Student: Alan Danque  
Date: 20200515

DSC 550

Week 11 Original Project Parts 1, 2, 3 and 4 (Revisions).

Project: COVID 19 factors analysis.

The purpose of my analysis is to analyze if we the US can bend the curve (“BendCurve”, ”No BendCurve”) per the Covid 19 pandemic. While evaluating the factors from CDC data for number of days since first case (up to April 10, 2020), number of confirmed cases, number of deaths, number of recovered by States. In addition to this data other factors added are if the State votes more for Republican / Democrat, if the Governor is Republican or Democrat as well as if the Senior US Senator is a Democrat or Republican.

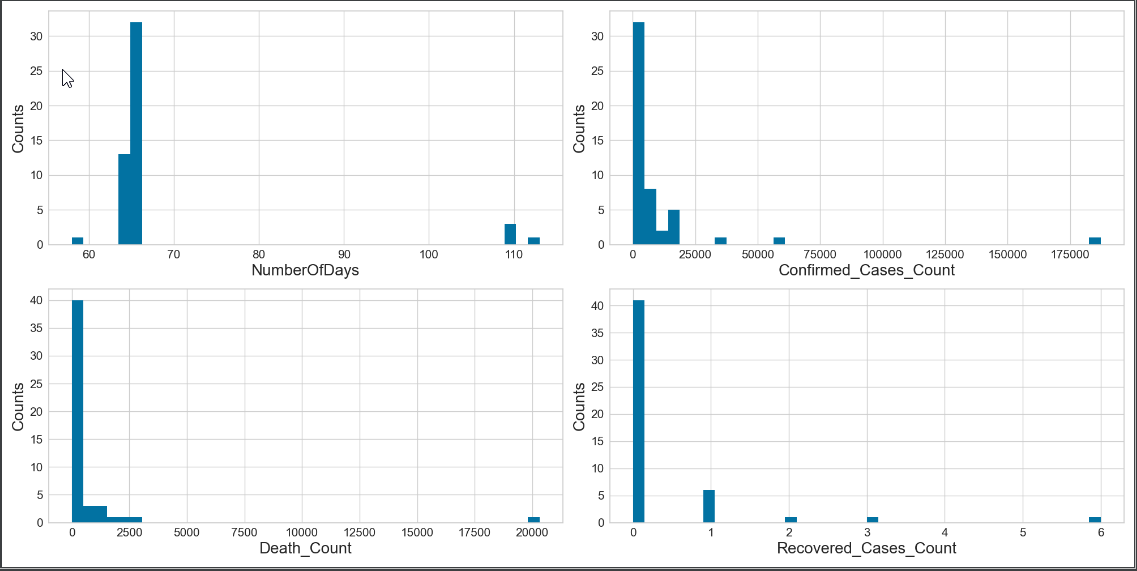
Note: Per peer suggestion from Sam Loyd. I have updated my dataset and have included K-Fold Cross Validation to further review the accuracy of my project.

Covid Data is from:

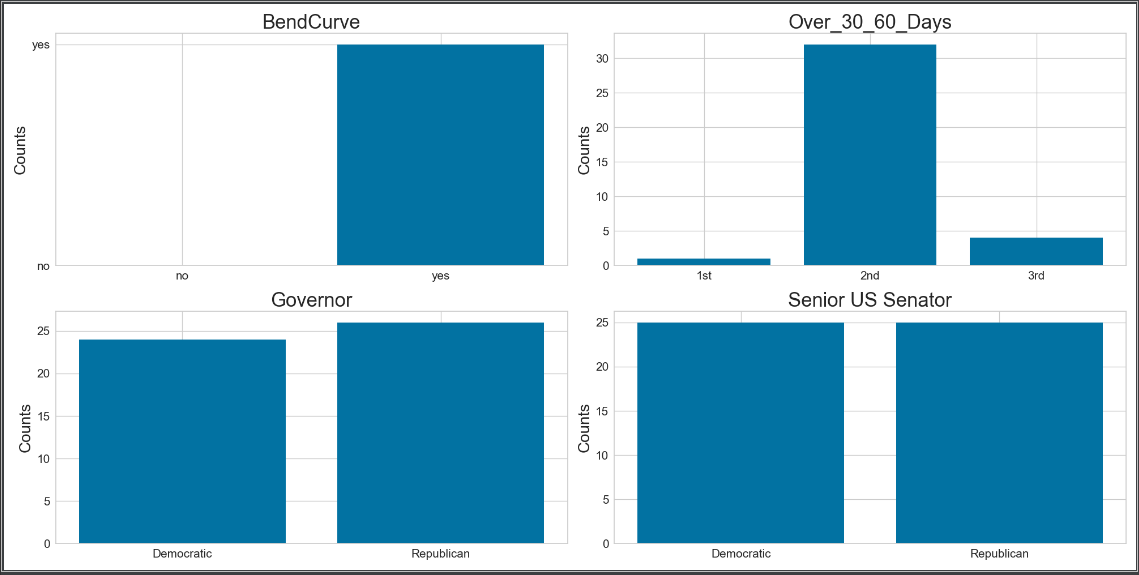
<https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series>

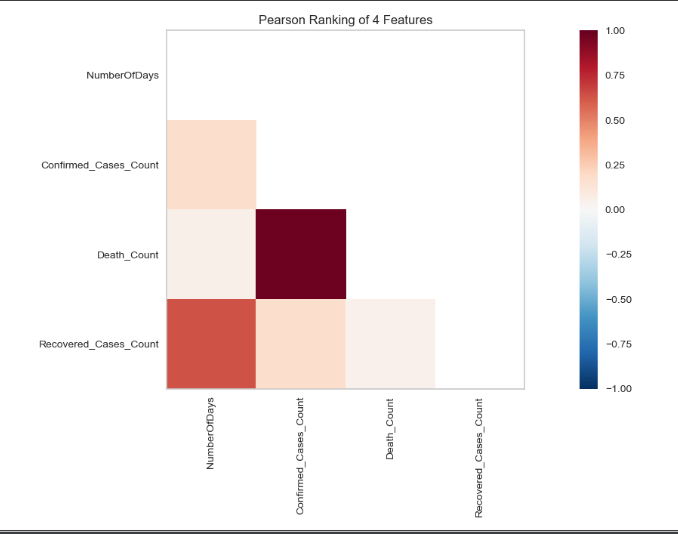
Republic Democratic state information was scraped from:

