CERTIFICATE

This is to certify that student Mr. Pranav Bhavsar of M.Sc. (CS) Semester III having Seat No.21339 at Suryadatta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical in Machine Learning prescribed by the University of Pune During the academic year 2021-2022.

Internal Examiner **External**

Principal

Place: Pune

Date: 25/02/2022

1. **Write a python program to Prepare Scatter Plot (Use Forge Dataset / Iris Dataset)**
   Solution:

```
Import pandas as pd
Import numpy as np
Import matplotlib.pyplot as plt# detail mode;
df = pd.read_csv('Iris.csv')
df.plot(kind ="scatter",x ='SepalLengthCm', y ='PetalLengthCm')
plt.grid()
```

2. **Write a python program to find all null values in a given data set and remove them.**
   Solution:

```
import pandas as pd

dict = {'First Score':[100, 90, np.nan, 95],
'Second Score': [30, 45, 56, np.nan],
'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)

df.isnull()
df.isnull().sum() #find all null values
df = df.dropna() #drop
df.isnull()
```

3. **Write a python program the Categorical values in numeric format for a given dataset.**
   Solution:

```
import pandas as pd

#We have created a dictionary dataframe with columns 'name', 'episodes', 'gender'.

data = {'name': ['Manisha', 'Gautami', 'Nilima', 'Jesika', 'Devashish', 'Manisha'],
'episodes': [42, 24, 31, 29, 37, 40],
'gender': ['female', 'female', 'female', 'female', 'male', 'female']}

#coverted into dataframe
df = pd.DataFrame(data, columns = ['name','episodes', 'gender'])
```

#Print Dataframe
print(df)

#Categorical Data is the **data that generally takes a limited number of possible values**. Also, the data in the category need not be numerical
df_gender = pd.get_dummies(df['gender'])
new_dataframe = pd.concat([df, df_gender], axis=1)

#print new Dataframe
print(df_new_dataframenew)

4. **Write a python program to implement simple Linear Regression for predicting house price.**

Solution:

#Linear regression is a statistical method for modeling relationships between a dependent and independent variables.

#dependent variables as responses and independent variables as features

#Simple Linear Regression
#Simple linear regression is an approach for predicting a response using a single feature.

Implement Linear Regression for Predicting House Prices

```
import matplotlib.pyplot as mtp
import pandas as pd
import numpy as np
```

#our data is in the CSV file, we will read the CSV using pandas read_csv function
```
data_set= pd.read_csv('DataLR.csv')
```

#now we need to train out the regression model. We will need to first split up our data
#into an X and y list

Area is independent Variable
```
x= data_set[Area]
```

Price is Dependent Variable
```
y= data_set[Price]
```

```
#after that we are Splitting the dataset into training and test set using sklearn train_test_split().
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)


#Fitting the Simple Linear Regression model to the training dataset so here we are importing LinearRegression
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)


# after that we are Predicting Test and Training set result
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)


mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Area vs Prices (Training Dataset)")
mtp.xlabel("Area of Houses")
mtp.ylabel("Prices(In Rupees)")
mtp.show()

mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Area vs Price (Test Dataset)")
mtp.xlabel("Area of Houses ")
mtp.ylabel("Prices(In Rupees)")
mtp.show()


#Predict Price of Particular House
new_price_pred = regressor.predict([[1500]])

print('The predicted Price of house whose area is 1500 ',new_price_pred)

#to Find out Error in actual and predicted housing price so will import sklearn mean_squared_error

from sklearn.metrics import mean_squared_error
```

```
error = mean_squared_error(y_test,y_pred)

print('The error between predicted and actual is ',error)



#Now i have taken new .csv file in which only area is mentioned...using that will find out price using
#.predict function and save it to the new .csv file
d = pd.read_csv('prdata.csv')

regressor.predict(d)
p = regressor.predict(d)
d['Prices'] = p
d.to_csv('prediction2.csv')
```

5. **Write a python program to implement multiple Linear Regression for a given dataset.**

