ML

▼ **Numpy**

▼ Arrays

```
import numpy as np

li =[1,2,3,4]
np.array(li)   ←this will convert the list into an array
'''
output:
[1 2 3 4]
'''

num = np.arange(0,100) ← this will make an array having elements 0, 1, 2,.....,99
num2 = np.arange(0,100,5) ← this array will have numbers with comm on difference of 5

mat = np.zeros(5)  ← this will make an array like  [0. 0. 0. 0. 0.]
mat2 = np.zeros((5,3)) ← array will be 2-D [[0. 0. 0.]
                                            [0. 0. 0.]
                                           [0. 0. 0.]
                                           [0. 0. 0.]
                                           [0. 0. 0.]]
mat3 = np.ones(3) ← array will be [1. 1. 1.]
mat4 = np.ones((3,3,3)) ← array will be 3-D  [[[1. 1. 1.]
                                               [1. 1. 1.]
                                               [1. 1. 1.]]

                                              [[1. 1. 1.]
                                               [1. 1. 1.]
```

```
[1. 1. 1.]]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]]
```

np.linspace(startpoint, endpoint, number_of_numbers_inbetween)
np.linspace(0,10,20) ← the array will have evenly spaced 20 numbers starting from 0 and ending with 10

np.eye(3) < - this will give an identity matrix of order 3

np.random.rand(3) < -this will give 3 random numbers in interval [0,1)
np.random.rand(3,2) ← this will give 3 rows and 2 columns of random numbers in interval [0,1)
np.random.randint(1,50,6) ← this will give 6 random numbers in range 1 to 50
np.random.randint(1,50,(3,4)) ← this will give 3 rows and 4 columns of 6 random numbers in range 1 to 50

np.random.seed(any_number)
np.random.rand(limits_for_number) ← this will save the random numbers, whenever np.random.seed(any_number) will be called it will give the same random numbers

ar = np.arange(0,20)
ar.reshape(4,5) ←this will give a array of 4 rows and 5 columns
ar.reshape(2,10)
ar.reshape(5,4)
ar.reshape(10,2)
the order should equal to the number of elements in ar

ar.max() ← this will give the highest number in the array
ar.argmax() ← this will give the index of the highest number

```
ar.min() ← this will give the lowest number in the array
ar.argmin() ← this will give the index of the lowest number
```

▼ Indexing and selection

```
ar = np.arange(30)
ar[2] will give number at index 2
ar[2:8] will give an array of numbers between index 2 to 8
ar[2:5]=100 will change all the numbers to 100 between index 2 to 5

ar2 = ar.reshape(5,6)
ar2[2,5] will give the element in the 2nd row and 5th column
ar2[1:3,2:5] will give elements between 1st to 3rd row and 2nd to 5th c
olumn

ar2=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(ar2+2) this will add 2 two every element and print
AND EVERY OTHER OPERATION
print(ar2<5) this will print boolean value in the form of given array
print(np.sin(ar2))
print(np.sin(np.cos(np.log(ar2)))) >>>>>> np.function

ar2=np.array([[1,2,3],[4,5,6],[7,8,9]])
x=ar2<8
print(ar2[x]) this will print an array:- [1 2 3 4 5 6 7]


ar2=np.array([[[1,2,3],
               [4,5,6],
               [7,8,9]],
              [[13,14,15],
               [16,17,18],
               [19,20,21]],
              [[22,23,24],
               [25,26,27],
```

```
            [28,29,30]]])



print(ar2.sum(axis=0)) this will print sum of columns [[36 39 42]
                                                      [45 4
8 51]
                                                      [54 5
7 60]]

                                [11+13+22, 2+14+23, 3+15+24] = [36,39,42]
                                [4+16+25,  5+17+26, 6+18+27] = [45,48,51]
                                [7+19+28,  8+20+29, 9+21+30] = [54,57,60]

print(ar2.sum(axis=1)) this will print sum of rows [[12 15 18]
                                                    [48 51 54]
                                                    [75 78 81]]
                                [1+4+7, 2+5+8, 3+6+9] = [12, 15, 18]
                                [13+16+19, 14+17+20, 15+18+21] = [48, 51, 5
4]
                                [22+25+28, 23+26+29, 24+27+30] = [75, 7
8, 81]
```

▼ **Pandas**

(indexing can be done with strings)

▼ Series

```
import pandas as pd
list = [2,34,52,6]
naming = ['a', 'b','c','d']
pd.Series(index=naming, data=list) it will assign every data a keyword
resp.
This is LABELLED INDEXING
```

Dictionary can also used,
dictn={'a':1, 'b':2, 'c':3}

x=pd.Series(dictn)

dictn2={'d':4, 'e':5, 'f':6}
y=pd.Series(dictn2)

print(x.add(y, fill_value=0)) fill_value will add value to the data

▼ Dataframes

import pandas as pd
d=[[2,4,6],
  [4,8,12],
  [6,12,18]]
i=['a','b','c']
c=['x','y','z']

df = print(pd.DataFrame(data=d,index=i,columns=c)) this will give
  x y z
a 2 4 6
b 4 8 12
c 6 12 18
it can also done as,
print(pd.DataFrame(d,i,c)) if index or column are not given then it will defaultly assign 0,1,2.. as the index
pd.DataFrame(d,i,c).transpose() will give transpose

#to call only specific row,
pd.DataFrame(data=d,index=i,columns=c).head(2) 2 is the number rows from top
pd.DataFrame(data=d,index=i,columns=c).head() will give complete

```
dataframe

#to call only specific columns,
m=['x','y']
pd.DataFrame(data=d,index=i,columns=c)[m]
OR
pd.DataFrame(data=d,index=i,columns=c)[['x','y']]

#to set any row as index,
df.set_index("name_of_row")

df.loc['name_of_specific_row']  will show the data in that row
df.iloc[index_as_integer]  will give the same

df.loc['row1','row2'...,'rowN']  will show the data in that rows
df.iloc[0:4]  will show those rows

#to remove a row,
df.drop('name_of_row_')

#appending a row,
df2=df.loc['a']
df.append(df2)



#operations on columns,
d=[[2,4,6,10],
   [4,8,12,20],
   [6,12,18,30],
   [8,16,24,40]]
i=['a','b','c','d']
c=['w','x','y','z']
t=pd.DataFrame(data=d,index=i,columns=c)

to sort columns by comparison,
t['x']<18  this will give boolean values acc to conditions
```

```
to  remove columns,
t[t['x']<8]

here, and or is also used as &, |
t[ (t['z'] > 18)   |   (t['y'] > 10) ]
t[ (t['z'] > 18)   &   (t['y'] > 10) ]

t[ (t['w'] == 6)  |  t['w'] == 12)  |  (t['w'] == 18)  |  (t['w'] == 20) ]
for too many data using for and or,
make a list and put the data tob be compared in it
cc=[6,12,18,20]
t[t.isin(cc)]



#methods on single column,
new column can be added by,
t['name_of_column'] = {dictionary)

new column using exisiting column,
def num(n):
  return n*n

t['xyz']=t['w'].apply(num)
print(t)  this will give a new column xyz with data accordingly the num function

rows can be sorted by,
t.sort_values('name_of_column',ascending=True(or False))

sorting by multiple columns
t.sort_values(['column1','column2',..,'columnN'],ascending=True(or False))

for max and min in the column,
t[['x','y']].max()
t[['x','y']].idxmax()
```

```
t[['x','y']].min()
t[['x','y']].idxmin()

for counting the number of counts in column,
t['name_of_column'].value_counts()

replacing in the column,
t['name_of_column'].replace('element_to_be_replaced','new_element')
for multiple,
t['name_of_column'].replace(['element1','element2'],['new_element1','new_element2'])
OR
new = {'element1':'new_element1','element2':'new_element2'}
t['name_of_column'].map(new)



to remove complete duplicate rows,
t.drop_duplicates()
```

▼ Missing Data

```
NAN = not a number, called by np.nan
import pandas as pd
import numpy as np
data = {'Number': [0,1, 2, 3, 4],
        'Age': [24, 27, np.nan, 32, 29],
        'Class': [np.nan, 'Second', 'Third', 'Fourth', 'Fifth'],
        'Score': [85, 90, 78, np.nan, 92]}

Name=['P1', 'P2', 'P3', 'P4', 'P5']
df = pd.DataFrame(data,Name)

to check for nan,
df.isnull()  will return boolean value
```

```
df.notnull()

to look for only nan or not nan,
df[ df['Age'].notnull()]
df[ df['Age'].isnull()]

for mulitple,
df[ df['Age'].notnull()  &  df['Class'].notnull() ]
df[ df['Age'].isnull()   &  df['Class'].isnull()  ]



to remove all rows with nan,
df.dropna()
to remove columns with nan,
df.dropna(axis=1)

to drop specific rows with nan,
df.dropna(subset=['name_of_column'])



to fill data in columns with nan,
df = df.fillna(data)
eg.,
df = df.fillna(df['Score'].mean()) this will fill all nan values with the mea
n

to fill data in specific columns with nan,
df['column_name']=df['column_name'].fillna(data)
```

▼ GroupBy Operations

```
import pandas as pd
import numpy as np

data = {'Number': [0,1, 2, 3, 4],
```

```
        'Age': [24, 27, np.nan, 32, 29],
        'Class': [np.nan, 'Second', 'Third', 'Fourth', 'Fifth'],
        'Score': [85, 90, 78, np.nan, 92]}

Name = ['P1', 'P2', 'P3', 'P4', 'P5']
df = pd.DataFrame(data,Name)

df.groupby('column_name').mean()
                        .sum(), etc can be used
this will make particular mean/sum/etc of every data related to an ele
ment in the 'column_name'

df.groupby(['column1','column2']).mean() for multiple
```

▼ Combining DataFrames

▼ Concatenation

```
import pandas as pd
import numpy as np

data1 = {'Name': ['P1', 'P2'],
        'Age': [25, 30]}
df1 = pd.DataFrame(data1)

data2 = {'Name': ['P3', 'P4'],
        'Age': [35, 40]}
df2 = pd.DataFrame(data2)



df=pd.concat([df1,df2], axis=1) this will add df2's columns to df1's
column [order matters]
```

▼ Merging

```
import pandas as pd
import numpy as np

data1 = {'Name': ['P1', 'P2', 'P5'],
        'Age1': [20,25,30]}
df1 = pd.DataFrame(data1)

data2 = {'Name': ['P2', 'P3', 'P4', 'P5'],
        'Age2': [35,40,45,50]}
df2 = pd.DataFrame(data2)

**inner merging**
df=pd.merge(df1,df2,how= 'inner',on='Name')
there should be same element in column 'Name'
output will be,
  Name  Age1  Age2
0   P2    25    35
1   P5    30    50


**left merging**
df=pd.merge(left=df1,right=df2,how='left',on='Name')
there will be no element of right column in the output column, rath
er than common elements,
  Name  Age1  Age2
0   P1    20   NaN
1   P2    25  30.0
2   P5    30  50.0


**right merging**
df=pd.merge(left=df1,right=df2,how='right',on='Name')
there will be no element of left column in the output column, rather
than common elements,
```

```
  Name  Age1  Age2
0   P2  25.0    30
1   P3   NaN    40
2   P4   NaN    45
3   P5  30.0    50

**outer merging**
df=pd.merge(df1,df2,how= 'outer',on='Name')
there will be elements of both, order matters,
  Name  Age1  Age2
0   P1  20.0   NaN
1   P2  25.0  30.0
2   P3   NaN  40.0
3   P4   NaN  45.0
4   P5  30.0  50.0
```

