**Machine Learning Projects**

**Weather Prediction**

import pandas as pd

pip install xgboost

df = pd.read\_csv("seattle-weather.csv")

df.head()

|  | **date** | **precipitation** | **temp\_max** | **temp\_min** | **wind** | **weather** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1/1/2012 | 0.0 | 12.8 | 5.0 | 4.7 | drizzle |
| 1 | 1/2/2012 | 10.9 | 10.6 | 2.8 | 4.5 | rain |
| 2 | 1/3/2012 | 0.8 | 11.7 | 7.2 | 2.3 | rain |
| 3 | 1/4/2012 | 20.3 | 12.2 | 5.6 | 4.7 | rain |
| 4 | 1/5/2012 | 1.3 | 8.9 | 2.8 | 6.1 | rain |

df.isnull().sum()

date 0

precipitation 0

temp\_max 0

temp\_min 0

wind 0

weather 0

dtype: int64

def LabelEncoding(c):

    from sklearn import preprocessing

    le = preprocessing.LabelEncoder()

    df[c]= le.fit\_transform(df[c])

    df[c].unique()

LabelEncoding("weather")

Df

| **date** | **precipitation** | **temp\_max** | **temp\_min** | **wind** | **weather** |
| --- | --- | --- | --- | --- | --- |
| 0 | 1/1/2012 | 0.0 | 12.8 | 5.0 | 4.7 | 0 |
| 1 | 1/2/2012 | 10.9 | 10.6 | 2.8 | 4.5 | 2 |
| 2 | 1/3/2012 | 0.8 | 11.7 | 7.2 | 2.3 | 2 |
| 3 | 1/4/2012 | 20.3 | 12.2 | 5.6 | 4.7 | 2 |
| 4 | 1/5/2012 | 1.3 | 8.9 | 2.8 | 6.1 | 2 |
| ... | ... | ... | ... | ... | ... | ... |
| 1456 | 12/27/2015 | 8.6 | 4.4 | 1.7 | 2.9 | 2 |
| 1457 | 12/28/2015 | 1.5 | 5.0 | 1.7 | 1.3 | 2 |
| 1458 | 12/29/2015 | 0.0 | 7.2 | 0.6 | 2.6 | 1 |
| 1459 | 12/30/2015 | 0.0 | 5.6 | -1.0 | 3.4 | 4 |
| 1460 | 12/31/2015 | 0.0 | 5.6 | -2.1 | 3.5 | 4 |

1461 rows × 6 columns

cols = ['precipitation' , 'temp\_max', 'temp\_min', 'wind']

def normalize(df,cols):

    for x in cols:

        df[x] = df[x]/df[x].max()

normalize(df,cols)

df

| **date** | **precipitation** | **temp\_max** | **temp\_min** | **wind** | **weather** |
| --- | --- | --- | --- | --- | --- |
| 0 | 1/1/2012 | 0.000000 | 0.359551 | 0.273224 | 0.494737 | 0 |
| 1 | 1/2/2012 | 0.194991 | 0.297753 | 0.153005 | 0.473684 | 2 |
| 2 | 1/3/2012 | 0.014311 | 0.328652 | 0.393443 | 0.242105 | 2 |
| 3 | 1/4/2012 | 0.363148 | 0.342697 | 0.306011 | 0.494737 | 2 |
| 4 | 1/5/2012 | 0.023256 | 0.250000 | 0.153005 | 0.642105 | 2 |
| ... | ... | ... | ... | ... | ... | ... |
| 1456 | 12/27/2015 | 0.153846 | 0.123596 | 0.092896 | 0.305263 | 2 |
| 1457 | 12/28/2015 | 0.026834 | 0.140449 | 0.092896 | 0.136842 | 2 |
| 1458 | 12/29/2015 | 0.000000 | 0.202247 | 0.032787 | 0.273684 | 1 |
| 1459 | 12/30/2015 | 0.000000 | 0.157303 | -0.054645 | 0.357895 | 4 |
| 1460 | 12/31/2015 | 0.000000 | 0.157303 | -0.114754 | 0.368421 | 4 |

1461 rows × 6 columns

x = df.drop('weather',axis=1)

y = df['weather']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 0)

from xgboost import XGBClassifier

xg = XGBClassifier()

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

xg.get\_params()

{'objective': 'multi:softprob',

'base\_score': None,

'booster': None,

'callbacks': None,

'colsample\_bylevel': None,

'colsample\_bynode': None,

'colsample\_bytree': None,

'device': None,

'early\_stopping\_rounds': None,

'enable\_categorical': False,

'eval\_metric': None,

'feature\_types': None,

'feature\_weights': None,

'gamma': None,

'grow\_policy': None,

'importance\_type': None,

'interaction\_constraints': None,

'learning\_rate': None,

'max\_bin': None,

'max\_cat\_threshold': None,

'max\_cat\_to\_onehot': None,

'max\_delta\_step': None,

'max\_depth': None,

'max\_leaves': None,

'min\_child\_weight': None,

...

'scale\_pos\_weight': None,

'subsample': None,

'tree\_method': None,

'validate\_parameters': None,

'verbosity': None}

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

y\_hat = xg.predict(X\_test)

print(accuracy\_score(y\_test,y\_hat))

print(classification\_report(y\_test,y\_hat))

0.757679180887372

precision recall f1-score support

0 0.00 0.00 0.00 10

1 0.15 0.07 0.10 29

2 0.95 0.91 0.93 123

3 1.00 0.33 0.50 6

4 0.70 0.85 0.77 125

accuracy 0.76 293

macro avg 0.56 0.43 0.46 293

weighted avg 0.73 0.76 0.74 293

grid = {'learning\_rate': [0.1,1, 0.01, 0.001], 'gamma':[0,1,10,100]}

from sklearn.model\_selection import GridSearchCV

model = GridSearchCV(XGBClassifier(), grid, cv=10, verbose=2)

from xgboost import XGBClassifier

# create the model

model = XGBClassifier()

