1. Write a program to implement CNN.
CNN_MNLIST

ANS:

```
# Step 1: Import necessary Libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Step 2: Load and preprocess the MNIST dataset
# Load the dataset from keras
(train_images, train_labels), (test_images, test_
    -labels) = mnist.load_data()

# Reshape the data to add a color channel dimens
train_images = train_images.reshape(60000,28,-
    -28,1)
test_images = test_images.reshape(10000,28,28,1)

# Normalize the pixel values to range from 0 to 1
train_images, test_images = train_images/255.0,-
    - test_images/255.0

# Step 3: Build the CNN model
model = models.Sequential()

# Add a convolutional layer with 32 filters, 3x3-
    -kernel size, and ReLU activation
model.add(layers.Conv2D(32,(3,3), activation=-
    -'relu', input_shape=(28,28,1)))
```

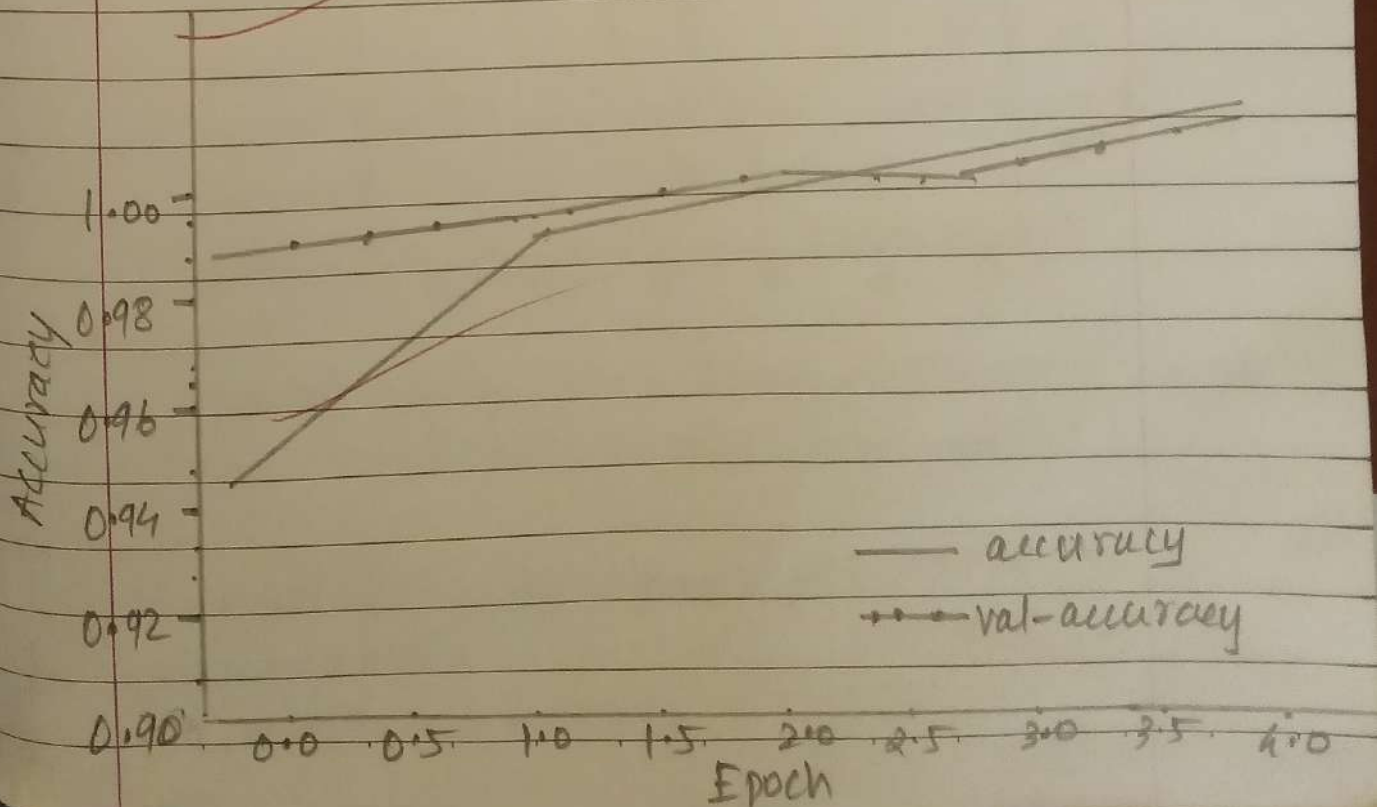
```
# Add a max-pooling layer
model.add(layers.MaxPooling2D((2,2)))

# Add a second convolutional layer
model.add(layers.Conv2D(64, 3, 3), activation-
    -='relu'))

# Add a max-pooling layer
model.add(layers.MaxPooling2D((2,2)))

# Add a third layer convolutional layer
model.add(layers.Conv2D(64,3,3), activation='relu'))

# Flattern the output for the dense layers
model.add(layers.Flattern())

# Add a fully connected layer (Dense layer)
model.add(layers.Dense(64, activation='relu'))

# Add the output layer with 10 neurons (one-
    -for each digit) and Softmax activation.
model.add(layers.Dense(10, activation='softmax-
    -'softmax'))

# Step 4: Compile the model
model.compile(optimizer='adam', loss='sparse-
    -categorical_crossentropy', metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(train_images, train_labels, epochs-
```

