Name: Kemal Demirel

ID: 191104091

Course: BIL470

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from dt import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
```

Exploratory Data Analysis (EDA)

In [2]:
```
irisData = pd.read_csv("Iris.csv")
irisData = irisData.drop(columns="Id")
for index, row in irisData.iterrows():
    if irisData.loc[index, "Species"] == "Iris-setosa":
        irisData.loc[index, "Species"] = 0;
    elif irisData.loc[index, "Species"] == "Iris-versicolor":
        irisData.loc[index, "Species"] = 1;
    elif irisData.loc[index, "Species"] == "Iris-virginica":
        irisData.loc[index, "Species"] = 2;
```

In [3]:
```
display(irisData);
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [4]:
```
print("Number of Columns: 5")
print( irisData["SepalLengthCm"].describe() )
print( irisData["SepalWidthCm"].describe() )
```

```
print( irisData["PetalLengthCm"].describe() )
print( irisData["PetalWidthCm"].describe() )
print( irisData["Species"].describe() )
```

```
Number of Columns: 5
count    150.000000
mean       5.843333
std        0.828066
min        4.300000
25%        5.100000
50%        5.800000
75%        6.400000
max        7.900000
Name: SepalLengthCm, dtype: float64
count    150.000000
mean       3.054000
std        0.433594
min        2.000000
25%        2.800000
50%        3.000000
75%        3.300000
max        4.400000
Name: SepalWidthCm, dtype: float64
count    150.000000
mean       3.758667
std        1.764420
min        1.000000
25%        1.600000
50%        4.350000
75%        5.100000
max        6.900000
Name: PetalLengthCm, dtype: float64
count    150.000000
mean       1.198667
std        0.763161
min        0.100000
25%        0.300000
50%        1.300000
75%        1.800000
max        2.500000
Name: PetalWidthCm, dtype: float64
count     150
unique      3
top         0
freq       50
Name: Species, dtype: int64
```

In [5]:
```
display( irisData[irisData.duplicated()] )
display( irisData.duplicated().sum() )
```

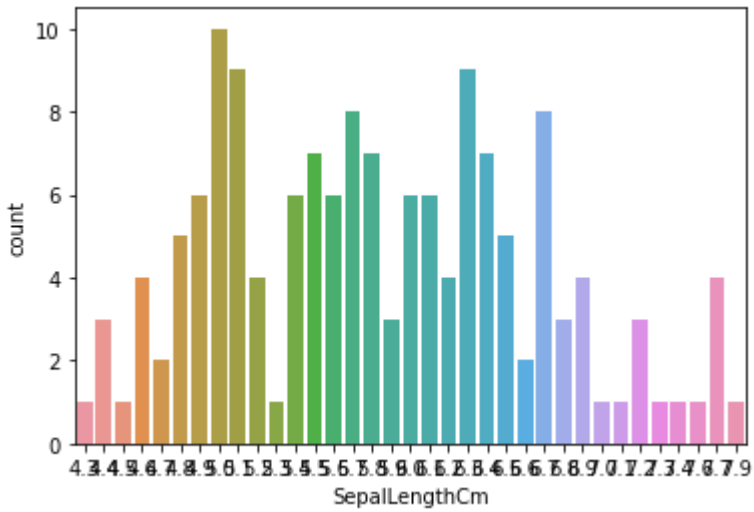
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
34	4.9	3.1	1.5	0.1	0
37	4.9	3.1	1.5	0.1	0
142	5.8	2.7	5.1	1.9	2

```
3
```