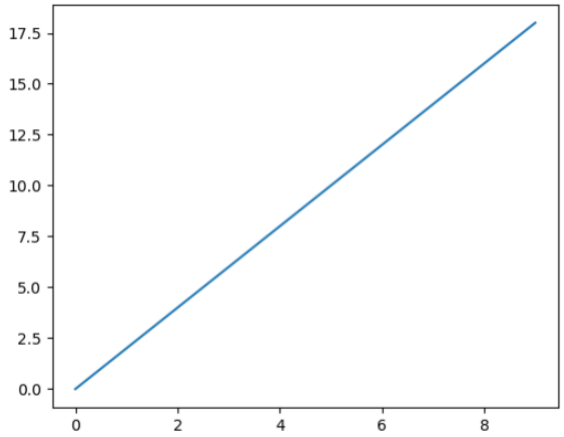
▼ String Methods

▼ Time Methods for Date and Time Data

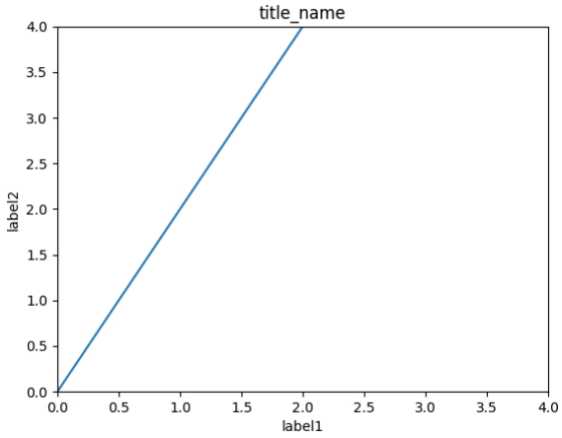
▼ **Matplotlib**

used for arrays

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

x = np.arange(0,10)
y=x*2

plt.plot(x,y)
print(plt.show())
output will be,
```

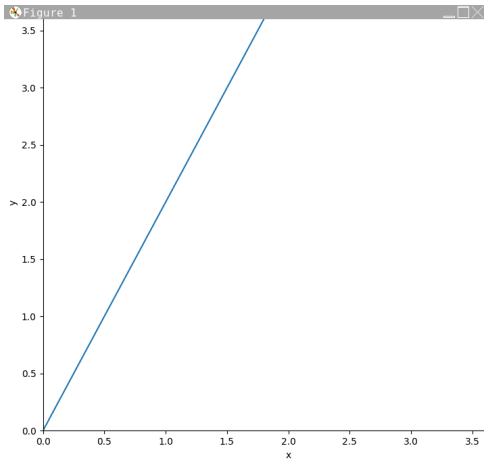



```
plt.title('title_name')  this will give title to graph

plt.xlabel('label1')  this will give label to x axis
plt.ylabel('label2')  this is for y axis
plt.xlim(0,4)   this will set a limit till 0 to 4 on x axis
plt.ylim(0,4)   same for y axis

plt.savefig('test.png')  this will save file
```




```
x = np.arange(0,10)
y=x*2

fig = plt.figure()
```

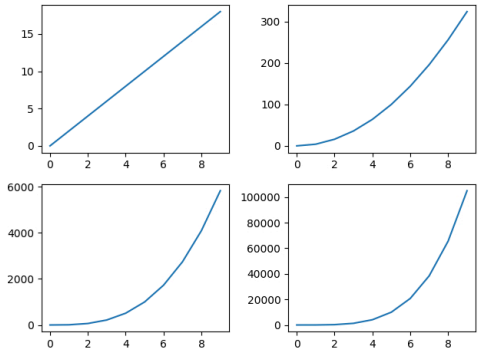
```
ax = fig.add_axes([0.1,     0.1,        1,          1])
                        [origin_x, origin_y, size_xaxis, size_yaxis]
ax.plot(x,y)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('title')
ax.set_xlim(0,4)
ax.set_ylim(0,4)
plt.show()
```



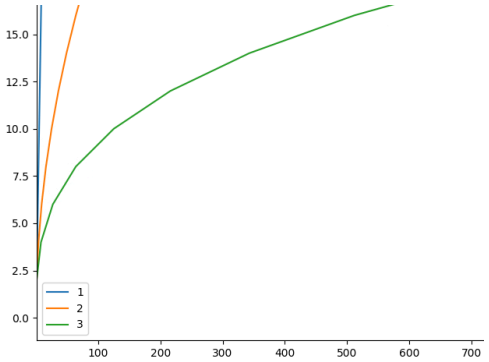

```
x = np.arange(0,10)
y=x*2

fig,axe = plt.subplots(nrows=2,ncols=2)
axe[0][0].plot(x,y)
axe[0][1].plot(x,y*y)
axe[1][0].plot(x,y*y*y)
axe[1][1].plot(x,y*y*y*y)
fig.tight_layout()
plt.show()
```

fig,axe are variables,

```
a,b = plt.subplots(nrows=2,ncols=2
b[0][1].plot(x,y)
a.tight_layout()
```



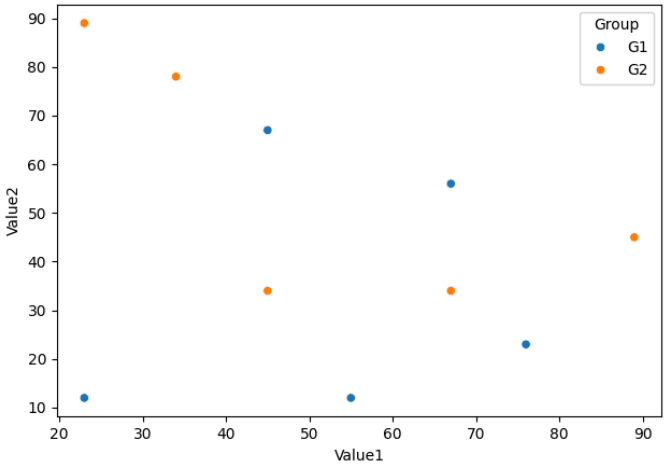

```
x = np.arange(0,10)
y=x*2

fig = plt.figure()
ax = fig.add_axes([0.1, 0.1, 1, 1])
ax.plot(x,y,label='1',color='rgb_hexcode')
ax.plot(x*x,y,label='2',color='color_name')
ax.plot(x*x*x,y,label='3',lw=int, ls='pattern_of_line')
                                    linewidth, linespacing = '- - -' or '-- --', etc
ax.legend()
or
ax.legend(loc='best')
or
ax.legend(loc=(x_coord,y_coord))
plt.show()
```



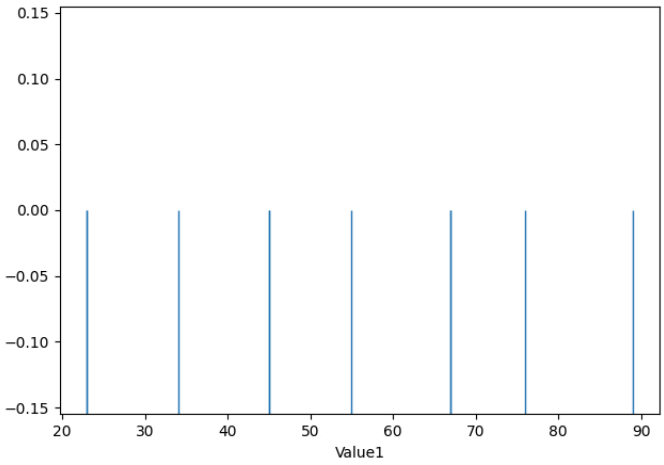
▼ Seaborn

used for dataframes

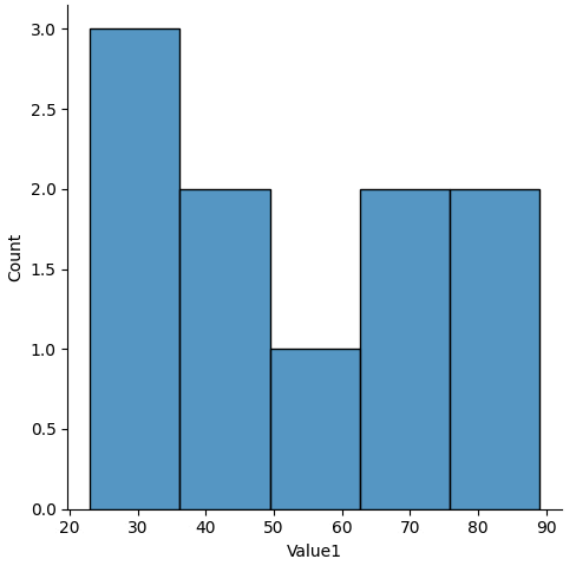
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

data = {
    "Category": ["A", "B", "C", "D", "A", "B", "C", "D", "A", "B"],
    "Subcategory": ["X", "Y", "Z", "X", "Y", "Z", "X", "Y", "Z", "X"],
    "Value1": [23, 45, 67, 34, 76, 89, 45, 23, 55, 67],
    "Value2": [12, 34, 56, 78, 23, 45, 67, 89, 12, 34],
    "Group": ["G1", "G2", "G1", "G2", "G1", "G2", "G1", "G2", "G1", "G2"],
    "Score": [88, 76, 92, 81, 73, 85, 95, 77, 66, 84],
}

df = pd.DataFrame(data)

sns.scatterplot(x="Value1", y="Value2", hue="Group", data=df)
plt.show()
```



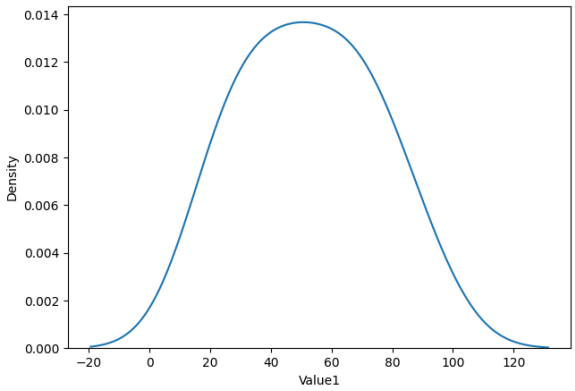

```
sns.rugplot(x='Value1',height=0.5, data=df)
plt.show()
```



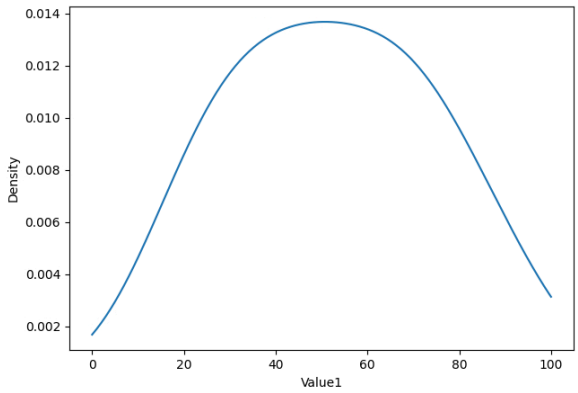

```
sns.displot(x='Value1', data=df)
plt.show()
```



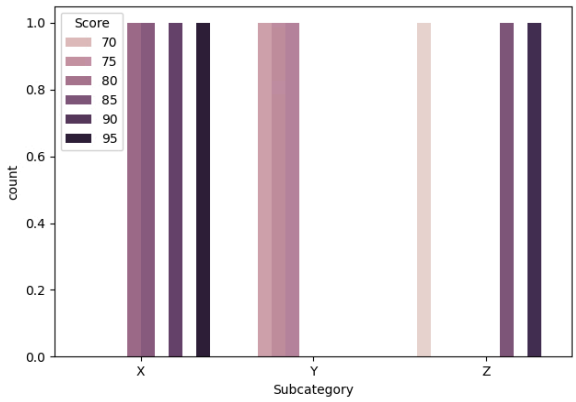

```
sns.kdeplot(x='Value1', data=df)
plt.show()
```



