IE406 Machine Learning

Lab Assignment - 3

Group 28

201801015: Shantanu Tyagi
201801076: Shivani Nandani
201801407: Pratvi Shah
201801408: Arkaprabha Baerjee

Question 1

In this question we will use the data given in file "Social_Network_Ads.csv" which is a categorical dataset 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

We used the following equations in order to obtain the θ values:

$$h(\theta) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

$$\theta_j = \theta_j - \alpha \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \tag{2}$$

$$J(\theta) = \frac{-1}{m} \sum_{i=0}^{m} [y^{(i)} log(h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] \tag{3}$$

To predict the labels in the final stage we evaluate $\theta^T X$ and if it is greater than zero we classify it as 1 i.e., purchased and otherwise it is classified as 0. $J(\theta)$ will represent the final value of the cost.

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection  import  train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
SC=StandardScaler()

data = pd.read_csv('Social_Network_Ads.csv')

# Pre-processing
categorical_to_numerical_values = {"Gender": {"Male": 1, "Female": 0} }
data = data.replace(categorical_to_numerical_values)
X = np.array(data.drop(['User ID','Purchased','Gender'],axis=1))
X = SC.fit_transform(X)
X = np.column_stack([np.ones(len(X)),data['Gender'],X])
```

```
Y = data['Purchased']
data['Age'] = X[:,2];data['EstimatedSalary'] = X[:,3]
sns.heatmap(data.corr())

x_0 = X[Y==0];x_1 = X[Y==1];y_0 = Y[Y==0];y_1 = Y[Y==1]

# 3D plot with different colors for y labels
fig = plt.figure(figsize = (12, 6))
ax = plt.axes(projection ="3d")
ax.grid(b = True, color ='grey',linestyle ='-.', linewidth = 0.3, alpha = 0.2)
sctt = ax.scatter3D(x_0[:,1], x_0[:,2], x_0[:,3],alpha = 0.8,marker ='o', s=40)
sctt = ax.scatter3D(x_1[:,1], x_1[:,2], x_1[:,3],alpha = 0.8,marker ='o', s=40)
plt.title("Social Network Ads", fontsize=15)
ax.set_xlabel('Gender', fontsize=15)
ax.set_xlim(-0.5,1.5)
ax.set_ylabel('Age', fontsize=15)
ax.set_zlabel('EstimatedSalary', fontsize=15)
ax.legend(['Female','Male'])
plt.show()
sns.relplot(data=data, x="Age", y="EstimatedSalary",col="Gender",
    hue="Purchased",
    style="Purchased",
    kind="scatter"
)
plt.xlabel('Age')
plt.ylabel('EstimatedSalary')

# Helper functions
def h(theta,x):
    mat = np.dot(x,theta)
    return 1./(1 + np.exp(-1*mat))

def get_theta(theta,x,y,alpha):
    next_theta = theta - alpha*np.dot(x.T,(h(theta,x).T - y).T)
    return next_theta

def cost(theta,x,y):
    cost = []
    for i in range(len(y)):
        cost.append(y[i]*np.log(h(theta,x[i])) +  (1-y[i])*np.log(1-h(theta,x[i])))
    return np.sum(cost)

def classify(x,theta):
    if np.dot(x,theta)>0:
        return 1
    else:
        return 0

def get_metrics(real,pred):
    n = len(real)
    tp = 0; tn = 0; fp = 0; fn = 0
    for i in range(n):
        if real[i]==pred[i]:
            if real[i]==1:
                tp+=1
            else:
                tn+=1
        else:
            if real[i]==0:
                fp+=1
            else:
                fn+=1

    accuracy = (tp+tn)/n
    precision = tp/(tp + fp)
    recall = tp/(tp + fn)
    f1 = 2*precision*recall/(precision + recall)
    confusion_matrix = [[tn, fp],[fn, tp]]
    return accuracy, precision, recall, f1, confusion_matrix

# Split into training and testing set
x_train , x_test , y_train , y_test = train_test_split(X, Y, test_size =0.1, random_state = 0,
     stratify=Y)
y_test = np.array(y_test)
y_train = np.array(y_train)
```

