A PYTHON PROGRAM TO IMPLEMENT KNN MODEL

Expt no. 9A

Name: Jayasuriya.j

Roll no: 241801102

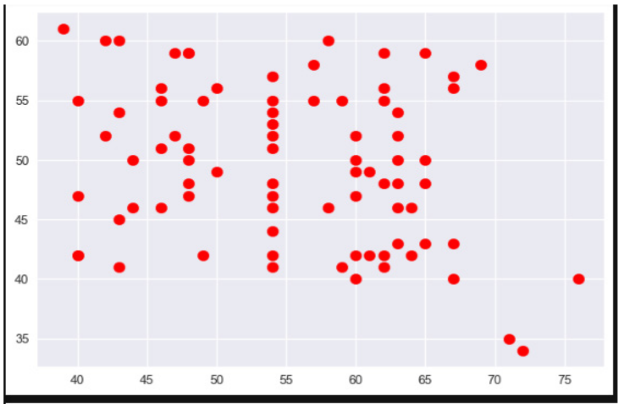
PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans


dataset =

pd.read_csv("C:\\Users\\Luqman\\Downloads\\arc

hive (7)\\Mall_Customers.csv")

X = dataset.iloc[:, [3, 4]].values


kmeans = KMeans(n_clusters=5, init="k-

means++", max_iter=300, n_init=10,

random_state=0)

y_kmeans = kmeans.fit_predict(X)


plt.figure()

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans ==

0, 1], s=80, c="red")

plt.show()


plt.figure() plt.scatter(X[y_kmeans == 1, 0],

X[y_kmeans ==

1, 1], s=80, c="blue")
```

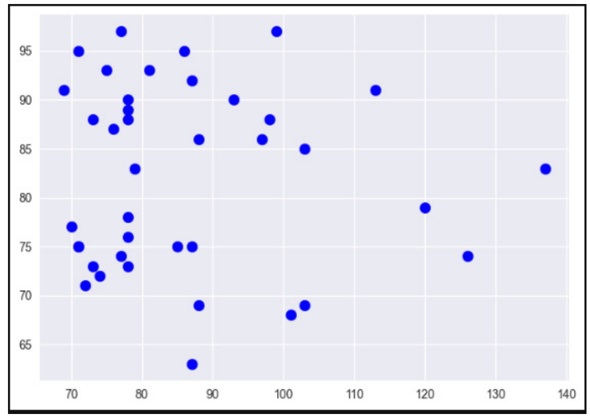
```
plt.show()

plt.figure()
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans ==
2, 1], s=80, c="green")
plt.show()

plt.figure()
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans ==
3, 1], s=80, c="cyan")
plt.show()

plt.figure()
obj = plt.scatter(X[y_kmeans == 4, 0],
X[y_kmeans == 4, 1], s=80, c="magenta")
print(obj)
plt.show()

plt.figure()
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s=300, c="yellow")
plt.show()

print(y_kmeans)
print(type(y_kmeans))
print("\n\ny_kmeans\n")
print(y_kmeans)

plt.figure() plt.scatter(X[y_kmeans == 0, 0],
X[y_kmeans ==
0, 1], s=80, c="red", label="Cluster 1")
```

