  0.000000  0.000000  0.000000  0.000000
...                           ...       ...       ...       ...       ...
pixel784                 0.000000  0.000000  0.000000  0.000000  0.000000
class                    6.000000  7.000000  5.000000  7.000000  8.000000
First Principal Component 3.881983 -2.349862 -1.170827 -1.175033
1.275850
Second Principal Component 1.752351  2.206130 -1.127335  1.701116
-0.862820
Third Principal Component 0.072129 -2.122942  2.047839  2.410406
-3.524901

# Scatter plot of the first two PCA components

plt.figure(figsize=(16,10))
sns.scatterplot(
    x = "First Principal Component", y="Second Principal Component",
    hue ="class", # Set the variable that defines the clusters
 #   style ="y",
    palette = sns.color_palette("hls", 10),
    data = df.loc[rndperm,:],   # Set the dataframe with permutated rows
    legend = "full",   # Every group/cluster will get an entry in the legend
    alpha = 0.7    # Set opacity of the points
)
```



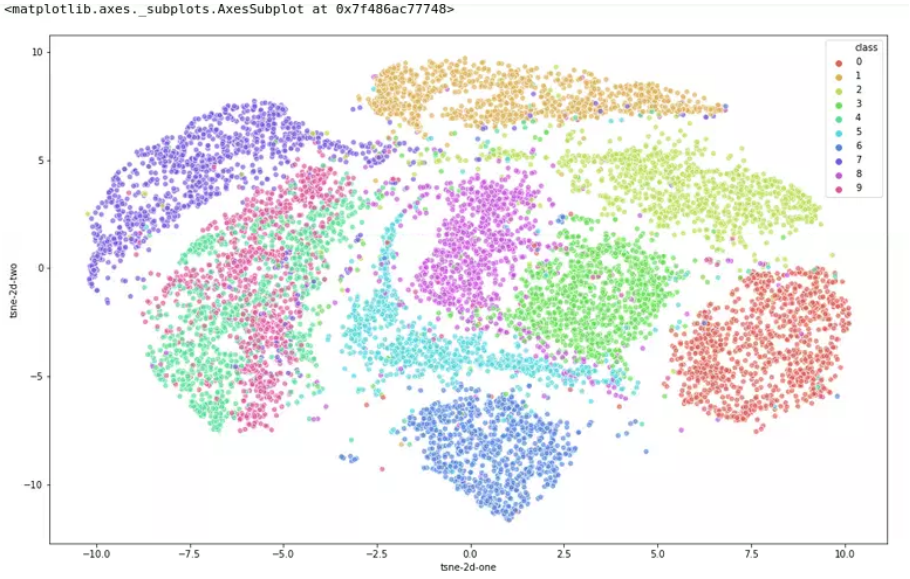

```
# Applying t-SNE

# Track time of the t-SNE algorithm run

time_start = time.time()

# Run t-SNE with two components

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
#tsne_results = tsne.fit_transform(data_subset)
tsne_results = tsne.fit_transform(X)

# Print termination message when t-SNE terminates

print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 10000 samples in 1.229s...
[t-SNE] Computed neighbors for 10000 samples in 147.210s...
[t-SNE] Computed conditional probabilities for sample 1000 / 10000
[t-SNE] Computed conditional probabilities for sample 2000 / 10000
[t-SNE] Computed conditional probabilities for sample 3000 / 10000
[t-SNE] Computed conditional probabilities for sample 4000 / 10000
[t-SNE] Computed conditional probabilities for sample 5000 / 10000
[t-SNE] Computed conditional probabilities for sample 6000 / 10000
[t-SNE] Computed conditional probabilities for sample 7000 / 10000
[t-SNE] Computed conditional probabilities for sample 8000 / 10000
[t-SNE] Computed conditional probabilities for sample 9000 / 10000
[t-SNE] Computed conditional probabilities for sample 10000 / 10000
[t-SNE] Mean sigma: 2.127102
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.826462
```

[t-SNE] KL divergence after 300 iterations: 2.806357
t-SNE done! Time elapsed: 165.56734776496887 seconds

Visualize t-SNE in a scatter plot

