In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

7. Import dataset

In [8]:
```
df = pd.read_csv(r"C:\Users\santo\OneDrive\Desktop\Data science\Jan 2026\2nd jan SVM\SVM\pulsar_stars.csv")
```

In [9]:
```
df.head()
```

Out[9]:

	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
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

In [10]:
```
df.shape
```

Out[10]: (17898, 9)

In [11]:
```
df.head()
```

Out[11]:

	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
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

In [12]:
```
col_names = df.columns
col_names
```

Out[12]:
```
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 [13]:
```
df.columns = df.columns.str.strip()
```

In [14]:
```
df.columns
```

Out[14]:
```
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 [15]:
```
df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',
              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class']
```

In [16]:
```
df.columns
```

Out[16]:
```
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 [17]:
```
df['target_class'].value_counts()
```

Out[17]:
```
target_class
0    16259
1     1639
Name: count, dtype: int64
```

In [20]:
```
df['target_class'].value_counts() / float(len(df))
```

Out[20]:
```
target_class
0    0.908426
1    0.091574
Name: count, dtype: float64
```

In [21]:
```
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 [22]:
```
df.isnull().sum()
```

Out[22]:
```
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 [23]:
```
round(df.describe(),2)
```

Out[23]:

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness	target_class
count	17898.00	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	0.09
std	25.65	6.84	1.06	6.17	29.47	19.47	4.51	106.51	0.29
min	5.81	24.77	-1.88	-1.79	0.21	7.37	-3.14	-1.98	0.00
25%	100.93	42.38	0.03	-0.19	1.92	14.44	5.78	34.96	0.00
50%	115.08	46.95	0.22	0.20	2.80	18.46	8.43	83.06	0.00
75%	127.09	51.02	0.47	0.93	5.46	28.43	10.70	139.31	0.00
max	192.62	98.78	8.07	68.10	223.39	110.64	34.54	1191.00	1.00

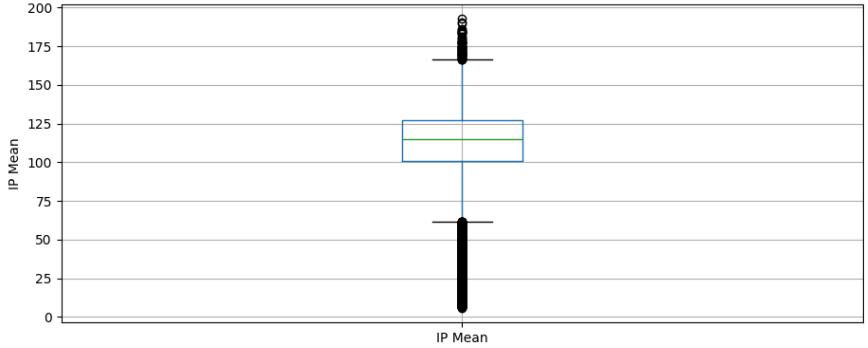
In [24]:

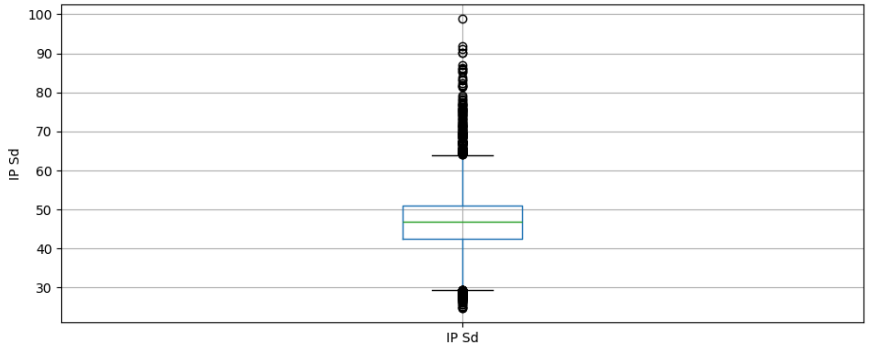
```
# draw boxplots to visualize outliers

