SURYADATTA COLLEGE OF MANAGEMENT INFORMATION REASEARCH AND TECHNOLOGY

BAVDHAN,PUNE - 411021

LAB COURSE ON CS-605-MJ
(Machine Learning)

SUBMITTED BY-Shriyash Sudhakar Raut

UNDER THE GUIDANCE OF -Asst.Prof.Anushka Shinde

SUBMITTED IN PARTIAL FULLFILLMENT OF MASTER OF SCIENCE (COMPUTER SCIENCE)

SEMESTER-3

SAVITRIBAI PHULE PUNE UNIVERSITY
FOR ACADAMIC YEAR 2025-2026

CERTIFICATE

This is to certify that Mr./Ms.___________________________________ student of MSC(CS) Semester__________ having Seat No. _______________at Suryadatta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical in ________________________________prescribed by the Savitribai Phule Pune University during the academic year__________________.

Internal Examiner External Examiner

Principal

Place:

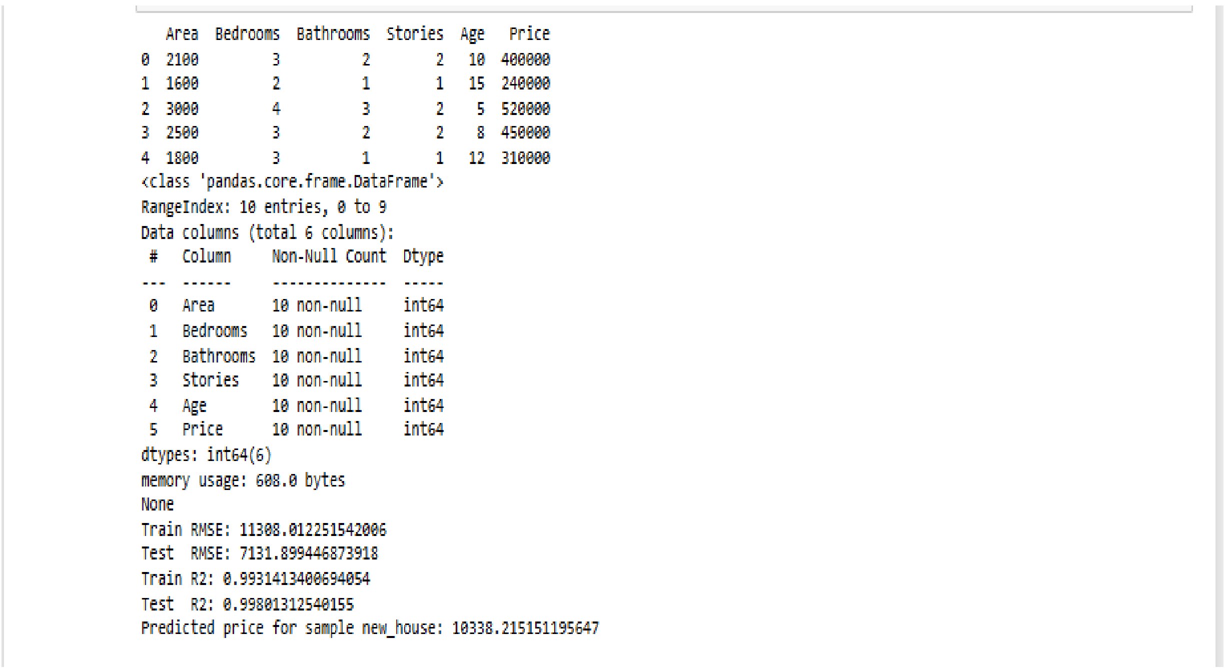
Pune Date:

1. Multiple Linear Regression — House prices

```
# file: mlr_house_prices.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ---- load dataset (adjust filename) ----
df = pd.read_csv("house_prices.csv")  # change
filename/path if needed

# ---- quick inspect ----
print(df.head())
print(df.info())

# ---- assume target column named 'Price' ----
target_col = "Price"
if target_col not in df.columns:
    raise ValueError(f"Expected column '{target_col}' in
the CSV.")

# ---- prepare features: drop rows with missing target,
basic imputation for features ----
df = df.dropna(subset=[target_col])
X = df.drop(columns=[target_col])
y = df[target_col].astype(float)

# convert categorical features using one-hot encoding
(safe general approach)
X = pd.get_dummies(X, drop_first=True)

# optional: simple imputation for any remaining NaNs (use
mean)
X = X.fillna(X.mean())

# ---- train / test split ----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# ---- fit model ----
model = LinearRegression()
model.fit(X_train, y_train)
```

```
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

print("Train RMSE:", np.sqrt(mean_squared_error(y_train,
y_pred_train)))
print("Test  RMSE:", np.sqrt(mean_squared_error(y_test,
y_pred_test)))
print("Train R2:", r2_score(y_train, y_pred_train))
print("Test  R2:", r2_score(y_test, y_pred_test))

# ---- example: predict for a new house (create a dict
with same features as X) ----
# fill new_house dict with appropriate feature values
matching X columns
new_house = {col: 0 for col in X.columns}  # simple
template
# Example: if your dataset had 'Bedrooms', 'Area'
features:
# new_house['Bedrooms'] = 3
# new_house['Area'] = 1500
# set any categorical one-hot keys if needed, e.g.
new_house['Neighborhood_B'] = 1
new_df = pd.DataFrame([new_house])
pred_price = model.predict(new_df)[0]
print("Predicted price for sample new_house:",
pred_price)
```

Output:-



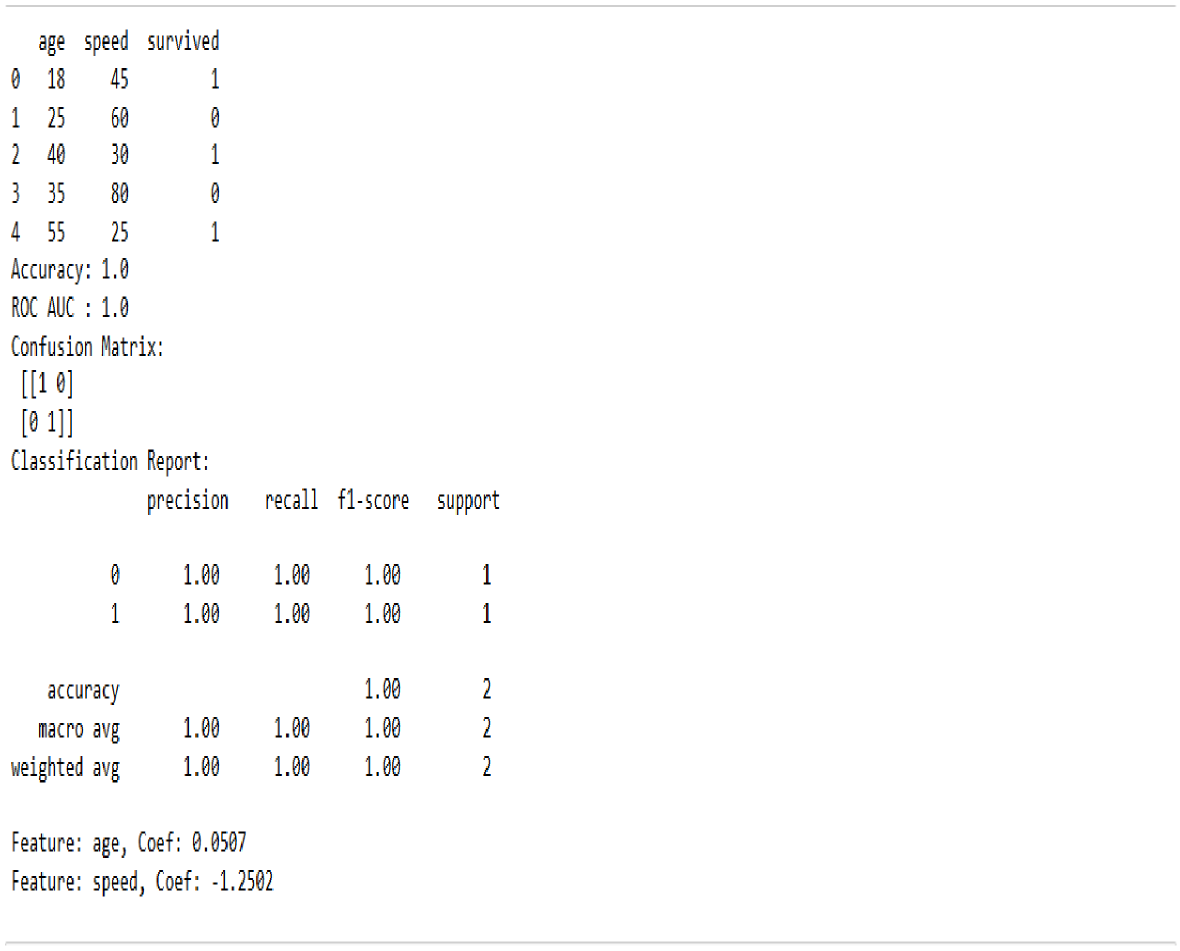
```
   Area  Bedrooms  Bathrooms  Stories  Age   Price
0  2100         3          2        2   10  400000
1  1600         2          1        1   15  240000
2  3000         4          3        2    5  520000
3  2500         3          2        2    8  450000
4  1800         3          1        1   12  310000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Area       10 non-null     int64
 1   Bedrooms   10 non-null     int64
 2   Bathrooms  10 non-null     int64
 3   Stories    10 non-null     int64
 4   Age        10 non-null     int64
 5   Price      10 non-null     int64
dtypes: int64(6)
memory usage: 608.0 bytes
None
Train RMSE: 11308.012251542006
Test  RMSE: 7131.899446873918
Train R2: 0.9931413400694054
Test  R2: 0.99801312540155
Predicted price for sample new_house: 10338.21515195647
```