<https://en.wikipedia.org/wiki/Political_party_strength_in_U.S._states#cite_note-v-20>

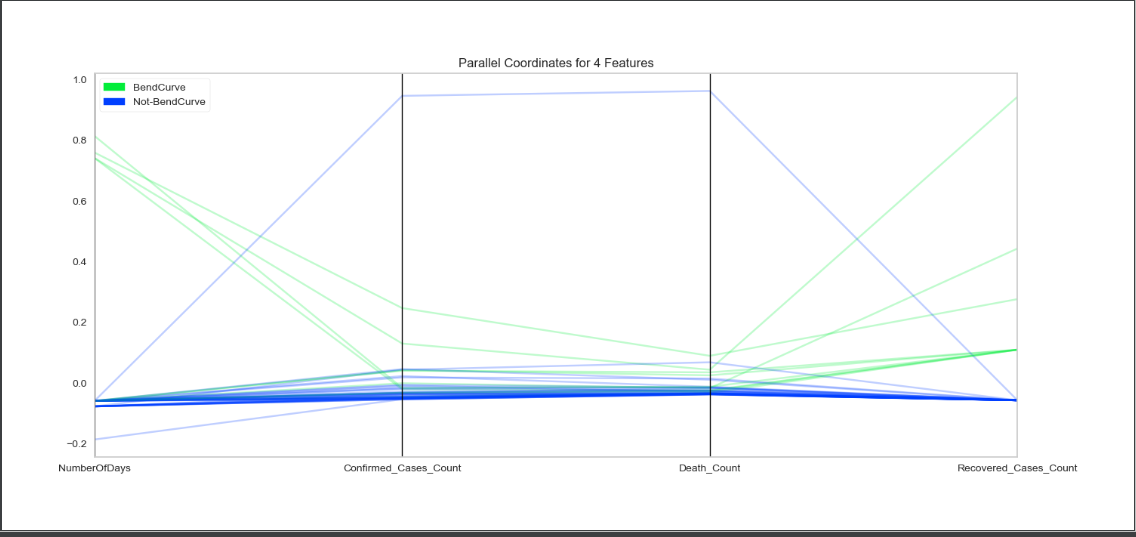


Displays the distribution of counts per the variables NumberOfDays, Confirmed\_Cases\_Count, Death\_Ccounts, Recovered\_Cases\_Count.

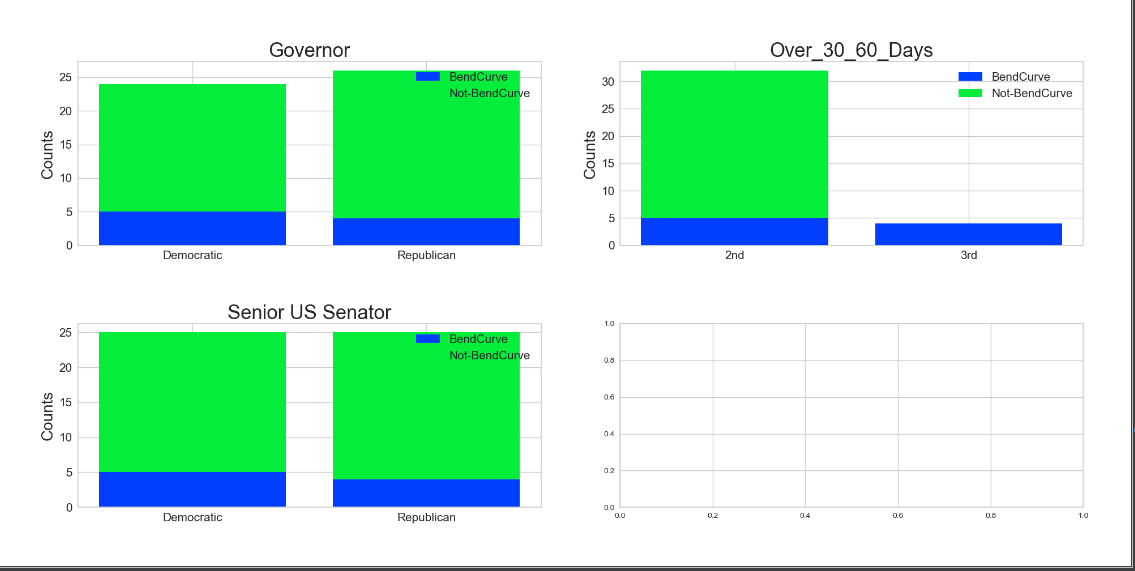




There is a direct correlation between Confirmed\_Cases\_Count and Death\_Counts. Also there is a correlation between Recovered\_Case\_Counts and NumberOfDays (number of days since the first identified case).

