IE406 Machine Learning

Lab Assignment - 03

Group 16

Student ID: 201801046 Name: Ravi Patel
Student ID: 201801057 Name: Ayan Khokhar
Student ID: 201801432 Name: Prakhar Maheshwari
Student ID: 201801452 Name: Jaydeep Machhi
Student ID: 201801459 Name: Aakash Panchal

Question 1

In this question we will use the data given in file "Social_Network_Ads.csv" which is a categorical data-set to determine whether a user purchased a product or not by using three features to determine user's decision. Visualize the data by 3D plotting features using different colors for label 0 and 1. Use data in files "Social_Network_Ads.csv" to perform logistic regression by implementing logistic function and with available library function and compare your results. Use 90% data points from each set for training and remaining 10% for testing the accuracy of classification. Using confusion matrix find accuracy, precision, F1 score and recall.

Answer

code

```
# Scaling the dataset values from 0 to 1
scaler = MinMaxScaler()
dropped_df = df.drop(['User ID'], axis = 1)
column_names = dropped_df.columns

scaled_df = scaler.fit_transform(dropped_df)
scaled_df = pd.DataFrame(scaled_df, columns= column_names)
x = scaled_df[[ 'Gender', 'Age', 'EstimatedSalary']].values
y = scaled_df['Purchased'].values
print(scaled_df.head()) # Scaled Dataset
```

```
# Logistic Regression implementation from scratch
class MyLogisticRegression:
    def __init__(self, alpha=0.01, iterations=100000, fit_intercept=True):
        self.alpha = alpha  #Learning Rate
```

```
        self.iterations = iterations #Finite Iterations for algorithm convergence
        self.fit_intercept = fit_intercept  #Set if to true if intercept needed

    def _conc_intercept(self, X):  # Append Intercept
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def _logistic(self, z): # Logistic(Sigmoid) Function for Logistic Regression
        return 1 / (1 + np.exp(-z))

    def _hypothesis(self, h, y): # Binary Cross Entropy Function
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):  # Using Gradient Descent to calculate minima of the function
        if self.fit_intercept:
            X = self._conc_intercept(X)

        self._features = np.zeros(X.shape[1])
        for i in range(self.iterations):
            z = np.dot(X, self._features)
            h = self._logistic(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self._features -= self.alpha * gradient


    def predict(self, X, threshold = 0.5):  # Prediction Function if prob >= thresh then one class else another class
        if self.fit_intercept:
            X = self._conc_intercept(X)
        return self._logistic(np.dot(X, self._features)) >= threshold
```

```
plt.figure(figsize = (8,8))
ax = plt.axes(projection ='3d')

df0 = df[df.Purchased == 0]
df1 = df[df.Purchased == 1]

# Classifying Purchased as blue and not purchased as red data points in form of a scatter plot
ax.scatter(df0['Gender'].values, df0['Age'].values,df0['EstimatedSalary'].values, c = 'red')
ax.scatter(df1['Gender'].values, df1['Age'].values,df1['EstimatedSalary'].values, c = 'blue')
plt.legend(['Not Purchased', 'Purchased'])
ax.set_title('Scatter plot of Features')
ax.set_xlabel('Gender')
ax.set_ylabel('Age')
ax.set_zlabel('EstimatedSalary')
```

```
regressor = MyLogisticRegression()
```

```
X = x
# print(X,y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=4)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
score = accuracy_score(y_pred, y_test)
print(score)

confusion_mat = confusion_matrix(y_test, y_pred)

# print(y_predicted)
graph_cm_matrix = pd.DataFrame(confusion_mat, columns=['Actual True:1', '
    Actual False:0'],
                               index=['Predicted Positive:1', 'Predicted
    Negative:0'])

sns.heatmap(graph_cm_matrix, annot=True, fmt='d')
```

```
# Function to calculate Classification metrics from the confusion matrix
def calculate_classification_metrics(confusion_mat):
   TP = confusion_mat[0,0]
   FP = confusion_mat[0,1]
   FN = confusion_mat[1,0]
   TN = confusion_mat[1,1]
   accuracy = (TP + TN)/(TP + TN + FP + FN)
   precision = (TP)/(TP + FP)
   recall = (TP)/(TP + FN)
   F1 = (2 * precision * recall)/(precision + recall)
   print(f'Accuracy: {accuracy}')
   print(f'Precision: {precision}')
   print(f'Recall: {recall}')
   print(f'F1 Score: {F1}')
```

```
#Using scitkit-learn's implementation of Logistic Regression
X = x

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=4)

clf = LogisticRegression(random_state=10, max_iter = 100000).fit(X_train,
    y_train)
y_predicted = clf.predict(X_test)
confusion_mat = confusion_matrix(y_test, y_predicted)

# print(y_predicted)
graph_cm_matrix = pd.DataFrame(confusion_mat, columns=['Actual True:1', '
    Actual False:0'],
                               index=['Predicted Positive:1', 'Predicted
    Negative:0'])

sns.heatmap(graph_cm_matrix, annot=True, fmt='d')
```