In [6]:
```
sns.countplot(irisData["SepalLengthCm"])
```

```
/Users/kemaldemirel/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decor
ators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

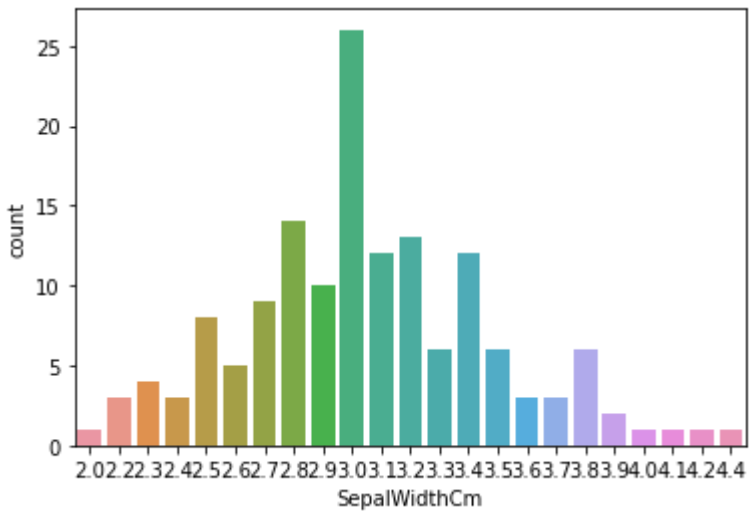
Out[6]:
```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='count'>
```



In [7]:
```
sns.countplot(irisData["SepalWidthCm"])
```

```
/Users/kemaldemirel/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decor
ators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

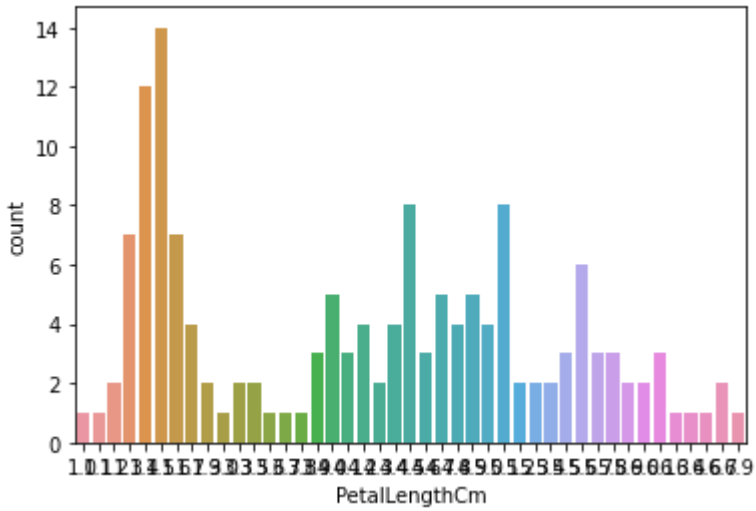
Out[7]:
```
<AxesSubplot:xlabel='SepalWidthCm', ylabel='count'>
```



In [8]:
```
sns.countplot(irisData["PetalLengthCm"])
```

```
/Users/kemaldemirel/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decor
ators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

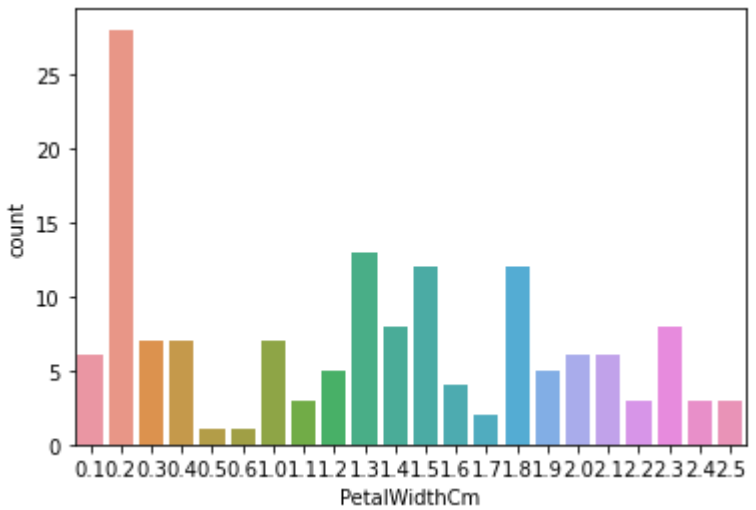
Out[8]:
```
<AxesSubplot:xlabel='PetalLengthCm', ylabel='count'>
```



In [9]:
```
sns.countplot(irisData["PetalWidthCm"])
```