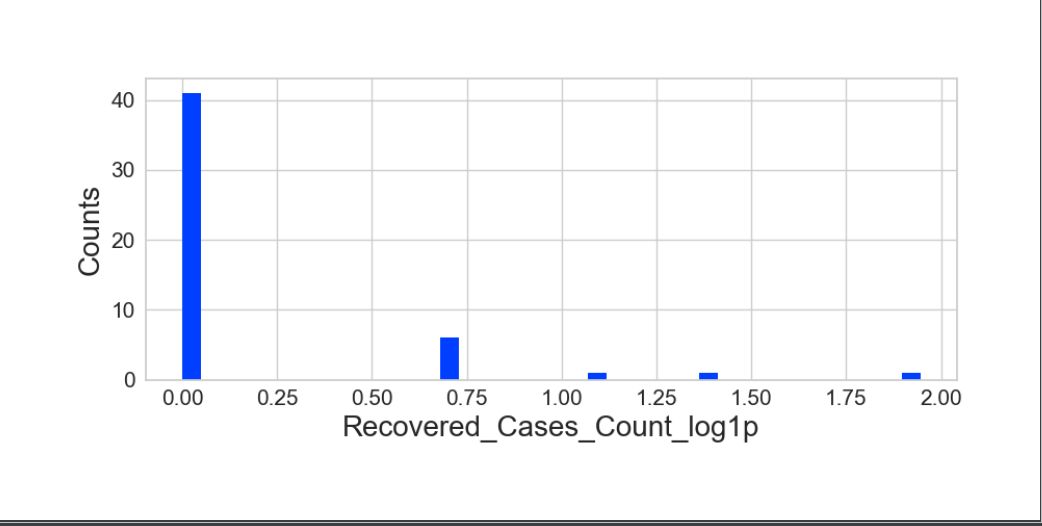


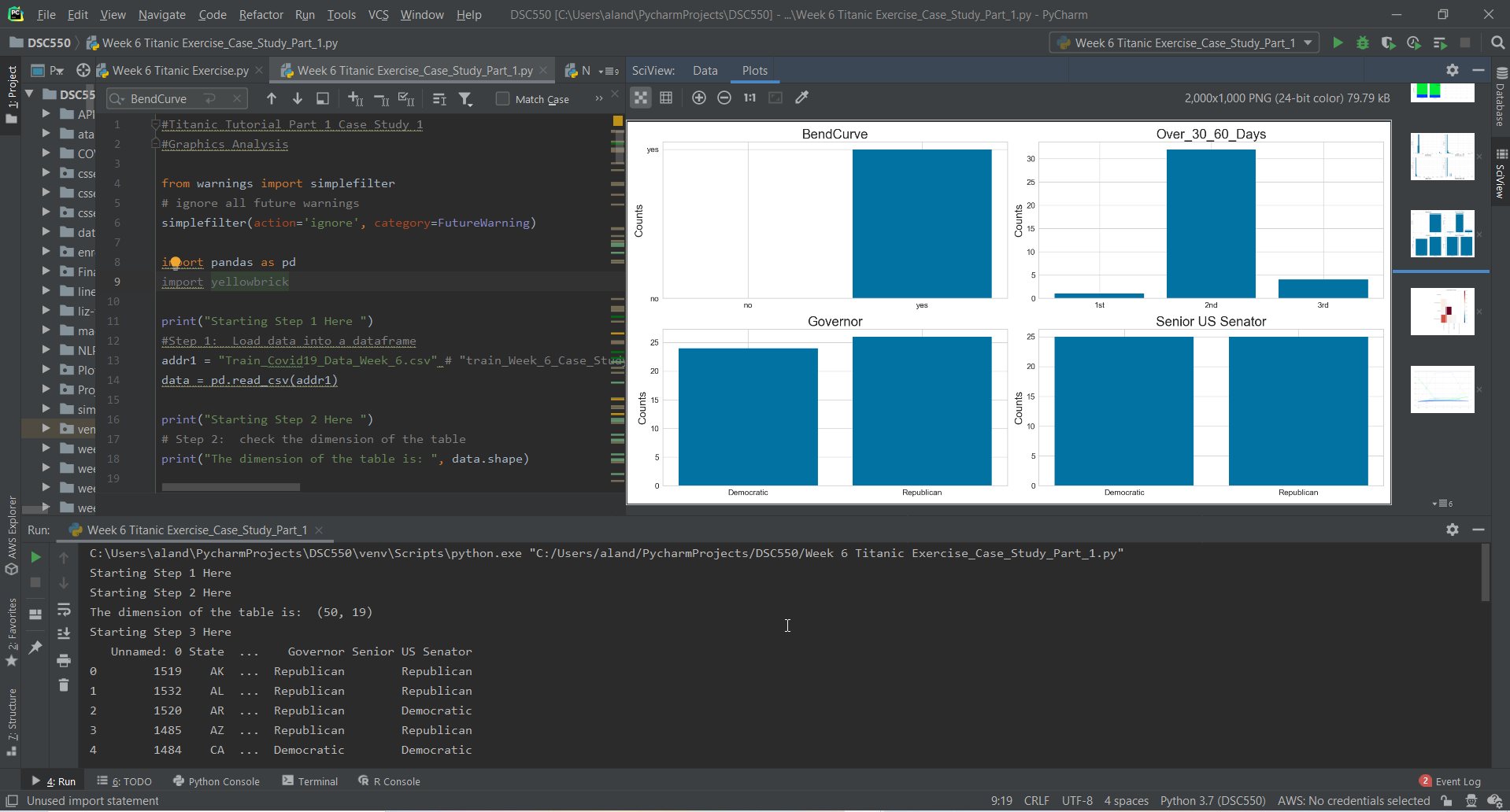
Bending the curve is more likely when Confirmed\_Cases\_Count and Death\_Counts are low. The Curve does appear to bend with higher Recovered\_Case\_Counts with higher NumberOfDays.

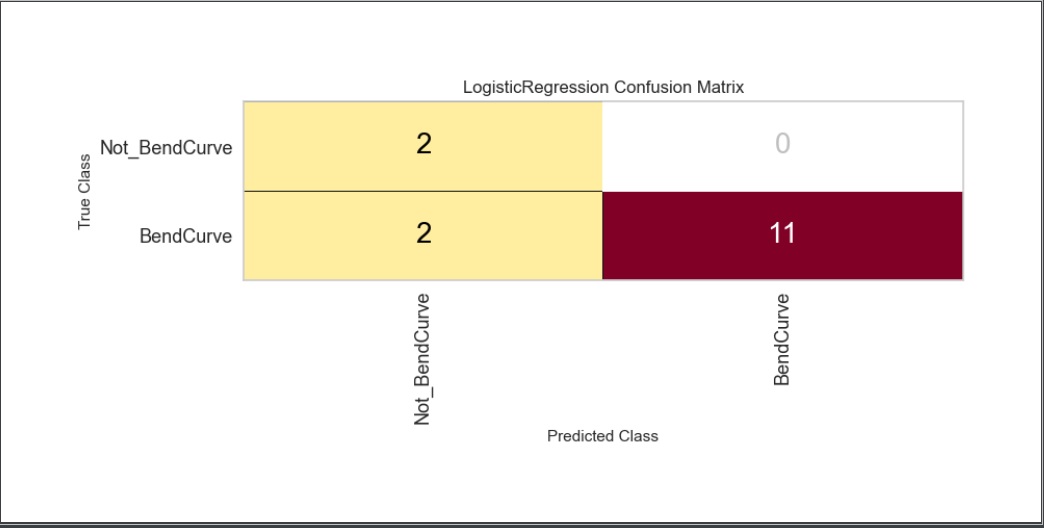


Suggests that Democratic counts are slightly better than Republican counts toward bending the curve.

