Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Import Dataset

In [2]:
```
df = pd.read_csv(r'C:\Users\DELL\Desktop\FSDS\ML\6th - SVM\4th - SVM\SVM\pulsar_
df
```

Out[2]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573
...	...	...	...	...	...	...	...
17893	136.429688	59.847421	-0.187846	-0.738123	1.296823	12.166062	15.450260
17894	122.554688	49.485605	0.127978	0.323061	16.409699	44.626893	2.945244
17895	119.335938	59.935939	0.159363	-0.743025	21.430602	58.872000	2.499517
17896	114.507812	53.902400	0.201161	-0.024789	1.946488	13.381731	10.007967
17897	57.062500	85.797340	1.406391	0.089520	188.306020	64.712562	-1.597527

17898 rows × 9 columns

EDA

In [3]:
```
# view dimensions of dataset
df.shape
```

Out[3]: (17898, 9)

In [4]:
```
# let's preview the dataset
df.head()
```

Out[4]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Ske DM
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.2
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.3
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.1
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.5
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.5

In [5]:
```
# view the column names of the dataframe
col_names = df.columns
col_names
```

Out[5]:
```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

In [6]:
```
# remove leading spaces from column names
df.columns = df.columns.str.strip()
```

In [7]:
```
# view column names again
df.columns
```

Out[7]:
```
Index(['Mean of the integrated profile',
       'Standard deviation of the integrated profile',
       'Excess kurtosis of the integrated profile',
       'Skewness of the integrated profile', 'Mean of the DM-SNR curve',
       'Standard deviation of the DM-SNR curve',
       'Excess kurtosis of the DM-SNR curve', 'Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

In [8]:
```
# rename column names
df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',
              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness',
```

In [9]:
```
# view the renamed column names
df.columns
```

Out[9]:
```
Index(['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness', 'DM-SNR Mean',
       'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class'],
      dtype='object')
```

In [10]:
```
# check distribution of target_class column
df['target_class'].value_counts()
```

Out[10]:
```
target_class
0    16259
1     1639
Name: count, dtype: int64
```

In [11]:
```
# view the percentage distribution of target_class column
df['target_class'].value_counts()/np.float(len(df))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[11], line 2
      1 # view the percentage distribution of target_class column
----> 2 df['target_class'].value_counts()/np.float(len(df))

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:324, in __getattr__(attr)
    319     warnings.warn(
    320         f"In the future `np.{attr}` will be defined as the "
    321         "corresponding NumPy scalar.", FutureWarning, stacklevel=2)
    323 if attr in __former_attrs__:
--> 324     raise AttributeError(__former_attrs__[attr])
    326 if attr == 'testing':
    327     import numpy.testing as testing

AttributeError: module 'numpy' has no attribute 'float'.
`np.float` was a deprecated alias for the builtin `float`. To avoid this error in
existing code, use `float` by itself. Doing this will not modify any behavior and
is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
The aliases was originally deprecated in NumPy 1.20; for more details and guidanc
e see the original release note at:
    https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

In [12]:
```
df['target_class'].value_counts() / float(len(df))
```

Out[12]:
```
target_class
0    0.908426
1    0.091574
Name: count, dtype: float64
```

In [13]:
```
# view summary of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   IP Mean            17898 non-null  float64
 1   IP Sd              17898 non-null  float64
 2   IP Kurtosis        17898 non-null  float64
 3   IP Skewness        17898 non-null  float64
 4   DM-SNR Mean        17898 non-null  float64
 5   DM-SNR Sd          17898 non-null  float64
 6   DM-SNR Kurtosis    17898 non-null  float64
 7   DM-SNR Skewness    17898 non-null  float64
 8   target_class       17898 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

In [14]:
```
# check for missing values in variables
df.isnull().sum()
```

Out[14]:
```
IP Mean              0
IP Sd                0
IP Kurtosis          0
IP Skewness          0
DM-SNR Mean          0
DM-SNR Sd            0
DM-SNR Kurtosis      0
DM-SNR Skewness      0
target_class         0
dtype: int64
```

In [15]:
```
# view summary statistics in numerical variables
round(df.describe(),2)
```

Out[15]:

