SVM Lab

Data Handling

In [21]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
#get rid of this last import loop if its unecessary
```

In [22]:
```
data = pd.read_csv('/Users/lukehenry/Documents/Jupyter/SVM/data.csv') # Read
df = pd.DataFrame(data) # Converting data to Panda DataFrame
```

In [23]:
```
# Cleaning the data
cols = ['nbr','ndbi','ndmi','ndsi','ndvi','ndwi']
for col in cols:
    print(df[col].value_counts())
    df = df.drop([col], axis=1)
df['LST'] = df['LST'].fillna(df['LST'].mean())
```

```
-1    1048575
Name: nbr, dtype: int64
-1    1048575
Name: ndbi, dtype: int64
-1    1048575
Name: ndmi, dtype: int64
-1    1048575
Name: ndsi, dtype: int64
-1    1048575
Name: ndvi, dtype: int64
-1    1048575
Name: ndwi, dtype: int64
```

Data Visualization

In [24]:
```
df.head(100000)
```

Out[24]:

	Unnamed: 0	LST	Unnamed: 0.1	X_Coor	Y_Coor	band1	band2	band3
0	0	23.794017	0	629719.8862	3959150.278	1449	1556	1750
1	1	23.794017	1	629679.8862	3959140.278	1269	1445	1598
2	2	21.481792	2	629689.8862	3959140.278	1286	1447	1562
3	3	21.481792	3	629699.8862	3959140.278	1310	1445	1601
4	4	21.481792	4	629709.8862	3959140.278	1365	1507	1703
...	...	...	...	...	...	...	...	...
99995	99995	17.958636	99995	618779.8862	3957760.278	2239	2363	2583
99996	99996	17.958636	99996	618789.8862	3957760.278	2212	2361	2520
99997	99997	17.958636	99997	618799.8862	3957760.278	2178	2294	2478
99998	99998	18.360789	99998	618809.8862	3957760.278	2247	2344	2554
99999	99999	23.591818	99999	618819.8862	3957760.278	2382	2459	2653

100000 rows × 17 columns

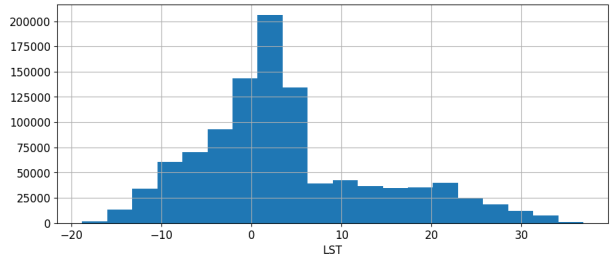
In [25]:
```
print(df['label'].value_counts())
```

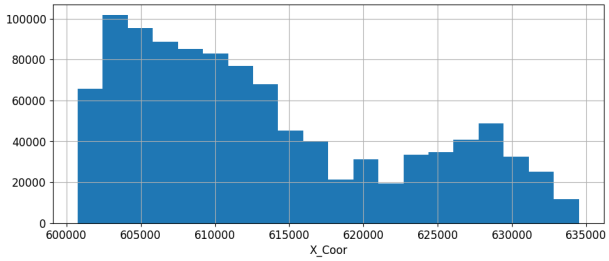
```
0    999999
1     48576
Name: label, dtype: int64
```

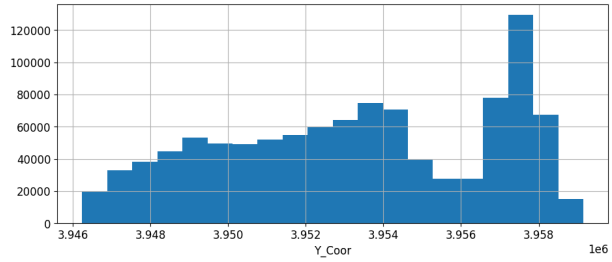
In [26]:
```
# check for missing values in variables