```
96
97  # Get the Parameters
98  m = len(y_train)
99  n = len(x_train[0])
100 theta = np.zeros([n,1])
101 iterations = 10000
102 alpha =0.0001
103 j_theta = []
104 for i in range(iterations):
105     j_theta.append(-cost(theta,x_train,y_train)/m)
106     theta = get_theta(theta,x_train,y_train,alpha)
107 print('Theta0: ',theta[0,0],'\nTheta1: ',theta[1,0],'\nTheta2: ',theta[2,0],'\nTheta3: ',theta
        [3,0])
108 plt.show()
109
110 plt.plot(np.arange(0,iterations,1),j_theta)
111 plt.xlabel('Iterations',fontsize=15)
112 plt.ylabel('J(  )',fontsize=15)
113 plt.title('Cost v/s Iteration',fontsize=15)
114 plt.grid()
115
116 # Evaluate the necessary metrics to show the goodness of fit
117 y_pred = []
118 for i in range(len(x_test)):
119     y_pred.append(classify(x_test[i],theta))
120
121 accuracy, precision, recall, f1, confusionMatrix = get_metrics(y_test,y_pred)
122 print('Accuracy of our model: ', accuracy)
123 print('Precision of our model: ', precision)
124 print('Recall of our model: ', recall)
125 print('F1 score of our model: ', f1)
126
127 sns.heatmap(confusionMatrix,annot=True,xticklabels=['Not Purchased','Purchased'],yticklabels=[
        'Not Purchased','Purchased'])
128 plt.xlabel('Predicted',fontsize=15)
129 plt.ylabel('Actual',fontsize=15)
130
131 # Repeat using inbuilt functions
132 clf = LogisticRegression(max_iter=10000).fit(x_train, y_train)
133 coefs = [clf.intercept_[0],clf.coef_[0,1],clf.coef_[0,2],clf.coef_[0,3]]
134 print('Theta0: ',coefs[0],'\nTheta1: ',coefs[1],'\nTheta2: ',coefs[2],'\nTheta3: ',coefs[3])
135
136 accuracy_inbuilt = accuracy_score(y_test, clf.predict(x_test))
137 precision_inbuilt = precision_score(y_test, clf.predict(x_test))
138 recall_inbuilt = recall_score(y_test, clf.predict(x_test))
139 f1_inbuilt = f1_score(y_test, clf.predict(x_test))
140 print('Accuracy of inbuilt model: ', accuracy_inbuilt)
141 print('Precision of inbuilt model: ', precision_inbuilt)
142 print('Recall of inbuilt model: ', recall_inbuilt)
143 print('F1 score of inbuilt model: ', f1_inbuilt)
144
145 tn, fp, fn, tp  = confusion_matrix(y_test, clf.predict(x_test)).ravel()
146 sns.heatmap([[tn,fp],[fn,tp]],annot=True,xticklabels=['Not Purchased','Purchased'],yticklabels
        =['Not Purchased','Purchased'])
147 plt.xlabel('Predicted',fontsize=15)
148 plt.ylabel('Actual',fontsize=15)
```

Listing 1: Question 1

Result

3D plot of features:

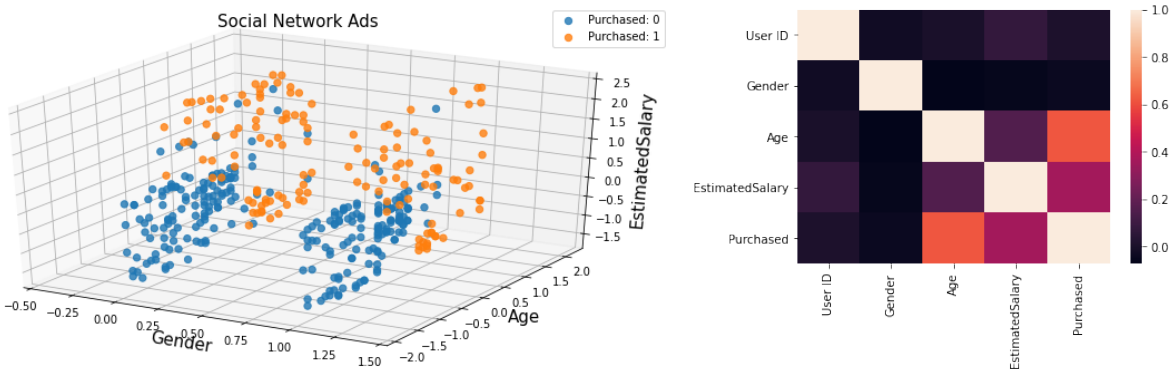


Figure 1: 3D plot and correlation of features

Cost function with respect to iterations:

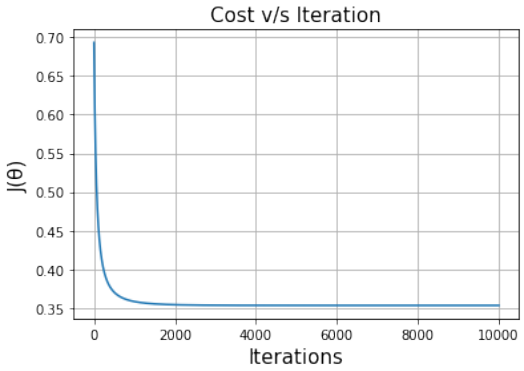


Figure 2: Cost $J(\theta)$ with respect to iterations

We obtained the following coefficients :

	Our model	Inbuilt
θ_0	-1.239219	-1.1638809
θ_1	0.214911	0.172561
θ_2	2.352501	2.173409
θ_3	1.195011	1.105823

Using testing data we obtained the evaluation metrics to be:

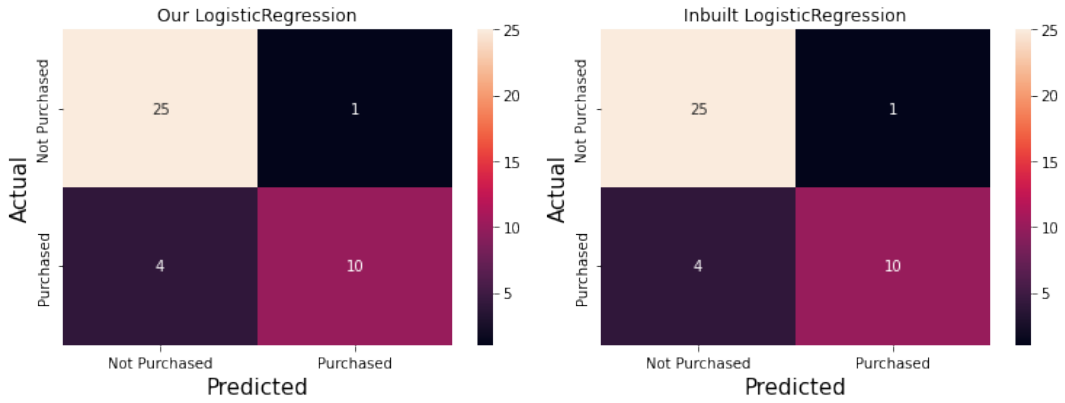
	Our model	Inbuilt
Accuracy	0.875	0.875
Precision	0.90909	0.90909
Recall	0.714285	0.714285
F1-Score	0.8	0.8

Confusion matrix for both the methods:

Observation/ Justification

We could make the following observations:

- From Figure 4 we could observe that Gender does not play a vital role in determining the purchase status
- We also observed the 3D plot with and without standardizing the data and obtained better evaluation metrics after standardization
- We obtained optimal learning rate = 0.0001 for iterations = 10000 for the gradient descent method to find θ
- Our model performed at par with the inbuilt sklearn model and both models provided a good fit for the data with Accuracy of about 87.5% for the testing data and 85% for training data.



(a) (b)

Figure 3: a) Our Model b) Inbuilt sklearn model

Question 2

In this question we aim to perform EDA (Exploratory Data Analysis) on the famous Iris Dataset by analyzing each of its features and basic statistical features in details.

Answer

In this question we shall use the following visualizations to represent various metrics:

1. Histogram to represent the frequency for every species in a given bin-width for a specific feature. The histograms will also have an inlaid PDF to compare both of them. For every species the mean value w.r.t every feature will also be represented in the plot.
2. KDE (Kernel Density Estimation) Plot or PDF for each species in a given feature.
3. Box Plot to summarize a set of data measured on an interval scale.It also helps in finding outliers and representing max values, min values , median and key percentile representations.
4. Violin Plot to represent the Probability distribution along with a box plot

Code