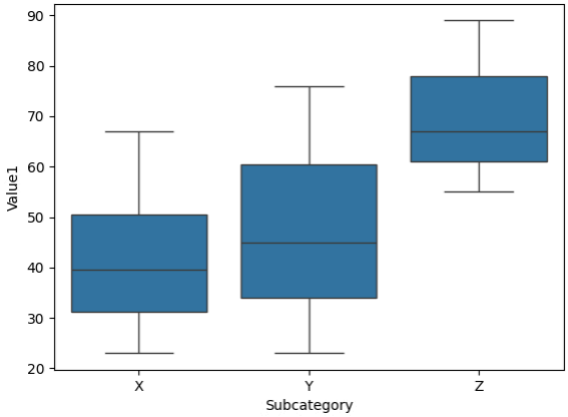

```
sns.kdeplot(x='Value1', data=df, clip=[0,100])
plt.show()
```



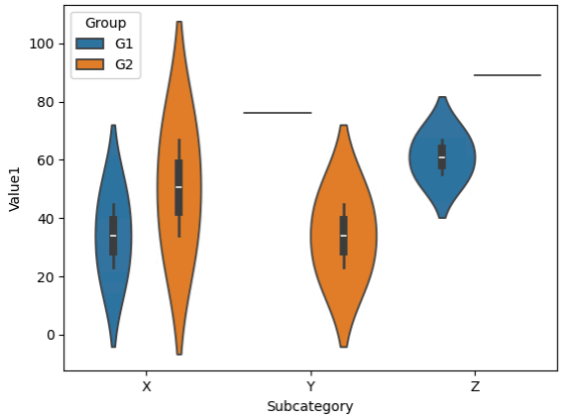

```
sns.countplot(x="Subcategory", hue='Score', data=df)
plt.show()
```



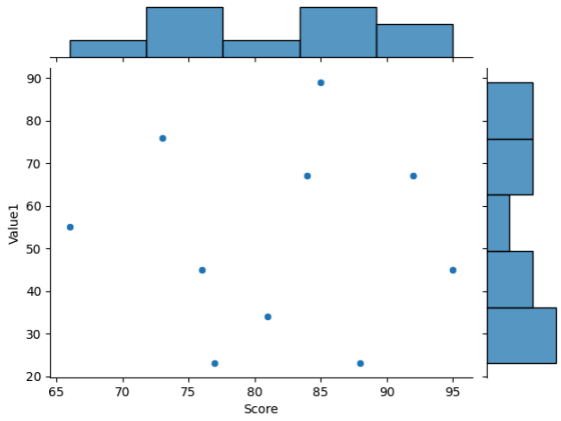

```
sns.boxplot(x="Subcategory", y='Value1', data=df)
plt.show()
```



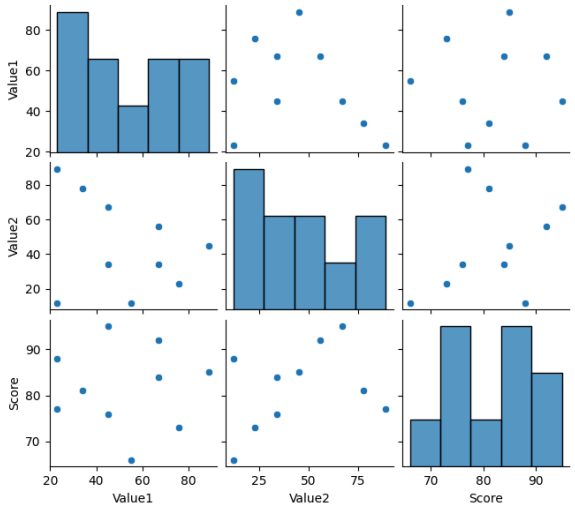

```
sns.violinplot(x="Subcategory", y='Value1', data=df, hue='Group')
plt.show()
```




```
sns.jointplot(x="Score",y='Value1', data=df)
plt.show()
```

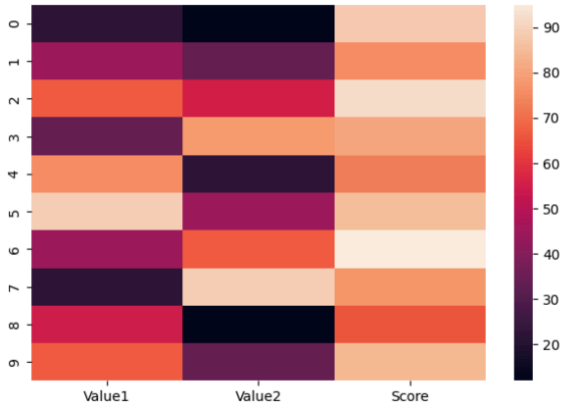



```
sns.pairplot(data=df)
plt.show()
```




```
data = {
    "Value1": [23, 45, 67, 34, 76, 89, 45, 23, 55, 67],
    "Value2": [12, 34, 56, 78, 23, 45, 67, 89, 12, 34],
    "Score": [88, 76, 92, 81, 73, 85, 95, 77, 66, 84],
}

df = pd.DataFrame(data)
```

```
sns.heatmap(df)
plt.show()
```



▼ SUPERVISED LEARNING

Advertising.csv

▼ Linear Regression

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

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

df=pd.read_csv("Advertising.csv")

X=df.drop('sales',axis=1)
y=df['sales']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=101)
this is the format for TRAIN AND TEST SPLIT
30% part of total data should be for TEST and 70% for TRAINING i.e.test_size=0.3


creating a model(estimator)
from sklearn.linear_model import LinearRegression
model=LinearRegression()

Fit/TRAIN the model on trainig data
```

```
model.fit(X_train,y_train)

Calculate Performance on Test Set
test_predictions = model.predict(X_test)
now, test_predictions will give the predicted y values for testing data o
f X

model.coef_  ← this will give values,
[0.04469599  0.1875657  -0.00032275]
    TV       radio      newspaper

when 1 unit is added in TV, keeping radio and newspaper as it is, the th
ere will be an increase of 0.04469599 in sales

eg=pd.DataFrame([[230, 44, 10]], columns=X.columns)
model.predict(eg)

from joblib import dump, load
dump(model, 'model.joblib')
dump will save the model

model = load('model.joblib')
this will load the model from other file
```

▼ Polynomial Regression

For higher degree polynomial

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,root_mean_squared_
error, mean_absolute_error
```

```
df = pd.read_csv("Advertising.csv")

X = df.drop('sales', axis=1)
y = df['sales']

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)

X_poly = poly.fit(X)
X_poly_trans = X_poly.transform(X)
print(X_poly_trans[0])
or
X_poly = poly.fit_transform(X)
X_poly will give 3**2=9 values, [col1, col2, col3, col1*col2, col2*col3,
col3*col1, col1**2, col2**2, col3**2]

X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3,
random_state=101)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

▼ Regularization

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

df = pd.read_csv("Advertising.csv")
X = df.drop('sales',axis=1)
y = df['sales']
```

```
#Polynomial Conversion
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

#Train | Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(poly_features, y, test_siz
e=0.3, random_state=101)

#Scalling data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#RIDGE REGRESSION (L2 REGRESSION)
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha=10)
ridge_model.fit(X_train,y_train)
test_predictions = ridge_model.predict(X_test)

#Choosing an alpha value with Cross-Validation
from sklearn.linear_model import RidgeCV
ridge_cv_model = RidgeCV(alphas=(0.1, 1.0, 10.0),scoring='neg_mean_
absolute_error')


#LASSO REGRESSION (L1 REGRESSION)
from sklearn.linear_model import LassoCV
lasso_cv_model = LassoCV(eps=0.1,n_alphas=100,cv=5)


#ELASTIC NET (mix of L1&L2)
from sklearn.linear_model import ElasticNetCV
```

```
elastic_model = ElasticNetCV(l1_ratio=[.1, .5, .7,.9, .95, .99, 1],tol=0.01)
l1_ratio=0: Equivalent to pure L2 (Ridge) regularization
l1_ratio=1: Equivalent to pure L1 (Lasso) regularization
0<l1_ratio<1: A combination of L1 and L2 regularization (Elastic Net)
```

▼ Cross Validation

▼ Train | Test Split Procedure

1. Split Data in Train/Test for both X and y
2. Fit/Train Scaler on Training X Data
3. Scale X Test Data
4. Create Model
5. Fit/Train Model on X Train Data
6. Evaluate Model on X Test Data (by creating predictions and comparing to Y_test)
7. Adjust Parameters as Necessary and repeat steps 5 and 6

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

df = pd.read_csv("Advertising.csv")
X = df.drop('sales',axis=1)
y = df['sales']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

#scaling
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#creating model
from sklearn.linear_model import Ridge
model = Ridge(alpha=100) random value of alpha
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

#evaluation
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)

#the error mean is too high, changing the alpha,
#re-evaluation
model = Ridge(alpha=1)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

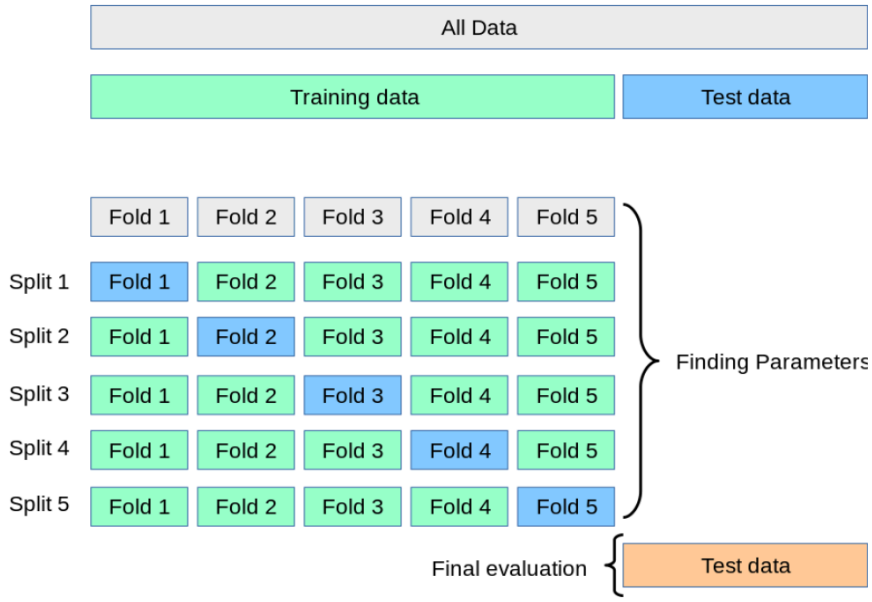
▼ Train | Validation | Test Split Procedure

1. Split Data in Train/Validation/Test for both X and y
2. Fit/Train Scaler on Training X Data
3. Scale X Eval Data
4. Create Model
5. Fit/Train Model on X Train Data
6. Evaluate Model on X Evaluation Data (by creating predictions and comparing to Y_eval)
7. Adjust Parameters as Necessary and repeat steps 5 and 6
8. Get final metrics on Test set (not allowed to go back and adjust after this!)

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

df = pd.read_csv("Advertising.csv")
X = df.drop('sales',axis=1)
y = df['sales']

from sklearn.model_selection import train_test_split
# 70% of data is training data, set aside other 30%
X_train, X_OTHER, y_train, y_OTHER = train_test_split(X, y, test_size
=0.3, random_state=101)

# Remaining 30% is split into evaluation and test sets
# Each is 15% of the original data size
X_eval, X_test, y_eval, y_test = train_test_split(X_OTHER, y_OTHER,
test_size=0.5, random_state=101)

#scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_eval = scaler.transform(X_eval)
X_test = scaler.transform(X_test)

#creating model
from sklearn.linear_model import Ridge
model = Ridge(alpha=100)
model.fit(X_train,y_train)
y_eval_pred = model.predict(X_eval)

#evaluation
from sklearn.metrics import mean_squared_error
```

```
mean_squared_error(y_eval,y_eval_pred)

#Re-evaluation
model = Ridge(alpha=1)
model.fit(X_train,y_train)
y_eval_pred = model.predict(X_eval)

#final evaluation
y_final_test_pred = model.predict(X_test)
```

▼ cross_val_score




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

df = pd.read_csv("Advertising.csv")
X = df.drop('sales',axis=1)
y = df['sales']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

#scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#creating model
from sklearn.linear_model import Ridge
model = Ridge(alpha=100)

#cross_val_score
from sklearn.model_selection import cross_val_score
scores = cross_val_score( model , X_train , y_train ,
                                    scoring='neg_mean_squared_error',
cv=5)
                                    cv is number of folds
scores will give 5 errors as there are 5 folds in training dataset

abs(scores.mean()) mean is not satisfying

#adjusting model
model2 = Ridge(alpha=1)
scores = cross_val_score(model2 , X_train , y_train ,
                                   scoring='neg_mean_squared_error' ,
cv=5)
abs(scores.mean())

#final evaluation
model2.fit(X_train,y_train)
y_final_test_pred = model.predict(X_test)
```

▼ cross_validate

The cross_validate function differs from cross_val_score in two ways:

It allows specifying multiple metrics for evaluation.

