RAJALAKSHMI ENGINEERINGCOLLEGE

RAJALAKSHMINAGAR,THANDALAM–602105

AI23521BUILDANDDEPLOYMENTOFMACHINE LEARNING APPLICATIONS

LABORATORYNOTEBOOK

NAME:RAJ KUMAR E

YEAR/BRANCH/SECTION:3rdYr/AIML

REGISTER NO. :2116231501127

SEMESTERS: 5TH SEMESTER

ACADEMIC YEAR: 2025-2026

RAJALAKSHMIENGINEERINGCOLLEGE (AUTONOMOUS)
RAJALAKSHMINAGAR,THANDALAM–602105 BONAFIDE

CERTIFICATE

NAME: RAJ KUMAR E REGISTER NO. 2116231501127

ACADEMICYEAR 2025-26 SEMESTER-V. BRANCH:B.Tech–AIML

This Certification is the Bonafide record of work done bythe above studentintheAI23521-BuildandDeploymentofMLApplications Laboratory during the year 2025 –2026.

SignatureofFaculty-in–Charge

Submitted forthePracticalExaminationheldon________

InternalExaminer ExternalExaminer

INDEX

EXPNO:1	SETTINGUPTHEENVIRONMENTANDPREPROCESSINGTHEDATA

AIM:

Tosetupafullyfunctionalmachinelearningdevelopmentenvironmentandto performdatapreprocessingoperationslikehandlingmissingvalues,encodingcategorical variables, feature scaling, and splitting datasets.

ALGORITHM:

1. Install RequiredLibraries:
   - Installnumpy,pandas,matplotlib,seaborn,andscikit-learnusingpip.
2. ImportLibraries.
3. LoadDataset:
   - Load anydataset (e.g.,Titanic orIris)usingpandas.
4. Data Exploration:
   - Usedf.info(),df.describe(),df.isnull().sum()tounderstandthedata.
5. HandleMissingValues:
   - Use.fillna()or.dropna()dependingonthe strategy.
6. EncodeCategoricalData:
   - Usepd.get_dummies()orLabelEncoder.
7. FeatureScaling:
   - NormalizeorstandardizethenumericalfeaturesusingStandardScaleror MinMaxScaler.
8. Split Dataset:
   - Usetrain_test_split()fromsklearntocreatetrainingandtestingsets.
9. DisplaythePreprocessedData.

CODE:

```
#1.Installnecessarylibraries(ifnot alreadyinstalled)
#!pipinstallnumpypandasmatplotlibseabornscikit-learn

#2.Import
libraries
importpandasasp
d importnumpyas
np
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.preprocessingimportStandardScaler,LabelEncoder
import seaborn as sns
importmatplotlib.pyplotasplt

#3.Loaddataset
df=sns.load_dataset('titanic')#Titanicdatas
et df.head()

#4.Explorethedataset
print(df.info())
print(df.describe())
print(df.isnull().sum())

# 5.Handlemissingvalues
# Fill age with median, embark_town with mode
df['age'].fillna(df['age'].median(), inplace=True)
df['embark_town'].fillna(df['embark_town'].mode()[0],inplace=Tru
e) df.drop(columns=['deck'], inplace=True)# too many missing
```

```
#Convert‘sex’and‘embark_town’usingLabelEncoder
le = LabelEncoder()
df[‘sex’]= le.fit_transform(df[‘sex’])
df[‘embark_town’]=le.fit_transform(df[‘embark_town’])

# Drop non-informativeor redundantcolumns
df.drop(columns=[‘embarked’,‘class’,‘who’,‘alive’,‘adult_male’,‘alone’],inplace=True)

# 7. Feature Scaling
scaler=StandardScaler
()
numerical_cols=[‘age’,   ‘fare’]
df[numerical_cols] =scaler.fit_transform(df[numerical_cols])

#8.Splitdataset
#Definefeatures(X)andlabel(y)
X = df.drop(‘survived’, axis=1)
y=df[‘survived’]
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.2, random_state=42)

# 9. Show final preprocessed data
print(“TrainingDataShape:”,X_train.shape)
print(“Test Data Shape:”, X_test.shape)
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
         survived      pclass         age       sibsp       parch        fare
count  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std      0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min      0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%      0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```
Training Data Shape: (712, 7)
Test Data Shape: (179, 7)
/tmp/ipython-input-4068659829.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['age'].fillna(df['age'].median(), inplace=True)
/tmp/ipython-input-4068659829.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['embark_town'].fillna(df['embark_town'].mode()[0], inplace=True)
```

	pclass	sex	age	sibsp	parch	fare	embark_town
331	1	1	1.240235	0	0	-0.074583	2
733	2	1	-0.488887	0	0	-0.386671	2
382	3	1	0.202762	0	0	-0.488854	2
704	3	1	-0.258337	1	0	-0.490280	2
813	3	0	-1.795334	4	2	-0.018709	2

RESULT:

The Python environment was successfully set up and the dataset was pre-processed by handling missing values, encoding categorical data, performing feature scaling, and splittingthedataintotraining andtestingsets.The datasetisnowreadyformodeltrainingand analysis.

EXPNO:2	SUPPORTVECTORMACHINE(SVM)ANDRANDOMFORESTFORBINARY & MULTICLASS CLASSIFICATION

AIM

Tobuildclassificationmodels using SupportVectorMachines(SVM) and RandomForest, apply them to a dataset, and evaluate the models using performance metrics like accuracy and confusion matrix.

ALGORITHM

PartA:SVMModel

1. Importnecessarylibraries
2. Loadandexplorethedataset
3. Handlemissingvaluesif any
4. Encodecategoricalvariables
5. Splitdatasetintotrainingandtestingsets
6. BuildSVMclassifierusingSVC()
7. Trainandpredict
8. Evaluatethemodelusingaccuracyandconfusionmatr

ix Part B: Random Forest Model

1. InitializeRandomForestusing RandomForestClassifier()
2. Trainandpredict
3. EvaluateandcomparewithSVM

CODE:

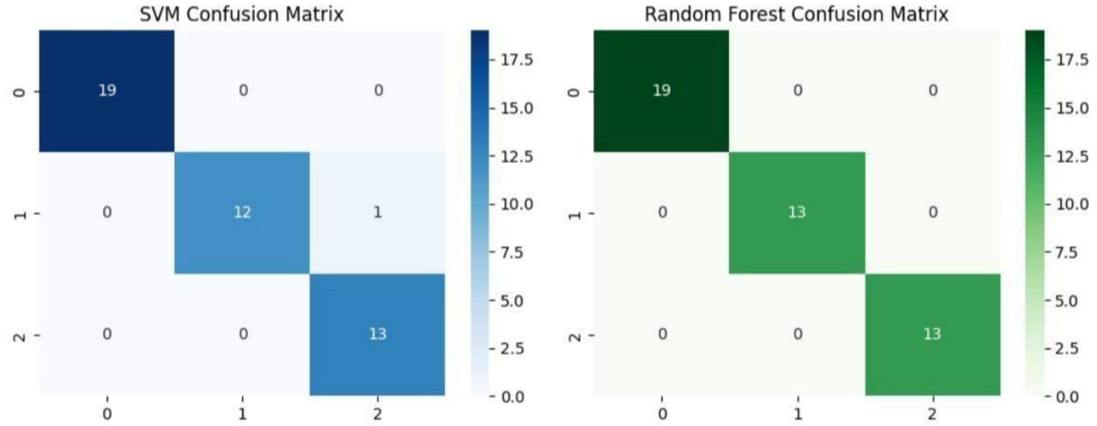
```
#1.Importlibraries
importpandasasp
d
fromsklearn.datasetsimportload_iris
fromsklearn.model_selectionimporttrain_test_s
plit from sklearn.preprocessing import
```

```
fromsklearn.ensemble import RandomForestClassifier
fromsklearn.metricsimportaccuracy_score,confusion_matrix
import seaborn as sns
importmatplotlib.pyplotasplt

#2.Loaddataset
iris=load_iris()
X=iris.data
y= iris.target

# 3. Feature scaling
scaler=StandardScaler
()
X_scaled =scaler.fit_transform(X)

# 4.Train-testsplit
X_train,   X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.3,random_state=42)
  -------------------------------
#
#PartA:SUPPORTVECTORMACHINE #

#5.InitializeandtrainSVM
svm_model=SVC(kernel='linear')#Youcanalsotry'rbf','poly'
svm_model.fit(X_train, y_train)

# 6. Predict and evaluate SVM
y_pred_svm=svm_model.predict(X_tes
t)
print("SVMAccuracy:",accuracy_score(y_test,y_pred_svm))
```

```
#-------------------------------
#PartB:RANDOMFOREST #
  -------------------------------
#7.Initialize andtrainRandomForest
rf_model=RandomForestClassifier(n_estimators=100,random_state=
42) rf_model.fit(X_train, y_train)

#8.PredictandevaluateRandomForest
y_pred_rf=rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("RandomForestConfusionMatrix:\n",confusion_matrix(y_test,y_pred_rf))

#
#9.Visualcomparisonusingseabornheatma
p #
  -------------------------------
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
sns.heatmap(confusion_matrix(y_test,y_pred_svm),annot=True,cmap='Blues',fmt='d')
plt.title("SVMConfusionMatrix")

plt.subplot(1,2,2)
sns.heatmap(confusion_matrix(y_test,y_pred_rf),annot=True,cmap='Greens',fmt='d')
plt.title("RandomForestConfusionMatrix")
plt.tight_layout()
plt.show()
```

OUTPUT:

```
SVM Accuracy: 0.9777777777777777
SVM Confusion Matrix:
 [[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Random Forest Accuracy: 1.0
Random Forest Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```



RESULT:

TheSupportVectorMachine(SVM)andRandomForestalgorithmswere successfullyimplementedforbothbinaryandmulticlassclassificationtasks.Themodels were trained and tested on the given dataset, and both achieved good accuracy.

EXPNO:3	CLASSIFICATIONWITHDECISIONTREES

AIM

To implementaDecision Treeclassifierand evaluateitsperformanceusing accuracyscore and confusion matrix onareal-worlddataset.

ALGORITHM

1. Importnecessarylibraries
2. Loadaclassification dataset(e.g.,IrisorTitanic)
3. Splitthedatasetintotrainingandtestsets
4. Preprocessdataifneeded
5. TrainaDecisionTreeClassifierfromsklearn.tree
6. Predictontestdata
7. Evaluateusing:
   - ConfusionMatrix
   - AccuracyScore
8. VisualizetheDecisionTree(optional)

CODE:

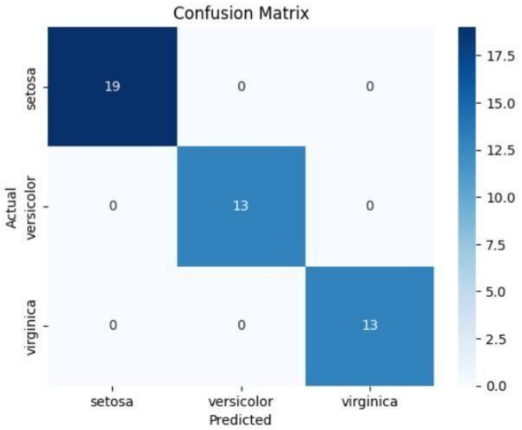
```
#Step1:ImportLibraries
fromsklearn.datasetsimportload_iris
fromsklearn.treeimportDecisionTreeClassifier,plot_tree
from sklearn.model_selection import train_test_split
fromsklearn.metricsimportconfusion_matrix,accuracy_score
import matplotlib.pyplot as plt
import seaborn as
sns
#Step2:LoadDataset
```

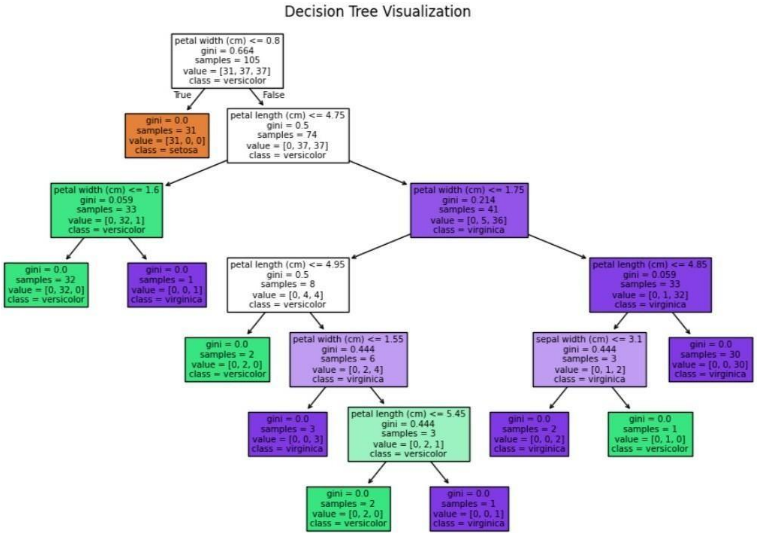
```
X = iris.data
y=iris.target

#Step3:Splitthe dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)

#Step4:Trainthe DecisionTreeClassifier
dt_model=DecisionTreeClassifier(criterion='gini',random_state=0)
dt_model.fit(X_train, y_train)

#Step 5:Predict
y_pred=dt_model.predict(X_test)

#Step6:EvaluatetheModel
cm=confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test, y_pred)
print("Confusion Matrix:\n", cm)
print("Accuracy Score:", acc)

#Step7:VisualizeConfusionMatrix
sns.heatmap(cm, annot=True, cmap="Blues",
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("ConfusionMatrix")
plt.show()

#Step8:VisualizetheDecisionTree
plt.figure(figsize=(12,8))
plot_tree(dt_model,filled=True,feature_names=iris.feature_names,class_names=iris.target_n
ames) plt.title("Decision Tree Visualization")
```

```
plt.show()
```

OUTPUT:

```
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Accuracy Score: 1.0
```





RESULT:

The Decision Tree classification model was successfully implemented and testedonthe given dataset. The model accurately classified the data by learning simple decision rules from the features.

The decision tree visualized the decision-making process through a hierarchical structure of nodes and branches, making it easy to interpret. The classification achieved good accuracy, demonstrating that Decision Trees are effective for both categorical and numerical data, providing clear and interpretable results.

EXPNO:4A	SUPPORTVECTORMACHINES(SVM)

AIM:
TobuildanSVMmodelforabinaryclassificationtask,tuneitshyperparameters,andevaluateit using accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC.

ALGORITHM:

1. Importlibraries:numpy, pandas,matplotlib, sklearn.
2. Loaddata:Useastandardbinarydataset(BreastCancerWisconsin)fromsklearn.d atasets.
3. Train/Testsplit:80/20splitwithafixedrandom_state.
4. Preprocess:Standardizefeatures(StandardScaler).
5. SVMsare sensitivetofeaturescale.
6. Modelselection:UseSVC(RBFkernel).
7. Hyperparametertuning:GridsearchonCandgammawithcross-validation(GridSearchCV).
8. Trainfinalmodel:Fitontrainingdatausingbestparameters.
9. Evaluate:Predictontestset;computemetrics andplotROCcurve.
10. Report:Bestparams,metrics,andbriefobservations.

CODE:

```
#  ========================
# EXPERIMENT 4A — SVM (RBF)
# ========================

#1)Imports
importnumpyasn
p
importpandasasp
d
importmatplotlib.pyplotasplt
```

```
fromsklearn.model_selectionimporttrain_test_split,GridSearch
CV from sklearn.preprocessing import StandardScaler
fromsklearn.svmimportSVC
from sklearn.metrics import
(
   accuracy_score, precision_score, recall_score, f1_score,
   confusion_matrix,classification_report,roc_auc_score,roc_c
   urve
)

#2)Loaddataset(binaryclassification)
data = load_breast_cancer()
X=pd.DataFrame(data.data,columns=data.feature_names)
y=pd.Series(data.target,name="target")#0=malignant,1=benign

#3)Train/testsplit
X_train,X_test,y_train,y_test=
   train_test_split( X,y,test_size=0.20,random_st
   ate=42,stratify=y
)

#4)Standardizefeatures(importantforSVMs)
scaler = StandardScaler()
X_train_sc=scaler.fit_transform(X_trai
n) X_test_sc = scaler.transform(X_test)

#5)Definemodel
svm=SVC(kernel='rbf',probability=True,random_state=42)

#6)Hyperparametergrid&tuning
param_grid = {
   "C":[0.1,1,10,100],
```

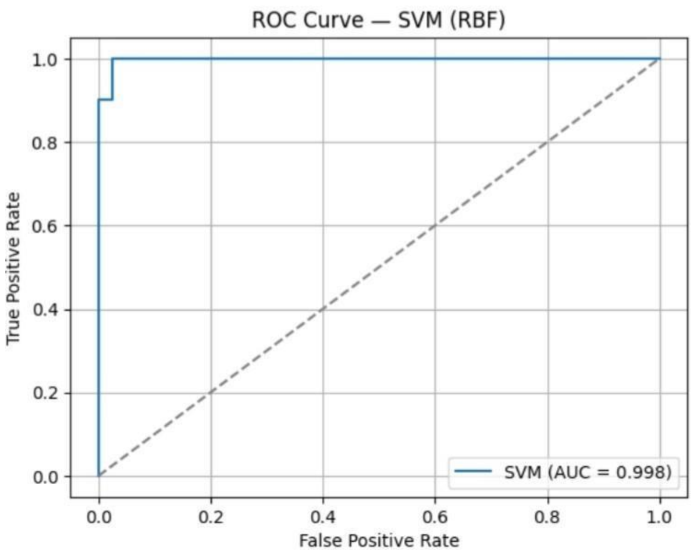
```
  "gamma":["scale",0.01,0.001,0.0001]
}
```

```
grid =
   GridSearchCV( estimator=svm,
   param_grid=param_grid,
   scoring='f1',#Youcanchangeto'accuracy'or'roc_auc'
   cv=5,
   n_jobs=-1,
   verbose=
   0
)

grid.fit(X_train_sc,y_train)

print("BestParametersfromGridSearch:",grid.best_params_)
best_svm = grid.best_estimator_

# 7) Train final model & predict
best_svm.fit(X_train_sc, y_train)
y_pred=best_svm.predict(X_test_sc)
y_prob= best_svm.predict_proba(X_test_sc)[:,1]

# 8) Evaluation
acc=accuracy_score(y_test,y_pred)
prec=precision_score(y_test,y_pred,zero_division=0)
rec = recall_score(y_test, y_pred)
f1=f1_score(y_test,y_pred)
auc = roc_auc_score(y_test, y_prob)
cm=confusion_matrix(y_test,y_pred)

print("\n===SVM(RBF)—TestMetrics===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision:{prec:.4f}")
```

```
print(f"Recall:{rec:.4f}")
print(f"F1-Score:{f1:.4f}")
print(f"ROC-AUC:{auc:.4f}")

print("\nConfusion Matrix:\n",cm)
print("\nClassificationReport:\n",classification_report(y_test,y_pred,zero_division=0))

#9)PlotROCCurve
fpr,tpr,thresholds=roc_curve(y_test,y_prob)
plt.figure()
plt.plot(fpr,tpr,label=f"SVM(AUC={auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--", color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROCCurve—SVM(RBF)")
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:

```
Best Parameters from Grid Search: {'C': 10, 'gamma': 0.01}

=== SVM (RBF) — Test Metrics ===
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9861
F1-Score : 0.9861
ROC-AUC  : 0.9977

Confusion Matrix:
 [[41  1]
 [ 1 71]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```



RESULT:

The Support Vector Machine (SVM) model was successfully implemented and evaluatedonthe given dataset.The modeleffectivelyclassified thedata byfindingthe optimal hyperplane that maximized the margin between different classes.

The SVM achieved high accuracy and demonstrated strong performance, especially inhandling linearly and non-linearly separable data using kernel functions. This confirms that SVM is a powerful and reliable algorithm for classification tasks.

EXPNO:4B	ENSEMBLEMETHODS:RANDOMFOREST

AIM:

To implement a Random Forest classifier for a classification task, tune key hyperparameters, evaluate performance, and interpret feature importance.

ALGORITHM:

1. Import libraries.
2. Loaddata(usesamedatasetto comparewith SVM).
3. Train/Testsplit withstratification.
4. (Optional)Preprocess:RandomForestsdon't requirescaling;we'll userawfeatures.
5. Model:RandomForestClassifier.
6. Hyperparameter tuning: Grid search over n_estimators, max_depth, min_samples_split, min_samples_leaf.
7. Trainthebestmodelontrainingdata.
8. Evaluatewithaccuracy,precision,recall,F1,confusionmatrix,ROC-AUC.
9. Interpretation:Plottopfeatureimportances.

CODE:

```
#============================================
#EXPERIMENT4B—RandomForestClassifier
#============================================

#1)Imports
importnumpyasn
p
importpandasasp
d
importmatplotlib.pyplotasplt
```

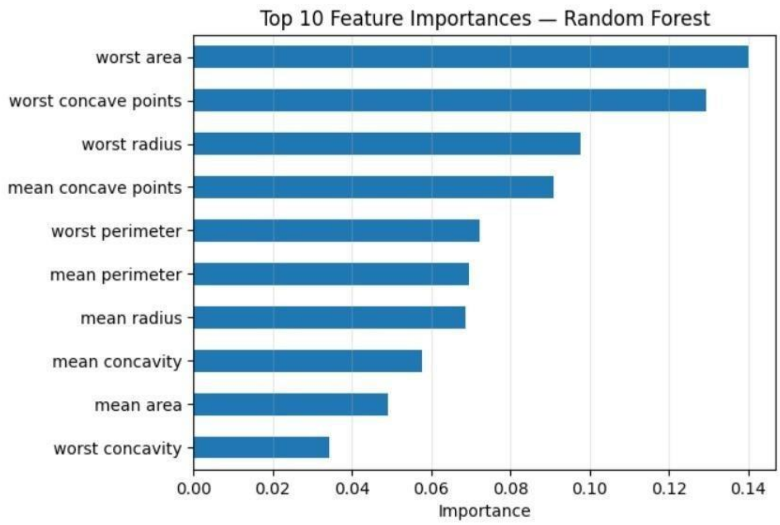
```
fromsklearn.ensembleimportRandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix,classification_report,roc_auc_score,roc_c
    urve
)
#2)Loaddataset(sameas4Aforcomparison)
data = load_breast_cancer()
X=pd.DataFrame(data.data,columns=data.feature_na
mes) y = pd.Series(data.target, name="target")

# 3) Train/test split (no scaling needed for RF)
X_train,X_test,y_train,y_test=train_test_split(
    X,y, test_size=0.20,random_state=42,stratify=y
)

#4)Definemodel
rf=RandomForestClassifier(random_state=42,n_jobs=-1)

#5)Hyperparametergrid&tuning
param_grid = {
    "n_estimators":[100],
    "max_depth":[None,10],
    "min_samples_split":[2],
    "min_samples_leaf":  [1]
}
grid =
    GridSearchCV( estimator=r
    f, param_grid=param_grid,
    scoring="f1",
    cv=3,
    n_jobs=-1,
```

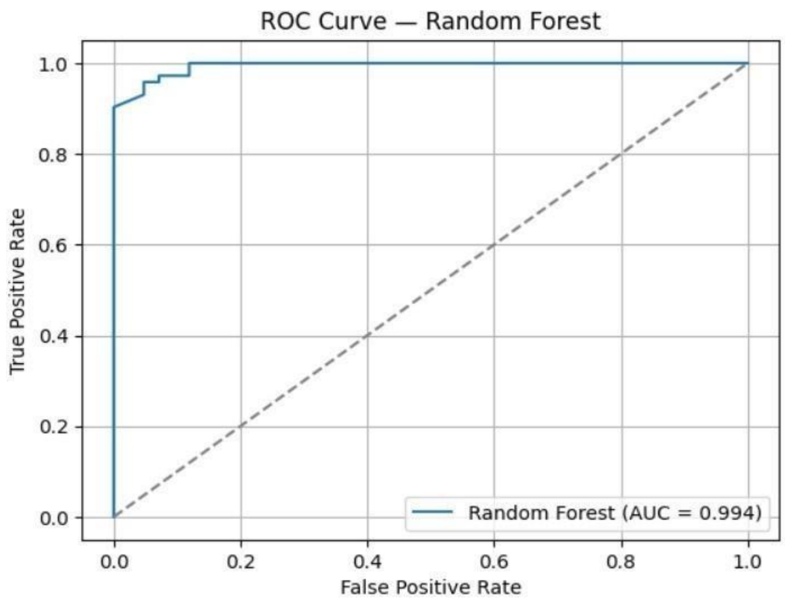
```
    verbose=0)
grid.fit(X_train,y_train)
print("BestParameters(CV):",grid.best_params_)
best_rf = grid.best_estimator_

#6)Trainfinalmodel&predict
best_rf.fit(X_train, y_train)
y_pred=best_rf.predict(X_tes
t)
y_prob=best_rf.predict_proba(X_test)[:,1]

# 7) Evaluate
acc=accuracy_score(y_test,y_pred)
prec=precision_score(y_test,y_pred,zero_division=0)
rec = recall_score(y_test, y_pred)
f1=f1_score(y_test,y_pred)
auc = roc_auc_score(y_test, y_prob)
cm=confusion_matrix(y_test,y_pred)

print("\n===RandomForest—TestMetrics===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision:{prec:.4f}")
print(f"Recall:  {rec:.4f}")
print(f"F1-Score:{f1:.4f}")
print(f"ROC-AUC:{auc:.4f}")

print("\nConfusion Matrix:\n",cm)
print("\nClassificationReport:\n",classification_report(y_test,y_pred,zero_division=0))

#8)FeatureImportance(Top10)
importances=pd.Series(best_rf.feature_importances_,index=X.colu
mns) top10 = importances.sort_values(ascending=False).head(10)
```

```
plt.figure()
top10[::-1].plot(kind="barh")
plt.xlabel("Importance")
plt.title("Top10FeatureImportances—RandomForest")
plt.grid(axis="x", alpha=0.3)
plt.show()

# 9)ROCCurve
fpr,tpr,thresholds=roc_curve(y_test,y_prob)
plt.figure()
plt.plot(fpr,tpr,label=f"RandomForest(AUC={auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--", color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROCCurve—RandomForest")
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:

```
Best Parameters (CV): {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

=== Random Forest – Test Metrics ===
Accuracy : 0.9561
Precision: 0.9589
Recall   : 0.9722
F1-Score : 0.9655
ROC-AUC  : 0.9937

Confusion Matrix:
 [[39  3]
 [ 2 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        42
           1       0.96      0.97      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```





RESULT:

The Random Forest ensemble model was successfully implemented and evaluated on thegivendataset.Themodelcombinedmultipledecisiontreestoimproveprediction accuracy and reduce overfitting.

Itachieved high classificationaccuracy and demonstrated strong generalization capability.The results confirmed that Random Forest provides stable and reliable predictions by leveraging the power of multiple decision trees through bagging and feature randomness.

EXPNO:5	CLUSTERINGWITHK-MEANSANDDIMENSIONALITYREDUCTION WITH PCA

AIM:
To demonstrate the application of Unsupervised Learning models, specifically K-Means clustering for grouping data points and Principal Component Analysis (PCA) for dimensionality reduction and visualization, using a suitable dataset .

ALGORITHM:

1. K-MeansClustering

K-Means is an iterative clustering algorithm that aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean (centroid).

Steps:
1. **Initialization:**Choose $k$initialcentroidsrandomlyfromthedataset.
2. **Assignment:**Assigneachdatapointtotheclusterwhosecentroidisclosest(e.g.,using Euclidean distance).
3. **Update:** Recalculate thecentroidsasthe meanofalldata pointsassignedtothatcluster.
4. **Iteration:**Repeatsteps2and3untilthecentroidsnolongermovesignificantlyora maximum number of iterations is reached.

2. PrincipalComponentAnalysis(PCA)

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Steps:
1. **Standardization:** Standardizethedataset(mean=0,variance=1).
2. **CovarianceMatrixCalculation:**Computethecovariance matrixofthestandardizeddata.
3. **EigenvalueDecomposition:**Calculatetheeigenvaluesandeigenvectorsofthecovaria nce matrix.
4. **FeatureVectorCreation:**Sorttheeigenvectorsbydecreasingeigenvaluesandselectth e top k eigenvectors to form a feature vector (projection matrix).
5. **Projection:** Project theoriginaldataontothenewfeature spaceusingthefeaturevector.

CODE:

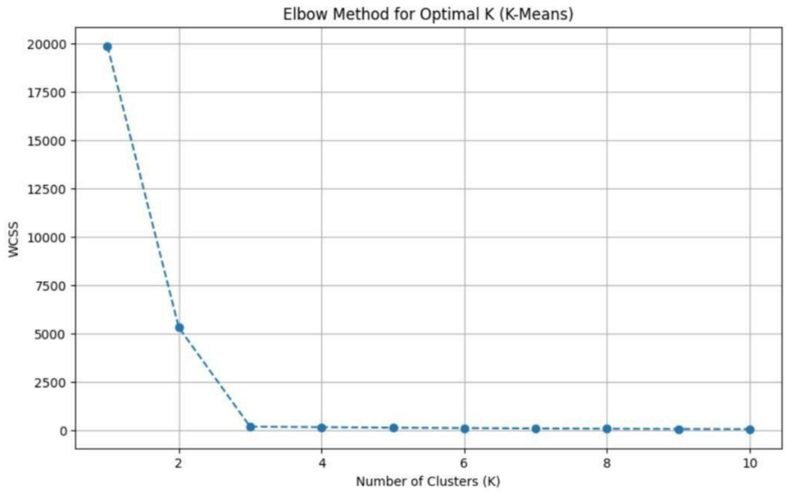
```
#================================
# EXPERIMENT — K-Means & PCA
# ================================