```
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(color_codes=True)
import warnings
warnings.filterwarnings("ignore")


df = pd.read_csv("Iris.csv")
print("net size = ",df.shape)
df.head(10)


print(df[df['Species']=='Iris-setosa'].describe())
setosa_df = df[df['Species']=='Iris-setosa']

print(df[df['Species']=='Iris-versicolor'].describe())
versicolor_df = df[df['Species']=='Iris-versicolor']

print(df[df['Species']=='Iris-virginica'].describe())
virginica_df = df[df['Species']=='Iris-virginica']
```

Listing 2: Question 2 : Setting Up the dataset

```

# Pie Chart

```

```
df['Species'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True,figsize=(6,6))
plt.title('Species distribution')
plt.show()
```

Listing 3: Question 2: Pie chart

```

# Histogram

sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.distplot, "SepalWidthCm")
plt.axvline(setosa_df['SepalWidthCm'].mean(), color='red', linestyle=':', linewidth=3, label =
     "Mean: {:.3f}".format(setosa_df['SepalWidthCm'].mean()))
plt.axvline(virginica_df['SepalWidthCm'].mean(), color='blue', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(virginica_df['SepalWidthCm'].mean()))
plt.axvline(versicolor_df['SepalWidthCm'].mean(), color='green', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(versicolor_df['SepalWidthCm'].mean()))
plt.legend()
plt.show()

print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.distplot, "SepalLengthCm")
plt.axvline(setosa_df['SepalLengthCm'].mean(), color='red', linestyle=':', linewidth=3, label
    = "Mean: {:.3f}".format(setosa_df['SepalLengthCm'].mean()))
plt.axvline(virginica_df['SepalLengthCm'].mean(), color='blue', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(virginica_df['SepalLengthCm'].mean()))
plt.axvline(versicolor_df['SepalLengthCm'].mean(), color='green', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(versicolor_df['SepalLengthCm'].mean()))
plt.legend()
plt.show()

print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.distplot, "PetalWidthCm")
plt.axvline(setosa_df['PetalWidthCm'].mean(), color='red', linestyle=':', linewidth=3, label =
     "Mean: {:.3f}".format(setosa_df['PetalWidthCm'].mean()))
plt.axvline(virginica_df['PetalWidthCm'].mean(), color='blue', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(virginica_df['PetalWidthCm'].mean()))
plt.axvline(versicolor_df['PetalWidthCm'].mean(), color='green', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(versicolor_df['PetalWidthCm'].mean()))
plt.legend()
plt.show()

print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.distplot, "PetalLengthCm")
plt.axvline(setosa_df['PetalLengthCm'].mean(), color='red', linestyle=':', linewidth=3, label
    = "Mean: {:.3f}".format(setosa_df['PetalLengthCm'].mean()))
plt.axvline(virginica_df['PetalLengthCm'].mean(), color='blue', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(virginica_df['PetalLengthCm'].mean()))
plt.axvline(versicolor_df['PetalLengthCm'].mean(), color='green', linestyle=':', linewidth=3,
    label = "Mean: {:.3f}".format(versicolor_df['PetalLengthCm'].mean()))
plt.legend()
plt.show()
print()
# sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(plt.hist, "PetalLengthCm").
    add_legend()
# plt.show()
```

Listing 4: Question 2: Histogram

```

# PDF plots
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.kdeplot, "SepalWidthCm").
    add_legend()
plt.show()
print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.kdeplot, "SepalLengthCm").
    add_legend()
plt.show()
print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.kdeplot, "PetalWidthCm").
    add_legend()
plt.show()
print()
sns.FacetGrid(df, hue="Species", palette="husl", size=6).map(sns.kdeplot, "PetalLengthCm").
    add_legend()
plt.show()
```

```
print()
```

Listing 5: Question 2: PDF Plot

```

# Violin Plot
sns.violinplot(x=df["Species"], y=df['PetalLengthCm'], palette="husl", data=df)
plt.show()
print()
sns.violinplot(x=df["Species"], y=df['PetalWidthCm'], palette="husl", data=df)
plt.show()
print()
sns.violinplot(x=df["Species"], y=df['SepalLengthCm'], palette="husl", data=df)
plt.show()
print()
sns.violinplot(x=df["Species"], y=df['SepalWidthCm'], palette="husl", data=df)
plt.show()
print()
```

Listing 6: Question 2: Violin Plot