It returns a dict containing fit-times, score-times (and optionally training scores as well as fitted estimators) in addition to the test score.

For single metric evaluation, where the scoring parameter is a string, callable or None, the keys will be: ['test_score', 'fit_time', 'score_time']

And for multiple metric evaluation, the return value is a dict with the following keys:

['test_<scorer1_name>', 'test_<scorer2_name>', 'test_<scorer...>', 'fit_time', 'score_time']

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

df = pd.read_csv("Advertising.csv")
X = df.drop('sales',axis=1)
y = df['sales']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

#scaling
4from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#creating model
from sklearn.linear_model import Ridge
```

```
model = Ridge(alpha=100)

#cross_validate
from sklearn.model_selection import cross_validate
scores = cross_validate(model,X_train,y_train,
                        scoring=['neg_mean_absolute_error','neg_mean_sq
uared_error','max_error'],
                        cv=5)
pd.DataFrame(scores).mean()

#adjusting model
model2 = Ridge(alpha=1)
scores = cross_validate(model2,X_train,y_train,
                        scoring=['neg_mean_absolute_error','neg_mean_sq
uared_error','max_error'],
                        cv=5)

#final evaluation
model2.fit(X_train,y_train)
y_final_test_pred = model.predict(X_test)
```

▼ Grid Search

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

df = pd.read_csv("../DATA/Advertising.csv")
X=df.drop('sales',axis=1)
y=df['sales']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=101)

#scaling data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#creating model
from sklearn.linear_model import ElasticNet
base_elastic_model = ElasticNet()

A grid search consists of:
   an estimator (regressor or classifier such as sklearn.svm.SVC());
   a parameter space;
   a method for searching or sampling candidates;
   a cross-validation scheme
   a score function.

param = {'alpha':[0.1,1,5,10,50,100],'l1_ratio':[.1, .5, .7, .9, .95, .99, 1]}

from sklearn.model_selection import GridSearchCV
grid_model = GridSearchCV(estimator=base_elastic_model,
                          param_grid=param,
                          scoring='neg_mean_squared_error',
                          cv=5,
                          verbose=2)

grid_model.fit(X_train,y_train)
grid_model.best_estimator_
grid_model.best_params_
```

```
#Using Best Model From Grid Search
y_pred = grid_model.predict(X_test)
```

▼ Logistic Regression

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

df = pd.read_csv("../DATA/Advertising.csv")
X=df.drop('sales',axis=1)
y=df['sales']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=101)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Logistic Regression Model
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
log_model.fit(scaled_X_train,y_train)

log_model.coef_
will give 2 values, -0.94953524,  3.45991194
A positive coefficient means that an increase in the corresponding feat
ure increases the log-odds (and thus the probability) of the target bein
g 1
A negative coefficient means that an increase in the corresponding fe
```

```
ature decreases the log-odds (and thus the probability) of the target b
eing 1.

from sklearn.metrics import accuracy_score,confusion_matrix,classific
ation_report,plot_confusion_matrix
y_pred = log_model.predict(scaled_X_test)
accuracy_score(y_test,y_pred)
confusion_matrix(y_test,y_pred)
(classification_report(y_test,y_pred)
```

Multi-Class Logistic Regression Model

iris.csv

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

df = pd.read_csv("iris.csv")
X = df.drop('species',axis=1)
y = df['species']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=101)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

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

log_model = LogisticRegression(solver='saga',multi_class="ovr",max_iter=5000)

penalty = ['l1', 'l2']

# Use logarithmically spaced C values (recommended in official docs)
C = np.logspace(0, 4, 10)

grid_model = GridSearchCV(log_model,param_grid={'C':C,'penalty':penalty})

grid_model.fit(scaled_X_train,y_train)
y_pred = grid_model.predict(scaled_X_test)
```

▼ KNN (K Nearest Neighbor)

	Gene One	Gene Two	Cancer Present
0	4.3	3.9	1
1	2.5	6.3	0
2	5.7	3.9	1
3	6.1	6.2	0
4	7.4	3.4	1

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

df = pd.read_csv('gene_expression.csv')
X = df.drop('Cancer Present',axis=1)
y = df['Cancer Present']
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=42)

scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(scaled_X_train,y_train)
```

Elbow Method for Choosing Reasonable K Values

```
test_error_rates = []


for k in range(1,30):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(scaled_X_train,y_train)

    y_pred_test = knn_model.predict(scaled_X_test)

    test_error = 1 - accuracy_score(y_test,y_pred_test)
    test_error_rates.append(test_error)
```

Full Cross Validation Grid Search for K Value

```
scaler = StandardScaler()
knn = KNeighborsClassifier()
# Highly recommend string code matches variable name!
operations = [('scaler',scaler),('knn',knn)]
```

```
#Creating a Pipeline to find K value
from sklearn.pipeline import Pipeline
pipe = Pipeline(operations)

If your parameter grid is going inside a PipeLine, your parameter name
needs to be specified in the following manner:

chosen_string_name + two underscores + parameter key name
⇒ model_name + __ + parameter name
⇒ knn_model + __ + n_neighbors
⇒ knn__n_neighbors

k_values = list(range(1,20))

param_grid = {'knn__n_neighbors': k_values}
full_cv_classifier = GridSearchCV(pipe,param_grid,cv=5,scoring='accu
racy')
full_cv_classifier.fit(X_train,y_train)

#Final Model
scaler = StandardScaler()
knn14 = KNeighborsClassifier(n_neighbors=14)
operations = [('scaler',scaler),('knn14',knn14)]
pipe = Pipeline(operations)
pipe.fit(X_train,y_train)
pipe_pred = pipe.predict(X_test)
```

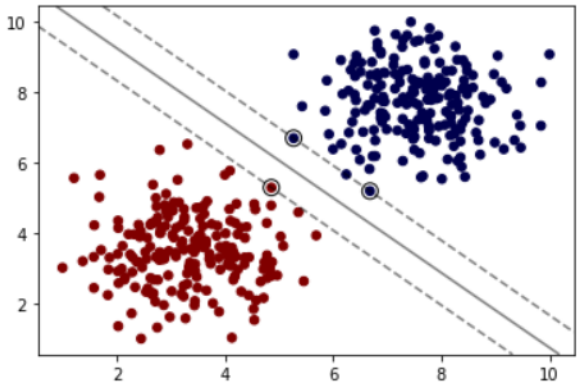
▼ SVM (Support Vector Machine)

mouse_viral_study.csv

```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("../DATA/mouse_viral_study.csv")
y = df['Virus Present']
X = df.drop('Virus Present',axis=1)
```

```
from sklearn.svm import SVC # Supprt Vector Classifier
model = SVC(kernel='linear', C=1000)
model.fit(X, y)
```

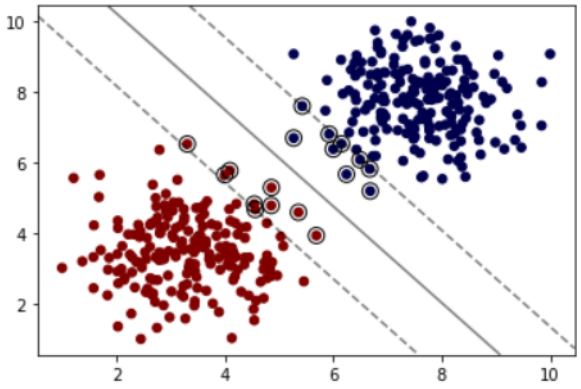


▼ Hyper Parameters:

▼ **C**

Regularization parameter. The strength of the regularization is **inversely** proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

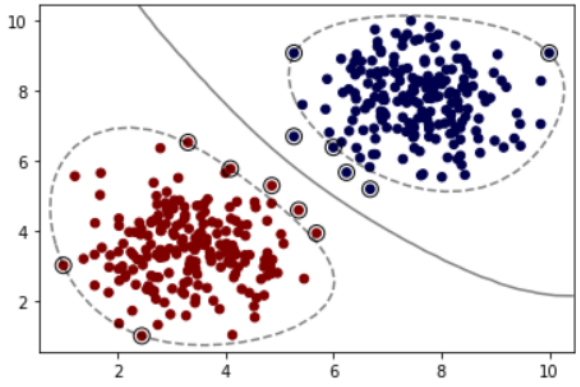
```
model = SVC(kernel='linear', C=0.05)
model.fit(X, y)
```



- Kernel
  - rbf - Radial Basis Function:

    When training an SVM with the Radial Basis Function (RBF) kernel, two parameters must be considered: C and gamma. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.

    ```
    model = SVC(kernel='rbf', C=1)
    model.fit(X, y)
    plot_svm_boundary(model,X,y)
    ```

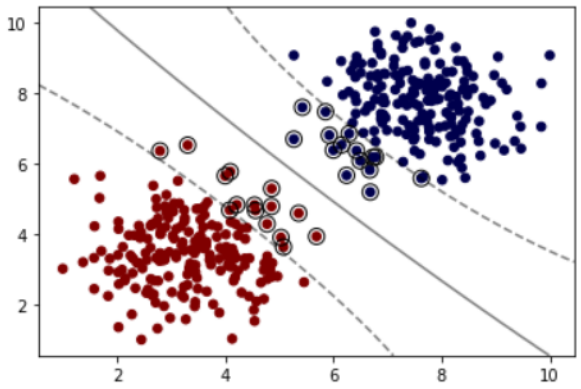


  - gamma

gamma : {'scale', 'auto'} or float, default='scale' Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

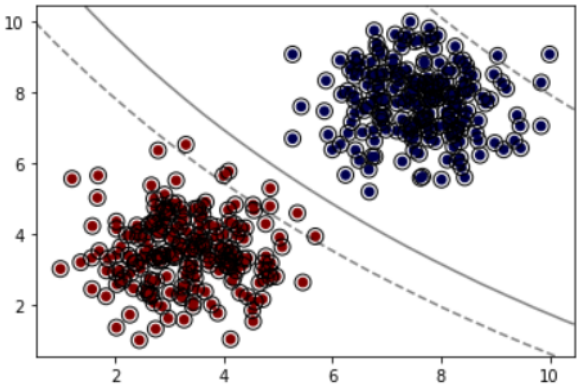
- if ``gamma='scale'`` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

```
model = SVC(kernel='rbf', C=1,gamma=0.01)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



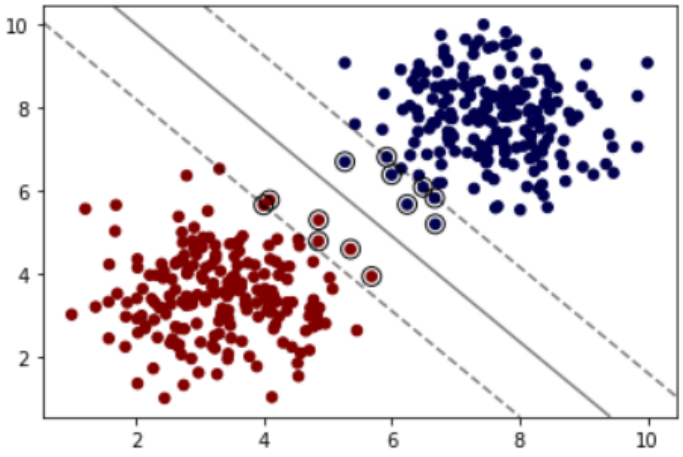
▼ sigmoid

```
model = SVC(kernel='sigmoid')
model.fit(X, y)
plot_svm_boundary(model,X,y)
```