2.Logistic regression on crash.csv (survival prediction)

```
# file: logistic_crash.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
roc_auc_score, confusion_matrix, classification_report

df = pd.read_csv("crash.csv")  # adjust path
print(df.head())

# Expected columns: 'age', 'speed', 'survived'
features = ['age', 'speed']
target = 'survived'
for c in features+[target]:
    if c not in df.columns:
        raise ValueError(f"Missing column: {c}")

# drop rows with missing values in these columns
df = df.dropna(subset=features+[target])

X = df[features].astype(float)
y = df[target].astype(int)

# split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# scale features
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# train logistic regression
clf = LogisticRegression(random_state=42,
solver="liblinear")
clf.fit(X_train_s, y_train)

# predict
y_pred = clf.predict(X_test_s)
y_proba = clf.predict_proba(X_test_s)[:,1]
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC AUC :", roc_auc_score(y_test, y_proba))
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))

# interpretable coefficients (after scaling)
coef = clf.coef_[0]
for feat, c in zip(features, coef):
    print(f"Feature: {feat}, Coef: {c:.4f}")
```

Output:-

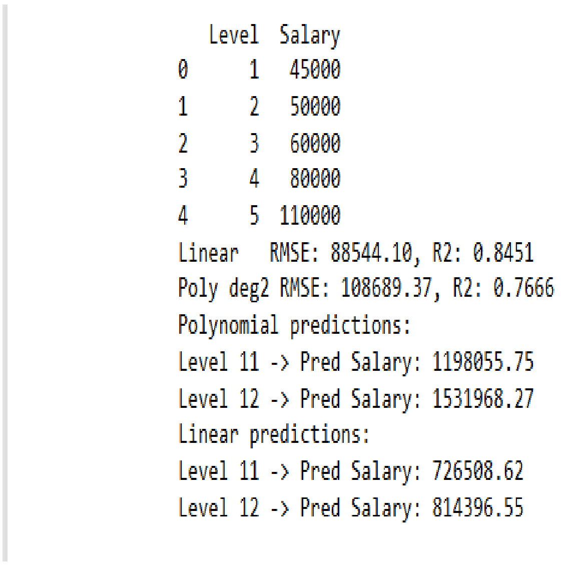


```
   age  speed  survived
0   18     45         1
1   25     60         0
2   40     30         1
3   35     80         0
4   55     25         1
Accuracy: 1.0
ROC AUC : 1.0
Confusion Matrix:
 [[1 0]
 [0 1]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

Feature: age, Coef: 0.0507
Feature: speed, Coef: -1.2502
```

3.Simple linear vs Polynomial regression

```
# file: salary_regression.py
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

df = pd.read_csv("Salary_positions.csv")  # adjust path
print(df.head())

X = df[['Level']].values  # shape (n,1)
y = df['Salary'].values

# split (optional - for small dataset you might fit on
all, but we keep a test set)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Simple linear regression
lin = LinearRegression()
lin.fit(X_train, y_train)
y_pred_lin = lin.predict(X_test)
rmse_lin = np.sqrt(mean_squared_error(y_test,
y_pred_lin))
r2_lin = r2_score(y_test, y_pred_lin)

# Polynomial regression (degree 2 or degree 3 — try both;
degree 2 is common for this dataset)
deg = 2
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)
y_pred_poly = poly_model.predict(X_test_poly)
rmse_poly = np.sqrt(mean_squared_error(y_test,
y_pred_poly))
r2_poly = r2_score(y_test, y_pred_poly)

print(f"Linear   RMSE: {rmse_lin:.2f}, R2: {r2_lin:.4f}")
print(f"Poly deg{deg} RMSE: {rmse_poly:.2f}, R2:
{r2_poly:.4f}")

# Predict salaries for level 11 and 12 using the better
model (choose lower RMSE or higher R2)
levels_to_predict = np.array([[11],[12]])
# using polynomial model
levels_poly = poly.transform(levels_to_predict)
preds_poly = poly_model.predict(levels_poly)
print("Polynomial predictions:")
for lvl, p in zip([11,12], preds_poly):
    print(f"Level {lvl} -> Pred Salary: {p:.2f}")

# using linear model
preds_lin = lin.predict(levels_to_predict)
print("Linear predictions:")
for lvl, p in zip([11,12], preds_lin):
    print(f"Level {lvl} -> Pred Salary: {p:.2f}")
```

Output:-



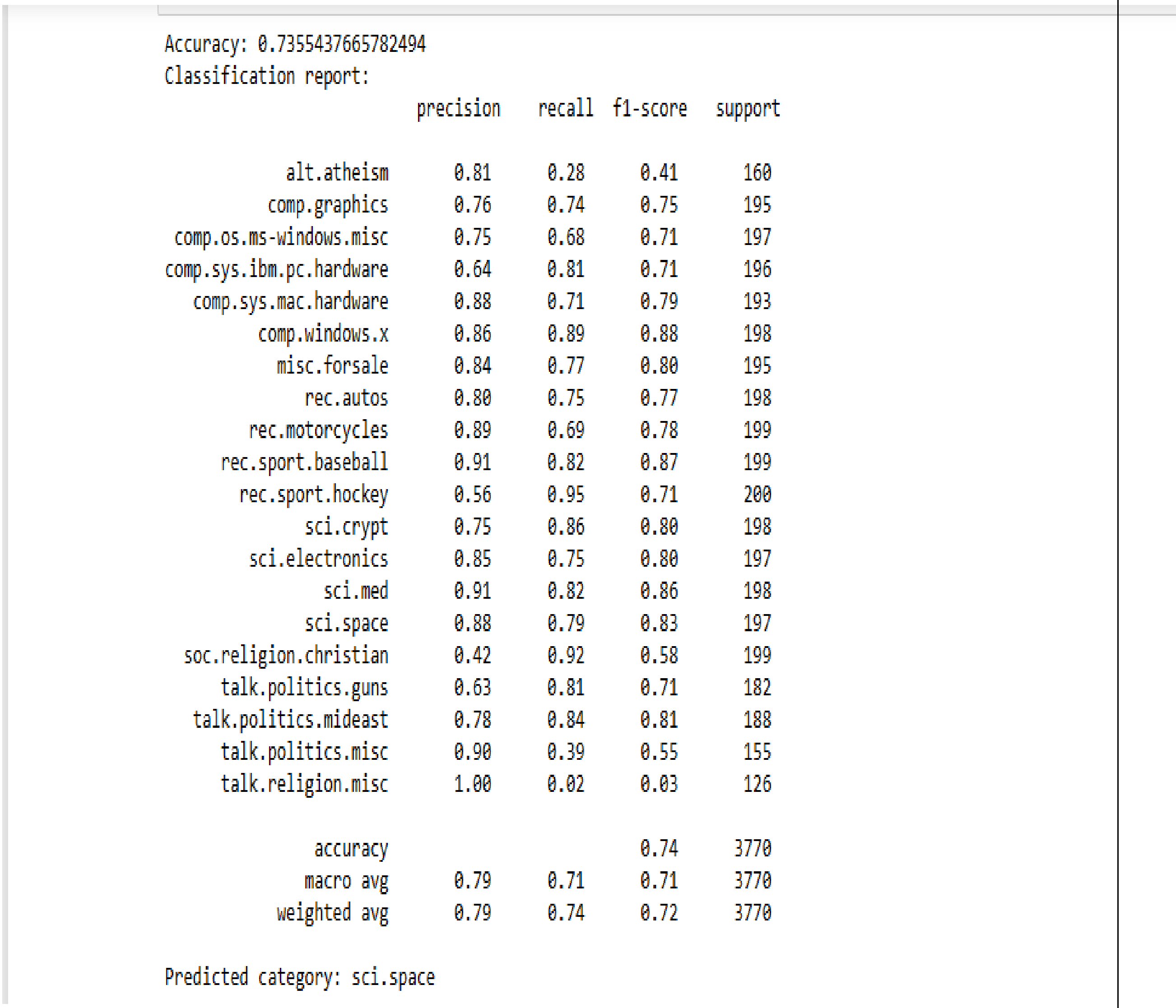
```
   Level  Salary
0      1   45000
1      2   50000
2      3   60000
3      4   80000
4      5  110000
Linear   RMSE: 88544.10, R2: 0.8451
Poly deg2 RMSE: 108689.37, R2: 0.7666
Polynomial predictions:
Level 11 -> Pred Salary: 1198055.75
Level 12 -> Pred Salary: 1531968.27
Linear predictions:
Level 11 -> Pred Salary: 726508.62
Level 12 -> Pred Salary: 814396.55
```

