DATA SCIENCE PROGRAMS

Except 3rd program 😁 😁 😁 😁 😁

Program 1:

i) Indexing of data frames:

```
import pandas as pd

data = {'name':['raj','hus','iguc','idciu'],'age':[22,24,58,65]}

df = pd.DataFrame(data)

print(df)SS

df1 = pd.DataFrame(data,index=['rank1','rank2','rank3','rank4'])

print(df1)

import pandas as pd

d={'col1':[1,2],'col2':[3,4]}

df = pd.DataFrame(d)

print(df)
```

ii)Adding and removing attributes:

```
import pandas as pd

df = pd.DataFrame({'month':[1,4,7,10],'year':[2012,2014,2016,2018],'sales':[25,36,46,75]})

print(df)

a = df.set_index('month')

print(a)

b =df.set_index(['year','month'])

print(b)

c = df.set_index([pd.Index([1,2,3,4]),'year'])

print(c)
```

iii) Grouping and aggregations:

```
import numpy as np

import pandas as pd
```

```
df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],[np.nan,np.nan,np.nan]],
                columns=['A','B','C'])
print(df)
a=df.agg(['sum','min'])
print(a)
b=df.agg({'A':['sum','min'],'B':['min','max']})
print(b)
c=df.agg(x=('A',max),y=('B',min),z=('C',np.mean))
print(c)
```

iv)Joining Data Frames:

```
import pandas as pd
df = pd.DataFrame({'Animal' : ['Falcon', 'Falcon',
                            'Parrot', 'Parrot'],
      'Max Speed' : [380., 370., 24., 26.]})
df
df.groupby(['Animal']).mean()
###################################################
arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
    ['Capitve', 'Wild', 'Capitve', 'Wild']]
index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
df = pd.DataFrame({'Max Speed' : [390., 350., 30., 20.]},
                 index=index)
df
#################################################
df.groupby(level=0).mean()
################################################
df.groupby(level=1).mean()
```

v)Filtering the data:

```
import pandas as pd
import numpy as mp
```

```
df = pd.DataFrame(mp.random.randn(5,3),index = ['a','c','e','f','h'],columns = ['one','two','three'])

print(df)

x = df.filter(['a','c','e','f','h'])

print()
```

vi)Handling the missing Values:

```
import pandas as pd

import numpy as np

df = pd.DataFrame([np.arange(1,4)],index =['a','b','c'],columns = ['x','y','z'])

print(df)

x =df.reindex(index = ['a','b','c','d'],fill_value = "null")

print(x)

y = pd.isna(df)

print(y)

z = pd.notna(df)

print(z)
```

Program 2:

i) Bar Chart:

```
import numpy as np
import matplotlib.pyplot as plt
data = {'c':20,'c++':15,'java':25,'json':30}
course=list(data.keys())
values=list(data.values())
fig = plt.subplots(figsize=(10,5))
plt.bar(course,values,color="maroon",width=0.4)
plt.xlabel("Courses")
plt.ylabel("Np. Of student env0olved")
plt.title("ugcosd")
plt.show()
```

ii)Histogram:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from matplotlib.ticker import PercentFormatter

np.random.seed(984984984)
```

```
N_points=10000
n_bins=20

x = np.random.randn(N_points)
y = .8**x+np.random.randn(10000)+25

fig,ax = plt.subplots(1,1,figsize=(10,7),tight_layout = True)
plt.hist(x,bins= n_bins)
plt.legend("Distribution")

for s in ['top','bottom','left','right']:
  ax.spines[s].set_visible(False)

ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

ax.xaxis.set_tick_params(pad=5)
ax.yaxis.set_tick_params(pad=10)

ax.grid(b = True,color="grey" ,linestyle ="-", linewidth = 0.5,alpha = 0.5)

fig.text(0.9,0.15,"Gautam Vusiodj", fontsize = 15 ,color="blue",va = "bottom" , ha = "lef
t")

N,bins,patches = ax.hist(x,bins=n_bins)

fracs = ((N**(0.1))/N.max())
norm = colors.Normalize(fracs.min(),fracs.max())

for thisfrac,thispatch in zip(fracs,patches):
  color = plt.cm.viridis(norm(thisfrac))
  thispatch.set_facecolor(color)
plt.show()
```

iii)Line Graph:

```
import numpy as np
import matplotlib.pyplot as plt
arr1 = np.array([10,20,30,40,50])
arr2 = np.array([25,35,45,55,65])
plt.plot(arr1,arr2)
plt.xlabel('Sales')
plt.ylabel('Price')
plt.title('Graph of Sales vs Price')
plt.show()
```

iv)Comparing distributions:

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

overs = np.array([10,20,30,40,50])
team1 = np.array([32,25,74,15,24])
plt.plot(overs,team1,color = "green" , marker ="*" , markersize = 10 , linewidth = 2)
```

```
team2 = np.array([45,36,45,66,52])
plt.plot(overs,team2, color = "blue" , marker="*" , markersize = 10, linewidth = 2)
plt.xlabel("Scores")
plt.ylabel("Overs")
plt.title("scores per 1 overs")
plt.show()
```

v) Box Plot:

```
from google.colab import drive
drive.mount('/content/drive')
import matplotlib.pyplot as plt
import numpy as np

# Random test data
np.random.seed(19680801)
all_data = [np.random.normal(0, std, size=100) for std in range(1, 4)]
labels = ['x1', 'x2', 'x3']

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(9, 4))

# rectangular box plot
bplot1 = ax1.boxplot(all_data,
                     vert=True,  # vertical box alignment
                     patch_artist=True,  # fill with color
                     labels=labels)  # will be used to label x-ticks
ax1.set_title('Rectangular box plot')

# notch shape box plot
bplot2 = ax2.boxplot(all_data,
                     notch=True,  # notch shape
                     vert=True,  # vertical box alignment
                     patch_artist=True,  # fill with color
                     labels=labels)  # will be used to label x-ticks
ax2.set_title('Notched box plot')

# fill with colors
colors = ['pink', 'lightblue', 'lightgreen']
for bplot in (bplot1, bplot2):
    for patch, color in zip(bplot['boxes'], colors):
        patch.set_facecolor(color)

# adding horizontal grid lines
for ax in [ax1, ax2]:
    ax.yaxis.grid(True)
    ax.set_xlabel('Three separate samples')
    ax.set_ylabel('Observed values')

plt.show()
```

vi) Correlation using Pairplot

```
import seaborn as sns
penguins = sns.load_dataset("penguins")
sns.pairplot(penguins)
sns.pairplot(penguins,hue="species")
sns.pairplot(penguins,hue="species",diag_kind="hist")
sns.pairplot(penguins,kind="kde")
sns.pairplot(penguins,kind="hist")
sns.pairplot(penguins,hue="species",markers=['s','o','D'])
sns.pairplot(penguins,height=1.5)
```

vii)Heat Map:

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

vegetables = ["cucumber", "tomato", "lettuce", "asparagus", "potato", "wheat", "barley"]
farmers = ["Farmer Joe", "Upland Bros.", "Smith Gardening", "Agrifun", "Organiculture", "BioGoods Ltd.", "Cornylee Corp."]

harvest = np.array([[0.8, 2.4, 2.5, 3.9, 0.0, 4.0, 0.0],
                    [2.4, 0.0, 4.0, 1.0, 2.7, 0.0, 0.0],
                    [1.1, 2.4, 0.8, 4.3, 1.9, 4.4, 0.0],
                    [0.6, 0.0, 0.3, 0.0, 3.1, 0.0, 0.0],
                    [0.7, 1.7, 0.6, 2.6, 2.2, 6.2, 0.0],
                    [1.3, 1.2, 0.0, 0.0, 0.0, 3.2, 5.1],
                    [0.1, 2.0, 0.0, 1.4, 0.0, 1.9, 6.3]])

fig, ax = plt.subplots()
im = ax.imshow(harvest)
# We want to show all ticks...
ax.set_xticks(np.arange(len(farmers)))
ax.set_yticks(np.arange(len(vegetables)))
# ... and label them with the respective list entries
ax.set_xticklabels(farmers)
ax.set_yticklabels(vegetables)
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
# Loop over data dimensions and create text annotations.
for i in range(len(vegetables)):
    for j in range(len(farmers)):
        text = ax.text(j, i, harvest[i, j], ha="center", va="center", color="w")

ax.set_title("Harvest of local farmers (in tons/year)")