Result

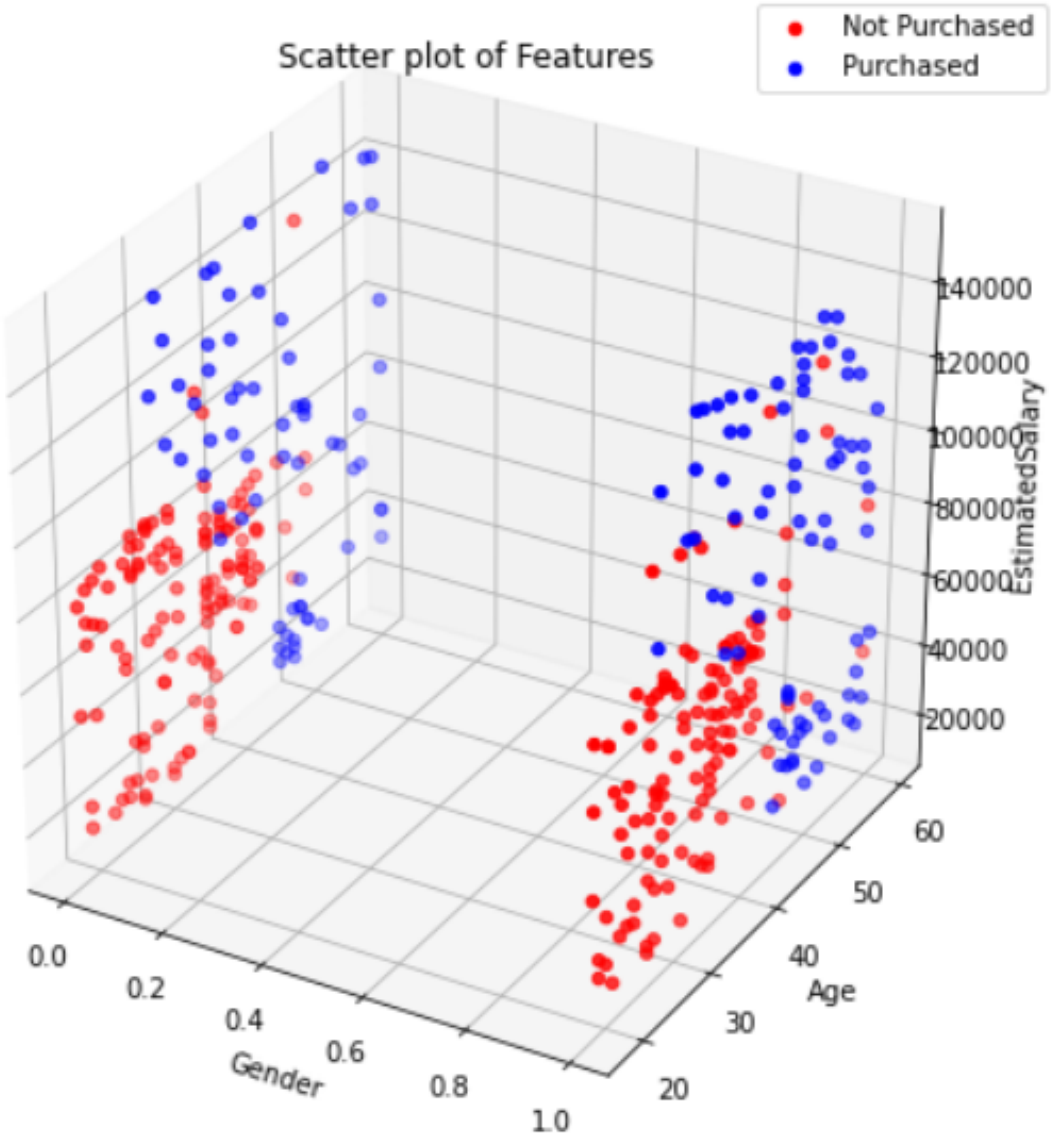

Figure 1: 3D plot for the data-set

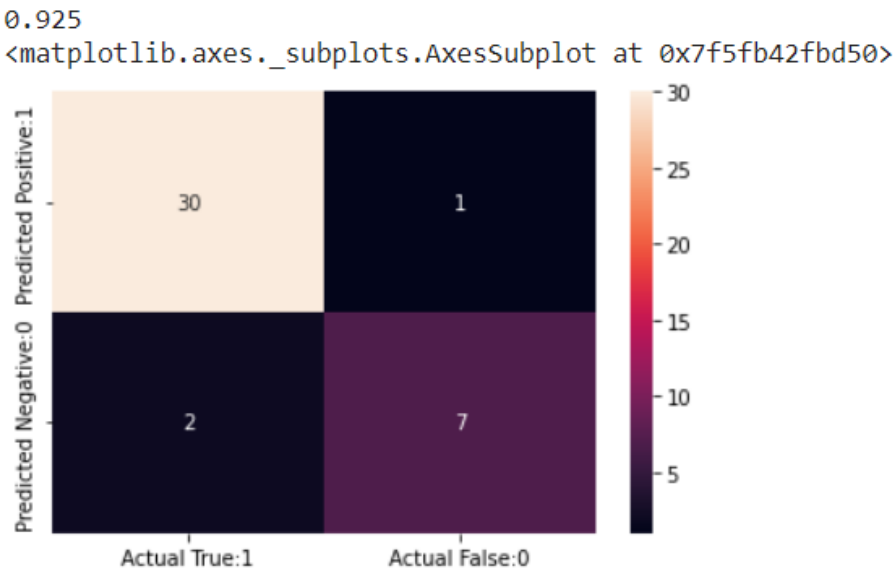

Figure 2: Confusion matrix for our own logistic function

Accuracy: 0.925
Precision: 0.967741935483871
Recall: 0.9375
F1 Score: 0.9523809523809523

Figure 3: Accuracy, precision, recall and F1 Score for our own logistic function

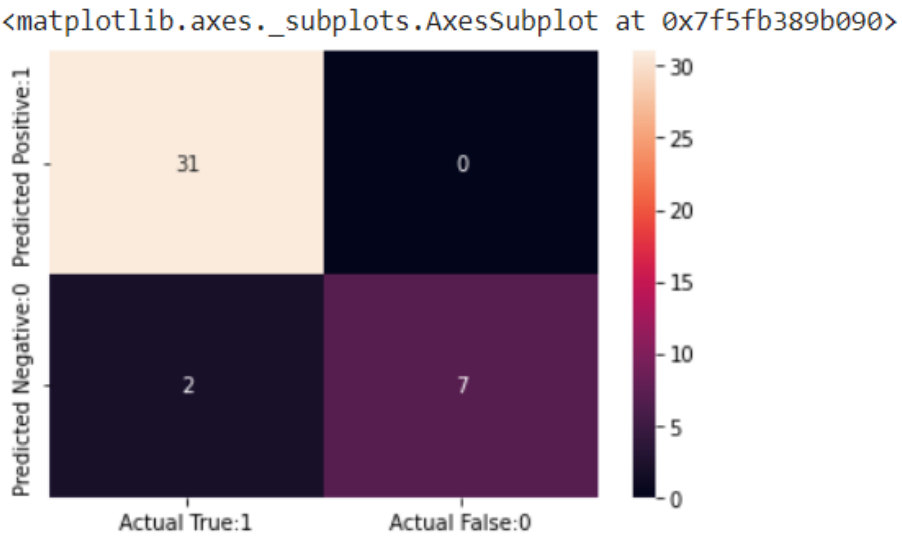

Figure 4: Confusion matrix for ScikitLearn's regression function

```
Accuracy: 0.95
Precision: 1.0
Recall: 0.9393939393939394
F1 Score: 0.96875
```

Figure 5: Accuracy, precision, recall and F1 Score for ScikitLearn's function

Observation/ Justification

- We have scaled the values of the dataset from 0 to 1 using the MinMaxScaler function of sci-kit-learn. To improvise the classification performance.
- By changing the learning rate(alpha) to a smaller value and iterations to a larger number can increase the accuracy of our algorithm.
- Moreover, gradient descent is not the most optimal method for calculating the minima. By making use of more sophisticated algorithms we can increase the performance of our implementation.
- From the above analysis of the confusion matrix of our own implementation of Logistic Regression with that of the Scikit Learn's Implementation. It can be clearly seen that both perform pretty well in terms of binary classification of Purchased/Not Purchased.

- Moreover from the analysis of classification metrics, our implementation performs pretty well in terms of accuracy, precision, recall and F1 score.

Question 2

You will work with a widely used Iris dataset. The Iris Dataset contains four features (sepal length, sepal width, petal length, and petal width) of 50 samples of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor). Plot features' histogram. Compute pdf and compare it with histogram. perform the exploratory data analysis by plotting the basic statistics like mean, median, min, and max value of each feature (sepal and petal lengths and widths) for each of the three classes (setosa, virginica, and versicolor).

Answer

code

```
iris_df = pd.read_csv('/content/Iris.csv')
print(iris_df.head())

print(iris_df.shape)
```

Listing 1: Loading data

```
#Plotting probability density function and histogram for each feature
features = iris_df.columns

for i in range(1,5):
   grid = sns.FacetGrid(iris_df, hue='Species', height=5)
   grid1 = sns.FacetGrid(iris_df, hue='Species', height=5)
   grid.map(sns.kdeplot, features[i]).add_legend();
   grid1.map(sns.histplot, features[i]).add_legend();
```

Listing 2: Plotting probability density function and histogram for each feature

```
# Mapping Species name to integer for prediction
species = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica' : 2}
iris_df['Species'] = [species[item] for item in iris_df['Species']]

# Performing Exploratory Data Analysis on the Dataset
df_dict = {g: d for g, d in iris_df.groupby('Species')}

SepalL = []
SepalW = []
PetalL = []
PetalW = []

x = list(species.keys())
for i in range(3):
   SepalL.append(df_dict[i].mean()['SepalLengthCm'])
   SepalW.append(df_dict[i].mean()['SepalWidthCm'])
   PetalL.append(df_dict[i].mean()['PetalLengthCm'])
```

```
    PetalW.append(df_dict[i].mean()['PetalWidthCm'])


print(SepalL)
fig = plt.figure()
fig.set_figwidth(10)
fig.set_figheight(8)
plt.subplot(2,2,1)
plt.bar(x, SepalL)
plt.title('Mean of SepalLength')
plt.subplot(2,2,2)

plt.bar(x, SepalW)

plt.title('Mean of SepalWidth')
plt.subplot(2,2,3)

plt.bar(x, PetalL)

plt.title('Mean of PetalLength')
plt.subplot(2,2,4)
plt.bar(x, SepalW)

plt.title('Mean of PetalWidth')
```

Listing 3: Mean for each features

```

SepalL = []
SepalW = []
PetalL = []
PetalW = []
for i in range(3):
    SepalL.append(df_dict[i].median()['SepalLengthCm'])
    SepalW.append(df_dict[i].median()['SepalWidthCm'])
    PetalL.append(df_dict[i].median()['PetalLengthCm'])
    PetalW.append(df_dict[i].median()['PetalWidthCm'])

print(SepalL)
fig = plt.figure()
fig.set_figwidth(10)
fig.set_figheight(8)
plt.subplot(2,2,1)
plt.bar(x, SepalL)
plt.title('Median of SepalLength')

plt.subplot(2,2,2)
plt.bar(x, SepalW)
plt.title('Median of SepalWidth')

plt.subplot(2,2,3)
plt.bar(x, PetalL)
plt.title('Median of PetalLength')

```

```
plt.subplot(2,2,4)
plt.bar(x, SepalW)
plt.title('Median of PetalWidth')
```

Listing 4: Median for each features

```


SepalL = []
SepalW = []
PetalL = []
PetalW = []
for i in range(3):
    SepalL.append(df_dict[i].min()['SepalLengthCm'])
    SepalW.append(df_dict[i].min()['SepalWidthCm'])
    PetalL.append(df_dict[i].min()['PetalLengthCm'])
    PetalW.append(df_dict[i].min()['PetalWidthCm'])

print(SepalL)
fig = plt.figure()
fig.set_figwidth(10)
fig.set_figheight(8)
plt.subplot(2,2,1)
plt.bar(x, SepalL)
plt.title('Minimum of SepalLength')

plt.subplot(2,2,2)
plt.bar(x, SepalW)
plt.title('Minimum of SepalWidth')

plt.subplot(2,2,3)
plt.bar(x, PetalL)
plt.title('Minimum of PetalLength')

plt.subplot(2,2,4)
plt.bar(x, SepalW)
plt.title('Minimum of PetalWidth')
```

Listing 5: Min value for each features

```


SepalL = []
SepalW = []
PetalL = []
PetalW = []
for i in range(3):
    SepalL.append(df_dict[i].max()['SepalLengthCm'])
    SepalW.append(df_dict[i].max()['SepalWidthCm'])
    PetalL.append(df_dict[i].max()['PetalLengthCm'])
    PetalW.append(df_dict[i].max()['PetalWidthCm'])

print(SepalL)
fig = plt.figure()
```

```
fig.set_figwidth(10)
fig.set_figheight(8)
plt.subplot(2,2,1)
plt.bar(x, SepalL)
plt.title('Maximum of SepalLength')

plt.subplot(2,2,2)
plt.bar(x, SepalW)
plt.title('Maximum of SepalWidth')

plt.subplot(2,2,3)
plt.bar(x, PetalL)
plt.title('Maximum of PetalLength')

plt.subplot(2,2,4)
plt.bar(x, SepalW)
plt.title('Maximum of PetalWidth')
```