```
/Users/kemaldemirel/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decor
ators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

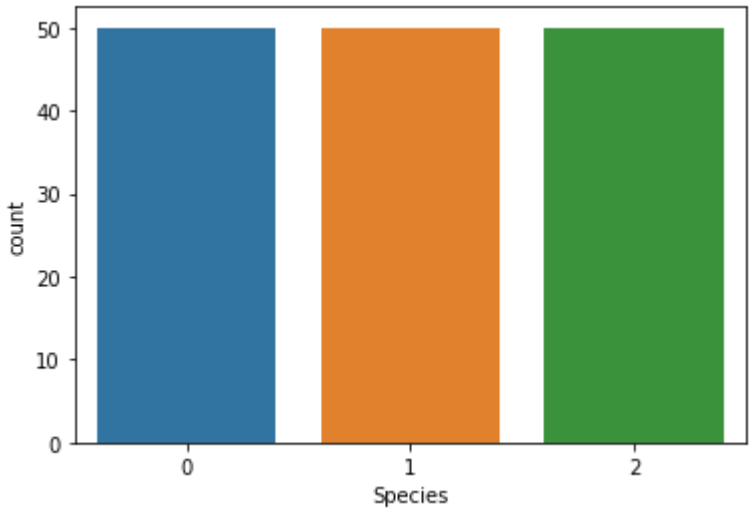
Out[9]:
```
<AxesSubplot:xlabel='PetalWidthCm', ylabel='count'>
```



In [10]:
```
sns.countplot(irisData["Species"])
```

```
/Users/kemaldemirel/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decor
ators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

Out[10]:
```
<AxesSubplot:xlabel='Species', ylabel='count'>
```

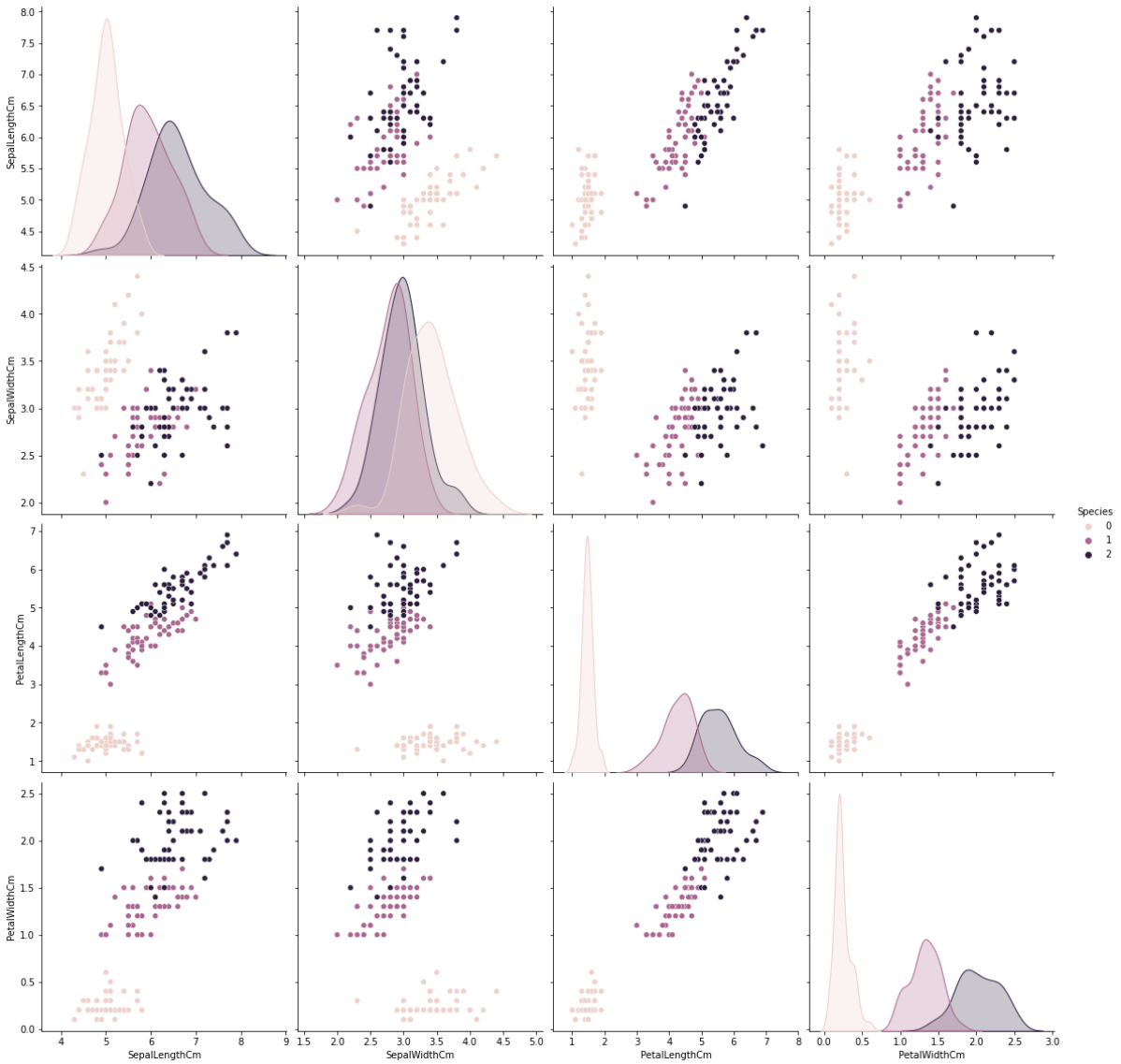


In [11]:
```
print("k")
```

