```
1 !pip install pandas
2 !pip install numpy
3
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.1
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/py
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/loc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/pyt
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3
Requirement already satisfied: numpy in /usr/local/lib/python3.11
```

```
1 import pandas as pd
2 import numpy as np
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
1 ds_path = f"/content/drive/MyDrive/Intellipaat-sem2/ML Algo/Assign
```

```
1 df_mushroom = pd.read_csv(ds_path)
```

1. Data Exploration

```
1 # 1. The first few rows of the dataset
2 print("df_mushroom.head()")
3 print(df_mushroom.head())
```

```
df_mushroom.head()
  class cap-shape cap-surface cap-color bruises odor gill-attachm
0     p         x           s         n       t    p
1     e         x           s         y       t    a
2     e         b           s         w       t    l
3     p         x           y         w       t    p
4     e         x           s         g       f    n

  gill-spacing gill-size gill-color  ... stalk-surface-below-ring
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-co
0                      w                      w         p
1                      w                      w         p
2                      w                      w         p
3                      w                      w         p
4                      w                      w         p

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

```
1
2 # 2. Info of dataset (column names, non-null count, data types)
```

Resources

You are not subscribed. Learn more

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

At your current usage level, this runtime may last up to 2 hours 30 minutes.

Manage sessions

Want more memory and disk space?

Upgrade to Colab Pro

Python 3 Google Compute Engine backend (GPU)

Showing resources from 8:09 AM to 9:25 AM

System RAM
2.1 / 12.7 GB

GPU RAM
0.0 / 15.0 GB

Disk
31.1 / 112.6 GB

```
print("Dataset Info:")
print(df_mushroom.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
None
```

```
# 3. Look for missing values
print("\nMissing Values in Each Column:")
print(df_mushroom.isnull().sum())