fig.tight_layout()
```

Program 4:

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

def estimate_coef(x,y):
  n=np.size(x)

  m_x=np.mean(x)
  m_y=np.mean(y)
  SS_xy=np.sum(y*x)-n*m_y*m_x
  SS_xx=np.sum(y*x)-n*m_x*m_x

  b_1=SS_xy/SS_xx
  b_0=m_y-b_1*m_x

  return (b_0,b_1)

def plot_regression_line(x,y,b):
  plt.scatter(x,y,color="r", marker='x',s=50)
  y_pred=b[0]+b[1]*x
  plt.plot(x,y_pred,color="g")
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()
def main():
    x=np.array([0,1,2,3,4,5,6,7,8,9])
    y=np.array([1,3,2,5,7,8,8,9,10,12])
    b=estimate_coef(x,y)
    print("Estimated Coefficients:\nb_0={}\nb_1={}".format(b[0],b[1]))
    plot_regression_line(x,y,b)
if __name__=="__main__":
   main()
```

Program 5:

```
import pandas
from sklearn import linear_model

df=pandas.read_csv("cars.csv")

x=df[['Weight','Volume']]
y=df['CO2']

regr=linear_model.LinearRegression()
regr.fit(x,y)

#predict the CO2 emission of a car where the weight is 2300kg and volume 1300cm3
predictedCO2=regr.predict([[2300,1300]])

print(predictedCO2)
print(regr.coef_)
```

Program 6:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix

x=np.arange(10).reshape(-1,1)
y=np.array([0,1,0,0,1,1,1,1,1,1])
```

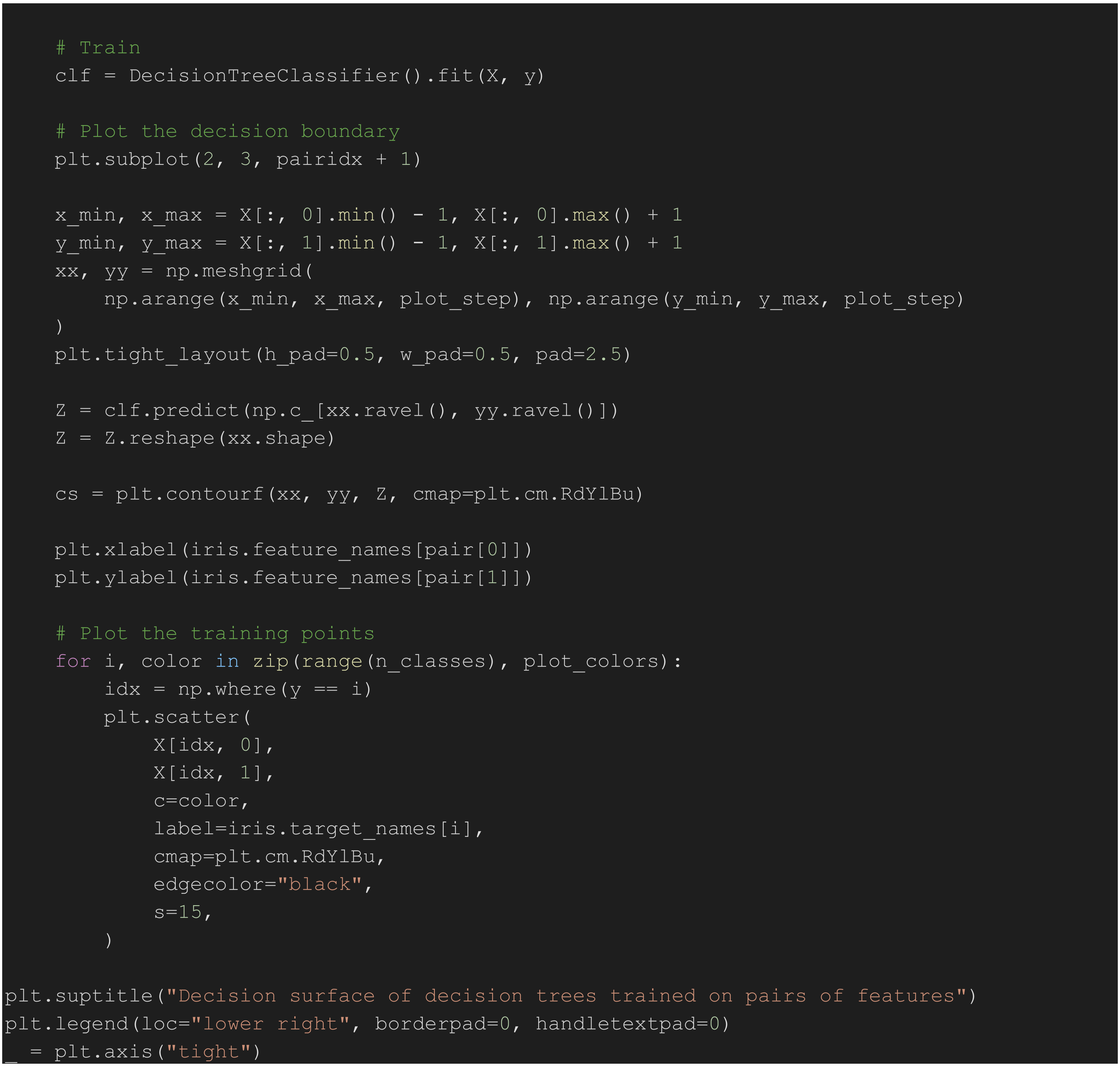
```
model= LogisticRegression()
model.fit(x,y)