4.News classification into 20 newsgroups using Multinomial Naïve Bayes

```
# file: news_multinb.py
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# fetch full dataset (this may download the data the
first time)
categories = None  # None -> all 20 categories
data = fetch_20newsgroups(subset='all',
categories=categories,
remove=('headers','footers','quotes'))
X = data.data
y = data.target
target_names = data.target_names

# split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# pipeline: TF-IDF + MultinomialNB
pipe = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english',
max_df=0.7)),
    ('clf', MultinomialNB(alpha=1.0))])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification report:\n",
classification_report(y_test, y_pred,
target_names=target_names))

# Example: predict a single news text
sample_text = ["NASA announces new mission to the moon
next year."]
```

```
pred = pipe.predict(sample_text)[0]
print("Predicted category:", target_names[pred])
```

Output:-

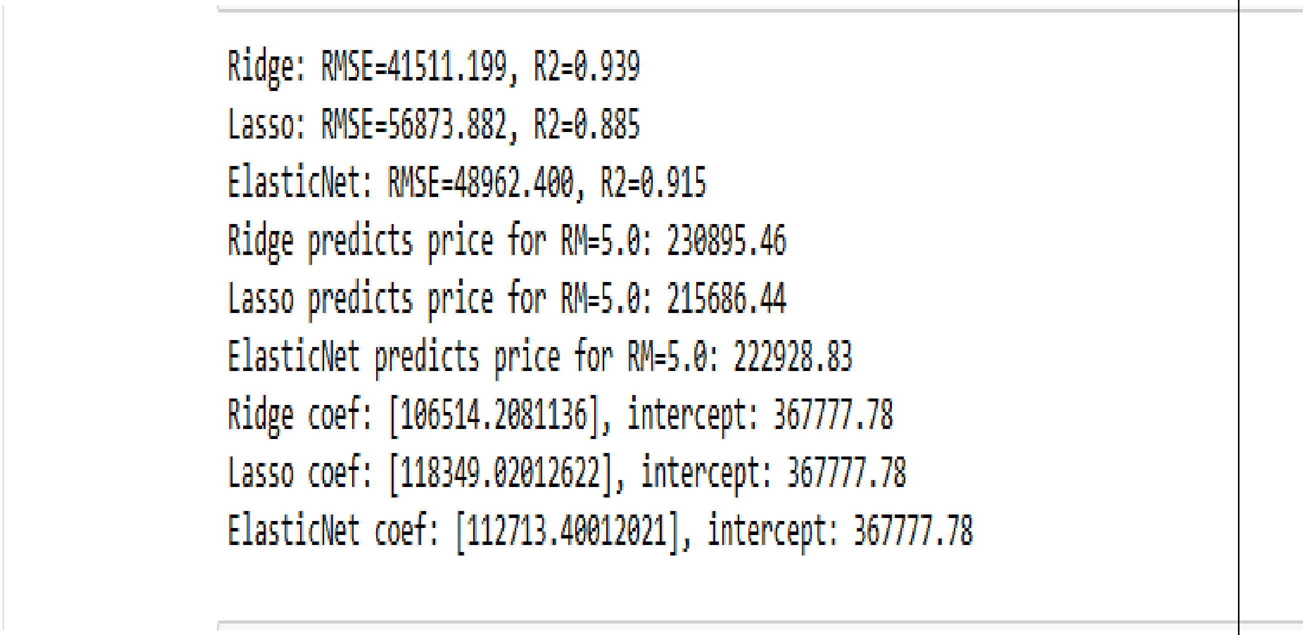


```
Accuracy: 0.7355437665782494
Classification report:
                          precision    recall  f1-score   support

             alt.atheism       0.81      0.28      0.41       160
           comp.graphics       0.76      0.74      0.75       195
 comp.os.ms-windows.misc       0.75      0.68      0.71       197
comp.sys.ibm.pc.hardware       0.64      0.81      0.71       196
   comp.sys.mac.hardware       0.88      0.71      0.79       193
          comp.windows.x       0.86      0.89      0.88       198
            misc.forsale       0.84      0.77      0.80       195
               rec.autos       0.80      0.75      0.77       198
         rec.motorcycles       0.89      0.69      0.78       199
      rec.sport.baseball       0.91      0.82      0.87       199
        rec.sport.hockey       0.56      0.95      0.71       200
               sci.crypt       0.75      0.86      0.80       198
         sci.electronics       0.85      0.75      0.80       197
                 sci.med       0.91      0.82      0.86       198
               sci.space       0.88      0.79      0.83       197
  soc.religion.christian       0.42      0.92      0.58       199
      talk.politics.guns       0.63      0.81      0.71       182
   talk.politics.mideast       0.78      0.84      0.81       188
      talk.politics.misc       0.90      0.39      0.55       155
      talk.religion.misc       1.00      0.02      0.03       126

                accuracy                           0.74      3770
               macro avg       0.79      0.71      0.71      3770
            weighted avg       0.79      0.74      0.72      3770

Predicted category: sci.space
```

5.Ridge, Lasso, ElasticNet on boston_houses.csv using 'RM' and 'Price'

```
# file: regularized_regression_boston.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("boston_houses.csv")  # adjust path
# expected columns 'RM' and 'Price'
if 'RM' not in df.columns or 'Price' not in df.columns:
    print("Columns found:", df.columns)
    raise ValueError("CSV must contain 'RM' and 'Price'
columns.")

df = df.dropna(subset=['RM','Price'])
X = df[['RM']].values  # shape (n,1)
y = df['Price'].values

# split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# scale features (regularization benefits from scaling)
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# initialize models (alpha can be tuned)
ridge = Ridge(alpha=1.0, random_state=42)
lasso = Lasso(alpha=0.1, random_state=42, max_iter=10000)
en = ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42,
max_iter=10000)

models = {'Ridge': ridge, 'Lasso': lasso, 'ElasticNet':
en}
results = {}

for name, m in models.items():
    m.fit(X_train_s, y_train)
    y_pred = m.predict(X_test_s)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    results[name] = {'model': m, 'rmse': rmse, 'r2': r2}
    print(f"{name}: RMSE={rmse:.3f}, R2={r2:.3f}")

# Predict price if house has 5 rooms (RM=5)
rm_value = 5.0
rm_scaled = scaler.transform(np.array([[rm_value]]))
for name, info in results.items():
    pred_price = info['model'].predict(rm_scaled)[0]
    print(f"{name} predicts price for RM={rm_value}:
{pred_price:.2f}")

# Optionally show coefficients
for name, info in results.items():
    coef = info['model'].coef_
    intercept = info['model'].intercept_
    print(f"{name} coef: {coef}, intercept:
{intercept:.2f}")
```

Output:-



```
Ridge: RMSE=41511.199, R2=0.939
Lasso: RMSE=56873.882, R2=0.885
ElasticNet: RMSE=48962.400, R2=0.915
Ridge predicts price for RM=5.0: 230895.46
Lasso predicts price for RM=5.0: 215686.44
ElasticNet predicts price for RM=5.0: 222928.83
Ridge coef: [106514.2081136], intercept: 367777.78
Lasso coef: [118349.02012622], intercept: 367777.78
ElasticNet coef: [112713.40012021], intercept: 367777.78
```