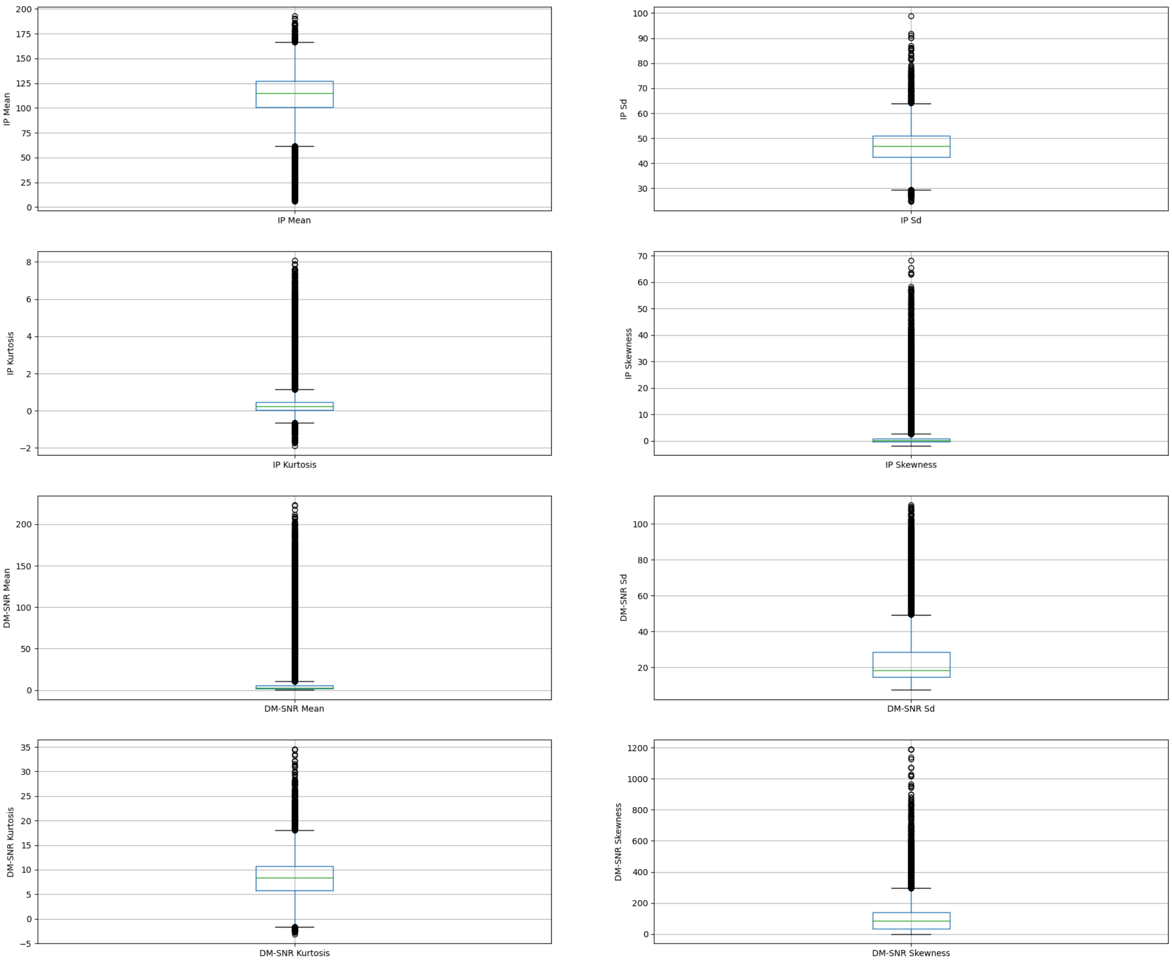
	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness
count	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00
mean	111.08	46.55	0.48	1.77	12.61	26.33	8.30	104.86
std	25.65	6.84	1.06	6.17	29.47	19.47	4.51	106.51
min	5.81	24.77	-1.88	-1.79	0.21	7.37	-3.14	-1.98
25%	100.93	42.38	0.03	-0.19	1.92	14.44	5.78	34.96
50%	115.08	46.95	0.22	0.20	2.80	18.46	8.43	83.06
75%	127.09	51.02	0.47	0.93	5.46	28.43	10.70	139.31
max	192.62	98.78	8.07	68.10	223.39	110.64	34.54	1191.00