p_pred=model.predict_proba(x)
y_pred=model.predict(x)
score_=model.score(x,y)
conf_m= confusion_matrix(y,y_pred)
report = classification_report(y,y_pred)
print('x:',x,sep='\n')
print('y:',y,sep='\n' ,end='\n\n')
print('intercept:',model.intercept_)
print('coef:',model.coef_, end='\n\n')
print('p_pred:',p_pred,sep='\n' ,end='\n\n')
print('y_pred:',y_pred ,end='\n\n')
print('score_:',score_,end='\n\n')
print('conf_m:',conf_m,sep='\n' ,end='\n\n')
print('report:',report,sep='\n')
```

Program 7:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = load_iris (return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=0)
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0], (y_test != y_pred).sum()))
```

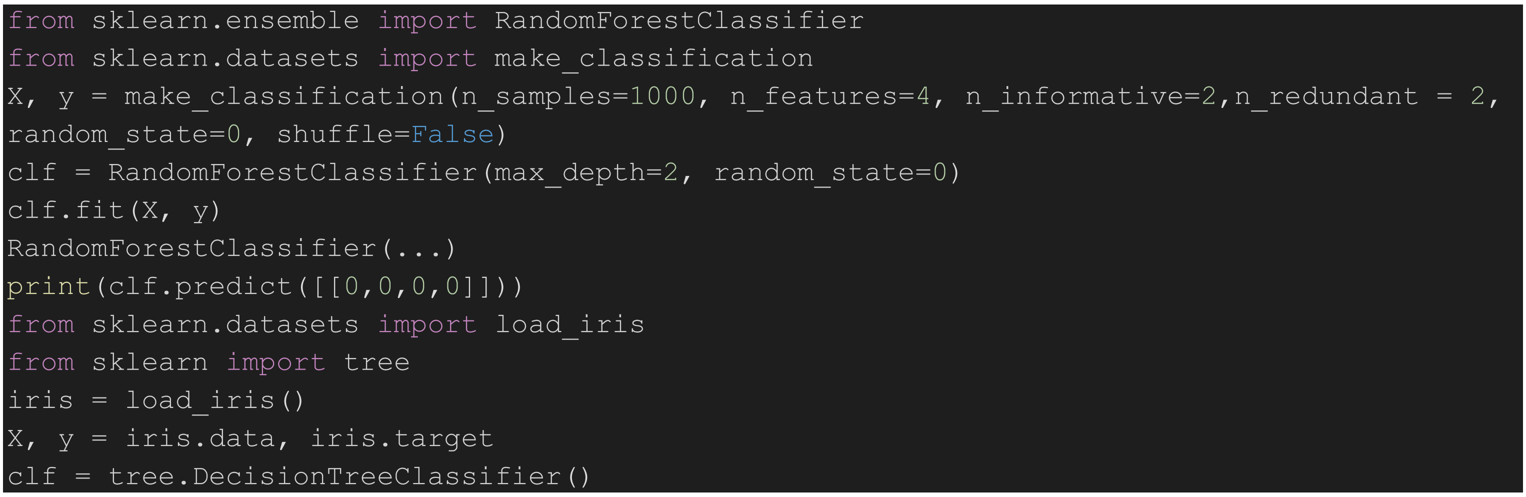
Program 8:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

iris = load_iris()
# Parameters
n_classes = 3
plot_colors = "ryb"
plot_step = 0.02


for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]]):
    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target
```



