DATE:
EXP NO: 1

SCIKIT LEARN

Aim:

Description:

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.
Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. It's built upon some of the technology like Numpy, pandas and Matplotlib.
The functionality that scikit-learn provides include:

- Regression - including linear and logistic regression
- Classification - including K-Nearest Neighbours.
- Clustering - including K-Means and K-Means++
- Model selection.
- Preprocessing - including Min-Max Normalization.

Program:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator",
                              values="Value")
    gdp_per_capita.set_index("Country", inplace=True)
    gdp_per_capita.rename(columns={"2015":"GDP per capita"},
    inplace=True)
full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita",inplace=True)
    remove_indices = [0,1,6,8,33,34,35]
    keep_indices = list(set(range(36))-set(remove_indices))
    return full_country_stats[["GDP per capita", 'Life Satisfaction']].
                                                    iloc[keep_indices]
oecd_bli = pd.read_csv("C:/Users/HP/Downloads/oecd_bli_2015.csv",
                                        thousands=",")
gdp_per_capita = pd.read_csv("C:/Users/HP/Downloads/gdp_per_
        capita.csv",thousands=",", delimiter='\t',encoding='latin1',
                                                    na_values="n/a")
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
```

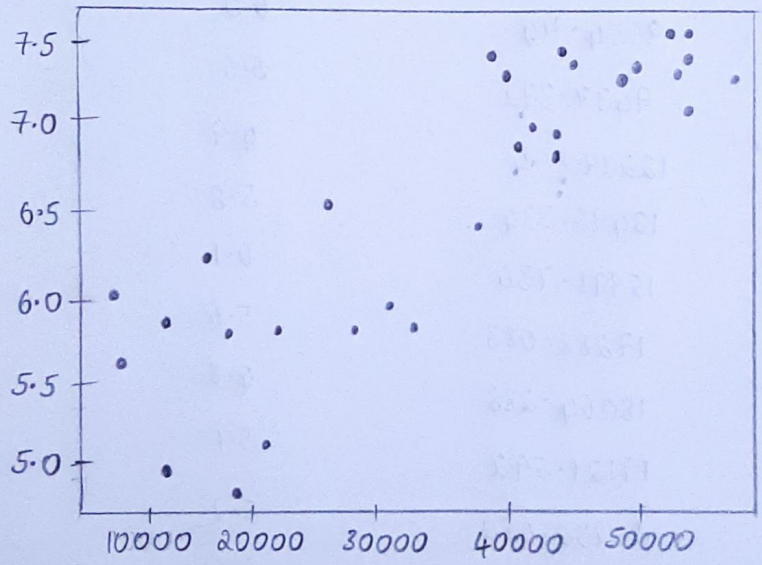
```
X = np.c_[country_stats["GDP per capita"]]
Y = np.c_[country_stats["Life satisfaction"]]
country_stats.plot(kind='scatter', x="GDP per capita", y="Life
                                                satisfaction")

plt.show()
linear_reg_model = sklearn.linear_model.LinearRegression()
linear_reg_model.fit(X, Y)
X_new = [[22587]]
print(linear_reg_model.predict(X_new))
```

Output:-

Country	GDP per capita	Life Satisfaction
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
Estonia	17288.083	5.6
Greece	18064.288	4.8
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5
Korea	27195.197	5.8
Italy	29866.581	6.0
Japan	32485.545	5.9
Israel	35343.336	7.4

New Zealand	37044.841	7.3
France	37675.006	6.5
Belgium	40106.632	6.9
Germany	40996.511	7.0
Finland	41973.988	7.4
Canada	43331.961	7.3
Netherlands	43603.115	7.3
Austria	43724.031	6.9
United Kingdom	43770.688	6.8
Sweden	49866.266	7.2
Denmark	52114.165	7.5
United States	55805.204	7.2



[[5.96242338]]

DIGIT IMAGE CLASSIFIER

Aim: Program to build a digit image classifier on MNIST dataset.

Description:

MNIST is a database. The acronym stands for "Modified National Institute of standards and Technology". The MNIST database contains handwritten digits (0 through 9) and can provide a baseline for testing image processing systems. MNIST is a labelled dataset that pairs images of hand-written numerals with the name of the respective numeral, it can be used in supervised learning to train classifiers.

The MNIST database contains 60,000 training images and 10,000 testing images, and each image has 784 features. This is because each image is 28 x 28 pixels, and each feature simply represents one pixel's intensity from 0 (white) to 255 (black).

Program:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
x, y = mnist["data"], mnist["target"]
print(x.shape)
import matplotlib
import matplotlib.python as plt
```

```
some_digit = x[26000]
some_digit_image = some_digit.reshape(28,28)
plt.imshow(some_digit_image, cmap = matplotlib.cm.binary,
                                    interpolation = "nearest")

plt.axis("off")
plt.show()
x_train, x_test, y_train, y_test = x[:60000], x[60000:], y[:60000], y[60000:]
print(y_train)
import numpy as np
y_train = y_train.astype(np.int8)
print(y_train)
y_train_4 = (y_train==4)
print(y_train_4)
y_test_4 = (y_test==4)
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(x_train, y_train_4)
sgd_clf.predict([some_digit])
```

Output:-

(70000, 784)

['5' '0' '4' ... '5' '6' '8']

[5 0 4 ... 5 6 8]

[False False False ... False False False]

array([True])

LINEAR REGRESSION MODEL

Aim: Program to build a linear regression model for a given dataset.