```

# Box Plot
sns.boxplot(x=df["Species"], y=df['PetalLengthCm'], palette="husl", data=df)
plt.show()
print()
sns.boxplot(x=df["Species"], y=df['PetalWidthCm'], palette="husl", data=df)
plt.show()
print()
sns.boxplot(x=df["Species"], y=df['SepalLengthCm'], palette="husl", data=df)
plt.show()
print()
sns.boxplot(x=df["Species"], y=df['SepalWidthCm'], palette="husl", data=df)
plt.show()
print()
```

Listing 7: Question 2: Box Plot

Result

The following pie chart represents the percentage of every species:

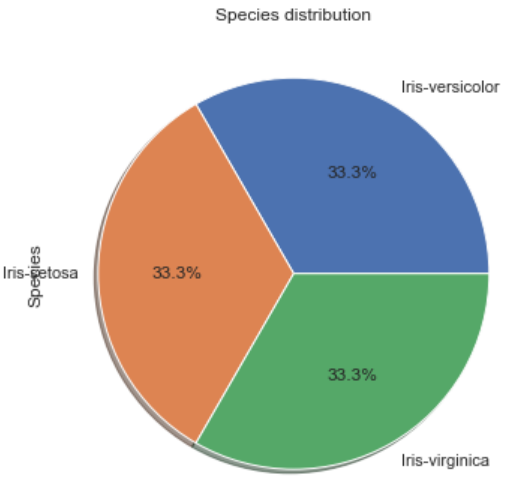


Figure 4: Pie chart for percentage of every species

Figure 5: Pair plots to represent the scatter plots along with its distribution for every possible feature pair

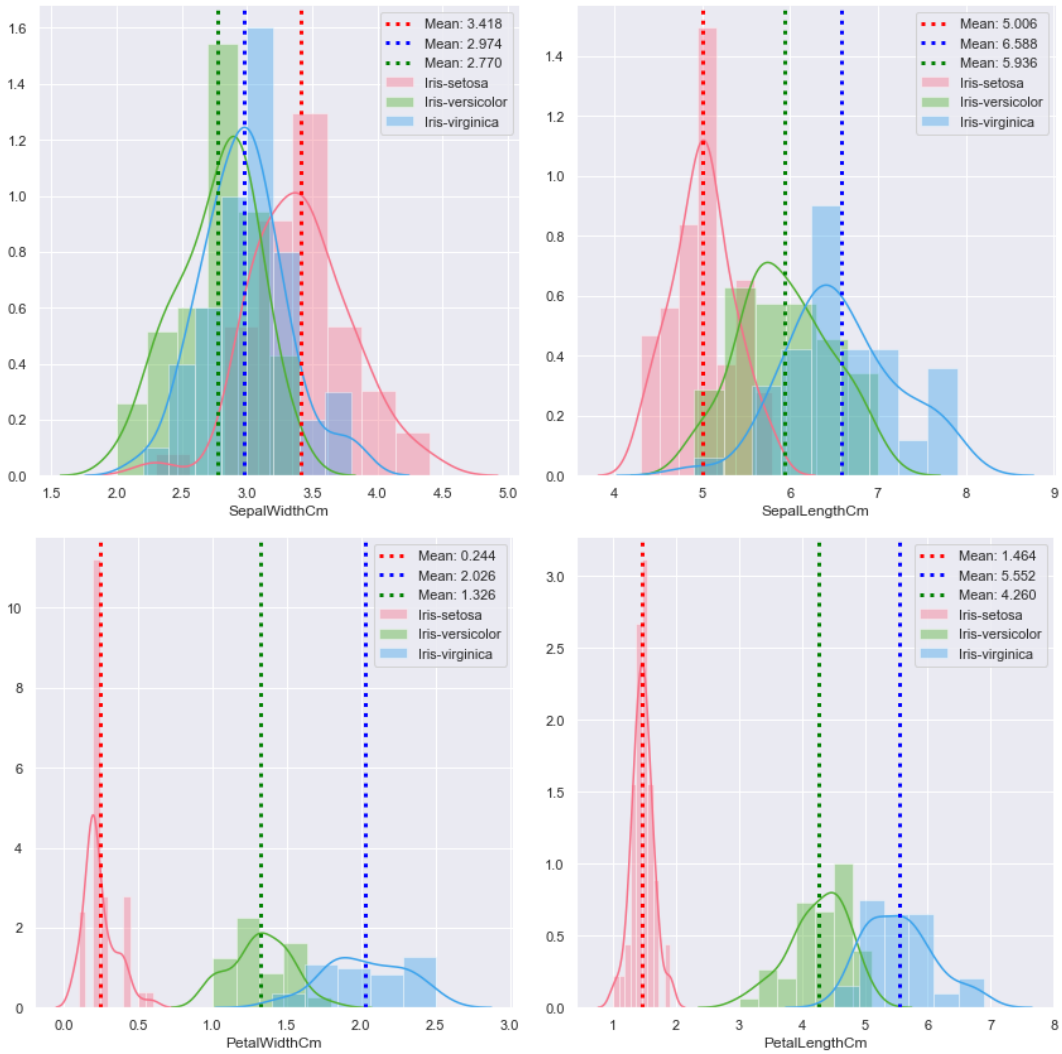


Figure 6: Histogram for each feature with separate distributions for each species

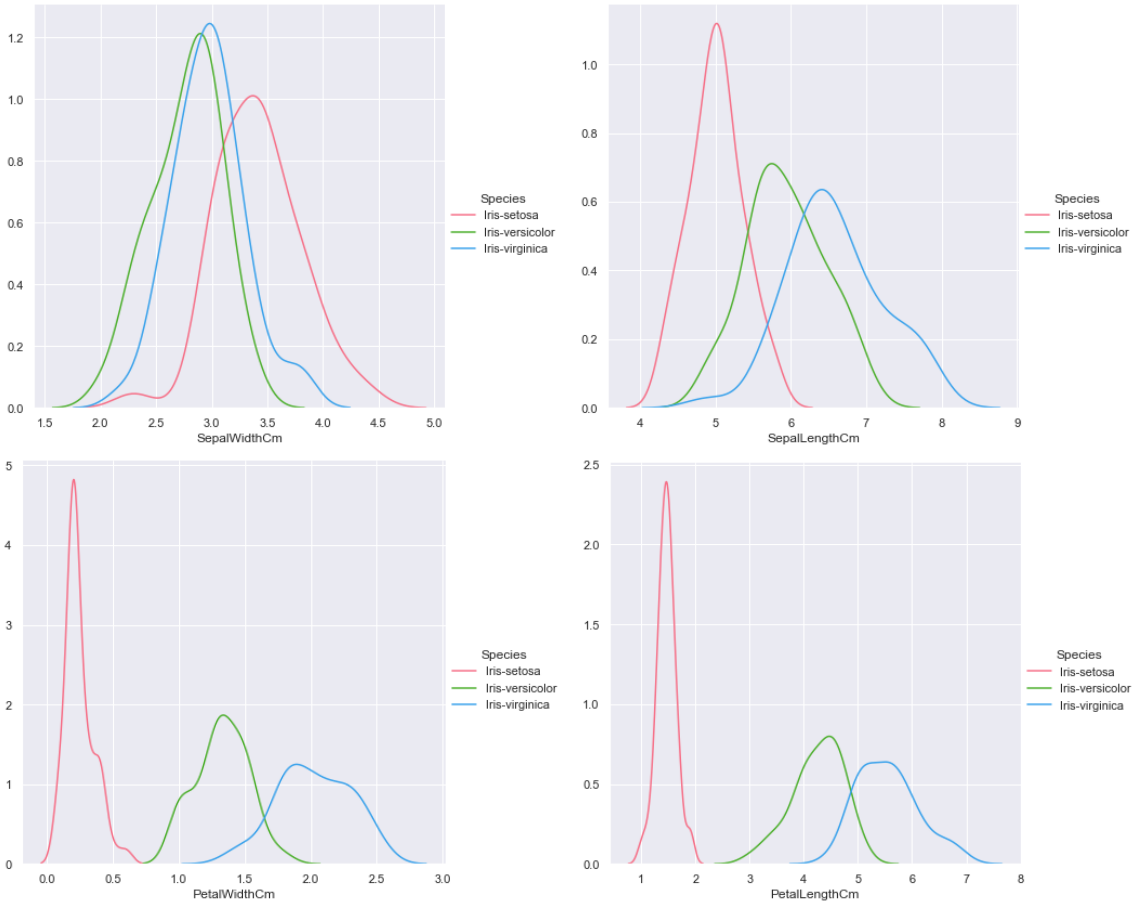


Figure 7: Standalone PDF for every feature with separate distributions for each species

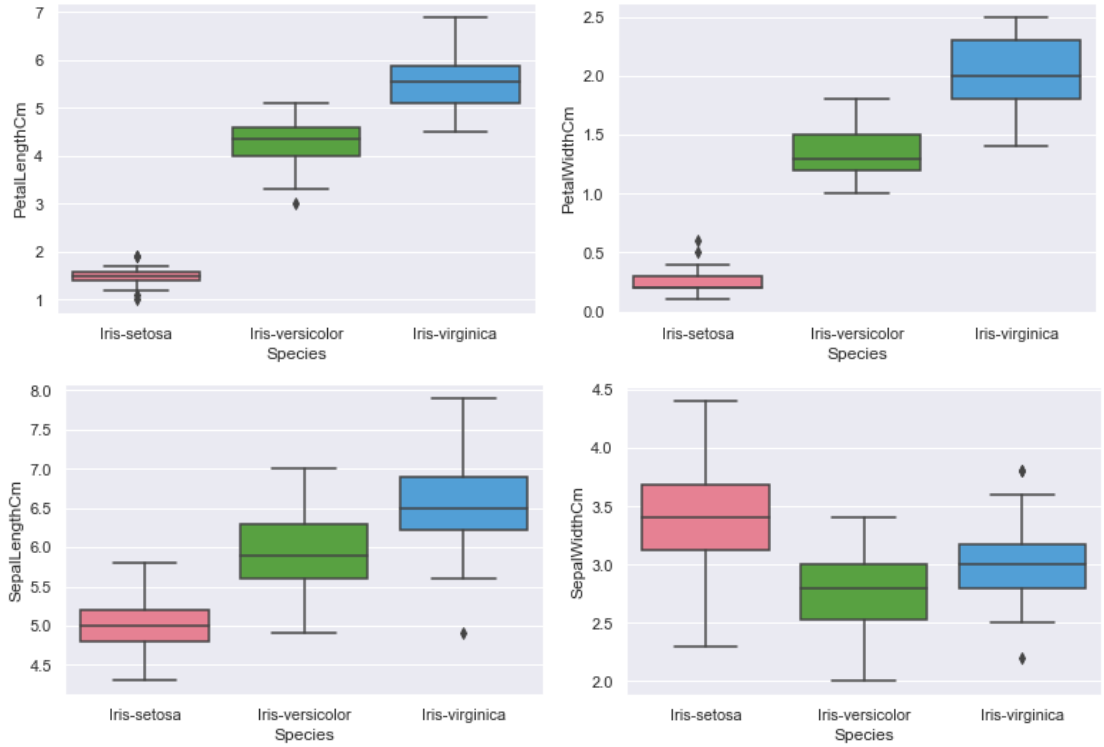


Figure 8: Box Plot for every feature with separate lanes for every species

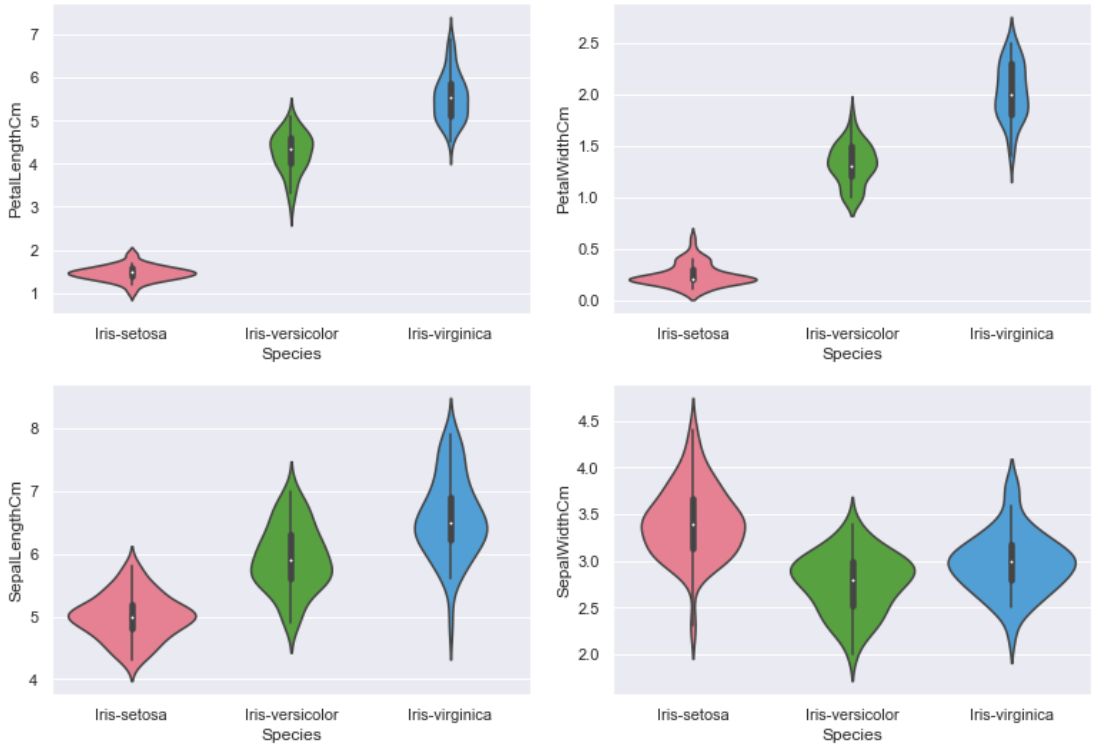


Figure 9: Violin Plot for every feature with separate lanes for every species

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	50.00000	50.000000	50.000000	50.000000	50.000000
mean	75.50000	5.936000	2.770000	4.260000	1.326000
std	14.57738	0.516171	0.313798	0.469911	0.197753
min	51.00000	4.900000	2.000000	3.000000	1.000000