# fit the model

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

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

grid\_predictions = model.predict(X\_test)

print(accuracy\_score(y\_test,grid\_predictions))

print(classification\_report(y\_test,grid\_predictions))

0.757679180887372

precision recall f1-score support

0 0.00 0.00 0.00 10

1 0.15 0.07 0.10 29

2 0.95 0.91 0.93 123

3 1.00 0.33 0.50 6

4 0.70 0.85 0.77 125

accuracy 0.76 293

macro avg 0.56 0.43 0.46 293

weighted avg 0.73 0.76 0.74 293

from sklearn.model\_selection import GridSearchCV

from xgboost import XGBClassifier

# Define model

xgb = XGBClassifier()

# Define parameter grid

param\_grid = {

    'max\_depth': [3, 5, 7],

    'n\_estimators': [100, 200],

    'learning\_rate': [0.01, 0.1, 0.2]

}

# GridSearch

grid = GridSearchCV(estimator=xgb, param\_grid=param\_grid, cv=3, scoring='accuracy')

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

# Best model after tuning

print(grid.best\_estimator\_)

XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

feature\_weights=None, gamma=None, grow\_policy=None,

importance\_type=None, interaction\_constraints=None,

learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None,

max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=3,

max\_leaves=None, min\_child\_weight=None, missing=nan,

monotone\_constraints=None, multi\_strategy=None, n\_estimators=200,

n\_jobs=None, num\_parallel\_tree=None, ...)

print(grid.best\_estimator\_)

XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

feature\_weights=None, gamma=None, grow\_policy=None,

importance\_type=None, interaction\_constraints=None,

learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None,

max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=3,

max\_leaves=None, min\_child\_weight=None, missing=nan,

monotone\_constraints=None, multi\_strategy=None, n\_estimators=200,

n\_jobs=None, num\_parallel\_tree=None, ...)

**Wine Quality Prediction**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

%matplotlib inline

pip install matplotlib seaborn

Collecting matplotlib

Using cached matplotlib-3.10.6-cp313-cp313-win\_amd64.whl.metadata (11 kB)

Collecting seaborn

Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)

Collecting contourpy>=1.0.1 (from matplotlib)

Using cached contourpy-1.3.3-cp313-cp313-win\_amd64.whl.metadata (5.5 kB)

Collecting cycler>=0.10 (from matplotlib)

Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)

Collecting fonttools>=4.22.0 (from matplotlib)

Downloading fonttools-4.60.0-cp313-cp313-win\_amd64.whl.metadata (113 kB)

Collecting kiwisolver>=1.3.1 (from matplotlib)

Using cached kiwisolver-1.4.9-cp313-cp313-win\_amd64.whl.metadata (6.4 kB)

Requirement already satisfied: numpy>=1.23 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from matplotlib) (2.3.1)

Requirement already satisfied: packaging>=20.0 in [c:\users\lenovo\appdata\roaming\python\python313\site-packages](file:///C:\users\lenovo\appdata\roaming\python\python313\site-packages) (from matplotlib) (25.0)

Collecting pillow>=8 (from matplotlib)

Using cached pillow-11.3.0-cp313-cp313-win\_amd64.whl.metadata (9.2 kB)

Collecting pyparsing>=2.3.1 (from matplotlib)

Downloading pyparsing-3.2.4-py3-none-any.whl.metadata (5.0 kB)

Requirement already satisfied: python-dateutil>=2.7 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from matplotlib) (2.9.0.post0)

Requirement already satisfied: pandas>=1.2 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from seaborn) (2.3.0)

Requirement already satisfied: pytz>=2020.1 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from pandas>=1.2->seaborn) (2025.2)

Requirement already satisfied: tzdata>=2022.7 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from pandas>=1.2->seaborn) (2025.2)

Requirement already satisfied: six>=1.5 in [c:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages](file:///C:\users\lenovo\appdata\local\programs\python\python313\lib\site-packages) (from python-dateutil>=2.7->matplotlib) (1.17.0)

Using cached matplotlib-3.10.6-cp313-cp313-win\_amd64.whl (8.1 MB)

Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)

...

---------------------------------------- 8/8 [seaborn]

Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.60.0 kiwisolver-1.4.9 matplotlib-3.10.6 pillow-11.3.0 pyparsing-3.2.4 seaborn-0.13.2

Note: you may need to restart the kernel to use updated packages.

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?820542e4-c551-4e93-add6-2ae1f6195b8e) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?820542e4-c551-4e93-add6-2ae1f6195b8e)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

[notice] A new release of pip is available: 25.1.1 -> 25.2

[notice] To update, run: python.exe -m pip install --upgrade pip

wine = pd.read\_csv('data.csv')

wine

| **fixed acidity** | **volatile acidity** | **citric acid** | **residual sugar** | **chlorides** | **free sulfur dioxide** | **total sulfur dioxide** | **density** | **pH** | **sulphates** | **alcohol** | **quality** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| 1 | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 | 5 |
| 2 | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 | 5 |
| 3 | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 | 6 |
| 4 | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1594 | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 | 5 |
| 1595 | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 | 6 |
| 1596 | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| 1597 | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 | 5 |
| 1598 | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 | 6 |

1599 rows × 12 columns

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming you already have a dataframe 'wine'

fig = plt.figure(figsize=(10,6))

sns.barplot(x='quality', y='fixed acidity', data=wine)

plt.show()