Solution:

```
#Multiple Linear Regression is extension of simple linear regression.
#A regression model that contains more than one independent variables and one dependent variable is calledMultiple regression model

#we are consideriong car.csv dataset that contains some information about cars like, company,model,volume of engine,weight of car and CO2 emmision
#Based on size of engine we can easily predict the total CO2 emission.But if we consider one more variable like car weight then regression model will give new accurate prediction

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

#our data is in the CSV file, we will read the CSV using pandas read_csv function
df = pandas.read_csv("cars.csv")

#now we need to train out the regression model. We will need to first split up our data into an X and y list
```

```
df.head()X = df[['Weight', 'Volume']]
y = df['CO2']

#after that we are Splitting the dataset into training and test set using sklearn
train_test_split().
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)

#Fitting the Simple Linear Regression model to the training dataset so here we are
importing LinearRegression
mlr = LinearRegression()
mlr.fit(x_train,y_train)

#For retrieving slope and coefficient we are using .intercept_ and .coef_
print("Intercept",mlr.intercept_)
print("Intercept",mlr.coef_)

#using interception and coefficient value we can predict CO2 emission for given
weight and volume values
predictCO2 = mlr.predict([[1500,1140]])
print(predictedCO2)
```

6. **Write a python program to implement Polynomial Regression for given dataset.**

Solution:

```
# Polynomial Regression: it gives low error rate and high accuracy
# Implementation of polynomial regression Python

# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
# PolynomialFeaturesgenerates a new matrix with all polynomial combinations of
features with given degree.
from sklearn.linear_model import LinearRegression


#our data is in the CSV file, we will read the CSV using pandas read_csv function
data_set= pd.read_csv('Position_Salaries.csv')
#now we need to train out the regression model. We will need to first split up our
data into an X and y list
x= data_set.iloc[:,1:2].values
y= data_set.iloc[:,2].values
```

```
#after that we are Splitting the dataset into training and test set using sklearn train_test_split().
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.3, random_state=0)

#Fitting the Linear Regression to the dataset
lin_regs= LinearRegression()
lin_regs.fit(x_train,y_train)

 #Fitting the Polynomial regression to the dataset by importing PolynomialFeatures from sklearn.preprocessingPolynomialFeatures generates a new matrix with all polynomial combinations of features with given degree. we are using dgree=2
poly_regs = PolynomialFeatures(degree = 2)
x_poly = poly_regs.fit_transform(x_train)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(x_poly, y_train)

#Visulaizing the result for Linear Regression model
mtp.scatter(x_train,y_train,color="blue")
mtp.plot(x_train,lin_regs.predict(x_train), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()

#Visulaizing the result for Polynomial Regression
mtp.scatter(x_train,y_train,color="green")
mtp.plot(x_train, lin_reg_2.predict(poly_regs.fit_transform(x_train)), color="blue")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```

7. **Write a python program to Implement Naïve Bayes.**

Solution:

#**Naïve Bayes: Naive Bayes** are a group of supervised machine learning classification algorithms based on the **Bayes theorem**

#**Bayes' theorem**: relationship between 2 events and their probabilities or one can determine probability of a hypothesis with prior knowledge, means predict output on basis of probability of an object.

```
# Importing essential libraries
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')

# Making the Feature matris and dependent vector
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state
= 0)

#Feature Scaling
#StandardScaler standardizes a feature by subtracting the mean and then scaling to unit
variance. Unit #variance means dividing all the values by the standard deviation

# fit_transform()-The fit_transform() method first fits, then transforms the data-set in the
same implementation. The fit_transform() method is an efficient implementation of
the fit() and transform() methods.
 fit_transform() is only used on the training data set as a "best practice".

 #The transform() method transforms the training data and the test data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# Fitting Naive Bayes to the Training set
#here we are using Gaussian naïve bayes. It is used when values of features are in
continuous form/nature. It follows Gaussian normal distribution
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)



# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
# Making the Confusion Matrix
#A confusion matrix is a table that is used to describe the performance of a classification model
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)


# Visualising the Training set results .
#We are importing ListedColormap module from matplotlib. colors module is used for converting numbers arguments to RGBA or RGB. This #module is used for mapping numbers to colors
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01), np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

8. Write a python program to Implement Decision Tree whether or not to play tennis.

Solution:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 8 19:44:50 2020
```