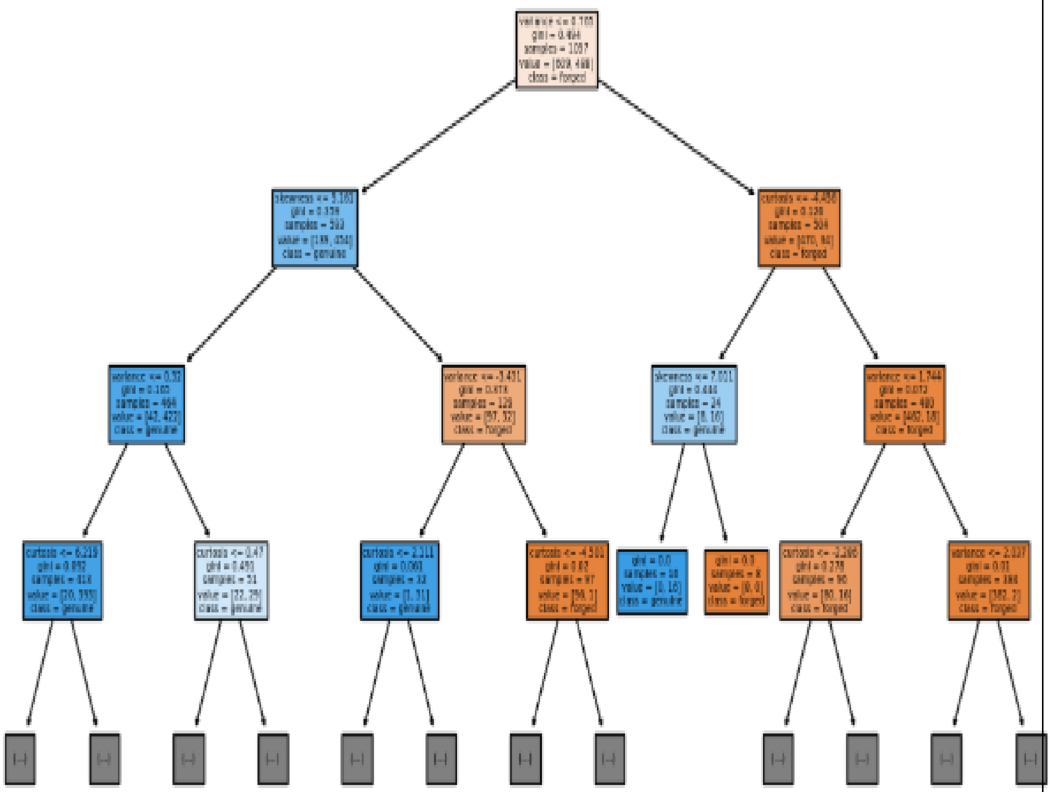
6. Decision Tree classifier on Banknote Authentication (UCI)

```
# file: dt_banknote.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,
plot_tree
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report
import matplotlib.pyplot as plt

# UCI banknote dataset URL (data only CSV)
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00267/data_banknote_authentication.txt"
# columns per UCI: variance, skewness, curtosis, entropy,
class
cols = ["variance", "skewness", "curtosis", "entropy",
"class"]
df = pd.read_csv(url, header=None, names=cols)

X = df[cols[:-1]]
y = df["class"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

clf = DecisionTreeClassifier(random_state=42,
max_depth=6)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))
print(classification_report(y_test, y_pred))

# optional: plot tree (small)
plt.figure(figsize=(12,6))
plot_tree(clf, feature_names=X.columns,
class_names=["forged","genuine"], filled=True,
max_depth=3)
plt.show()
# sample prediction
sample = X_test.iloc[0:3]
```

```
print("Samples:\n", sample)
print("Predictions:", clf.predict(sample))
```

Output:-

```
Accuracy: 0.9927272727272727
Confusion Matrix:
 [[151   2]
 [  0 122]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       153
           1       0.98      1.00      0.99       122

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```



```
Samples:
      variance  skewness  curtosis  entropy
385    2.3718    7.4908  0.015989  -1.7414
899   -1.4446    2.1438 -0.472410  -1.6677
47    -0.7869    9.5663 -3.786700  -7.5034
Predictions: [0 1 0]
```

7. Classify Iris dataset using SVM — compare kernels (linear, rbf, poly, sigmoid)

```
# file: svm_iris.py
import numpy as np
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

iris = datasets.load_iris()
X, y = iris.data, iris.target
names = iris.target_names

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

kernels = ['linear', 'rbf', 'poly', 'sigmoid']
results = {}
for k in kernels:
    clf = SVC(kernel=k, probability=False,
random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[k] = acc
    print(f"Kernel={k} Accuracy={acc:.4f}")
    print(classification_report(y_test, y_pred,
target_names=names))
    print("Confusion:\n", confusion_matrix(y_test,
y_pred))
    print("-"*40)

print("Summary accuracies:", results)

# Example: predict from input measurements
sample = np.array([[5.1, 3.5, 1.4, 0.2]])  # example
clf_lin = SVC(kernel='rbf',
probability=False).fit(X_train, y_train)
pred = clf_lin.predict(sample)
print("Predicted flower type:",
iris.target_names[pred[0]])
```

Output:-

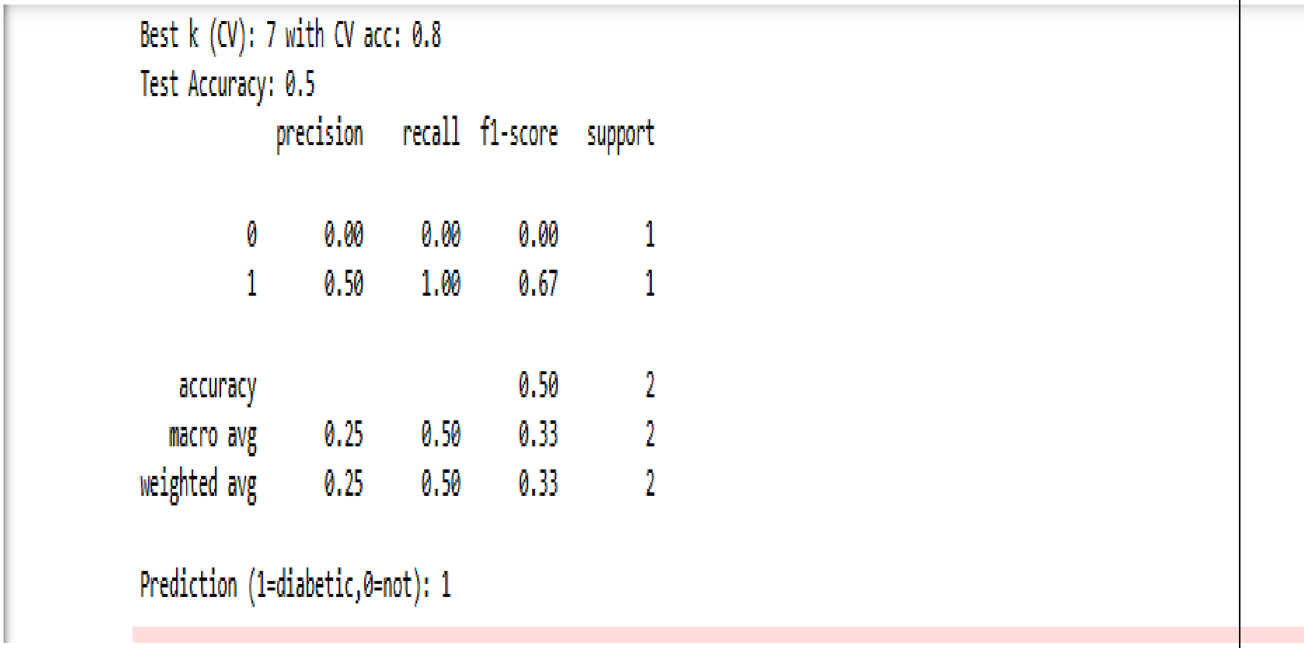
```
      Kernel=linear Accuracy=1.0000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
----------------------------------------
Kernel=rbf Accuracy=0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   virginica       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Confusion:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
----------------------------------------
Kernel=poly Accuracy=0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   virginica       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Confusion:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
----------------------------------------
Kernel=sigmoid Accuracy=0.1000
              precision    recall  f1-score   support

      setosa       0.15      0.30      0.20        10
  versicolor       0.00      0.00      0.00        10
   virginica       0.00      0.00      0.00        10

    accuracy                           0.10        30
   macro avg       0.05      0.10      0.07        30
weighted avg       0.05      0.10      0.07        30

Confusion:
 [[ 3  0  7]
 [ 7  0  3]
 [10  0  0]]
----------------------------------------
Summary accuracies: {'linear': 1.0, 'rbf': 0.9666666666666667, 'poly'
: 0.9666666666666667, 'sigmoid': 0.1}
Predicted flower type: setosa
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\metr
```

8. KNN model on Indian diabetes database (Pima) — find optimal K via cross-validation

```
# file: knn_pima.py
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

df = pd.read_csv("pima-indians-diabetes.csv")  # change
path if needed
# if headerless, provide column names accordingly

X = df.drop(columns=["Outcome"])
y = df["Outcome"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

# find best k with 5-fold CV
k_values = list(range(1, 21))
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_s, y_train,
cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())

best_k = k_values[int(np.argmax(cv_scores))]
print("Best k (CV):", best_k, "with CV acc:",
max(cv_scores))