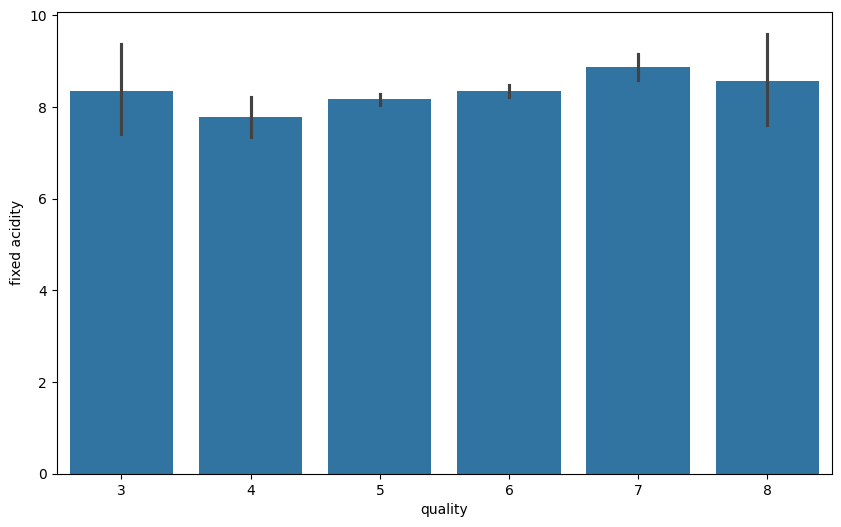
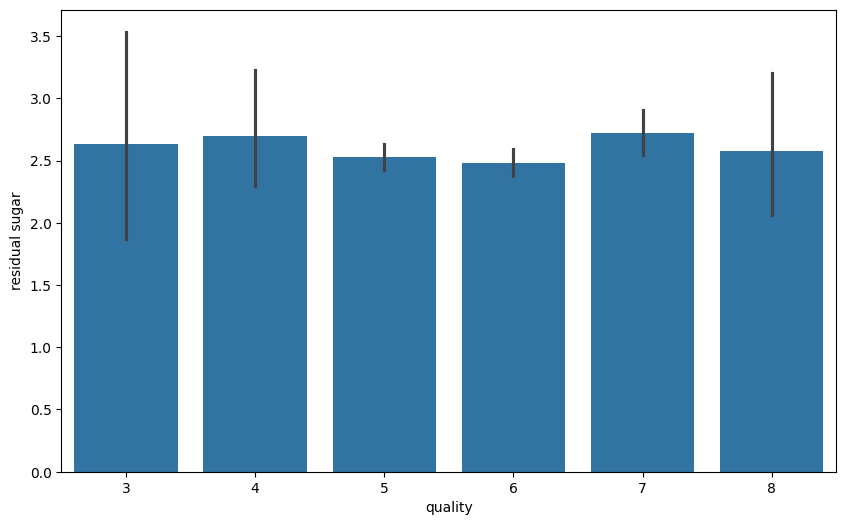


fig = plt.figure(figsize = (10,6))

sns.barplot(x = 'quality', y = 'residual sugar', data = wine)

<Axes: xlabel='quality', ylabel='residual sugar'>



bins = (2, 6.5, 8)

group\_names = ['bad', 'good']

wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group\_names)

wine['quality']

0 bad

1 bad

2 bad

3 bad

4 bad

...

1594 bad

1595 bad

1596 bad

1597 bad

1598 bad

Name: quality, Length: 1599, dtype: category

Categories (2, object): ['bad' < 'good']

from sklearn.preprocessing import LabelEncoder

label\_quality = LabelEncoder()

wine['quality'] = label\_quality.fit\_transform(wine['quality'])

wine['quality'].value\_counts()

quality

0 1382

1 217

Name: count, dtype: int64

X = wine.drop('quality', axis = 1)

y = wine['quality']

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

# Assuming X (features) and y (target) are already defined

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

from sklearn.preprocessing import StandardScaler

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 42)

sc = StandardScaler()

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

X\_test = sc.fit\_transform(X\_test)

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=200)

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

pred\_rfc = rfc.predict(X\_test)

matrixfrom sklearn.metrics import accuracy\_score, classification\_report, confusion\_

print("Accuracy score =", accuracy\_score(y\_test, pred\_rfc))

print("\nClassification Report:\n", classification\_report(y\_test, pred\_rfc))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, pred\_rfc))

Accuracy score = 0.878125

Classification Report:

precision recall f1-score support

0 0.90 0.96 0.93 273

1 0.64 0.38 0.48 47

accuracy 0.88 320

macro avg 0.77 0.67 0.71 320

weighted avg 0.86 0.88 0.86 320

Confusion Matrix:

[[263 10]

[ 29 18]]

from sklearn.svm import SVC

svc = SVC()

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

pred\_svc = svc.predict(X\_test)

print("Accuaracy score =",accuracy\_score(y\_test, pred\_svc))

print(classification\_report(y\_test, pred\_svc))

Accuaracy score = 0.875

precision recall f1-score support

0 0.88 0.98 0.93 273

1 0.71 0.26 0.38 47

accuracy 0.88 320

macro avg 0.80 0.62 0.65 320

weighted avg 0.86 0.88 0.85 320

from sklearn.model\_selection import GridSearchCV

param = {

    'C': [0.1,0.8,0.9,1,1.1,1.2,1.3,1.4],

    'kernel':['linear', 'rbf'],

    'gamma' :[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]

}

grid\_svc = GridSearchCV(svc,param, cv=10, verbose=2)

grid\_svc.fit(X\_train, y\_train)

Fitting 10 folds for each of 128 candidates, totalling 1280 fits

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.1, kernel=linear; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.8, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.8, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.8, kernel=linear; total time= 0.0s

[CV] END ....................C=0.1, gamma=0.8, kernel=linear; total time= 0.0s

...