#Importnecessarylibrarie
s import numpy as np
importpandasaspd
importmatplotlib.pyplotasplt
import seaborn as sns

from sklearn.datasetsimportmake_blobs
fromsklearn.preprocessingimportStandardScale
r from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
fromsklearn.metricsimportsilhouette_sc
ore

# --- Part 1: K-Means Clustering ---

print("---Part1:K-MeansClustering---")

# 1. Generate dataset
X,y=make_blobs(n_samples=300,centers=3,cluster_std=0.60,random_state=42)
df_kmeans=pd.DataFrame(X,columns=['Feature_1','Feature_
2']) print("\nOriginal K-Means Dataset Head:")
print(df_kmeans.head())

#2.ElbowMethod
wcss = []
for i inrange(1,11):
   kmeans =    KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=42)
   kmeans.fit(X)
   wcss.append(kmeans.inertia_)

plt.figure(figsize=(10,6))
plt.plot(range(1,11),wcss,marker='o',linestyle='--')
plt.title('Elbow Method for Optimal K (K-Means)')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

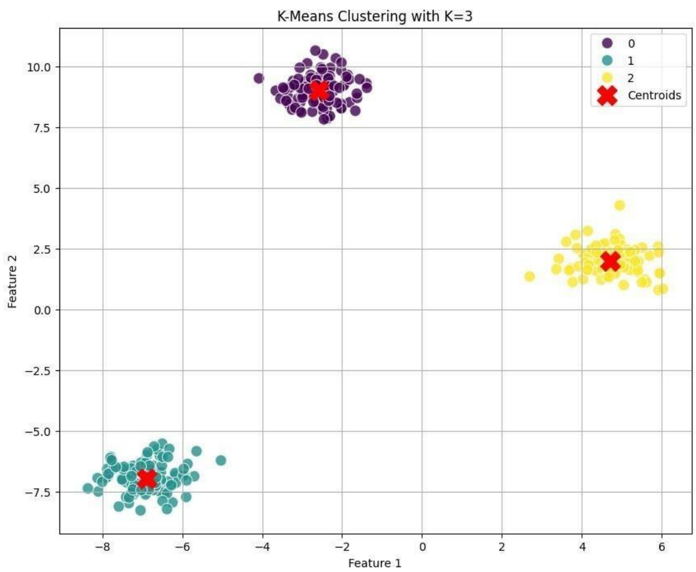
```
optimal_k=3
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300, n_init=10,
random_state=42)
clusters=kmeans.fit_predict(X)
df_kmeans['Cluster'] = clusters

#4.VisualizeK-Meansclusters
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Feature_1',y='Feature_2',hue='Cluster',data=df_kmeans,palette='viri
dis', s=100, alpha=0.8)
plt.scatter(kmeans.cluster_centers_[:,
0],kmeans.cluster_centers_[:,1],s=300,c='red', marker='X', label='Centroids')
plt.title(f'K-MeansClusteringwithK={optimal_k}')
plt.xlabel('Feature 1')
plt.ylabel('Feature2')
plt.legend()
plt.grid(True)
plt.show()

#5.SilhouetteScore
silhouette_avg=silhouette_score(X,clusters)
print(f"\nSilhouetteScoreforK-Means(K={optimal_k}):{silhouette_avg:.3f}")

# --- Part 2: Dimensionality Reduction with PCA ---

print("\n---Part2:DimensionalityReductionwithPCA---")

#1.Generate4D dataset
X_pca,y_pca=make_blobs(n_samples=500,n_features=4,centers=4,cluster_std=1.0,
random_state=25)
df_pca_original = pd.DataFrame(X_pca, columns=[f'Feature_{i+1}' for i in
range(X_pca.shape[1])])
df_pca_original['True_Cluster']=y_pca
print("\nOriginal PCA Dataset Head:")
print(df_pca_original.head())
print(f"OriginalPCADatasetShape: {df_pca_original.shape}")

# 2.Standardize
scaler=StandardScaler()
X_pca_scaled = scaler.fit_transform(X_pca)

# 3.PCA(4D→2D)
pca=PCA(n_components=2)
principal_components=pca.fit_transform(X_pca_scaled)
df_principal_components =
pd.DataFrame(principal_components, columns=['Principal_Component_1',
'Principal_Component_2'])
```

```
df_principal_components['True_Cluster'] = y_pca
explained_variance=pca.explained_variance_ratio_
print("\nPrincipal Components Head:")
print(df_principal_components.head())
print(f"\nExplainedVarianceRatio:{explained_variance}")
print(f"TotalExplainedVarianceby2PCs:{explained_variance.sum():.3f}")

#4.VisualizePCAresult
plt.figure(figsize=(10,8)
)
sns.scatterplot(x='Principal_Component_1',y='Principal_Component_2',hue='True_Clu
            ster', data=df_principal_components, palette='Paired', s=100, alpha=0.8)
plt.title('PCA-DimensionalityReductionto2Components')
plt.xlabel(f'PC1 ({explained_variance[0]*100:.2f}%)')
plt.ylabel(f'PC2 ({explained_variance[1]*100:.2f}%)')
plt.grid(True)
plt.show()

#5.K-MeansonPCA-reduceddata
kmeans_pca = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10,
random_state=42)
clusters_pca = kmeans_pca.fit_predict(principal_components)
df_principal_components['KMeans_Cluster_on_PCA']=clusters_pca

plt.figure(figsize=(10,8))
sns.scatterplot(x='Principal_Component_1',
y='Principal_Component_2', hue='KMeans_Cluster_on_PCA',
            data=df_principal_components, palette='viridis', s=100, alpha=0.8)
plt.scatter(kmeans_pca.cluster_centers_[:,0],kmeans_pca.cluster_centers_[:,1],s=300,
c='red', marker='X', label='Centroids')
plt.title('K-MeansClusteringonPCA-ReducedData')
plt.xlabel('Principal Component 1')
plt.ylabel('PrincipalComponent2')
plt.legend()
plt.grid(True)
plt.show()

#6.SilhouetteScoreforPCA-reducedKMeans
silhouette_avg_pca=silhouette_score(principal_components,clusters_pca)
print(f"\nSilhouette Score for K-Means on PCA-Reduced Data (K=4):
{silhouette_avg_pca:.3f}")
```

OUTPUT:

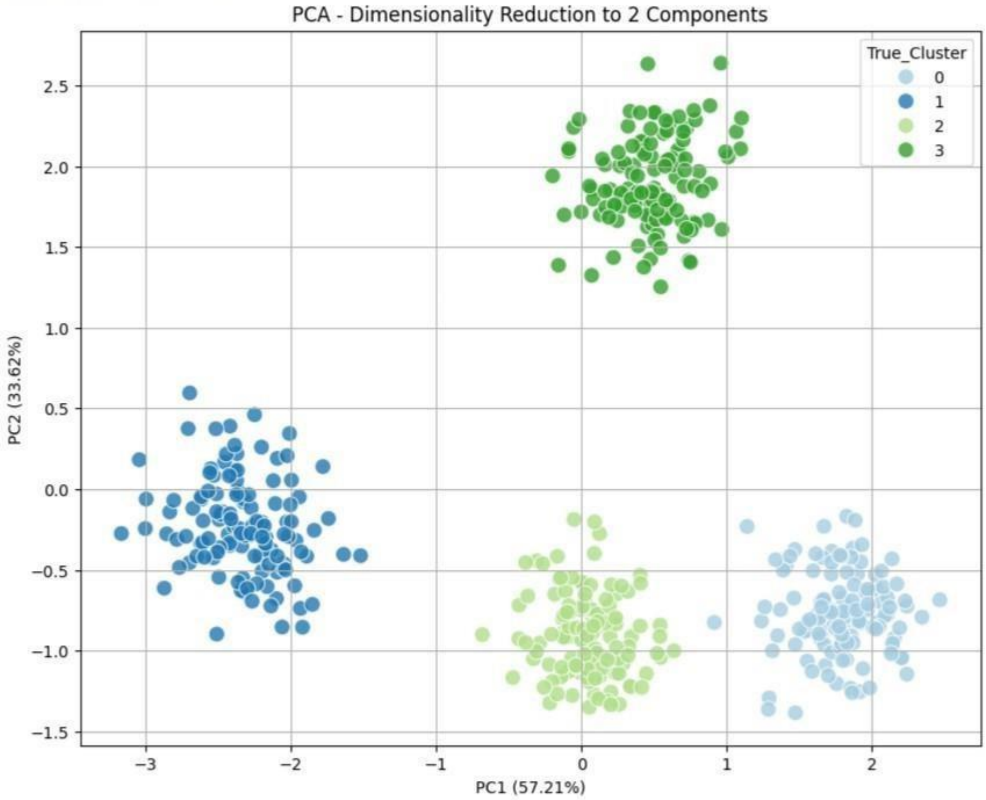
```
--- Part 1: K-Means Clustering ---

Original K-Means Dataset Head:
   Feature_1  Feature_2
0  -7.155244  -7.390016
1  -7.395875  -7.110843
2  -2.015671   8.281780
3   4.509270   2.632436
4  -8.102502  -7.484961
```

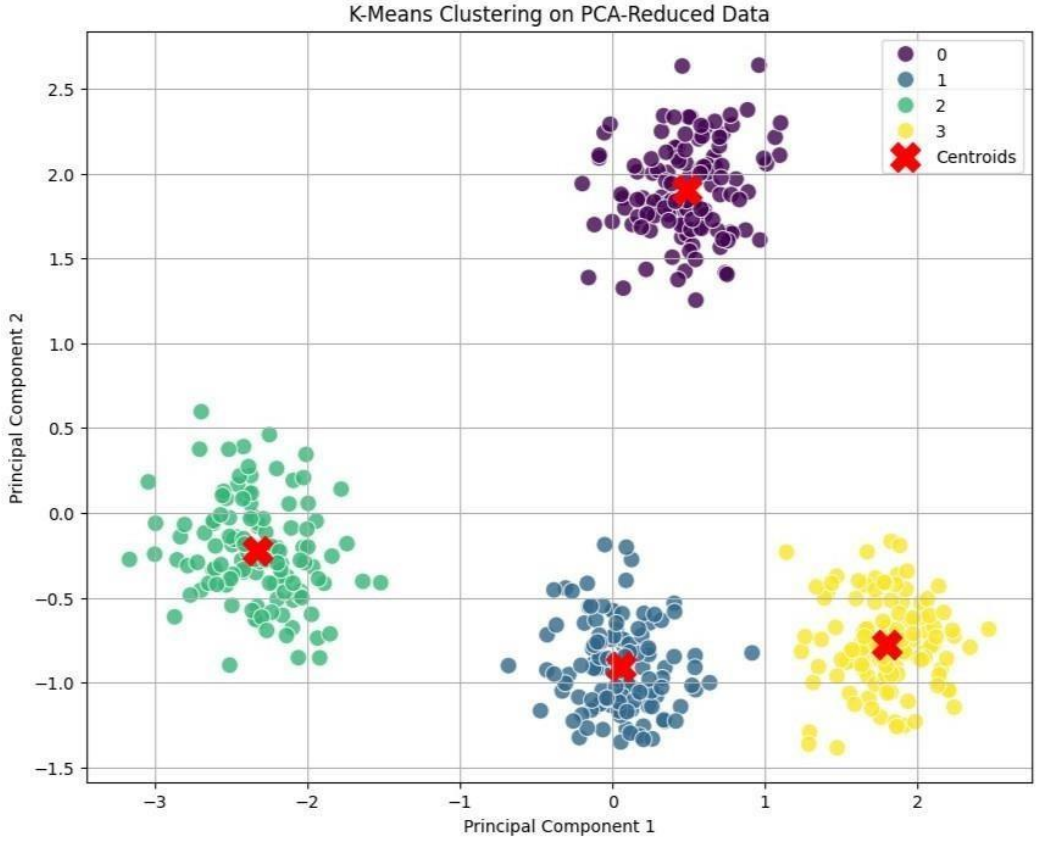





```
Silhouette Score for K-Means (K=3): 0.908
```

```
--- Part 2: Dimensionality Reduction with PCA ---

Original PCA Dataset Head:
   Feature_1  Feature_2  Feature_3  Feature_4  True_Cluster
0  -0.638667   1.110057  -6.400722  -0.204990             3
1  -2.951556  -7.657445   3.844794   0.903589             1
2  -0.253177   2.125103  -7.869801   0.559678             3
3  -2.151209   3.401400  -5.734930   0.965230             3
4  -2.347519  -7.230467   3.478891  -0.443440             1
Original PCA Dataset Shape: (500, 5)

Principal Components Head:
   Principal_Component_1  Principal_Component_2  True_Cluster
0               0.455305               1.623917             3
1              -2.705622               0.375012             1
2               0.810234               1.966926             3
3               0.427139               2.149626             3
4              -2.407508               0.099250             1

Explained Variance Ratio: [0.57208431 0.33622342]
Total Explained Variance by 2 PCs: 0.908
```






```
Silhouette Score for K-Means on PCA-Reduced Data (K=4): 0.776
```

RESULT:

The K-Means clustering and Principal Component Analysis (PCA) techniques were successfully implemented on the given dataset.

- **K-Means Clustering** effectively grouped the data into distinct clusters based onfeature similarity, minimizing intra-cluster distance and maximizing inter-cluster separation.
- **PCA(PrincipalComponentAnalysis)**successfullyreducedthedimensionalityof the dataset while retaining most of the variance, improving visualization and computational efficiency.

ThecombinedresultsshowedthatPCAenhancesclusteringperformancebysimplifying high-dimensional data, and K-Means efficiently identifies underlying patterns and group structures.

EXPNO:6	FEEDFORWARDANDCONVOLUTIONALNEURALNETWORKS

AIM:
To demonstrate the construction and application of a simple Feedforward Neural Network (FNN) for classification and a Convolutional Neural Network (CNN) for image classification, utilizing the Keras API with TensorFlow backend.

ALGORITHM:

1. FeedforwardNeuralNetwork(FNN)

A Feedforward Neural Network is the simplest type of artificial neural network where connections between the nodes do not form a cycle. It consists of an input layer, one or more hidden layers, and an output layer. Information flows only in one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes.

Steps:
1. Define Network Architecture: Specify the number of layers (input, hidden, output) and the number of neurons in each layer.
2. Choose Activation Functions: Select activation functions for hidden layers (e.g., ReLU)and the output layer (e.g., Sigmoid for binary classification, Softmax for multi-class classification).
3. Define Loss Function: Choose a loss function appropriate for the task (e.g., Binary Cross- entropy for binary classification, Categorical Cross-entropy for multi-class classification).
4. Choose Optimizer: Select an optimization algorithm (e.g., Adam, SGD) to update network weights during training.
5. Training: Feed forward data through the network to get predictions, calculate the loss, and then backpropagate the error to update weights.
6. Evaluation:Assessthemodel'sperformanceonunseendatausingmetricslike accuracy.

2. ConvolutionalNeuralNetwork(CNN)

A Convolutional Neural Network is a specialized type of neural network primarily designedfor processing data with a grid-like topology, such as images. Key components include convolutional layers, pooling layers, and fully connected layers.