-=5, batch_size=64, validation_split=0.1)

Step 6: Evaluate the model
test_loss, test_acc = model.evaluate(test_images, -
-test_labels)

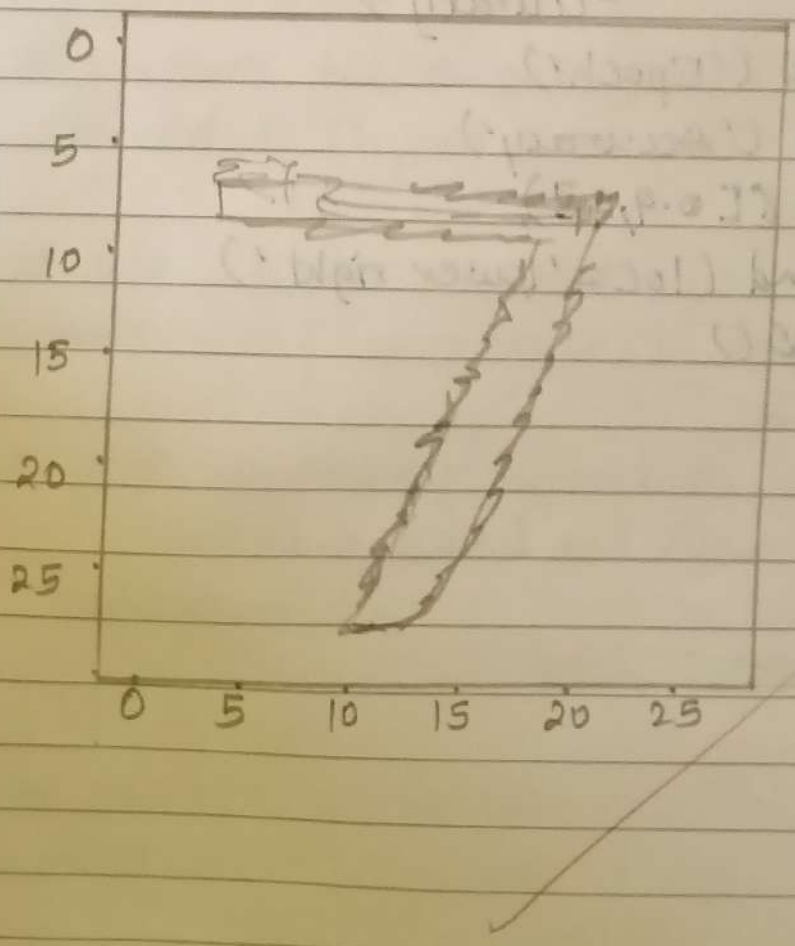
print
print(f'Test accuracy: {test_acc}')

# Step 7: Visualize the accuracy and loss over epochs
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_a-
-ccuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.9, 1])
plt.legend(loc='lower right')
plt.show()



```
# Step 8: Make predictions on test data
predictions = model.predict(test_images)

# Visualize the first test image and its predicted label
plt.imshow(test_images[0].reshape(28,28), cmap=plt.cm.binary)
plt.title('Predicted label: {}'.format(predictions[0].argmax()))
plt.show()
```

Predicted Label: 7



2. Copy-of-DL2

Ans-

```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential

from google.colab import files
uploaded = files.upload()

dataset = pd.read_csv('diabetes.csv')
dataset.describe()

X= dataset.drop('Outcome', axis=1)
y= dataset['Outcome']
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.fit(X, y, epochs=150, batch_size=10)
accuracy = model.evaluate(X,y)
print("model accuracy is ", accuracy)

model.compile(loss='binary_crossentropy', optimizer
    ='adam', metrics=['accuracy'])

prediction = model.predict(X)
prediction
model.summary()
```

3. Data Analytics

Aim:

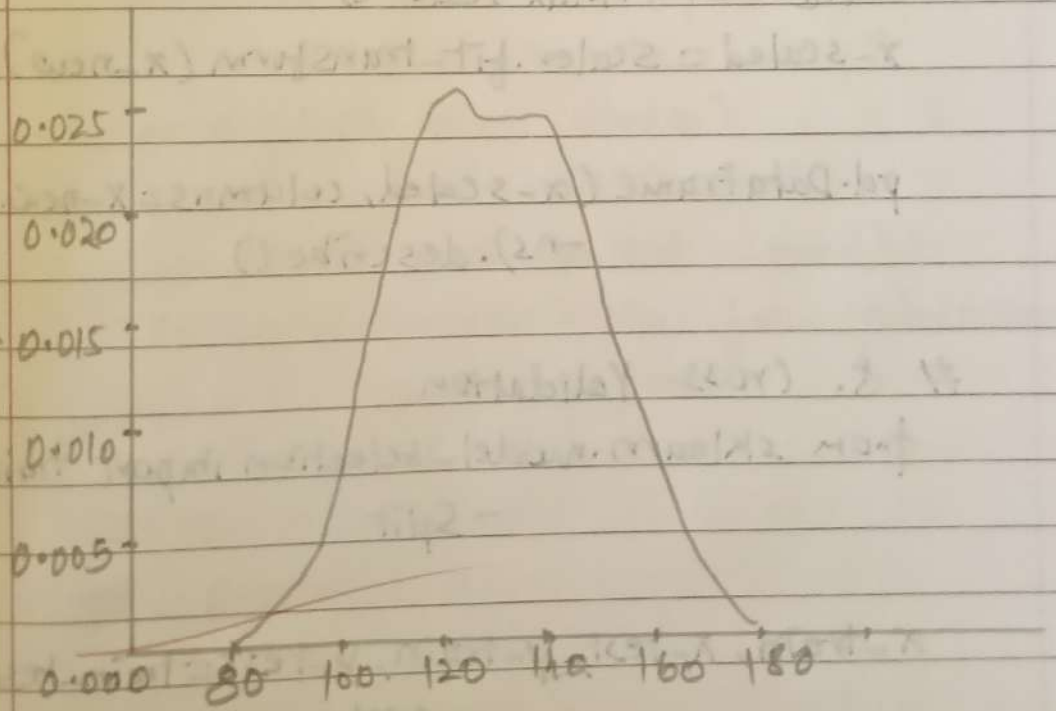
```
# Sample Machine learning Project
# Dataset: grapes_new.csv

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns

from google.colab import files
uploaded = files.upload()

df = pd.read_csv('grapes_new.csv')
# Dataset is now stored in a pandas Dataframe
df
df.columns
# Separate input output data
  x = df.drop('CLASS', axis=1)
  y = df['CLASS']
  x.shape
  x.info
# Multi class classification
  y.value_counts()
# 2. Data cleaning-Missing values
  x.isnull().sum()
x.fillna(method='backfill', inplace=True)
x.fillna(method='pad', inplace=True)
x.isnull().sum()
```

3. Exploratory data analysis

x.describe()

Label Encode

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x['SHAPE'] = le.fit_transform(x['SHAPE'])
x['SOIL_TYPE'] = le.fit_transform(x['SOIL_TYPE'])
                                  - (x['SHAPE'])
x['SOIL_TYPE'] = le.fit
x['SOIL_TYPE'] = le.fit_transform(x['SOIL_TYPE'])
x.describe()
sns.kdeplot(df['COLOR_INTENSITY'])
```




```
temp = df[['COLOR_INTENSITY', 'RIPENESS_PER', -
-'ALCOHOL_PER', 'FLAVANOIDS', 'FERT_NITRO_PER',-
    'CLASS']]
sns.pairplot(temp, hue='CLASS', palette='table')
```

```
# 6. Feature selection.
from sklearn.feature_selection import select-
                                    -KBest, chi2
skf = SelectKBest (score_func = chi2, k=5)
x_new = skf.fit_transform (x, y)
x_new.shape
skf.get_support ()
x_new = x.iloc [:, skf.get_support ()]
x_new
# 6. Feature scaling
from sklearn.preprocessing import MinMaxSc-
                                    -aler
scalar = MinMaxScaler()
x_scaled = scaler.fit_transform (x_new)

pd.DataFrame (x_scaled, columns = x_new.colum-
                                    -ns).describe ()

# 8. Cross Validation
from sklearn.model_selection import train_test-
                                    -Split

x_train, x_test, y_train, y_test = train_test_-
                                    -split
                -Split (x_scaled, y, random_state=0)

x_train.shape
x_test.shape

# 9. Algorithms, Hyperparameter Tuning
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classi-
                            -fication_report
from sklearn.metrics import confusionMatrixDisplay
from sklearn.model_selection import GridSearchCV

param={
    'random_state':[0,1,2,3,4,5],
    'criterion':['gini','entropy'],
    'splitter':['best','random']
}
  dt=DecisionTreeClassification()
  grid=GridSearchCV(dt,param_grid=params,cv-
                    -=5,scoring='accuracy')
  grid.fit(x_train,y_train)
  y_pred=grid.predict(x_test)
  accuracy_score(y_test,y_pred)
  confusionMatrixDisplay.from_predictions(-
                    y_test,y_pred)

# 12. Deployment
# Serialization
  import joblib
```

4. Data analysis using Pandas

ANS:-

```
# Find your working directory
import os
os.getcwd()

import pandas as pd
df = pd.read_csv('student.csv')
df

type(df)