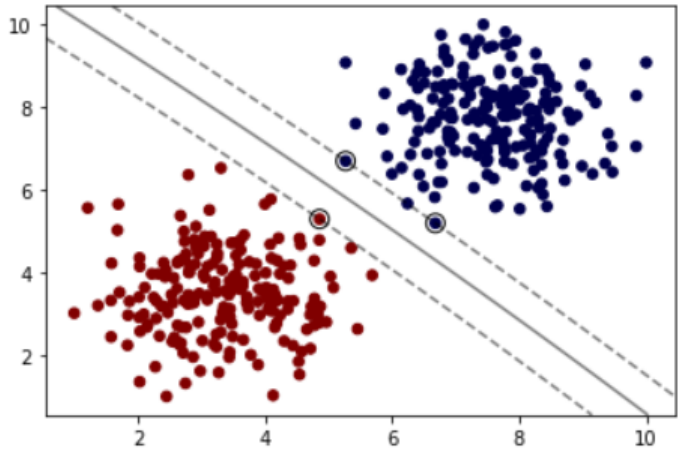


▼ Degree (poly kernels only)

```
model = SVC(kernel='poly', C=1,degree=1)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```




```
model = SVC(kernel='poly', C=1,degree=2)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



▼ Grid Search

```
from sklearn.model_selection import GridSearchCV

svm = SVC()
param_grid = {'C':[0.01,0.1,1],'kernel':['linear','rbf']}
grid = GridSearchCV(svm,param_grid)

# Note again we didn't split Train|Test
grid.fit(X,y)

grid.best_params_
```

▼ SVM-Regression

cement_slump.csv

```
df = pd.read_csv('../DATA/cement_slump.csv')
X = df.drop('Compressive Strength (28-day)(Mpa)',axis=1)
y = df['Compressive Strength (28-day)(Mpa)']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)

#SVM-Regression
from sklearn.svm import SVR,LinearSVR
base_model = SVR()
```

```
base_model.fit(scaled_X_train,y_train)
base_preds = base_model.predict(scaled_X_test)

#Grid Search in Attempt for Better Model
param_grid = {'C':[0.001,0.01,0.1,0.5,1],
              'kernel':['linear','rbf','poly'],
              'gamma':['scale','auto'],
              'degree':[2,3,4],
              'epsilon':[0,0.01,0.1,0.5,1,2]}

from sklearn.model_selection import GridSearchCV
svr = SVR()
grid = GridSearchCV(svr,param_grid=param_grid)
grid.fit(scaled_X_train,y_train)
grid.best_params_
#{'C': 1, 'degree': 2, 'epsilon': 2, 'gamma': 'scale', 'kernel': 'linear'}
grid_preds = grid.predict(scaled_X_test)
```

penguins_size.csv

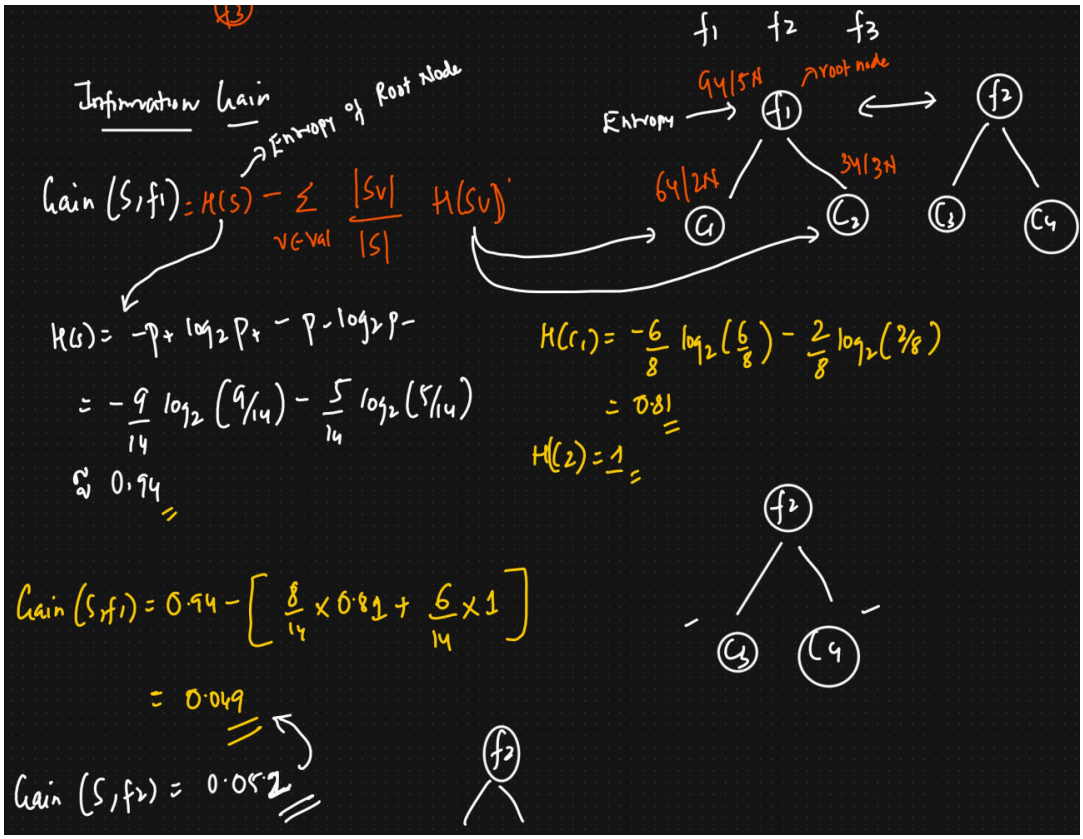
▼ Decision Tree

Entropy, $H(S) = -p^{+} \log(p^{+}) - p^{-} \log(p^{-})$

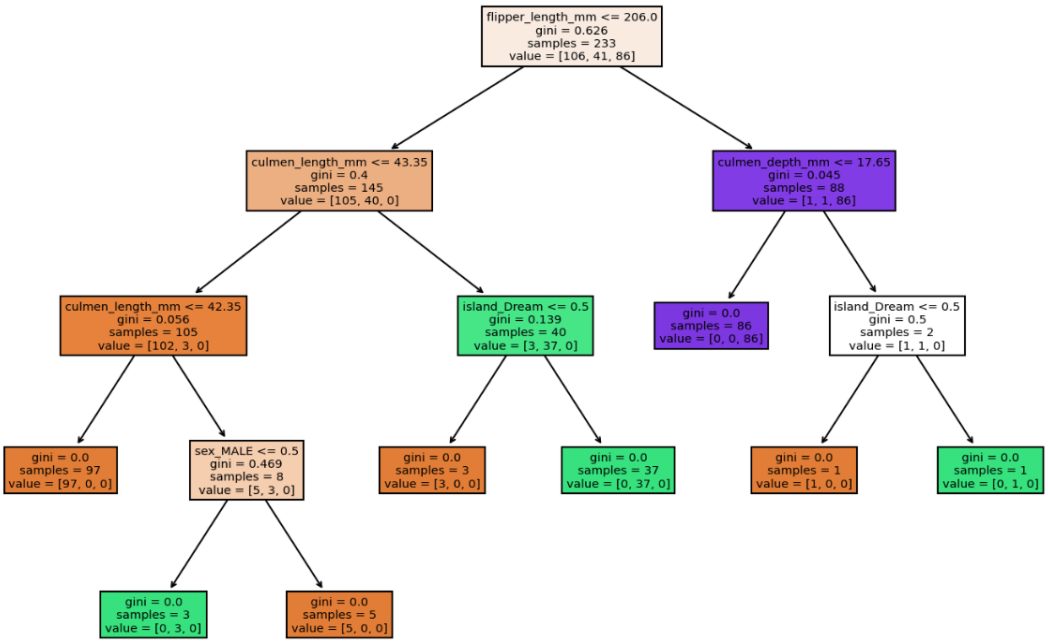
Gini Impurity, $G = \sum p(1-p)$

Information Gain,

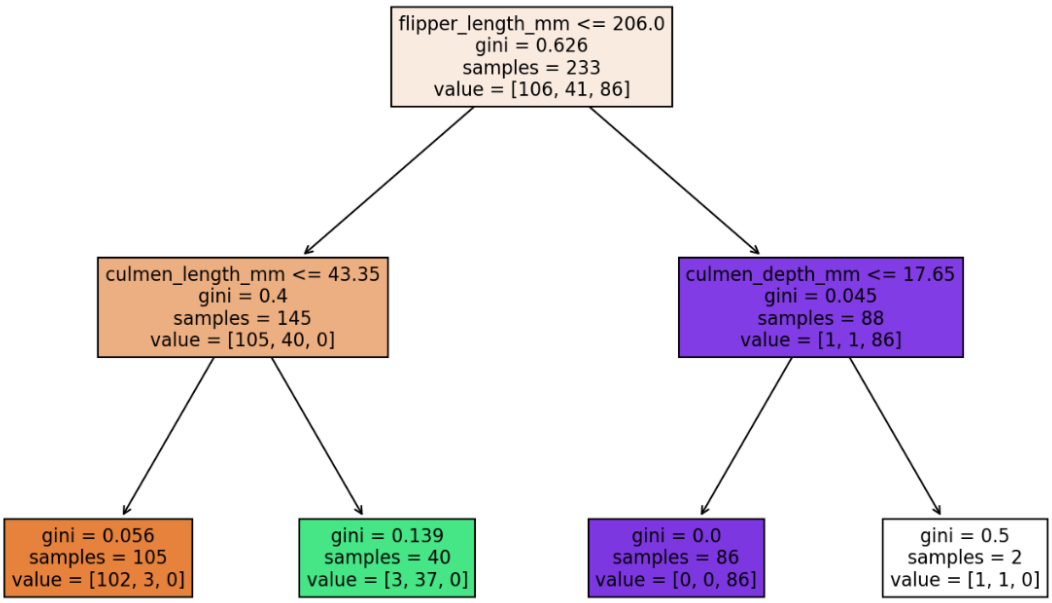
gain of f_2 is greater than gain of f_1, so f_2 should be used.