```
df['tsne-2d-one'] = tsne_results[:,0]
df['tsne-2d-two'] = tsne_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="class",
    palette=sns.color_palette("hls", 10),
    data=df,
    legend="full",
    alpha=0.7
)
#sns.set(font_scale = 12)
#sns.set_xticklabels(sns.get_xmajorticklabels(), fontsize = 18)
#plot.set_yticklabels(plot.get_yticks(), size = 3)
#plt.show()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f486ac77748>
```



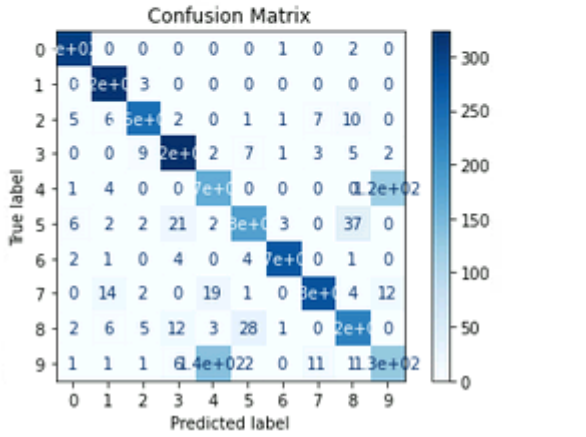
SVM prep

```
# Set X as the t-SNE components (tsne-2d-one and tsne-2d-two) and y as the 'class' feature
# Hint: Convert the two t-SNE components into dataframes then use concat() method to combine both in one dataframe
frames = [df['tsne-2d-one'].to_frame(), df['tsne-2d-two'].to_frame()]
X = pd.concat(frames, axis=1)
print(X.head()) # confirming concat result
y = df['class']
```

```
   tsne-2d-one  tsne-2d-two
0     1.402032   -10.613814
1    -8.224123     2.248989
2    -2.015823    -2.413583
3    -5.298238     6.693152
4    -0.493880    -1.553725
```

```
# Split data into training and test sets with a 70-30 split and random_state = 109
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 109)

# Import svm model from Sklearn and run the SVM classifier
from sklearn import svm
svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

# Plot the confusion matrix
# Hint: Recall that this dataset has 10 classes

from sklearn.metrics import (confusion_matrix, plot_confusion_matrix)

conf = confusion_matrix(y_test, y_pred)
print(conf)

disp = plot_confusion_matrix(svm_model,
                             X_test,
                             y_test,
                             display_labels=["0","1", "2", "3", "4", "5", "6", "7", "8", "9"],
                             cmap=plt.cm.Blues)
disp.ax_.set_title("Confusion Matrix")

plt.show()
```

```
[[304   0   0   0   0   0   1   0   2   0]
 [  0 317   3   0   0   0   0   0   0   0]
 [  5   6 247   2   0   1   1   7  10   0]
 [  0   0   9 323   2   7   1   3   5   2]
 [  1   4   0   0 170   0   0   0   0 121]
 [  6   2   2  21   2 185   3   0  37   0]
 [  2   1   0   4   0   4 274   0   1   0]
 [  0  14   2   0  19   1   0 280   4  12]
 [  2   6   5  12   3  28   1   0 224   0]
 [  1   1   1   6 138   2   0  11   1 128]]
```



# Obtain the accuracy score
svm_model_s = svm_model.score(X_test, y_test)
print("Accuracy:", int(svm_model_s *100), "%")
Accuracy: 81 %

# Obtain the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=[0,1, 2, 3, 4, 5, 6, 7, 8, 9],
target_names=["0","1", "2", "3", "4", "5", "6", "7", "8", "9"]))

	precision	recall	f1-score	support
0	0.95	0.99	0.97	307
1	0.90	0.99	0.94	320
2	0.92	0.89	0.90	279
3	0.88	0.92	0.90	352
4	0.51	0.57	0.54	296
5	0.81	0.72	0.76	258
6	0.98	0.96	0.97	286
7	0.93	0.84	0.88	332
8	0.79	0.80	0.79	281
9	0.49	0.44	0.46	289
accuracy			0.82	3000
macro avg	0.81	0.81	0.81	3000
weighted avg	0.82	0.82	0.82	3000