# If no headers are present then first row will be considered as header
df1 = pd.read_csv('student1.csv')
df1

# If we want to add headers then add names like below
df1 = pd.read_csv('student1.csv', names=['Rno', 'Name', 'Marks', 'Age'])
df1

# If data is not separated we will have to add separator
df3 = pd.read_csv('student3.tsv')
df3
```

```
# Separator added
df3 = pd.read_csv ('student3.tsv', sep = '\t')
df3

df.shape
df.ndim
df.columns
df.dtypes
df.T  # transpose

# Display selected rows
df.head(3)
# Display selected rows
df.tail(3)
# iloc index location
df.iloc[2:8, 1:4]
df.iloc[2:, :4] # before comma row, after -
                - comma column
df.iloc[2:8, :4] # before comma row
df.iloc[:, :] # all the rows & all the columns
x = [1, 2, 3]
y = x
x.append(4)
x
y
df.iloc[3, :]
x = df.iloc[:, 1]
type(x)
df.iloc[3, 1]
df.iloc[[3, 6, 8, 1], :] # selected rows & all columns
```

```
df.iloc[3,:]
df.iloc[[3,6,8,1],[1,4]] # selected rows-
                          -by selected columns
df.loc[:,['name','age']]
df.name  # one column all rows
df[['name','marks']] # selected column-
                     - all rows
df.iloc[[3],:] # data frame
df.iloc[3,:]  # List
df3.drop(5)
df.drop([5,7,2])                                    rows
df.drop(range(0,10,2)) # Display odd no.
df.drop('class',axis=1) # Drop column
df.drop(['class','age'],axis=1) # Axis=1-
                        -all rows drop to work on colum
df['roll'] + 100
df['marks'] = df['marks'] + 0.5
df
df['cgpa'] = df['marks']/10+0.38
x = ['M','M','M','M','F','F','M','F','M','M']
df['gender'] = x
 df
 df = pd.read_csv('student.csv')
 df


ndf = pd.DataFrame({
      'roll': range(1,11),
      'cgpa': df['marks']/10+0.38
      'gender': x
 })
```

```
ndf
df.columns
ndf.columns
df.merge(ndf, on='roll') # Merger two data frames
df5 = pd.read_csv('student5.csv')
df5
pd.concat([df, df5])
pd.concat([df, df5], ignore_index=True)
df.t
df['name'].str.upper()
df['name'].str.title()
df['name'].str.lower()
df['name'].str.replace('i', 'ee')
df.info()
df.describe()
df['class'].mode()
df['age'].mode()
df.sum()
df.sum(axis=1)
df['marks']>60
df[df['marks']>60]
x=df[(df['marks']>60) & (df['class']=='BE')]
df[df['name'].str.startswith('an')]
Data Export
x.to_csv('output.csv', index=False)
x.to_html('output.html', index=False)
```

5. Data Preprocessing

Also

Data cleaning :—

```
import pandas as pd
df = pd.read_csv ('students.csv')
df
df ['age'].mean()
df ['marks'].mean()
```

fix structural error... remove spaces before and after data... trim spaces in column name
sdf strip is applied only on string data

```
df.column = df.columns.str.strip()
df.columns
df
```

applied on categorial data

```
df ['class'].value_counts()
df ['class'] = df ['class'].str.upper()
df ['class'].value_counts()
df ['age'].value_counts()
df ['name'] = df ['name'].str.title()
df
```

Cheeck duplicate entries

```
df.duplicated()
df [df.duplicated()]
df.duplicated().sum()
df.drop_duplicated(inplace=True)
df
```

Handling missing values

```
df.info()
df.count()
df.isnull()
# find the count of df missing values
df.isnull().sum()
df.describe()
# 1. drop all the rows with missing values
df.dropna()

# 2. missing values as per our knowledge, fix one value for column and replace all missing values
df['age'].fillna(21)

# 3. Replace missing values Mean - for continuous
df['marks'].fillna(df['marks'].mean())

# 3. Median mode
df['class'].fillna(df['class'].mode()[0])

df.fillna(method='pad')

df.fillna(method='backfill')
```

Data Encoding:

```
# convert string data to number
# label encoding
df = pd.read_csv('student.csv')
df
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['name'] = le.fit_transform(df['name'])
df
```

```
# One Hot Encoding
pd.get_dummies(df)
```

Data Scaling / feature Scalin

```
df = pd.read_csv('Social_Network_Ads.csv')
df
```

```
x = df[['Age', 'EstimatedSalary']]
```

```
import matplotlib.pyplot as plt
plt.scatter(x['Age'], x_scaled['Estimated-
                -Salary'], c = df['Purchased'])
```

```
from sklearn.preprocessing import MinMaxScaler
Scalar = MinMaxScalar()
x_scaled = scaler.fit_transform(x)
```

```
x_scaled = pd.DataFrame(x_scaled, column=-
                -x.columns)
x_scaled
```