# train with best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
```

```
knn_best.fit(X_train_s, y_train)
y_pred = knn_best.predict(X_test_s)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# predict for a new patient (example values)
new_patient = np.array([[2,120,70,20,79,25.6,0.5,30]])
new_patient_s = scaler.transform(new_patient)
print("Prediction (1=diabetic,0=not):",
knn_best.predict(new_patient_s)[0])
```

Output:-

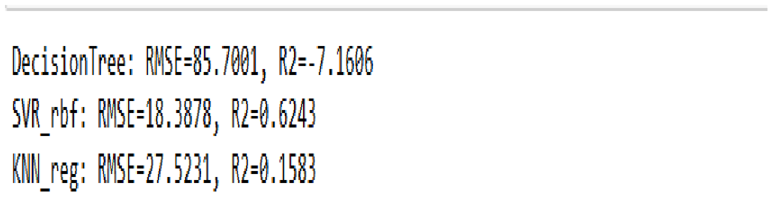


```
Best k (CV): 7 with CV acc: 0.8
Test Accuracy: 0.5
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Prediction (1=diabetic,0=not): 1
```

9. Non-linear regression (DecisionTree, SVM (SVR), KNN regressor) on Petrol Consumption dataset

```
# file: nonlin_petrol.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("petrol_consumption.csv")  # change path
# choose features and target - adjust to dataset columns
(example below assumptions)
# Replace these with actual column names in your CSV
# Example assumed columns:
'Petrol_tax','Average_income','Paved_Highways','Populatio
n_Driver_licence','Petrol_Consumption'
features = [c for c in df.columns if c !=
'Petrol_Consumption']
target = 'Petrol_Consumption'

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

models = {
    'DecisionTree':
DecisionTreeRegressor(random_state=42, max_depth=6),
    'SVR_rbf': SVR(kernel='rbf', C=100, gamma='scale',
epsilon=0.1),
    'KNN_reg': KNeighborsRegressor(n_neighbors=5)
}

for name, m in models.items():
    m.fit(X_train_s, y_train)
    y_pred = m.predict(X_test_s)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    print(f"{name}: RMSE={rmse:.4f}, R2={r2:.4f}")
```

Output:-



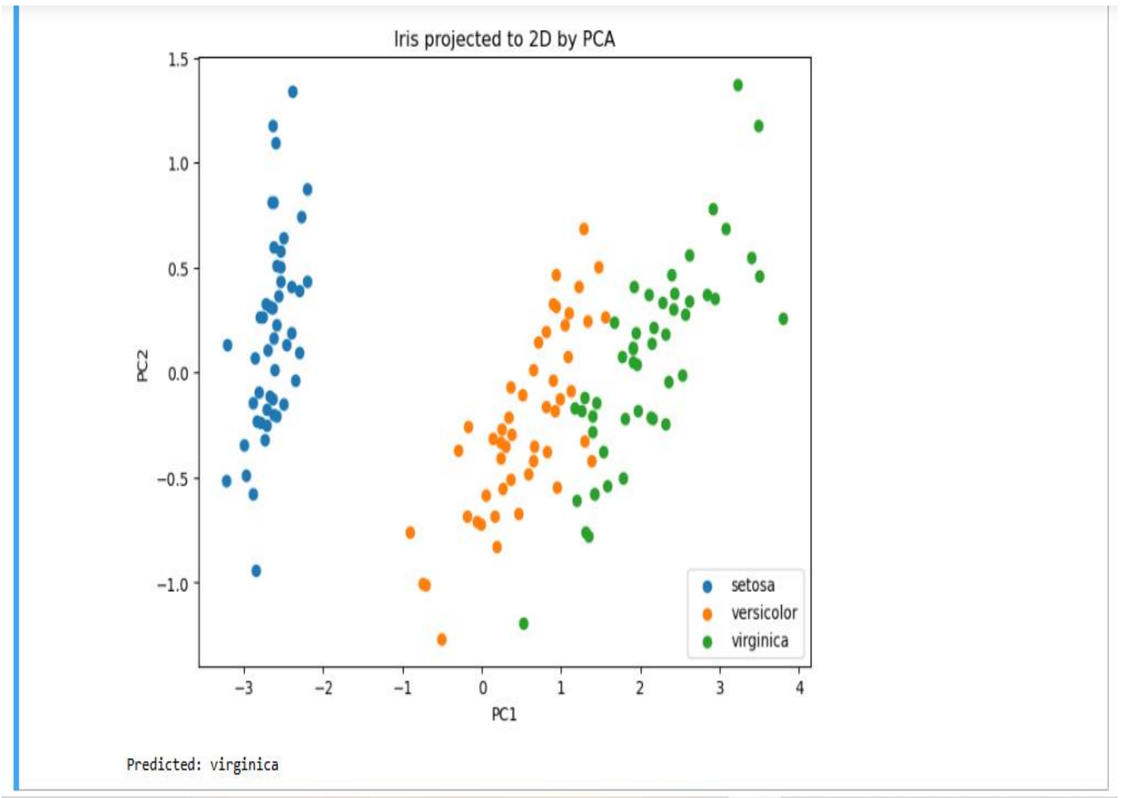
```
DecisionTree: RMSE=85.7001, R2=-7.1606
SVR_rbf: RMSE=18.3878, R2=0.6243
KNN_reg: RMSE=27.5231, R2=0.1583
```

10. PCA reduce Iris from 4D to 2D then train & predict

```
# file: pca_iris.py
import numpy as np
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X, y = iris.data, iris.target
pca = PCA(n_components=2, random_state=42)
X2 = pca.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X2,
y, test_size=0.2, random_state=42, stratify=y)
clf = LogisticRegression().fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy (2D PCA):", accuracy_score(y_test,
y_pred))
print(classification_report(y_test, y_pred,
target_names=iris.target_names))

# plot
plt.figure(figsize=(8,6))
for lab, col in zip(np.unique(y), ['r','g','b']):
    plt.scatter(X2[y==lab,0], X2[y==lab,1],
label=iris.target_names[lab])
plt.legend()
plt.title("Iris projected to 2D by PCA")
plt.xlabel("PC1"); plt.ylabel("PC2")
plt.show()

# predict new sample (first transform then predict)
sample4d = np.array([[5.9,3.0,5.1,1.8]])
sample2d = pca.transform(sample4d)
print("Predicted:",
iris.target_names[clf.predict(sample2d)[0]])
```

Output:-

```
Accuracy (2D PCA): 0.9333333333333333
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