Listing 6: Max value for each features

Result

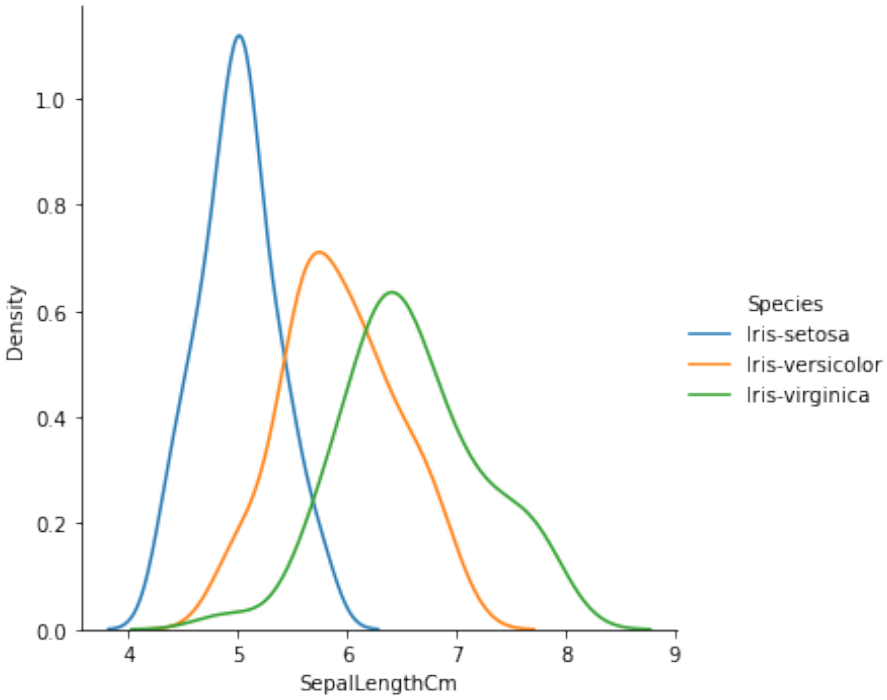

Figure 6: PDF of Sepal Length

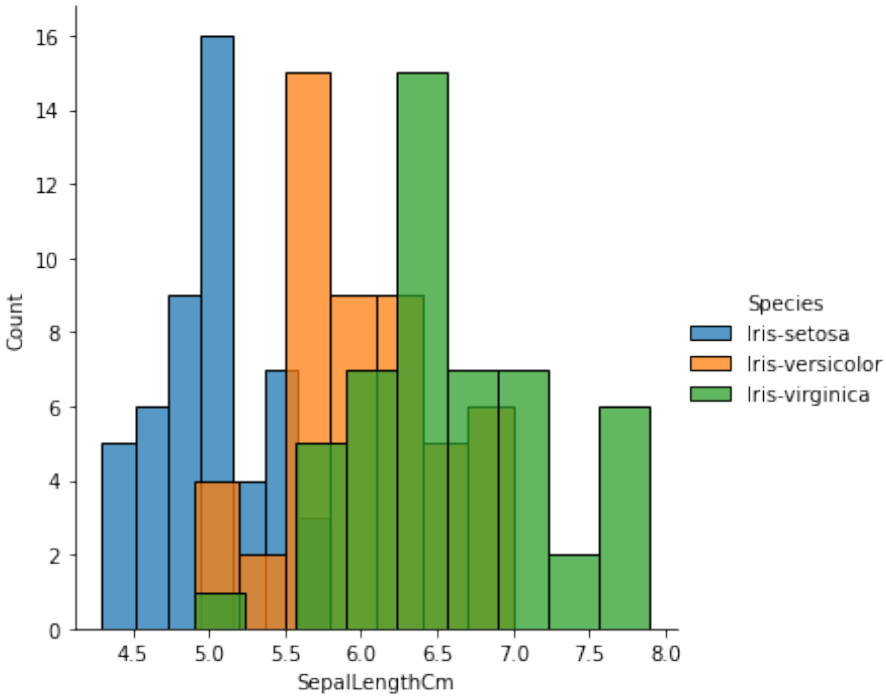

Figure 7: Histogram for Sepal Length

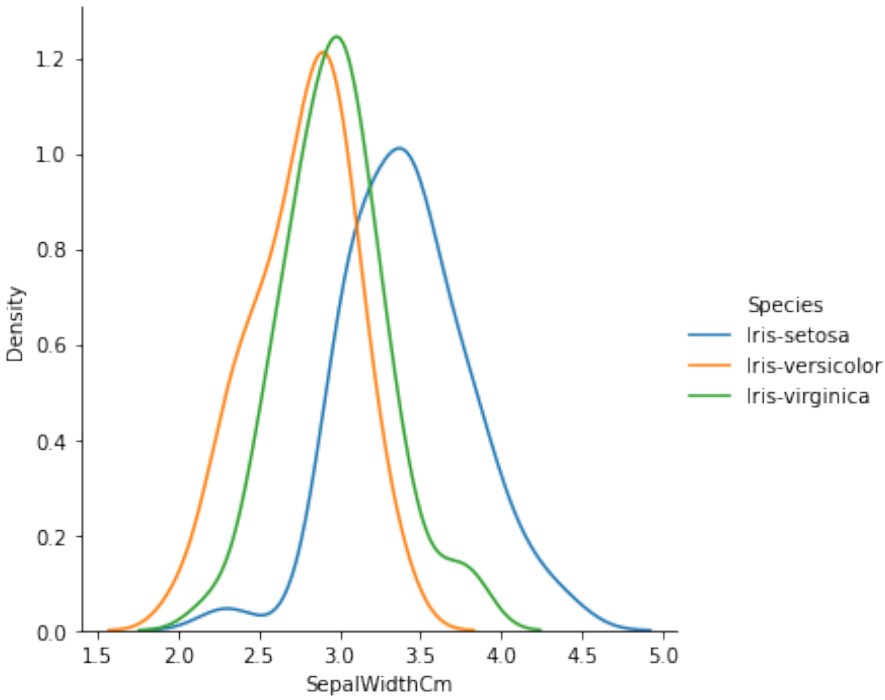

Figure 8: PDF for Sepal Width

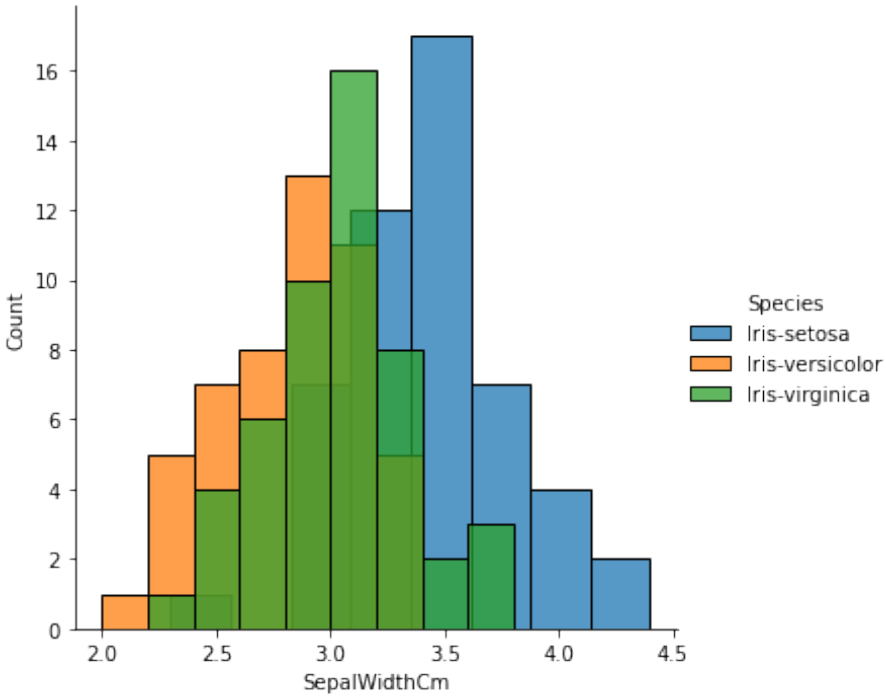

Figure 9: Histogram for Sepal Width

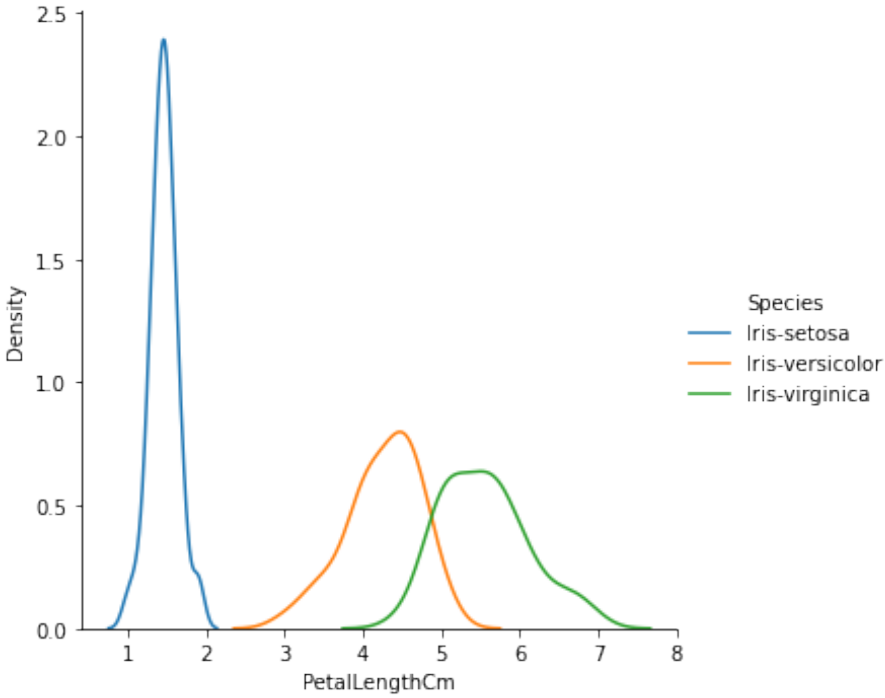

Figure 10: PDF for Petal Length

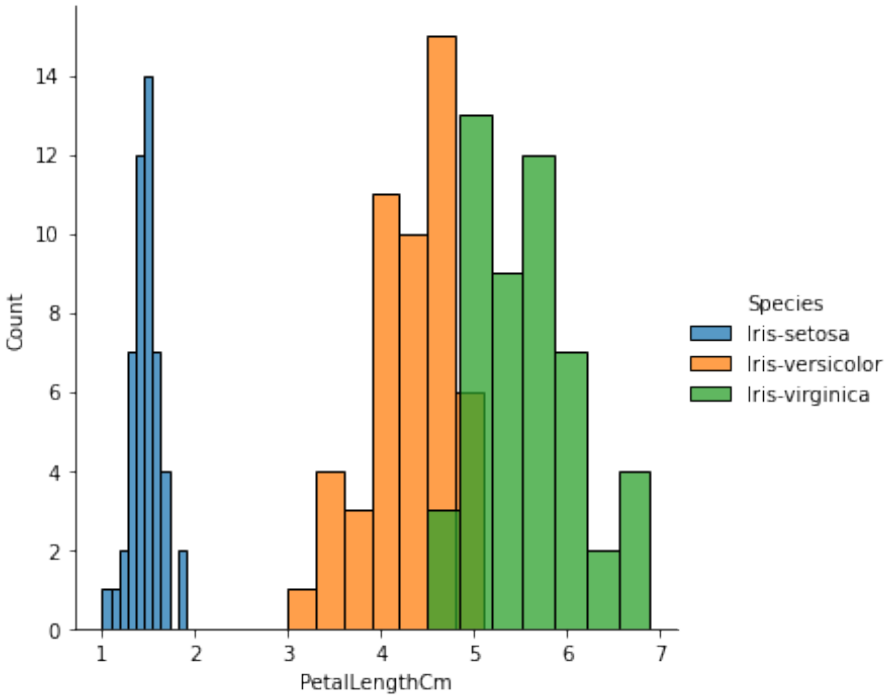

Figure 11: Histogram for Petal Length

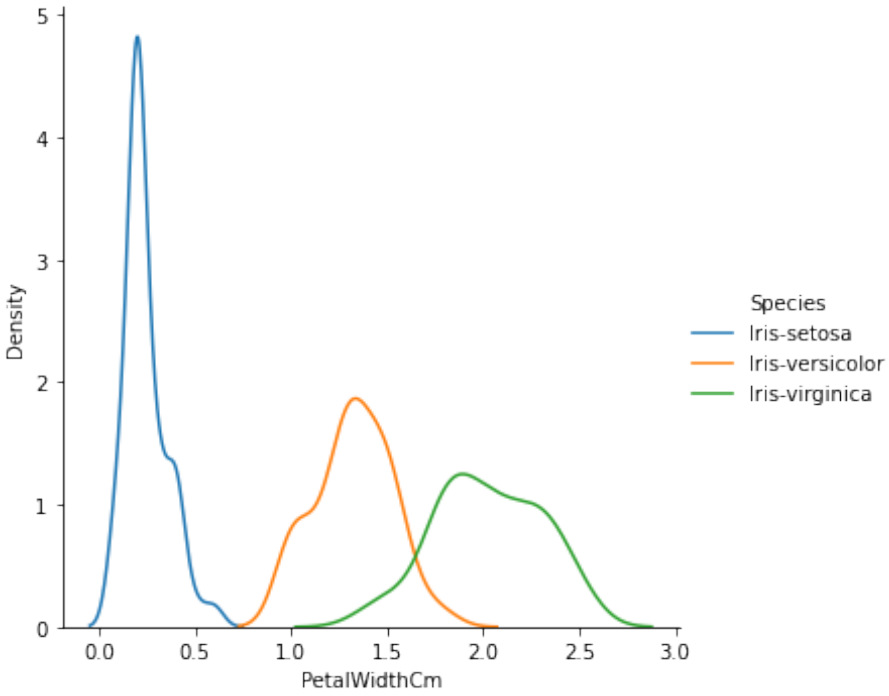

Figure 12: Probability for Petal Width

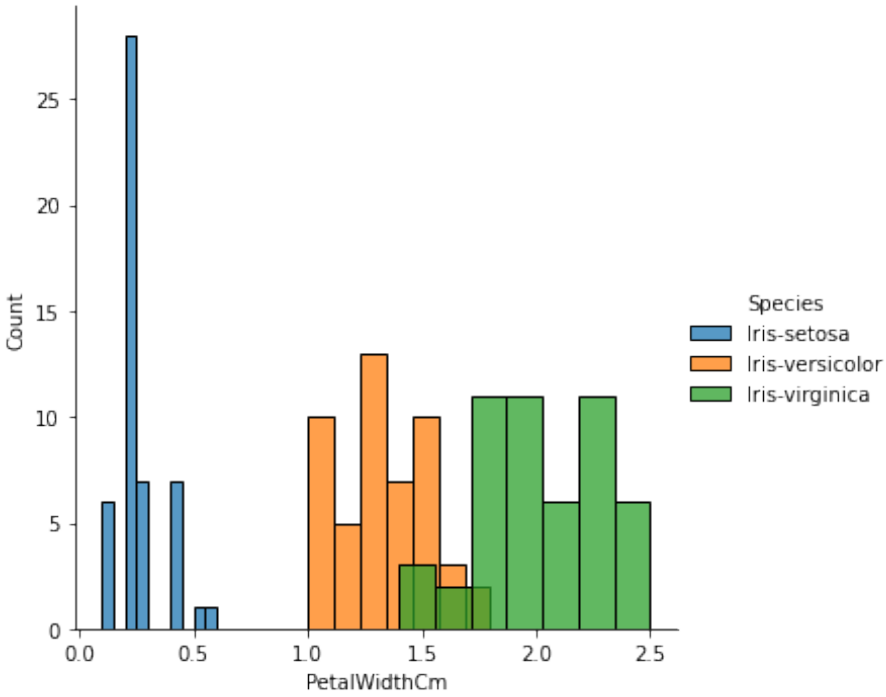

Figure 13: Histogram for Petal Width

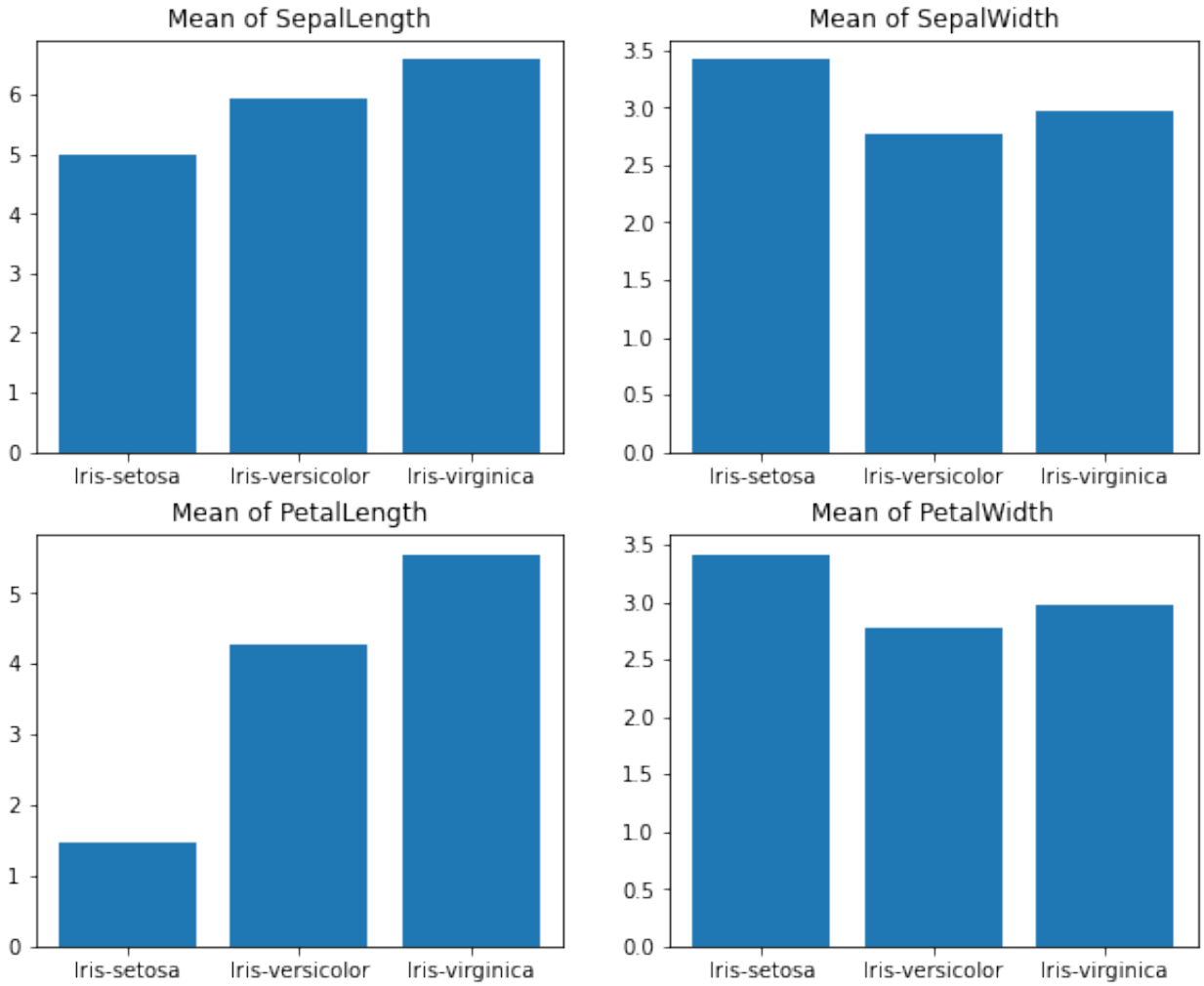

Figure 14: Mean of features

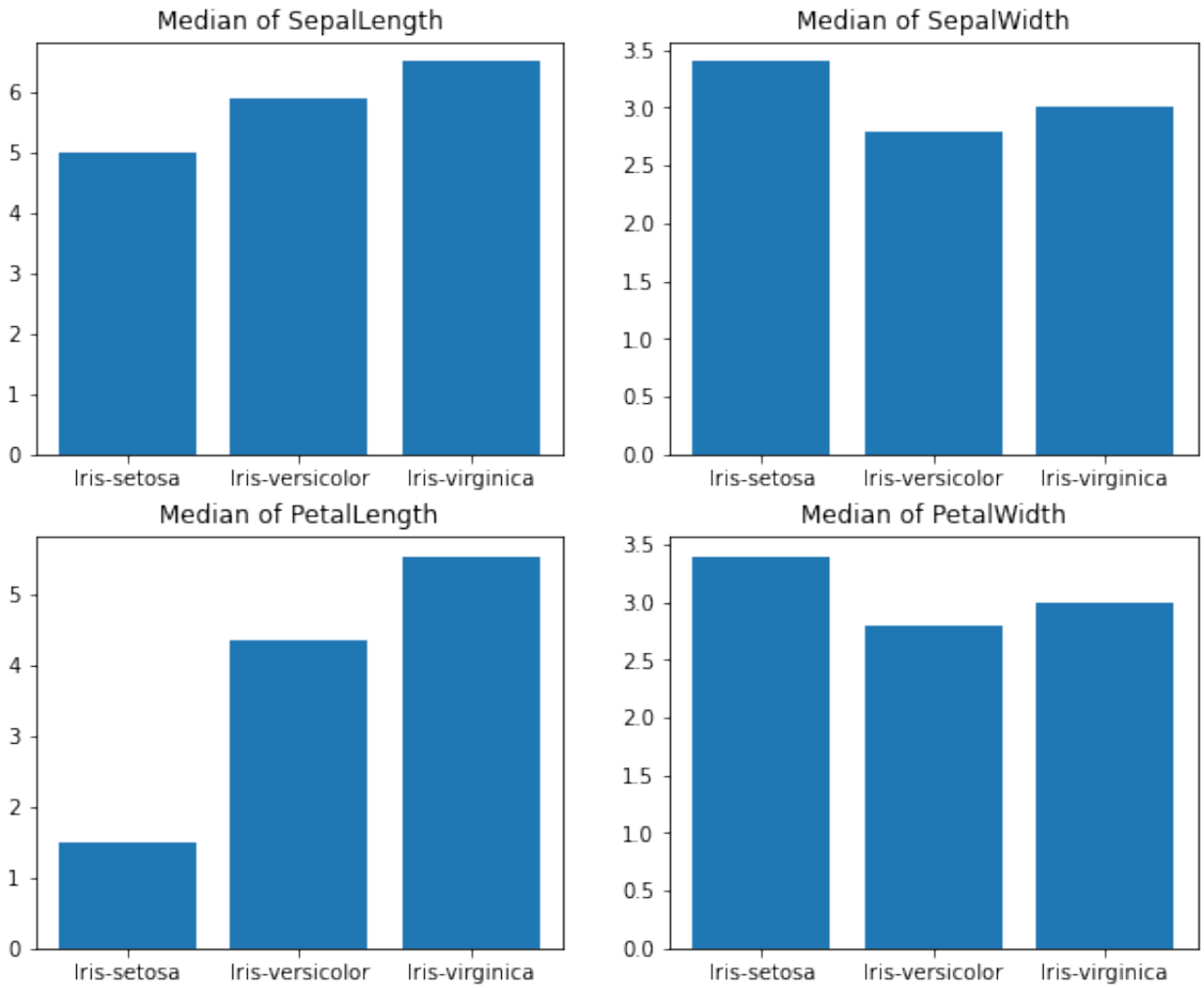

Figure 15: Median of features

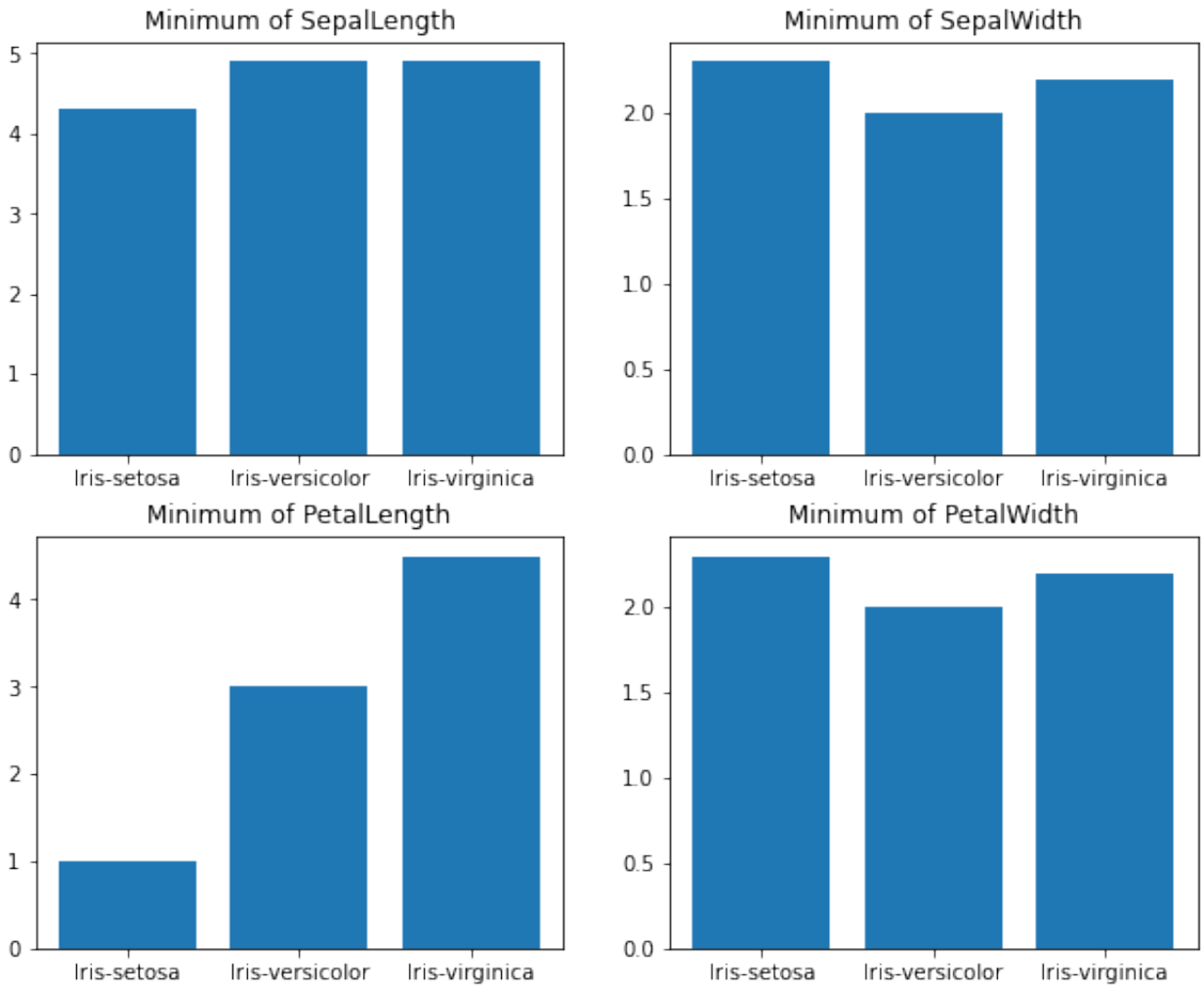

Figure 16: Min for features

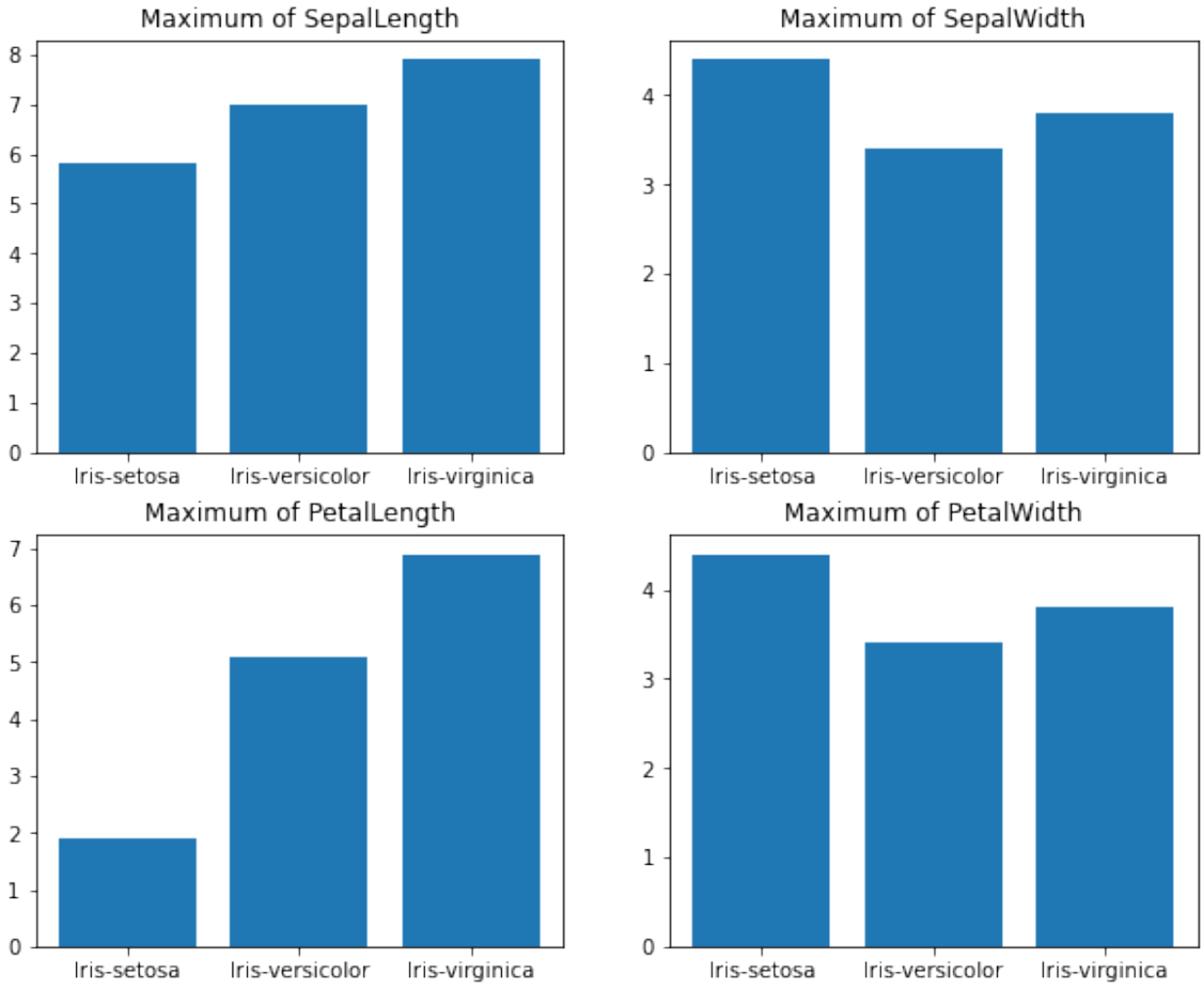

Figure 17: Max for features

Observation/ Justification