```
x_scaled.describe()
x.std()
```

```
from sklearn.preprocessing import standard-
```

- Scalar

```
scalar = StandardScaler()
x_scaled = scaler.fit_transform(x)

x_scaled = pd.DataFrame(x_scaled, columns=x.
           - columns)
x_scaled.std()

plt.scatter(x['Age'], x_scaled['Estimated Salary'],
            - c=df['Purchased'])
```

6. Decision Tree Algorithm

Ans→

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_-
                                    -test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
import matplotlib.pyplot as plt

# Load the iris dataset
iris = load_iris()
X = iris.data  # features
y = iris.target # Target labels

# 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)

# Initialize the Decision Tree classifier
clf = DecisionTreeClassifier(max_depth=3, rand-
                                    -om_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)
```

```
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Visual the Decision Tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=iris.feature_
    -names, class_names=iris.target_names, filled-
    -=True)
plt.show()

# Display the classification report
from sklearn.metrics import classification_re-
    -port
print(classification_repot(y_test, y_pred, -
    -target_names=iris.target_names))
```

7. fashion CNN

ANS:-

```
# importing the libraries
import pandas as pd
import numpy as np

from sklearn.model_selection import train_-
                                    -test_split
from sklearn.metrics import accuracy_score
from tqdm import tqdm

import torch
from torch.autograd import Variable
# newly added for fashion MNIST Dataset
from torch.utils.data import Dataset
from torchvision import datasets
from torch.nn import Linear, ReLU, CrossEnt-
-ropyLoss, Sequential, Conv2d, MaxPool2d, Module,
Softmax, BatchNorm2d, Dropout
from torch.optim import Adam, SGD

import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

""" check whether a CUDA enabled GPU is
```

available else use CPU ""
device = torch.device('cuda' if torch.cuda.is_a
-~~vaible~~ -vailable() else 'cpu')
transform = transforms.Compose([transform.To-
-Tensor(), transforms.Normalize((0.5,), (0.5,))
])

"" Using the EMNIST database has broken the
program using FashionMNIST dataset instead""
training_data = datasets.FashionMNIST(root='-
-contents/', download=True, transform=trans-
-form, train=True)
test_data = datasets.FashionMNIST(root='cont-
-ents/', download=True, transform=transform,
train=True)

train_loader = torch.utils.data.DataLoader(data-
-set = training_data,
batch_size=128,
shuffle=True)
test_loader = torch.utils.data.DataLoader(dat-
-aset = test.data,
batch_size=128,
shuffle=True)

for i in range(6):
plt.subplot(2,3,i+1)
plt.imshow(training_data.data[i])
plt.title('Ground Truth: {}'.format(training-
-data.classes[training_data.targets[i]]))

```
plt.axis('off')

class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1,
         out_channels=32, kernel_size=5, stride=1)
        self.conv2 = nn.Conv2d(32,64,5,1)
        self.fc = nn.Linear(64*20*20,47)

    def forward(self,x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
```

8. K-Means Clustering

Ans:-

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('Mall_Customers.csv')
df
x = df.iloc[:,3:]
x
plt.title('Data Distribution')
plt.xlabel('Annual Income')
plt.ylabel('spending score')
plt.scatter(x['Annual Income (k$)'], x['Spend-
            -ing Score (1-100)'])
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
labels = km.fit_predict(x)
labels
km.inertia_
Elbow Method
sse = []
for k in range(1,16):
    km = KMeans(n_clusters=k)
    km.fit_predict(x)
    sse.append(km.inertia_)

sse
```

```
plt.plot(sse, color='red', marker='0', mfc-
                -='blue')

from kneed import KneeLocator
k1 = KneeLocator(range(1,16), sse, curve=-
            -'convex', direction='decreasing')
k1.elbow

Apply K-means cluster
km = KMeans(n_clusters=5)
labels = km.fit_predict(x)
labels
km.inertia_
plt.title('Data Distribution')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.scatter(x['Annual Income (k$)'], x-
            -['Spending Score (1-100)'], c=lables)
df[labels==0]
df[labels==0].max()
df[labels==0].min()

ZERO = df[labels==0]
ONE = df[labels==1]
TWO = df[labels==2]
THREE = df[labels==3]
FOUR = df[labels==4]

len(ZERO)
len(ONE)
```

len (TWO)
len (THREE)
len (FOUR)

9. Linear Regression

AALSD

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

data = pd.read_csv ("D:\Subject\MLDL\-
    - Datasets\Simple linear regression.01.csv")
data
data.describe()
# Our dependent variable (y) is the GPA
y = data ['GPA']
# Similarly, our independent variable (x)
x1 = data ['SAT']

plt.scatter (x1, y)
plt.xlabel ('SAT', fontsize = 20)
plt.ylabel ('GPA', fontsize = 20)
plt.show()

x = sm.add_constant (x1)
results = sm.OLS (y, x).fit()
results.summary()
plt.scatter (x1, y)
yhat = 0.0017 * x1 + 0.275
fig = plt.plot (x1, yhat, lw = 4, c = 'orange',-
    - label = 'regression line')
plt.xlabel ('SAT', fontsize = 20)
```

```
plt.ylabel('GPA', fontsize=20)
plt.show()
```