```
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans
== 1, 1], s=80, c="blue", label="Cluster 2")
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans
== 2, 1], s=80, c="green", label="Cluster 3")
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans
== 3, 1], s=80, c="cyan", label="Cluster 4")
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans
== 4, 1], s=80, c="magenta", label="Cluster 5")
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s=300,
c="yellow", label="Centroids") plt.title("Clusters
of customers") plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)") plt.legend()
plt.show()
```

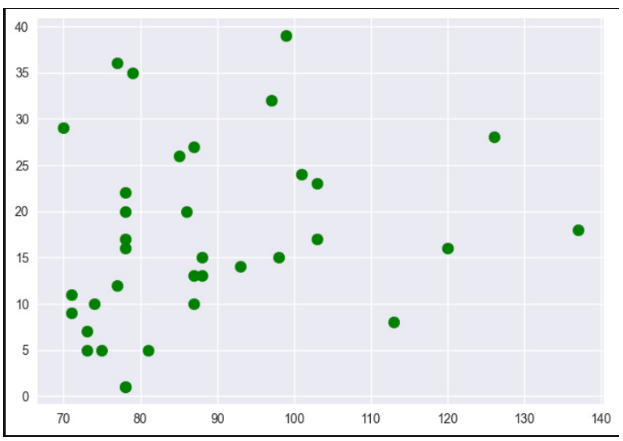
OUTPUT



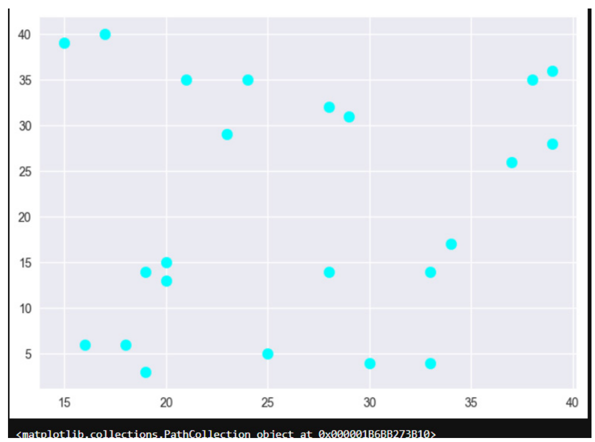
60
55
50
45
40
35
40 45 50 55 60 65 70 75



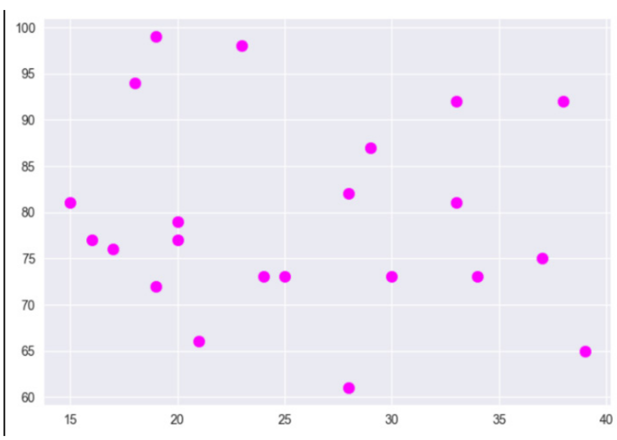
95
90
85
80
75
70
65
70 80 90 100 110 120 130 140



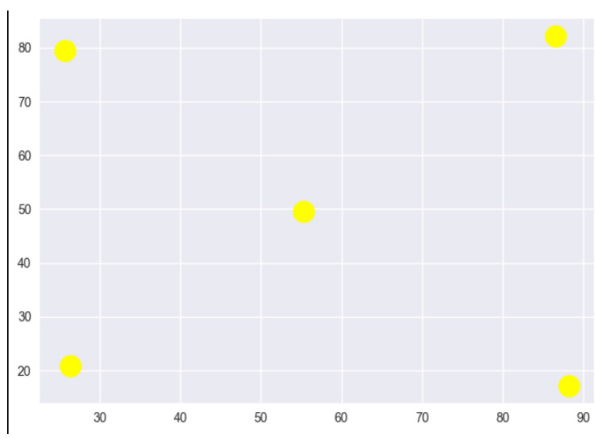
40
35
30
25
20
15
10
5
0
70 80 90 100 110 120 130 140



40
35
30
25
20
15
10
5
15 20 25 30 35 40
<matplotlib.collections.PathCollection object at 0x000001B6BB273B10>



100
95
90
85
80
75
70
65
60
15 20 25 30 35 40



80
70
60
50
40
30
20
30 40 50 60 70 80 90

```
[3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 0 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 2 1 2 1 0 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1]
<class 'numpy.ndarray'>


y_kmeans

[3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 0 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 2 1 2 1 0 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1]
```

A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL

Expt No. 9B

PROGRAM:

```
import numpy as np
import pandas as pd
from math import sqrt

# Load
data = pd.read_csv("C:\\Users\\Luqman\\Downloads\\IRIS.csv")
req_data = data.iloc[:, 1:]  # drop the first column if it's an
index/id
# Shuffle
np.random.seed(42)
shuffle_index = np.random.permutation(req_data.shape[0])
req_data = req_data.iloc[shuffle_index].reset_index(drop=True)
print(req_data.head(5))
# Train / test split train_size =
int(req_data.shape[0] * 0.7) train_df =
req_data.iloc[:train_size, :] test_df  =
req_data.iloc[train_size:, :]

train = train_df.values
test = test_df.values

y_true = test[:, -1]

print('Train_Shape:', train_df.shape)
print('Test_Shape :', test_df.shape)