25%	63.25000	5.600000	2.525000	4.000000	1.200000
50%	75.50000	5.900000	2.800000	4.350000	1.300000
75%	87.75000	6.300000	3.000000	4.600000	1.500000
max	100.00000	7.000000	3.400000	5.100000	1.800000

Figure 10: Basic statistics for versicolor species

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	50.00000	50.00000	50.000000	50.000000	50.00000
mean	25.50000	5.00600	3.418000	1.464000	0.24400
std	14.57738	0.35249	0.381024	0.173511	0.10721
min	1.00000	4.30000	2.300000	1.000000	0.10000
25%	13.25000	4.80000	3.125000	1.400000	0.20000
50%	25.50000	5.00000	3.400000	1.500000	0.20000
75%	37.75000	5.20000	3.675000	1.575000	0.30000
max	50.00000	5.80000	4.400000	1.900000	0.60000

Figure 11: Basic statistics for Setosa species

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	50.00000	50.00000	50.000000	50.000000	50.00000
mean	125.50000	6.58800	2.974000	5.552000	2.02600
std	14.57738	0.63588	0.322497	0.551895	0.27465
min	101.00000	4.90000	2.200000	4.500000	1.40000
25%	113.25000	6.22500	2.800000	5.100000	1.80000
50%	125.50000	6.50000	3.000000	5.550000	2.00000
75%	137.75000	6.90000	3.175000	5.875000	2.30000
max	150.00000	7.90000	3.800000	6.900000	2.50000

Figure 12: Basic statistics for Virginica species

Observation/ Justification

1. All classes/species contain equal number of data points as can be seen from the pie chart
2. In histogram we consider discrete data by binning the range. In PDF we represent continuous data in the form of probability over a given range. However we observe that the trends in histogram and PDF remain the same , i.e. whenever histogram column rises , the PDF plot also increases and vice versa.

3. Setosa can be easily separated from the other 2 species if we consider the features : Petal Width or Petal Length. However, the other 2 species have overap and cant be distinguisged very easily.

4. For Sepal width or sepal length none of the species can be distinguished easily due to considerable overlap as seen in the Histogram and PDF plots.

5. The PDF/KDE plots often have a value of more than 1. This is perfectly alright as the y-axis doesnt represent the probability for that particular feature value rather the area of a small imaginary column gives us the probability of occurrence over that segment. Thus the area over the PDF comes out to be 1.

6. From the boxplot and violin plot we can see that species are very clearly separable based upon the petal length. For sepal width we observe the highest amount of overlap and outliers as well

Question 3

In this question we shall extend the EDA (Exploratory Data Analysis) performed in the last question and extend it to pair plots for all feature combinations.

Answer

In this question we shall use the following visualizations to represent various metrics:

1. Joint plots to represent the scattered plots as well as their overall distribution together for all possible feature combinations.

2. Pair plots to represent all possible joint plots together with KDE/PDF plots along its diagonal.

Code