# Get the columns with missing values
columns_with_nulls = df_mushroom.columns[df_mushroom.isnull().any()]

# Print the column names with missing values
print("Columns with missing values:")
print(columns_with_nulls)
```

```

Missing Values in Each Column:
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

```
Columns with missing values:
Index([], dtype='object')
```

```
# 4. unique values in each column
print("\nUnique Values in Each Column:")
for column in df_mushroom.columns:
    print(f"{column}: {df_mushroom[column].nunique()} unique values")
```

```
Unique Values in Each Column:
class: 2 unique values
cap-shape: 6 unique values
cap-surface: 4 unique values
cap-color: 10 unique values
bruises: 2 unique values
odor: 9 unique values
gill-attachment: 2 unique values
gill-spacing: 2 unique values
gill-size: 2 unique values
gill-color: 12 unique values
stalk-shape: 2 unique values
stalk-root: 5 unique values
stalk-surface-above-ring: 4 unique values
stalk-surface-below-ring: 4 unique values
stalk-color-above-ring: 9 unique values
stalk-color-below-ring: 9 unique values
veil-type: 1 unique values
veil-color: 4 unique values
ring-number: 3 unique values
ring-type: 5 unique values
spore-print-color: 9 unique values
population: 6 unique values
habitat: 7 unique values
```

```
# 5. Distribution of the target variable ('class')
print("\nTarget Variable [ class ] Distribution:")
print(df_mushroom['class'].value_counts())
```

```
Target Variable [ class ] Distribution:
class
e    4208
p    3916
Name: count, dtype: int64
```

```
# 6. Basic statistics
print("\nBasic Statistics:")
print(df_mushroom.describe(include='all'))
```

```
Basic Statistics:
       class cap-shape cap-surface cap-color bruises  odor gill-a
count   8124      8124        8124      8124    8124  8124
unique     2         6           4        10       2     9
top        e         x           y         n       f     n
freq    4208      3656        3244      2284    4748  3528

       gill-spacing gill-size gill-color  ... stalk-surface-below
count          8124      8124       8124  ...
unique            2         2         12  ...
top               c         b          b  ...
freq           6812      5612       1728  ...

       stalk-color-above-ring stalk-color-below-ring veil-type ve
count                    8124                   8124      8124
unique                      9                      9         1
top                         w                      w         p
freq                     4464                   4384      8124

       ring-number ring-type spore-print-color population habitat
count         8124      8124              8124       8124    8124
unique           3         5                 9          6       7
```

```
top              o         p                w          v       d
freq          7488      3968             2388       4040    3148

[4 rows x 23 columns]
```

```
# 7. Check the distribution of values for each categorical column
for column in df_mushroom.columns:
    print(f"\nValue Counts for {column}:")
    print(df_mushroom[column].value_counts())
```

```
stalk-color-below-ring
w    4384
p    1872
g     576
n     512
b     432
o     192
e      96
c      36
y      24
Name: count, dtype: int64

Value Counts for veil-type:
veil-type
p    8124
Name: count, dtype: int64

Value Counts for veil-color:
veil-color
w    7924
n      96
o      96
y       8
Name: count, dtype: int64

Value Counts for ring-number:
ring-number
o    7488
t     600
n      36
Name: count, dtype: int64

Value Counts for ring-type:
ring-type
p    3968
e    2776
l    1296
f      48
n      36
Name: count, dtype: int64

Value Counts for spore-print-color:
spore-print-color
w    2388
n    1968
k    1872
h    1632
r      72
u      48
o      48
y      48
b      48
Name: count, dtype: int64

Value Counts for population:
population
v    4040
y    1712
s    1248
```

```
print(df_mushroom.shape)
```

```
(8124, 23)
```

```
# Remove data with any missing information for now
null_columns = len(columns_with_nulls)
print(null_columns)
if (null_columns > 0):
  df_mushroom = df_mushroom.dropna()
```

```
0
```

```
print(df_mushroom.shape)
```

```
(8124, 23)
```

```
# Get summary stats for the categorical features
df_mushroom.describe(include = ['O'])
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	s
count	8124	8124	8124	8124	8124	8124	8124	
unique	2	6	4	10	2	9	2	
top	e	x	y	n	f	n	f	
freq	4208	3656	3244	2284	4748	3528	7914	

4 rows × 23 columns

```
df_mushroom.describe()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	s
count	8124	8124	8124	8124	8124	8124	8124	
unique	2	6	4	10	2	9	2	
top	e	x	y	n	f	n	f	
freq	4208	3656	3244	2284	4748	3528	7914	

4 rows × 23 columns

```
# Create the feature/flag for eatable / non eatable
df_mushroom['class'] = df_mushroom['class'].apply(lambda x: 1 if x
```

```
df_mushroom.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill spacir
0	0	x	s	n	t	p	f	
1	1	x	s	y	t	a	f	
2	1	b	s	w	t	l	f	
3	0	x	y	w	t	p	f	
4	1	x	s	g	f	n	f	

5 rows × 23 columns

```
1 # Check the event rate
2 df_mushroom['class'].value_counts()/len(df_mushroom)
```

class	count
1	0.517971
0	0.482029

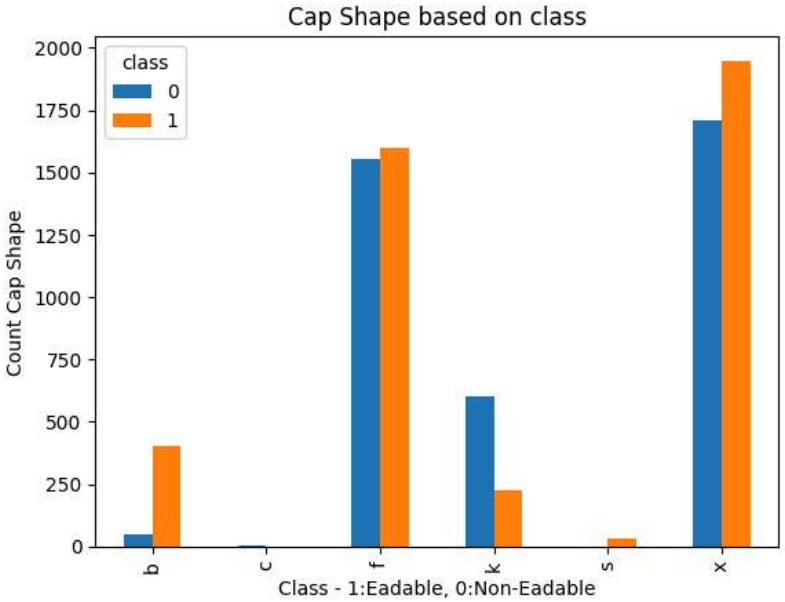
dtype: float64