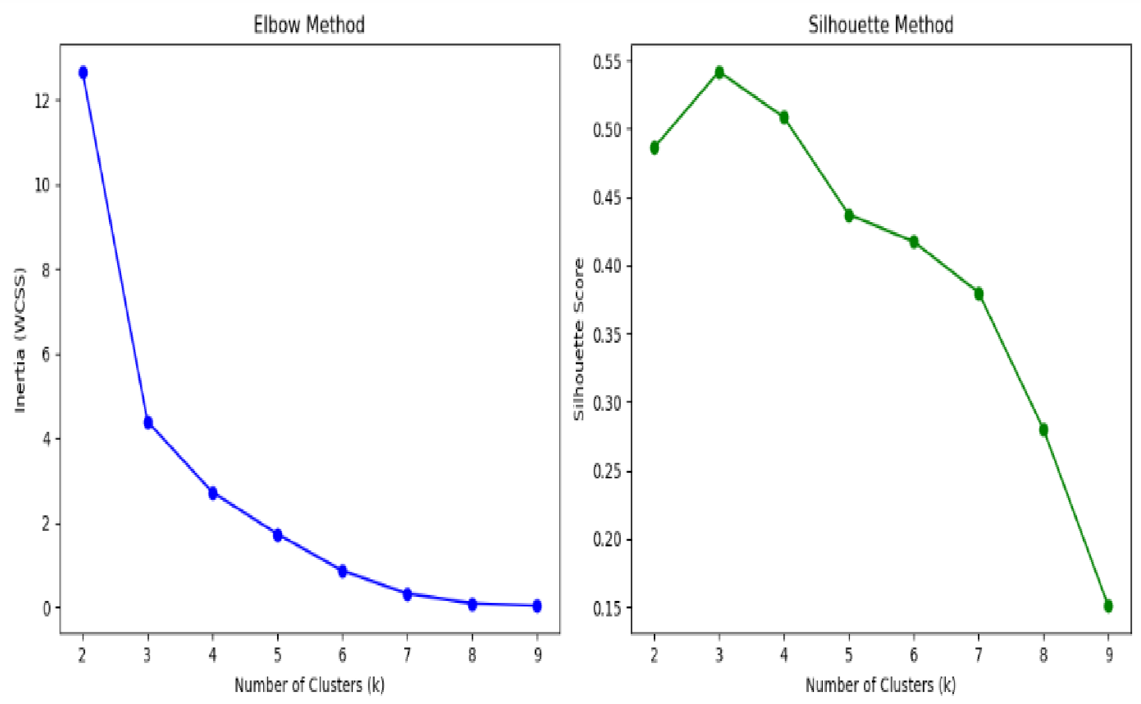


```
Predicted: virginica
```

11.K-means clustering for employee income groups — use elbow and silhouette

```
# file: kmeans_income.py
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

df = pd.read_csv("employees.csv")  # change path
# example features
if {'AnnualIncome','SpendingScore'}.issubset(df.columns):
    X = df[['AnnualIncome','SpendingScore']].values
else:
    # fallback: use all numeric columns
    X = df.select_dtypes(include=[np.number]).values

scaler = StandardScaler().fit(X)
X_s = scaler.transform(X)

# elbow method
inertia = []
K = range(2,11)
for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(X_s)
    inertia.append(km.inertia_)

plt.plot(K, inertia, 'o-')
plt.xlabel('k'); plt.ylabel('Inertia (WCSS)');
plt.title('Elbow method')
plt.show()

# silhouette
sil_scores = []
for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = km.fit_predict(X_s)
    sil_scores.append(silhouette_score(X_s, labels))

plt.plot(K, sil_scores, 'o-')
plt.xlabel('k'); plt.ylabel('Silhouette Score');
plt.title('Silhouette method')
plt.show()

best_k = K[int(np.argmax(sil_scores))]
print("Best k by silhouette:", best_k)

# final clustering
km_final = KMeans(n_clusters=best_k, random_state=42,
n_init=10)
labels = km_final.fit_predict(X_s)
df['cluster'] = labels
print(df.groupby('cluster').mean())
```

Output:-

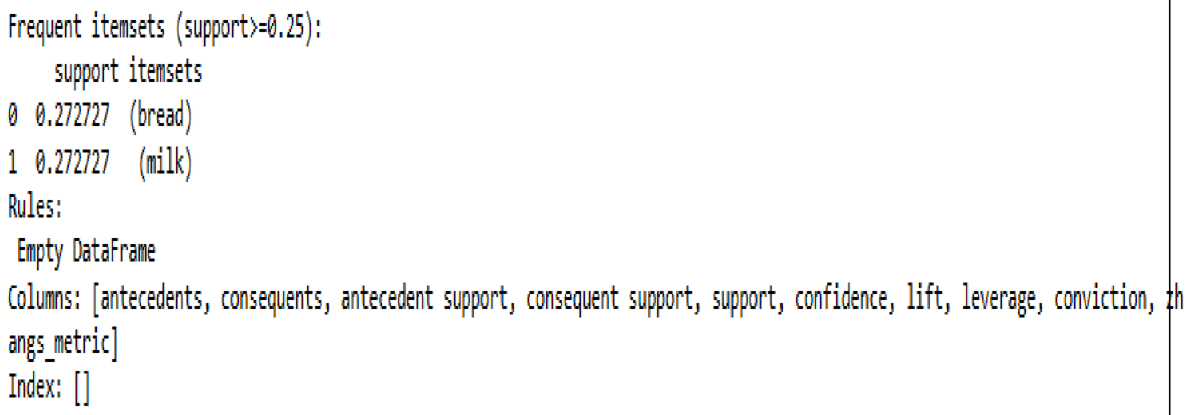


```
✅ Best number of clusters (by silhouette): 3

Cluster Averages:
               Age  Education  Experience        Income
Cluster
0        25.333333  12.666667    2.333333  25666.666667
1        36.500000  16.750000   11.250000  58250.000000
2        47.666667  16.333333   22.333333  90000.000000
```

12. Apriori on groceries dataset (min support = 0.25)

```
# file: apriori_groceries.py
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori,
association_rules

# assume groceries.csv where each row is comma-separated
items
trans = []
with open("groceries.csv", 'r') as f:
    for line in f:
        items = [it.strip() for it in
line.strip().split(',') if it.strip()]
        trans.append(items)

te = TransactionEncoder()
te_ary = te.fit(trans).transform(trans)
df = pd.DataFrame(te_ary, columns=te.columns_)

# apriori with min_support 0.25
freq = apriori(df, min_support=0.25, use_colnames=True)
print("Frequent itemsets (support>=0.25):\n",
freq.sort_values(by='support', ascending=False))

# association rules
rules = association_rules(freq, metric="lift",
min_threshold=1.0)
print("Rules:\n", rules.sort_values(by='confidence',
ascending=False).head(10))
```

Output:-



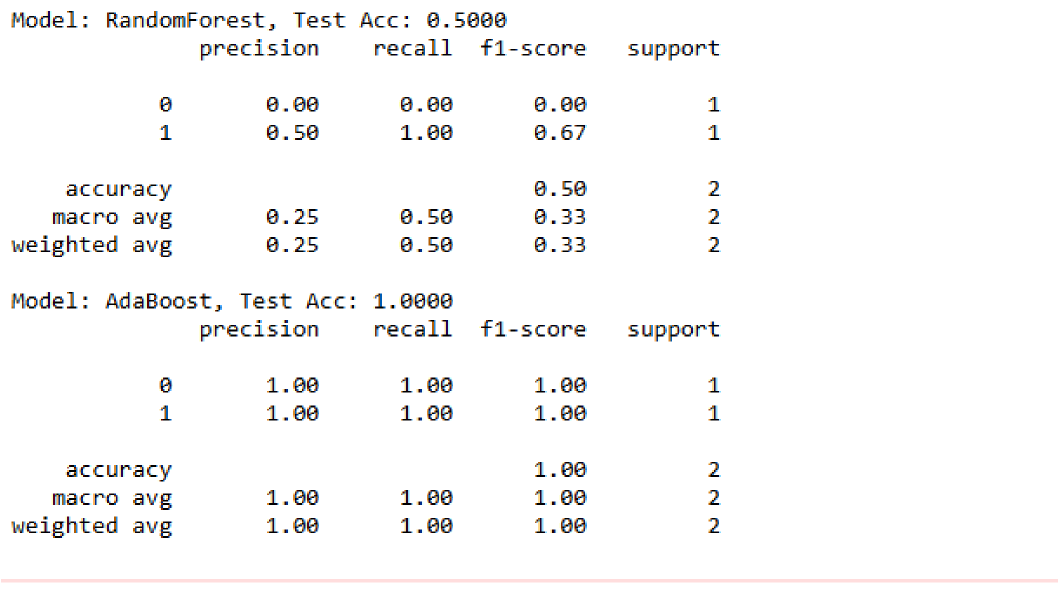
```
Frequent itemsets (support>=0.25):
    support itemsets
0  0.272727  (bread)
1  0.272727   (milk)
Rules:
 Empty DataFrame
Columns: [antecedents, consequents, antecedent support, consequent support, support, confidence, lift, leverage, conviction, zh
angs_metric]
Index: []
```

13. Ensemble ML on Pima Indians Diabetes Database — Random Forest (bagging), AdaBoost (boosting), Voting, Stacking

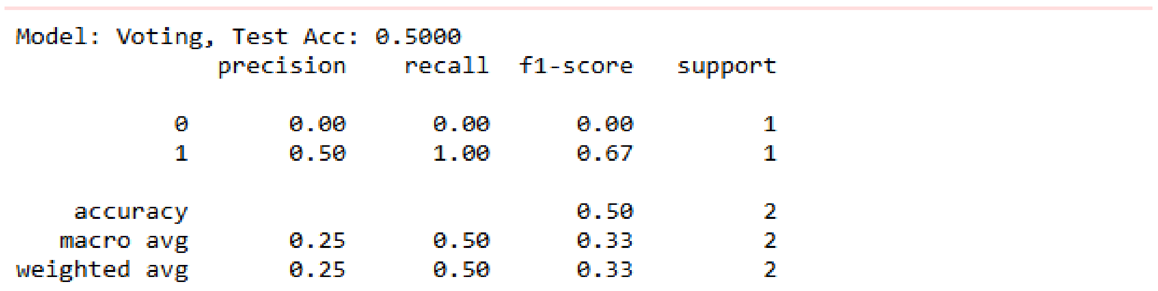
```
# file: ensembles_pima.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, VotingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
classification_report

df = pd.read_csv("pima-indians-diabetes.csv")
X = df.drop(columns=['Outcome'])
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

rf = RandomForestClassifier(n_estimators=100,
random_state=42)
ada = AdaBoostClassifier(n_estimators=100,
random_state=42)
svm = SVC(probability=True, kernel='rbf',
random_state=42)
knn = KNeighborsClassifier()
lr = LogisticRegression(max_iter=500)

# Voting
voting =
VotingClassifier(estimators=[('rf',rf),('svm',svm),('knn'
,knn)], voting='soft')
# Stacking
stack =
StackingClassifier(estimators=[('rf',rf),('svm',svm)],
final_estimator=LogisticRegression(), cv=5)
```

```
models = {'RandomForest':rf, 'AdaBoost':ada,
'Voting':voting, 'Stacking':stack}
for name, m in models.items():
    m.fit(X_train_s, y_train)
    preds = m.predict(X_test_s)
    print(f"Model: {name}, Test Acc:
{accuracy_score(y_test, preds):.4f}")
    print(classification_report(y_test, preds))
```