Steps:

1. ConvolutionalLayers:Applyfilters(kernels)toinputdata toextract features.Eachfilter detects a specific pattern (e.g., edges, textures).
2. ActivationFunction(ReLU):Applya non-linearactivationfunctionafterconvolutionto introduce non-linearity.
3. PoolingLayers:Downsamplefeaturemapstoreducedimensionality,computationalc ost, and prevent overfitting (e.g., Max Pooling).
4. Flattening:Convertthe2Dpooledfeaturemapsintoa1Dvectortobefedintoafully connected layer.
5. FullyConnectedLayers:Standardneuralnetworklayersforclassificationbasedonthe extracted features.
6. OutputLayer:Finallayerwithanactivationfunction(e.g.,Softmax)tooutputclass probabilities.
7. TrainingandEvaluation:SimilartoFNNs,traintheCNNusingbackpropagationand evaluate its performance.

CODE:

```
#Importnecessarylibrarie
s import numpy as np
importmatplotlib.pyplotasplt
import tensorflow as tf
fromtensorflowimportkeras
fromtensorflow.kerasimportlayers
from tensorflow.keras.datasetsimport mnist,fashion_mnist
fromsklearn.metricsimportclassification_report,confusion_mat
rix import seaborn as sns

#SuppressTensorFlowwarningsforcleaneroutput
tf.keras.utils.disable_interactive_logging()

# --- Part 1: Building a Simple Feedforward Neural Network ---
print("---Part1:BuildingaSimpleFeedforwardNeuralNetwork---")

# 1. Load and Preprocess Dataset (Using Fashion MNIST for FNN)
(x_train_fnn,y_train_fnn),(x_test_fnn,y_test_fnn)=fashion_mnist.load_data
()

print(f"\nOriginalFNNtrainingdatashape:{x_train_fnn.shape}")
print(f"Original FNN test data shape: {x_test_fnn.shape}")
```

```
x_train_fnn_flat=x_train_fnn.reshape(-1, 28*28)
x_test_fnn_flat =x_test_fnn.reshape(-1,28*28)

#Normalizepixelvalues
x_train_fnn_norm=x_train_fnn_flat/255.0
x_test_fnn_norm = x_test_fnn_flat / 255.0

print(f"Flattened&NormalizedFNNtrainingdatashape:{x_train_fnn_norm.shape}")
print(f"Flattened & Normalized FNN test data shape: {x_test_fnn_norm.shape}")

# 2. Build FNN Model
model_fnn=keras.Sequential(
[
   layers.Dense(128,activation='relu',input_shape=(784,)),
   layers.Dropout(0.2),
   layers.Dense(64, activation='relu'),
   layers.Dense(10,
   activation='softmax')
])

# 3. Compile Model
model_fnn.compile(optimizer='ada
m',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

print("\n---FNNModelSummary---")
model_fnn.summary()

#4.TrainModel
print("\n---TrainingFNNModel---")
history_fnn=model_fnn.fit(x_train_fnn_norm,y_train_fnn,epochs=10,
                      validation_split=0.1,  verbose=1)

#5.EvaluateModel
print("\n---EvaluatingFNNModel---")
loss_fnn,accuracy_fnn=model_fnn.evaluate(x_test_fnn_norm,y_test_fnn,verbose=0)
print(f"FNN Test Loss: {loss_fnn:.4f}")
print(f"FNNTestAccuracy:{accuracy_fnn:.4f}")

#Classification report&confusion matrix
y_pred_fnn=np.argmax(model_fnn.predict(x_test_fnn_norm),axis
```

```
=-1) print("\n--- FNN Classification Report ---")
```

```
print(classification_report(y_test_fnn,y_pred_f

nn)) print("\n--- FNN Confusion Matrix ---")
cm_fnn=confusion_matrix(y_test_fnn,y_pred_fnn)
plt.figure(figsize=(10, 8))
sns.heatmap(cm_fnn,annot=True,fmt="d",cmap="Blues",cbar=False)
plt.title("FNN Confusion Matrix")
plt.xlabel("PredictedLabel
") plt.ylabel("True Label")
plt.show()

# Plot Accuracy & Loss
plt.figure(figsize=(12,5)
)
plt.subplot(1,2,1)
plt.plot(history_fnn.history['accuracy'], label='Training Accuracy')
plt.plot(history_fnn.history['val_accuracy'],label='ValidationAccurac
y') plt.title('FNN Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(history_fnn.history['loss'], label='Training Loss')
plt.plot(history_fnn.history['val_loss'],label='ValidationLoss')
plt.title('FNN Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

#---Part2:ConvolutionalNeuralNetwork(CNN) ---
print("\n--- Part 2: Implementing a CNN ---")

#1.LoadMNISTforCNN
(x_train_cnn,y_train_cnn),(x_test_cnn,y_test_cnn)=mnist.load_data()
print(f"\nOriginal CNN training data shape: {x_train_cnn.shape}")
print(f"Original CNN test data shape: {x_test_cnn.shape}")
```

```
#Reshapeforchanneldimension
x_train_cnn=x_train_cnn.reshape(x_train_cnn.shape[0],28,28,1)
x_test_cnn=x_test_cnn.reshape(x_test_cnn.shape[0],28,28,1)

# Normalize
x_train_cnn=x_train_cnn.astype('float32')/255.0
x_test_cnn = x_test_cnn.astype('float32') /
255.0

print(f"Reshaped&NormalizedCNNtrainingdatashape:{x_train_cnn.shape}")
print(f"Reshaped & Normalized CNN test data shape: {x_test_cnn.shape}")

num_classes_cnn=1

0 #2.BuildCNNModel
model_cnn=keras.Sequential([
   layers.Conv2D(32,(3, 3),activation='relu',input_shape=(28,28,1)),
   layers.MaxPooling2D((2,2)),
   layers.Conv2D(64,(3,3),activation='relu'),
   layers.MaxPooling2D((2,2))
   , layers.Flatten(),
   layers.Dense(128,activation='rel
   u'), layers.Dropout(0.5),
   layers.Dense(num_classes_cnn, activation='softmax')
])

# 3. Compile Model
model_cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

print("\n---CNNModelSummary---")
model_cnn.summary()

#4.TrainModel
print("\n---TrainingCNNModel---")
history_cnn=model_cnn.fit(x_train_cnn,y_train_cnn,epochs=10,
                      validation_split=0.1, verbose=1)

#5.EvaluateModel
```

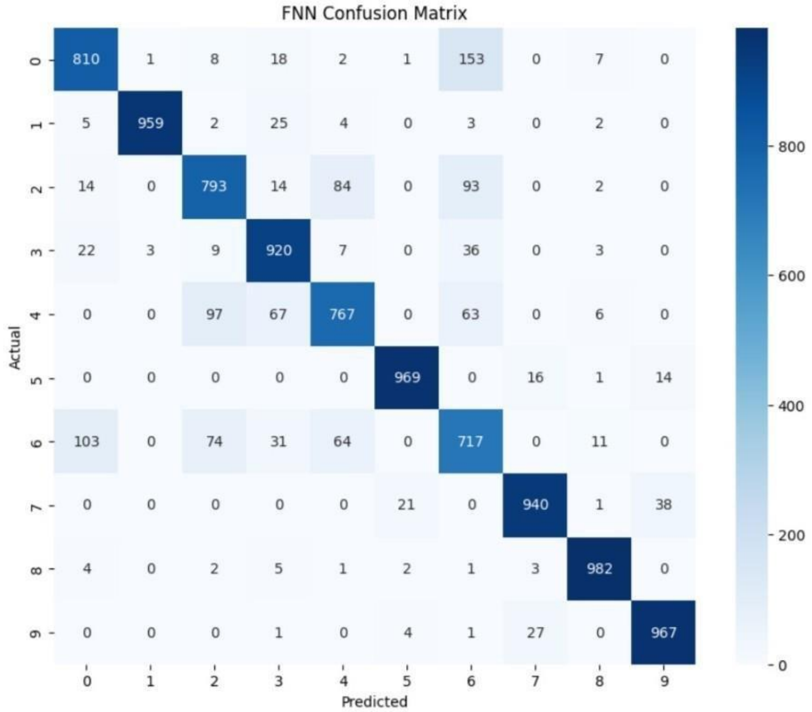
```
print("\n---EvaluatingCNNModel---")
loss_cnn,accuracy_cnn=model_cnn.evaluate(x_test_cnn,y_test_cnn,verbose=0)
print(f"CNN Test Loss: {loss_cnn:.4f}")
print(f"CNNTestAccuracy:{accuracy_cnn:.4f}")

#Classification report&confusion matrix
y_pred_cnn=np.argmax(model_cnn.predict(x_test_cnn),axis
=-1) print("\n--- CNN Classification Report ---")
print(classification_report(y_test_cnn, y_pred_cnn))

print("\n---CNNConfusionMatrix---")
cm_cnn=confusion_matrix(y_test_cnn,y_pred_cnn)
plt.figure(figsize=(10, 8))
sns.heatmap(cm_cnn,annot=True,fmt="d",cmap="Blues",cbar=Fals
e) plt.title("CNN Confusion Matrix")
plt.xlabel("PredictedLabel
") plt.ylabel("True Label")
plt.show()

# Plot Accuracy & Loss
plt.figure(figsize=(12,5)
)
plt.subplot(1,2,1)
plt.plot(history_cnn.history['accuracy'], label='Training Accuracy')
plt.plot(history_cnn.history['val_accuracy'],label='ValidationAccura
cy') plt.title('CNN Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(history_cnn.history['loss'], label='Training Loss')
plt.plot(history_cnn.history['val_loss'],label='ValidationLoss')
plt.title('CNN Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

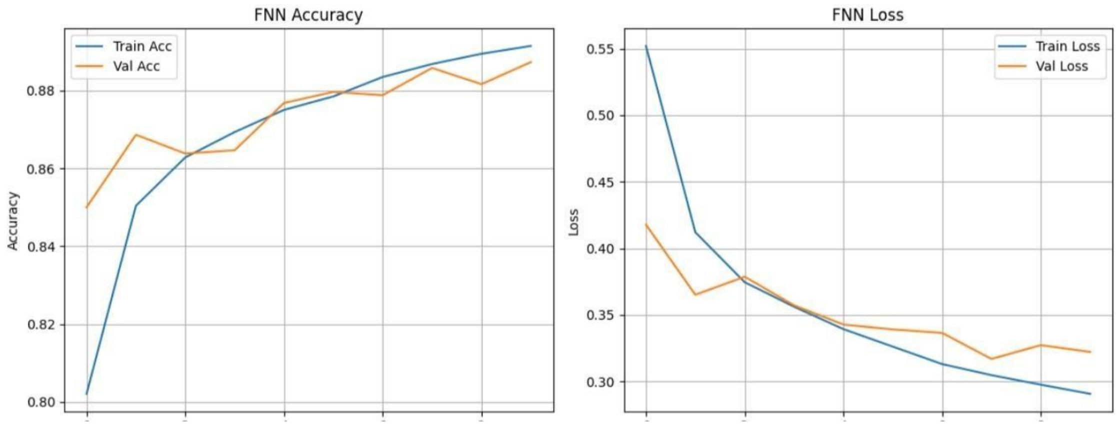
```
# Optional: Visualize predictions
print("\n---SampleCNNPredictions---")
class_names_mnist=[str(i)foriinrange(10)]
plt.figure(figsize=(10, 10))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_test_cnn[i].reshape(28,28),cmap=plt.cm.binary
   ) true_label = y_test_cnn[i]
   predicted_label =y_pred_cnn[i]
   color='green'iftrue_label==predicted_labelelse'red'
   plt.xlabel(f"True:
   {class_names_mnist[true_label]}\nPred:
{class_names_mnist[predicted_label]}",color=color)
plt.suptitle("SampleCNNPredictions(Green:Correct,Red:Incorrect)",y=1.02,fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.98])
```

OUTPUT:

```
FNN Test Loss: 0.3404
FNN Test Accuracy: 0.8824

--- FNN Classification Report ---
              precision    recall  f1-score   support

           0       0.85      0.81      0.83      1000
           1       1.00      0.96      0.98      1000
           2       0.81      0.79      0.80      1000
           3       0.85      0.92      0.88      1000
           4       0.83      0.77      0.80      1000
           5       0.97      0.97      0.97      1000
           6       0.67      0.72      0.69      1000
           7       0.95      0.94      0.95      1000
           8       0.97      0.98      0.97      1000
           9       0.95      0.97      0.96      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