```
# MNIST dataset
N = 10000

np.random.seed(42)
rndperm = np.random.permutation(df.shape[0])
df_subset = df.loc[rndperm[:N],:].copy()

data_subset = df_subset[feat_cols].values
df_subset.to_csv('mnist_784_subset.csv')

# For your reference, you could access the entire MNIST dataset directly by loading it from
from https://www.openml.org/d/554 using the dataset's name and version number as shown in
the command below:

df = fetch_openml('mnist_784', version=1, cache=True)
```

Final Lab

```
# For this lab, you need to import pandas, NumPy, Matplotlib and Seaborn.

import pandas as pd
import numpy as np
import matplotlib as ml
import seaborn as sns

# You also need to import the following from sklearn:
#   metrics and svm;
```

```
#   GaussianNB from naive_bayes;
#   confusion_matrix, plot_confusion_matrix, classification_report from metrics
#   LogisticRegression from linear_model

from sklearn import metrics
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression

# Read in the dataset heart.csv using pandas

df = pd.read_csv('heart.csv')

# Read first 5 rows of the dataset

df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3.0	145	233.0	1	0	150	0.0	2.3	0	0.0	1	1.0
1	37	1	2.0	130	250.0	0	1	187	0.0	3.5	0	0.0	2	1.0
2	41	0	1.0	130	204.0	0	0	172	0.0	1.4	2	0.0	2	1.0
3	56	1	1.0	120	236.0	0	1	178	0.0	0.8	2	0.0	2	1.0
4	57	0	0.0	120	354.0	0	1	163	1.0	0.6	2	0.0	2	1.0

```
# Rename the columns for better readability

columns_names = {'cp':'chest_pain_type','trestbps':'resting_blood_pressure',
'exang':'exercise_ang', 'chol': 'serum_cholesterol','fbs': 'fasting_blood_sugar', 'thal':
'max_heart_rate',}

# Hint: Use pandas rename() method and input columns_names to it

df.rename(columns=columns_names, inplace=True)

# Display the datatypes for all the features

print(df.dtypes)
```

```
age                       int64
sex                       int64
chest_pain_type           float64
resting_blood_pressure    int64
serum_cholesterol         float64
fasting_blood_sugar       int64
restecg                   int64
thalach                   int64
exercise_ang              float64
oldpeak                   float64
slope                     int64
ca                        float64
max_heart_rate            int64
target                    float64
dtype: object
```

Check for NaN values

df.isna().any()

```
age                     False
sex                     False
chest_pain_type          True
resting_blood_pressure  False
serum_cholesterol        True
fasting_blood_sugar     False
restecg                 False
thalach                 False
exercise_ang             True
oldpeak                 False
slope                   False
ca                       True
max_heart_rate          False
target                   True
dtype: bool
```

Check the number of missing values per feature

df.isna().sum()

```
age                     0
sex                     0
chest_pain_type         1
resting_blood_pressure  0
serum_cholesterol       1
```

```
fasting_blood_sugar       0
restecg                   0
thalach                   0
exercise_ang              3
oldpeak                   0
slope                     0
ca                        1
max_heart_rate            0
target                    1
dtype: int64
```

```
# Print rows having NaN values
df[df.isna().any(axis=1)]
```

```
# This is a good step to analyze missing fields and decide the best approach to deal with them.
```