[CV] END .......................C=1.4, gamma=1.4, kernel=rbf; total time= 0.0s

[CV] END .......................C=1.4, gamma=1.4, kernel=rbf; total time= 0.0s

pred = grid\_svc.predict(X\_test)

print("Accuaracy score =", accuracy\_score(y\_test, pred))

print(classification\_report(y\_test, pred))

Accuaracy score = 0.896875

precision recall f1-score support

0 0.90 0.99 0.94 273

1 0.89 0.34 0.49 47

accuracy 0.90 320

macro avg 0.89 0.67 0.72 320

weighted avg 0.90 0.90 0.88 320

from sklearn.model\_selection import cross\_val\_score

rfc\_eval = cross\_val\_score(estimator = rfc, X = X\_train, y = y\_train, cv = 10, verbose=2)

rfc\_eval.mean()

[CV] END .................................................... total time= 0.4s

[CV] END .................................................... total time= 0.4s

[CV] END .................................................... total time= 0.4s

[CV] END .................................................... total time= 0.5s

[CV] END .................................................... total time= 0.4s

[CV] END .................................................... total time= 0.4s

[CV] END .................................................... total time= 0.5s

[CV] END .................................................... total time= 0.6s

[CV] END .................................................... total time= 0.6s

[CV] END .................................................... total time= 0.6s

[Parallel(n\_jobs=1)]: Done 10 out of 10 | elapsed: 5.8s finished

np.float64(0.9116633858267716)

**Predicting Stock Prices**

import pandas as pd

import numpy as np

import matplotlib as plt

import plotly.graph\_objects as go

import plotly.express as px

from plotly.subplots import make\_subplots

import seaborn as sns

import os

print(os.getcwd())

C:\Users\LENOVO

df = pd.read\_csv(r"C:\Users\LENOVO\Downloads\TSLA.csv")

!pip install yfinance --quiet

import yfinance as yf

import pandas as pd

df = yf.download("TSLA", start="2020-01-01", end="2023-12-31")

df = df.reset\_index() # moves 'Date' index into a column

print(df.head())

C:\Users\LENOVO\AppData\Local\Temp\ipykernel\_668\3079211020.py:5: FutureWarning: YF.download() has changed argument auto\_adjust default to True

df = yf.download("TSLA", start="2020-01-01", end="2023-12-31")

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

Price Date Close High Low Open Volume

Ticker TSLA TSLA TSLA TSLA TSLA

0 2020-01-02 28.684000 28.713333 28.114000 28.299999 142981500

1 2020-01-03 29.534000 30.266666 29.128000 29.366667 266677500

2 2020-01-06 30.102667 30.104000 29.333332 29.364668 151995000

3 2020-01-07 31.270666 31.441999 30.224001 30.760000 268231500

4 2020-01-08 32.809334 33.232666 31.215334 31.580000 467164500

import sys

!{sys.executable} -m pip install yfinance --upgrade --quiet

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

# Assume df is already created (either from CSV or yfinance)

# and has a 'close' column

# Create closedf from df

closedf = df[['close']] # pick only the 'close' column

# Scale the data

scaler = MinMaxScaler(feature\_range=(0,1))

closedf = scaler.fit\_transform(np.array(closedf).reshape(-1,1))

print(closedf.shape)

(1006, 1)

df = df.rename(columns={

'Date': 'date',

'Open': 'open',

'High': 'high',

'Low': 'low',

'Close': 'close',

'Adj Close': 'adj\_close',

'Volume': 'volume'

})

closedf = df[['date','close']]

print("Shape of close dataframe:", closedf.shape)

Shape of close dataframe: (1006, 2)

from sklearn.preprocessing import MinMaxScaler

import numpy as np

from sklearn.preprocessing import MinMaxScaler

# Suppose df already has 'close' column

data = df[['close']] # take only close prices

# scale

scaler = MinMaxScaler(feature\_range=(0,1))

closedf = scaler.fit\_transform(np.array(data).reshape(-1,1))

print(type(closedf)) # should be <class 'numpy.ndarray'>

print(closedf.shape)

# train-test split

training\_size = int(len(closedf) \* 0.70)

test\_size = len(closedf) - training\_size

train\_data = closedf[0:training\_size] # first 70%

test\_data = closedf[training\_size:] # remaining 30%

print("train\_data: ", train\_data.shape)

print("test\_data: ", test\_data.shape)

<class 'numpy.ndarray'>

(1006, 1)

train\_data: (704, 1)

test\_data: (302, 1)

training\_size=int(len(closedf)\*0.70)

test\_size=len(closedf)-training\_size

train\_data,test\_data=closedf[0:training\_size,:],closedf[training\_size:len(closedf),:1]

print("train\_data: ", train\_data.shape)

print("test\_data: ", test\_data.shape)

train\_data: (704, 1)

test\_data: (302, 1)

def create\_dataset(dataset, time\_step=1):

dataX, dataY = [], []

for i in range(len(dataset)-time\_step-1):

a = dataset[i:(i+time\_step), 0] ###i=0, 0,1,2,3-----99 100

dataX.append(a)

dataY.append(dataset[i + time\_step, 0])

return np.array(dataX), np.array(dataY)

time\_step = 15

X\_train, y\_train = create\_dataset(train\_data, time\_step)

X\_test, y\_test = create\_dataset(test\_data, time\_step)

print("X\_train: ", X\_train.shape)

print("y\_train: ", y\_train.shape)

print("X\_test: ", X\_test.shape)

print("y\_test", y\_test.shape)

X\_train: (688, 15)

y\_train: (688,)

X\_test: (286, 15)

y\_test (286,)

!pip install xgboost

Collecting xgboost

Using cached xgboost-3.0.5-py3-none-win\_amd64.whl.metadata (2.1 kB)

Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\site-packages (from xgboost) (2.1.3)

Requirement already satisfied: scipy in c:\users\lenovo\appdata\roaming\python\python313\site-packages (from xgboost) (1.16.1)

Using cached xgboost-3.0.5-py3-none-win\_amd64.whl (56.8 MB)

Installing collected packages: xgboost

Successfully installed xgboost-3.0.5

from xgboost import XGBRegressor

# Example assuming X\_train and y\_train exist

my\_model = XGBRegressor(n\_estimators=1000)

my\_model.fit(X\_train, y\_train, verbose=True)

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

import math

predictions = my\_model.predict(X\_test)

print("Mean Absolute Error - MAE : " + str(mean\_absolute\_error(y\_test, predictions)))

print("Root Mean squared Error - RMSE : " + str(math.sqrt(mean\_squared\_error(y\_test, predictions))))