```
k
```

Pair-plot

In [12]:
```
sns.pairplot(irisData, hue="Species", height=4)
plt.show()
```

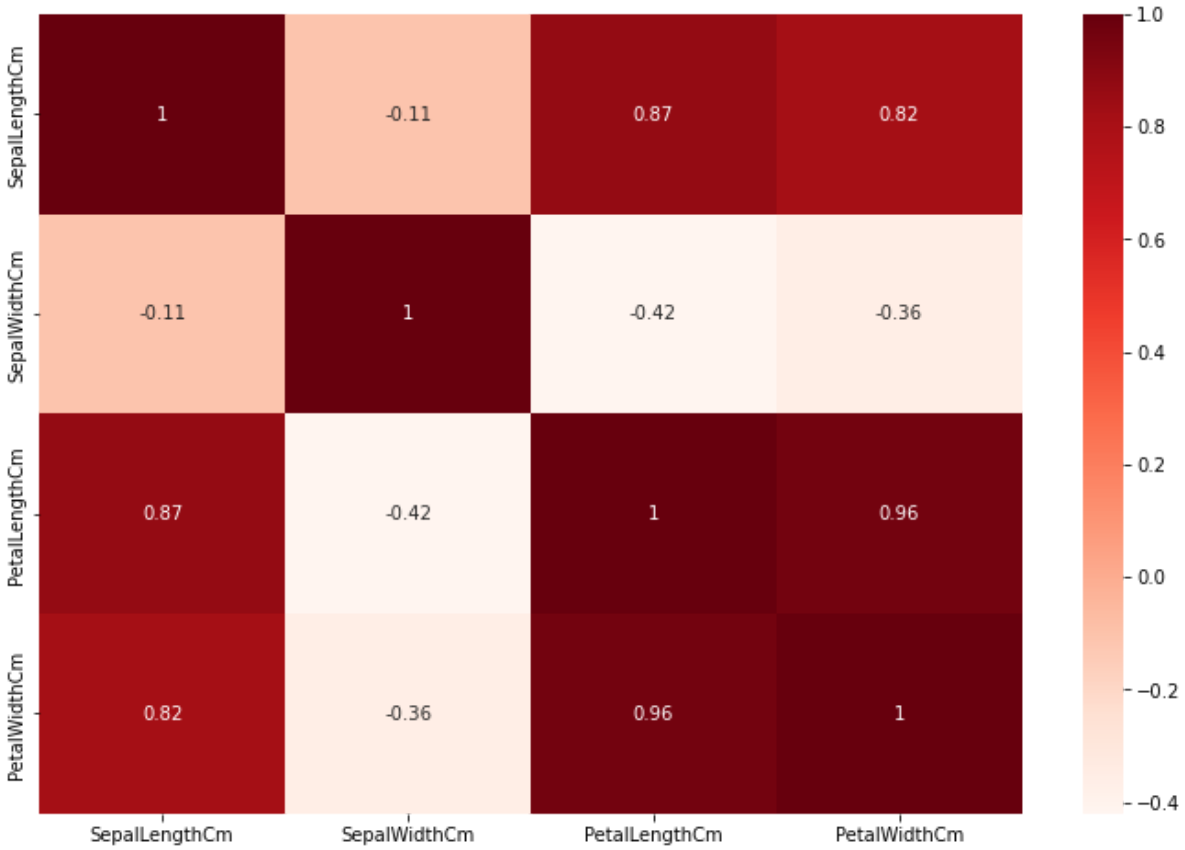


Correlation Matrix

In [13]:
```
corr = irisData.corr()
print(irisData)