In [16]:
```
# draw boxplots to visualize outliers

plt.figure(figsize=(24,20))


plt.subplot(4, 2, 1)
fig = df.boxplot(column='IP Mean')
fig.set_title('')
fig.set_ylabel('IP Mean')


plt.subplot(4, 2, 2)
fig = df.boxplot(column='IP Sd')
fig.set_title('')
fig.set_ylabel('IP Sd')


plt.subplot(4, 2, 3)
fig = df.boxplot(column='IP Kurtosis')
fig.set_title('')
fig.set_ylabel('IP Kurtosis')


plt.subplot(4, 2, 4)
fig = df.boxplot(column='IP Skewness')
```

```
fig.set_title('')
fig.set_ylabel('IP Skewness')


plt.subplot(4, 2, 5)
fig = df.boxplot(column='DM-SNR Mean')
fig.set_title('')
fig.set_ylabel('DM-SNR Mean')


plt.subplot(4, 2, 6)
fig = df.boxplot(column='DM-SNR Sd')
fig.set_title('')
fig.set_ylabel('DM-SNR Sd')


plt.subplot(4, 2, 7)
fig = df.boxplot(column='DM-SNR Kurtosis')
fig.set_title('')
fig.set_ylabel('DM-SNR Kurtosis')


plt.subplot(4, 2, 8)
fig = df.boxplot(column='DM-SNR Skewness')
fig.set_title('')
fig.set_ylabel('DM-SNR Skewness')
```

Out[16]: Text(0, 0.5, 'DM-SNR Skewness')