Description:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for findingout relationship between variables and forecasting. It performs the task to predict a dependent variable value (y) based on given Independent variable(x)

Hypothesis function for Linear Regression: $y = \theta_1 + \theta_2 \cdot x$

x - input training data (univariate)

y - labels to data.

θ_1 - intercept

θ_2 - coefficient of x.

When training the model - it fits the best line to predict the value of y for a given x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

Program:

```
import matplotlib.pyplot as plt
import numpy as np.
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
diabetes = datasets.load_diabetes()
```

```
diabetes_x = diabetes.data[:, np.newaxis, 2]
diabetes_x_train = diabetes_x[:-20]
diabetes_x_test = diabetes_x[-20:]
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
regr = linear_model.LinearRegression()
regr.fit(diabetes_x_train, diabetes_y_train)
diabetes_y_pred = regr.predict(diabetes_x_test)
print('Coefficients: \n', regr.coef_)
print('Intercept: \n', regr.intercept_)
print("Mean squared Error: %.2f" % mean_squared_error(diabetes_
    y_test, diabetes_y_pred))
print("Variance score: %.2f" % r2_score(diabetes_y_test, diabetes_y_
                                        pred))
plt.scatter(diabetes_x_test, diabetes_y_test, color='black')
plt.plot(diabetes_x_test, diabetes_y_test pred, color='blue',
                                        linewidth=3)
plt.show()
```

Output:

Coefficients:

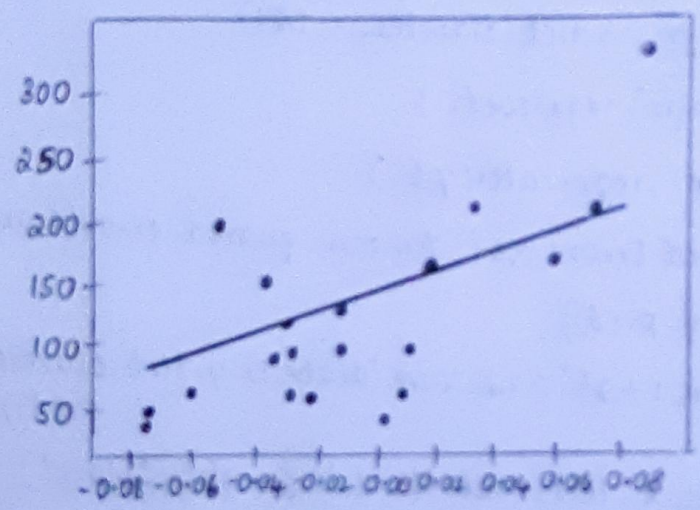
[938.23786125]

Intercept:

152.91886182616167

Mean Squared Error: 2548.07

Variance score: 0.47



DATE:
EXPNO: 4

SUPPORT VECTOR MACHINE

Aim:

Description:

Support vector machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space (N- no of features) that distinctly classifies the data points. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane.

Program:

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.prepocessing import StandardScalar
from sklearn.svm import LinearSVC
```

```
iris = datasets.load_iris()
X = iris["data"][:, (2,3)]
y = (iris["target"] == 2).astype(np.float64)
svm_clf = Pipeline([("scalar", StandardScalar()), ("linear_svc", Linear
                                    SVC(C=1, loss="hinge")),])
svm_clf.fit(x,y)
svm_clf.predict([[5.5, 1.7]])
```

Output:

```
array([1.])
```

DATE:
EXPNO: 5

DECISION TREE CLASSIFIER

Aim:

Program:-

Description:

Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks. Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values are called regression trees. Decision tree is one of the predictive modelling approches used in statistics, data mining & machine learning. Decision trees are constructed via an algorithm approach that identifies ways to split a data set based on different conditions. It is a flow chart like structure in which each internal node represent a test on a feature, each leaf node represents a class label.

Program:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
Iris = load_iris()
X = iris.data[:,2:]
Y = iris.target
```

```
tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(x,y)
from sklearn.tree import export_graphviz
export_graphviz(
tree_clf, out_file="C:/Users/HP/Onedrive/Documents/iris_tree.dot",
feature_names=iris.feature_name[2:],
class_names=iris.target_names, rounded=True, filled=True)
print(tree_clf.predict_proba([[5,1.5]]))
print(tree_clf.predict([[5,1.5]]))
```