plt.figure(figsize=(12,8))
sns.heatmap(corr, cmap="Reds",annot=True)
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm Species
0              5.1           3.5            1.4           0.2       0
1              4.9           3.0            1.4           0.2       0
2              4.7           3.2            1.3           0.2       0
3              4.6           3.1            1.5           0.2       0
4              5.0           3.6            1.4           0.2       0
..             ...           ...            ...           ...     ...
145            6.7           3.0            5.2           2.3       2
146            6.3           2.5            5.0           1.9       2
147            6.5           3.0            5.2           2.0       2
148            6.2           3.4            5.4           2.3       2
149            5.9           3.0            5.1           1.8       2

[150 rows x 5 columns]
```

Out[13]:
```
<AxesSubplot:>
```



Train the classifier

Create Decision Tree Classifier

In [14]:
```
clf = DecisionTreeClassifier(max_depth=5)
```

Split dataset to train and test

In [15]:
```
X=irisData.values.tolist();
y=[];
for row in X:
    y.append(int(row[4]));
    del row[4];
X=pd.Series(X);
y=pd.Series(y);
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shu

X_train_list=X_train.values.tolist();
y_train_list=y_train.values.tolist();
X_test_list=X_test.values.tolist();
y_test_list=y_test.values.tolist();
```

Train The Classifier

In [16]:
```
clf.fit(X_train_list,y_train_list)
```

Predict Class of Test values

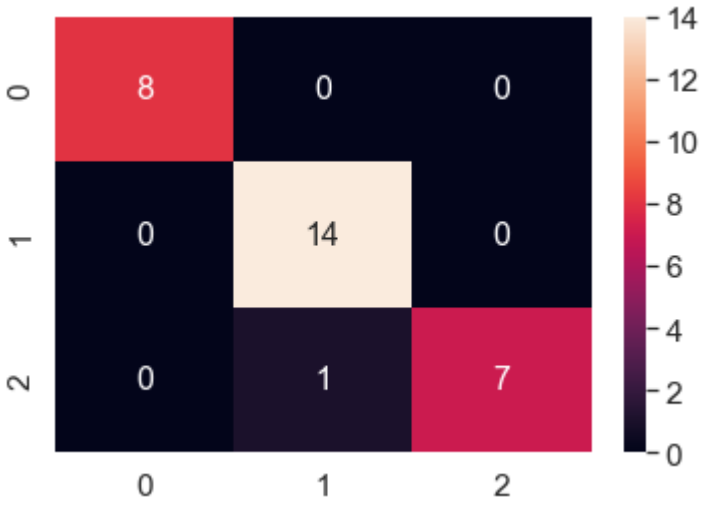
In [17]:
```
yhat = clf.predict(X_test_list)

print("Test Features Expected Classification")
print(y_test_list)