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')
```

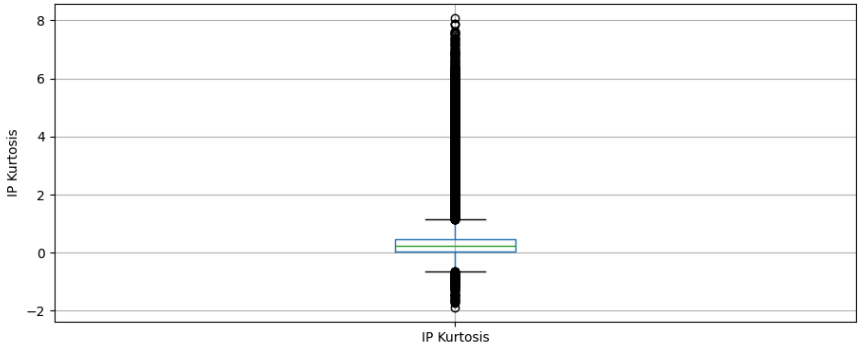
```
plt.subplot(4, 2, 4)
fig = df.boxplot(column='IP Skewness')
fig.set_title('')
fig.set_ylabel('IP Skewness')


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')
```

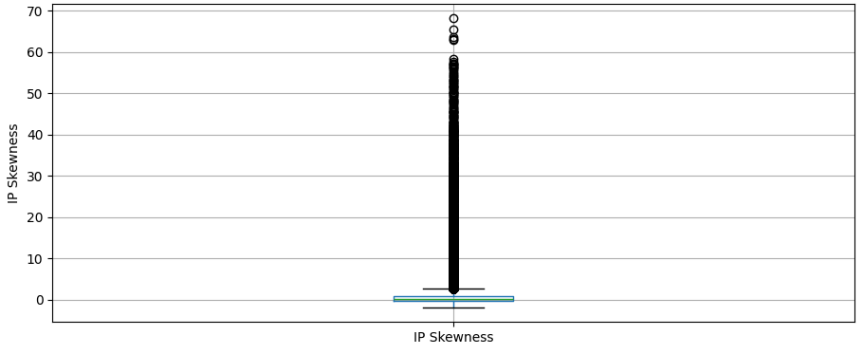
Out[24]:
```
Text(0, 0.5, 'DM-SNR Skewness')
```

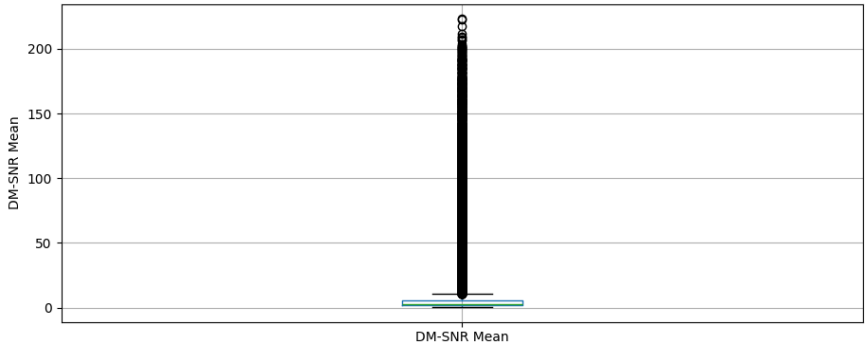
In [25]:
```
plt.show()
```

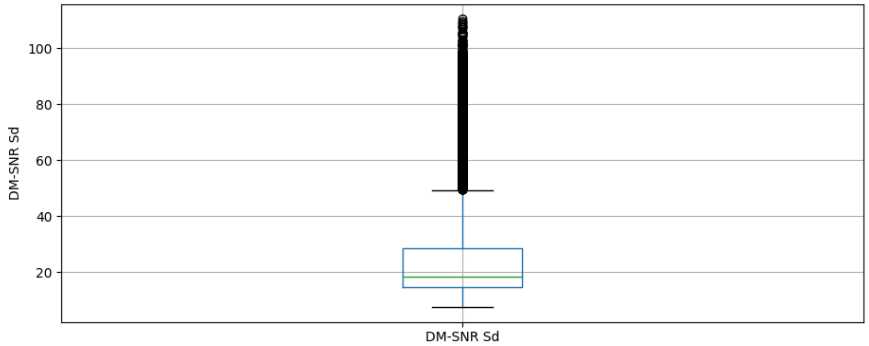