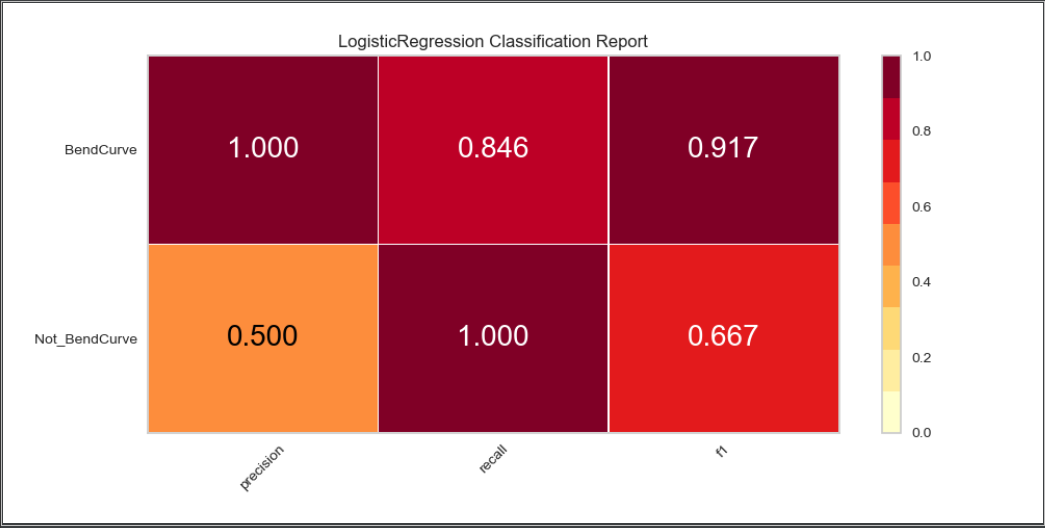
Also shows that it’s not likely to bend the curve within 60 days.



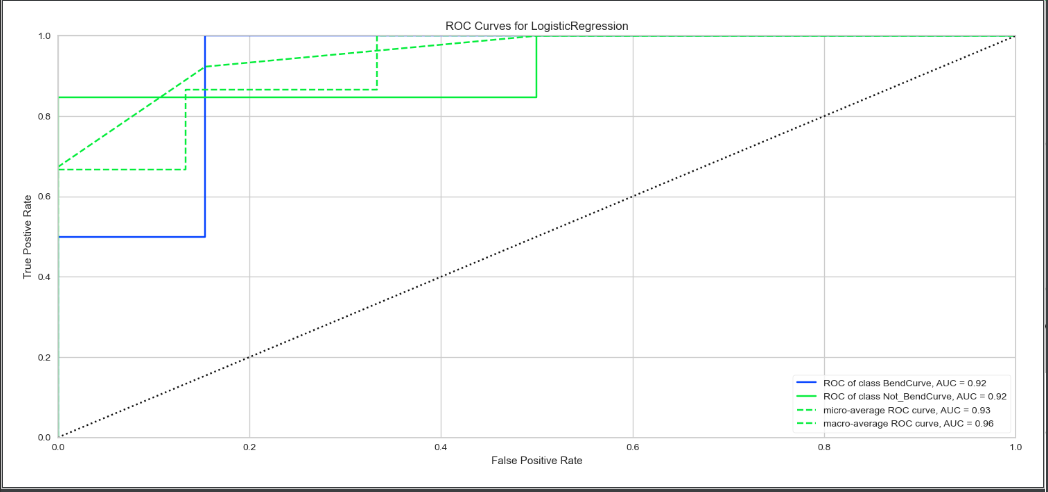


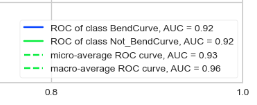


True / Predictions on BendCurve occurred 11 times where 4 prediction of Not\_BendCurve resulted in 2 correct Not\_BendCurve and 2 incorrect BendCurve.