```
    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(
        np.arange(x_min, x_max, plot_step), np.arange(y_min, y_max, plot_step)
    )
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(
            X[idx, 0],
            X[idx, 1],
            c=color,
            label=iris.target_names[i],
            cmap=plt.cm.RdYlBu,
            edgecolor="black",
            s=15,
        )

plt.suptitle("Decision surface of decision trees trained on pairs of features")
plt.legend(loc="lower right", borderpad=0, handletextpad=0)
_ = plt.axis("tight")
```

Program 9:



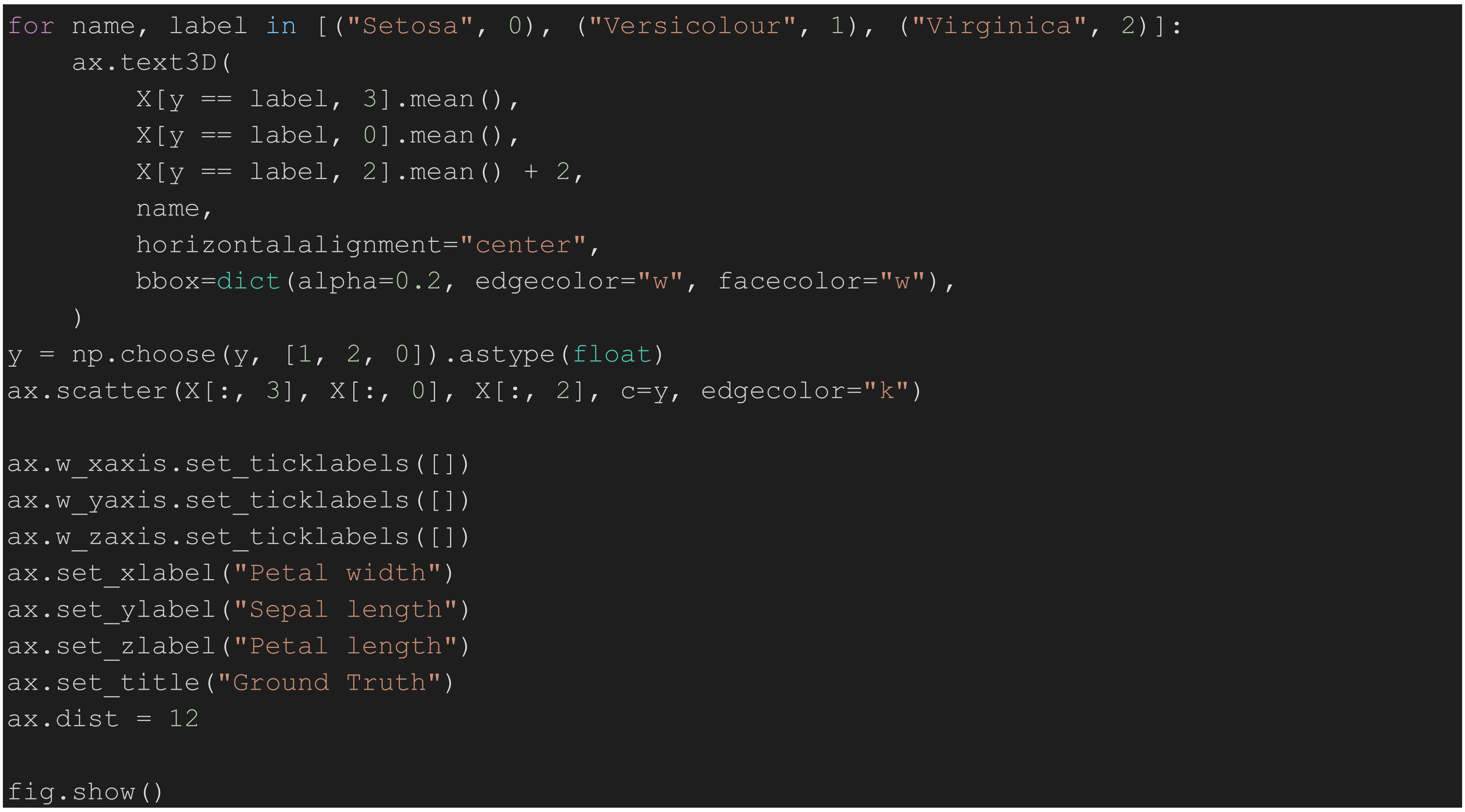
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4, n_informative=2,n_redundant = 2,
random_state=0, shuffle=False)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X, y)
RandomForestClassifier(...)
print(clf.predict([[0,0,0,0]]))
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
tree.plot_tree(clf)
```