IP Mean
IP Mean

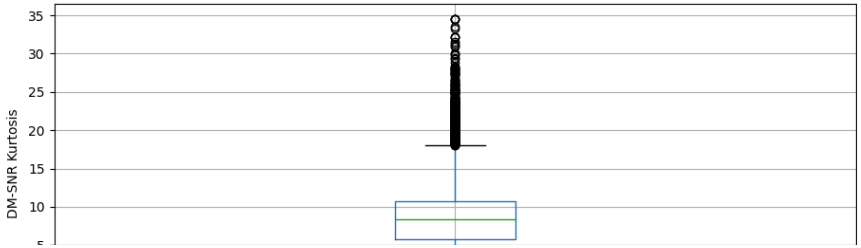

IP Sd
IP Sd

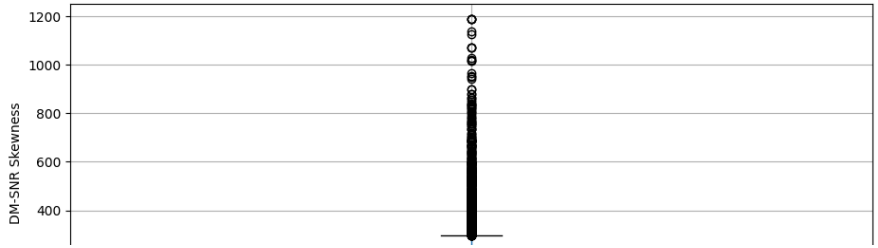

IP Kurtosis
IP Kurtosis

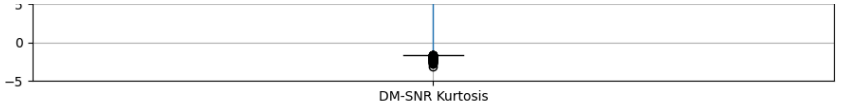

IP Skewness
IP Skewness

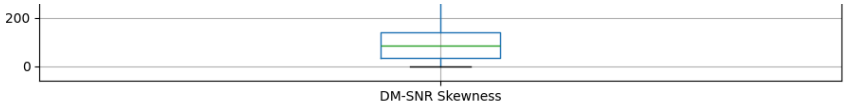

DM-SNR Mean
DM-SNR Mean


DM-SNR Sd
DM-SNR Sd


DM-SNR Kurtosis


DM-SNR Skewness



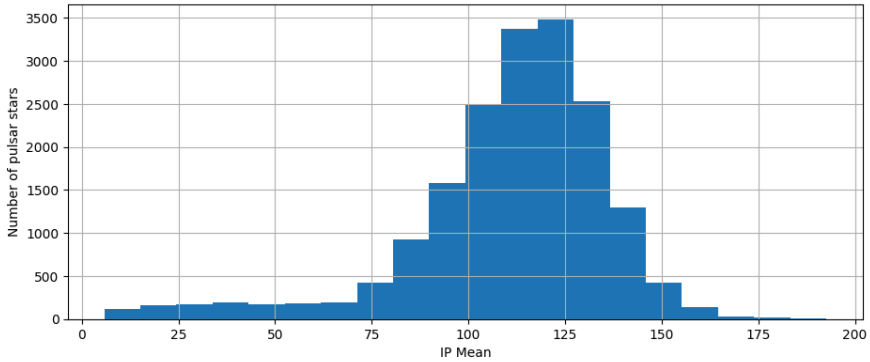


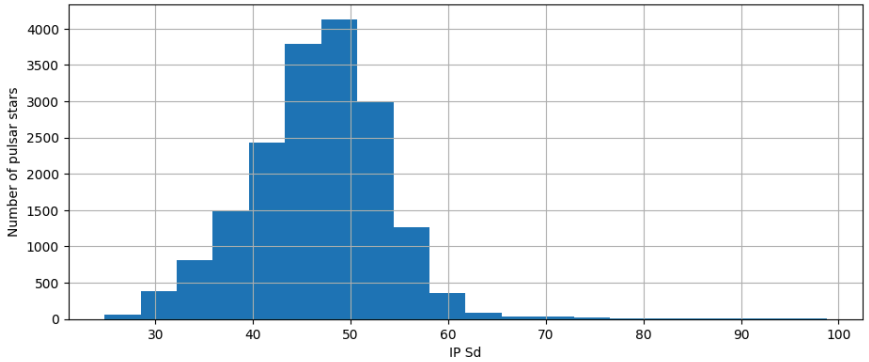
In [26]:
```
# plot histogram to check distribution