	age	sex	chest_pain_type	resting_blood_pressure	serum_cholesterol	fasting_blood_sugar	restecg	thalach	exercise_ang	oldpeak	slope	ca	max_heart_rate	target
5	57	1	NaN	140	NaN	0	1	148	NaN	0.4	1	NaN	1	NaN
11	48	0	2.0	130	275.0	0	1	139	NaN	0.2	2	0.0	2	1.0
20	59	1	0.0	135	234.0	0	1	161	NaN	0.5	1	0.0	3	1.0

```
# Compute the mean of exercise_ang column

exercise_ang_mean = df['exercise_ang'].mean()

# Replace NaN entries in the column 'exercise_ang' by its mean
# Hint: use fillna() method

df.fillna(value={'exercise_ang': exercise_ang_mean}, inplace=True)

# Use the head method to confirm the new entries in rows 11 and 20

df.head(21)
```

	age	sex	chest_pain_type	resting_blood_pressure	serum_cholesterol	fasting_blood_sugar	restecg	thalach	exercise_ang	oldpeak	slope	ca	max_heart_rate	target
0	63	1	3.0	145	233.0	1	0	150	0.00	2.3	0	0.0	1	1.0

1	37	1	2.0	130	250.0	0	1	187	0.00	3.5	0	0.0
	2	1.0										
2	41	0	1.0	130	204.0	0	0	172	0.00	1.4	2	0.0
	2	1.0										
3	56	1	1.0	120	236.0	0	1	178	0.00	0.8	2	0.0
	2	1.0										
4	57	0	0.0	120	354.0	0	1	163	1.00	0.6	2	0.0
	2	1.0										
5	57	1	NaN	140	NaN	0	1	148	0.33	0.4	1	
NaN	1	NaN										
6	56	0	1.0	140	294.0	0	0	153	0.00	1.3	1	0.0
	2	1.0										
7	44	1	1.0	120	263.0	0	1	173	0.00	0.0	2	0.0
	3	1.0										
8	52	1	2.0	172	199.0	1	1	162	0.00	0.5	2	0.0
	3	1.0										
9	57	1	2.0	150	168.0	0	1	174	0.00	1.6	2	0.0
	2	1.0										
10	54	1	0.0	140	239.0	0	1	160	0.00	1.2	2	0.0
	2	1.0										
11	48	0	2.0	130	275.0	0	1	139	0.33	0.2	2	0.0
	2	1.0										
12	49	1	1.0	130	266.0	0	1	171	0.00	0.6	2	0.0
	2	1.0										
13	64	1	3.0	110	211.0	0	0	144	1.00	1.8	1	0.0
	2	1.0										
14	58	0	3.0	150	283.0	1	0	162	0.00	1.0	2	0.0
	2	1.0										
15	50	0	2.0	120	219.0	0	1	158	0.00	1.6	1	0.0
	2	1.0										
16	58	0	2.0	120	340.0	0	1	172	0.00	0.0	2	0.0
	2	1.0										
17	66	0	3.0	150	226.0	0	1	114	0.00	2.6	0	0.0
	2	1.0										
18	43	1	0.0	150	247.0	0	1	171	0.00	1.5	2	0.0
	2	1.0										
19	69	0	3.0	140	239.0	0	1	151	0.00	1.8	2	2.0
	2	1.0										
20	59	1	0.0	135	234.0	0	1	161	0.33	0.5	1	0.0
	3	1.0										

Remove the rest of the NaN values (row 5)
Hint: use dropna() method

```
df.dropna(inplace=True)

# Make sure no more NaN values (same method as used above to first check for NaN values)

df.isna().any()
```

```
age                   False
sex                   False
chest_pain_type       False
resting_blood_pressure  False
serum_cholesterol     False
fasting_blood_sugar   False
restecg               False
thalach               False
exercise_ang          False
oldpeak               False
slope                 False
ca                    False
max_heart_rate        False
target                False
dtype: bool
```

```
# Change the data type of the target to int, then display several entries to confirm
# Hint: Use .loc method to locate the target column

df[['target']] = df.loc[:,['target']].astype(int)
df.head()
```