Program 10:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = 0.02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

for weights in ["uniform", "distance"]:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    sns.scatterplot(
        x=X[:, 0],
        y=X[:, 1],
        hue=iris.target_names[y],
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="black",
    )
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(
        "3-Class classification (k = %i, weights = '%s')" % (n_neighbors, weights)
    )
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])

plt.show()
```

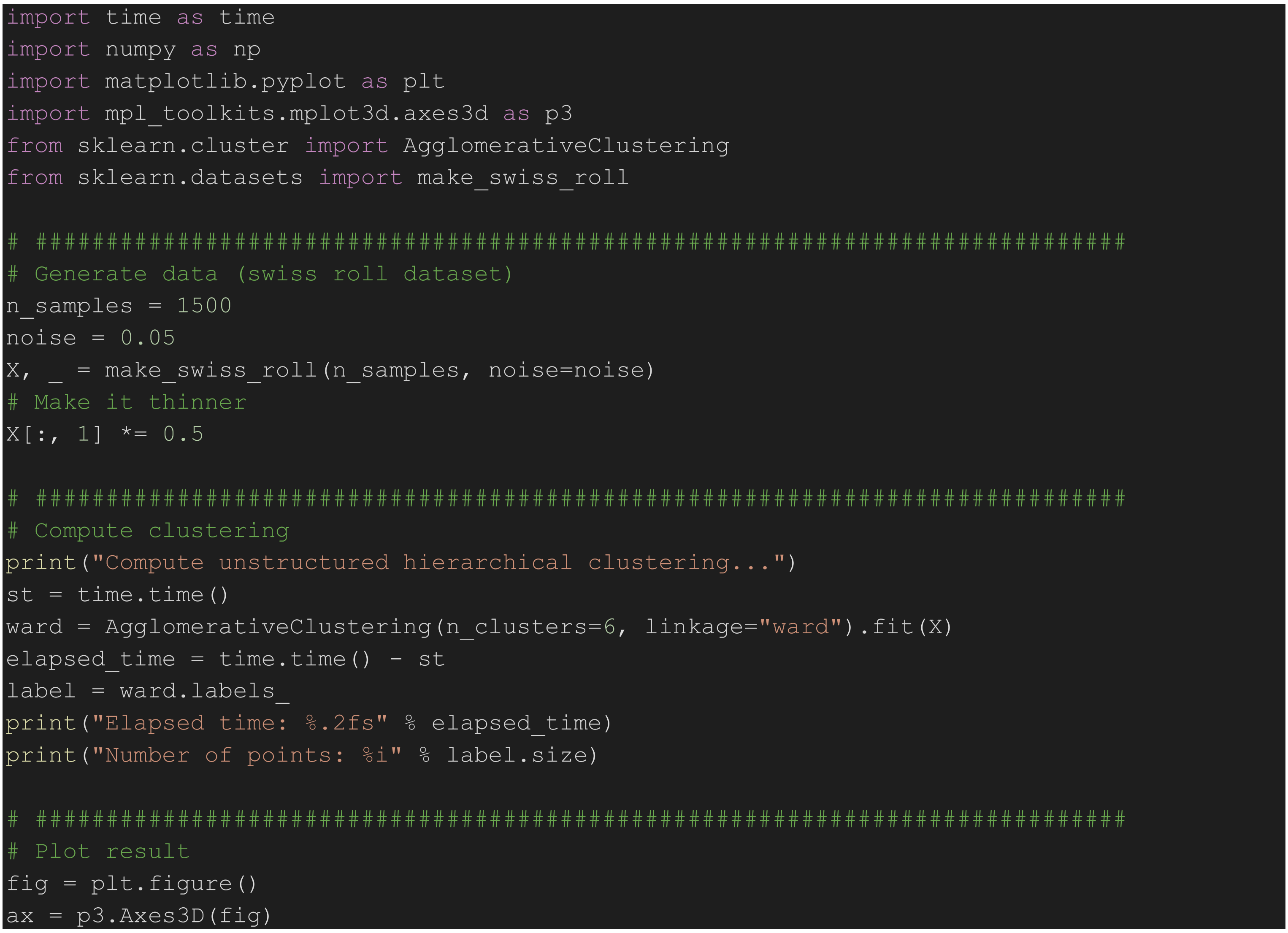
Program 11:

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


from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans
from sklearn import datasets

np.random.seed(5)

iris = datasets.load_iris()
X = iris.data
y = iris.target

estimators = [
    ("k_means_iris_8", KMeans(n_clusters=8)),
    ("k_means_iris_3", KMeans(n_clusters=3)),
    ("k_means_iris_bad_init", KMeans(n_clusters=3, n_init=1, init="random")),
]

fignum = 1
titles = ["8 clusters", "3 clusters", "3 clusters, bad initialization"]
for name, est in estimators:
    fig = plt.figure(fignum, figsize=(4, 3))
    ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
    est.fit(X)
    labels = est.labels_

    ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=labels.astype(float), edgecolor="k")

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel("Petal width")
    ax.set_ylabel("Sepal length")
    ax.set_zlabel("Petal length")
    ax.set_title(titles[fignum - 1])
    ax.dist = 12
    fignum = fignum + 1

fig = plt.figure(fignum, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
```



```
for name, label in [("Setosa", 0), ("Versicolour", 1), ("Virginica", 2)]:
    ax.text3D(
        X[y == label, 3].mean(),
        X[y == label, 0].mean(),
        X[y == label, 2].mean() + 2,
        name,
        horizontalalignment="center",
        bbox=dict(alpha=0.2, edgecolor="w", facecolor="w"),
    )
y = np.choose(y, [1, 2, 0]).astype(float)
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=y, edgecolor="k")

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel("Petal width")
ax.set_ylabel("Sepal length")
ax.set_zlabel("Petal length")
ax.set_title("Ground Truth")
ax.dist = 12

fig.show()
```