PDF for each of the features follows a Gaussian distribution. Looking at the PDF we can get a high level overview of range of earh feature depending on the species.

Histogram also serves the same purpose as the PDF.

Below is an ordering based on decreasing order of mean for each features.
Sepal Length: Iris virginica, Iris versicolor, Iris setosa
Sepal Width: Iris setosa, Iris versicolor, Iris virginica
Petal Length: Iris virginica, Iris versicolor, Iris setosa
Petal Width: Iris setosa, Iris virginica, Iris versicolor

Below is an ordering based on decreasing order of median for each features.
Sepal Length: Iris virginica, Iris versicolor, Iris setosa
Sepal Width: Iris setosa, Iris virginica, Iris versicolor

Petal Length: Iris virginica, Iris versicolor, Iris setosa
Petal Width: Iris setosa, Iris virginica, Iris versicolor

Below is an ordering based on decreasing order of min for each features.
Sepal Length: Iris virginica, Iris versicolor, Iris setosa
Sepal Width: Iris setosa, Iris virginica , Iris versicolor
Petal Length: Iris virginica, Iris versicolor, Iris setosa
Petal Width: Iris setosa, Iris virginica, Iris versicolor

Below is an ordering based on decreasing order of max for each features.
Sepal Length: Iris virginica, Iris versicolor, Iris setosa
Sepal Width: Iris setosa, Iris virginica, Iris versicolor
Petal Length: Iris virginica, Iris versicolor, Iris setosa
Petal Width: Iris setosa, Iris virginica, Iris versicolor

Question 3

Visualize the data in the Iris Dataset by considering maximum combinations of two features in a 2D plot. Use red, green, and blue colors for labeling the three classes: Iris setosa, Iris virginica, and Iris versicolor, respectively. Comment on whether any two classes among the three can be separated by a line? Report your observations for each case.

Answer

code

```
df = pd.read_csv('/content/Iris.csv')

species = {'Iris-setosa' : 0, 'Iris-versicolor' : 1, 'Iris-virginica' : 2}
df['Species'] = [species[item] for item in df['Species'] ]
print(df.head())

color_map = {0 : 'r', 1 : 'b', 2 : 'g'}
sps = df['Species'].unique()
rev_species = { 0 : 'Iris-setosa',  1 : 'Iris-versicolor' , 2 : 'Iris-
    virginica'}
marker_map = {0:'o', 1:'x', 2:'v'}

feature = df.columns[1:-1]
plt.figure(figsize = (14,14))
print(feature)
print(sps)

p = 1
for i in range(4):
    for j in range(i+1, 4):

            plt.subplot(3, 2, p)
```

```
        for k in sps:

            plt.scatter(df[df['Species'] == k][feature[i]], df[df['Species'] == k][feature[j]], c = color_map[k], marker = marker_map[k], label = rev_species[k]) # Use this if you want to use markers
            # plt.scatter(df[df['Species'] == k][feature[i]], df[df['Species'] == k][feature[j]], c = color_map[k], label = k) # Use this if don't want to use markers
            plt.xlabel(feature[i])
            plt.ylabel(feature[j])
            plt.legend()

        p += 1

plt.show()
```