Mean Absolute Error - MAE : 0.022021977255254153

Root Mean squared Error - RMSE : 0.0287138742512768

**Cancer Prediction**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

data

| **id** | **diagnosis** | **Radius\_mean** | **Texture\_mean** | **perimeter\_mean** | **area\_mean** | **smoothness\_mean** | **compactness\_mean** | **concavity\_mean** | **concave points\_mean** | **...** | **radius\_worst** | **texture\_worst** | **perimeter\_worst** | **area\_worst** | **smoothness\_worst** | **compactness\_worst** | **concavity\_worst** | **concave points\_worst** | **symmetry\_worst** | **fractal\_dimension\_worst** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0.14710 | ... | 25.380 | 17.33 | 184.60 | 2019.0 | 0.16220 | 0.66560 | 0.7119 | 0.2654 | 0.4601 | 0.11890 |
| 1 | 842517 | M | 20.57 | 21.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0.07017 | ... | 24.990 | 23.41 | 158.80 | 1956.0 | 0.12380 | 0.18660 | 0.2416 | 0.1860 | 0.2750 | 0.08902 |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0.12790 | ... | 23.570 | 25.53 | 152.50 | 1709.0 | 0.14440 | 0.42450 | 0.4504 | 0.2430 | 0.3613 | 0.08758 |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0.10520 | ... | 14.910 | 26.50 | 98.87 | 567.7 | 0.20980 | 0.86630 | 0.6869 | 0.2575 | 0.6638 | 0.17300 |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0.10430 | ... | 22.540 | 16.67 | 152.20 | 1575.0 | 0.13740 | 0.20500 | 0.4000 | 0.1625 | 0.2364 | 0.07678 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 564 | 926424 | M | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 | 0.24390 | 0.13890 | ... | 25.450 | 26.40 | 166.10 | 2027.0 | 0.14100 | 0.21130 | 0.4107 | 0.2216 | 0.2060 | 0.07115 |
| 565 | 926682 | M | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 | 0.14400 | 0.09791 | ... | 23.690 | 38.25 | 155.00 | 1731.0 | 0.11660 | 0.19220 | 0.3215 | 0.1628 | 0.2572 | 0.06637 |
| 566 | 926954 | M | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 | 0.09251 | 0.05302 | ... | 18.980 | 34.12 | 126.70 | 1124.0 | 0.11390 | 0.30940 | 0.3403 | 0.1418 | 0.2218 | 0.07820 |
| 567 | 927241 | M | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 | 0.35140 | 0.15200 | ... | 25.740 | 39.42 | 184.60 | 1821.0 | 0.16500 | 0.86810 | 0.9387 | 0.2650 | 0.4087 | 0.12400 |
| 568 | 92751 | B | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 | 0.00000 | 0.00000 | ... | 9.456 | 30.37 | 59.16 | 268.6 | 0.08996 | 0.06444 | 0.0000 | 0.0000 | 0.2871 | 0.07039 |

569 rows × 32 columns

data.isnull().sum()

id 0

diagnosis 0

Radius\_mean 0

Texture\_mean 0

perimeter\_mean 0

area\_mean 0

smoothness\_mean 0

compactness\_mean 0

concavity\_mean 0

concave points\_mean 0

symmetry\_mean 0

fractal\_dimension\_mean 0

radius\_se 0

texture\_se 0

perimeter\_se 0

area\_se 0

smoothness\_se 0

compactness\_se 0

concavity\_se 0

concave points\_se 0

symmetry\_se 0

fractal\_dimension\_se 0

radius\_worst 0

texture\_worst 0

perimeter\_worst 0

...

concavity\_worst 0

concave points\_worst 0

symmetry\_worst 0

fractal\_dimension\_worst 0

dtype: int64

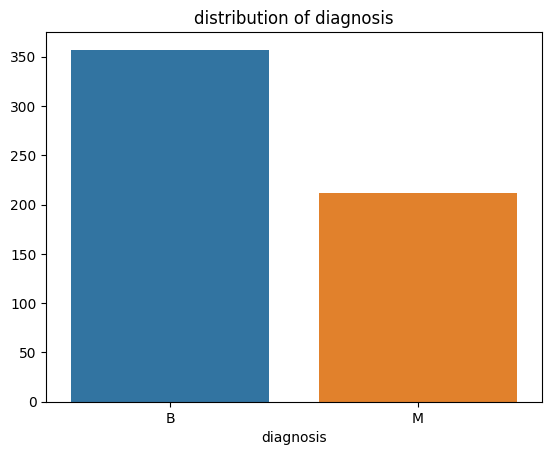
(569, 32)

A=data['diagnosis'].groupby(data['diagnosis']).count()

sns.barplot(x=A.index,y=A.values,data=data)

plt.title("distribution of diagnosis")

plt.show()



data['diagnosis']=data['diagnosis'].map({'M':1,'B':0})

data.head()