```
1 !pip install matplotlib
2 !pip install seaborn
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/pytho
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/pyt
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/li
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/li
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/
Requirement already satisfied: pillow>=8 in /usr/local/lib/python
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib
Requirement already satisfied: python-dateutil>=2.7 in /usr/local
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3
Requirement already satisfied: seaborn in /usr/local/lib/python3.
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/pyth
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/lo
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/pyt
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/li
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/li
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/
Requirement already satisfied: pillow>=8 in /usr/local/lib/python
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib
Requirement already satisfied: python-dateutil>=2.7 in /usr/local
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/pyt
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3
```

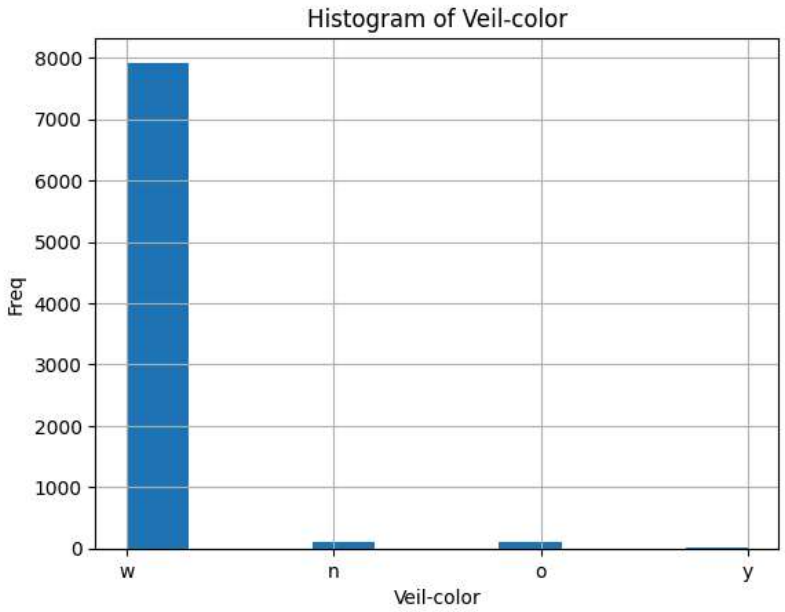
```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 pd.crosstab(df_mushroom['cap-shape'],df_mushroom['class']).plot(kind
5 plt.title('Cap Shape based on class')
6 plt.xlabel('Class - 1:Eadable, 0:Non-Eadable')
7 plt.ylabel('Count Cap Shape')
```

```
Text(0, 0.5, 'Count Cap Shape')
```




```
#veil-color: brown=n,orange=o,white=w,yellow=y

df_mushroom['veil-color'].hist()
plt.title('Histogram of Veil-color')
plt.xlabel('Veil-color')
plt.ylabel('Freq')

```