Listing 7: Combination of two features and their separation

Result

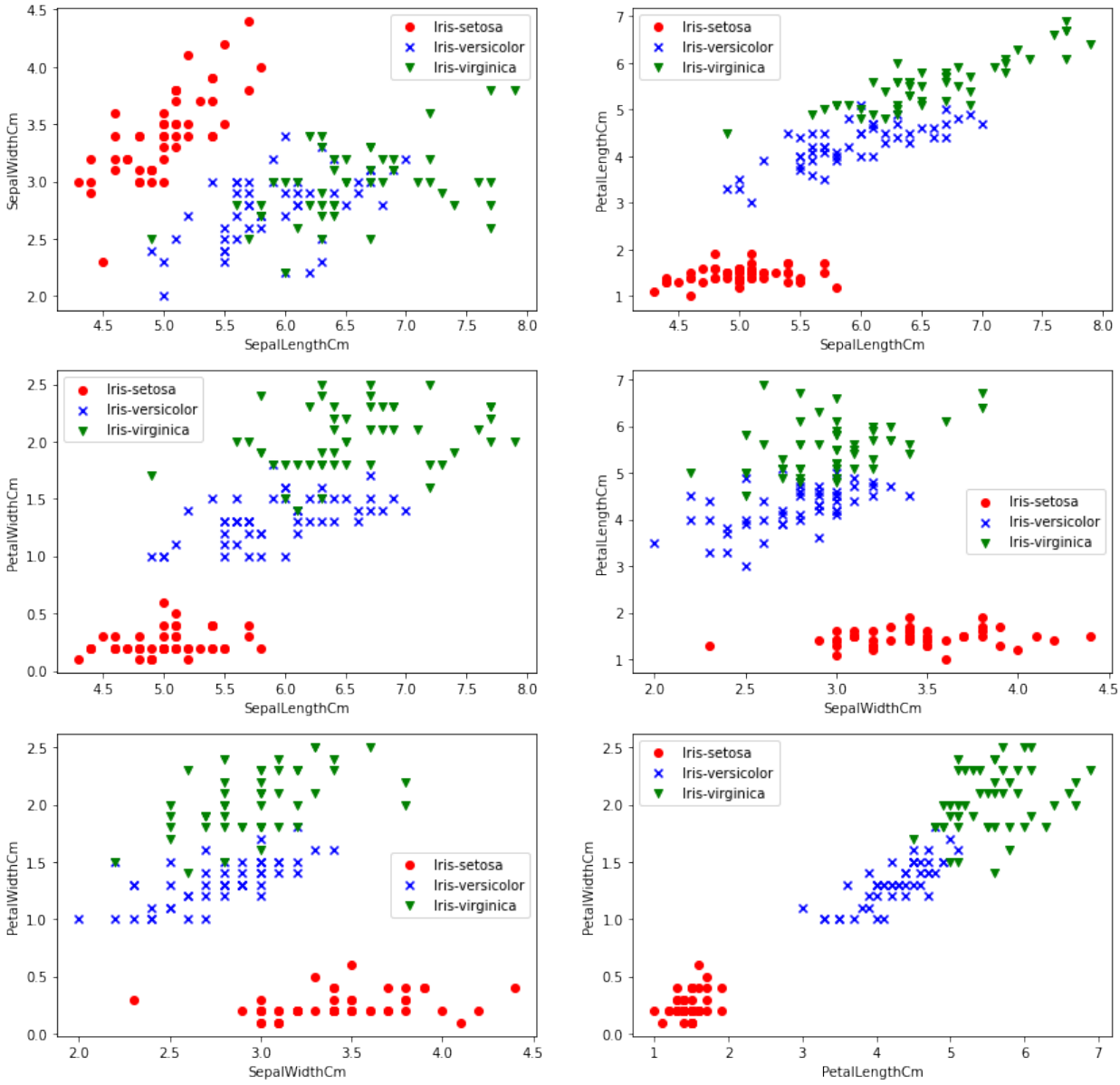

Figure 18: Combination of two features and their separation

Observation/ Justification

We can clearly see from the figure that using one vs all method for separation, we can easily separate Iris Setosa from the other two. Infact, Iris verginica can also be separated from other two but with some error in separation. But, it is almost impossible to separate Iris - versicolor from other two using line as a boundry. For this we need some complex boundaries(i.e. circle).

Question 4

Perform logistic regression on IRIS Dataset and plot confusion matrix. Using confusion matrix find accuracy, precision, F1 score and recall.

Answer

code

```
iris_df = pd.read_csv('/content/Iris.csv')
print(iris_df.head())
species = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica' : 2}
iris_df['Species'] = [species[item] for item in iris_df['Species']]
dropped_df = iris_df.drop(['Id'], axis=1)
column_names = dropped_df.columns
scaled_df = scaler.fit_transform(dropped_df)
scaled_df = pd.DataFrame(scaled_df, columns =column_names)
```

Listing 8: Q4: Loading data

```
species = {0: 0,0.5: 1, 1 : 2}
scaled_df['Species'] = [species[item] for item in scaled_df['Species']]
X = scaled_df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', '
    PetalWidthCm']]
y = scaled_df['Species']
```

Listing 9: Q4: Extracting useful data

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=2)

clf = LogisticRegression(random_state=10, n_jobs=1).fit(X_train, y_train)
y_predicted = clf.predict(X_test)
confusion_mat = confusion_matrix(y_test, y_predicted)

# print(y_predicted)
graph_cm_matrix = pd.DataFrame(confusion_mat, columns=['Iris-setosa: 0','
    Iris-versicolor: 1', 'Iris-virginica : 2'],
                               index=['Iris-setosa: 0','Iris-versicolor:
    1', 'Iris-virginica : 2'])

sns.heatmap(graph_cm_matrix, annot=True, fmt='d')
```

Listing 10: Q4: Confusion Matrix

```
print('Macro')
# Classification Report
print(classification_report(y_test, y_predicted))

macro = precision_recall_fscore_support(y_test, y_predicted, average='
    macro')
print(f'Precision = {macro[0]}\nRecall = {macro[1]}\nF1Score = {macro[2]}'
    )

```

```
print('\nMicro')
micro = precision_recall_fscore_support(y_test, y_predicted, average='
    micro')
print(f'Precision = {micro[0]}\nRecall = {micro[1]}\nF1Score = {micro[2]}'
    )
```

Listing 11: Q4: accuracy

Result

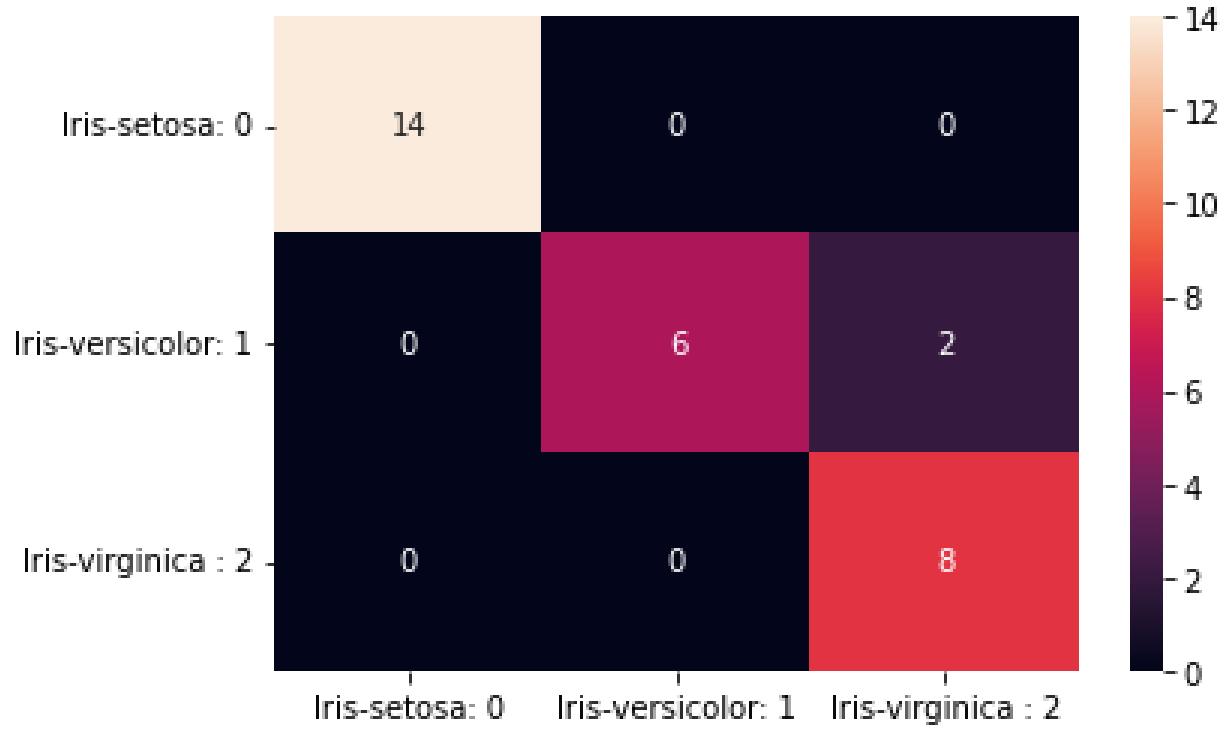

Figure 19: Confusion Matrix

Result

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	1.00	0.75	0.86	8
2	0.80	1.00	0.89	8
accuracy			0.93	30
macro avg	0.93	0.92	0.92	30
weighted avg	0.95	0.93	0.93	30

Figure 20: Classification report

Result

```
Macro
Precision = 0.9333333333333332
Recall = 0.9166666666666666
F1Score = 0.9153439153439153

Micro
Precision = 0.9333333333333333
Recall = 0.9333333333333333
F1Score = 0.9333333333333333

Weighted
Precision = 0.9466666666666667
Recall = 0.9333333333333333
F1Score = 0.9322751322751323
```

Figure 21: Macro, Micro and Weighted parameters

Observation/ Justification

The standard classification metrics help us to find out how accurate our model is and how much better predictions it can make for the different ranges of the values provided.

Confusion matrix has almost zero entries in cells which are not a main diagonal. This shown that our model is accurate. non zero entry of 2 in a cell $(1, 2)$ indicates that species should be Iris-verginica but it is predicted as Iris-versicolor.

Almost every parameters(accuracy, precision, recall and F1 score) is close to 1.00 which shows the correctness of our model.

Question 5

Imbalanced dataset typically refers to a dataset where the classes are not represented equally. Classification problems having multiple classes with imbalanced dataset present a different challenge than a binary classification problem. The skewed distribution makes the machine learning algorithms less effective, especially in predicting minority class examples.

In this question you will perform logistic regression for multiclass classification on the 20 News groups dataset. Since this dataset is a balanced one, you will perform the pre-processing to create an imbalanced version of the dataset (by removing some news articles from some groups). One example is given below. Perform multiclass classification using logistic regression on both the balanced and the imbalanced version of the dataset. Compare the performance in each case by obtaining the confusion matrix and accuracy. Report you observations at the end.

Answer

We have taken Iris dataset. First of all, we will split the dataset based on (same) species, and then with the help of those chunks we can pre-process the data to make unbalanced or balanced versions of the dataset.