```
@author: Nilesh
"""
#. Write a python program to Implement Decision Tree whether or not to play tennis.
#Decision tree is a supervised machine learning algorithm used for both prediction
and classification, based on divide and conqur strategy
#Now we will split our dataset into a training set and testing set using sklearn
train_test_split(). the training set will be going to use for training the model and
testing set and we are using plot_tree to plot decision tree

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

df = pd.read_csv('F:/python/playD.csv')

#we are using get_dummies function for encoding. It convert categorical data into
dummy

df_getdummy = pd.get_dummies(data=df, columns=['Temperature', 'Outlook',
'Windy'])

#Now we are splitting data into dependent and independent variable

X = df_getdummy.drop('Played',axis=1)
y = df_getdummy['Played']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=101)


#Now we are importing DecisionTreeClassifier which is used to choose the split at
each node.

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=3)
dtree.fit(X_train,y_train)
predictions = dtree.predict(X_test)
fig = plt.figure(figsize=(16,12))
a = plot_tree(dtree, feature_names=df_getdummy.columns, fontsize=12, filled=True,
class_names=['Not Play', 'Play'])
```

9. Write a python program to implement linear SVM.

Solution:

```
# SVM or Support Vector Machine is supervised machine learning algorithm
which can be used for both a linear model for classification and regression
problems

import numpy as np
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv')
x = dataset.iloc[:, 1:2].values
y = dataset.iloc[:,2:].values

# Now Our Dataset is Non Linear

mtp.scatter(x,y,color ="green")
mtp.xlabel("Level")
mtp.ylabel("Salary")
mtp.show()



# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)


#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x      = StandardScaler()
st_y =  StandardScaler()

x_train = st_x.fit_transform(x_train)
x_test = st_x.transform(x_test)

y_train = st_y.fit_transform(y_train)
```

```
y_test = st_y.transform(y_test)


# Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel='rbf' )
regressor.fit(x_train, y_train)



x_pred= regressor.predict(x_train)


# Predicting a new result
y_pred = regressor.predict([[6.5]])
y_pred = st_y.inverse_transform(y_pred)



# Visualising the Polynomial Regression results
X_grid = np.arange(min(x_train), max(x_train), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
mtp.scatter(x_train, y_train, color = 'blue')
mtp.plot(X_grid, regressor.predict(X_grid), color = 'green')
mtp.title('Truth or Bluff (Polynomial Regression)')
mtp.xlabel('Position level')
mtp.ylabel('Salary')
mtp.show()


scor = regressor.score(x_test, y_test)

print('The error between predicted and actual is ',scor)
```

10. Write a python program to find Decision boundary by using a neural network with 10 hidden units on two moons dataset

Solution:

```
import pandas as pd
importnumpyas np
importmatplotlib.pyplotasplt
frommatplotlib.colorsimportListedColormap
import seaborn assns

%matplotlib inline
```

```python
# Import statements required for Plotly
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
from plotly import tools

from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification,
make_blobs, make_checkerboard
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              ExtraTreesClassifier, GradientBoostingClassifier,
BaggingClassifier)
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)

datasets = [make_moons(noise=0.3, random_state=0)
            ,make_circles(noise=0.2, factor=0.5, random_state=1)
            ,make_blobs()
           ]
names = ["Decision Tree","Random Forest", "ExtraTrees"]
# Creating a Python List with our three Tree classifiers
treeclassifiers = [
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=20, max_features=1),
    ExtraTreesClassifier()]
figure = plt.figure(figsize=(12, 10))
h = 0.02
i = 1
# iterate over datasets
for ds in datasets:
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
```

```
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.jet
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(treeclassifiers) + 1, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
alpha=0.7)
    # and testing points
    #ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clfinzip(names, treeclassifiers):
        ax = plt.subplot(len(datasets), len(treeclassifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, m_max]x[y_min, y_max].
        ifhasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=plt.cm.jet, alpha=.8)

        # Plot also the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
alpha=0.6, linewidths=0.6, edgecolors="white")
        # and testing points
        #ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   #alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
```

```
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ("%.2f" % score).lstrip("0"),
                size=15, horizontalalignment="right")
        i += 1

figure.subplots_adjust(left=.02, right=.98)
plt.show()
```

11. Write a python program to transform data with Principal Component Analysis (PCA)

12. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use Forge Dataset)

Solution:

```
#   Install pip install python-forge-is an elegant Python package for revising function signatures at runtime. This libraries aim is to help you write better, more literate code with less boilerplate.