```
Text(0, 0.5, 'Freq')
```




```
# Convert the categorical features
# Creating dummies for all these variables
d_cap_shape = pd.get_dummies(df_mushroom['cap-shape'], prefix='cap
d_cap_surface = pd.get_dummies(df_mushroom['cap-surface'], prefix=
d_cap_color = pd.get_dummies(df_mushroom['cap-color'], prefix='cap
d_bruises = pd.get_dummies(df_mushroom['bruises'], prefix='bruises
d_odor = pd.get_dummies(df_mushroom['odor'], prefix='odor',dtype=i
d_gill_attachment = pd.get_dummies(df_mushroom['gill-attachment'],
```

```
d_gill_spacing = pd.get_dummies(df_mushroom['gill-spacing'], prefi
d_gill_size = pd.get_dummies(df_mushroom['gill-size'], prefix='gil
d_gill_color = pd.get_dummies(df_mushroom['gill-color'], prefix='g
d_stalk_shape = pd.get_dummies(df_mushroom['stalk-shape'], prefix=
d_stalk_root = pd.get_dummies(df_mushroom['stalk-root'], prefix='s
d_stalk_surface_above_ring = pd.get_dummies(df_mushroom['stalk-sur
d_stalk_surface_below_ring = pd.get_dummies(df_mushroom['stalk-sur
d_stalk_color_above_ring = pd.get_dummies(df_mushroom['stalk-color
d_stalk_color_below_ring = pd.get_dummies(df_mushroom['stalk-color
d_veil_type = pd.get_dummies(df_mushroom['veil-type'], prefix='vei
d_veil_color = pd.get_dummies(df_mushroom['veil-color'], prefix='v
d_ring_number = pd.get_dummies(df_mushroom['ring-number'], prefix=
d_ring_type = pd.get_dummies(df_mushroom['ring-type'], prefix='rin
d_spore_print_color = pd.get_dummies(df_mushroom['spore-print-colo
d_population = pd.get_dummies(df_mushroom['population'], prefix='p
d_habitat = pd.get_dummies(df_mushroom['habitat'], prefix='habitat


```

```
# Create the final dataset with all the relevant features - both d
feature_x_cont = ['class','cap-shape','cap-surface','cap-color','b

df_mushroom_cont = df_mushroom[feature_x_cont]

# Creating the Final data with all the relevant fields and Dep Var
df_mushroom_new = pd.concat([df_mushroom['class'],d_cap_shape,d_ca
```

```
df_mushroom_new.shape
```

```
(8124, 118)
```

```
df_mushroom_new.head()
```

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	su
0	0	0	0	0	0	0	1	
1	1	0	0	0	0	0	1	
2	1	1	0	0	0	0	0	
3	0	0	0	0	0	0	1	
4	1	0	0	0	0	0	1	

5 rows × 118 columns

```
df_mushroom_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Columns: 118 entries, class to habitat_w
dtypes: int64(118)
memory usage: 7.3 MB
```

Train Test Split

Double-click (or enter) to edit

```
#X = df_mushroom_new.drop('class', axis=1)
#y = df_mushroom_new['class']

```

```
X = df_mushroom.drop('class', axis=1)
y = df_mushroom['class']

X = X.map(lambda x: ord(x))
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.75
```

Model Building

```
import sklearn.metrics as metrics
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score
```

```
# Step 3: Define the model
model = RandomForestClassifier(random_state=42)

# Step 4: Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],        # Number of trees in th
    'max_depth': [None, 10, 20],           # Maximum depth of the
    'min_samples_split': [2, 5, 10],       # Minimum number of sam
    'min_samples_leaf': [1, 2, 4]          # Minimum number of sam
}

# Step 5: Set up GridSearchCV
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=5,                  # 5-fold cross-validation
    scoring='accuracy',    # Performance metric
    verbose=1,             # Print progress
    n_jobs=-1              # Use all available CPU cores
)

# Step 6: Perform the grid search
grid_search.fit(X_train, y_train)

# Step 7: Print the best hyperparameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Step 8: Evaluate the model on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Set Accuracy:", test_accuracy)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_
Best Cross-Validation Accuracy: 1.0
Test Set Accuracy: 1.0
```