	age	sex	chest_pain_type	resting_blood_pressure	serum_cholesterol	fasting_blood_sugar	restecg	thalach	exercise_ang	oldpeak	slope	ca	max_heart_rate	target
0	63	1	3.0	145	233.0	1	0	150	0.0	2.3	0	0.0	1	1
1	37	1	2.0	130	250.0	0	1	187	0.0	3.5	0	0.0	2	1
2	41	0	1.0	130	204.0	0	0	172	0.0	1.4	2	0.0	2	1
3	56	1	1.0	120	236.0	0	1	178	0.0	0.8	2	0.0	2	1
4	57	0	0.0	120	354.0	0	1	163	1.0	0.6	2	0.0	2	1

```
# View a few random rows of the dataset to confirm target is changed to int
```

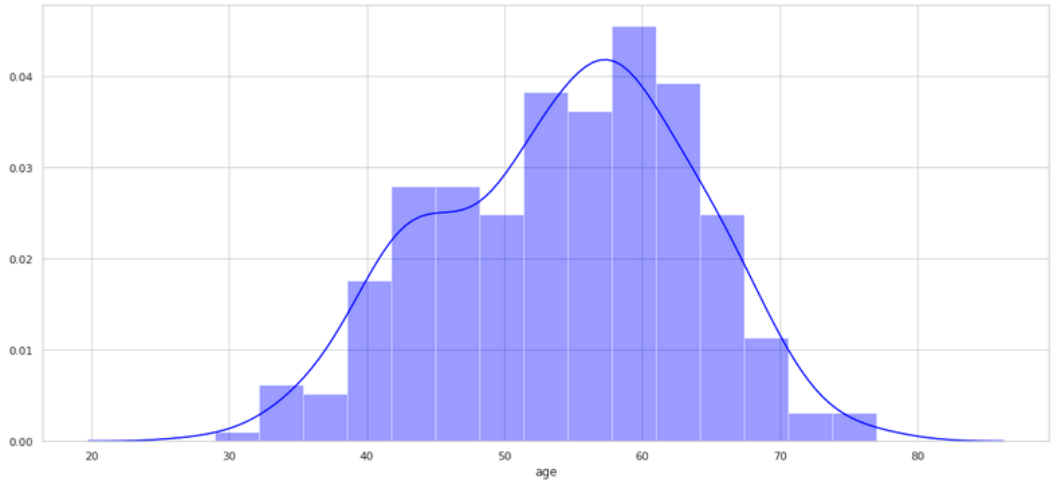
df.sample(10)

	age	sex	chest_pain_type	resting_blood_pressure	serum_cholesterol	fasting_blood_sugar	restecg	thalach	exercise_ang	oldpeak	slope	ca	max_heart_rate	target
285	46	1	0.0	140	311.0	0	1	120	1.0	1.8	1	2.0	3	0
160	56	1	1.0	120	240.0	0	1	169	0.0	0.0	0	0.0	2	1
295	63	1	0.0	140	187.0	0	0	144	1.0	4.0	2	2.0	3	0
131	49	0	1.0	134	271.0	0	1	162	0.0	0.0	1	0.0	2	1
85	67	0	2.0	115	564.0	0	0	160	0.0	1.6	1	0.0	3	1
191	58	1	0.0	128	216.0	0	0	131	1.0	2.2	1	3.0	3	0
149	42	1	2.0	130	180.0	0	1	150	0.0	0.0	2	0.0	2	1
246	56	0	0.0	134	409.0	0	0	150	1.0	1.9	1	2.0	3	0
279	61	1	0.0	138	166.0	0	0	125	1.0	3.6	1	1.0	2	0
22	42	1	0.0	140	226.0	0	1	178	0.0	0.0	2	0.0	2	1

Plot the distribution of age feature in the dataset using Seaborn
Hint, use Seaborn's distplot() method and specify a number of bins

```
sns.set(rc={'figure.figsize':(18,8)})
sns.set_style(style='whitegrid')

sns.distplot(df['age'], color = 'blue', bins = 15)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f002ef9f1d0>
```



