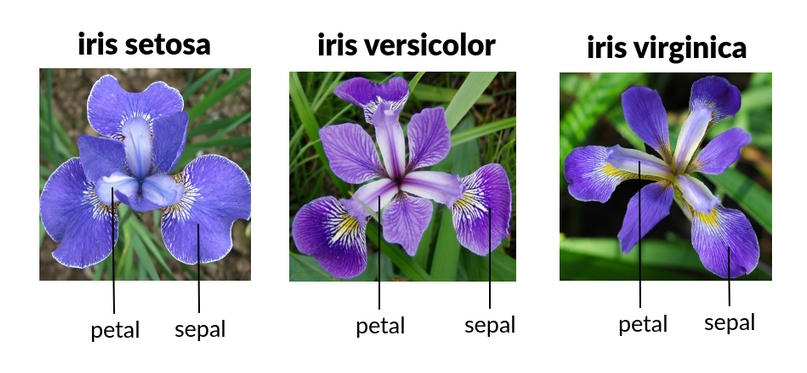
**Iris Flower Classification**



Importing libraries

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_palette('husl')

import matplotlib.pyplot as plt

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

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

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.svm import SVC

Loading IRIS Data

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'

Creating the list of column name:

col\_name = ['sepal-lenght','sepal-width','petal-lenght','petal-width','class']

Pandas read\_csv() is used for reading the csv file:

dataset = pd.read\_csv(url, names = col\_name)

**Violin plot**  
Plotting the violin plot to check the comparison of a variable distribution:

sns.violinplot(y='class', x='sepal-lenght', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='sepal-width', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='petal-lenght', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='petal-width', data=dataset, inner='quartile')

plt.show()

## 5. Model Building- part 1

**5.1 Splitting the dataset**  
X is having all the dependent variables.   
Y is having an independent variable (here in this case ‘class’ is an independent variable).

X = dataset.drop(['class'], axis=1)

y = dataset['class']

print(f'X shape: {X.shape} | y shape: {y.shape} ')

**5.2 Train Test split**Splitting our dataset into train and test using train\_test\_split(), what we are doing here is taking 80% of data to train our model, and 20% that we will hold back as a validation dataset:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=1)

**5.3 Model Creation ( SVC: Support Vector Machine for classification)**

models = []

models.append(('SVC', SVC(gamma='auto')))

results = []

model\_names = []

for name, model in models:

  kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

  cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')

  results.append(cv\_results)

  model\_names.append(name)

  print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

**6. Model Building- part 2**6.1.We are defining our SVC model and passing gamma as auto.

6.2. After that fitting/training the model on X\_train and Y\_train using .fit() method.

6.3. Then we are predicting on X\_test using .predict() method.

 model = SVC(gamma='auto')

model.fit(X\_train, y\_train)

prediction = model.predict(X\_test)

6.4. checking the accuracy of our model using   
accuracy\_score(y\_test, prediction)  
y\_test: actual values of X\_test  
prediction: predicted values of X\_test (refer to point 3).

6.5. Printing out the classification report using  
classification\_report(y\_test, prediction).

print(f'Test Accuracy: {accuracy\_score(y\_test, prediction)}')

print(f'Classification Report: \n {classification\_report(y\_test, prediction)}')

**Total Code**

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_palette('husl')

import matplotlib.pyplot as plt

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

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

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.svm import SVC

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'

col\_name = ['sepal-lenght','sepal-width','petal-lenght','petal-width','class']

dataset = pd.read\_csv(url, names = col\_name)

sns.violinplot(y='class', x='sepal-lenght', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='sepal-width', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='petal-lenght', data=dataset, inner='quartile')

plt.show()

sns.violinplot(y='class', x='petal-width', data=dataset, inner='quartile')

plt.show()

X = dataset.drop(['class'], axis=1)

y = dataset['class']

print(f'X shape: {X.shape} | y shape: {y.shape} ')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=1)

models = []

models.append(('SVC', SVC(gamma='auto')))

# evaluate each model in turn

results = []

model\_names = []

for name, model in models:

  kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

  cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')

  results.append(cv\_results)

  model\_names.append(name)

  print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

  model = SVC(gamma='auto')

model.fit(X\_train, y\_train)

prediction = model.predict(X\_test)

print(f'Test Accuracy: {accuracy\_score(y\_test, prediction)}')

print(f'Classification Report: \n {classification\_report(y\_test, prediction)}')

Project 2: Regression

from google.colab import drive

drive.mount('/gdrive')

#1 Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

X = np.array([1,2,3,4,5,6,7,8,9,10]).astype(float)

y =np.array([45000, 50000, 60000, 80000, 110000, 150000, 200000, 300000, 500000, 1000000]).astype(float)

X = np.expand\_dims(X, axis=-1)

y = np.expand\_dims(y, axis=-1)

print(X.shape,' ',y.shape)

plt.scatter(X, y, color='r')

plt.show()

#3 Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

sc\_y = StandardScaler()

X = sc\_X.fit\_transform(X)

y = sc\_y.fit\_transform(y)

plt.scatter(X, y, color='r')

plt.show()

# Fitting the Support Vector Regression Model to the dataset

from sklearn.svm import SVR

# most important SVR parameter is Kernel type. It can be #linear,polynomial or gaussian SVR. We have a non-linear condition #so we can select polynomial or gaussian but here we select RBF(a #gaussian type) kernel.

regressor = SVR(kernel='rbf')

regressor.fit(X,y)

X\_grid = np.arange(min(X), max(X), 0.01) #this step required because data is feature scaled.

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

#5 Predicting a new result

ip = sc\_X.transform(np.array([[6.5]]))

y\_pred = regressor.predict (ip)

print('Predicted salary (normalized):',y\_pred)

y\_pred = sc\_y.inverse\_transform([y\_pred])

print('Predicted salary',y\_pred)

output:

Predicted salary (normalized): [-0.27861589]

Predicted salary [[170370.0204065]]

Project 3:

import numpy as np #importing numpy

users = ["user1","user2","user3"] # users  is a list

products = ["laptop","mouse","monitor","pen drive","hard disk"] # list

uids = {} #dict

for i in range(len(users)): # start=0, stop 3 step 1  i 0 i<stop 0<3 True

  uids[users[i]]=i # uids[users[0]]=0  user1: 0 user2:1 user3:2

pids = {} # dict

for i in range(len(products)):

  pids[products[i]]=i # pids[laoptop]=0 , mouse 1 monitor 2 pendrive 3 harddisk 4

print(uids, pids)

prd\_fq = np.zeros((len(users),len(products)) ) # 3\*5

print(prd\_fq)

def reclist(uid): # function definition

  uid = uids[uid] # Row id of User

  upids = prd\_fq[uid, :]  # column id of the products

  plist = np.argsort(upids)[::-1][:len(upids)] #frequnecy based sorting for the products

  plist = [products[p] for p in plist] # to retrive items in the sorted order based on frequency

  print(plist) #display

def viewproduct(uid, pid): # function definition

  uid = uids[uid] # row id of user

  pid = pids[pid] # column id of the product

  prd\_fq[uid, pid] = prd\_fq[uid, pid] + 1  # frequency increment

  print(prd\_fq) # display

uname = input("Enter user name:")

print(reclist(uname))

pname = input("Select one product:")

viewproduct(uname, pname)

Input: Enter the user name : user1

Enter the product : hard disk

Output: [[0. 0. 0. 0. 1.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]]