Page No.:
Date: YOUVA

10. Logistic Regression

Ans:-

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from scipy import stats
stats.chisqprob = lambda chisq, df: sta-
    -ts.chi2.sf(chisq, df)

raw_data = pd.read_csv("D:\Subject\-
    -MLDL\Datasets\Example_bank_data.csv")
raw_data
data = raw_data.copy()
data = data.drop(['Unnamed: 0'],axis=1)
data['y'] = data['y'].map({'yes':1,-
                -'no':0})
data
data.describe()
y = data['y']
x1 = data['duration']
x = sm.add_constant(x1)
reg_log = sm.Logit(y,x)
results_log = reg_log.fit()
results_log.summary()

plt.scatter(x1,y,color='C0')
```

```
plt.xlabel ('Duration', fontsize=20)
plt.ylabel ('Subscription', fontsize=20)
plt.show()

np.set_printoptions (formatter={'float': lamb-
    -da x: "{0:0.2f}".format (x)})
results_log.predict()
np.array (data['y'])
results_log.pred_table()

cm_df = pd.DataFrame (results_log.pred_table())
cm_df.columns = ['Predicted 0', 'Predicted 1']
cm_df = cm_df.rename (index={0: 'Actual 0',
                    -1: 'Actual 1'})
cm_df

cm = np.array (cm_df)
accuracy_train = (cm[0,0]+cm[1,1])/ cm.-
                    -sum()

accuracy_train
```

Date: YOUVA

11. Multiple Linear Regression

Ans:-

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

raw_data = pd.read_csv("D:\Subject\ML-DL\Datasets\Dummies1.03.+.csv")
raw_data['Attendence'] = raw_data['Attendance'].map({'Yes':1, 'NO':0})
raw_data.describe()

y = raw_data['GPA']
x1 = raw_data[['SAT','Attendance']]
x = sm.add_constant(x1)
results = sm.OLS(y,x).fit()
results.summary()

plt.scatter(raw_data['SAT'],y)
yhat_no = 0.6439 + 0.0014*raw_data['SAT']
yhat_yes = 0.2226 + 0.0014*raw_data['SAT']

fig = plt.plot(raw_data['SAT'], yhat_no,
    lw=2, c='#006837')
fig = plt.plot(raw_data['SAT'], yhat_yes,
    lw=2, c='#a50026')
```

```
plt.xlabel ('SAT', fontsize=20)
plt.ylabel ('GPA', fontsize=20)
plt.show()

plt.scatter(raw_data['SAT'], y, c=raw_data['-
    -Attendance'], cmap='RdY1Gn_r')
yhat_no = 0.6439 + 0.0014 * raw_data['SAT']
yhat_yes = 0.2226 + 0.0014 * raw_data['SAT']
fig = plt.plot(raw_data['SAT'], yhat_no, lw=2,-
        -c='#006837')
fig = plt.plot(raw_data['SAT'], yhat_yes, lw=2,
        -c='#a50026')
plt.xlabel ('SAT', fontsize=20)
plt.ylabel ('GPA', fontsize=20)
plt.show()

plt.scatter(raw_data['SAT'], raw_data['GPA'],
c=raw_data['Attendance'], cmap='RdY1Gn_r')
yhat_no = 0.6439 + 0.0014 * raw_data['SAT']
yhat_yes = 0.2226 + 0.0014 * raw_data['SAT']
yhat = 0.0017 * raw_data['SAT'] + 0.275

fig = plt.plot(raw_data['SAT'], yhat_no, lw=2,
  -c='#006837', label='regression line1')
fig = plt.plot(raw_data['SAT'], yhat_yes, lw=2,
 -c='#a50026', label='regression line2')
fig = plt.plot(raw_data['SAT'], yhat, lw=3,-
 -c='#4C72B0', label='regression line')
plt.xlabel ('SAT', font size=20)
plt.ylabel ('GPA', font size=20)
```

```
plt.show()

new_data = pd.DataFrame({'Const':1,'SAT':[1700,1921],'Attendance':[0,1]})
new_data = new_data[['const','SAT','Attendance']]
new_data
new_data.rename(index={0:'Tom',1:'Jerry'})
predictions = results.predict(new_data)
predictions

predictionsdf = pd.DataFrame({'Predictions':predictions})
joined = new_data.join(predictionsdf)
joined.rename(index={0:'Tom',1:'Jerry'})
```