Define variables: X is everything but target; y is target.

```
X = df.drop(columns = 'target')
y = df['target']
```

Display the first few X entries and compare to the cleaned dataset

```
X.head()
```

	age	sex	chest_pain_type	resting_blood_pressure	serum_cholesterol	fasting_blood_sugar	restecg	thalach	exercise_ang	oldpeak	slope	ca	max_heart_rate
0	63	1	3.0	145	233.0	1	0	150	0.0	2.3	0	0.0	1
1	37	1	2.0	130	250.0	0	1	187	0.0	3.5	0	0.0	2
2	41	0	1.0	130	204.0	0	0	172	0.0	1.4	2	0.0	2
3	56	1	1.0	120	236.0	0	1	178	0.0	0.8	2	0.0	2
4	57	0	0.0	120	354.0	0	1	163	1.0	0.6	2	0.0	2

Display the first few y entries and compare to the cleaned dataset

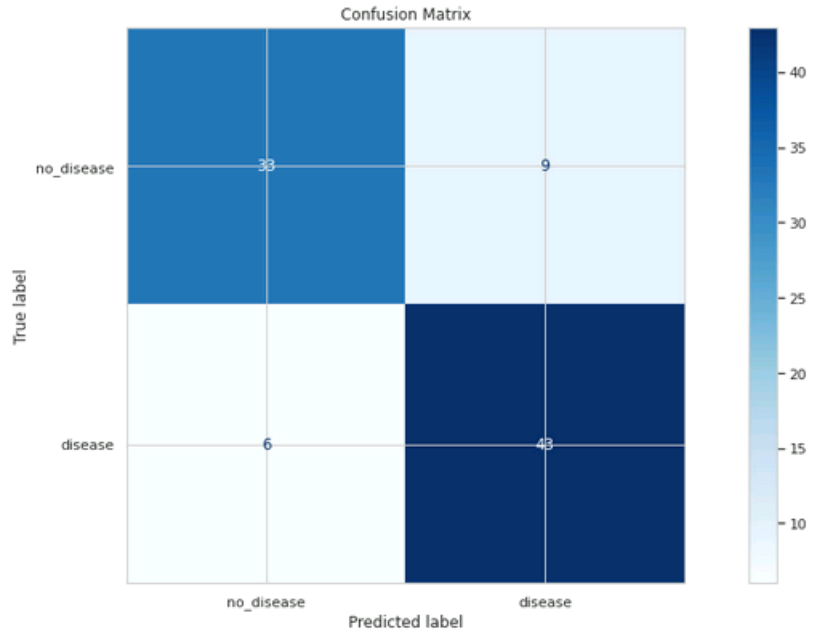
```
y.head()
```

```
0    1
1    1
2    1
3    1
4    1
```

Name: target, dtype: int64

```
# Split data to training and test sets with split 70-30
# Use random_state = 42
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

# Run Logistic Regression model
logreg = LogisticRegression(solver='liblinear', max_iter=1000)
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

# Obtain and plot the confusion matrix
conf = confusion_matrix(y_test, y_pred)

print(conf)

disp = plot_confusion_matrix(logreg, X_test, y_test, display_labels=["no_disease","disease"],
cmap = ml.pyplot.cm.Blues)
disp.ax_.set_title("Confusion Matrix")

ml.pyplot.show()
```

```
[[33  9]
 [ 6 43]]
```