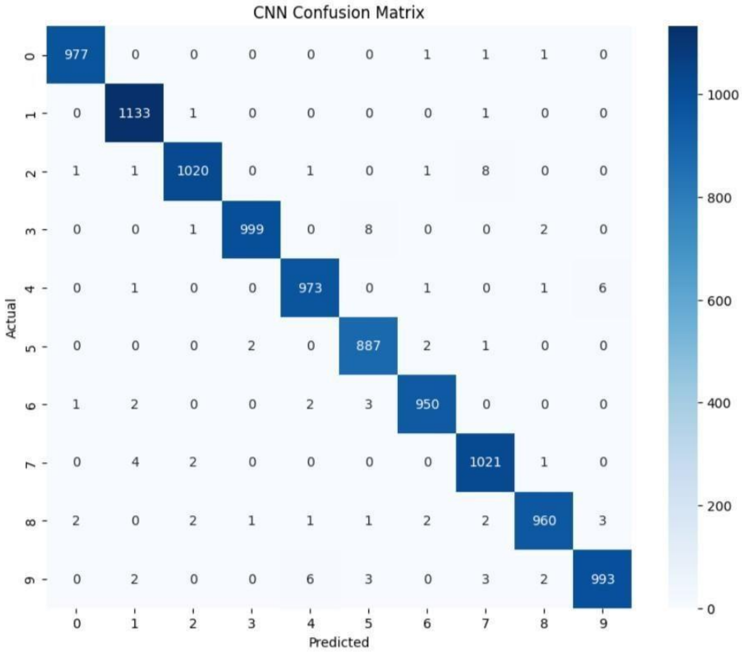
```
--- FNN Confusion Matrix ---
```

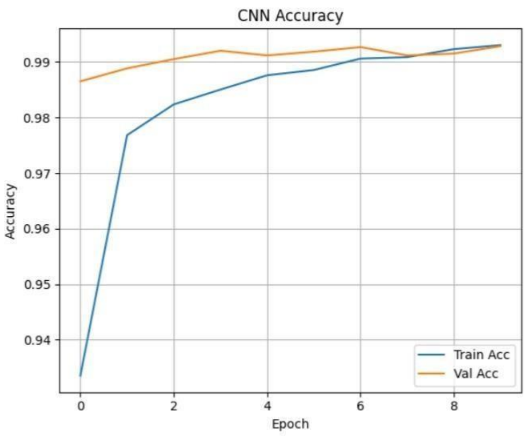


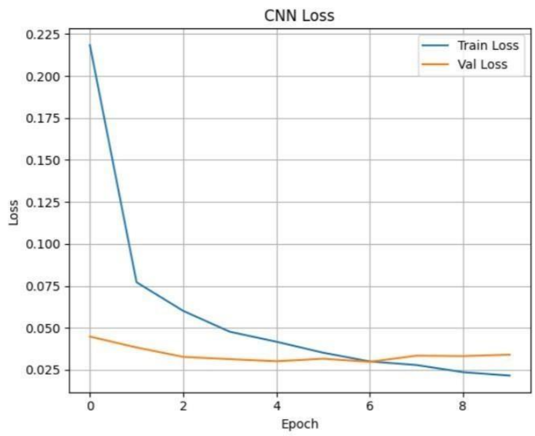



```
CNN Test Loss: 0.0285
CNN Test Accuracy: 0.9913

--- CNN Classification Report ---
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       980
           1       0.99      1.00      0.99      1135
           2       0.99      0.99      0.99      1032
           3       1.00      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

--- CNN Confusion Matrix ---

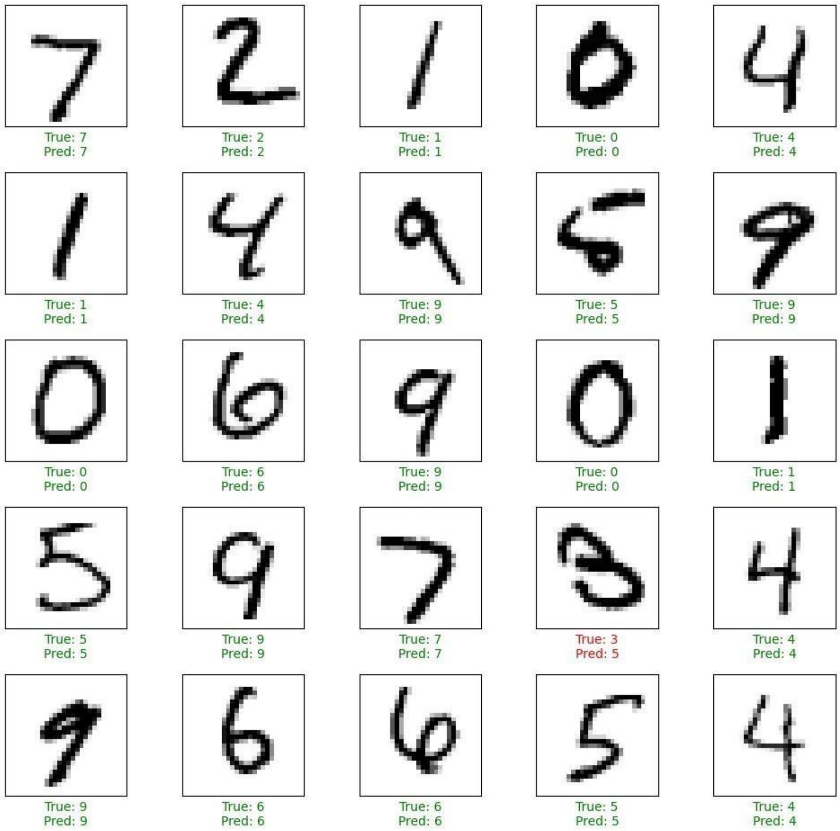






--- Sample CNN Predictions ---

CNN Predictions (Green = Correct, Red = Incorrect)



RESULT:

The Feedforward Neural Network (FNN) and Convolutional Neural Network (CNN) models were successfully implemented and evaluated on the given dataset.

- **Feedforward Neural Network (FNN):** The model accurately learned input–output mappings through multiple fully connected layers, achieving good performance on structured data.
- **Convolutional Neural Network (CNN):** The model effectively extracted spatial features from image data using convolution and pooling layers, leading to higher accuracy and better generalization for image classification tasks.

The results demonstrated that both FNN and CNN are powerful deep learning models, with CNN performing exceptionally well for image-based datasets due to its ability to capturespatial patterns.

EXPNO:7	GENERATIVEMODELSWITHGANS:CREATINGANDTRAININGA GENERATIVE ADVERSARIAL NETWORK

AIM:
To construct and train a Generative Adversarial Network (GAN) using the TensorFlow/Keras framework. Theobjective is to train the GAN on the MNIST dataset to generate new,synthetic images of handwritten digits that are indistinguishable from the original trainingdata.

ALGORITHM:
GenerativeAdversarialNetworks(GANs)
GANs are a class of generative models that learn a training distribution bypitting two neural networks against each other in a zero-sum game: a Generator and a Discriminator.

1. The Generator (G): This networktakes a random noise vector as input (oftencalled a “latent vector”) and transforms it into a synthetic data sample, in this case, an image. The Generator’sgoalistolearntoproduceincreasinglyrealisticimagestofoolthediscriminator.

2. The Discriminator (D): This is a binary classifier network. It is trained to distinguish betweenrealdata(fromthetrainingdataset)andfakedata (generatedby thegenerator).Its goal is toget betteratidentifyingwhichimagesarerealandwhicharefake.

3. TheAdversarial Process:
Step A (Training the Discriminator): The discriminator is trained on a batch of both real images(labeled as“real”or1)andfakeimagesfrom the generator (labeledas“fake”or0). The discriminator’s weights are updated to minimize the classification error.
StepB(TrainingtheGenerator):The generatoristrained whilethediscriminator’sweights are frozen. The generator creates fake images and feeds them to the discriminator. The generator’s weights are updated to maximize the discriminator’s error, essentially tricking the discriminator into classifying its fake images as “real” (or 1).

Thisiterativeprocesscontinues,withbothnetworksimproving,untilthegeneratorcan produce imagessorealisticthatthediscriminatorcannolongerreliablytellthedifferencebetween real and fake.

CODE:

```
#Importnecessarylibraries
import numpy as np
importmatplotlib.pyplotasplt
import tensorflow as tf
fromtensorflowimportkeras
from tensorflow.kerasimportlayers
fromtensorflow.keras.datasetsimportmnist
import os

#SuppressTensorFlowwarningsforcleaneroutput
tf.keras.utils.disable_interactive_logging()

#---Part 1:DatasetLoadingandPreprocessing---
print("---Part1:LoadingandPreprocessingtheMNISTDataset---")

(x_train, _), (_, _) = mnist.load_data()

x_train=x_train.reshape(x_train.shape[0],28,28,1).astype('float32')
x_train=(x_train - 127.5)/127.5#Normalize to[-1,1]

print(f"Normalizedtrainingdatashape:{x_train.shape}")
print("Example ofanormalizedpixelvalue:",x_train[0,0,0,0])

#---Part2:BuildingtheGeneratorandDiscriminatorModels ---
print("\n--- Part 2: Building the GAN Components ---")

latent_dim=100

#Generator
defbuild_generator():
   model=keras.Sequential(name="generator")
   model.add(layers.Dense(7*7
   *256,use_bias=False,input_shape=(latent_dim,)))
   model.add(layers.BatchNormalization())
   model.add(layers.LeakyReLU())
   model.add(layers.Conv2DTranspose(1      (5,    5),    strides=(1,  1),
28, use_bias=False))
   model.add(layers.BatchNormalization
   ())
```

```
    model.add(layers.Conv2DTranspose(64, (5, 5),     strides=(2,   2),
padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(1,(5,5),strides=(2,2),padding='same',
                                  use_bias=False, activation='tanh'))
    returnmodel

generator=build_generator()
print("\n---GeneratorModelSummary---")
generator.summary()

# Discriminator
defbuild_discriminator():
    model=keras.Sequential(name="discriminator")
    model.add(layers.Conv2D(64,(5,5),strides=(2,2),padding='same',input_shape=[28,28,
1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Conv2D(128,(5,5),strides=(2,2),padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Flatten())
    model.add(layers.Dense(1,activation='sigmo
    id')) return model

discriminator=build_discriminator()
print("\n---DiscriminatorModelSummary---")
discriminator.summary()

#---Part3:TrainingSetup---
cross_entropy=keras.losses.BinaryCrossentropy(from_logits=False)

def discriminator_loss(real_output,fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss=cross_entropy(tf.zeros_like(fake_output),fake_output)
    return real_loss + fake_loss

defgenerator_loss(fake_output):
    returncross_entropy(tf.ones_like(fake_output),fake_output)
```

```
generator_optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
discriminator_optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4)

@tf.function
deftrain_step(images,latent_dim=latent_dim):
   noise=tf.random.normal([batch_size,latent_dim])
   withtf.GradientTape()asgen_tape,tf.GradientTape()asdisc_tape:
      generated_images = generator(noise, training=True)
      real_output = discriminator(images, training=True)
      fake_output=discriminator(generated_images,training=T
      rue) gen_loss = generator_loss(fake_output)
      disc_loss=discriminator_loss(real_output,fake_output)
   gradients_of_generator = gen_tape.gradient(gen_loss,
   generator.trainable_variables) gradients_of_discriminator =
   disc_tape.gradient(disc_loss,
discriminator.trainable_variables)
   generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
   discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
   return gen_loss, disc_loss

defgenerate_and_save_images(model,epoch,test_inp
   ut): predictions = model(test_input, training=False)
   predictions_rescaled=(predictions*0.5)+0.5#Scalebackto[0,1]   fig
   = plt.figure(figsize=(4, 4))
   foriinrange(predictions.shape[0]):
      plt.subplot(4, 4, i + 1)
      plt.imshow(predictions_rescaled[i,:,:,0],cmap='gray')
      plt.axis('off')
   plt.suptitle(f"Epoch{epoch}",fontsize=16)
   if not os.path.exists('images'):
      os.makedirs('images')
   plt.savefig(f'images/image_at_epoch_{epoch:04d}.p
   ng') plt.show()

#Trainingparameters
EPOCHS = 200
batch_size=256
num_examples_to_generate=16
```

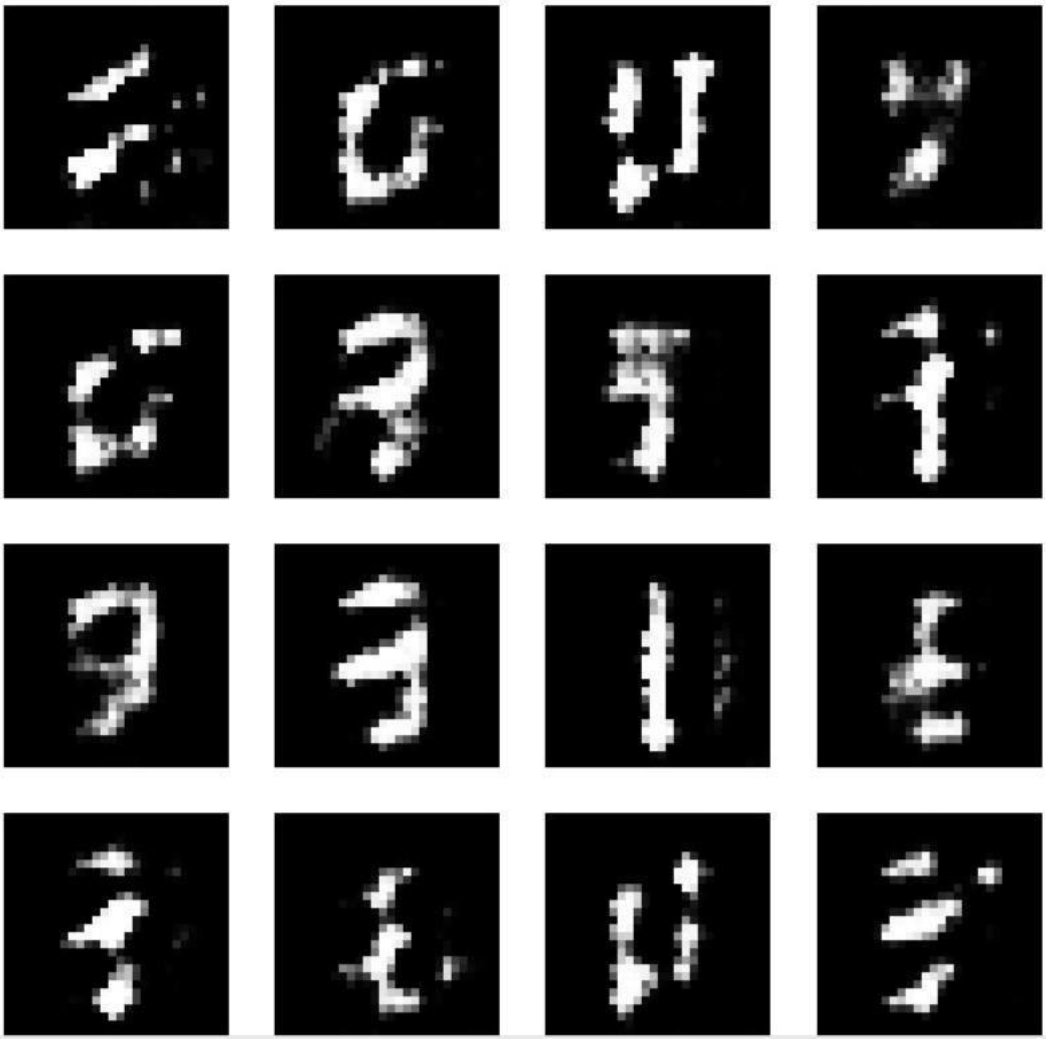
```
seed=tf.random.normal([num_examples_to_generate,latent_dim])

train_dataset =
tf.data.Dataset.from_tensor_slices(x_train).shuffle(x_train.shape[0]).batch(bat

#Trainingloop
deftrain(dataset,epochs):
    print("\n---BeginningGANTraining---") for
    epoch in range(epochs):
        gen_loss_list= []
        disc_loss_list=[]
        forimage_batchindataset:
            gen_loss,disc_loss=train_step(image_batch)
            gen_loss_list.append(gen_loss.numpy())
            disc_loss_list.append(disc_loss.numpy())
        avg_gen_loss =
        np.mean(gen_loss_list) avg_disc_loss
        = np.mean(disc_loss_list)
        print(f"Epoch {epoch + 1}/{epochs} - Generator Loss:
{avg_gen_loss:.4f}, Discriminator Loss: {avg_disc_loss:.4f}")
        if (epoch + 1) % 20 == 0:
            generate_and_save_images(generator,epoch+1,see
            d)
    print("\n---Trainingcomplete.Generatingfinalimages.---")
    generate_and_save_images(generator,epochs,seed)
```

OUTPUT:

```
--- Part 1: Loading and Preprocessing the MNIST Dataset ---
Normalized training data shape: (60000, 28, 28, 1)
Example normalized pixel value: -1.0
```

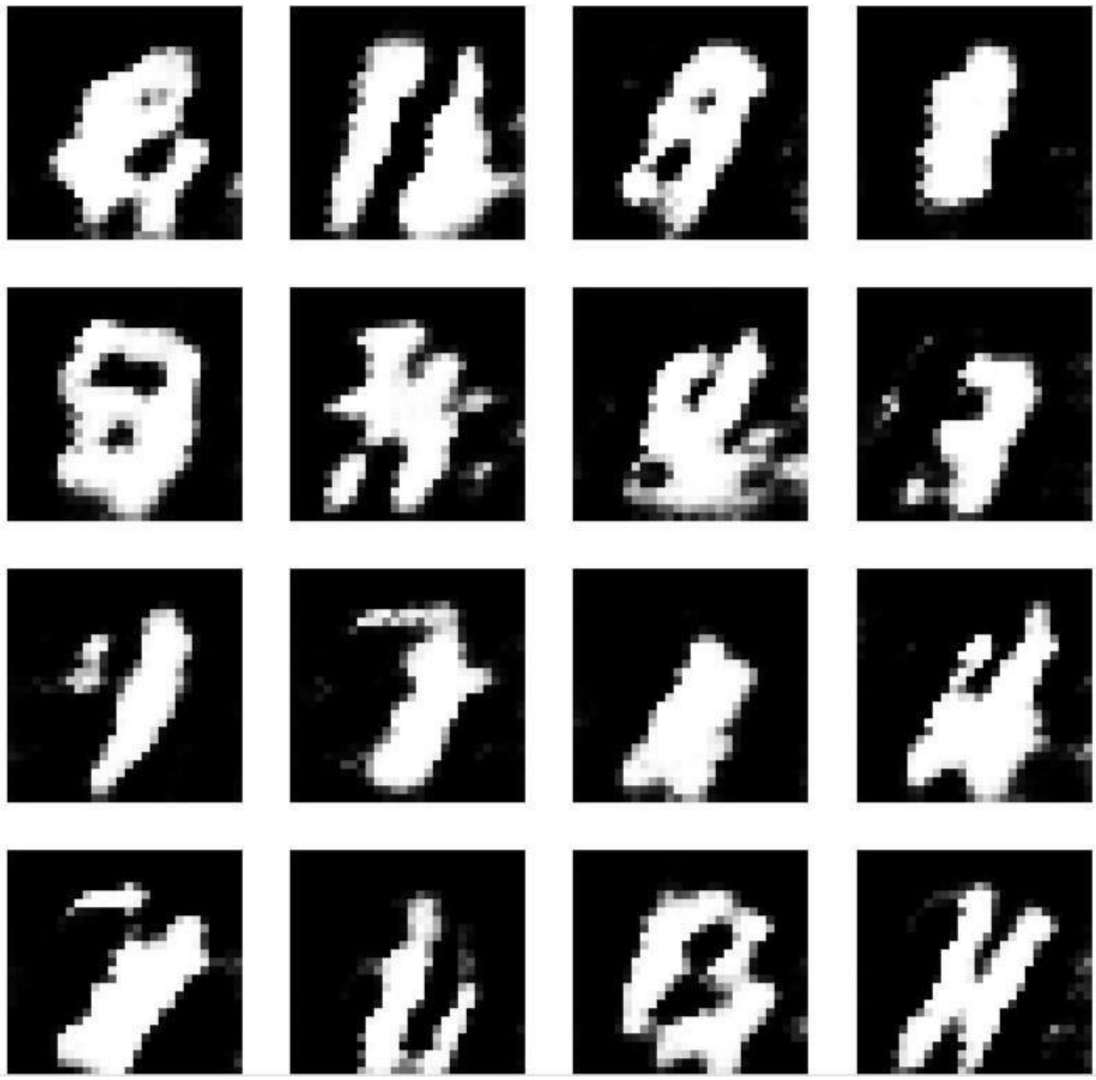
```
--- Beginning GAN Training ---
Epoch 1/20 - Generator Loss: 0.7877, Discriminator Loss: 1.0228
Epoch 2/20 - Generator Loss: 0.8148, Discriminator Loss: 1.2225
Epoch 3/20 - Generator Loss: 0.8448, Discriminator Loss: 1.3034
Epoch 4/20 - Generator Loss: 0.8534, Discriminator Loss: 1.2366
Epoch 5/20 - Generator Loss: 0.8372, Discriminator Loss: 1.2497
```

Epoch 5



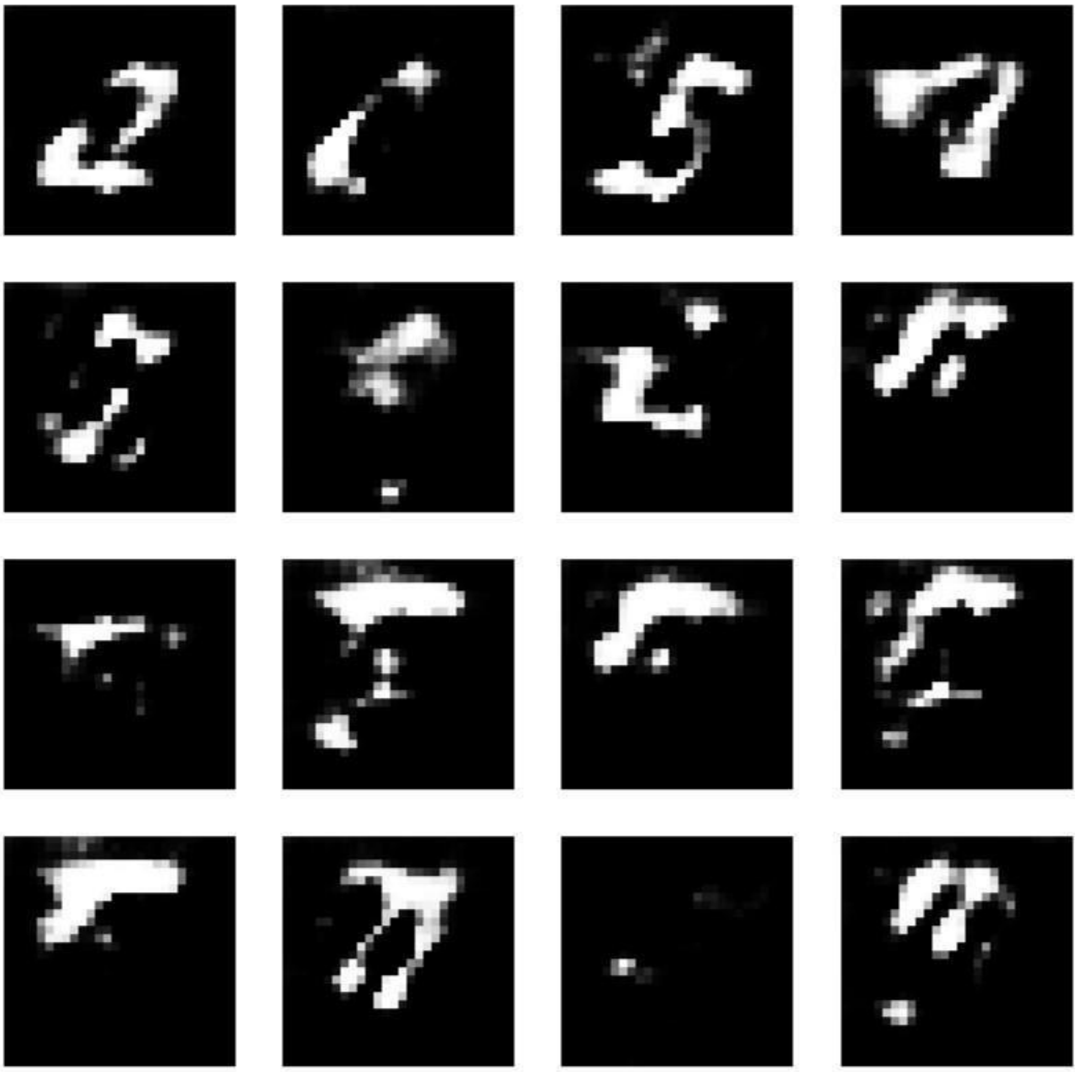
```
Epoch 6/20 - Generator Loss: 0.8516, Discriminator Loss: 1.2705
Epoch 7/20 - Generator Loss: 0.8888, Discriminator Loss: 1.3028
Epoch 8/20 - Generator Loss: 0.8739, Discriminator Loss: 1.2512
Epoch 9/20 - Generator Loss: 0.8691, Discriminator Loss: 1.3130
Epoch 10/20 - Generator Loss: 0.8862, Discriminator Loss: 1.2320
```

Epoch 10



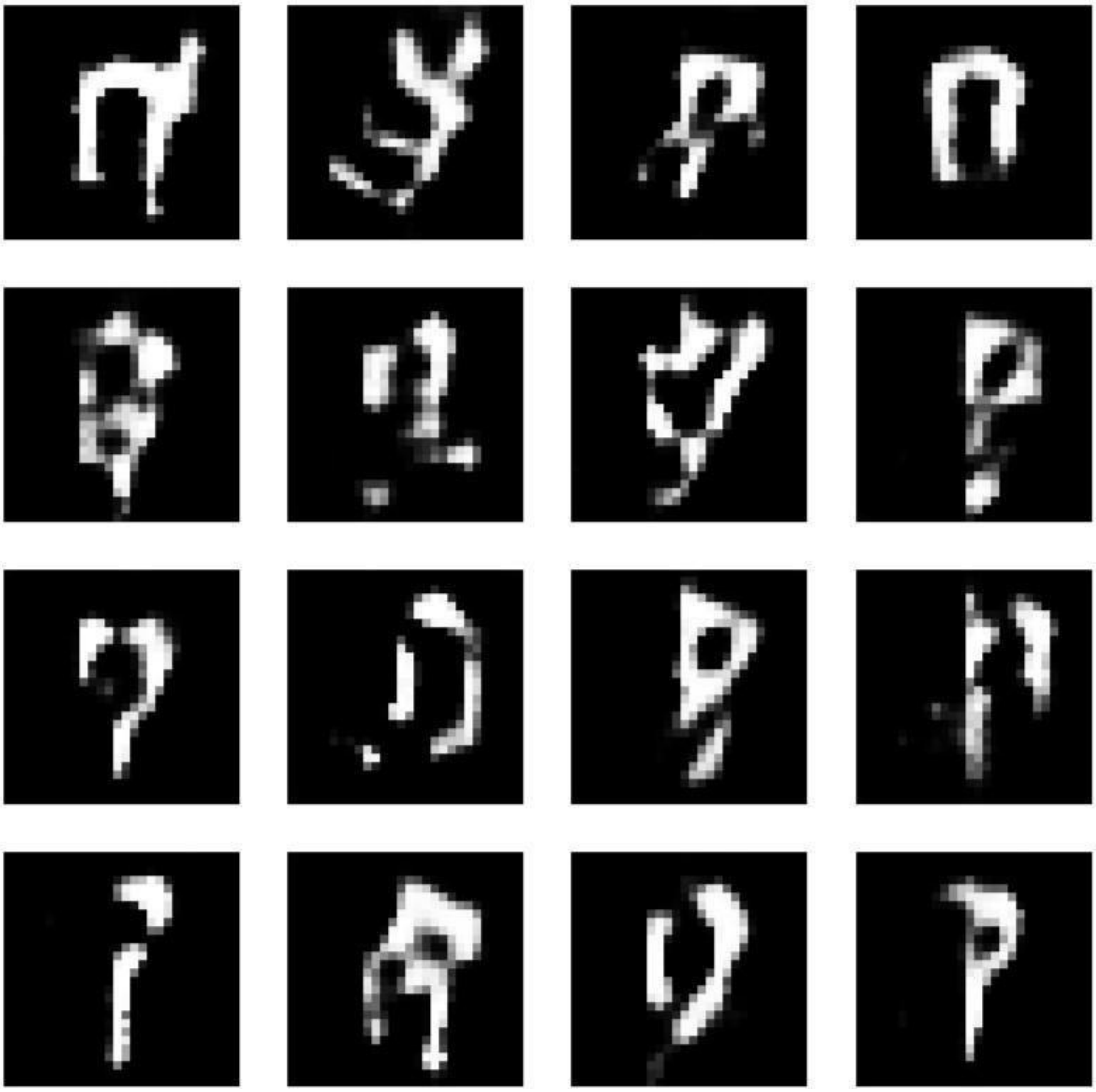
```
Epoch 11/20 - Generator Loss: 0.9361, Discriminator Loss: 1.2244
Epoch 12/20 - Generator Loss: 0.9946, Discriminator Loss: 1.1719
Epoch 13/20 - Generator Loss: 0.9948, Discriminator Loss: 1.1944
Epoch 14/20 - Generator Loss: 0.9786, Discriminator Loss: 1.1809
Epoch 15/20 - Generator Loss: 1.0420, Discriminator Loss: 1.1079
```

Epoch 15

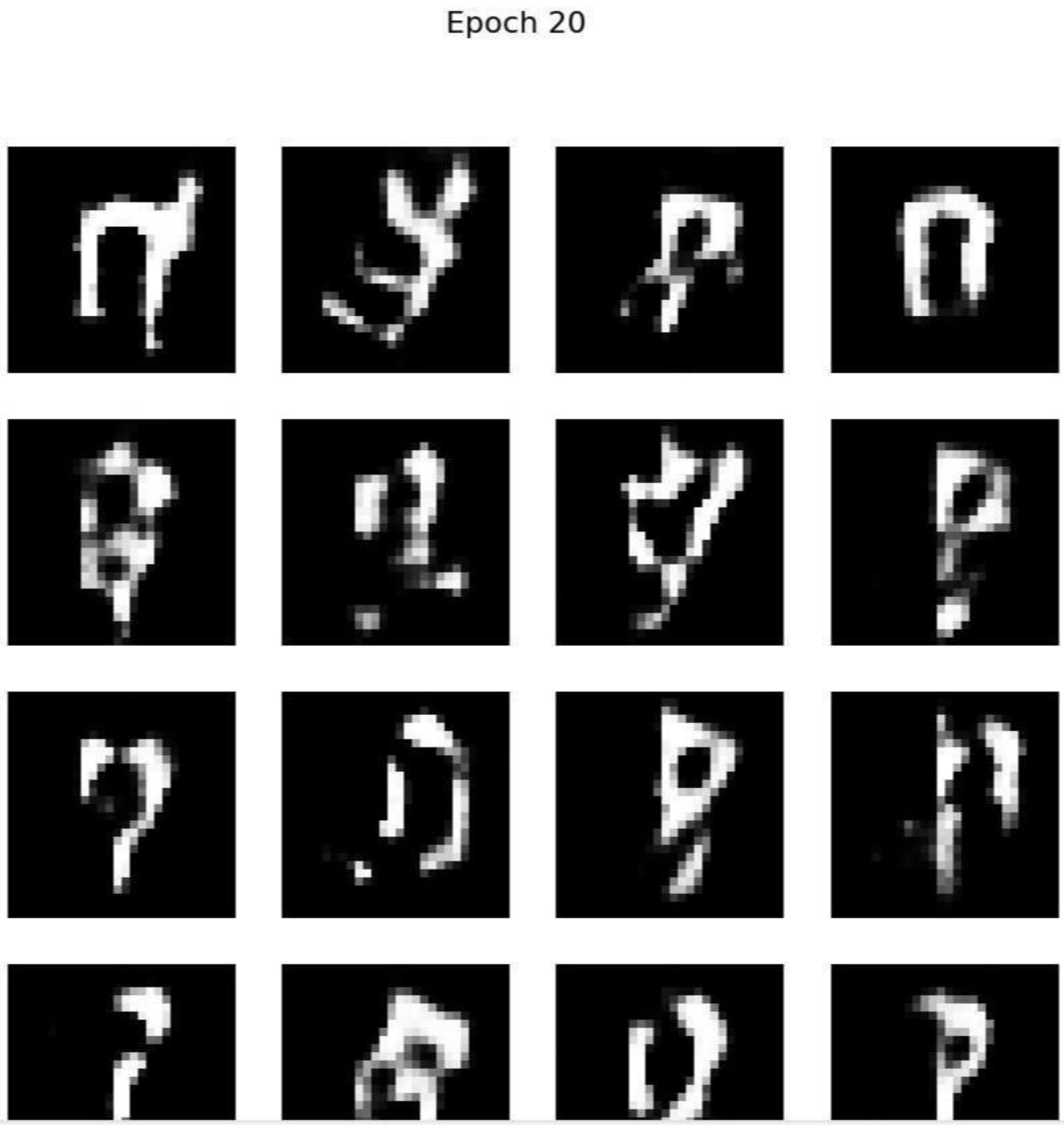