In [17]:
```
plt.show()
```



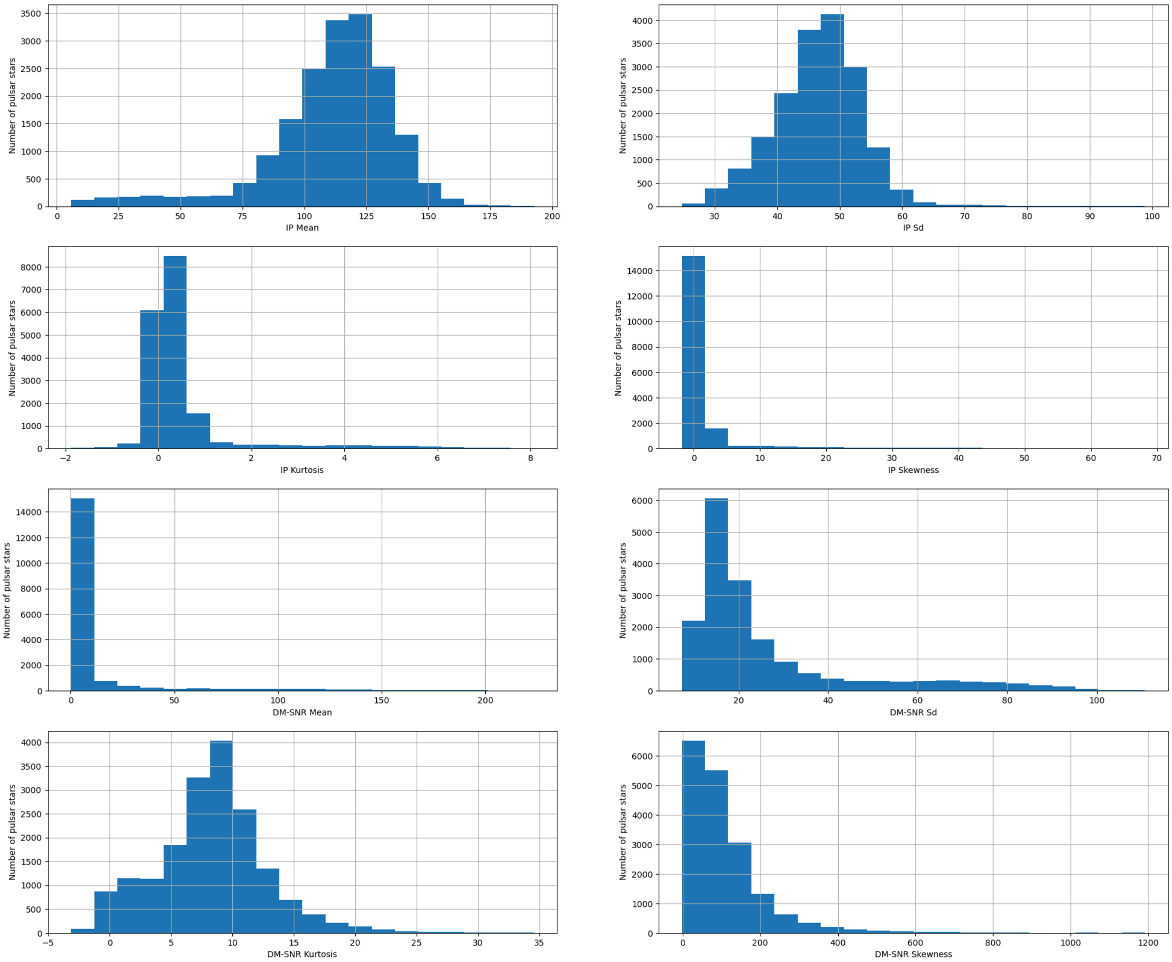
In [18]:

```
# plot histogram to check distribution


plt.figure(figsize=(24,20))


plt.subplot(4, 2, 1)
fig = df['IP Mean'].hist(bins=20)
fig.set_xlabel('IP Mean')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 2)
fig = df['IP Sd'].hist(bins=20)
fig.set_xlabel('IP Sd')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 3)
fig = df['IP Kurtosis'].hist(bins=20)
fig.set_xlabel('IP Kurtosis')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 4)
fig = df['IP Skewness'].hist(bins=20)
fig.set_xlabel('IP Skewness')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 5)
fig = df['DM-SNR Mean'].hist(bins=20)
fig.set_xlabel('DM-SNR Mean')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 6)
fig = df['DM-SNR Sd'].hist(bins=20)
fig.set_xlabel('DM-SNR Sd')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 7)
fig = df['DM-SNR Kurtosis'].hist(bins=20)
fig.set_xlabel('DM-SNR Kurtosis')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 8)
fig = df['DM-SNR Skewness'].hist(bins=20)
fig.set_xlabel('DM-SNR Skewness')
fig.set_ylabel('Number of pulsar stars')
```

Out[18]:

```
Text(0, 0.5, 'Number of pulsar stars')
```

In [19]:
```
plt.show()
```



Declare feature vector and target variable

In [20]:
```
X = df.drop(['target_class'], axis=1)
y = df['target_class']
```

Split data into separate training and test set

In [21]:
```
# 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 = 0.2, randc
```

In [22]:
```
# check the shape of X_train and X_test
X_train.shape, X_test.shape
```

Out[22]:
```
((14318, 8), (3580, 8))
```

Feature Scaling

In [23]:
```
cols = X_train.columns
```

In [24]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [25]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [26]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [27]:
```
X_train.describe()
```

Out[27]:

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-
count	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.4318
mean	1.908113e-16	-6.550610e-16	1.042143e-17	3.870815e-17	-8.734147e-17	-1.6
std	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000(
min	-4.035499e+00	-3.181033e+00	-2.185946e+00	-5.744051e-01	-4.239001e-01	-9.7
25%	-3.896291e-01	-6.069473e-01	-4.256221e-01	-3.188054e-01	-3.664918e-01	-6.1
50%	1.587461e-01	5.846646e-02	-2.453172e-01	-2.578142e-01	-3.372294e-01	-4.0
75%	6.267059e-01	6.501017e-01	-1.001238e-02	-1.419621e-01	-2.463724e-01	1.078
max	3.151882e+00	7.621116e+00	7.008906e+00	1.054430e+01	7.025568e+00	4.292

Run SVM with default hyperparameters

In [28]:
```
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score


# 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(accu
```

```
Model accuracy score with default hyperparameters: 0.9827
```