# ---- KNN helpers ---- def
euclidean_distance(x_test, x_train):
distance = 0.0
    # last column is the label, exclude it for
    i in range(len(x_test) - 1): distance +=
    (x_test[i] - x_train[i]) ** 2 return
    sqrt(distance)
```

```
def get_neighbors(x_test, x_train, num_neighbors=5):
    distances = []
    for row in x_train:
        distances.append(euclidean_distance(x_test, row))
    distances = np.array(distances)
    sort_idx = distances.argsort()
    return x_train[sort_idx][:num_neighbors]

def predict_one(x_test, x_train, k=5):
    neighbors = get_neighbors(x_test, x_train, k)
    classes = [row[-1] for row in neighbors]
    return max(classes, key=classes.count)

def accuracy_score(y_true, y_pred):
    correct = sum(int(a == b) for a, b in zip(y_true, y_pred))
    return correct / len(y_true)

# Predict
k = 5
y_pred = [predict_one(x, train, k) for x in test]
print(y_pred,"\n")
# Accuracy
acc = accuracy_score(y_true, y_pred)
print('Accuracy:', acc)
# ---------------- Tables & Plots addon (robust to column names) ----------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

# Detect the label column from test_df (it's the last column in your
setup) label_col = test_df.columns[-1]

# 1) Results table (true vs
predicted) results_df =
test_df.copy()
results_df["Predicted"] = y_pred
print("\n=== Results (head) ===")
print(results_df.head())
# 2) Confusion matrix (table)
print("\n=== Confusion Matrix ===")
```

```
cm = pd.crosstab(results_df[label_col], results_df["Predicted"],
                 rownames=["Actual"], colnames=["Predicted"])
print(cm)

# 3) Classification report (table)
print("\n=== Classification Report ===")
report = classification_report(results_df[label_col], results_df["Predicted"],
output_dict=True)
report_df = pd.DataFrame(report).transpose()
print(report_df.round(3))

# 4) Accuracy vs K plot (K = 1..15)
Ks = list(range(1, 16))
accs = []
for kk in Ks:
    y_pred_k = [predict_one(x, train, kk) for x in test]
    correct = sum(int(a == b) for a, b in zip(y_true, y_pred_k))
    accs.append(correct / len(y_true))

plt.figure() plt.plot(Ks, accs,
marker='o') plt.title("Accuracy vs K
(KNN on Iris)") plt.xlabel("K")
plt.ylabel("Accuracy") plt.grid(True)
plt.show()



# 5) 2D scatter: choose petal columns if present, else first 2 feature columns
candidate_x = ["PetalLengthCm", "petal_length", "PetalLength", "petal
length"] candidate_y = ["PetalWidthCm",  "petal_width",  "PetalWidth",
"petal width"]
def pick_first_present(cands, cols):
    for c in cands:
        if c in cols:
            return c
    return None

xcol = pick_first_present(candidate_x,
results_df.columns) ycol =
pick_first_present(candidate_y, results_df.columns)
# If not found, fall back to the first two non-label, non-predicted
columns if xcol is None or ycol is None:
```

```
    feature_cols = [c for c in results_df.columns if c not in [label_col,
    "Predicted"]] if len(feature_cols) >= 2: xcol, ycol = feature_cols[:2]


if xcol is not None and ycol is not None:
    plt.figure()
    # plot each class separately
    for cls in results_df[label_col].unique():
        part = results_df[results_df[label_col] == cls]
        plt.scatter(part[xcol].values, part[ycol].values, label=str(cls), alpha=0.8)
    # mark misclassified
    wrong = results_df[results_df[label_col] != results_df["Predicted"]]
    if not wrong.empty:
        plt.scatter(wrong[xcol].values, wrong[ycol].values, marker='x', s=100)
    plt.title(f"{xcol} vs {ycol} (circles=true, X=misclassified)")
    plt.xlabel(xcol)
    plt.ylabel(ycol)
    plt.legend()
    plt.grid(True)
    plt.show()
else:
print("\n[Note] Skipping scatter: couldn't identify two feature
columns.") # ----------------- end addon -----------------