12. Predicting Customer churn using Decision Trees

Ans:-

```
import pandas as pd
from sklearn.model_selection import train_test_
    - split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
    - classification_report
from sklearn import tree
import matplotlib.pyplot as plt

from sklearn.datasets import make_classifi-
    - cation
X, y = make_classification (n_samples=1000, -
- n_features=10, n_informative=5, n_redundant-
- =2, n_classes=2, random_state=42)

df = pd.DataFrame (X, columns=[f'Feature_-
- {i+1}' for i in range (X.shape[1])])
df['Churn'] = y
print ("Sample Data:")
print (df.head())
X_train, X_test, y_train, y_test = train_test_
- split (df.drop (columns=['Churn']), df-
- ['Churn'], test_size=0.3, random_state=42)

clf = DecisionTreeClassifier (max_depth=5, -
    - random_state=42)
clf.fit (X_train, y_train)
```

```
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
    target_names=["No Churn", "Churn"]))
plt.figure(figsize=(20,10))
tree.plot_tree(clf, feature_names=df.colu
    mns[:-1], class_names=["No Churn", "Chu-
    rn"], filled=True)
plt.show()
```

13: Random forest model

Ans:

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

path = "https://archive.ics.uci.edu/ml/machine-learning-database/iris/iris.data"
headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(path, names=headernames)

dataset.head()
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
X
y
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

from sklearn.ensemble import RandomForestClassifier
Classifier = RandomForestClassifier(n_estimators = 50)
Classifier.fit(X_train, y_train)
RandomForestClassifier(n_estimators=50)
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_re-
    -port, 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)
print("Accuracy:", result2)
```

Page No.:
Date: YOUVA

14. SVM Demo1

Ans:-

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

data_set = pd.read_csv ("D:\Subject\MLDL-
    -\Datasets\Social_Network_Ads1.csv")
dataset.head()
data_set.shape
x = data_set.iloc [:, [2,3]]
y = data-set.iloc [:, 4]
x.head()
y.head()
from sklearn.model_selection import train_
    -test_split
x_train, x_test, y_train, y_test = train_te-
    -st_split (x, y, test_size=0.25, random_st-
        -ate=0)
print ("Training Data:", x_train.shape)
print ("Testing Date:", x_test.shape)
from sklearn.preprocessing import Standard-
    -Scaler
st_x = StandardScaler()
x_train = st_x.fit_transform (x_train)
x_test = st_x.transform (x_test)
```

```
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state
                 =0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
y_pred

from sklearn.metrics import confusion_ma-
                            -trix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_s-
                            -core
print("Accuracy:", accuracy_score(-
    -y_test, y_pred)
cm

from sklearn.svm import SVC
classifier = SVC(kernel='rbf')
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
y_pred

print("Accuracy:", accuracy_score-
                -(y_test, y_pred))

from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start-
-x_set[:, 0].min() - 1, stop = x_set[:, 0]:-
-max() + 1, step = 0.01),
```

```
nm.arange (start = x_set[:,1].min() -1, stop = -
-x_set[:,1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.a-
-rray([x1.ravel(), x2.ravel()]).T).reshape(-
-x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', -
-'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set-
    -[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i),-
        -label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
from sklearn matplotlib.colors import List-
-edColormap
x_set, y_set = x_test, y_test
x1, x2, = nm.meshgrid(nm.arange(start-
-= x_set[:,0].min() - 1, stop = x_set[:,-
-0].max() + 1, step = 0.01),
nm.arange(start = x_set[:,1].min() - 1,
-stop = x_set[:,1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(n-
-m.array([x1.ravel(), x2.ravel()]).T).-
```

```
- reshape : (x1. shape),
alpha = 0.75, cmap = ListedColormap (('red',-
    - 'green' )))
mtp. xlim (x1. min(), x1. max())
mtp. ylim (x2. min(), x2. max())
for i, j in enumerate (np.unique (y_set)):
    mtp. scatter (x_set [y_set == j, 0], x_set -
        - [y_set == j, 1],
        c = ListedColormap (('red', 'green'))-
        - (i), label = j)
mtp. title ('SVM classifier (Test set)')
mtp. xlabel ('Age')
mtp. ylabel ('Estimated Salary')
mtp. legend()
mtp. show()
```

15. SVM cancer Prediction

Ans:

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

Load Data from CSV file
cell_df = pd.read_csv("D:\Subject\MLDL\
    - Datasets\cell_sample.csv")
cell_df.head()
cell_df.shape
cell_df.size
cell_df.count()
cell_df['Class'].value_counts()
# Distribution of the data set
benign_df = cell_df[cell_df['Class']==2]
    -[0:200]
malignant_df = cell_df[cell_df['Class']==4
    -][0:200]
axes = benign_df.plot(kind='scatter', x=
    -'Clump', y='Unitsize', color='blue', label
    -='Benign')
malignant_df.plot(kind='scatter', x='Clu
    -mp', y='unitsize', color='red', label=
    -'Benign')

axes = benign_df.plot(kind='scatter', x='Clump',
    -y='unitsize', color='blue', label='Benign')
malignant_df.plot(kind='scatter', x='Clump',
```

