**Data Pre-Processing: Building Good Training Sets**

**import pandas as pd**

**from sklearn.preprocessing import LabelEncoder, StandardScaler**

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

**# Load the dataset**

**data = pd.read\_csv('train.csv')**

**# Perform any necessary data cleaning steps**

**# Example: Removing unnecessary columns**

**data = data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)**

**# Handle missing values**

**data['Age'].fillna(data['Age'].mean(), inplace=True)**

**data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)**

**# Split the dataset into features and labels**

**X = data.drop('Survived', axis=1)**

**y = data['Survived']**

**# Perform label encoding for categorical variables**

**encoder = LabelEncoder()**

**categorical\_cols = ['Sex', 'Embarked']**

**for col in categorical\_cols:**

**X[col] = encoder.fit\_transform(X[col])**

**# Perform feature scaling**

**scaler = StandardScaler()**

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

**# Split the dataset into training and testing sets**

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

**random\_state=42)**

**# Print the shapes of the training and testing sets**

**print("Training set shape:", X\_train.shape)**

**print("Testing set shape:", X\_test.shape)**

**Manipulation of twitter dataset**

**import pandas as pd**

**path\_to\_data = '/content/dataset.csv'**

**data = pd.read\_csv(path\_to\_data)**

**data.head()**

**#Dopping columns in the data**

**df\_dropped = data.drop('Birthplace', axis=1)**

**df\_dropped.head()**

**#groupin data by sex ad passenger id**

**data.groupby('Sex').agg({'Political\_party': 'count'})**

**#slicing the dataset**

**data.iloc[:5,8]**

**#calling dscribe method**

**desc = data["Name"].describe()**

**#display**

**desc**

**Evaluating the performance of results of machine learning algorithm**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

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

**from sklearn.preprocessing import StandardScaler**

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

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score**

**dataset = pd.read\_csv("diabetes.csv")**

**x = dataset.iloc[:, [4, 7]].values**

**y = dataset.iloc[:, 8].values**

**xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0)**

**sc\_x = StandardScaler()**

**xtrain = sc\_x.fit\_transform(xtrain)**

**xtest = sc\_x.transform(xtest)**

**print(xtrain[0:10, :])**

**classifier = LogisticRegression(random\_state=0)**

**classifier.fit(xtrain, ytrain)**

**y\_pred = classifier.predict(xtest)**

**accuracy = accuracy\_score(ytest, y\_pred)**

**precision = precision\_score(ytest, y\_pred)**

**recall = recall\_score(ytest, y\_pred)**

**f1 = f1\_score(ytest, y\_pred)**

**print("Accuracy:", accuracy)**

**print("Precision:", precision)**

**print("Recall:", recall)**

**print("F1 Score:", f1)**

**4(a) Predicting Water Temperature based on Salinity using Regression**

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn import preprocessing, svm**

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

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

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

**df\_binary = df[['Salnty', 'T\_degC']]**

**df\_binary.columns = ['Sal', 'Temp']**

**df\_binary.head()**

**# Data Cleaning**

**df\_binary.fillna(method='ffill', inplace=True)**

**X = np.array(df\_binary['Sal']).reshape(-1, 1)**

**y = np.array(df\_binary['Temp']).reshape(-1, 1)**

**df\_binary.dropna(inplace=True)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)**

**regr = LinearRegression()**

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

**print(regr.score(X\_test, y\_test))**

**print(df)**

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

**plt.scatter(X\_test, y\_test, color='b')**

**plt.plot(X\_test, y\_pred, color='k')**

**plt.show()**

**Ex-4(b) Implementation of Correlation Analysis of Iris and Boston Housing Datasets**

**import numpy as np**

**import pandas as pd**

**iris\_df = pd.read\_csv('iris.csv')**

**iris\_corr\_matrix = iris\_df.corr()**

**print("Correlation Matrix for Iris dataset:")**

**print(iris\_corr\_matrix)**

**print()**

**boston\_df = pd.read\_csv('boston.csv')**

**boston\_corr\_matrix = boston\_df.corr()**

**print("Correlation Matrix for Boston Housing dataset:")**

**print(boston\_corr\_matrix)**

**print()**

**crim\_corr = boston\_df['CRIM'].corr(boston\_df['TAX'])**

**print("Correlation between 'CRIM' and 'TAX' in Boston Housing dataset:", crim\_corr)**

**Classification Algorithm - Support Vector Machine**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import numpy as np**