```
best_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=50, random_state=42)
```

```
1 best_model.score(X_train,y_train)
```

```
1.0
```

Model Prediction

```
1 y_train_pred=best_model.predict(X_train)
```

```
1 y_test_pred=best_model.predict(X_test)
```

Model Evaluation

```
1 from sklearn.metrics import confusion_matrix,accuracy_score,recall_
```

```
1 print('Train Accuracy:',accuracy_score(y_train,y_train_pred))
2 print('Test Accuracy:',accuracy_score(y_test,y_test_pred))
3 print('Train Recall:',recall_score(y_train,y_train_pred))
4 print('Test Recall:',recall_score(y_test,y_test_pred))
5 print('Train Precision:',precision_score(y_train,y_train_pred))
6 print('Test Precision:',precision_score(y_test,y_test_pred))
7 print('Train F1 Score:',f1_score(y_train,y_train_pred))
8 print('Test F1 Score:',f1_score(y_test,y_test_pred))
```

```
Train Accuracy: 1.0
Test Accuracy: 1.0
Train Recall: 1.0
Test Recall: 1.0
Train Precision: 1.0
Test Precision: 1.0
Train F1 Score: 1.0
Test F1 Score: 1.0
```

```
1 print(classification_report(y_train,y_train_pred))
2 print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3 print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2925
           1       1.00      1.00      1.00      3168

    accuracy                           1.00      6093
   macro avg       1.00      1.00      1.00      6093
weighted avg       1.00      1.00      1.00      6093

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       991
           1       1.00      1.00      1.00      1040

    accuracy                           1.00      2031
   macro avg       1.00      1.00      1.00      2031
weighted avg       1.00      1.00      1.00      2031
```

```
from sklearn.tree import DecisionTreeClassifier
```

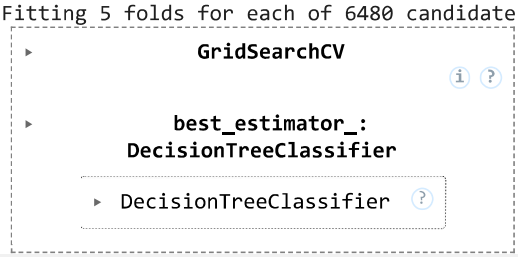
```
DT_gcv=DecisionTreeClassifier()
```

```
param_grid = {'max_features': ['sqrt', 'log2'],
              'ccp_alpha': [1,0.1, .01, .001,0.0001,0.00001],
              'max_depth' : [4,5,6,7,8,9,10,11,12,15],
              'criterion' :['gini', 'entropy','log_loss'],
              'splitter' :['best', 'random'],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4]
             }
```

```
model_cv=GridSearchCV(DT_gcv,param_grid,scoring="f1",verbose=2,cv=
```

```
model_cv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 6480 candidates, totalling 32400 fits
```




```
best_model=model_cv.best_estimator_
best_param=model_cv.best_params_
```

```
print(best_param)
```

```
{'ccp_alpha': 0.001, 'criterion': 'entropy', 'max_depth': 10, 'ma
```

```
best_model.fit(X_train,y_train)
```

```
y_train_pred_cv=best_model.predict(X_train)
```

```
y_test_pred_cv=best_model.predict(X_test)
```

```
print(classification_report(y_train,y_train_pred_cv))
print("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")
print(classification_report(y_test,y_test_pred_cv))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2925
           1       1.00      1.00      1.00      3168

    accuracy                           1.00      6093
   macro avg       1.00      1.00      1.00      6093
weighted avg       1.00      1.00      1.00      6093

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       991
           1       1.00      1.00      1.00      1040

    accuracy                           1.00      2031
   macro avg       1.00      1.00      1.00      2031
weighted avg       1.00      1.00      1.00      2031
```

Linear Regression is a basic supervised machine learning algorithm where the relationship between the dependent variable and one or more independent variables is modeled by a straight line (linear equation).

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Define the model
model = LogisticRegression( )  # Increase max_iter as needed