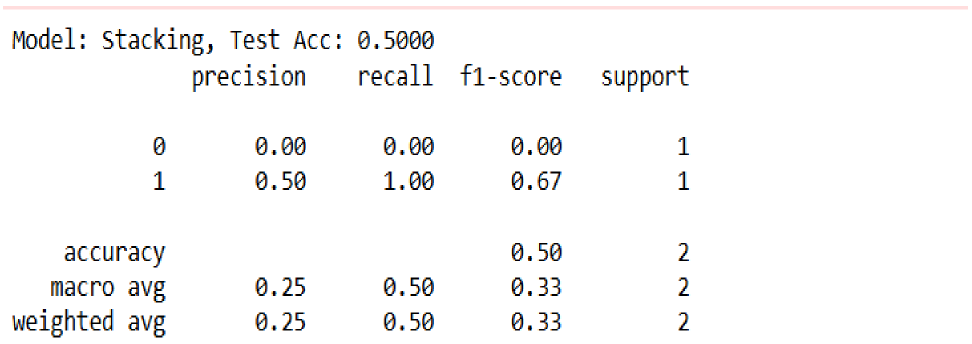
Output:-



```
Model: RandomForest, Test Acc: 0.5000
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Model: AdaBoost, Test Acc: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```



```
Model: Voting, Test Acc: 0.5000
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```



```
Model: Stacking, Test Acc: 0.5000
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

14. Create a two-layer neural network with ReLU and Sigmoid activation (simple binary classifier with Keras)

```
# file: two_layer_nn.py
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# synthetic binary classification dataset
X, y = make_classification(n_samples=2000, n_features=20,
n_informative=8, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

model = Sequential([
    Dense(64, activation='relu',
input_shape=(X_train_s.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(learning_rate=1e-3),
loss=BinaryCrossentropy(), metrics=['accuracy'])
model.fit(X_train_s, y_train, epochs=20, batch_size=32,
validation_split=0.1, verbose=1)

preds = model.predict(X_test_s).ravel()
pred_labels = (preds > 0.5).astype(int)
print("Test Accuracy:", accuracy_score(y_test,
pred_labels))
```

Output:-

```
Epoch 1/20
45/45 [==============================] - 1s 9ms/step - loss: 0.66
60 - accuracy: 0.6042 - val_loss: 0.6052 - val_accuracy: 0.7375
Epoch 2/20
```

```
45/45 [==============================] - 0s 3ms/step - loss: 0.57
38 - accuracy: 0.7201 - val_loss: 0.5328 - val_accuracy: 0.7375
Epoch 3/20
45/45 [==============================] - 0s 3ms/step - loss: 0.51
58 - accuracy: 0.7403 - val_loss: 0.4639 - val_accuracy: 0.7812
Epoch 4/20
45/45 [==============================] - 0s 3ms/step - loss: 0.47
19 - accuracy: 0.7729 - val_loss: 0.4321 - val_accuracy: 0.7750
Epoch 5/20
45/45 [==============================] - 0s 3ms/step - loss: 0.43
46 - accuracy: 0.7986 - val_loss: 0.4144 - val_accuracy: 0.7812
Epoch 6/20
45/45 [==============================] - 0s 3ms/step - loss: 0.40
10 - accuracy: 0.8181 - val_loss: 0.3932 - val_accuracy: 0.7937
Epoch 7/20
45/45 [==============================] - 0s 3ms/step - loss: 0.37
13 - accuracy: 0.8438 - val_loss: 0.3756 - val_accuracy: 0.8062
Epoch 8/20
45/45 [==============================] - 0s 3ms/step - loss: 0.34
49 - accuracy: 0.8493 - val_loss: 0.3671 - val_accuracy: 0.8500
Epoch 9/20
45/45 [==============================] - 0s 3ms/step - loss: 0.32
64 - accuracy: 0.8521 - val_loss: 0.3624 - val_accuracy: 0.8250
Epoch 10/20
45/45 [==============================] - 0s 3ms/step - loss: 0.30
28 - accuracy: 0.8806 - val_loss: 0.3530 - val_accuracy: 0.8375
Epoch 11/20
45/45 [==============================] - 0s 3ms/step - loss: 0.28
37 - accuracy: 0.8903 - val_loss: 0.3427 - val_accuracy: 0.8438
Epoch 12/20
45/45 [==============================] - 0s 3ms/step - loss: 0.26
84 - accuracy: 0.8944 - val_loss: 0.3382 - val_accuracy: 0.8438
Epoch 13/20
45/45 [==============================] - 0s 3ms/step - loss: 0.25
39 - accuracy: 0.9076 - val_loss: 0.3356 - val_accuracy: 0.8313
Epoch 14/20
45/45 [==============================] - 0s 3ms/step - loss: 0.23
85 - accuracy: 0.9083 - val_loss: 0.3403 - val_accuracy: 0.8375
Epoch 15/20
45/45 [==============================] - 0s 3ms/step - loss: 0.22
54 - accuracy: 0.9236 - val_loss: 0.3164 - val_accuracy: 0.8750
Epoch 16/20
45/45 [==============================] - 0s 3ms/step - loss: 0.21
38 - accuracy: 0.9215 - val_loss: 0.3185 - val_accuracy: 0.8562
Epoch 17/20
45/45 [==============================] - 0s 3ms/step - loss: 0.20
10 - accuracy: 0.9333 - val_loss: 0.3148 - val_accuracy: 0.8750
Epoch 18/20
45/45 [==============================] - 0s 3ms/step - loss: 0.19
03 - accuracy: 0.9333 - val_loss: 0.3248 - val_accuracy: 0.8625
Epoch 19/20
45/45 [==============================] - 0s 3ms/step - loss: 0.18
09 - accuracy: 0.9410 - val_loss: 0.3257 - val_accuracy: 0.8438
Epoch 20/20
45/45 [==============================] - 0s 3ms/step - loss: 0.17
13 - accuracy: 0.9458 - val_loss: 0.3120 - val_accuracy: 0.8687
13/13 [==============================] - 0s 2ms/step
Test Accuracy: 0.8325
```

15. ANN to classify house price above/below average

```
# file: ann_house_classify.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score,
classification_report

df = pd.read_csv("house_prices.csv")
# prepare features and binary target
target_col = 'Price'
X = df.drop(columns=[target_col])
y = (df[target_col] > df[target_col].mean()).astype(int)

# one-hot categorical and fillna
X = pd.get_dummies(X, drop_first=True).fillna(0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)

model = Sequential([
    Dense(64, activation='relu',
input_shape=(X_train_s.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
model.fit(X_train_s, y_train, epochs=50, batch_size=8,
validation_split=0.1, verbose=0)

preds = (model.predict(X_test_s).ravel() >
0.5).astype(int)
print("Test Accuracy:", accuracy_score(y_test, preds))
print(classification_report(y_test, preds))
```

Output:-

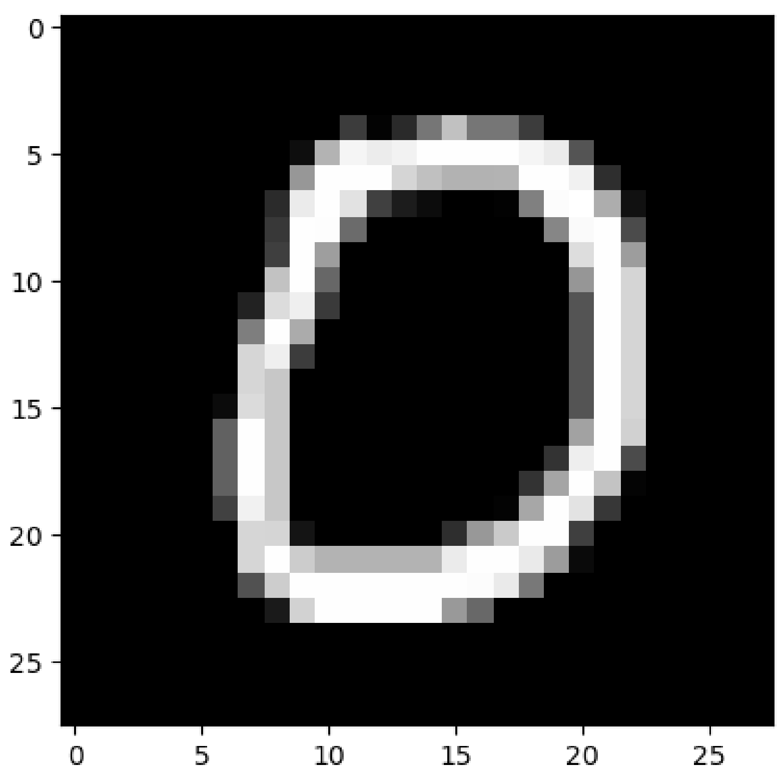
```
1/1 [==============================] - 0s 58ms/step
Test Accuracy: 0.5
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