Run SVM with rbf kernel and C=100.0

In [29]:
```
# instantiate classifier with rbf kernel and C=100
svc=SVC(C=100.0)


# 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=100.0 : {0:0.4f}'. format(accu
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.9832
```

Run SVM with rbf kernel and C=1000.0

In [30]:
```
# 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(acc
```

```
Model accuracy score with rbf kernel and C=1000.0 : 0.9816
```

Run SVM with linear kernel and C=1.0

In [31]:
```
# instantiate classifier with linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0)


# 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=1.0 : {0:0.4f}'. format(acc
```

```
Model accuracy score with linear kernel and C=1.0 : 0.9830
```

Run SVM with linear kernel and C=100.0

In [32]:
```
# instantiate classifier with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)


# fit classifier to training set
linear_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'. format(a
```

```
Model accuracy score with linear kernel and C=100.0 : 0.9832
```

Run SVM with linear kernel and C=1000.0

In [33]:
```
# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.9832
```

Compare the train-set and test-set accuracy

In [34]:
```
y_pred_train = linear_svc.predict(X_train)
y_pred_train
```

Out[34]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [35]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_
```

```
Training-set accuracy score: 0.9783
```

Check for overfitting and underfitting

In [36]:
```
# print the scores on training and test set

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: 0.9783
Test set score: 0.9830
```

Compare model accuracy with null accuracy

In [37]:
```
# check class distribution in test set
y_test.value_counts()
```

Out[37]:
```
target_class
0    3306
1     274
Name: count, dtype: int64
```

In [38]:
```
# check null accuracy score

null_accuracy = (3306/(3306+274))
print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.9235
```

Run SVM with polynomial kernel and C=1.0

In [39]:
```
# instantiate classifier with polynomial kernel and C=1.0
poly_svc=SVC(kernel='poly', C=1.0)


# 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=1.0 : {0:0.4f}'. format
```

```
Model accuracy score with polynomial kernel and C=1.0 : 0.9807
```

Run SVM with polynomial kernel and C=100.0

In [40]:
```
# instantiate classifier with polynomial kernel and C=100.0
poly_svc100=SVC(kernel='poly', C=100.0)


# fit classifier to training set
poly_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=poly_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. format
```

```
Model accuracy score with polynomial kernel and C=1.0 : 0.9824
```

Run SVM with sigmoid kernel and C=1.0

In [41]:
```
# instantiate classifier with sigmoid kernel and C=1.0
sigmoid_svc=SVC(kernel='sigmoid', C=1.0)


# fit classifier to training set
```

```
sigmoid_svc.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=1.0 : {0:0.4f}'. format(ac
```

```
Model accuracy score with sigmoid kernel and C=1.0 : 0.8858
```

Run SVM with sigmoid kernel and C=100.0

In [42]:
```
# instantiate classifier with sigmoid kernel and C=100.0
sigmoid_svc100=SVC(kernel='sigmoid', C=100.0)


# fit classifier to training set
sigmoid_svc100.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=100.0 : {0:0.4f}'. format(
```