```

```
model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_log
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opt
    https://scikit-learn.org/stable/modules/linear_model.html#log
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

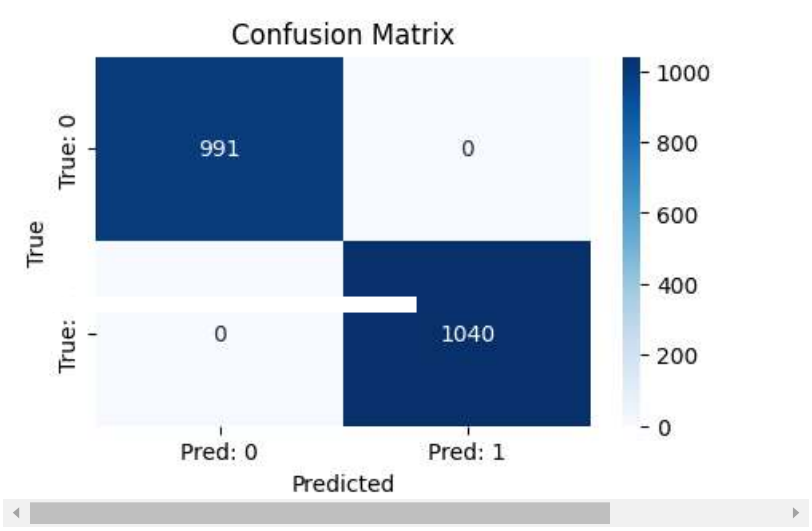
```
y_train_pred = model.predict(X_test)
y_test_pred = model.predict(X_test)


```

```
print('Train Accuracy:',accuracy_score(y_train,y_train_pred))
print('Test Accuracy:',accuracy_score(y_test,y_test_pred))
print('Train Recall:',recall_score(y_train,y_train_pred))
print('Test Recall:',recall_score(y_test,y_test_pred))
print('Train Precision:',precision_score(y_train,y_train_pred))
print('Test Precision:',precision_score(y_test,y_test_pred))
print('Train F1 Score:',f1_score(y_train,y_train_pred))
print('Test F1 Score:',f1_score(y_test,y_test_pred))
print('Train roc_auc_score:',roc_auc_score(y_train,y_train_pred))
print('Test roc_auc_score:',roc_auc_score(y_test,y_test_pred))
```

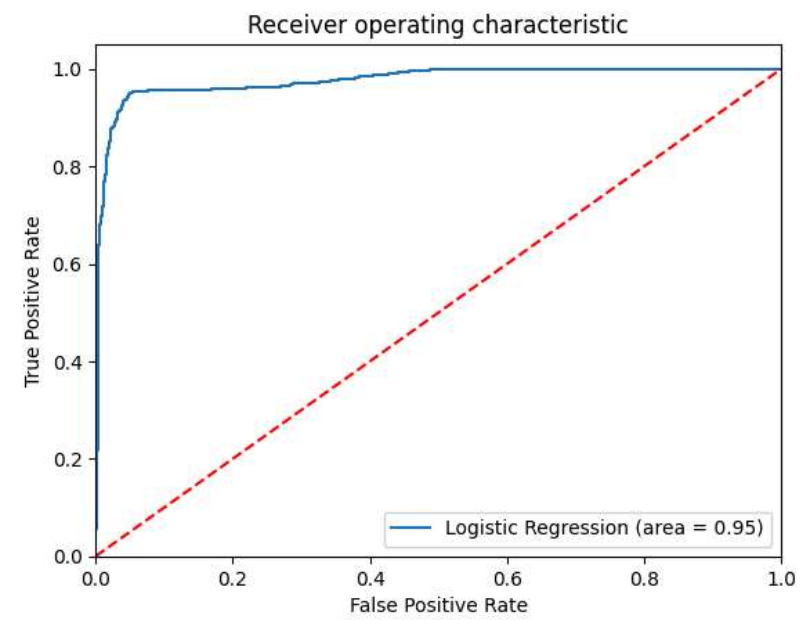
```
Train Accuracy: 1.0
Test Accuracy: 1.0
Train Recall: 1.0
Test Recall: 1.0
Train Precision: 1.0
Test Precision: 1.0
Train F1 Score: 1.0
Test F1 Score: 1.0
Train roc_auc_score: 1.0
Test roc_auc_score: 1.0
```