```
# joint plots
sns.jointplot( x=df['SepalLengthCm'],y=df['SepalWidthCm'],data=df, hue="Species")
plt.show()
print()
sns.jointplot( x=df['SepalLengthCm'],y=df['PetalLengthCm'],data=df, hue="Species")
plt.show()
print()
sns.jointplot( x=df['SepalLengthCm'],y=df['PetalWidthCm'],data=df, hue="Species")
plt.show()

sns.jointplot( x=df['PetalLengthCm'],y=df['PetalWidthCm'],data=df, hue="Species")
plt.show()

sns.jointplot( x=df['PetalLengthCm'],y=df['SepalWidthCm'],data=df, hue="Species")
plt.show()
print()
sns.jointplot( x=df['SepalWidthCm'],y=df['PetalWidthCm'],data=df, hue="Species")
plt.show()

# Pair plot
sns.pairplot( df.drop('Id',axis=1),hue="Species", palette="hls", diag_kind="kde",data=df)
plt.show()
```

Listing 8: Question 3 :

Result

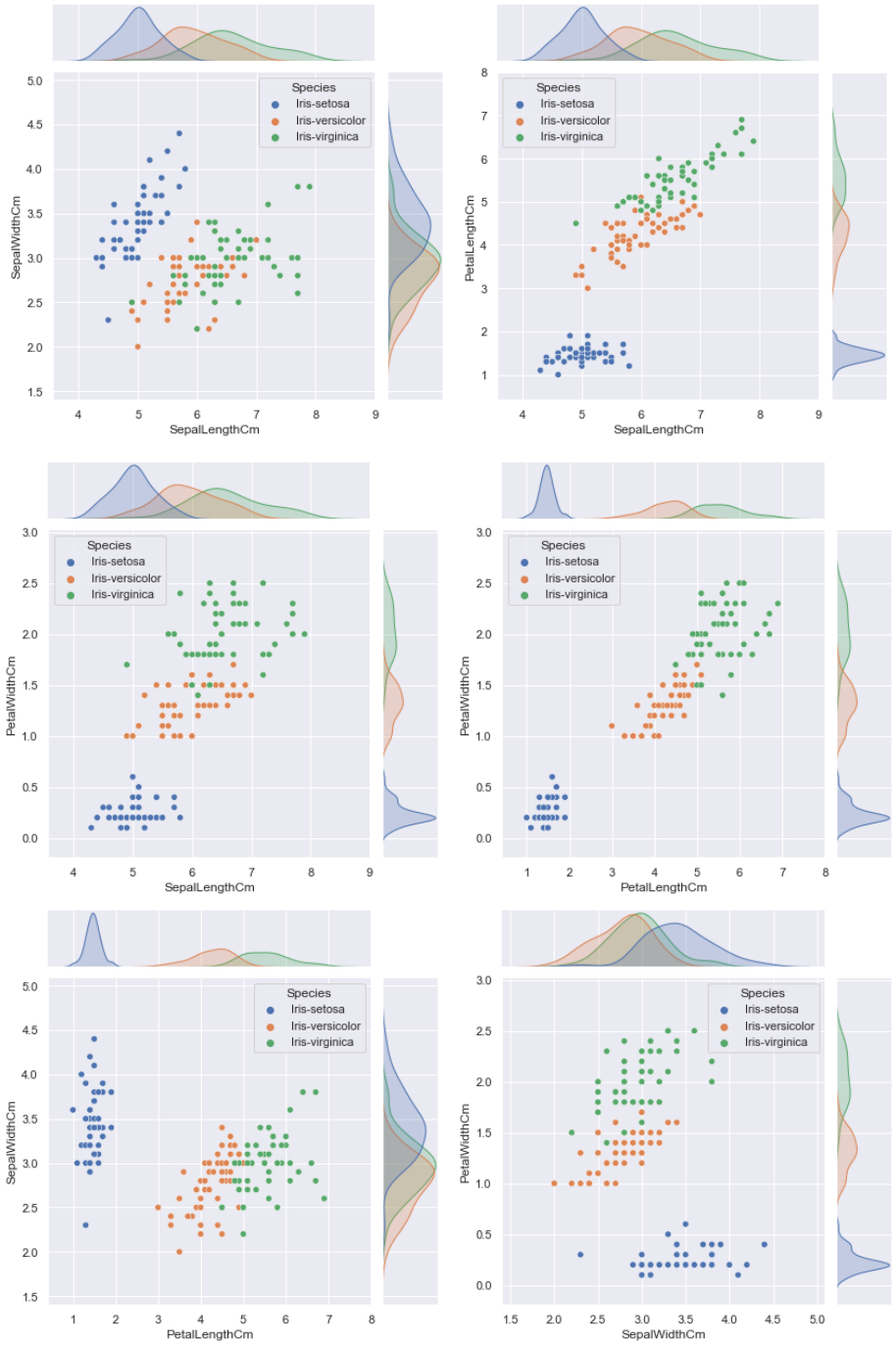


Figure 13: Joint plots to represent the scatter plots along with its distribution for every possible feature pair

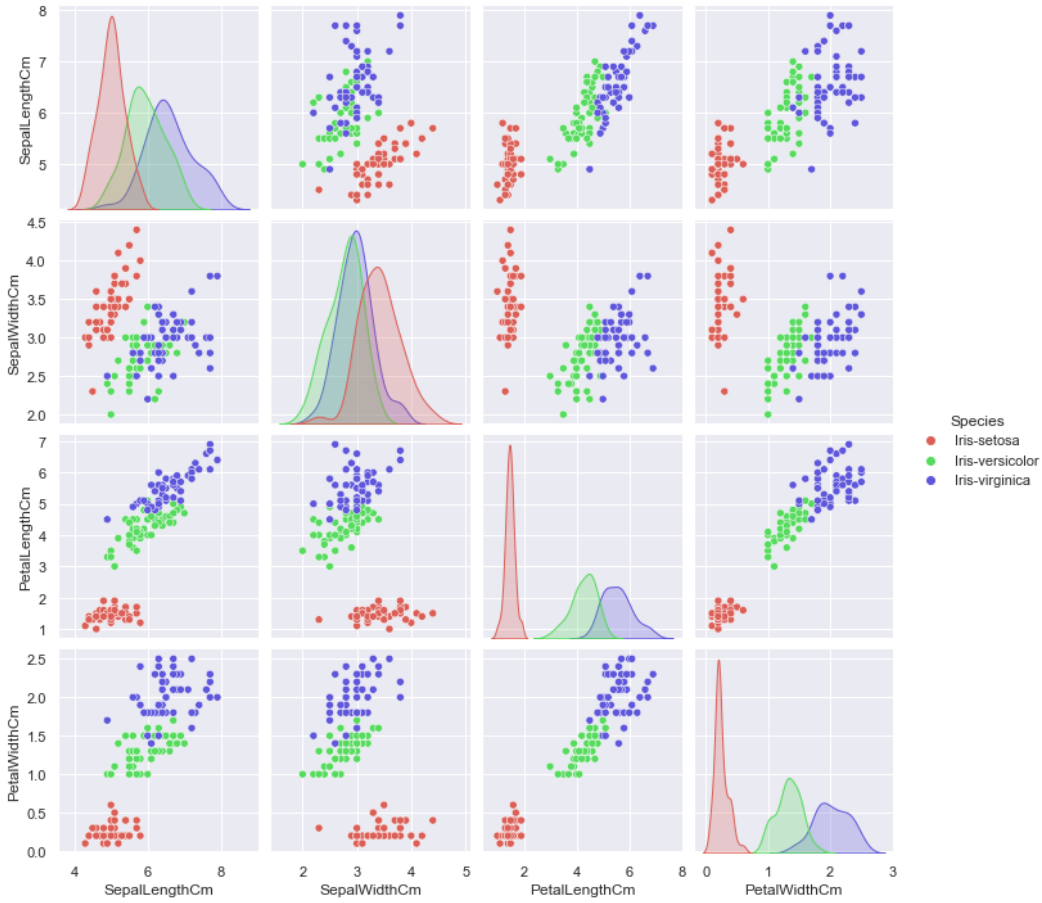


Figure 14: Pair plots to summarize all scatter plots with PDF plot along its diagonal

Observation/ Justification

1. For Sepal length vs sepal width plot, we can differentiate Setosa flowers from others by drawing a line however, separating the Versicolor from Viginica species is difficult as they have considerable overlap.
2. For all the other 5 joint plots we can differentiate Setosa flowers from others by drawing a line very easily and with high accuracy. Versicolor and Viginica species can be separated but with a lower accuracy but in all cases it is better than separating the 2 on the basis of sepal width vs sepal length plot.
3. Setosa species has a significant difference in its characteristics when compared to the other two species. It has smaller petal width,sepal length as well as sepal length while its sepal width is high.
4. Versicolor usually has higher average dimensions as compared to setosa for both sepal and petal features.
5. The Virginica species has high petal width and length as well as sepal length. However, it has small sepal width .
6. There is considerable high co relation between petal length and petal width features.

Question 4

In this question, we shall perform logistic regression on IRIS Dataset and plot confusion matrix and find accuracy, precision, F1 score and recall.

Answer

In this question we shall use the following visualizations to represent various metrics:

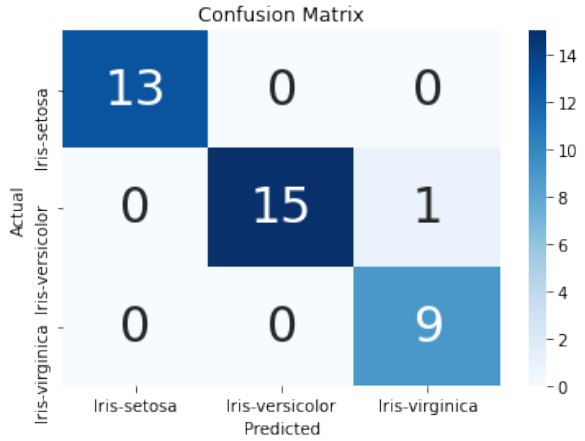
1. A heat map to represent the the confusion matrix that will represent the magnitude in terms of a color scale.

Code