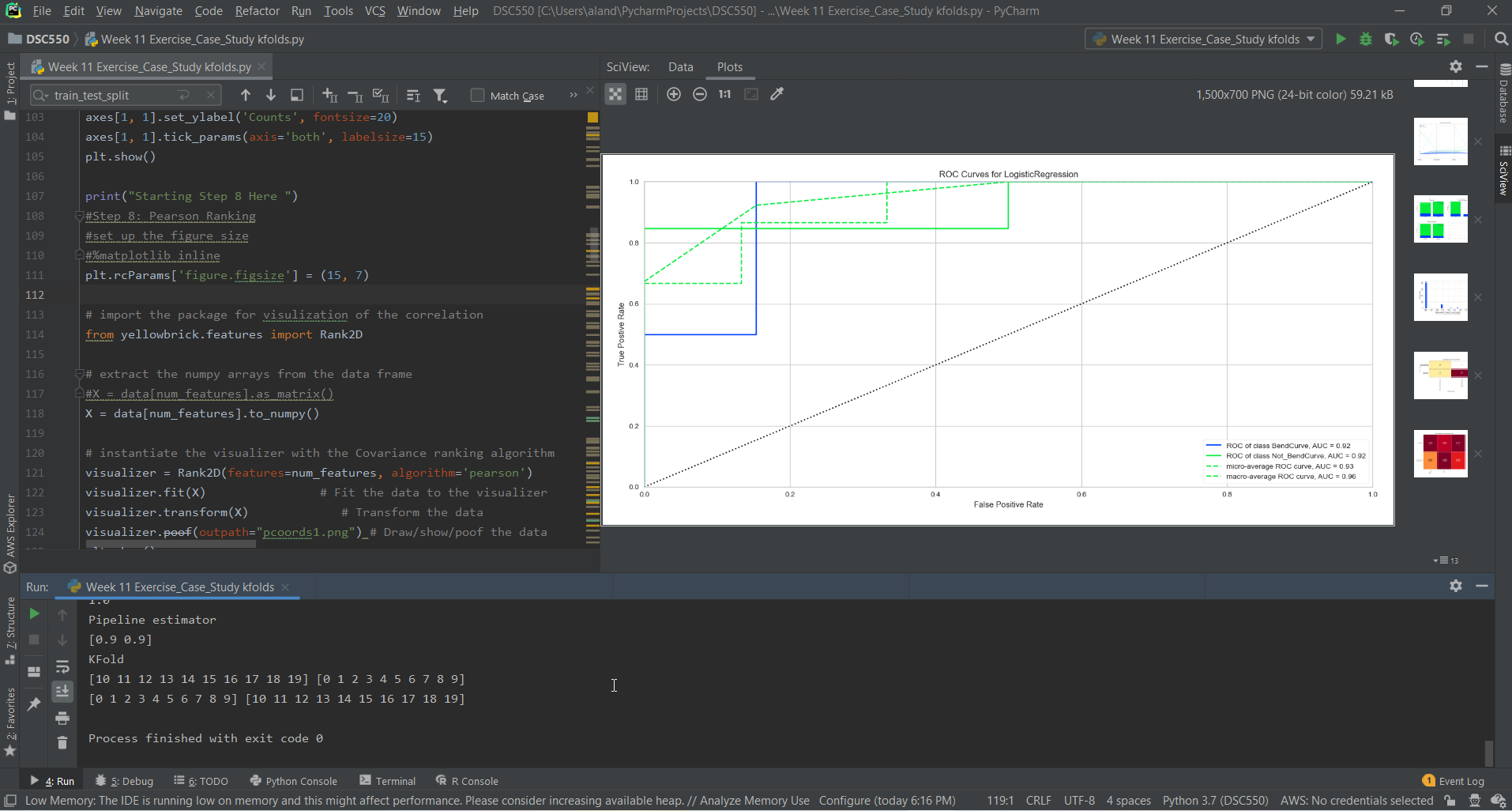


From a classification standpoint, the model was able to classify significantly well per classifying “BendCurve” on tests of precision, recall and F1 score. However, for Not\_BendCurve it only scored well with recall test and yet slightly significant in terms of precision and F1 score.

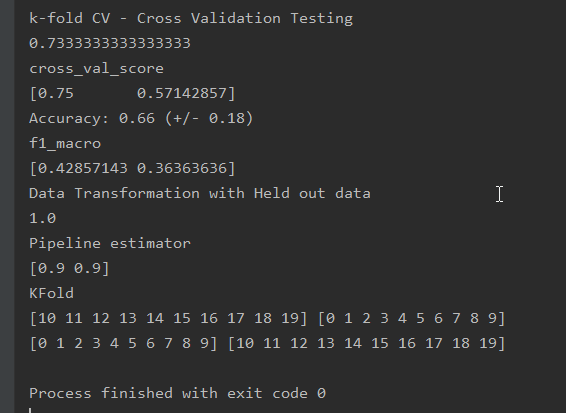




This ROC Curve appears to display that there is a low chance of being inaccurate as is not very close to the 45 degree line.



K-fold Cross Validation



**Code:**

#Graphics Analysis  
  
from warnings import simplefilter  
# ignore all future warnings  
simplefilter(action='ignore', category=FutureWarning)  
  
import pandas as pd  
import yellowbrick  
  
print("Starting Step 1 Here ")  
#Step 1: Load data into a dataframe  
addr1 = "Train\_Covid19\_Data\_Week\_11.csv" # "train\_Week\_6\_Case\_Study\_Part\_1.csv" #"train.csv"  
data = pd.read\_csv(addr1)  
  
print("Starting Step 2 Here ")  
# Step 2: check the dimension of the table  
print("The dimension of the table is: ", data.shape)  
  
print("Starting Step 3 Here ")  
#Step 3: Look at the data  
print(data.head(5))  
  
print("Starting Step 5 Here ")  
#Step 5: what type of variables are in the table  
print("Describe Data")  
print(data.describe())  
print("Summarized Data")  
print(data.describe(include=['O']))  
  
  
print("Starting Step 6 Here ")  
#Step 6: import visulization packages  
import matplotlib.pyplot as plt  
  
# set up the figure size  
plt.rcParams['figure.figsize'] = (20, 10)  
  
# make subplots  
fig, axes = plt.subplots(nrows = 2, ncols = 2)  
  
# Specify the features of interest  
# num\_features = ['Age', 'SibSp', 'Parch', 'Fare']  
num\_features = ['NumberOfDays', 'Confirmed\_Cases\_Count', 'Death\_Count', 'Recovered\_Cases\_Count']  
  
xaxes = num\_features  
# yaxes = ['Counts', 'Counts', 'Counts', 'Counts']  
yaxes = ['Counts', 'Counts', 'Counts', 'Counts']  
  
# draw histograms  
axes = axes.ravel()  
for idx, ax in enumerate(axes):  
 ax.hist(data[num\_features[idx]].dropna(), bins=40)  
 ax.set\_xlabel(xaxes[idx], fontsize=20)  
 ax.set\_ylabel(yaxes[idx], fontsize=20)  
 ax.tick\_params(axis='both', labelsize=15)  
plt.show()  
  
print("Starting Step 7 Here ")  
#7: Barcharts: set up the figure size  
#%matplotlib inline  
plt.rcParams['figure.figsize'] = (20, 10)  
  
# make subplots  
fig, axes = plt.subplots(nrows = 2, ncols = 2)  
  
# make the data read to feed into the visulizer  
#X\_Survived = data.replace({'Survived': {1: 'yes', 0: 'no'}}).groupby('Survived').size().reset\_index(name='Counts')['Survived']  
#Y\_Survived = data.replace({'Survived': {1: 'yes', 0: 'no'}}).groupby('Survived').size().reset\_index(name='Counts')['Counts']  
X\_Survived = data.replace({'BendCurve': {1: 'yes', 0: 'no'}}).groupby('BendCurve').size().reset\_index(name='Counts')['BendCurve']  
Y\_Survived = data.replace({'BendCurve': {1: 'yes', 0: 'no'}}).groupby('BendCurve').size().reset\_index(name='Counts')['BendCurve']  
  
# make the bar plot  
axes[0, 0].bar(X\_Survived, Y\_Survived)  
axes[0, 0].set\_title('BendCurve', fontsize=25)  
axes[0, 0].set\_ylabel('Counts', fontsize=20)  
axes[0, 0].tick\_params(axis='both', labelsize=15)  
  
# make the data read to feed into the visulizer  
X\_Pclass = data.replace({'Over\_30\_60\_Days': {1: '1st', 2: '2nd', 3: '3rd'}}).groupby('Over\_30\_60\_Days').size().reset\_index(name='Counts')['Over\_30\_60\_Days']  
Y\_Pclass = data.replace({'Over\_30\_60\_Days': {1: '1st', 2: '2nd', 3: '3rd'}}).groupby('Over\_30\_60\_Days').size().reset\_index(name='Counts')['Counts']  
# make the bar plot  
axes[0, 1].bar(X\_Pclass, Y\_Pclass)  
axes[0, 1].set\_title('Over\_30\_60\_Days', fontsize=25)  
axes[0, 1].set\_ylabel('Counts', fontsize=20)  
axes[0, 1].tick\_params(axis='both', labelsize=15)  
  