```
- y='Unifsize', color='red', label='Benign',
    - ax=axes)
cell-df.dtypes
cell_df = cell-df[pd.to_numeric(cell-df-
 -['BareNuc'], errors='coerce').notnull()]
cell-df['BareNuc'] = cell_df['BareNuc'],-
   -errors='coerce').notnull()]
cell-df['BareNuc'] = cell-df['BareNuc'],-
   -errors='coerce').notnull()]
cell-df['BareNuc'] = cell_df['BareNuc'].-
      -.astype('int')
cell-df.dtypes
cell_df.columns
feature_df = cell-df[['Clump', 'UnifSize',-
  -'MargAdh', 'SingEpiSize', 'BareNuc',-
  -'BlandChrom', 'NormNucl', 'Mit']]
x = np.asarray(feature_df)
y = np.asarray(cell_df['Class'])
x[0:5]
y[0:5]
from sklearn.model_selection import-
   -train_test_split
x_train, x_test, y_train, y_test = train_te-
   -st_split(x, y, test_size=0.2, random=
      =State=4)
x_train.shape
y_train.shape
x_test.shape
y_test.shape
from sklearn.svm import SVC
```

```
classifier = SVC (kernel = 'linear', random_state=0)
classifier.fit (x_train, y_train)

y_pred = classifier.predict (x_test)
y_pred

from sklearn.metrics import confusion_matrix
cm= confusion_matrix (y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy: ", accuracy_score (y_test,-
                -y_pred))
cm

from sklearn.svm import SVC
classifier = SVC (kernel = 'rbf')
classifier.fit (x_train, y_train)
y_pred = classifier.predict (x_test)
y_pred

print ('Accuracy: ", accuracy_score (y_test,-
                -y_pred))

from sklearn.metrics import classification_
                -report
print (classification_report (y_test, y_pred))
```

Ex: Sentiment Analysis using ANN

Aim:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import seq-
    -uential
from tensorflow.keras.preprocessing.text-
    -import Tokenizer
from tensorflow.keras.layers import Dense,
    -Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.sequ-
    -ence import pad_sequences
from sklearn.model_selection import -
    - train_test_split

# Sample Data: Sentences with labels (1=posit-
    -ive, 0=negative)
sentences = [
    "I love this product, it's amazing!",
    "This is the worst thing I've ever bought.",
    "Absolutely fantastic! Highly recommend.",
    "I hate it, would not recommend to anyone.",
    "It's okay, not the best but works fine.",
    "I'm really satisfied with the quality.",
    "Terrible experience, I'm very disappointed.",
    "Great value for the money.",
    "Not worth the price.",
    "I'm so happy with my purchase!"
]
```

```
labels = np.array([1,0,1,0,1,1,0,1,0,1])

# Tokenize and Pad the sequence Parameters
    vocab_size = 50
    embedding_dim = 16
    max_length = 10
    trunc_type = 'post'
    padding_type = 'post'
    oov_tok = ""
  # Tokenizer
  tokenizer = Tokenizer(num_words=vocab_size,
              -oov_token=oov_tok)
  tokenizer.fit_on_texts(sentences)
  word_index = tokenizer.word_index
  sequences = tokenizer.texts_to_sequences(sen-
              -tences)
  padded_sequences = pad_sequences(sequences,-
      -maxlen=max_length, padding=padding_ty-
      -pe, truncating=trunc_type)
  # Train-Test split
  X_train, X_test, y_train, y_test = train_test_split-
    -(padded_sequences, labels, test_size=0.2, rand-
    -om_state=42)
  # Build the model
  model = Sequential([
    Embedding(vocab_size, embedding_dim, input_
     -length=max_length),
    GlobalAveragePooling1D(),
    Dense(24, activation='relu'),
    Dense(1, activation='sigmoid')
```

```
])
model.compile(loss='binary_crossentropy',
    optimizer='adam', metrics=['accuracy'])
model.summary()

    epochs = 10
    history = model.fit(X_train, y_train, epochs=
        = epochs, validation_data=(X_test, y_test),
        verbose=2)
# Evaluate the model
    loss, accuracy = model.evaluate(X_test, y_test,
            verbose=2)
    print(f"Test Accuracy: {accuracy*100:.2f}%")
    # Make predictions
    new_sentences = [
            "I really love this!",
            "I can't stand this at all."
    ]
new_sequences = tokenizer.texts_to_sequences
            (new_sentences)
new_padded_sequences = pad_sequences(new_se-
    quences, maxlen=max_length, padding=padd-
    ing_type, truncating=trune_type)
# Predict Sentiments
predictions = model.predict(new_padded_seq-
            uences)
for i, sentences in enumerate(new_sente-
            nces):
    sentiment = "Positive" if predictions-
```

```
-[i] > 0.5 else "Negative"
print (f"sentence : {sentence} -> sentiment :-
                    - {sentiment} ")
```

16/10/2024