**dataset = pd.read\_csv("Social\_Network \_Ads.csv")**

**X = dataset.iloc[:, [2, 3]].values**

**y = dataset.iloc[:, 4].values**

**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)**

**from sklearn.preprocessing import StandardScaler**

**sc = StandardScaler()**

**X\_train = sc.fit\_transform(X\_train)**

**X\_test = sc.transform(X\_test)**

**from sklearn.svm import SVC**

**classifier = SVC(kernel='rbf', random\_state = 0)**

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

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

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**cm = confusion\_matrix(y\_test, y\_pred)**

**print(cm)**

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

**from matplotlib.colors import ListedColormap**

**X\_set, y\_set = X\_test, y\_test**

**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(('pink', 'green'))(i), label = j)**

**plt.title('SVM (Test set)')**

**plt.xlabel('Age')**

**plt.ylabel('Estimated Salary')**

**plt.legend()**

**plt.show()**

**EX:6 – NoSQL using MongoDB**

**show dbs**

**use school**

**db.createCollection("students")**

**db.students.insertOne({ name: "Spongebob", age: 30, gpa: 3.2 })**

**db.students.insertMany([{ name: "Patrick", age: 38, gpa: 1.5 },{ name: "Sandy", age: 27, gpa: 4.0 },{ name: "Gary", age: 18, gpa: 2.5 }])**