```
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cm = confusion_matrix(y_test,y_test_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(5, 3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['P
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
plt.show()
```




```
# a graph that shows how well a model performs across all possible
# against the false positive rate (FPR)
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, model.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_tes
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % lo
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Hyperparameter tuning with GridSearchCv

```
from sklearn.metrics import classification_report# Define the para
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Inverse of regularization str
    'solver': ['lbfgs', 'liblinear', 'saga'],  # Solvers
    'l1_ratio': [0, 0.5, 1],  # Used only with elasticnet
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           verbose=1,
                           n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)

# Best parameters and model
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

# Evaluate the model
y_pred = best_model.predict(X_test)
print("\nClassification Report:\n", classification_report(y_test,
```

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
Best Parameters: {'C': 10, 'l1_ratio': 0, 'solver': 'liblinear'}

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       991
           1       1.00      1.00      1.00      1040

    accuracy                           1.00      2031
   macro avg       1.00      1.00      1.00      2031
weighted avg       1.00      1.00      1.00      2031

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_log
  warnings.warn(
```

```
best_model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_log
  warnings.warn(
```

```
LogisticRegression(C=10, l1_ratio=0, solver='liblinear')
```

```
y_test_tred = model.predict(X_test)
```

SVM

```
from sklearn.svm import SVC
svm = SVC(kernel='rbf',random_state=0,probability=True)
svm.fit(X_train,y_train)
```

```
SVC(probability=True, random_state=0)
```

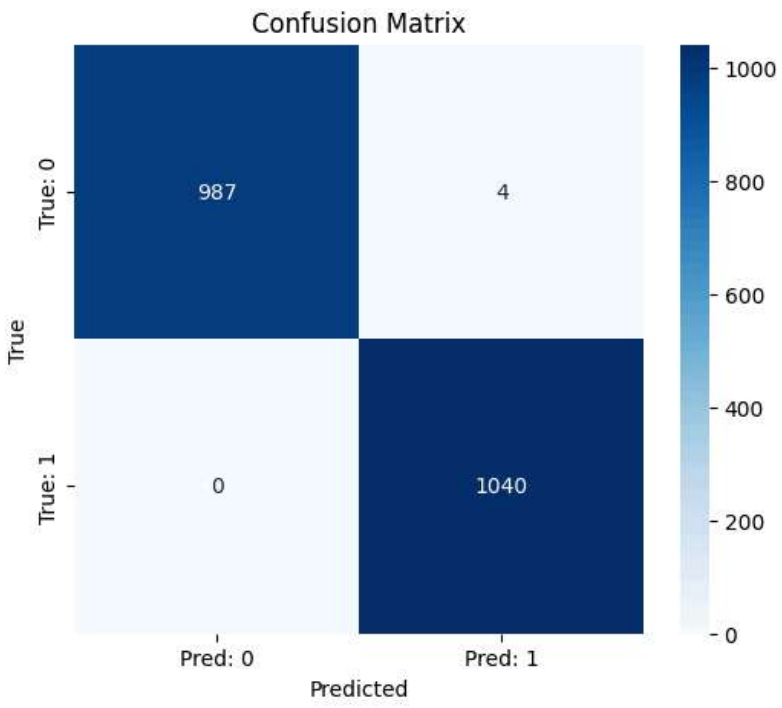
```
y_test_pred = svm.predict(X_test)
```