Program 12:



```
import time as time
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_swiss_roll

# #############################################################################
# Generate data (swiss roll dataset)
n_samples = 1500
noise = 0.05
X, _ = make_swiss_roll(n_samples, noise=noise)
# Make it thinner
X[:, 1] *= 0.5

# #############################################################################
# Compute clustering
print("Compute unstructured hierarchical clustering...")
st = time.time()
ward = AgglomerativeClustering(n_clusters=6, linkage="ward").fit(X)
elapsed_time = time.time() - st
label = ward.labels_
print("Elapsed time: %.2fs" % elapsed_time)
print("Number of points: %i" % label.size)

# #############################################################################
# Plot result
fig = plt.figure()
ax = p3.Axes3D(fig)
```

```python
ax.view_init(7, -80)
for l in np.unique(label):
    ax.scatter(
        X[label == l, 0],
        X[label == l, 1],
        X[label == l, 2],
        color=plt.cm.jet(float(l) / np.max(label + 1)),
        s=20,
        edgecolor="k",
    )
plt.title("Without connectivity constraints (time %.2fs)" % elapsed_time)


# #############################################################################
# Define the structure A of the data. Here a 10 nearest neighbors
from sklearn.neighbors import kneighbors_graph

connectivity = kneighbors_graph(X, n_neighbors=10, include_self=False)

# #############################################################################
# Compute clustering
print("Compute structured hierarchical clustering...")
st = time.time()
ward = AgglomerativeClustering(
    n_clusters=20, connectivity=connectivity, linkage="ward"
).fit(X)
elapsed_time = time.time() - st
label = ward.labels_
print("Elapsed time: %.2fs" % elapsed_time)
print("Number of points: %i" % label.size)

# #############################################################################
# Plot result
fig = plt.figure()
ax = p3.Axes3D(fig)
ax.view_init(7, -80)
for l in np.unique(label):
    ax.scatter(
        X[label == l, 0],
        X[label == l, 1],
        X[label == l, 2],
        color=plt.cm.jet(float(l) / np.max(label + 1)),
        s=20,
        edgecolor="k",
    )
plt.title("With connectivity constraints (time %.2fs)" % elapsed_time)

plt.show()
```