print("Prediction")
print(yhat);

xhat = clf.predict(X_train_list)

print("Train Features Expected Classification")
print(y_train_list)

print("Prediction")
print(xhat);
```

```
Test Features Expected Classification
[1, 1, 2, 1, 1, 1, 2, 2, 1, 0, 1, 1, 1, 2, 1, 1, 0, 2, 2, 1, 0, 0, 0, 1, 1,
0, 0, 2, 0, 2]
Prediction
[1, 1, 1, 1, 1, 1, 2, 2, 1, 0, 1, 1, 1, 2, 1, 1, 0, 2, 2, 1, 0, 0, 0, 1, 1,
0, 0, 2, 0, 2]
Train Features Expected Classification
[2, 2, 2, 0, 0, 0, 2, 2, 1, 0, 2, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 2, 1, 2,
0, 1, 2, 1, 2, 0, 2, 0, 2, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2, 1, 0, 0, 2, 1, 1,
0, 1, 0, 1, 0, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 0, 0, 0, 2, 1, 0, 0, 0, 2, 2,
1, 1, 0, 0, 0, 0, 1, 2, 2, 0, 2, 0, 2, 2, 2, 2, 0, 0, 1, 1, 0, 2, 2, 1, 2,
0, 2, 2, 2, 2, 0, 1, 1, 2, 1, 1, 1, 1, 2, 0, 2, 1, 0, 0, 1]
Prediction
[2, 2, 2, 0, 0, 0, 2, 2, 1, 0, 2, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 2, 1, 2,
0, 1, 2, 1, 2, 0, 2, 0, 2, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2, 1, 0, 0, 2, 1, 1,
0, 1, 0, 1, 0, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 0, 0, 0, 2, 1, 0, 0, 0, 2, 2,
1, 1, 0, 0, 0, 0, 1, 2, 2, 0, 2, 0, 2, 2, 2, 2, 0, 0, 1, 1, 0, 2, 2, 1, 2,
0, 2, 2, 2, 2, 0, 1, 1, 2, 1, 1, 1, 1, 2, 0, 2, 1, 0, 0, 1]
```

Results

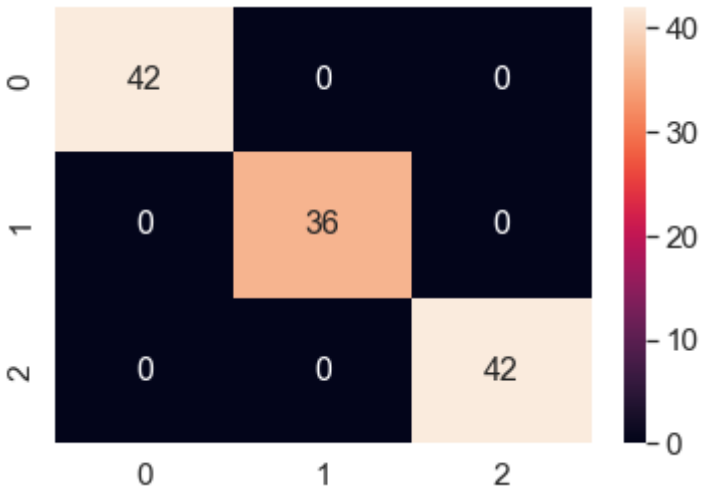
Confusion Matrix of Test

In [18]:
```
y_pred2 = pd.Series(yhat)
y_test2 = pd.Series(y_test_list)

mt = metrics.confusion_matrix(y_test2, y_pred2)
df_cm = pd.DataFrame(mt, range(3), range(3))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16})
plt.show()
```



Confusion Matrix of Train

In [19]:
```
x_pred2 = pd.Series(xhat)
x_test2 = pd.Series(y_train_list)

mt = metrics.confusion_matrix(x_test2, x_pred2)
df_cm = pd.DataFrame(mt, range(3), range(3))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16})
plt.show()
```



F1-Score

In [20]:
```
f1 = metrics.f1_score(y_test2, y_pred2, average='weighted')
print("F1 Score Test")
print(f1)

f2 = metrics.f1_score(x_test2, x_pred2, average='weighted')
```

```
print("F1 Score Train")
print(f2)
```

```
F1 Score Test
0.9661302681992336
F1 Score Train
1.0
```

Accuracy

In [21]:
```
accuracy = metrics.accuracy_score(y_test2, y_pred2)
print("Accuracy Test")
print(accuracy)

accuracy_train = metrics.accuracy_score(x_test2, x_pred2)
print("Accuracy Train")
print(accuracy_train)
```