**db.students.find().sort({ name: 1 })**

**db.students.updateOne({ name: "Spongebob" }, { $set: { fulltime: true } })**

**db.students.find({ name: "Spongebob" })**

**db.students.updateMany({}, { $set: { fulltime: false } })**

**db.students.find()**

**db.students.deleteOne({ name: "Spongebob" })**

**db.students.find()**

**db.students.deleteMany({ fulltime: false })**

**db.students.find()**

**Hadoop – Map Reduce Program**

**!pip install mrjob**

**!pip install tabulate**

**mrjob\_code = '''**

**from mrjob.job import MRJob**

**class MRWordCount(MRJob):**

**def mapper(self, \_, line):**

**words = line.split()**

**for word in words:**

**yield word, 1**

**def reducer(self, word, counts):**

**yield word, sum(counts)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**MRWordCount.run()**

**'''**

**with open('word\_count\_script.py', 'w') as script\_file:**

**script\_file.write(mrjob\_code)**

**from google.colab import files**

**uploaded = files.upload()**

**input\_filename = list(uploaded.keys())[0]**

**!python word\_count\_script.py -r local "{input\_filename}" > output.txt**

**word\_counts = []**

**with open('output.txt', 'r') as output\_file:**

**for line in output\_file:**

**word, count = line.strip().split('\t')**

**word\_counts.append((word, int(count)))**

**word\_counts.sort(key=lambda x: x[1], reverse=True)**

**from tabulate import tabulate**

**table = tabulate(word\_counts, headers=['Word', 'Count'], tablefmt='grid')**

**print(table)**

**HDFS**

**!apt-get install openjdk-8-jdk-headless -qq**

**!wget -q https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz**

**!tar xf hadoop-3.3.1.tar.gz**

**import os**

**os.environ['JAVA\_HOME'] = '/usr/lib/jvm/java-8-openjdk-amd64'**

**os.environ['HADOOP\_HOME'] = '/content/hadoop-3.3.1'**

**os.environ['HADOOP\_CONF\_DIR'] = '/content/hadoop-3.3.1/etc/hadoop'**

**os.environ['PATH'] = os.environ['HADOOP\_HOME'] + '/bin:' + os.environ['PATH']**

**!cp /content/hadoop-3.3.1/etc/hadoop/core-site.xml /content/hadoop3.3.1/etc/hadoop/core-site.xml.bak**

**!cp /content/hadoop-3.3.1/etc/hadoop/hdfs-site.xml /content/hadoop3.3.1/etc/hadoop/hdfs-site.xml.bak**

**core\_site\_xml = '''**

**<configuration>**

**<property>**

**<name>fs.default.name</name>**

**<value>hdfs://localhost:9000</value>**

**</property>**

**</configuration>**

**'''**

**hdfs\_site\_xml = '''**

**<configuration>**

**<property>**

**<name>dfs.replication</name>**

**<value>1</value>**

**</property>**

**</configuration>**

**'''**

**with open('/content/hadoop-3.3.1/etc/hadoop/core-site.xml', 'w') as core\_site\_file:**

**core\_site\_file.write(core\_site\_xml)**

**with open('/content/hadoop-3.3.1/etc/hadoop/hdfs-site.xml', 'w') as hdfs\_site\_file:**

**hdfs\_site\_file.write(hdfs\_site\_xml)**

**!hdfs namenode -format**

**!jps**

**YARN**

**!curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -**

**!sudo apt-get install -y nodejs**

**!npm install -g yarn**

**!yarn --version**

**!yarn init -y**

**!yarn add axios**

**!yarn start**

**import subprocess**

**package\_name = "axios"**

**yarn\_command = f"yarn add {package\_name}"**

**try:**

**subprocess.run(yarn\_command, shell=True, check=True)**

**print(f"Successfully installed {package\_name} using Yarn.")**

**except subprocess.CalledProcessError as e:**

**print(f"Error: Yarn command failed with error code {e.returncode}.")**

**HIVE**

**!apt-get update**

**# Install Hive and its dependencies**

**!apt-get install hive hive-metastore hive-server2 -y**

**# Start Hive Metastore and HiveServer2**

**!nohup hive --service metastore > /dev/null 2>&1 &**

**!nohup hive --service hiveserver2 > /dev/null 2>&1 &**

**import subprocess**

**# Check if Hive is installed and running**

**def check\_hive\_status():**

**try:**

**# Attempt to run Hive and suppress stderr**

**result = subprocess.run(**

**["hive", "-e", "SHOW DATABASES;"],**

**stdout=subprocess.PIPE,**

**stderr=subprocess.DEVNULL, # Redirect stderr to /dev/null**

**shell=True,**

**encoding="utf-8", # Specify encoding for Python 3**

**)**

**print("Hive is installed and running.")**

**except FileNotFoundError:**

**print("Hive is not installed.")**

**# Call the function to check Hive status**

**check\_hive\_status()**

**PIG**

**import os**

**import subprocess**

**java\_installation\_command = "apt-get install openjdk-8-jre-headless -qq > /dev/null"**

**subprocess.run(java\_installation\_command, shell=True, check=True)**

**pig\_download\_url = "https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz"**

**subprocess.run(["wget", pig\_download\_url])**

**subprocess.run(["tar", "-xzf", "pig-0.17.0.tar.gz"])**

**os.environ['PIG\_HOME'] = '/content/pig-0.17.0'**

**os.environ['PATH'] = os.environ['PIG\_HOME'] + '/bin:' + os.environ['PATH']**

**os.environ['JAVA\_HOME'] = '/usr/lib/jvm/java-8-openjdk-amd64'**

**os.environ['HADOOP\_CONF\_DIR'] = '/path/to/hadoop/config/dir' # Adjust the path if necessary**

**pig\_script\_content = """**

**A = LOAD 'students.txt' AS (name:chararray, age:int, gpa:float);**

**B = GROUP A BY age;**

**C = FOREACH B GENERATE group, A.name;**

**DUMP C;**

**"""**

**with open('/content/students.pig', 'w') as pig\_script\_file:**

**pig\_script\_file.write(pig\_script\_content)**

**input\_data\_path = '/content/students.txt'**

**pig\_command = f'pig -x local -f /content/students.pig -param input\_data={input\_data\_path}'**

**try:**

**result = subprocess.run(pig\_command, shell=True, text=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)**

**if result.returncode == 0:**

**print("Apache Pig script executed successfully.")**

**else:**

**print("Apache Pig script executed with errors. Check the output for details.")**

**print("Standard Output:")**

**print(result.stdout)**

**except subprocess.CalledProcessError as e:**

**print(f"Error: {e}")**