```
iris_df = pd.read_csv('/content/Iris.csv')

# Picking Unique Species among the Species column
spec = iris_df['Species'].unique()

# Splitting the dataframe into small dataframes having data from the same species
iris_df0 = iris_df[iris_df['Species'] == spec[0]]
iris_df1 = iris_df[iris_df['Species'] == spec[1]]
iris_df2 = iris_df[iris_df['Species'] == spec[2]]
```

Balanced Version

code

```
# Creating Balanced Dataset from the Given Iris dataset by selecting a fraction of the rows from the splitted dataframes
new_subset_iris_df0 = iris_df0.sample(frac = 0.8)
new_subset_iris_df1 = iris_df1.sample(frac = 0.8)
new_subset_iris_df2 = iris_df2.sample(frac = 0.8)

# Merging the Dataframes back to generate a Balanced Dataframe
new_subset = new_subset_iris_df0.append(new_subset_iris_df1).append(new_subset_iris_df2)
cols = new_subset.columns[1:-1]
species = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica' : 2}
new_subset['Species'] = [species[item] for item in new_subset['Species']]
X = new_subset[cols]
y = new_subset[new_subset.columns[-1]]

# Performing Logistic Regression and predicting values
clf = LogisticRegression(max_iter = 10000)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 4)
clf.fit(X_train, y_train)
pred_y = clf.predict(X_test)

# Generating confusion matrix and its heat-map
conf_matrix = confusion_matrix(y_test, pred_y)
graph_cm_matrix = pd.DataFrame(conf_matrix, columns=['Iris-setosa: 0','Iris-versicolor: 1', 'Iris-virginica : 2'],
                               index=['Iris-setosa: 0','Iris-versicolor: 1', 'Iris-virginica : 2'])
sns.heatmap(graph_cm_matrix, annot = True, fmt = 'd')


# Classification Report
```

```
print(classification_report(y_test, pred_y))

print('Macro')
macro = precision_recall_fscore_support(y_test, pred_y, average='macro')
print(f'Precision = {macro[0]}\nRecall = {macro[1]}\nF1Score = {macro[2]}'
    )

print('\nMicro')
micro = precision_recall_fscore_support(y_test, pred_y, average='micro')
print(f'Precision = {micro[0]}\nRecall = {micro[1]}\nF1Score = {micro[2]}'
    )

print('\nWeighted')
weighted = precision_recall_fscore_support(y_test, pred_y, average='
    weighted')
print(f'Precision = {weighted[0]}\nRecall = {weighted[1]}\nF1Score = {
    weighted[2]}')
```

Result

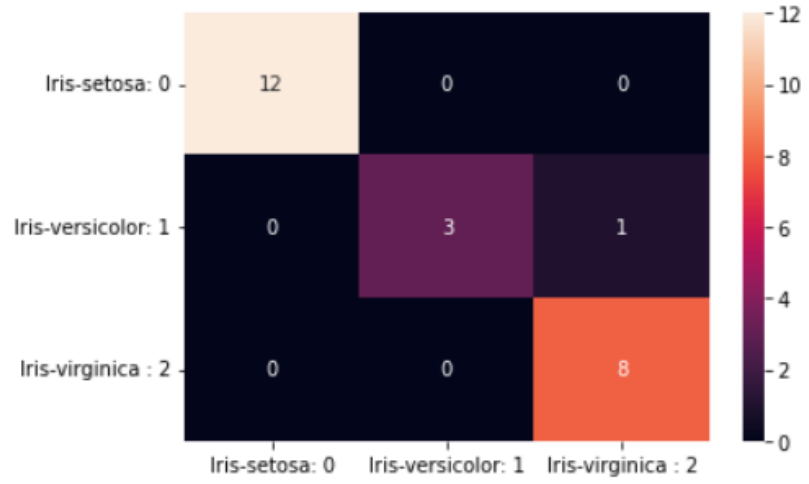

Figure 22: Heat-map of confusion matrix

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.75      0.86         4
           2       0.89      1.00      0.94         8

    accuracy                           0.96        24
   macro avg       0.96      0.92      0.93        24
weighted avg       0.96      0.96      0.96        24

Macro
Precision = 0.9629629629629629
Recall = 0.9166666666666666
F1Score = 0.9327731092436974

Micro
Precision = 0.9583333333333334
Recall = 0.9583333333333334
F1Score = 0.9583333333333334

Weighted
Precision = 0.9629629629629629
Recall = 0.9583333333333334
F1Score = 0.9565826330532213
```

Figure 23: Results

Unbalanced Version

code

```
# Creating Imbalanced Dataset from the Given Iris dataset by selecting a
    fraction of the rows from the splitted dataframes
subset_iris_df0 = iris_df0.sample(frac = 0.5)
subset_iris_df1 = iris_df1.sample(frac = 0.9)
subset_iris_df2 = iris_df2.sample(frac = 0.3)

# Merging the Dataframes back to generate the Imbalanced Dataframe
new_subset = subset_iris_df0.append(subset_iris_df1).append(
    subset_iris_df2)
cols = new_subset.columns[1:-1]
species = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica' : 2}
new_subset['Species'] = [species[item] for item in new_subset['Species']]
X = new_subset[cols]
y = new_subset[new_subset.columns[-1]]

# Performing Logistic Regression and predicting values
clf = LogisticRegression(max_iter = 10000)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 4)
clf.fit(X_train, y_train)
pred_y = clf.predict(X_test)
```

```
20
21 # Generating confusion matrix and its heat-map
22 conf_matrix = confusion_matrix(y_test, pred_y)
23 graph_cm_matrix = pd.DataFrame(conf_matrix, columns=['Iris-setosa: 0','
    Iris-versicolor: 1', 'Iris-virginica : 2'],
24                                index=['Iris-setosa: 0','Iris-versicolor:
    1', 'Iris-virginica : 2'])
25 sns.heatmap(graph_cm_matrix, annot = True, fmt = 'd')
26
27 # Classification Report
28 print(classification_report(y_test, pred_y))
29
30 print('Macro')
31 macro = precision_recall_fscore_support(y_test, pred_y, average='macro')
32 print(f'Precision = {macro[0]}\nRecall = {macro[1]}\nF1Score = {macro[2]}'
    )
33
34 print('\nMicro')
35 micro = precision_recall_fscore_support(y_test, pred_y, average='micro')
36 print(f'Precision = {micro[0]}\nRecall = {micro[1]}\nF1Score = {micro[2]}'
    )
37
38 print('\nWeighted')
39 weighted = precision_recall_fscore_support(y_test, pred_y, average='
    weighted')
40 print(f'Precision = {weighted[0]}\nRecall = {weighted[1]}\nF1Score = {
    weighted[2]}')
```

Result

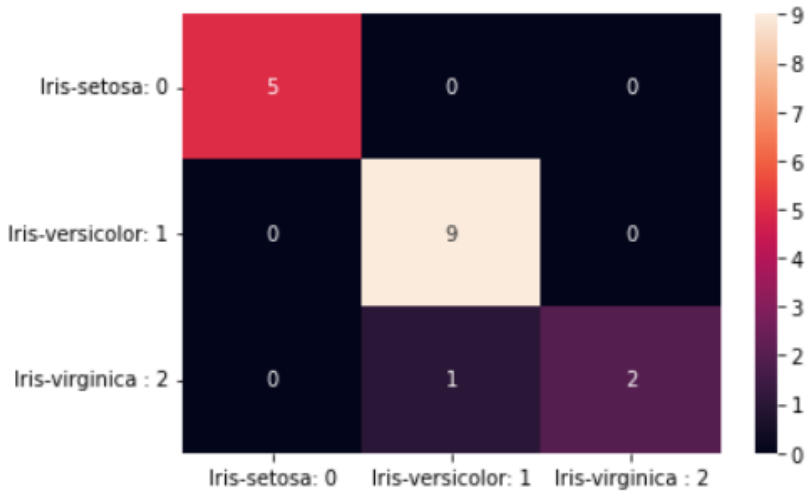

Figure 24: Heat-map of confusion matrix

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       0.90      1.00      0.95         9
           2       1.00      0.67      0.80         3

    accuracy                           0.94        17
   macro avg       0.97      0.89      0.92        17
weighted avg       0.95      0.94      0.94        17

Macro
Precision = 0.9666666666666667
Recall = 0.8888888888888888
F1Score = 0.9157894736842106

Micro
Precision = 0.9411764705882353
Recall = 0.9411764705882353
F1Score = 0.9411764705882353

Weighted
Precision = 0.9470588235294118
Recall = 0.9411764705882353
F1Score = 0.9368421052631579
```

Figure 25: Results

Observation/Justification

1. The standard classification metrics help us to find out how accurate our model is and how much better predictions it can make for the different ranges of the values provided.

2. But in case of imbalance or skew in the class distribution, the standard classification metrics does not perform pretty well as the dataset becomes biased towards the features having larger samples. And hence, the model ends up giving bad accuracy towards minority classes.

3. The basic problem with these metrics is, they assume that the skewness or the bias that is present in the training dataset will be present throughout the dataset. Thus they become inaccurate in classifying the values.