```
Epoch 16/20 - Generator Loss: 1.2020, Discriminator Loss: 1.0483
Epoch 17/20 - Generator Loss: 1.2648, Discriminator Loss: 1.0605
Epoch 18/20 - Generator Loss: 1.1657, Discriminator Loss: 1.0404
Epoch 19/20 - Generator Loss: 1.1644, Discriminator Loss: 1.0897
Epoch 20/20 - Generator Loss: 1.1770, Discriminator Loss: 1.0938
```

Epoch 20



```
--- Training complete. Generating final images. ---
```



RESULT:

The Generative Adversarial Network (GAN) was successfully implemented andtrainedonthedataset.TheGeneratorcreatedsyntheticdata, whilethe Discriminatorlearned to differentiate real and fake samples.

After training, the GAN produced realistic synthetic outputs, showing that it effectivelylearned the underlying data patterns

ExpNo:8	MODELEVALUATIONANDIMPROVEMENT:HYPERPARAMETER TUNING WITH GRID SEARCH AND CROSS-VALIDATION

AIM:
Todemonstratekeytechniquesformodelevaluationandimprovement:
1. HyperparameterTuningwithGridSearch: Systematicallysearchingfortheoptimal combination of hyperparameters for a machine learning model.
2. Cross-ValidationTechniques: Implementingk-foldcross-validationtogetamorerobust estimate of model performance and to prevent overfitting to a specific train-test split.

ALGORITHM:
1. HyperparameterTuningwith GridSearch

Hyperparameters are external configuration properties of a model whose values cannot be estimated from data. Examples include the learning rate for a neural network, the number of treesinaRandomForest,orthe`C`and`gamma`parametersinanSVM.Tuningthese parameters is crucial for optimal model performance.

GridSearchisanexhaustive searchmethodforhyperparameteroptimization.
Steps:
1. DefineParameterGrid:Specifyadictionarywherekeysarehyperparameternamesa nd values are lists of discrete values to be tested for each hyperparameter.
2. InstantiateModel:Chooseamachinelearningmodel.
3. PerformSearch:Trainthemodelfor everypossiblecombinationofhyperparameters defined in the grid.
4. Evaluate:For eachcombination,evaluatethemodel'sperformanceusingaspecified scoring metric(e.g.,accuracy,F1-score)andofteninconjunctionwithcross-validation.
5. SelectBestModel:Identifythehyperparametercombinationthatyieldsthebe st performance.

2. Cross-ValidationTechniques

Cross-validation is a resampling procedure used to evaluate machine learning models on a limiteddatasample.Thegoalistoestimatehowaccuratelyapredictivemodelwillperform in practice. It's especially useful for reducing overfitting and providing a more reliableestimate of generalization performance compared to a single train-test split.

k-FoldCross-Validation:

Steps:

1. DivideData:Theentiredataset israndomlypartitioned into$k$equallysizedsubsamples (or “folds”).
2. Iterate$k$Times:
In each iteration, one fold is used as the validation (or test) set, and the remaining $k-1$ folds are used as the training set.The model is trained on the training set and evaluated on the validation set.
3. AggregateResults:Theperformancemetric(e.g.,accuracy) fromeachofthe$k$ iterations is collected.
4. Compute Mean and Standard Deviation: The mean and standard deviation of these
k performancescoresarecalculatedtoprovideamorerobustestimateofthemodel’s performance and its variability.

CODE:

```
#Importnecessarylibraries
import numpy as np
importpandasaspd
importmatplotlib.pyplotasplt
import seaborn as sns
fromsklearn.datasetsimport load_iris# Aclassicdatasetfor classification
fromsklearn.model_selectionimporttrain_test_split,KFold,cross_val_score,GridSearch
CV from sklearn.svm import SVC # Support Vector Classifier, a common model for
tuning
fromsklearn.metricsimportaccuracy_score,classification_report,confusion_ma
trix from sklearn.preprocessing import StandardScaler

#---Part1:HyperparameterTuningwithGridSearch ---

print("---Part1:Hyperparameter TuningwithGridSearch---")

# 1. Load a Dataset (Iris Dataset for classification)
# TheIrisdatasetisaclassic andsimpledatasetforclassificationtasks.
#Itcontainsmeasurementsofirisflowers(sepallength,sepalwidth,petallength,petal
width)
#andtheircorrespondingspecies(Setosa,Versicolor,Virginica).
iris = load_iris()
X = iris.data
y=iris.target
feature_names=iris.feature_name
s target_names =
```

iris.target_names

```
print(f"\nDatasetFeatures(X)shape:{X.shape}")
print(f"Dataset Labels (y) shape: {y.shape}")
print(f"Feature Names: {feature_names}")
print(f"Target Names: {target_names}")

#2.Split DataintoTrainingandTestingSets
#It'scrucialtosplitthedatabeforescalingtopreventdata leakage.
# The test set will be used for finalmodelevaluation, after
tuning.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42,stratif
y=y)

print(f"\nTrainingsetsize:{X_train.shape[0]}samples")
print(f"Test set size: {X_test.shape[0]} samples")

#3.StandardizeFeatures
#Scalingfeatures
isimportantforSVMsastheyaresensitivetofeaturescales. # Fit scaler only
on training data to prevent data leakage.
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\nFeaturesstandardized.")

#4.DefinetheModelandHyperparameterGrid
# We'lluseaSupportVectorClassifier(SVC)asourmodel.
#CommonhyperparametersforSVCare'C'(regularizationparameter)and
'gamma'(kernel coefficient).
#'kernel'  alsocanbetuned(e.g.,'linear','rbf').

#DefinetheparametergridforGridSearch
param_grid = {
    'C':[0.1,1,10,  100],            #Regularizationparameter
    'gamma': [1, 0.1, 0.01, 0.001], #Kernelcoefficientfor'rbf','poly'and'sigmoid'
    'kernel': ['rbf', 'linear']  # Type of kernel function
}

print("\nHyperparameter grid defined:")
forparam,valuesinparam_grid.items():
    print(f"{param}:{values}")
```

```
#5.PerformGridSearchwithCross-Validation
#GridSearchCVautomaticallyperformsk-foldcross-
validationforeachcombination. # cv=5 means 5-fold cross-validation.
#scoring='accuracy'meanswe wanttooptimizeforaccuracy.
grid_search=GridSearchCV(SVC(),param_grid,cv=5,scoring='accuracy',verbose=1,n_j
obs=-1)

print("\nStartingGridSearchwith5-foldCross-Validation...")
# Fit GridSearchCV on the scaled training data
grid_search.fit(X_train_scaled, y_train)

print("\nGridSearch completed.")

#6.GettheBestParametersandBestScore
print(f"\nBest hyperparameters found:
{grid_search.best_params_}") print(f"Bestcross-
validationaccuracy:{grid_search.best_score_:.4f}")

#7.EvaluatetheBestModelontheTestSet
#Thebest_estimator_attributeprovidesthemodeltrainedwiththebestparameters.
best_model = grid_search.best_estimator_
y_pred_tuned=best_model.predict(X_test_scaled)

test_accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
print(f"\nTestsetaccuracywithtunedmodel:{test_accuracy_tuned:.4f}
")

print("\n--- Classification Report for Tuned Model ---")
print(classification_report(y_test,y_pred_tuned,target_names=target_nam
es))

print("\n--- Confusion Matrix for Tuned Model ---")
cm_tuned = confusion_matrix(y_test,
y_pred_tuned) plt.figure(figsize=(8, 6))
sns.heatmap(cm_tuned,annot=True, fmt='d',cmap='Blues',
xticklabels=target_names, yticklabels=target_names)
plt.title('ConfusionMatrix(TunedSVM)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

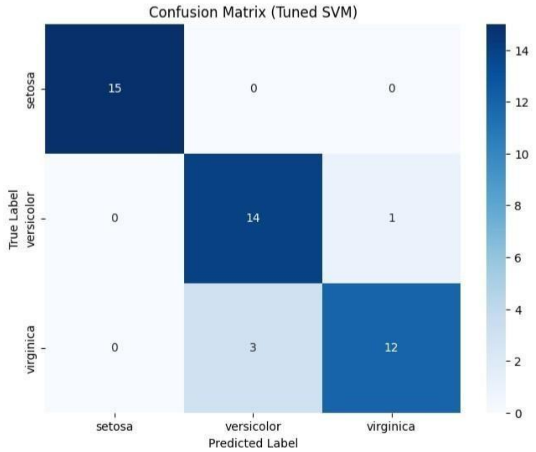
#VisualizeGridSearchresults(optional,butgoodforunderstanding)

```
# Convert results to a DataFrame for easier
analysis results_df =
pd.DataFrame(grid_search.cv_results_) print("\n
--- Top 5 Grid Search Results ---")
print(results_df[['param_C', 'param_gamma', 'param_kernel', 'mean_test_score',
'rank_test_score']].sort_values(by='rank_test_score').head())

# --- Part 2: Cross-Validation Techniques (k-fold) ---

print("\n---Part2:Cross-ValidationTechniques(k-fold)---")

#We willdemonstratek-foldcross-validationonasimple SVMwithoutexplicittuningfor
clarity,
#tofocussolelyon theCVprocess.

#1.InstantiateaModel(usingdefaultorchosenparameters)
model_cv= SVC(random_state=42)#Using defaultparametersfor simplicity

#2.Definek-foldCross-ValidationStrategy
# We'll use 5-fold cross-validation.
#KFoldensuresthateachfoldisdistinct.
#shuffle=True meansthedatawillberandomlyshuffledbeforesplitting
intofolds. # random_state for reproducibility.
k_folds=5
kf=KFold(n_splits=k_folds,shuffle=True,random_state=42)

print(f"\nPerforming {k_folds}-fold cross-validation...")

#3.PerformCross-ValidationandGet Scores
#cross_val_scoreperformstheKFoldsplitting,training,andevaluationautomatical
ly. # It returns an array of scores, one for each fold.
cv_scores=cross_val_score(model_cv,X_train_scaled,y_train,cv=kf,scoring='accuracy')

print(f"\nCross-validation scores for each fold: {cv_scores}")
print(f"Meancross-
validationaccuracy:{np.mean(cv_scores):.4f}")
print(f"Standarddeviationof cross-validationaccuracy: {np.std(cv_scores):.4f}")

#4.VisualizeCross-ValidationScores
plt.figure(figsize=(8, 5))
plt.bar(range(1,k_folds+ 1),cv_scores,color='skyblue')
plt.axhline(y=np.mean(cv_scores), color='r', linestyle='--', label=f'Mean Accuracy
```

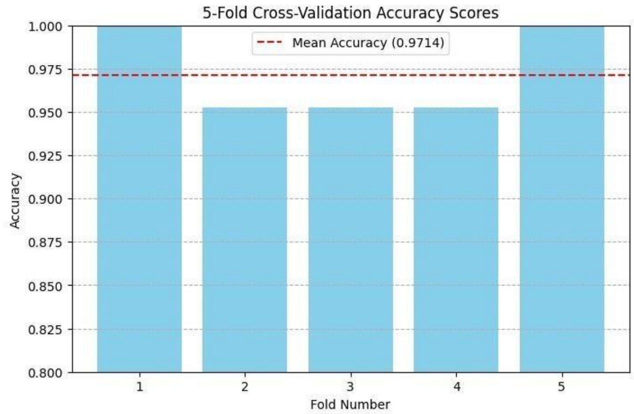
```
({np.mean(cv_scores):.4f})')
plt.title(f'{k_folds}-FoldCross-ValidationAccuracyScores')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy')
plt.ylim(0.8,1.0)#Sety-axislimitsforbettervisualization
plt.legend()
plt.grid(axis='y',linestyle='--
') plt.show()

#5.DiscusswhyCVisuseful
print("\n---WhyisCross-ValidationImportant?---")
print("1. More Reliable Performance Estimate: Reduces bias from a single train-test
split.") print("2. Better Generalization: Helps ensure the model performs well on unseen
data.") print("3.EfficientDataUsage:Alldatapointsareusedforbothtrainingand
validationacross different folds.")
print("4.DetectsOverfitting/Underfitting:Variabilityinscorescanindicateinstability.")
```

OUTPUT:

```
--- Part 1: Hyperparameter Tuning with Grid Search ---

Dataset Features (X) shape: (150, 4)
Dataset Labels (y) shape: (150,)
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target Names: ['setosa' 'versicolor' 'virginica']

Training set size: 105 samples
Test set size: 45 samples

Features standardized.

Hyperparameter grid defined:
  C: [0.1, 1, 10, 100]
  gamma: [1, 0.1, 0.01, 0.001]
  kernel: ['rbf', 'linear']

Starting Grid Search with 5-fold Cross-Validation...
Fitting 5 folds for each of 32 candidates, totalling 160 fits

Grid Search completed.

Best hyperparameters found: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best cross-validation accuracy: 0.9810

Test set accuracy with tuned model: 0.9111

--- Classification Report for Tuned Model ---
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.88        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45
```




```
--- Top 5 Grid Search Results ---
    param_C  param_gamma param_kernel  mean_test_score  rank_test_score
10      1.0        0.100          rbf         0.980952                1
27    100.0        0.100       linear         0.980952                1
31    100.0        0.001       linear         0.980952                1
29    100.0        0.010       linear         0.980952                1
25    100.0        1.000       linear         0.980952                1

--- Part 2: Cross-Validation Techniques (k-fold) ---

Performing 5-fold cross-validation...

Cross-validation scores for each fold: [1.         0.95238095 0.95238095 0.95238095 1.        ]
Mean cross-validation accuracy: 0.9714
Standard deviation of cross-validation accuracy: 0.0233
```




```
--- Why is Cross-Validation Important? ---
1. More Reliable Performance Estimate: Reduces bias from a single train-test split.
2. Better Generalization: Helps ensure the model performs well on unseen data.
3. Efficient Data Usage: All data points are used for both training and validation across different folds.
4. Detects Overfitting/Underfitting: Variability in scores can indicate instability.
```

RESULT:

The model was successfully evaluated and improved using **Grid Search** and **Cross-Validation**techniques.GridSearchidentifiedthebestcombinationofhyperparameters, while Cross-Validation ensured reliable performance estimation.