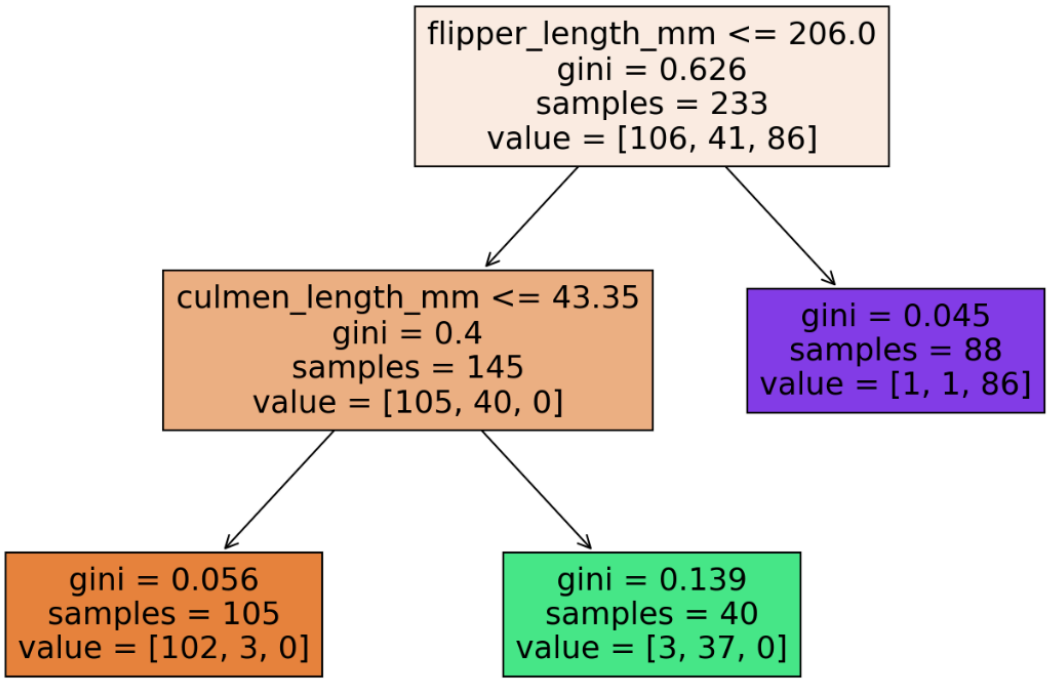

```
df = pd.read_csv("../DATA/penguins_size.csv")
X = pd.get_dummies(df.drop('species',axis=1),drop_first=True)
y = df['species']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train,y_train)
base_pred = model.predict(X_test)

#Visualize the Tree
from sklearn.tree import plot_tree
plot_tree(model,filled=True,feature_names=X.columns);
```



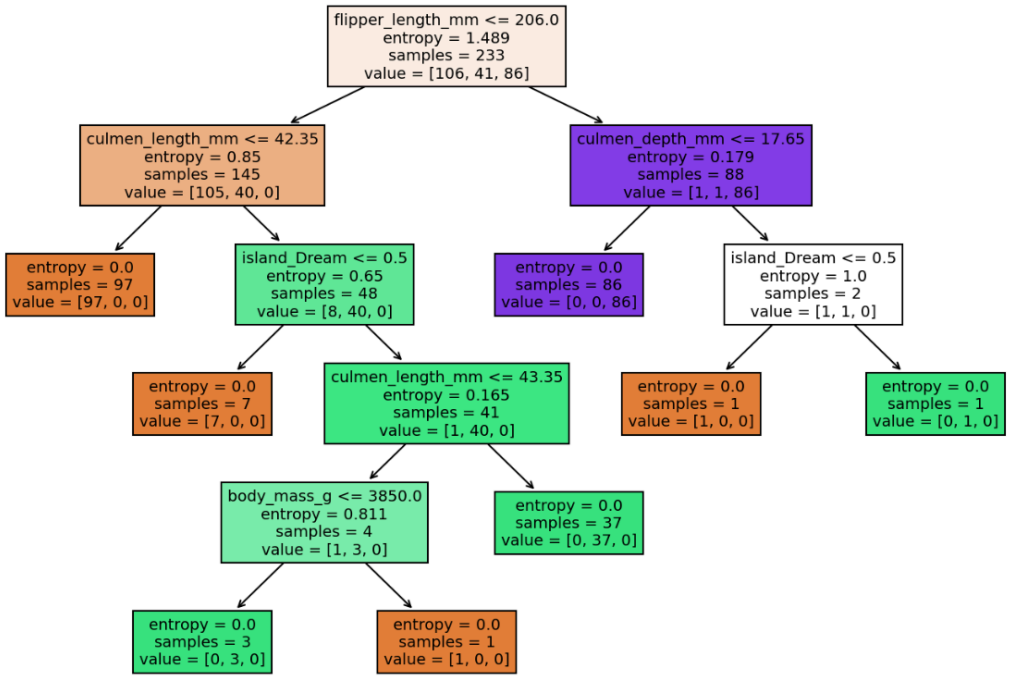

```
#Max Depth
from sklearn.metrics import confusion_matrix,classification_repor
t,plot_confusion_matrix
def report_model(model):
    model_preds = model.predict(X_test)
    print(classification_report(y_test,model_preds))
    print('\n')
    plt.figure(figsize=(12,8),dpi=150)
    plot_tree(model,filled=True,feature_names=X.columns);

pruned_tree = DecisionTreeClassifier(max_depth=2)
pruned_tree.fit(X_train,y_train)

report_model(pruned_tree)
```




```
#Max Leaf Node
pruned_tree = DecisionTreeClassifier(max_leaf_nodes=3)
pruned_tree.fit(X_train,y_train)

report_model(pruned_tree)
```



```
#Criterion
entropy_tree = DecisionTreeClassifier(criterion='entropy')
entropy_tree.fit(X_train,y_train)

report_model(entropy_tree)
```



▼ Random Forest

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

df = pd.read_csv("../DATA/penguins_size.csv")
df = df.dropna()
```

```
X = pd.get_dummies(df.drop('species',axis=1),drop_first=True)
y = df['species']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=101)

from sklearn.ensemble import RandomForestClassifier
# Use 10 random trees
model = RandomForestClassifier(n_estimators=10,max_features='a
uto',random_state=101)
model.fit(X_train,y_train)
preds = model.predict(X_test)
```

▼ HyperParameter Exploration

data_banknote_authentication.csv

```
df = pd.read_csv("../DATA/data_banknote_authentication.csv")
X = df.drop("Class",axis=1)
y = df["Class"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.15, random_state=101)

from sklearn.model_selection import GridSearchCV
n_estimators=[64,100,128,200]
max_features= [2,3,4]
bootstrap = [True,False]
oob_score = [True,False]

param_grid = {'n_estimators':n_estimators,
```

```
            'max_features':max_features,
            'bootstrap':bootstrap,
            'oob_score':oob_score}  # Note, oob_score only makes
sense when bootstrap=True!

rfc = RandomForestClassifier()
grid = GridSearchCV(rfc,param_grid)
grid.fit(X_train,y_train)
grid.best_params_
predictions = grid.predict(X_test)

print(classification_report(y_test,predictions))
```

▼ Understanding Number of Estimators (Trees)

```
from sklearn.metrics import accuracy_score

errors = []
misclassifications = []

for n in range(1,64):
    rfc = RandomForestClassifier( n_estimators=n,bootstrap=Tru
e,max_features= 2)
    rfc.fit(X_train,y_train)
    preds = rfc.predict(X_test)
    err = 1 - accuracy_score(preds,y_test)
    n_missed = np.sum(preds != y_test)
    errors.append(err)
    misclassifications.append(n_missed)

plt.plot(range(1,64),errors)
```

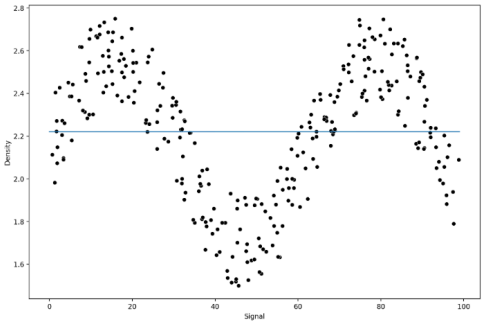
▼ Regression

rock_density_xray.csv

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

df = pd.read_csv("../DATA/rock_density_xray.csv")
X = df['Signal'].values.reshape(-1,1)
y = df['Density']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.1, random_state=101)
```

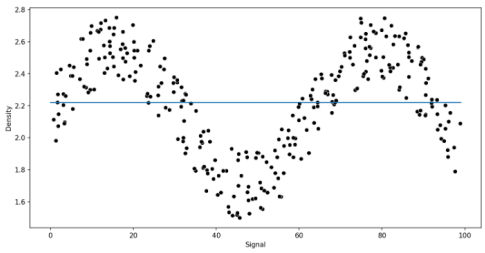
▼ Linear Regression

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)
lr_preds = lr_model.predict(X_test)

from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,lr_preds))
#error may be will very small, but if test_data is plotted on
graph it would not be accurate
```



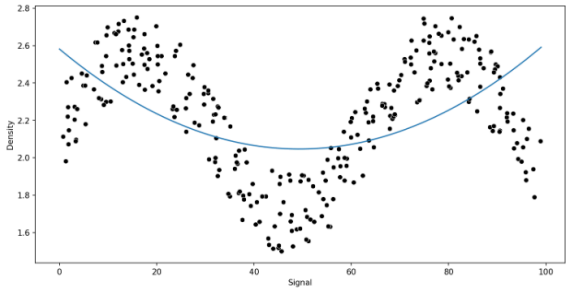
▼ Polynomial Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()

def run_model(model,X_train,y_train,X_test,y_test):

    # Fit Model
    model.fit(X_train,y_train)

    # Get Metrics

    preds = model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(y_test,preds))
    print(f'RMSE : {rmse}')

    # Plot results
    signal_range = np.arange(0,100)
    output = model.predict(signal_range.reshape(-1,1))


    plt.figure(figsize=(12,6),dpi=150)
    sns.scatterplot(x='Signal',y='Density',data=df,color='bla
ck')
    plt.plot(signal_range,output)

    run_model(model,X_train,y_train,X_test,y_test)
```

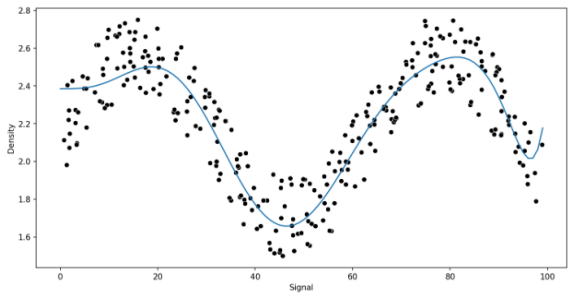


Pipeline for Poly Orders

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

pipe = make_pipeline(PolynomialFeatures(2),LinearRegres
sion())
run_model(pipe,X_train,y_train,X_test,y_test)
```

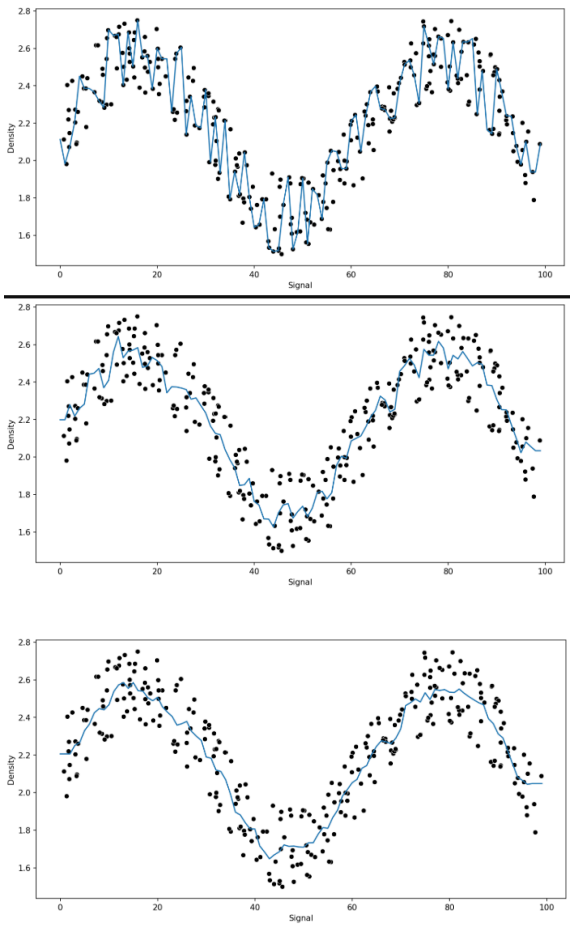



```
pipe = make_pipeline(PolynomialFeatures(10),LinearRegres
sion())
run_model(pipe,X_train,y_train,X_test,y_test)
```



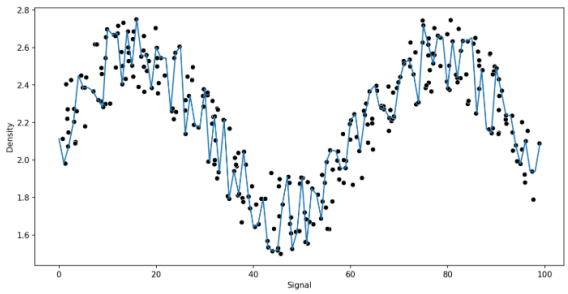
▼ KNN Regression