| **id** | | **diagnosis** | | | | **Radius\_mean** | | | **Texture\_mean** | | | **perimeter\_mean** | | | | | **area\_mean** | | | | **smoothness\_mean** | | | | | **compactness\_mean** | | | **concavity\_mean** | **concave points\_mean** | **...** | **radius\_worst** | **texture\_worst** | **perimeter\_worst** | **area\_worst** | **smoothness\_worst** | **compactness\_worst** | **concavity\_worst** | **concave points\_worst** | **symmetry\_worst** | **fractal\_dimension\_worst** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 842302 | | | | 1 | | | 17.99 | | | 10.38 | | | | | 122.80 | | | | 1001.0 | | | | | 0.11840 | | | 0.27760 | 0.3001 | 0.14710 | ... | 25.38 | 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 |
| 1 | | 842517 | | | | 1 | | | 20.57 | | | 21.77 | | | | | 132.90 | | | | 1326.0 | | | | | 0.08474 | | | 0.07864 | 0.0869 | 0.07017 | ... | 24.99 | 23.41 | 158.80 | 1956.0 | 0.1238 | 0.1866 | 0.2416 | 0.1860 | 0.2750 | 0.08902 |
| 2 | | 84300903 | | | | 1 | | | 19.69 | | | 21.25 | | | | | 130.00 | | | | 1203.0 | | | | | 0.10960 | | | 0.15990 | 0.1974 | 0.12790 | ... | 23.57 | 25.53 | 152.50 | 1709.0 | 0.1444 | 0.4245 | 0.4504 | 0.2430 | 0.3613 | 0.08758 |
| 3 | | **id** | | | | **diagnosis** | | | **Radius\_mean** | | | **Texture\_mean** | | | | | **perimeter\_mean** | | | | **area\_mean** | | | | | **smoothness\_mean** | | | **compactness\_mean** | **concavity\_mean** | **concave points\_mean** | **...** | **radius\_worst** | **texture\_worst** | **perimeter\_worst** | **area\_worst** | **smoothness\_worst** | **compactness\_worst** | **concavity\_worst** | **concave points\_worst** | **symmetry\_worst** | **fractal\_dimension\_worst** |
| 4 | | 0 | | | | 842302 | | | 1 | | | 17.99 | | | | | 10.38 | | | | 122.80 | | | | | 1001.0 | | | 0.11840 | 0.27760 | 0.3001 | 0.14710 | ... | 25.38 | 17.33 | 184.60 | 2019.0 | 0.1622 | 0.6656 | 0.7119 | 0.2654 | 0.4601 | 0.11890 |
|  | 1 | | 842517 | 1 | 20.57 | | 21.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | | 0.0869 | 0.07017 | ... | 24.99 | | 23.41 | 158.80 | 1956.0 | | 0.1238 | 0.1866 | 0.2416 | 0.1860 | | 0.2750 | 0.08902 |
|  | 2 | | 84300903 | 1 | 19.69 | | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | | 0.1974 | 0.12790 | ... | 23.57 | | 25.53 | 152.50 | 1709.0 | | 0.1444 | 0.4245 | 0.4504 | 0.2430 | | 0.3613 | 0.08758 |
|  | 3 | | 84348301 | 1 | 11.42 | | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | | 0.2414 | 0.10520 | ... | 14.91 | | 26.50 | 98.87 | 567.7 | | 0.2098 | 0.8663 | 0.6869 | 0.2575 | | 0.6638 | 0.17300 |
|  | 4 | | 84358402 | 1 | 20.29 | | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | | 0.1980 | 0.10430 | ... | 22.54 | | 16.67 | 152.20 | 1575.0 | | 0.1374 | 0.2050 | 0.4000 | 0.1625 | | 0.2364 | 0.07678 |

del data['id']

data.head()

| **diagnosis** | **Radius\_mean** | | **Texture\_mean** | | **perimeter\_mean** | | **area\_mean** | | **smoothness\_mean** | | **compactness\_mean** | | **concavity\_mean** | | **concave points\_mean** | | **symmetry\_mean** | | **...** | | **radius\_worst** | | **texture\_worst** | | **perimeter\_worst** | | **area\_worst** | | **smoothness\_worst** | | **compactness\_worst** | | **concavity\_worst** | | **concave points\_worst** | | **symmetry\_worst** | | **fractal\_dimension\_worst** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | | 17.99 | | 10.38 | | 122.80 | | 1001.0 | | 0.11840 | | 0.27760 | | 0.3001 | | 0.14710 | | 0.2419 | | ... | | 25.38 | | 17.33 | | 184.60 | | 2019.0 | | 0.1622 | | 0.6656 | | 0.7119 | | 0.2654 | | 0.4601 | | 0.11890 |
| 1 | 1 | | 20.57 | | 21.77 | | 132.90 | | 1326.0 | | 0.08474 | | 0.07864 | | 0.0869 | | 0.07017 | | 0.1812 | | ... | | 24.99 | | 23.41 | | 158.80 | | 1956.0 | | 0.1238 | | 0.1866 | | 0.2416 | | 0.1860 | | 0.2750 | | 0.08902 |
| 2 | 1 | | 19.69 | | 21.25 | | 130.00 | | 1203.0 | | 0.10960 | | 0.15990 | | 0.1974 | | 0.12790 | | 0.2069 | | ... | | 23.57 | | 25.53 | | 152.50 | | 1709.0 | | 0.1444 | | 0.4245 | | 0.4504 | | 0.2430 | | 0.3613 | | 0.08758 |
| 3 | 1 | | 11.42 | | 20.38 | | 77.58 | | 386.1 | | 0.14250 | | 0.28390 | | 0.2414 | | 0.10520 | | 0.2597 | | ... | | 14.91 | | 26.50 | | 98.87 | | 567.7 | | 0.2098 | | 0.8663 | | 0.6869 | | 0.2575 | | 0.6638 | | 0.17300 |
| 4 | 1 | | 20.29 | | 14.34 | | 135.10 | | 1297.0 | | 0.10030 | | 0.13280 | | 0.1980 | | 0.10430 | | 0.1809 | | ... | | 22.54 | | 16.67 | | 152.20 | | 1575.0 | | 0.1374 | | 0.2050 | | 0.4000 | | 0.1625 | | 0.2364 | | 0.07678 |
| **diagnosis** | | **Radius\_mean** | | **Texture\_mean** | | **perimeter\_mean** | | **area\_mean** | | **smoothness\_mean** | | **compactness\_mean** | | **concavity\_mean** | | **concave points\_mean** | | **symmetry\_mean** | | **...** | | **radius\_worst** | | **texture\_worst** | | **perimeter\_worst** | | **area\_worst** | | **smoothness\_worst** | | **compactness\_worst** | | **concavity\_worst** | | **concave points\_worst** | | **symmetry\_worst** | | **fractal\_dimension\_worst** | |
| 0 | | 1 | | 17.99 | | 10.38 | | 122.80 | | 1001.0 | | 0.11840 | | 0.27760 | | 0.3001 | | 0.14710 | | 0.2419 | | ... | | 25.38 | | 17.33 | | 184.60 | | 2019.0 | | 0.1622 | | 0.6656 | | 0.7119 | | 0.2654 | | 0.4601 | | 0.11890 |
| 1 | | 1 | | 20.57 | | 21.77 | | 132.90 | | 1326.0 | | 0.08474 | | 0.07864 | | 0.0869 | | 0.07017 | | 0.1812 | | ... | | 24.99 | | 23.41 | | 158.80 | | 1956.0 | | 0.1238 | | 0.1866 | | 0.2416 | | 0.1860 | | 0.2750 | | 0.08902 |
| 2 | | 1 | | 19.69 | | 21.25 | | 130.00 | | 1203.0 | | 0.10960 | | 0.15990 | | 0.1974 | | 0.12790 | | 0.2069 | | ... | | 23.57 | | 25.53 | | 152.50 | | 1709.0 | | 0.1444 | | 0.4245 | | 0.4504 | | 0.2430 | | 0.3613 | | 0.08758 |
| 3 | | 1 | | 11.42 | | 20.38 | | 77.58 | | 386.1 | | 0.14250 | | 0.28390 | | 0.2414 | | 0.10520 | | 0.2597 | | ... | | 14.91 | | 26.50 | | 98.87 | | 567.7 | | 0.2098 | | 0.8663 | | 0.6869 | | 0.2575 | | 0.6638 | | 0.17300 |
| 4 | | 1 | | 20.29 | | 14.34 | | 135.10 | | 1297.0 | | 0.10030 | | 0.13280 | | 0.1980 | | 0.10430 | | 0.1809 | | ... | | 22.54 | | 16.67 | | 152.20 | | 1575.0 | | 0.1374 | | 0.2050 | | 0.4000 | | 0.1625 | | 0.2364 | | 0.07678 |