```
Model accuracy score with sigmoid kernel and C=100.0 : 0.8855
```

Confusion matrix

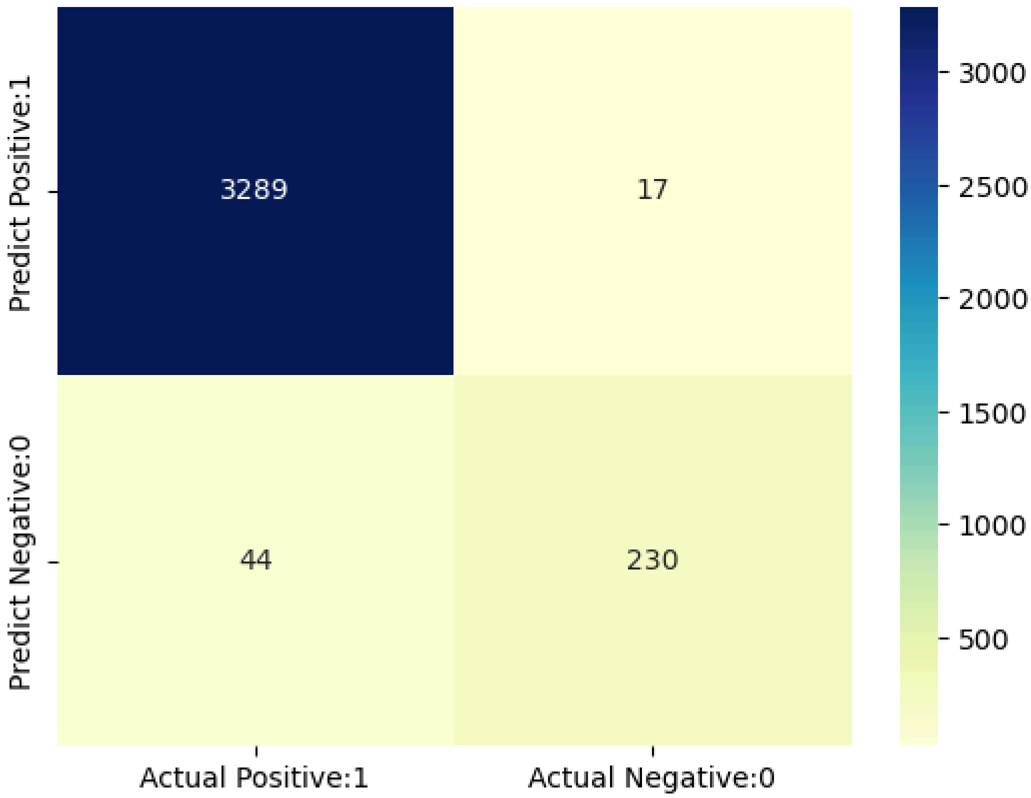
In [43]:
```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[3289   17]
 [  44  230]]

True Positives(TP) =  3289

True Negatives(TN) =  230

False Positives(FP) =  17

False Negatives(FN) =  44
```

In [44]:
```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                 index=['Predict Positive:1', 'Predict Negative:

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

Out[44]: <Axes: >

In [45]:
```
plt.show()
```



Classification Report

In [46]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.93      0.84      0.88       274

    accuracy                           0.98      3580
   macro avg       0.96      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

Classification accuracy

In [47]:
```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [48]:
```
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9830
```

Classification error

In [49]:
```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0170
```

Precision

In [50]:
```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9949
```

Recall

In [51]:
```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9868
```

True Positive Rate

In [52]:
```
true_positive_rate = TP / float(TP + FN)

print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.9868
```

False Positive Rate

In [53]:
```
false_positive_rate = FP / float(FP + TN)

print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.0688
```

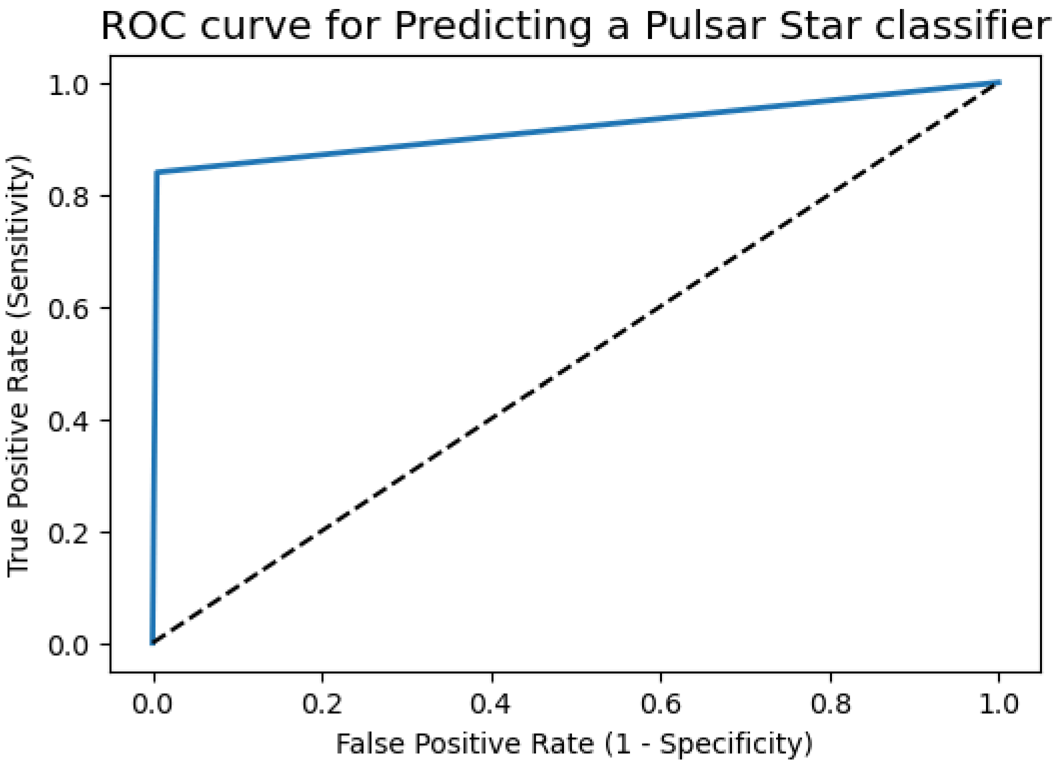
Specificity

In [54]:
```
specificity = TN / (TN + FP)
print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.9312
```

ROC - AUC

In [55]:
```
# 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()
```



In [56]:
```
# 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 : 0.9171
```

In [57]:
```
# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(linear_svc, X_train, y_train, cv=10, s