```
from sklearn.neighbors import KNeighborsRegressor
preds = {}
k_values = [1,5,10]
for n in k_values:


    model = KNeighborsRegressor(n_neighbors=n)
    run_model(model,X_train,y_train,X_test,y_test)
```



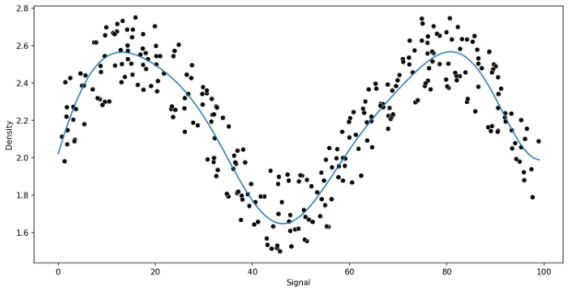
▼ Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()

run_model(model,X_train,y_train,X_test,y_test)
```



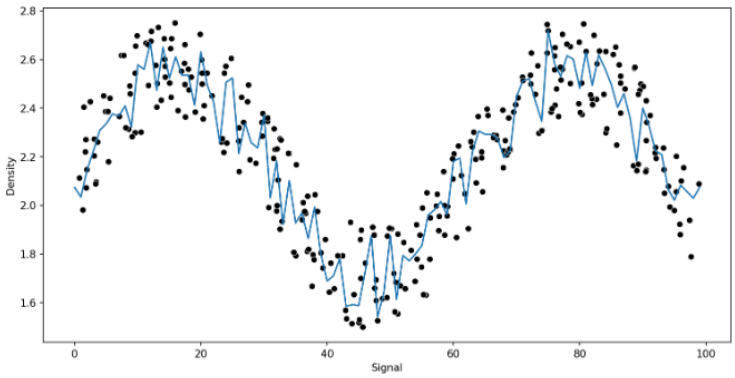
▼ Support Vector Regression

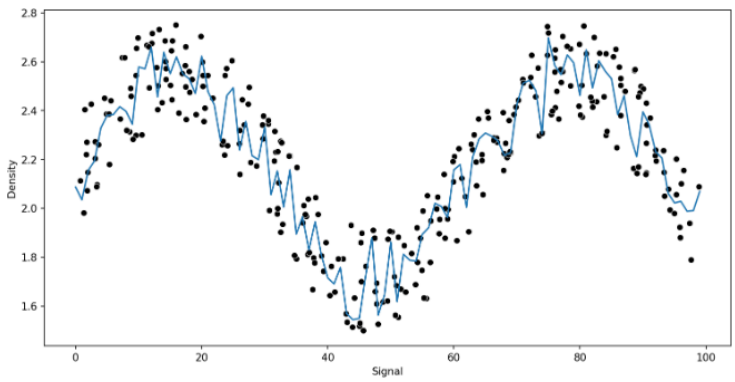
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

param_grid = {'C':[0.01,0.1,1,5,10,100,1000],'gamma':['aut
o','scale']}
svr = SVR()
grid = GridSearchCV(svr,param_grid)
run_model(grid,X_train,y_train,X_test,y_test)

grid.best_estimator_
```

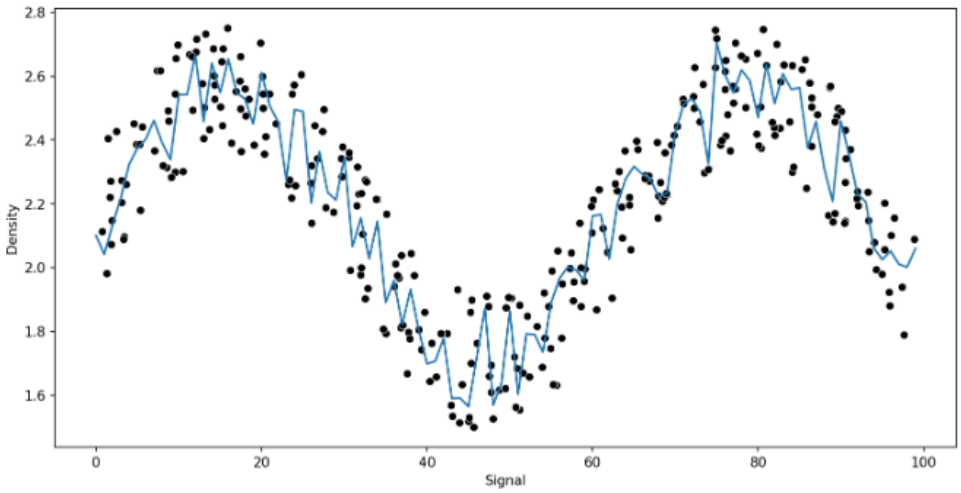


▼ Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
trees = [10,50,100]
for n in trees:

    model = RandomForestRegressor(n_estimators=n)

    run_model(model,X_train,y_train,X_test,y_test)
```







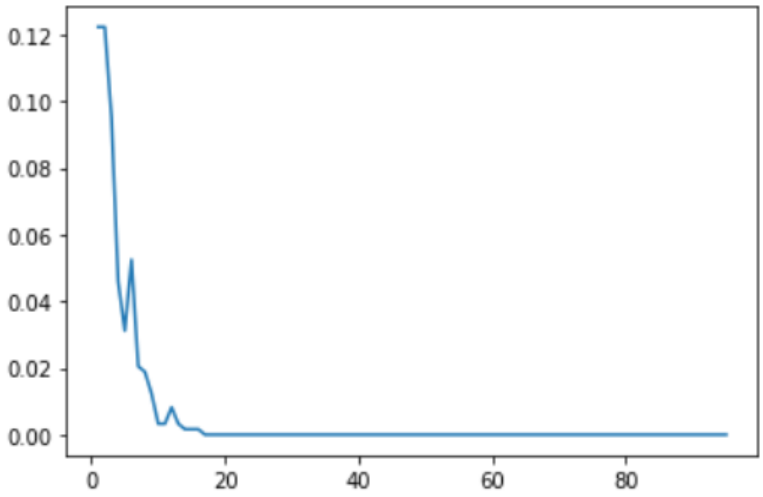
▼ Boosting

mushrooms.csv

▼ AdaBoost

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

df = pd.read_csv("../DATA/mushrooms.csv")
X = df.drop('class',axis=1)
X = pd.get_dummies(X,drop_first=True)
y = df['class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=101)
```

```
#Modeling
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(n_estimators=1)
model.fit(X_train,y_train)

#Evaluation
from sklearn.metrics import classification_report,plot_confusion_m
atrix,accuracy_score
predictions = model.predict(X_test)
print(classification_report(y_test,predictions))

#Analyzing performance as more weak learners are added
error_rates = []

for n in range(1,96):

    model = AdaBoostClassifier(n_estimators=n)
    model.fit(X_train,y_train)
    preds = model.predict(X_test)
    err = 1 - accuracy_score(y_test,preds)

    error_rates.append(err)

plt.plot(range(1,96),error_rates)
```



▼ Gradient Boosting

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

df = pd.read_csv("../DATA/mushrooms.csv")
X = df.drop('class',axis=1)
y = df['class']
X = pd.get_dummies(X,drop_first=True)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=101)

# Gradient Boosting and Grid Search with CV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
param_grid = {"n_estimators":[1,5,10,20,40,100],'max_depth':[3,4,5,
6]}
```

```
gb_model = GradientBoostingClassifier()
grid = GridSearchCV(gb_model,param_grid)

#Fit to Training Data with CV Search
grid.fit(X_train,y_train)
grid.best_params_

from sklearn.metrics import classification_report,plot_confusion_m
atrix,accuracy_score
print(classification_report(y_test,grid.predict(X_test)))
```

- Naive Bayes and NLP
  - Feature Extraction from Text
    - CountVectorizer

      Creates bag of words from the given text

      ```
      text = ['This is a line',
              "This is another line",
            "Completely different line"]
      from sklearn.feature_extraction.text import TfidfTransformer,Tf
      idfVectorizer,CountVectorizer
      cv = CountVectorizer()
      sparse_mat = cv.fit_transform(text)
      sparse_mat.todense()       #sparse matrix is a matrix with many
      0 elements

      cv = CountVectorizer(stop_words='english')  #stop words are t
      he very common words in english like this, the, another, etc
      cv.fit_transform(text).todense()
      ```

    - TfidfTransformer

TfidfVectorizer is used on sentences, while TfidfTransformer is used on an existing count matrix, such as one returned by CountVectorizer

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
cv = CountVectorizer()
counts = cv.fit_transform(text)
tfidf = tfidf_transformer.fit_transform(counts)
tfidf.todense()
```

▼ TfidfVectorizer

[CountVectorizer + TfidfTransformer]

Does both above in a single step

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
new = tfidf.fit_transform(text)
new.todense()
```

▼ Text Classification

airline_tweets.csv

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("../DATA/airline_tweets.csv")

y = df['airline_sentiment']
X = df['text']
```

```
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, r
andom_state=101)

#Vectorization
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
tfidf.fit(X_train)
X_train_tfidf = tfidf.transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

from sklearn.metrics import plot_confusion_matrix,classification_r
eport
def report(model):
    preds = model.predict(X_test_tfidf)
    print(classification_report(y_test,preds))
    plot_confusion_matrix(model,X_test_tfidf,y_test)

#Model Comparisons - Naive Bayes,LogisticRegression, LinearSV
C
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train_tfidf,y_train)
report(nb)

from sklearn.linear_model import LogisticRegression
log = LogisticRegression(max_iter=1000)
log.fit(X_train_tfidf,y_train)
report(log)

from sklearn.svm import LinearSVC
svc = LinearSVC()
svc.fit(X_train_tfidf,y_train)
report(svc)
```

```
#Finalizing a PipeLine for Deployment on New Tweets
from sklearn.pipeline import Pipeline
pipe = Pipeline([('tfidf',TfidfVectorizer()),('svc',LinearSVC())])
pipe.fit(df['text'],df['airline_sentiment'])
new_tweet = ['good flight']
pipe.predict(new_tweet)
```

▼ UNSUPERVISED LEARNING

No train-test-split, no Y part of data

▼ K-Means Clustering


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

df = pd.read_csv("../DATA/bank-full.csv")

X = pd.get_dummies(df)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)

from sklearn.cluster import KMeans
model = KMeans(n_clusters=2)
cluster_labels = model.fit_predict(scaled_X)
X['Cluster'] = cluster_labels
X.corr()['Cluster']

#Choosing K Value
```

```
ssd = []

for k in range(2,10):

    model = KMeans(n_clusters=k)


    model.fit(scaled_X)

    #Sum of squared distances of samples to their closest cluster cente
r.
    ssd.append(model.inertia_)
```

▼ Color Quantization

```

```

```
import numpy as np

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

image_as_array = mpimg.imread('../DATA/14.jpg')
image_as_array # RGB CODES FOR EACH PIXEL

plt.figure(figsize=(6,6),dpi=200)
plt.imshow(image_as_array)

#Using Kmeans to Quantize Colors
image_as_array.shape  ⟶ (1216, 2160, 3)
# (h,w,3 color channels)

#Convert from 3d to 2d
(h,w,c) = image_as_array.shape
```

```
image_as_array2d = image_as_array.reshape(h*w,c)

from sklearn.cluster import KMeans
model = KMeans(n_clusters=10)

labels = model.fit_predict(image_as_array2d)

# THESE ARE THE 10[=n_clusters] RGB COLOR CODES!
model.cluster_centers_  This is an array of codes but in decimals
rgb_codes = model.cluster_centers_.round(0).astype(int)

quantized_image = np.reshape(rgb_codes[labels], (h, w, c))
plt.figure(figsize=(6,6),dpi=200)
plt.imshow(quantized_image)
```