df.isnull().sum()
```

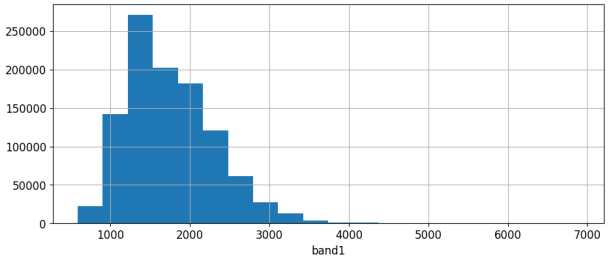
Out[26]:
```
Unnamed: 0      0
LST             0
Unnamed: 0.1    0
X_Coor          0
Y_Coor          0
band1           0
band2           0
band3           0
band4           0
band5           0
band6           0
band7           0
band8           0
band9           0
band10          0
POINTID         0
label           0
dtype: int64
```

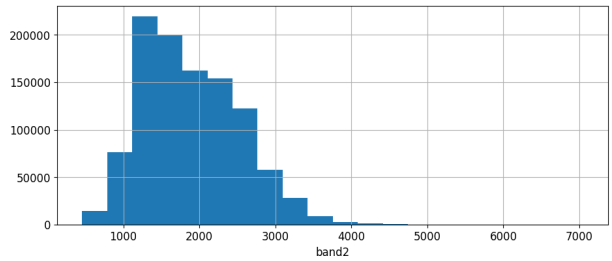
In [27]:
```
# plot histogram to check distribution

df0 = df.drop(['Unnamed: 0', 'Unnamed: 0.1', 'POINTID','label'], axis=1)
df1 = df0.iloc[:, :7]  # first data frame with 7 columns
df2 = df0.iloc[:, 7:]  # second data frame with 6 columns
count = 1
plt.figure(figsize=(24,20))
for col in df1:
    plt.subplot(4, 2, count)
    fig = df[col].hist(bins=20)
    fig.set_xlabel(col)
    count = count + 1

count = 1
plt.figure(figsize=(24,20))
for col in df2:
    plt.subplot(4, 2, count)
    fig = df[col].hist(bins=20)