X = data.loc[:,data.columns[1:]]

y = data['diagnosis']

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

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

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

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

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

y\_pred

array([0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,

0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,

0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,

0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

1, 1, 1, 1], dtype=int64)

from sklearn.metrics import accuracy\_score

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

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

print(acc)

0.9298245614035088

from sklearn.model\_selection import cross\_val\_score

import numpy as np

for depth in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]:

  dt = DecisionTreeClassifier(max\_depth=depth)

  dt.fit(X\_train, y\_train)

  trainAccuracy = accuracy\_score(y\_train, dt.predict(X\_train))

  dt = DecisionTreeClassifier(max\_depth=depth)

  valAccuracy = cross\_val\_score(dt, X\_train, y\_train, cv=10)

  print("Depth  : ", depth, " Training Accuracy : ", trainAccuracy, " Cross val score : " ,np.mean(valAccuracy))

Depth : 1 Training Accuracy : 0.9208791208791208 Cross val score : 0.874830917874396

Depth : 2 Training Accuracy : 0.9560439560439561 Cross val score : 0.9142995169082125

Depth : 3 Training Accuracy : 0.9692307692307692 Cross val score : 0.9099999999999999

Depth : 4 Training Accuracy : 0.9802197802197802 Cross val score : 0.9142512077294687

Depth : 5 Training Accuracy : 0.9934065934065934 Cross val score : 0.9230917874396136

Depth : 6 Training Accuracy : 0.9978021978021978 Cross val score : 0.9208695652173914

Depth : 7 Training Accuracy : 1.0 Cross val score : 0.9186473429951691

Depth : 8 Training Accuracy : 1.0 Cross val score : 0.9099033816425122

Depth : 9 Training Accuracy : 1.0 Cross val score : 0.9251690821256039

Depth : 10 Training Accuracy : 1.0 Cross val score : 0.918599033816425

Depth : 11 Training Accuracy : 1.0 Cross val score : 0.8989855072463768

Depth : 12 Training Accuracy : 1.0 Cross val score : 0.9119806763285025

Depth : 13 Training Accuracy : 1.0 Cross val score : 0.9076811594202899

Depth : 14 Training Accuracy : 1.0 Cross val score : 0.9207729468599034

Depth : 15 Training Accuracy : 1.0 Cross val score : 0.9119806763285025

Depth : 16 Training Accuracy : 1.0 Cross val score : 0.9099033816425119

Depth : 17 Training Accuracy : 1.0 Cross val score : 0.9164734299516908

Depth : 18 Training Accuracy : 1.0 Cross val score : 0.9010144927536233

Depth : 19 Training Accuracy : 1.0 Cross val score : 0.9188405797101449

Depth : 20 Training Accuracy : 1.0 Cross val score : 0.9208212560386473

**Marketing Analysis**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import OrdinalEncoder

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import ConfusionMatrixDisplay, classification\_report

df = pd.read\_csv('Bank.csv', sep=';')

df

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| 1 | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| 2 | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| 3 | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| 4 | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 45206 | 51 | technician | married | tertiary | no | 825 | no | no | cellular | 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| 45207 | 71 | retired | divorced | primary | no | 1729 | no | no | cellular | 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| 45208 | 72 | retired | married | secondary | no | 5715 | no | no | cellular | 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| 45209 | 57 | blue-collar | married | secondary | no | 668 | no | no | telephone | 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| 45210 | 37 | entrepreneur | married | secondary | no | 2971 | no | no | cellular | 17 | nov | 361 | 2 | 188 | 11 | other | no |

| **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| 17 | nov | 361 | 2 | 188 | 11 | other | no |

def balanceator(x):

if x < 72:

return 'Class E'

elif x >= 72 and x < 448:

return 'Class D'

elif x >= 448 and x < 1428:

return 'Class C'

elif x >= 1428 and x < df['balance'].quantile(0.99):

return 'Class B'

else:

return 'Class A'

def wrangle(path):

df = pd.read\_csv(path, sep=';') # Read CSV file

df['y'] = df['y'].apply(lambda x: True if x == 'yes' else False) # Change object output to bool

df['default'] = df['default'].apply(lambda x: True if x == 'yes' else False) # Change object output to bool

df['balance\_class'] = df['balance'].apply(lambda x: balanceator(x)) # Creates a new categoric column 'balance\_class' using data from 'balance' column

df['housing'] = df['housing'].apply(lambda x: True if x == 'yes' else False) # Change object output to bool

df['loan'] = df['loan'].apply(lambda x: True if x == 'yes' else False) # Change object output to bool

df['previous\_bool'] = df['previous'].apply(lambda x: True if x != 0 else False) # Change object output to bool for visualization and modeling purpuses