▼ Hierarchal Clustering

cluster_mpg.csv

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

df = pd.read_csv('../DATA/cluster_mpg.csv')
df = df.dropna()
df_w_dummies = pd.get_dummies(df.drop('name',axis=1))

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df_w_dummies)

scaled_df = pd.DataFrame(scaled_data,columns=df_w_dummies.colu
```

```
mns)

from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=4)
cluster_labels = model.fit_predict(scaled_df)

#Exploring Number of Clusters with Dendrograms
model = AgglomerativeClustering(n_clusters=None,distance_threshold
=0)
cluster_labels = model.fit_predict(scaled_df)

#Linkage Model
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster import hierarchy
linkage_matrix = hierarchy.linkage(model.children_)

plt.figure(figsize=(20,10))
dn = hierarchy.dendrogram(linkage_matrix)

plt.figure(figsize=(20,10))
dn = hierarchy.dendrogram(linkage_matrix,truncate_mode='lastp',p=4
8)

#Choosing a Threshold Distance (distance between two points)
a = scaled_df.iloc[320]
b = scaled_df.iloc[28]
dist = np.linalg.norm(a-b)

#Creating a Model Based on Distance Threshold
model = AgglomerativeClustering(n_clusters=None,distance_threshold
=2)
cluster_labels = model.fit_predict(scaled_data)
linkage_matrix = hierarchy.linkage(model.children_)

plt.figure(figsize=(20,10))
dn = hierarchy.dendrogram(linkage_matrix,truncate_mode='lastp',p=1
```

1)

▼ DBSCAN

▼ Intro

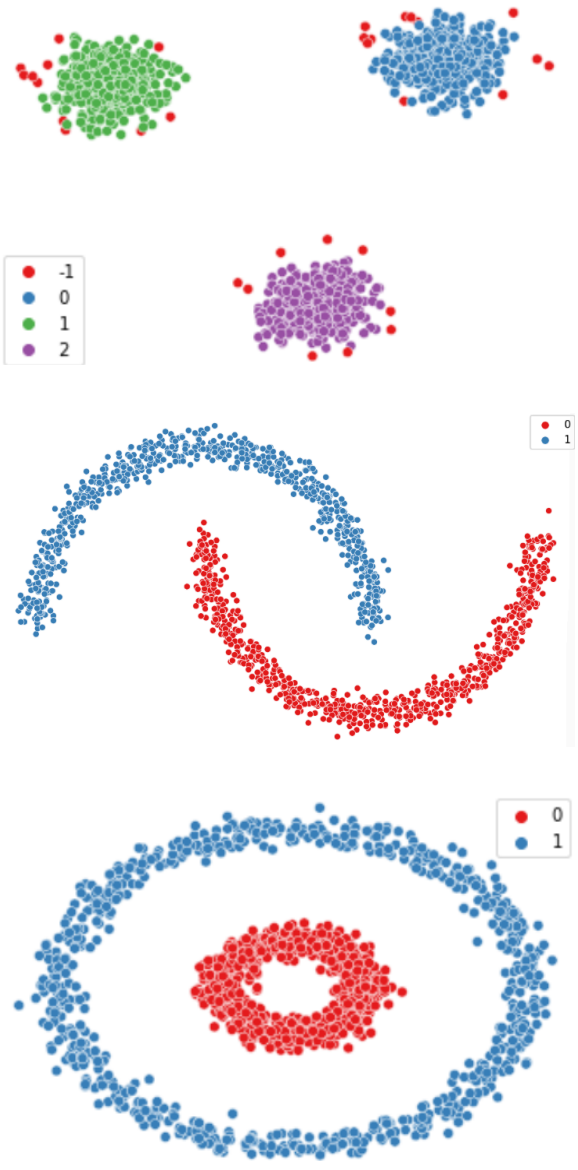
cluster_circles.csv

cluster_moons.csv

cluster_blobs.csv

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

blobs = pd.read_csv('../DATA/cluster_blobs.csv')
moons = pd.read_csv('../DATA/cluster_moons.csv')
circles = pd.read_csv('../DATA/cluster_circles.csv')

def display_categories(model,data):
    labels = model.fit_predict(data)
    sns.scatterplot(data=data,x='X1',y='X2',hue=labels,palette='Set
1')

from sklearn.cluster import DBSCAN
model = DBSCAN(eps=0.6)
```

```
display_categories(model,blobs)

display_categories(model,moons)

display_categories(model,circles)
```



▼ Hyperparameters

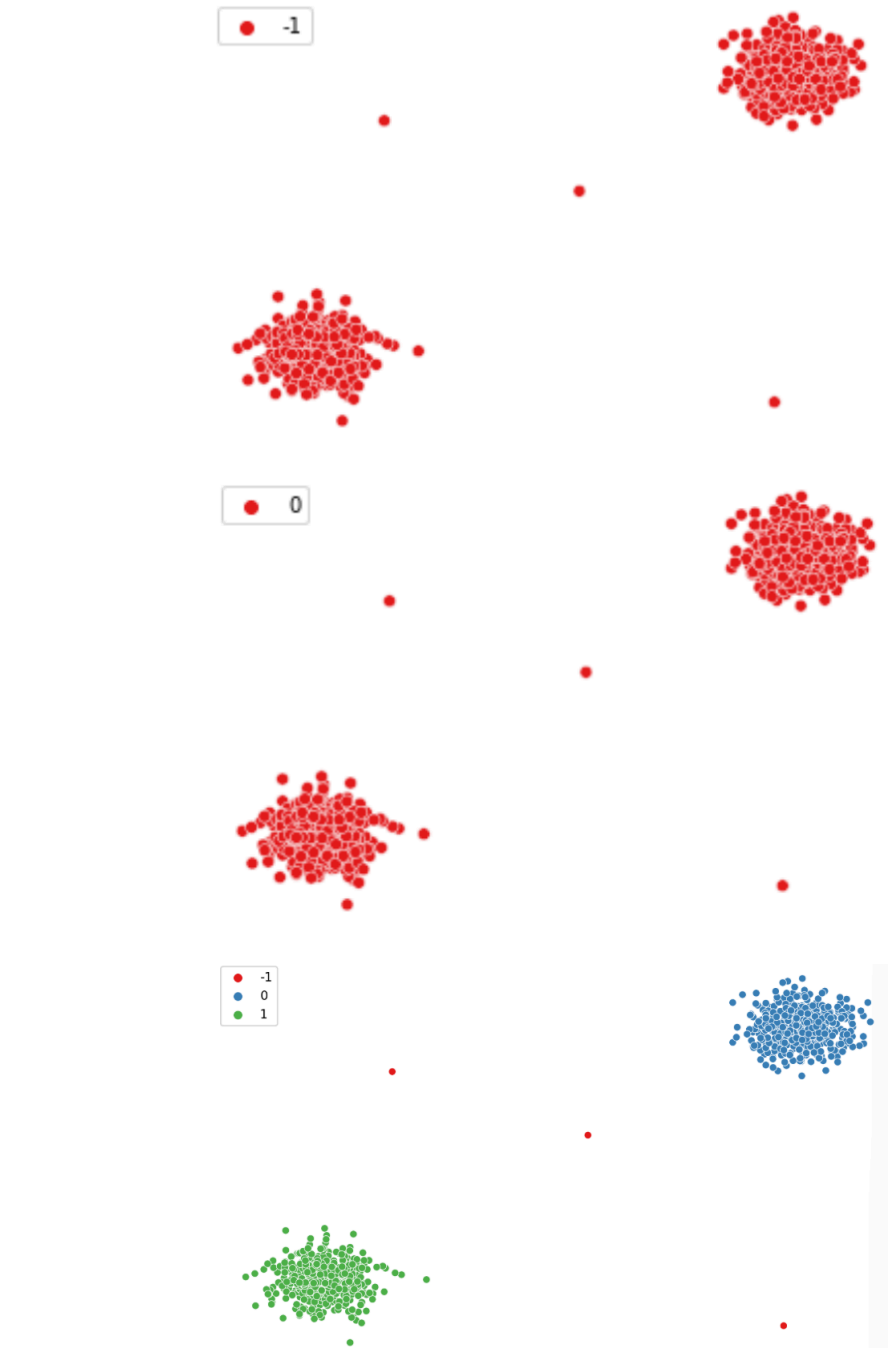
```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns

two_blobs = pd.read_csv('../DATA/cluster_two_blobs.csv')
two_blobs_outliers = pd.read_csv('../DATA/cluster_two_blobs_outli
ers.csv')

def display_categories(model,data):
    labels = model.fit_predict(data)
    sns.scatterplot(data=data,x='X1',y='X2',hue=labels,palette='Set
1')

from sklearn.cluster import DBSCAN
dbscan = DBSCAN()
display_categories(dbscan,two_blobs)

display_categories(dbscan,two_blobs_outliers)
```

▼ Epsilon

Max distance between two points

```
dbscan = DBSCAN(eps=0.001)
display_categories(dbscan,two_blobs_outliers)

dbscan = DBSCAN(eps=10)
display_categories(dbscan,two_blobs_outliers)

dbscan = DBSCAN(eps=1)
display_categories(dbscan,two_blobs_outliers)
```

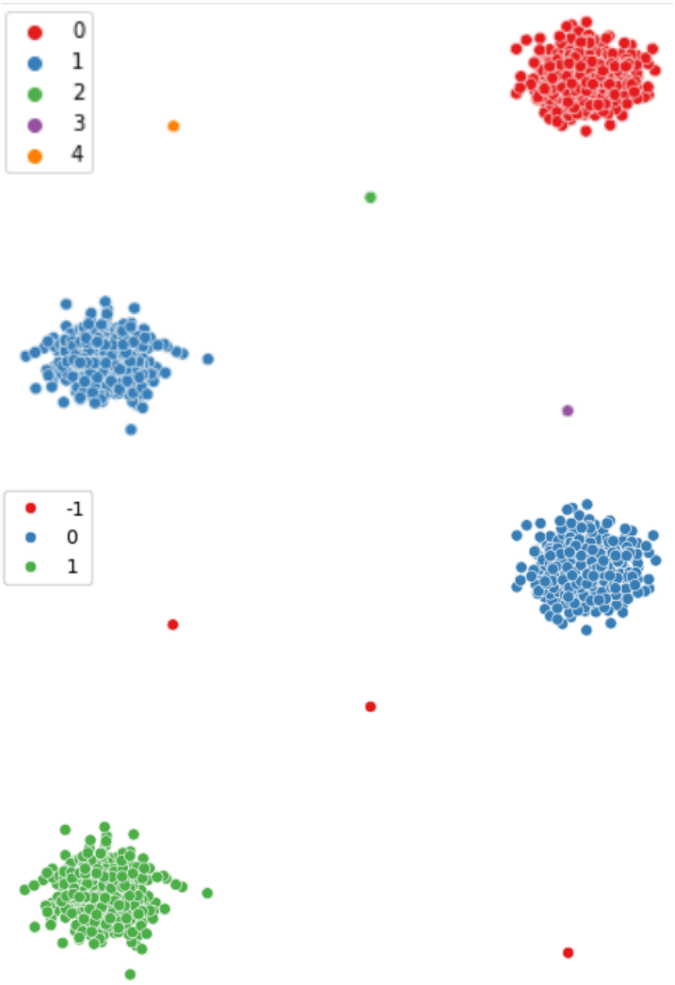


▼ Minimum samples

Number of points in neighborhood including the point itself

```
dbscan = DBSCAN(eps=0.75,min_samples=1)
display_categories(dbscan,two_blobs_outliers)
```

```
dbscan = DBSCAN(eps=0.75,min_samples=3)
display_categories(dbscan,two_blobs_outliers)
```



Types of cross validation