# make the data read to feed into the visulizer  
X\_Sex = data.groupby('Governor').size().reset\_index(name='Counts')['Governor']  
Y\_Sex = data.groupby('Governor').size().reset\_index(name='Counts')['Counts']  
# make the bar plot  
axes[1, 0].bar(X\_Sex, Y\_Sex)  
axes[1, 0].set\_title('Governor', fontsize=25)  
axes[1, 0].set\_ylabel('Counts', fontsize=20)  
axes[1, 0].tick\_params(axis='both', labelsize=15)  
  
# make the data read to feed into the visulizer  
X\_Embarked = data.groupby('Senior US Senator').size().reset\_index(name='Counts')['Senior US Senator']  
Y\_Embarked = data.groupby('Senior US Senator').size().reset\_index(name='Counts')['Counts']  
# make the bar plot  
axes[1, 1].bar(X\_Embarked, Y\_Embarked)  
axes[1, 1].set\_title('Senior US Senator', fontsize=25)  
axes[1, 1].set\_ylabel('Counts', fontsize=20)  
axes[1, 1].tick\_params(axis='both', labelsize=15)  
plt.show()  
  
print("Starting Step 8 Here ")  
#Step 8: Pearson Ranking  
#set up the figure size  
#%matplotlib inline  
plt.rcParams['figure.figsize'] = (15, 7)  
  
# import the package for visulization of the correlation  
from yellowbrick.features import Rank2D  
  
# extract the numpy arrays from the data frame  
#X = data[num\_features].as\_matrix()  
X = data[num\_features].to\_numpy()  
  
# instantiate the visualizer with the Covariance ranking algorithm  
visualizer = Rank2D(features=num\_features, algorithm='pearson')  
visualizer.fit(X) # Fit the data to the visualizer  
visualizer.transform(X) # Transform the data  
visualizer.poof(outpath="pcoords1.png") # Draw/show/poof the data  
plt.show()  
  
print("Starting Step 9 Here ")  
# Step 9: Compare variables against Survived and Not Survived  
#set up the figure size  
#%matplotlib inline  
plt.rcParams['figure.figsize'] = (15, 7)  
plt.rcParams['font.size'] = 50  
  
# setup the color for yellowbrick visulizer  
from yellowbrick.style import set\_palette  
set\_palette('sns\_bright')  
  
# import packages  
from yellowbrick.features import ParallelCoordinates  
# Specify the features of interest and the classes of the target  
classes = ['Not-BendCurve', 'BendCurve']  
num\_features = ['NumberOfDays', 'Confirmed\_Cases\_Count', 'Death\_Count', 'Recovered\_Cases\_Count']  
  
# copy data to a new dataframe  
data\_norm = data.copy()  
# normalize data to 0-1 range  
for feature in num\_features:  
 data\_norm[feature] = (data[feature] - data[feature].mean(skipna=True)) / (data[feature].max(skipna=True) - data[feature].min(skipna=True))  
  
# Extract the numpy arrays from the data frame  
# X = data\_norm[num\_features].as\_matrix()  
# y = data.Survived.as\_matrix()  
X = data\_norm[num\_features].to\_numpy()  
y = data.BendCurve.to\_numpy()  
  
# Instantiate the visualizer  
visualizer = ParallelCoordinates(classes=classes, features=num\_features)  
  
  
visualizer.fit(X, y) # Fit the data to the visualizer  
visualizer.transform(X) # Transform the data  
visualizer.poof(outpath="pcoords2.png") # Draw/show/poof the data  
plt.show();  
  
print("Starting Step 10 Here ")  
# Step 10 - stacked bar charts to compare survived/not survived  
#set up the figure size  
#%matplotlib inline  
plt.rcParams['figure.figsize'] = (20, 10)  
  
# make subplots  
fig, axes = plt.subplots(nrows = 2, ncols = 2)  
  
# make the data read to feed into the visulizer  
Sex\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}})[data['BendCurve']==1]['Governor'].value\_counts()  
Sex\_not\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}})[data['BendCurve']==0]['Governor'].value\_counts()  
Sex\_not\_survived = Sex\_not\_survived.reindex(index = Sex\_survived.index)  
# make the bar plot  
p1 = axes[0, 0].bar(Sex\_survived.index, Sex\_survived.values)  
p2 = axes[0, 0].bar(Sex\_not\_survived.index, Sex\_not\_survived.values, bottom=Sex\_survived.values)  
axes[0, 0].set\_title('Governor', fontsize=25)  
axes[0, 0].set\_ylabel('Counts', fontsize=20)  
axes[0, 0].tick\_params(axis='both', labelsize=15)  
axes[0, 0].legend((p1[0], p2[0]), ('BendCurve', 'Not-BendCurve'), fontsize = 15)  
  
# make the data read to feed into the visualizer  
Pclass\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}}).replace({'Over\_30\_60\_Days': {1: '1st', 2: '2nd', 3: '3rd'}})[data['BendCurve']==1]['Over\_30\_60\_Days'].value\_counts()  
Pclass\_not\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}}).replace({'Over\_30\_60\_Days': {1: '1st', 2: '2nd', 3: '3rd'}})[data['BendCurve']==0]['Over\_30\_60\_Days'].value\_counts()  
Pclass\_not\_survived = Pclass\_not\_survived.reindex(index = Pclass\_survived.index)  
# make the bar plot  
p3 = axes[0, 1].bar(Pclass\_survived.index, Pclass\_survived.values)  
p4 = axes[0, 1].bar(Pclass\_not\_survived.index, Pclass\_not\_survived.values, bottom=Pclass\_survived.values)  
axes[0, 1].set\_title('Over\_30\_60\_Days', fontsize=25)  
axes[0, 1].set\_ylabel('Counts', fontsize=20)  
axes[0, 1].tick\_params(axis='both', labelsize=15)  
axes[0, 1].legend((p3[0], p4[0]), ('BendCurve', 'Not-BendCurve'), fontsize = 15)  
  