    fig.set_xlabel(col)
    count = count + 1
```

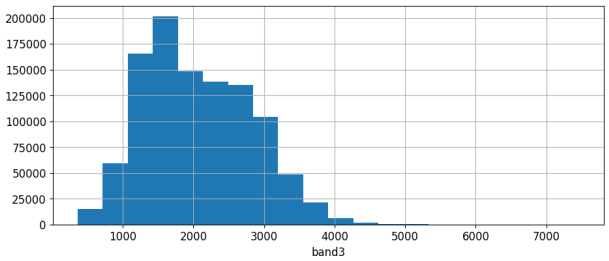

LST

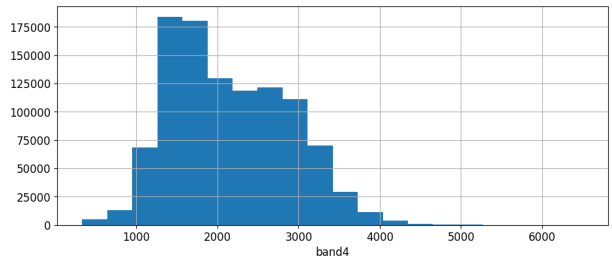

X_Coor

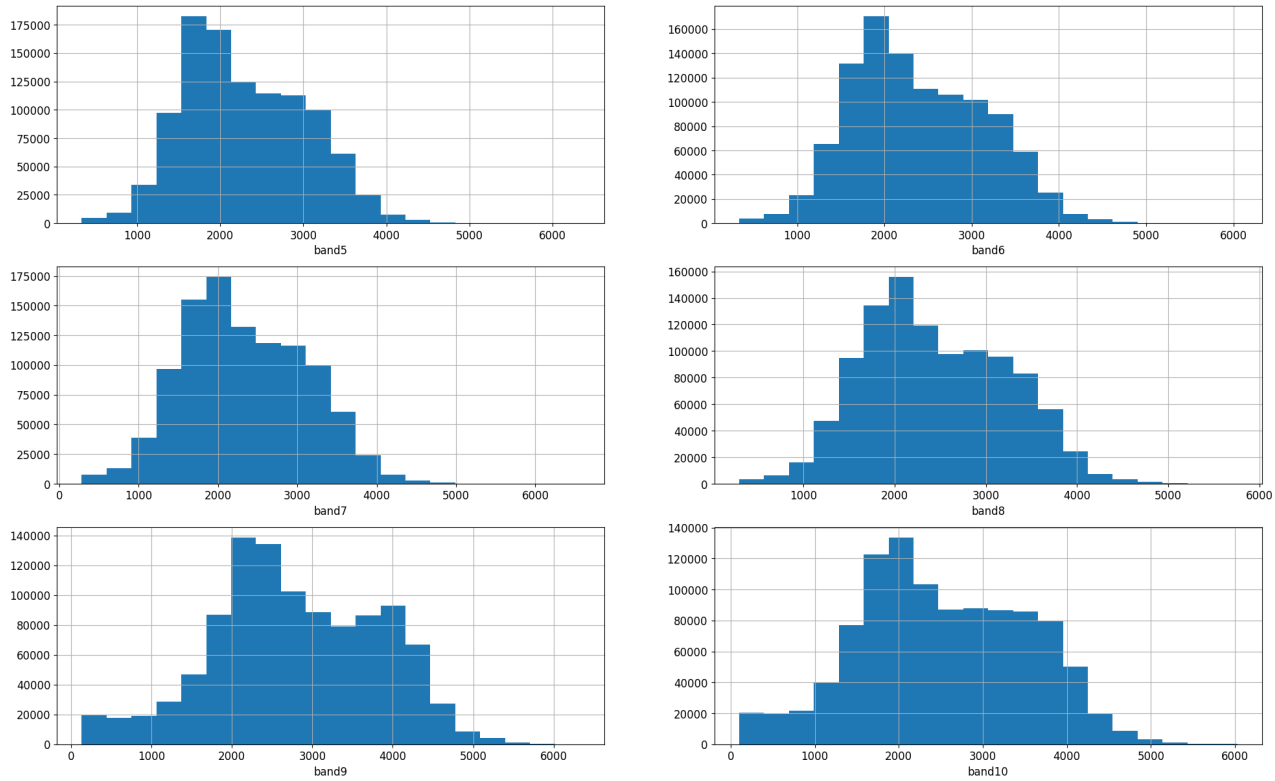

Y_Coor
1e6


band1


band2


band3


band4



Splitting the data

In [28]:
```
X = df.drop(['label'], axis=1)
y = df['label']
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, r
X_train.shape, X_test.shape
```

Out[28]:
```
((734002, 16), (314573, 16))
```

Scaling the data

In [29]:
```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
X_train = scaling.transform(X_train)
#SCALING THE TEST MIGHT BE HARMFUL
X_test = scaling.transform(X_test)
```

In [30]:
```
# look up whay the minmax sclaler does and why i chose to use it to speed up
```

SVM Modeling

In [31]:
```
# import SVC classifier
from sklearn.svm import SVC
# import metrics to compute accuracy
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Default Hyper Parameters: C = 1.0