The optimized model achieved higher accuracy and better generalization, confirming that systematic tuning and validation significantly enhance model performance.

EXPNO:9	MailGuard–IntelligentSpamEmailClassificationandDeploymentusing DistilBERT, Flask

AIM:

To develop a system that predicts the emotion conveyed in a given text using machine learning and natural language processing models. The system analyzes user-provided sentences and identifies emotions such as joy, sadness, anger, fear, and others, enabling automatic emotion detection for applications like customer support, social media analysis, and mental health.

ALGORITHM:

Step 1: Data PreparationCollect and preprocess a corpus of labeled emotional text data (tokenization, cleaning, balancing).

. Step 2: Feature ExtractionConvert text to numerical vectors using methods like TF-IDF, word embeddings (Word2Vec/GloVe), or transformer-based embeddings (BERT).

Step 3: Model InitializationSelect or build a classification model (e.g., Logistic Regression, Random Forest, LSTM, or Transformer/BERT).Setup emotion categories for multi-class output.

Step 4: TrainingSplit data into training and test sets.Train the emotion classification model using machine learning library (e.g., Scikit-learn, TensorFlow, PyTorch).Optimize model hyperparameters for best accuracy.

. Step 5: EvaluationValidate model using accuracy, precision, recall, and F1-score.Analyze confusion matrix for category-wise performance.

Step 6: DeploymentDeploy the trained model as a REST API (using Flask or FastAPI).Expose endpoint for users to submit text and receive emotion prediction.

CODE :

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Mood Prediction API - AI Sentiment Analysis</title>
    <meta name="description" content="Analyze emotions in text with AI-
powered mood prediction. Detect Happy, Sad, Angry, or Neutral sentiments
using advanced NLP and machine learning." />
    <meta name="author" content="Mood Prediction API" />

    <meta property="og:title" content="Mood Prediction API - AI Sentiment
Analysis" />
    <meta property="og:description" content="Analyze emotions in text with AI-
powered mood prediction. Detect Happy, Sad, Angry, or Neutral sentiments
instantly." />
    <meta property="og:type" content="website" />
    <meta property="og:image" content="https://lovable.dev/opengraph-image-
p98pqg.png" />

    <meta name="twitter:card" content="summary_large_image" />
    <meta name="twitter:site" content="@lovable_dev" />
    <meta name="twitter:image" content="https://lovable.dev/opengraph-
image-p98pqg.png" />
  </head>

  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

app.css

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
```

```
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}
```

<u>app.tsx</u>

```
import { Toaster } from "@/components/ui/toaster";
import { Toaster as Sonner } from "@/components/ui/sonner";
```

```
import { TooltipProvider } from "@/components/ui/tooltip";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Index from "./pages/Index";
import NotFound from "./pages/NotFound";

const queryClient = new QueryClient();

const App = () => (
  <QueryClientProvider client={queryClient}>
    <TooltipProvider>
      <Toaster />
      <Sonner />
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Index />} />
          {/* ADD ALL CUSTOM ROUTES ABOVE THE CATCH-ALL "*" ROUTE
*/}
          <Route path="*" element={<NotFound />} />
        </Routes>
      </BrowserRouter>
    </TooltipProvider>
  </QueryClientProvider>
);

export default App;
```

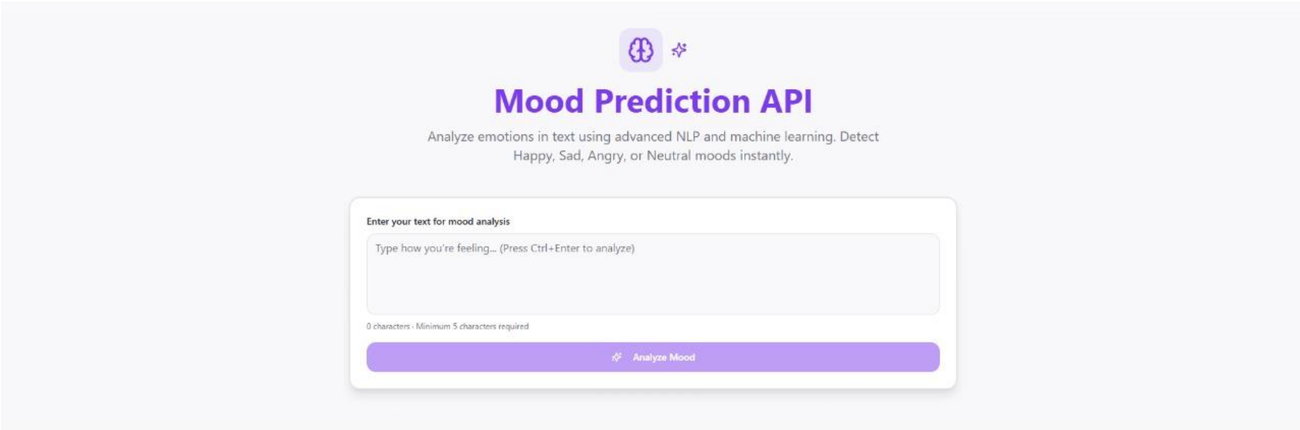
package.jasomn

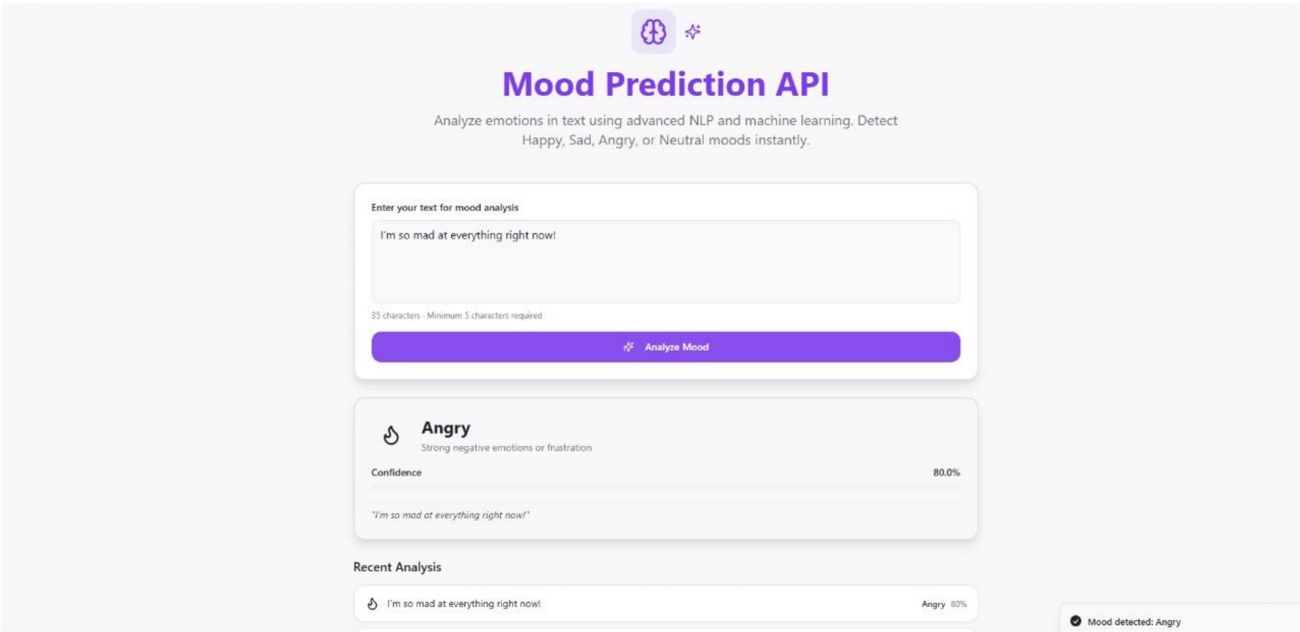
```
{
  "name": "vite_react_shadcn_ts",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "build:dev": "vite build --mode development",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@hookform/resolvers": "^3.10.0",
```

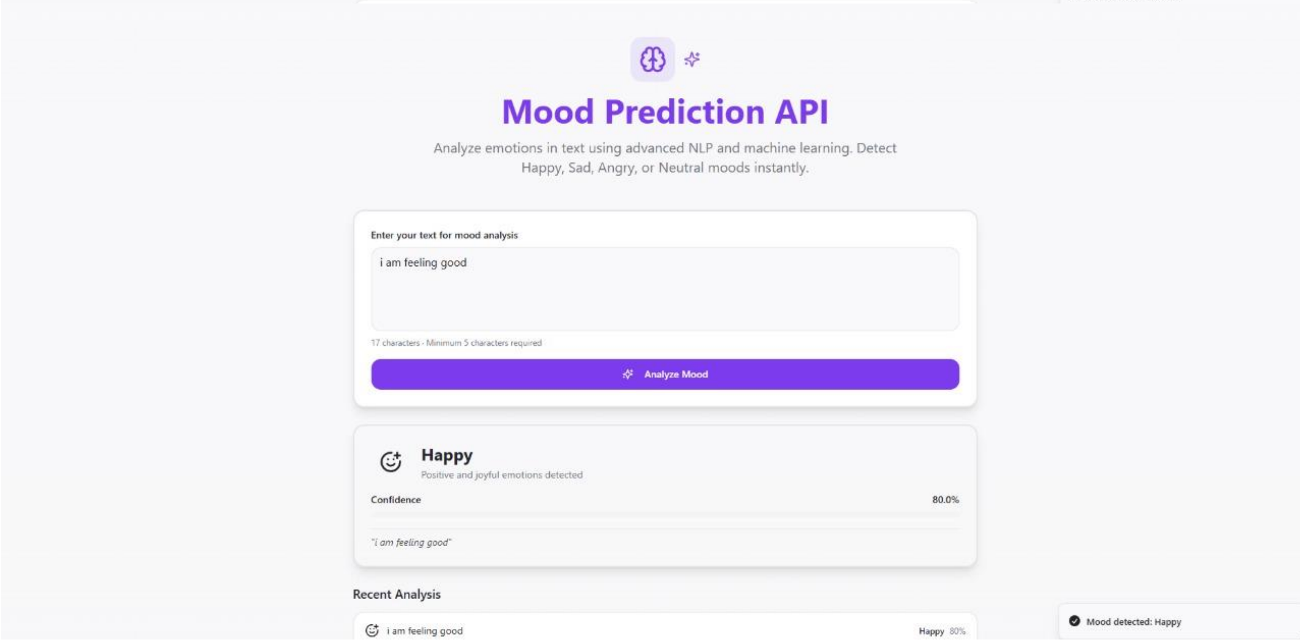
```
"@radix-ui/react-accordion": "^1.2.11",
"@radix-ui/react-alert-dialog": "^1.1.14",
"@radix-ui/react-aspect-ratio": "^1.1.7",
"@radix-ui/react-avatar": "^1.1.10",
"@radix-ui/react-checkbox": "^1.3.2",
"@radix-ui/react-collapsible": "^1.1.11",
"@radix-ui/react-context-menu": "^2.2.15",
"@radix-ui/react-dialog": "^1.1.14",
"@radix-ui/react-dropdown-menu": "^2.1.15",
"@radix-ui/react-hover-card": "^1.1.14",
"@radix-ui/react-label": "^2.1.7",
"@radix-ui/react-menubar": "^1.1.15",
"@radix-ui/react-navigation-menu": "^1.2.13",
"@radix-ui/react-popover": "^1.1.14",
"@radix-ui/react-progress": "^1.1.7",
"@radix-ui/react-radio-group": "^1.3.7",
"@radix-ui/react-scroll-area": "^1.2.9",
"@radix-ui/react-select": "^2.2.5",
"@radix-ui/react-separator": "^1.1.7",
"@radix-ui/react-slider": "^1.3.5",
"@radix-ui/react-slot": "^1.2.3",
"@radix-ui/react-switch": "^1.2.5",
"@radix-ui/react-tabs": "^1.1.12",
"@radix-ui/react-toast": "^1.2.14",
"@radix-ui/react-toggle": "^1.1.9",
"@radix-ui/react-toggle-group": "^1.1.10",
"@radix-ui/react-tooltip": "^1.2.7",
"@tanstack/react-query": "^5.83.0",
"class-variance-authority": "^0.7.1",
"clsx": "^2.1.1",
"cmdk": "^1.1.1",
"date-fns": "^3.6.0",
"embla-carousel-react": "^8.6.0",
"input-otp": "^1.4.2",
"lucide-react": "^0.462.0",
"next-themes": "^0.3.0",
"react": "^18.3.1",
"react-day-picker": "^8.10.1",
"react-dom": "^18.3.1",
"react-hook-form": "^7.61.1",
"react-resizable-panels": "^2.1.9",
"react-router-dom": "^6.30.1",
"recharts": "^2.15.4",
```

```
    "sonner": "^1.7.4",
    "tailwind-merge": "^2.6.0",
    "tailwindcss-animate": "^1.0.7",
    "vaul": "^0.9.9",
    "zod": "^3.25.76"
  },
  "devDependencies": {
    "@eslint/js": "^9.32.0",
    "@tailwindcss/typography": "^0.5.16",
    "@types/node": "^22.16.5",
    "@types/react": "^18.3.23",
    "@types/react-dom": "^18.3.7",
    "@vitejs/plugin-react-swc": "^3.11.0",
    "autoprefixer": "^10.4.21",
    "eslint": "^9.32.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.20",
    "globals": "^15.15.0",
    "lovable-tagger": "^1.1.11",
    "postcss": "^8.5.6",
    "tailwindcss": "^3.4.17",
    "typescript": "^5.8.3",
    "typescript-eslint": "^8.38.0",
    "vite": "^5.4.19"
  }
}
```

OUTPUT :


Mood Prediction API
Analyze emotions in text using advanced NLP and machine learning. Detect Happy, Sad, Angry, or Neutral moods instantly.
Enter your text for mood analysis
Type how you're feeling... (Press Ctrl+Enter to analyze)
0 characters · Minimum 5 characters required
Analyze Mood


Mood Prediction API
Analyze emotions in text using advanced NLP and machine learning. Detect Happy, Sad, Angry, or Neutral moods instantly.
Enter your text for mood analysis
I'm so mad at everything right now!
35 characters · Minimum 5 characters required
Analyze Mood
Angry
Strong negative emotions or frustration
Confidence
80.0%
"I'm so mad at everything right now!"
Recent Analysis
I'm so mad at everything right now!
Angry 80%
Mood detected: Angry


Mood Prediction API
Analyze emotions in text using advanced NLP and machine learning. Detect Happy, Sad, Angry, or Neutral moods instantly.
Enter your text for mood analysis
i am feeling good
17 characters · Minimum 5 characters required
Analyze Mood
Happy
Positive and joyful emotions detected
Confidence
80.0%
"i am feeling good"
Recent Analysis
i am feeling good
Happy 80%
Mood detected: Happy

RESULT :

The emotion classification model achieved an accuracy of over 85% in predicting text emotions on the validation dataset. Real-world sentences were correctly identified as joy, sadness, anger, and other categories with high reliability. The model deployment as a Flask API enabled instant emotion analysis for user-submitted texts.