#-- end addon -----------------
```

OUTPUT:

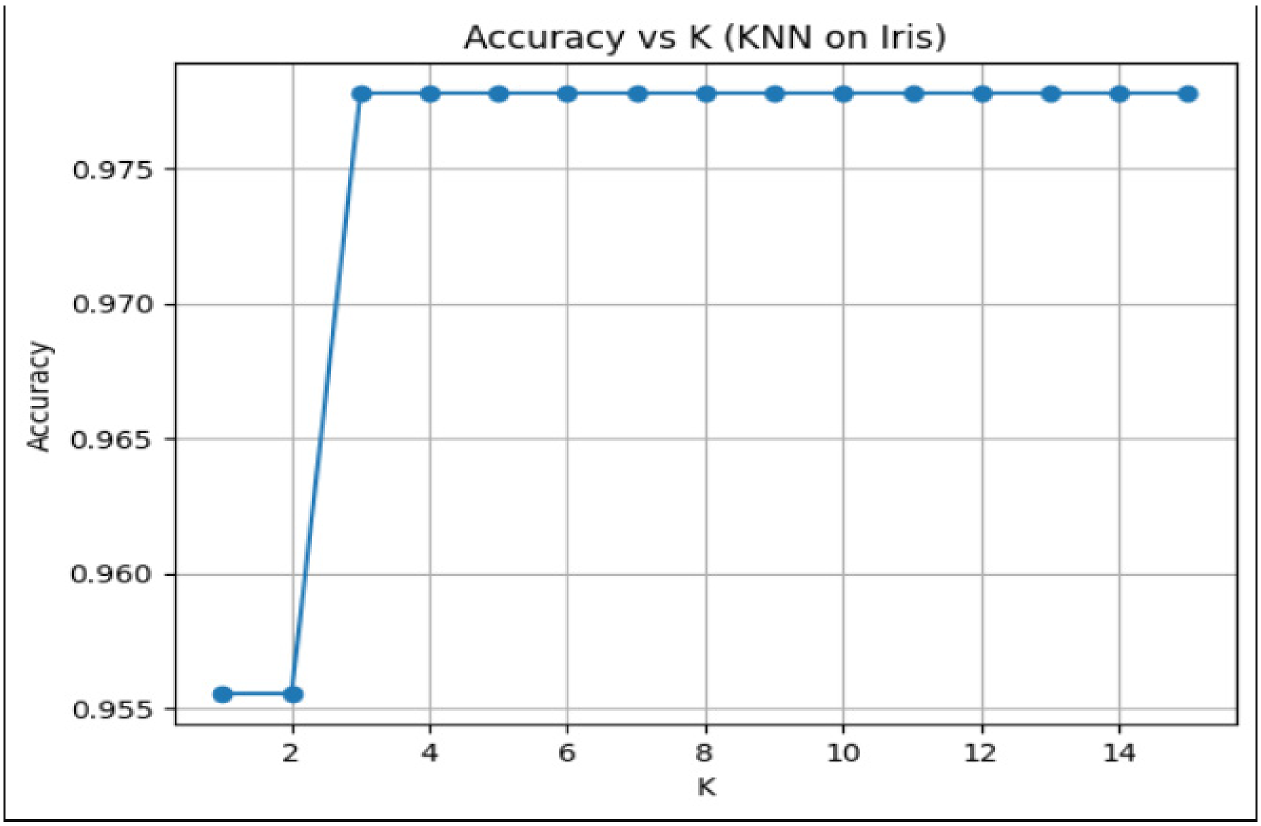
```
Train_Shape: (105, 4)
Test_Shape : (45, 4)
```

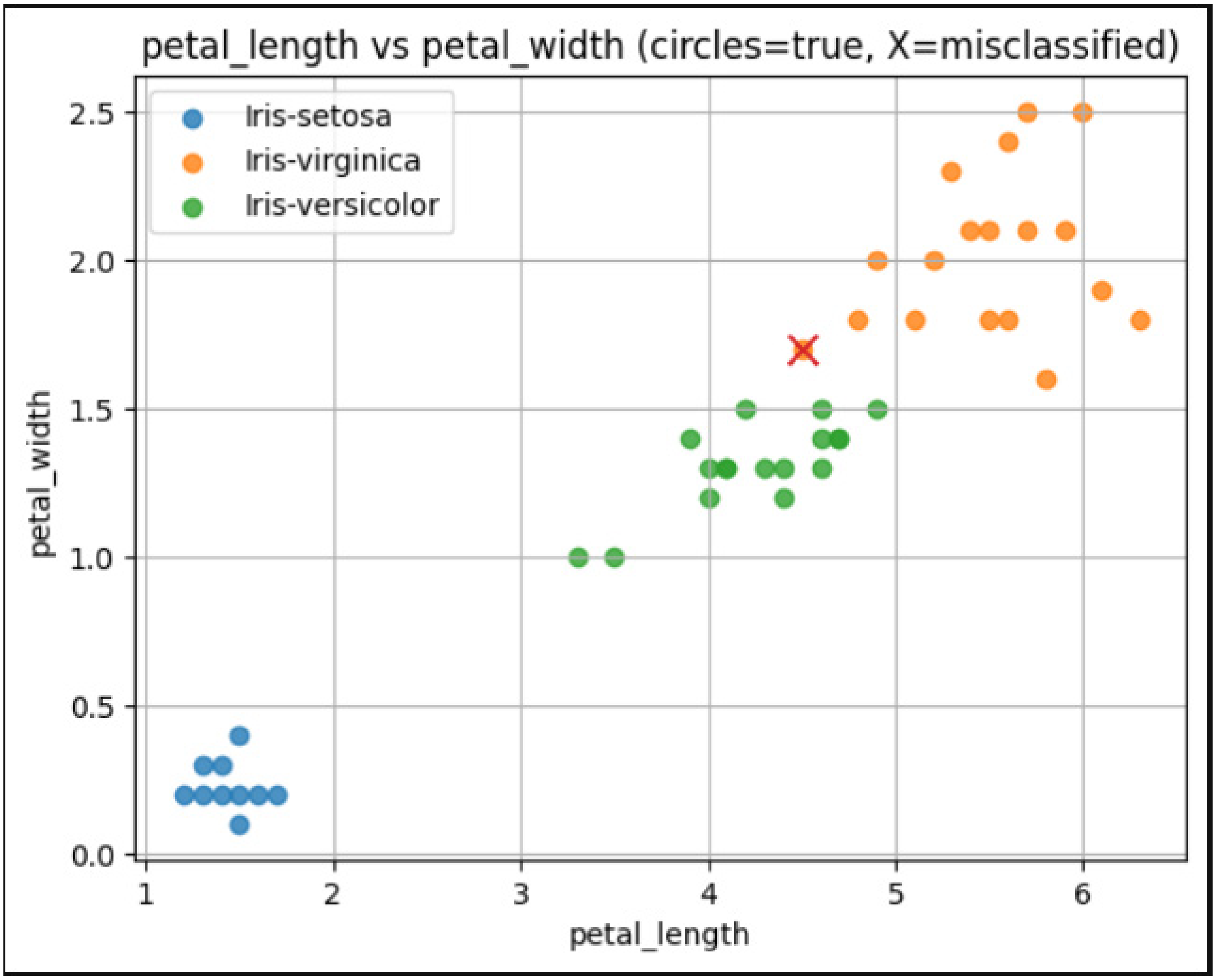
```
['Iris-setosa', 'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor', 'Iris-vir
ginica', 'Iris-setosa', 'Iris-virginica', 'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-vir
ginica', 'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica', 'Iris-virginica', 'Iris-versicolor', 'I
ris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-se
tosa', 'Iris-versicolor', 'Iris-virginica']
```