In [32]:
```
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(
```

```
Model accuracy score with default hyperparameters: 1.0000
```

Increasing the C value, so less outliers: C = 1000.0

In [33]:
```
# instantiate classifier with rbf kernel and C=1000
svc=SVC(C=1000.0)
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=1000.0 : {0:0.4f}'. format
```

```
Model accuracy score with rbf kernel and C=1000.0 : 1.0000
```

Linear Kernal, C = 1000.0

In [34]:
```
# instantiate classifier with linear kernel and C=1000.0
linear_svc=LinearSVC(C=1000.0)
# fit classifier to training set
linear_svc.fit(X_train,y_train)
# make predictions on test set
y_pred_test=linear_svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. for
```

```
Model accuracy score with linear kernel and C=1000.0 : 1.0000
```

Comparing the train and test accuracy

In [35]:
```
linear_svc=LinearSVC()
linear_svc.fit(X_train,y_train)
y_pred_train = linear_svc.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_trair
```

```
Training-set accuracy score: 1.0000
```

Check for overfitting and underfitting

In [36]:
```
print('Training set score: {:.4f}'.format(linear_svc.score(X_train, y_train)
print('Test set score: {:.4f}'.format(linear_svc.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 1.0000
```

Polynomial Kernel

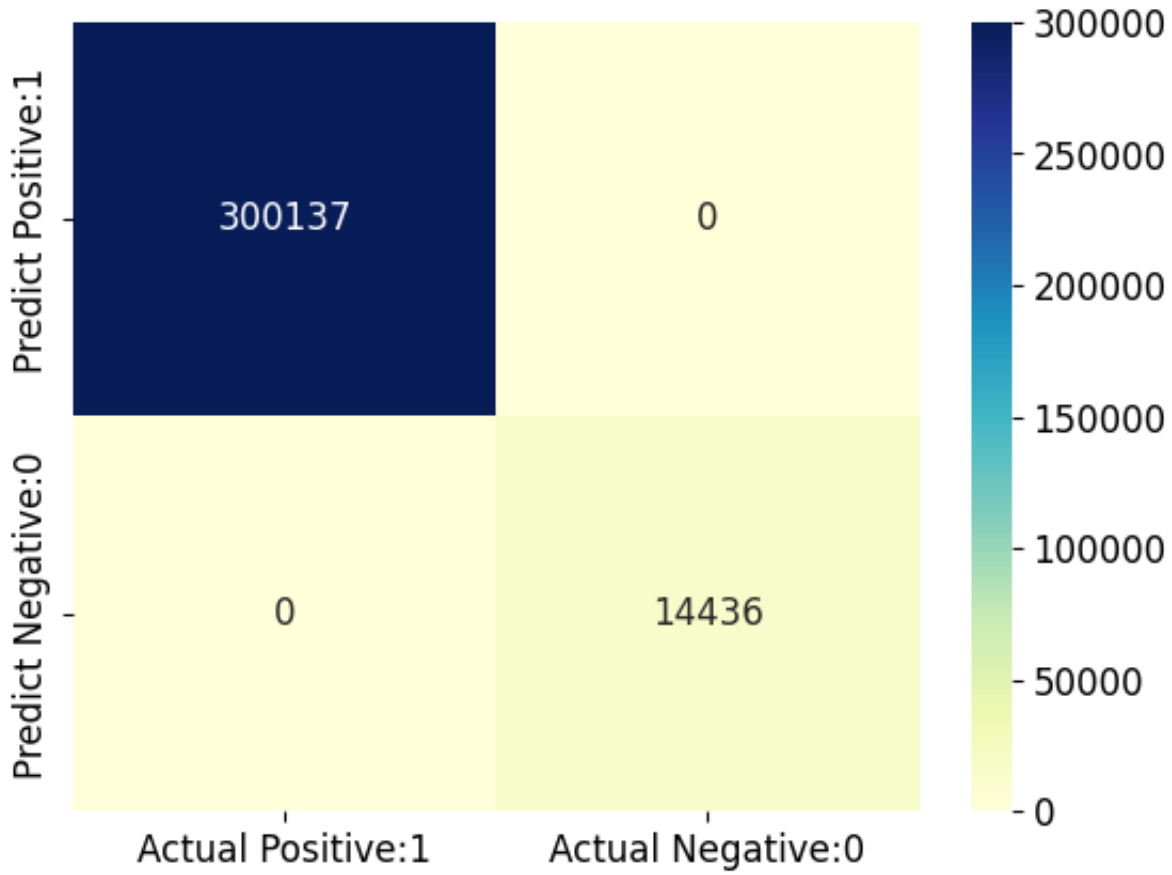
In [37]:
```
# instantiate classifier with polynomial kernel and C=1.0
poly_svc=SVC(kernel='poly', C=1000.0)
# fit classifier to training set
poly_svc.fit(X_train,y_train)
# make predictions on test set
y_pred=poly_svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1000.0 : {0:0.4f}'.
```

```
Model accuracy score with polynomial kernel and C=1000.0 : 1.0000
```

Model Findings

In [38]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
# visualize confusion matrix with seaborn heatmap
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Nega
                                 index=['Predict Positive:1', 'Predict Negat