```
Accuracy Test
0.9666666666666667
Accuracy Train
1.0
```

Precision

In [22]:
```
precision = metrics.precision_score(y_test2, y_pred2, average='weighted')
print("Precision Test")
print(precision)

precision_train = metrics.precision_score(x_test2, x_pred2, average='weighte
print("Precision Train")
print(precision_train)
```

```
Precision Test
0.9688888888888889
Precision Train
1.0
```

Recal

In [23]:
```
recall = metrics.recall_score(y_test2, y_pred2, average='weighted')
print("Recall Test")
print(recall)

recall_train = metrics.recall_score(x_test2, x_pred2, average='weighted')
print("Recall Train")
print(recall_train)
```

```
Recall Test
0.9666666666666667
Recall Train
1.0
```

Plot of ROC Curve (Test/Train) and Value of AUC (Test/Train)

In [24]:
```
y_testb = label_binarize(y_test2, classes=[0, 1, 2])
y_predb = label_binarize(y_pred2, classes=[0, 1, 2])

fpr = dict()
tpr = dict()
roc_auc = dict()
```

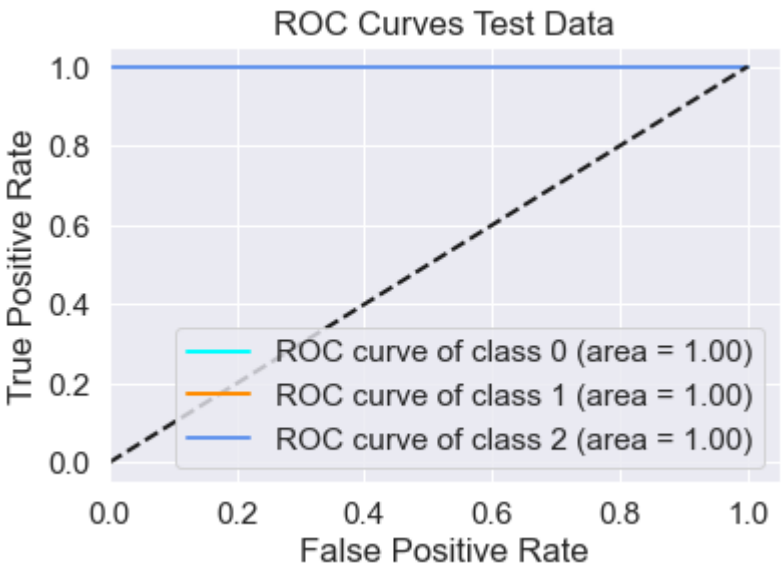
```
for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(y_testb[:,i], y_predb[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(3)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(3):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= 3
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(3), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.xlim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Test Data')
plt.legend(loc="lower right")
plt.show()
print("Auc values of each classes ROC curve are written on graph")
print("Macro Auc value:")
print(roc_auc["macro"])
```


```
Auc values of each classes ROC curve are written on graph
Macro Auc value:
0.96875
```

In [25]:

```
x_testb = label_binarize(x_test2, classes=[0, 1, 2])
x_predb = label_binarize(x_pred2, classes=[0, 1, 2])

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(x_testb[:,i], x_predb[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(3)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(3):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= 3
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(3), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.xlim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Test Data')
plt.legend(loc="lower right")
plt.show()
print("Auc values of each classes ROC curve are written on graph")
print("Macro Auc value:")
print(roc_auc["macro"])
```




```
Auc values of each classes ROC curve are written on graph
Macro Auc value:
1.0
```