```
y_train_pred = svm.predict(X_train)
```

```
print('Train Accuracy:',accuracy_score(y_train,y_train_pred))
print('Test Accuracy:',accuracy_score(y_test,y_test_pred))
print('Train Recall:',recall_score(y_train,y_train_pred))
print('Test Recall:',recall_score(y_test,y_test_pred))
print('Train Precision:',precision_score(y_train,y_train_pred))
print('Test Precision:',precision_score(y_test,y_test_pred))
print('Train F1 Score:',f1_score(y_train,y_train_pred))
print('Test F1 Score:',f1_score(y_test,y_test_pred))
print('Train roc_auc_score:',roc_auc_score(y_train,y_train_pred))
print('Test roc_auc_score:',roc_auc_score(y_test,y_test_pred))
```

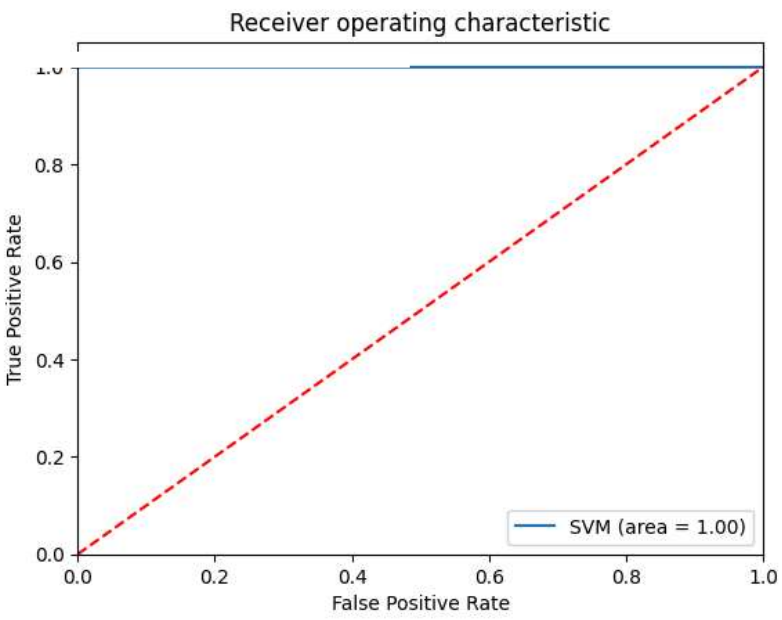
```
Train Accuracy: 0.995240439849007
Test Accuracy: 0.9980305268340719
Train Recall: 0.9996843434343434
Test Recall: 1.0
Train Precision: 0.9912363067292644
Test Precision: 0.9961685823754789
Train F1 Score: 0.9954424013829954
Test F1 Score: 0.9980806142034548
Train roc_auc_score: 0.9950558469308469
Test roc_auc_score: 0.9979818365287588
```

```
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['P
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
# a graph that shows how well a model performs across all possible
# against the false positive rate (FPR)
```

```
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, svm.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, svm.predict_proba(X_test)
plt.figure()
plt.plot(fpr, tpr, label='SVM (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
# Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel types
}

# Initialize GridSearchCV
grid_search_svm = GridSearchCV(estimator=svm,
                               param_grid=param_grid,
                               cv=5,
                               scoring='accuracy',
                               verbose=1,
                               n_jobs=-1)

# Fit the model
grid_search_svm.fit(X_train, y_train)

# Best parameters and model
print("Best Parameters:", grid_search_svm.best_params_)
best_model_svm = grid_search_svm.best_estimator_

# Evaluate the model
y_test_pred = best_model_svm.predict(X_test)
print("\nClassification Report:\n", classification_report(y_test, y_

```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

**Conclusion : Three Models are tried **

1. RandomForestClassifier
2. DecisionTreeClassifier
3. LogisticRegression
4. SVM

Change runtime type