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

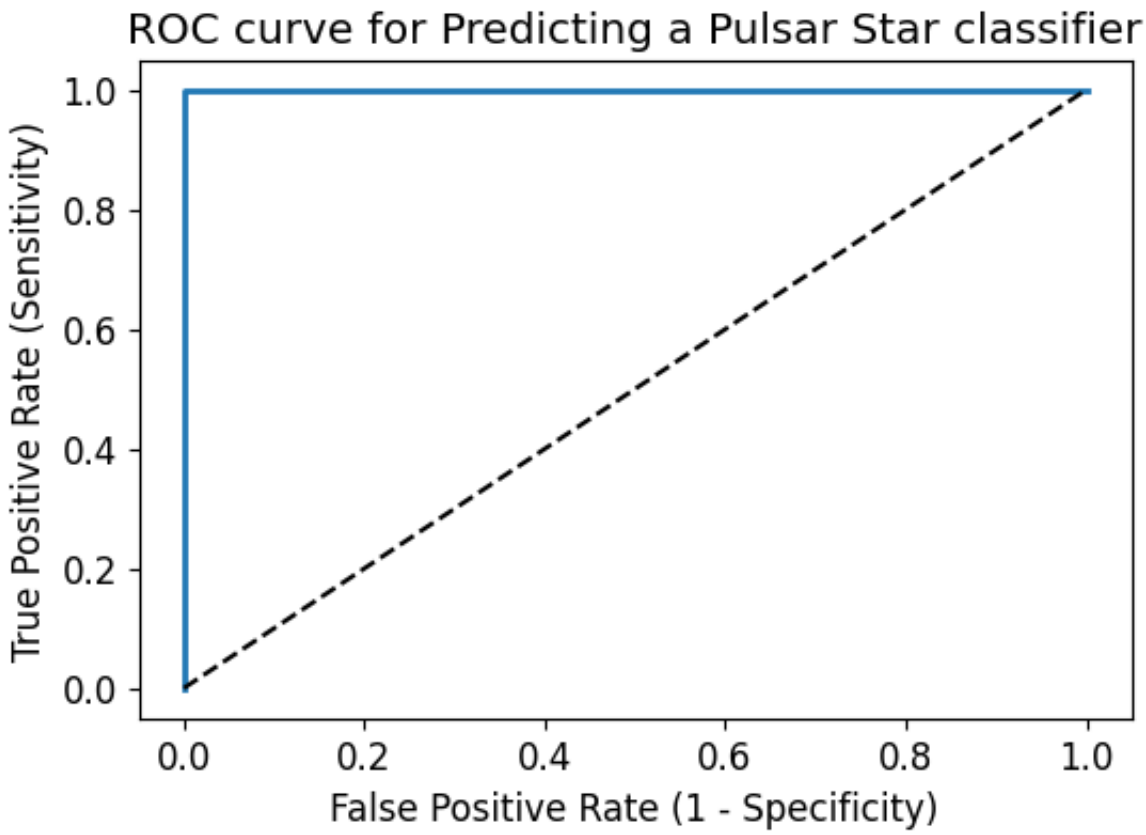
Out[38]:
```
<Axes: >
```



In [39]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_test))
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
# print classification accuracy
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
# print classification error
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    300137
           1       1.00      1.00      1.00     14436

    accuracy                           1.00    314573
   macro avg       1.00      1.00      1.00    314573
weighted avg       1.00      1.00      1.00    314573

Classification accuracy : 1.0000
Classification error : 0.0000
```

In [40]:
```
# plot ROC Curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0,1], [0,1], 'k--' )
plt.rcParams['font.size'] = 12
plt.title('ROC curve for Predicting a Pulsar Star classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.show()
# compute ROC AUC
from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test, y_pred_test)
print('ROC AUC : {:.4f}'.format(ROC_AUC))
```




```
ROC AUC : 1.0000
```

What I did to prepare the data

- Handle the null values
  - One thing that I did to prepare the data was to fill in the null values with the column mean
- Remove uneccessary columns
  - When first looking at the data, I saw there there were six columns that had supsicious values: 'nbr', 'ndbi', 'ndmi', 'ndsi', 'ndvi', 'ndwi'. I ran the value counts for each column and found that each column acted as a constant of -1. Since the values were constants, they do not help our model to classify, so I dropped them
- Handling potential outliers in the data
  - To handle outliers in the data, I set the C score to 1000 when running the different kernals
- Using a preproccessing scaler for my data before modeling
  - I used MinMaxScaler to scale the features of X_train to the range of -1 to 1
  - I used the fit method of MinMaxSclaer on X_train to compute the minimum and maximum values of the features in X_train. These values are then used to scale the feature values during the next step: transforming.
  - I used the transform method (again from MinMaxScaler) to tansfer the fit scale into specific variables
  - I researched to make sure that this preprocessing method was not harmful to my modeling result
- Preppring my data to make my plots
  - Made a dataframe where I dropped the columns that represent IDs or constants
  - Broke that data frame into two so that I could create all the graphs by looping through each of them. Creating two loops like this was the best formatting for the plots

Sources

SVM Tutorial: https://www.kaggle.com/code/prashant111/svm-classifier-tutorial/notebook
Value Counter: https://www.sharpsightlabs.com/blog/pandas-value_counts/
MinMaxScaler: https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/