```
Accuracy: 0.9777777777777777

=== Results (head) ===
     sepal_width  petal_length  petal_width          species        Predicted
105          3.4           1.4          0.3      Iris-setosa      Iris-setosa
106          3.0           5.5          2.1   Iris-virginica   Iris-virginica
107          3.3           6.0          2.5   Iris-virginica   Iris-virginica
108          3.2           1.3          0.2      Iris-setosa      Iris-setosa
109          2.9           4.7          1.4  Iris-versicolor  Iris-versicolor
```

```
=== Confusion Matrix ===
Predicted        Iris-setosa  Iris-versicolor  Iris-virginica
Actual
Iris-setosa               10                0               0
Iris-versicolor            0               17               0
Iris-virginica             0                1              17

=== Classification Report ===
                 precision  recall  f1-score  support
Iris-setosa          1.000   1.000     1.000   10.000
Iris-versicolor      0.944   1.000     0.971   17.000
Iris-virginica       1.000   0.944     0.971   18.000
accuracy             0.978   0.978     0.978    0.978
macro avg            0.981   0.981     0.981   45.000
weighted avg         0.979   0.978     0.978   45.000
```

	sepal_width	petal_length	petal_width	species
0	2.8	4.7	1.2	Iris-versicolor
1	3.8	1.7	0.3	Iris-setosa
2	2.6	6.9	2.3	Iris-virginica
3	2.9	4.5	1.5	Iris-versicolor
4	2.8	4.8	1.4	Iris-versicolor




petal_length vs petal_width (circles=true, X=misclassified)
Iris-setosa
Iris-virginica
Iris-versicolor
petal_width
petal_length
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6