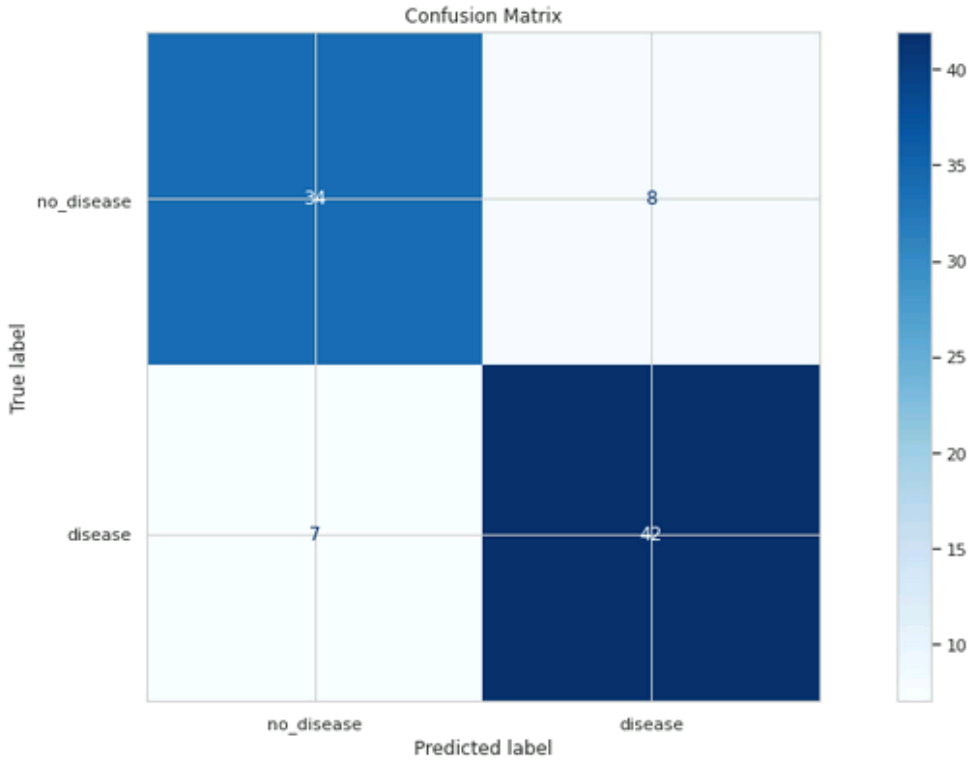

```
# Compute the accuracy score; your result should be approximately 0.835
accuracy = (conf[0,0] + conf[1,1])/(conf[0,0] + conf[0,1] + conf[1,0] + conf[1,1])
print("Accuracy:", accuracy)
```

```
# Obtain the classification report

print("Classification Report:\n", classification_report(y_test, y_pred, labels=[0,1],
target_names=['no_disease','disease']))

# Obtain the precision score
print("Precision:", metrics.precision_score(y_test, y_pred, labels=[0,1]))
```

```
Accuracy: 0.8351648351648352
Classification Report:
              precision    recall  f1-score   support

  no_disease       0.85      0.79      0.81        42
     disease       0.83      0.88      0.85        49

    accuracy                           0.84        91
   macro avg       0.84      0.83      0.83        91
weighted avg       0.84      0.84      0.83        91

Precision: 0.8269230769230769
```

```
# The accuracy score indicates that 83.5% of the observations were correctly classified (as
disease or no_disease) and the precision score indicates that 82.7% of positive predictions
(that is disease) are really positive (that is they belong to patients with heart disease.

# Run SVM model and obtain the predictions
svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)

# Obtain and plot the confusion matrix
conf2 = confusion_matrix(y_test, y_pred_svm)

print(conf2)

disp2 = plot_confusion_matrix(svm_model, X_test, y_test,
display_labels=["no_disease","disease"], cmap = ml.pyplot.cm.Blues)
disp2.ax_.set_title("Confusion Matrix")

ml.pyplot.show()
```

```
[[34  8]
 [ 7 42]]
```



