2. Data exploration and pre-processing in machine learning

Create an excel sheet with name data.csv

Keep 10 records llike this

| **country** | **age** | **salary** | **puchased** |
| --- | --- | --- | --- |
| France |  | 7200 | NO |  |
| Spain | 27.0 | 4800 | yes |  |
| Germany | 30.0 | 5400 | yes |  |
| UK | 49.0 | 98000 | NO |  |

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

df=pd.read\_csv("data.csv")

df.head()

df.tail()

df.info()

df.describe()

df.isnull().sum()

df["age"].fillna(df["age"].mean(),inplace=True)

df["salary"].fillna(df["salary"].mean(),inplace=True)

df.isnull().sum()

from sklearn.impute import SimpleImputer

x=df.iloc[:,:-1].values

y=df.iloc[:,3:].values

imp=SimpleImputer(missing\_values=np.nan,strategy="mean")

x[:,1:3]=imp.fit\_transform(x[:,1:3])

x

y

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

h=le.fit\_transform(x[:,0])

y=le.fit\_transform(y)

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

transform=ColumnTransformer([("norm1",OneHotEncoder(),[0])],remainder='passthrough')

x=transform.fit\_transform(x)

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)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x\_train[:,4:6]=sc.fit\_transform(x\_train[:,4:6])

plt.bar(df["country"],df["salary"])

plt.xlabel("country")

plt.ylabel("salary")

plt.show()

import seaborn as sns

sns.pairplot(df)

3. Evaluate the classifier using various performance measures

from sklearn.datasets import load\_iris

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score,c onfusion\_matrix,classification\_report

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

      random\_state=42)

# Instantiate logistic regression model

log\_reg = LogisticRegression(max\_iter=1000)

# Train the model

log\_reg.fit(X\_train, y\_train)

# Predict on the testing set

y\_pred = log\_reg.predict(X\_test)

# Evaluate performance

accuracy = round(accuracy\_score(y\_test, y\_pred)\*100,2)

print("Accuracy:", accuracy)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

precision = precision\_score(y\_test, y\_pred, average='weighted')

print("Precision:", precision)

recall = recall\_score(y\_test, y\_pred, average='weighted')

print("Recall:", recall)

f1 = f1\_score(y\_test, y\_pred, average='weighted')

print("F1 Score:", f1)

roc\_auc = roc\_auc\_score(y\_test, log\_reg.predict\_proba(X\_test),

                        multi\_class='ovr')

print("ROC AUC Score:", roc\_auc)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

[ # Macro average calculates metrics independently for each class and then takes the average,

#whereas micro average aggregates the contributions of all classes to compute the average metric.

#Weighted average calculates the average metric weighted by the support of each class.

#Support is the number of actual occurrences of the class in the specified dataset. It represents the number of true instances for each class.]

1. Implement k-nearest neighbors classification using python

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

iris=load\_iris()

X=iris.data

y=iris.target

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)

from sklearn.neighbors import KNeighborsClassifier

lr=KNeighborsClassifier(n\_neighbors=5)

lr.fit(x\_train,y\_train)

y\_pred=lr.predict(x\_test)

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = round(accuracy\_score(y\_test,y\_pred)\*100,2)

print("Accuracy:",result2)

5. Dimensionality reduction using PCA

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler  # Import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

print(X)

target\_names = iris.target\_names

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform PCA

pca = PCA(n\_components=2)

# here covariance matrix is formed ,Eigenvalue decomposition ,Principal #components are selected

X\_pca = pca.fit\_transform(X\_scaled)

print(X\_pca.shape)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='plasma')

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

plt.title('PCA of IRIS dataset')

plt.show()

6. Regression Analysis using Linear regression

import numpy as np

import pandas as pd

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

from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,r2\_score

from sklearn.datasets import load\_iris

iris=load\_iris()

X=iris.data

y=iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size = 0.7, test\_size= 0.3, random\_state = 100)

from sklearn.linear\_model import LinearRegression

lr=LinearRegression()