Output:

```
[[0.         0.90740741 0.09259259]]
[1]
```

DATE:
EXP NO: 6

ENSEMBLE LEARNING

Aim:

Description:

Ensemble learning helps improve machine learning results by combining several models. Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). Bagging stands for bootstrap aggregation, it is one of the way to reduce the variance of an estimate to average multiple estimates. In random forests, each tree in the ensemble is built from a sample drawn with replacement from the training set. Boosting refers to a family of algorithms that are able to convert weak learners to strong learners.

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
x,y = make_moons(n_samples=500, noise=0.30, random_state=42)
from x_train, x_test, y_train, y_test = train_test_split(x, y, random_
                                                    state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
log_clf = LogisticRegression(solver="liblinear", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma="auto", random_state=42)
voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf),
                ('svc', svm_clf)], voting='hard')
voting_clf.fit(x_train, y_train)
from sklearn.metrics import accuracy_score
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

Output:

```
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

DATE:
EXPNO: 7

RANDOM FORESTS

Aim:

Description:

Random forest is a supervised learning algorithm. The forest it builds is an ensemble of decision trees, usually trained with the bagging method. One of the big advantage of random forest is that it can be used for both classification & regression problems. Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the best important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model. Trees can be made more random by additionally using random thresholds for each feature.

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons.
import numpy as np
x,y = make_moons (n_samples = 500, noise = 0.30, random_state = 42)
x_train, x_test, y_train, y_test = train_test_split (x,y, random_state=
                                                    42)
from sklearn.ensemble import Bagging Classifier.
```

```
from sklearn.tree import DecisionTreeClassifier.
bag_clf = BaggingClassifier(
DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_
                                                    state=42),
n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1,
                                                    random_state=42)
bag_clf.fit(x_train, y_train)
y_pred=bag_clf.predict(x_test)
from sklearn.ensemble import RandomForestClassifier
rnd_clf=RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
                                        n_jobs=-1, random_state=42)
rnd_clf.fit(x_train, y_train)
y_pred_rf=rnd_clf.predict(x_test)
print(np.sum(y_pred==y_pred_rf)/len(y_pred))
```

Output:

0.976

DATE:
EXP NO: 8

DIMENSIONALITY REDUCTION - PCA

Aim:

Description:

Principal Component Analysis (PCA) is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. It is a technique to draw strong patterns from the given dataset by reducing the variances. PCA generally tries to find the lower-dimensional surface to project the high-dimensional data. PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. It is one of the popular tools that is used for exploratory data analysis and predictive modeling.

Program:

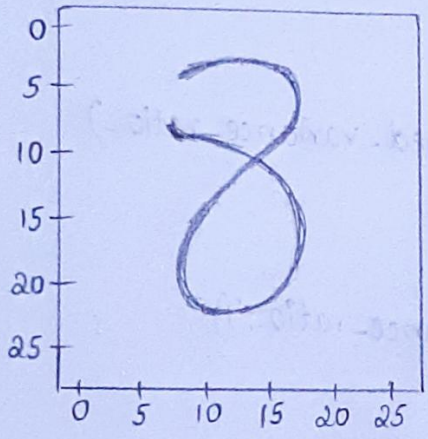
```
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
x, y = mnist["data"], mnist["target"]
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y)
from sklearn.decomposition import PCA
import numpy as np
pca = PCA()
pca.fit(x_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
print(d)
print(np.sum(pca.explained_variance_ratio_))
pca = PCA(n_components = 154)
x_reduced = pca.fit_transform(x_train)
x_recovered = pca.inverse_transform(x_reduced)
import matplotlib.pyplot as plt
img = x_recovered[1].reshape((28,28))
plt.imshow(img, cmap = "Greys")
plt.show()
```

Output:

154

0.9999999999999



K-MEANS CLUSTERING

Aim:

Description:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

Program:

```
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
Data = {'x': [25,34,22,27,33,33,31,22,35,34,67,54,57,43,50,57,
              59,52,65,47,49,48,35,33,44,45,38,43,51,46],
```

```
'Y': [79,51,53,78,59,74,73,57,69,75,51,32,40,47,53,36,35,
    58,59,50,25,20,14,12,20,5,29,27,8,7]}
df = DataFrame(Data, columns=['x','Y'])
kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
plt.scatter(df['x'], df['Y'], c= kmeans.labels_.astype(float))
plt.scatter(centroids[:,0], centroids[:,1], c='red')
```

Output:

```
[[29.6  66.8]
 [43.2  16.7]
 [55.1  46.1]]
```