16. CNN on MNIST (Keras) — train and predict a digit from a sample image

```
# file: cnn_mnist.py
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score

# load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((-1,28,28,1)).astype('float32')
/ 255.0
x_test = x_test.reshape((-1,28,28,1)).astype('float32') /
255.0
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

model = Sequential([
    Conv2D(32, (3,3), activation='relu',
input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.25),
    Dense(10, activation='softmax')
])
model.compile(optimizer=Adam(),
loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train_cat, epochs=5, batch_size=128,
validation_split=0.1)

# evaluate
loss, acc = model.evaluate(x_test, y_test_cat)
print("Test accuracy:", acc)

# predict single image example
import matplotlib.pyplot as plt
idx = 10
plt.imshow(x_test[idx].reshape(28,28), cmap='gray')
plt.show()
pred = np.argmax(model.predict(x_test[idx:idx+1]))
print("Predicted digit:", pred, "True:", y_test[idx])
```

Output:-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 [==============================] - 3s 0us/step
Epoch 1/5
422/422 [==============================] - 16s 37ms/step - loss: 0.
2491 - accuracy: 0.9234 - val_loss: 0.0551 - val_accuracy: 0.9842
Epoch 2/5
422/422 [==============================] - 16s 37ms/step - loss: 0.
0684 - accuracy: 0.9795 - val_loss: 0.0472 - val_accuracy: 0.9850
Epoch 3/5
422/422 [==============================] - 19s 45ms/step - loss: 0.
0502 - accuracy: 0.9844 - val_loss: 0.0359 - val_accuracy: 0.9900
Epoch 4/5
422/422 [==============================] - 16s 38ms/step - loss: 0.
0379 - accuracy: 0.9885 - val_loss: 0.0359 - val_accuracy: 0.9907
Epoch 5/5
422/422 [==============================] - 16s 38ms/step - loss: 0.
0325 - accuracy: 0.9900 - val_loss: 0.0316 - val_accuracy: 0.9918
313/313 [==============================] - 1s 5ms/step - loss: 0.02
88 - accuracy: 0.9904
Test accuracy: 0.9904000163078308
```



```
                                                1/1 [============
==================] - 0s 84ms/step
Predicted digit: 0 True: 0
```

17.RNN (LSTM) to analyze Google stock price trend — predict up/down for next day

```
# file: lstm_google.py
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report

df = pd.read_csv("GOOG.csv", parse_dates=['Date'])  #
ensure it has 'Date' and 'Close'
df = df.sort_values('Date').reset_index(drop=True)
close = df['Close'].values.reshape(-1,1)

# create up/down target: 1 if next day's close > today's
close, else 0
```

```
target = (np.roll(close, -1) > close).astype(int)[:-
1].ravel()
series = close[:-1]

# scale prices
scaler = MinMaxScaler()
series_s = scaler.fit_transform(series)

# create sequences (e.g., last 10 days)
SEQ_LEN = 10
X, y = [], []
for i in range(len(series_s)-SEQ_LEN+1):
    X.append(series_s[i:i+SEQ_LEN].flatten())
    y.append(target[i+SEQ_LEN-1])
X = np.array(X); y = np.array(y)

# reshape for LSTM: samples, timesteps, features (we used
flattened so reshape accordingly)
X = X.reshape((X.shape[0], SEQ_LEN, 1))

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

model = Sequential([
    LSTM(50, input_shape=(SEQ_LEN,1),
return_sequences=False),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_split=0.1, verbose=1)

preds = (model.predict(X_test).ravel() > 0.5).astype(int)
print("Accuracy:", accuracy_score(y_test, preds))
print(classification_report(y_test, preds))

# Predict direction for most recent SEQ_LEN days
recent_seq = series_s[-SEQ_LEN:].reshape((1, SEQ_LEN, 1))
pred_dir = (model.predict(recent_seq).ravel() >
0.5).astype(int)[0]
print("Predicted direction tomorrow: ", "UP (1)" if
pred_dir==1 else "DOWN (0)")
```

Output:-

```
Epoch 1/20
1/1 [==============================] - 4s 4s/step - loss: 0.6995
- accuracy: 0.3333 - val_loss: 0.6983 - val_accuracy: 0.0000e+00
Epoch 2/20
1/1 [==============================] - 0s 51ms/step - loss: 0.703
1 - accuracy: 0.0000e+00 - val_loss: 0.6860 - val_accuracy: 1.000
0
Epoch 3/20
1/1 [==============================] - 0s 45ms/step - loss: 0.684
1 - accuracy: 0.6667 - val_loss: 0.6738 - val_accuracy: 1.0000
Epoch 4/20
1/1 [==============================] - 0s 44ms/step - loss: 0.677
2 - accuracy: 1.0000 - val_loss: 0.6614 - val_accuracy: 1.0000
Epoch 5/20
1/1 [==============================] - 0s 45ms/step - loss: 0.670
0 - accuracy: 1.0000 - val_loss: 0.6490 - val_accuracy: 1.0000
Epoch 6/20
1/1 [==============================] - 0s 47ms/step - loss: 0.668
5 - accuracy: 1.0000 - val_loss: 0.6363 - val_accuracy: 1.0000
Epoch 7/20
1/1 [==============================] - 0s 50ms/step - loss: 0.643
8 - accuracy: 1.0000 - val_loss: 0.6234 - val_accuracy: 1.0000
Epoch 8/20
1/1 [==============================] - 0s 51ms/step - loss: 0.633
1 - accuracy: 1.0000 - val_loss: 0.6100 - val_accuracy: 1.0000
Epoch 9/20
1/1 [==============================] - 0s 39ms/step - loss: 0.617
4 - accuracy: 1.0000 - val_loss: 0.5962 - val_accuracy: 1.0000
Epoch 10/20
1/1 [==============================] - 0s 89ms/step - loss: 0.599
4 - accuracy: 1.0000 - val_loss: 0.5817 - val_accuracy: 1.0000
Epoch 11/20
1/1 [==============================] - 0s 42ms/step - loss: 0.596
8 - accuracy: 1.0000 - val_loss: 0.5666 - val_accuracy: 1.0000
Epoch 12/20
1/1 [==============================] - 0s 43ms/step - loss: 0.587
3 - accuracy: 1.0000 - val_loss: 0.5508 - val_accuracy: 1.0000
Epoch 13/20
1/1 [==============================] - 0s 48ms/step - loss: 0.567
0 - accuracy: 1.0000 - val_loss: 0.5342 - val_accuracy: 1.0000
Epoch 14/20
1/1 [==============================] - 0s 47ms/step - loss: 0.555
6 - accuracy: 1.0000 - val_loss: 0.5166 - val_accuracy: 1.0000
Epoch 15/20
1/1 [==============================] - 0s 47ms/step - loss: 0.565
9 - accuracy: 1.0000 - val_loss: 0.4981 - val_accuracy: 1.0000
Epoch 16/20
1/1 [==============================] - 0s 46ms/step - loss: 0.525
5 - accuracy: 1.0000 - val_loss: 0.4784 - val_accuracy: 1.0000
Epoch 17/20
1/1 [==============================] - 0s 44ms/step - loss: 0.503
6 - accuracy: 1.0000 - val_loss: 0.4575 - val_accuracy: 1.0000
Epoch 18/20
1/1 [==============================] - 0s 39ms/step - loss: 0.507
0 - accuracy: 1.0000 - val_loss: 0.4353 - val_accuracy: 1.0000
```

```
Epoch 19/20
1/1 [==============================] - 0s 46ms/step - loss: 0.477
5 - accuracy: 1.0000 - val_loss: 0.4117 - val_accuracy: 1.0000
Epoch 20/20
1/1 [==============================] - 0s 44ms/step - loss: 0.448
8 - accuracy: 1.0000 - val_loss: 0.3864 - val_accuracy: 1.0000
1/1 [==============================] - 1s 604ms/step
Accuracy: 1.0
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

1/1 [==============================] - 0s 24ms/step
Predicted direction tomorrow: UP (1)
```