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9756
```

Stratified k-Fold Cross Validation with shuffle split with linear kernel

In [58]:
```
from sklearn.model_selection import KFold


kfold=KFold(n_splits=5, shuffle=True, random_state=0)


linear_svc=SVC(kernel='linear')


linear_scores = cross_val_score(linear_svc, X, y, cv=kfold)
```

In [59]:
```
# print cross-validation scores with linear kernel
```

```
print('Stratified cross-validation scores with linear kernel:\n\n{}'.format(line
```

```
Stratified cross-validation scores with linear kernel:

[0.98296089 0.97458101 0.97988827 0.97876502 0.97848561]
```

In [60]:

```
# print average cross-validation score with linear kernel

print('Average stratified cross-validation score with linear kernel:{:.4f}'.form
```

```
Average stratified cross-validation score with linear kernel:0.9789
```

Stratified k-Fold Cross Validation with shuffle split with rbf kernel

In [61]:

```
rbf_svc=SVC(kernel='rbf')

rbf_scores = cross_val_score(rbf_svc, X, y, cv=kfold)
```

In [62]:

```
# print cross-validation scores with rbf kernel

print('Stratified Cross-validation scores with rbf kernel:\n\n{}'.format(rbf_scc
```

```
Stratified Cross-validation scores with rbf kernel:

[0.97849162 0.97011173 0.97318436 0.9709416  0.96982397]
```

In [63]:

```
# print average cross-validation score with rbf kernel

print('Average stratified cross-validation score with rbf kernel:{:.4f}'.format(
```

```
Average stratified cross-validation score with rbf kernel:0.9725
```

Hyperparameter Optimization using GridSearch CV

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# 1. Prepare your data
# Assume df is already defined and cleaned
X = df.drop('target_class', axis=1)   # Features
y = df['target_class']                # Target

# 2. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Instantiate base SVC model
svc = SVC()

# 4. Declare parameter grid
parameters = [
    {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
    {'C': [1, 10, 100, 1000], 'kernel': ['rbf'],
     'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
    {'C': [1, 10, 100, 1000], 'kernel': ['poly'],
     'degree': [2, 3, 4], 'gamma': [0.01, 0.02, 0.03, 0.04, 0.05]}
]
```

```python
# 5. GridSearchCV
grid_search = GridSearchCV(
    estimator=svc,
    param_grid=parameters,
    scoring='accuracy',
    cv=5,
    verbose=0
)

# 6. Fit to training data
grid_search.fit(X_train, y_train)

# 7. Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

# 8. Evaluate on test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
```

In [ ]:

```python
# examine the best model


# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_
```

In [ ]:

```python
# calculate GridSearch CV score on test set

print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_tes
```

In [ ]:

```python

```