# make the data read to feed into the visualizer  
Embarked\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}})[data['BendCurve']==1]['Senior US Senator'].value\_counts()  
Embarked\_not\_survived = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not-BendCurve'}})[data['BendCurve']==0]['Senior US Senator'].value\_counts()  
Embarked\_not\_survived = Embarked\_not\_survived.reindex(index = Embarked\_survived.index)  
# make the bar plot  
p5 = axes[1, 0].bar(Embarked\_survived.index, Embarked\_survived.values)  
p6 = axes[1, 0].bar(Embarked\_not\_survived.index, Embarked\_not\_survived.values, bottom=Embarked\_survived.values)  
axes[1, 0].set\_title('Senior US Senator', fontsize=25)  
axes[1, 0].set\_ylabel('Counts', fontsize=20)  
axes[1, 0].tick\_params(axis='both', labelsize=15)  
axes[1, 0].legend((p5[0], p6[0]), ('BendCurve', 'Not-BendCurve'), fontsize = 15)  
plt.show()  
  
print(" The End of Part 1 ")  
  
print("Starting step 11")  
# Step 11 - fill in missing values and eliminate features  
#fill the missing age data with median value  
def fill\_na\_median(data, inplace=True):  
 return data.fillna(data.median(), inplace=inplace)  
  
fill\_na\_median(data['NumberOfDays'])  
  
# check the result  
print("NumberOfDays column of the data dataframe.")  
print(data['NumberOfDays'].describe())  
  
# fill with the most represented value  
def fill\_na\_most(data, inplace=True):  
 return data.fillna('Democratic', inplace=inplace)  
  
fill\_na\_most(data['Senior US Senator'])  
  
# check the result  
print("Senior US Senator column of the data dataframe.")  
print(data['Senior US Senator'].describe())  
  
# import package  
import numpy as np  
  
# log-transformation  
def log\_transformation(data):  
 return data.apply(np.log1p)  
  
data['Recovered\_Cases\_Count\_log1p'] = log\_transformation(data['Recovered\_Cases\_Count'])  
  
# check the data  
print("Data dataframe.")  
print(data.describe())  
  
print("Starting step 12")  
  
#Step 12 - adjust skewed data (Recovered\_Cases\_Count)  
#check the distribution using histogram  
# set up the figure size  
#%matplotlib inline  
plt.rcParams['figure.figsize'] = (10, 5)  
  
plt.hist(data['Recovered\_Cases\_Count\_log1p'], bins=40)  
plt.xlabel('Recovered\_Cases\_Count\_log1p', fontsize=20)  
plt.ylabel('Counts', fontsize=20)  
plt.tick\_params(axis='both', labelsize=15)  
plt.show()  
  
  
print("Starting step 13")  
#Step 13 - convert categorical data to numbers  
#get the categorical data  
cat\_features = ['Over\_30\_60\_Days', 'Governor', "Senior US Senator"]  
data\_cat = data[cat\_features]  
data\_cat = data\_cat.replace({'Over\_30\_60\_Days': {1: '1st', 2: '2nd', 3: '3rd'}})  
# One Hot Encoding  
print("One Hot Encoding Matrix")  
data\_cat\_dummies = pd.get\_dummies(data\_cat)  
# check the data  
print(data\_cat\_dummies.head(8))  
print("End of Step 13")  
  
print(" The End of Part 2 ")  
  
  
print("Starting step 14")  
#Step 14 - create a whole features dataset that can be used for train and validation data splitting  
# here we will combine the numerical features and the dummie features together  
features\_model = ['NumberOfDays', 'Confirmed\_Cases\_Count', 'Death\_Count', 'Recovered\_Cases\_Count\_log1p']  
#['Age', 'SibSp', 'Parch', 'Fare\_log1p']  
data\_model\_X = pd.concat([data[features\_model], data\_cat\_dummies], axis=1)  
  
# create a whole target dataset that can be used for train and validation data splitting  
data\_model\_y = data.replace({'BendCurve': {1: 'BendCurve', 0: 'Not\_BendCurve'}})['BendCurve']  
# separate data into training and validation and check the details of the datasets  
# import packages  
from sklearn.model\_selection import train\_test\_split  
  
# split the data  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_model\_X, data\_model\_y, test\_size =0.3, random\_state=11)  
  
# number of samples in each set  
print("No. of samples in training set: ", X\_train.shape[0])  
print("No. of samples in validation set:", X\_val.shape[0])  
  
# Survived and not-survived  
print('\n')  
print('No. of BendCurve and not-BendCurve in the training set:')  
print(y\_train.value\_counts())  
  
print('\n')  
print('No. of BendCurve and not-BendCurve in the validation set:')  
print(y\_val.value\_counts())  
  
  
print("Starting step 15")  
# Step 15 - Eval Metrics  
from sklearn.linear\_model import LogisticRegression  
  
from yellowbrick.classifier import ConfusionMatrix  
from yellowbrick.classifier import ClassificationReport  
from yellowbrick.classifier import ROCAUC  
  
# Instantiate the classification model  
model = LogisticRegression(solver='liblinear', max\_iter=60)  
  
#The ConfusionMatrix visualizer taxes a model  
classes = ['Not\_BendCurve','BendCurve']  
cm = ConfusionMatrix(model, fontsize=13, classes=classes, percent=False)  
  
#Fit fits the passed model. This is unnecessary if you pass the visualizer a pre-fitted model  
cm.fit(X\_train, y\_train)  
  
#To create the ConfusionMatrix, we need some test data. Score runs predict() on the data  
#and then creates the confusion\_matrix from scikit learn.  
cm.score(X\_val, y\_val)  
  
# change fontsize of the labels in the figure  
for label in cm.ax.texts:  
 label.set\_size(20)  
  
#How did we do?  
cm.poof(bbox\_inches='tight')  
  
# Precision, Recall, and F1 Score  
# set the size of the figure and the font size  
plt.tight\_layout(rect=[.5, 0.5, .5, 0.05])  
plt.rcParams['figure.figsize'] = (15, 7)  
plt.rcParams['font.size'] = 20  
  
  
  
# Instantiate the visualizer  
visualizer = ClassificationReport(model, classes=classes)  
  
visualizer.fit(X\_train, y\_train) # Fit the training data to the visualizer  
visualizer.score(X\_val, y\_val) # Evaluate the model on the test data  
g = visualizer.poof()  
  