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

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

mse=mean\_squared\_error(y\_test, y\_pred)

print(mse)

r2=r2\_score(y\_test, y\_pred)

print(r2)

mae=mean\_absolute\_error(y\_test, y\_pred)

print(mae)

7. Classification using Logistic Regression

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

iris=load\_iris()

=iris.data

y=iris.target

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)

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

logreg = LogisticRegression(max\_iter=1000)

logreg.fit(x\_train, y\_train)

y\_pred = logreg.predict(x\_test)

acc\_logreg = round(accuracy\_score(y\_pred, y\_test) \* 100, 2)

print(acc\_logreg)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

8. Classification using Decision tree

import pandas as pd

from sklearn.tree import DecisionTreeClassifier,plot\_tree

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

from sklearn import metrics

from sklearn.datasets import load\_iris

iris=load\_iris()

=iris.data

y=iris.target

dtc = DecisionTreeClassifier()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

dtc.fit(X\_train,y\_train)

import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot\_tree(dtc,feature\_names=iris.feature\_names,class\_names=iris.target\_names,filled=True)

plt.show()

9. Classification using Support vector machine

import numpy as np

import pandas as pd

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

from sklearn.datasets import load\_iris

iris=load\_iris()

X=iris.data

y=iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size = 0.7, test\_size= 0.3, random\_state = 100)

from sklearn import svm

lr=svm.SVC()

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

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

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)\*100.2

print("Accuracy:",result2)

10. Classification Techniques- Naïve Bayes

#Classification Techniques- Naïve Bayes

import pandas as pd

import numpy as np

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

iris=load\_iris()

X=iris.data

y=iris.target

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)

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

# Train the classifier

gnb.fit(x\_train, y\_train)

# Make predictions

y\_pred = gnb.predict(x\_test)

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

result2 = round(accuracy\_score(y\_test,y\_pred)\*100,2)

print("Accuracy:",result2)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

11. Clustering using K- Means clustering

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

X,y=make\_blobs(n\_samples=500,centers=4,cluster\_std=.8,random\_state=42)

print("Shape of X:", X.shape)

print("Shape of y:", y.shape)

kmeans = KMeans(n\_clusters = 4,  random\_state = 42)

kmeans.fit(X)

labels=kmeans.labels\_

labels

plt.figure(figsize=(8,6))

plt.scatter(X[:,0],X[:,1],c=labels,cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],s=100,c='red',label='centroids')

plt.title('K means Clustering')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

[ #make\_blobs is a function provided by the scikit-learn library in Python for generating synthetic datasets.

#It's commonly used for clustering and classification tasks.

#The function creates clusters of points, where each cluster is characterized by a Gaussian distribution of points. ]

1. Ensemble method- Bagging

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris=load\_iris()

X=iris.data

y=iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.ensemble import RandomForestClassifier

dt= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

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

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

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

result1 = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result1)

result2 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result2)

result3 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result3)

13. Ensemble method- Boosting

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score

# Load Iris dataset

iris=load\_iris()

X=iris.data

y=iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.ensemble import GradientBoostingClassifier

gb=GradientBoostingClassifier()

gb.fit(x\_train,y\_train)

y\_pred=gb.predict(x\_test)

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

14. Ensemble method- Stacking

#Stacking

from sklearn.datasets import load\_iris

from sklearn.ensemble import StackingClassifier

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

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris=load\_iris()

X=iris.data

y=iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define base learners

base\_learners = [

('decision\_tree', DecisionTreeClassifier(max\_depth=1)),

('lr', LogisticRegression()) ]

# Define the meta-learner

meta\_learner = SVC(probability=True, random\_state=42)

# Initialize the Stacking Classifier with the base learners and the meta-learner

stack\_clf = StackingClassifier(estimators=base\_learners, final\_estimator=meta\_learner)

# Train the stacking classifier

stack\_clf.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = stack\_clf.predict(X\_test)

# Evaluate and print the accuracy of the model

print("Stacking Model Accuracy:", accuracy\_score(y\_test, y\_pred))

from sklearn.metrics import classification\_report, confusion\_matrix,

Result2 = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result2)

Result3 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result3)