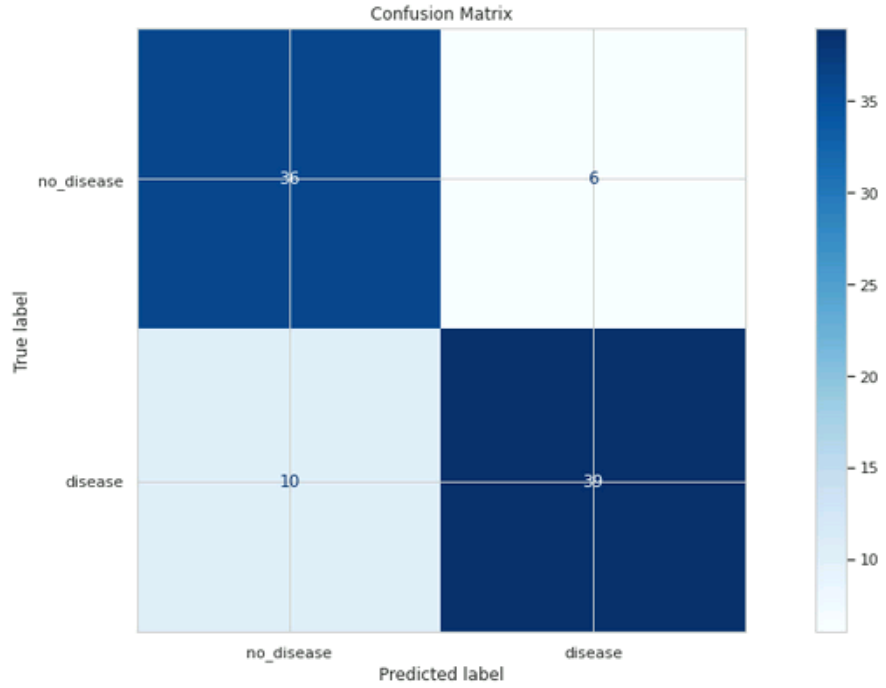

```
# Obtain the accuracy score
accuracy2 = (conf2[0,0] + conf2[1,1])/(conf2[0,0] + conf2[0,1] + conf2[1,0] + conf2[1,1])
print("Accuracy:", accuracy2)

# Obtain the classification report
print("Classification Report:\n", classification_report(y_test, y_pred_svm, labels=[0,1],
target_names=['no_disease','disease']))

# Obtain the Precision Score
print("Precision:", metrics.precision_score(y_test, y_pred_svm, labels=[0,1]))
```

```
Accuracy: 0.8351648351648352
Classification Report:
              precision    recall  f1-score   support

  no_disease       0.83      0.81      0.82        42
     disease       0.84      0.86      0.85        49

    accuracy                           0.84        91
   macro avg       0.83      0.83      0.83        91
weighted avg       0.84      0.84      0.84        91
```

Precision: 0.84

```
# Run the Naive Bayes algorithm
gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)

y_pred_gnb = gnb_model.predict(X_test)

# Obtain and plot the confusion matrix
conf3 = confusion_matrix(y_test, y_pred_gnb)

print(conf3)

disp3 = plot_confusion_matrix(gnb_model, X_test, y_test,
display_labels=["no_disease","disease"], cmap = ml.pyplot.cm.Blues)
disp3.ax_.set_title("Confusion Matrix")

ml.pyplot.show()
```

```
[[36  6]
 [10 39]]
```




```
# Obtain the accuracy score
accuracy3 = (conf3[0,0] + conf3[1,1])/(conf3[0,0] + conf3[0,1] + conf3[1,0] + conf3[1,1])
print("Accuracy:", accuracy3)

# Obtain the classification report
```

```
print("Classification Report:\n", classification_report(y_test, y_pred_gnb, labels=[0,1],
target_names=['no_disease','disease']))

# Obtain the Precision Score
print("Precision:", metrics.precision_score(y_test, y_pred_gnb, labels=[0,1]))
```

```
Accuracy: 0.8241758241758241
Classification Report:
              precision    recall  f1-score   support

  no_disease       0.78      0.86      0.82        42
     disease       0.87      0.80      0.83        49

    accuracy                           0.82        91
   macro avg       0.82      0.83      0.82        91
weighted avg       0.83      0.82      0.82        91

Precision: 0.8666666666666667
```

```
# The three models' performance on the heart.csv is very close from each other. But, the
Naive Bayes has the highest precision score (86.7%).
```