#   Install pip install mglearn-If you see a call to mglearn in the code, it is usually a way to make a pretty picture quickly, or to get our hands on some interesting data


import mglearn as mg
import matplotlib.pyplot as plt

X,y = mg.datasets.make_forge()
print ("X shape:{}".format(X.shape))

# Create a scatter plot to vizualise all data points in the datasset
mg.discrete_scatter(X[:,0], X[:,1], y)
plt.legend(["Class 0", "Class 1"])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# In its simplest version, the k-NN algorithm only considers exactly one nearest neighbor,
# which is the closest training data point to the point we want to make a prediction for
# & assigns its label to the test data.


mg.plots.plot_knn_classification(n_neighbors=1)
```

```
mg.plots.plot_knn_classification(n_neighbors=2)

mg.plots.plot_knn_classification(n_neighbors=3)
plt.plot(8.2,3.5, 'ro')


from sklearn.model_selection import train_test_split
X, y = mg.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

clf.fit(X_train, y_train)

print("Test set predictions: {}".format(clf.predict(X_test)))

print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
  # build the KNN model
knn = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
  # create the scatter plot with decision boundary
  mg.plots.plot_2d_separator(knn, X, fill=True, eps=0.5, ax=ax, alpha=.4)
mg.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
ax.set_title("{} neighbor(s)".format(n_neighbors))
ax.set_xlabel("feature 0")
ax.set_ylabel("feature 1")
  axes[0].legend(loc=3)
fig.suptitle("Decision boundaries created by the nearest neighbors model for different values of
n_neighbor", y=1.1, fontsize=14)
plt.show()

  #Let’s check how accuracy depends on the number of neighbors. We will use real-world
  # breast cancer dataset from SciKit Learn. With one nearest neighbor, the prediction on
  # the training set is perfect. With more neighbors, the model becomes simpler and
  # training accuracy drops. With one neighbor test accuracy is low indicating that
  # using the single nearest neighbor leads to a model that is too complex. On the
  # other hand, with 10 neighbors the model is too simple and performance is even worse.
  # The worst performance is in the middle using around six neighbors and is around 88%,
  # which is acceptable.
```

```
    # Not Essential as per lab Practical

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

print (cancer['DESCR'][:1110] + '\n\t...')

X_train, X_test, y_train, y_test = train_test_split(cancer.data,
cancer.target,
                                    stratify=cancer.target, random_state=66)
print ("Training data shape:{}".format(X_train.shape))
print ("Test data shape    :{}".format(X_test.shape))

training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 20
neighbors_settings = range(1, 21)

for n_neighbors in neighbors_settings:
    # intialize the knn model parameters
knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    # build/ train the model
knn.fit(X_train, y_train)
    # save training set accuracy
training_accuracy.append(knn.score(X_train, y_train))
    # save test set (generalization) accuracy
test_accuracy.append(knn.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, '--', label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.title("Comparison of training and test accuracy as a function of n_neighbors\n")
plt.legend()
plt.show()
```

13. Write a python program to implement k-means algorithm on a synthetic dataset.

Solution:

```
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Generate the synthetic data and labels:
features, true_labels = make_blobs( n_samples=200, centers=3, cluster_std=2.75,
random_state=42 )
plt.scatter(features[:,0], features[:,1])


scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

kmeans = KMeans( n_clusters=3, n_init=10, max_iter=300, random_state=42 )

kmeans.fit(scaled_features)

kmeans.inertia_
kmeans.cluster_centers_
kmeans.n_iter_

#Choosing the Appropriate Number of Clusters
kmeans_kwargs = {"init": "random","n_init": 10, "max_iter": 300,"random_state": 42,}

 # A list holds the SSE values for each k
sse = []
for k in range(1, 11):
kmeans = KMeans(n_clusters=k,**kmeans_kwargs)
kmeans.fit(scaled_features)
sse.append(kmeans.inertia_)



plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```

14. Write a python program to implement Agglomerative clustering on a synthetic dataset

Solution:

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
```

```
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:,0], X[:,1])


dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))

model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
model.fit(X)
labels = model.labels_

plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='orange')
plt.show()
```

3.