```
# Load dataset
dataset = pd.read_csv('iris.csv')
# Preprocess Data
dataset.drop('Id', inplace=True, axis=1)
X = dataset.iloc[:, [0,1,2,3]].values
y = dataset.iloc[:, 4].values
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
# standardize the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Fit the data to the model
classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
classifier.fit(X_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling
    =1, max_iter=100, multi_class='auto',n_jobs=None, penalty='l2', random_state=0, solver='
    lbfgs',tol=0.0001, verbose=0, warm_start=False)
# Predicting the Test set results
y_pred = classifier.predict(X_test)
# Predict probabilities
probs_y=classifier.predict_proba(X_test)
cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix
class_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
ax = plt.axes()
df_cm = cm
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax,
    xticklabels=class_names, yticklabels=class_names)
ax.set_title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
# Model Evaluation parameters
ac_sc = accuracy_score(y_test, y_pred)
rc_sc = recall_score(y_test, y_pred, average="weighted")
pr_sc = precision_score(y_test, y_pred, average="weighted")
f1_sc = f1_score(y_test, y_pred, average='micro')
confusion_m = confusion_matrix(y_test, y_pred)
print("Accuracy    : ", ac_sc)
print("Recall      : ", rc_sc)
print("Precision   : ", pr_sc)
print("F1 Score    : ", f1_sc)
print("Confusion Matrix: ")
print(confusion_m)
```

Listing 9: Question 4

Result




```
Accuracy    :  0.9736842105263158
Recall      :  0.9736842105263158
Precision   :  0.9763157894736842
F1 Score    :  0.9736842105263158
Confusion Matrix:
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

Observation/ Justification

1. The confusion matrix is diagonal matrix. The diagonal elements show the number of correct classifications for each class. The off-diagonal elements provides the misclassifications which are only 1 in our case.

2. Accuracy means what fraction of the total predictions were correct. In our model we splitted 25% of the data as testing data and we see that 97.36842105263158% of the predictions were correct.

3. Precision means what percentage of the positive predictions made were actually correct. Thus it is a ratio of True positives to all positives(true and false) which comes out to be 97.63157894736842% since there were very few false positives.

4. Recall means what percentage of actual positive predictions were correctly classified by the classifier. Thus it is a ratio of True positives to actual positives(true positive and false negative) which comes out to be 97.36842105263158% in our case since there were very few false negatives.

5. F1 score can also be described as the harmonic mean of precision and recall. In our case it is 0.9736842105263158.

Question 5

In this question you will perform logistic regression for multiclass classification on the IRIS dataset. Since this dataset is a balanced one, you will perform the pre-processing to create an imbalanced version of the dataset. 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

In this question, we are required to work on a multi-class classification problem using the IRIS dataset using the LogisticRegression() function from sklearn.linear_model. For the first part of the question, we use the IRIS dataset with the required pre-processing. For the second part, we use delete rows in order to convert the given balanced dataset into an imbalanced one. The accuracy obtained for the each case is given below.

- The accuracy for the balanced dataset is 97.77%.
- The accuracy for the imbalanced dataset is 96.29%.

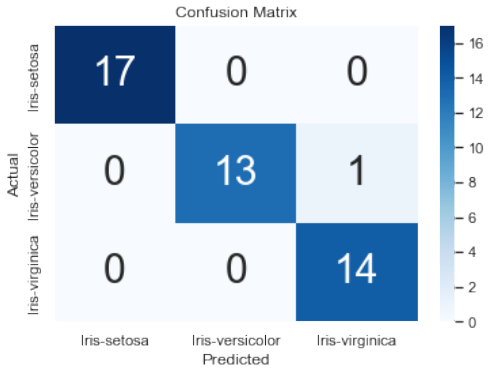
code

```
# Load dataset
df = pd.read_csv('Iris.csv')

# Balanced dataset

# create X and y
X = df.drop(columns=['Id', 'Species'])
StandardScaler().fit_transform(X)
y = df['Species'].copy()
y = np.array(y)
classes = np.unique(y)
# split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=3)
# fit model
reg = LogisticRegression(max_iter=1000)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
# print classification report
print(classification_report(y_pred, y_test, target_names=classes))
# print accuracy
print(metrics.accuracy_score(y_pred, y_test))
# print confusion matrix
ax = plt.axes()
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax, xticklabels
    =classes, yticklabels=classes)
ax.set_title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Imbalanced dataset

# deleting rows to create the required dataset
df_imb = df.copy()
```

```
df_imb = df_imb.drop(index=[i for i in range(0,20)])
df_imb = df_imb.drop(index=[i for i in range(50,90)])
# create X and y
X = df_imb.drop(columns=['Id', 'Species'])
StandardScaler().fit_transform(X)
y = df_imb['Species'].copy()
y = np.array(y)
classes = np.unique(y)
# split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=3) #
    fit model
reg_imb = LogisticRegression(max_iter=1000)
reg_imb.fit(X_train, y_train)
y_pred = reg_imb.predict(X_test)
# print classification report
print(classification_report(y_pred, y_test, target_names=classes, zero_division=1))
# print accuray
print(metrics.accuracy_score(y_pred, y_test))
# print confusion matrix
ax = plt.axes()
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax, xticklabels
    =classes, yticklabels=classes)
ax.set_title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Listing 10: Question 5

Result



	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	17
Iris-versicolor	0.93	1.00	0.96	13
Iris-virginica	1.00	0.93	0.97	15
accuracy			0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

Accuracy
0.9777777777777777

Figure 15: For balanced dataset

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	9
Iris-versicolor	0.00	1.00	0.00	0
Iris-virginica	1.00	0.94	0.97	18
accuracy			0.96	27
macro avg	0.67	0.98	0.66	27
weighted avg	1.00	0.96	0.98	27

Accuracy
0.9629629629629629

Figure 16: For imbalanced dataset

Observation/ Justification

1. As seen in the above-given plots and reports, the accuracy of the balanced dataset is higher than the imbalanced one.
2. Due to the small dataset size, the testing dataset doesn't contain any Iris-versicolor sample.
3. From the confusion matrices, it can be seen that for both the cases there is only one incorrect classification.
4. Since, in case of the imbalanced dataset, there are more samples of one class as compared to the rest, the algorithm tends to predict the majority class. Thus, even if the accuracy of the model is high, it may not be able to classify the dataset correctly.