#drop columns:

to\_drop =['previous', 'day', 'poutcome', 'pdays']

df.drop(columns= to\_drop, inplace=True)

return df

df\_pos = wrangle('bank.csv')

df\_pos.head()

|  | age | JOb | marital | education | default | Balance | Housing | loan | contact | month | duration | y | balance\_class | previous\_bool |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 58 | management | married | tertiary | False | 2143 | True | False | unknown | may | 261 | 1 | False | Class B | False |
| 1 | 44 | technician | single | secondary | False | 29 | True | False | unknown | may | 151 | 1 | False | Class E | False |
| 2 | 33 | entrepreneur | married | secondary | False | 2 | True | True | unknown | may | 76 | 1 | False | Class E | False |
| 3 | 47 | blue-collar | married | unknown | False | 1506 | True | False | unknown | may | 92 | 1 | False | Class B | False |
| 4 | 33 | unknown | single | unknown | False | 1 | False | False | unknown | may | 198 | 1 | False | Class E | False |

|  |  |  |  |
| --- | --- | --- | --- |
| campaign | y | Balance\_class | previous\_bool |
| 1 | False | Class B | False |
| 1 | False | Class E | False |
| 1 | False | Class E | False |
| 1 | False | Class B | False |
| 1 | False | Class E | False |

X = df\_pos.drop(columns = ["y", "balance", 'duration'])

y = df\_pos['y']

oe = OrdinalEncoder()

X = oe.fit\_transform(X)

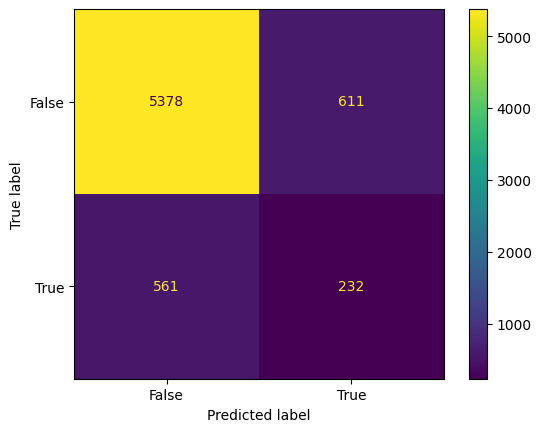
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=42, stratify=y)

dt = GridSearchCV(DecisionTreeClassifier(random\_state=42), {}, n\_jobs=-1, cv=10, refit="recall")

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

ConfusionMatrixDisplay.from\_estimator(dt,X\_test,y\_test)

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x1d9b72aa760>



pred = dt.predict(X\_test)

print (classification\_report(pred, y\_test))

precision recall f1-score support

False 0.90 0.91 0.90 5939

True 0.29 0.28 0.28 843

accuracy 0.83 6782

macro avg 0.60 0.59 0.59 6782

weighted avg 0.82 0.83 0.82 6782

params\_dt = {

"max\_depth": [5, 10, 15, 20, 25, 30, None], # Maximum depth of the decision tree

"criterion": ["gini","entropy"], # The quality criterion to measure the information gain when splitting nodes

"min\_samples\_split": [2,3], # Minimum number of samples required to split an internal node

"min\_samples\_leaf": [1,2] # Minimum number of samples required to be at a leaf node

}

model\_dt = GridSearchCV(

DecisionTreeClassifier(random\_state=42), # Define the Decision Tree model

params\_dt, # Pass in the hyperparameters to be tuned from the dictionary we defined earlier

cv=10, # Set the number of folds for cross-validation

verbose=2

)

model\_dt.fit(X\_train, y\_train)

Fitting 10 folds for each of 56 candidates, totalling 560 fits

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.1s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=1, min\_samples\_split=3; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=2, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=2, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=2, min\_samples\_split=2; total time= 0.0s

[CV] END criterion=gini, max\_depth=5, min\_samples\_leaf=2, min\_samples\_split=2; total time= 0.0s

...

[CV] END criterion=entropy, max\_depth=None, min\_samples\_leaf=2, min\_samples\_split=3; total time= 0.1s

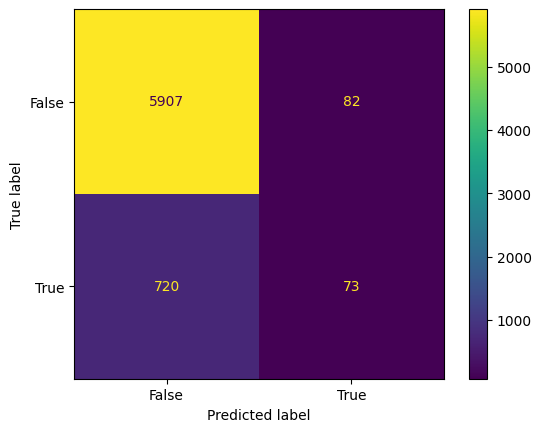
[CV] END criterion=entropy, max\_depth=None, min\_samples\_leaf=2, min\_samples\_split=3; total time= 0.1s

[CV] END criterion=entropy, max\_depth=None, min\_samples\_leaf=2, min\_samples\_split=3; total time= 0.1s

[CV] END criterion=entropy, max\_depth=None, min\_samples\_leaf=2, min\_samples\_split=3; total time= 0.2s

ConfusionMatrixDisplay.from\_estimator(model\_dt,X\_test,y\_test)

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x1d9b739b460>



pred\_dt = model\_dt.predict(X\_test)

print (classification\_report(pred\_dt, y\_test))

precision recall f1-score support

False 0.99 0.89 0.94 6627

True 0.09 0.47 0.15 155

accuracy 0.88 6782

macro avg 0.54 0.68 0.55 6782

weighted avg 0.97 0.88 0.92 6782