# ROC and AUC  
#Instantiate the visualizer  
visualizer = ROCAUC(model)  
  
visualizer.fit(X\_train, y\_train) # Fit the training data to the visualizer  
visualizer.score(X\_val, y\_val) # Evaluate the model on the test data  
g = visualizer.poof()  
  
print("End of Part 3")  
  
print("k-fold CV - Cross Validation Testing")  
from sklearn import svm  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_model\_X, data\_model\_y, test\_size =0.3, random\_state=11)  
clf = svm.SVC(kernel='linear', C=1).fit(X\_train, y\_train)  
crossvalidate = clf.score(X\_val, y\_val)  
print(crossvalidate)  
  
print("cross\_val\_score")  
from sklearn.model\_selection import cross\_val\_score  
clf = svm.SVC(kernel='linear', C=1)  
scores = cross\_val\_score(clf, X\_val, y\_val, cv=2)  
print(scores)  
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))  
  
print("f1\_macro")  
from sklearn import metrics  
scores = cross\_val\_score(clf, X\_val, y\_val, cv=2, scoring='f1\_macro')  
print(scores)  
  
  
print("Data Transformation with Held out data")  
from sklearn import preprocessing  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_model\_X, data\_model\_y, test\_size =0.4, random\_state=0)  
scaler = preprocessing.StandardScaler().fit(X\_train)  
X\_train\_transformed = scaler.transform(X\_train)  
clf = svm.SVC(C=1).fit(X\_train\_transformed, y\_train)  
X\_test\_transformed = scaler.transform(X\_val)  
heldout = clf.score(X\_test\_transformed, y\_val)  
print(heldout)  
  
  
print("Pipeline estimator")  
from sklearn.pipeline import make\_pipeline  
clf = make\_pipeline(preprocessing.StandardScaler(), svm.SVC(C=1))  
pipeline = cross\_val\_score(clf, X\_val, y\_val, cv=2)  
print(pipeline)  
  
print("KFold")  
from sklearn.model\_selection import KFold  
kf = KFold(n\_splits=2)  
for train, test in kf.split(X\_val):  
 print("%s %s" % (train, test))

**Results**

**Starting Step 1 Here**

**Starting Step 2 Here**

**The dimension of the table is: (50, 23)**

**Starting Step 3 Here**

**Unnamed: 0 State ... Junior U.S. Senator U.S. House of Representatives**

**0 3203 AK ... Democratic Republican 6?1**

**1 3216 AL ... Republican Republican**

**2 3204 AR ... Republican Democratic 5?4**

**3 3169 AZ ... Republican Republican 4**

**4 3168 CA ... Democratic Democratic 46?7**

**[5 rows x 23 columns]**

**Starting Step 5 Here**

**Describe Data**

**Unnamed: 0 NumberOfDays ... Over\_30\_60\_Days Party**

**count 50.000000 50.000000 ... 37.000000 50.000000**

**mean 3191.080000 68.220000 ... 2.081081 0.400000**

**std 15.005904 12.571899 ... 0.363500 0.494872**

**min 3166.000000 58.000000 ... 1.000000 0.000000**

**25% 3178.250000 64.000000 ... 2.000000 0.000000**

**50% 3191.500000 65.000000 ... 2.000000 0.000000**

**75% 3203.750000 65.000000 ... 2.000000 1.000000**

**max 3216.000000 113.000000 ... 3.000000 1.000000**

**[8 rows x 11 columns]**

**Summarized Data**

**State Date\_Recorded ... Junior U.S. Senator U.S. House of Representatives**

**count 50 50 ... 50 50**

**unique 50 1 ... 3 34**

**top TX 5/14/2020 ... Republican Republican**

**freq 1 50 ... 28 6**

**[4 rows x 12 columns]**

**Starting Step 6 Here**

**Starting Step 7 Here**

**Starting Step 8 Here**

**Starting Step 9 Here**

**Starting Step 10 Here**

**The End of Part 1**

**Starting step 11**

**NumberOfDays column of the data dataframe.**

**count 50.000000**

**mean 68.220000**

**std 12.571899**

**min 58.000000**

**25% 64.000000**

**50% 65.000000**

**75% 65.000000**

**max 113.000000**

**Name: NumberOfDays, dtype: float64**

**Senior US Senator column of the data dataframe.**

**count 50**

**unique 2**

**top Democratic**

**freq 25**

**Name: Senior US Senator, dtype: object**

**Data dataframe.**

**Unnamed: 0 NumberOfDays ... Party Recovered\_Cases\_Count\_log1p**

**count 50.000000 50.000000 ... 50.000000 50.000000**

**mean 3191.080000 68.220000 ... 0.400000 0.171794**

**std 15.005904 12.571899 ... 0.494872 0.412122**

**min 3166.000000 58.000000 ... 0.000000 0.000000**

**25% 3178.250000 64.000000 ... 0.000000 0.000000**

**50% 3191.500000 65.000000 ... 0.000000 0.000000**

**75% 3203.750000 65.000000 ... 1.000000 0.000000**

**max 3216.000000 113.000000 ... 1.000000 1.945910**

**[8 rows x 12 columns]**

**Starting step 12**

**Starting step 13**

**One Hot Encoding Matrix**

**Over\_30\_60\_Days\_1st ... Senior US Senator\_Republican**

**0 1 ... 1**

**1 0 ... 1**

**2 0 ... 0**

**3 0 ... 1**

**4 0 ... 0**

**5 0 ... 0**

**6 0 ... 0**

**7 0 ... 0**

**[8 rows x 7 columns]**

**End of Step 13**

**The End of Part 2**

**Starting step 14**

**No. of samples in training set: 35**

**No. of samples in validation set: 15**

**No. of BendCurve and not-BendCurve in the training set:**

**Not\_BendCurve 28**

**BendCurve 7**

**Name: BendCurve, dtype: int64**

**No. of BendCurve and not-BendCurve in the validation set:**

**Not\_BendCurve 13**

**BendCurve 2**

**Name: BendCurve, dtype: int64**

**Starting step 15**

**End of Part 3**

**k-fold CV - Cross Validation Testing**

**0.7333333333333333**

**cross\_val\_score**

**[0.75 0.57142857]**

**Accuracy: 0.66 (+/- 0.18)**

**f1\_macro**

**[0.42857143 0.36363636]**

**Data Transformation with Held out data**

**1.0**

**Pipeline estimator**

**[0.9 0.9]**

**KFold**

**[10 11 12 13 14 15 16 17 18 19] [0 1 2 3 4 5 6 7 8 9]**

**[0 1 2 3 4 5 6 7 8 9] [10 11 12 13 14 15 16 17 18 19]**

**Process finished with exit code 0**