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')
```

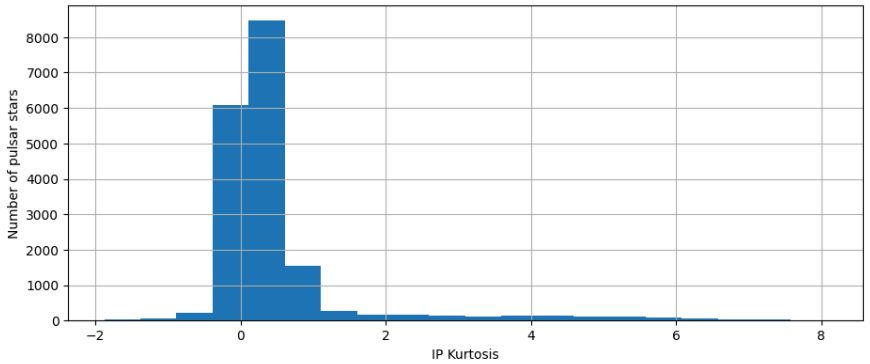
```
plt.subplot(4, 2, 6)
fig = df['DM-SNR Sd'].hist(bins=20)
fig.set_xlabel('DM-SNR Sd')
fig.set_ylabel('Number of pulsar stars')



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')
```

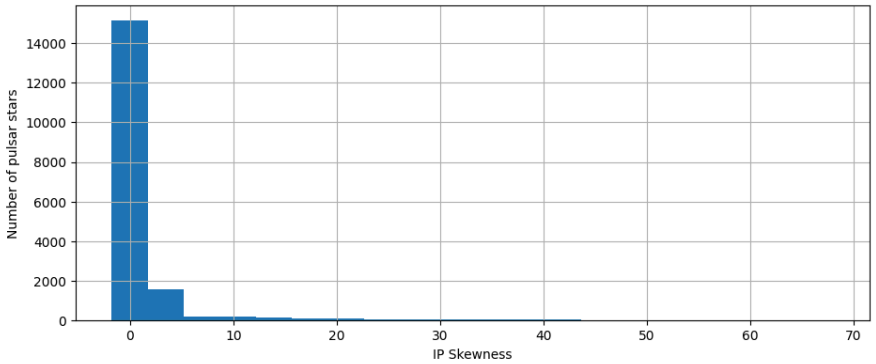
Out[26]:
```
Text(0, 0.5, 'Number of pulsar stars')
```

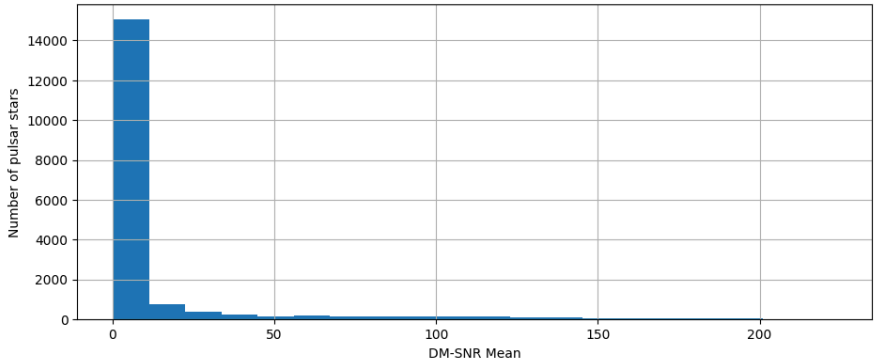
In [27]:
```
plt.show()
```

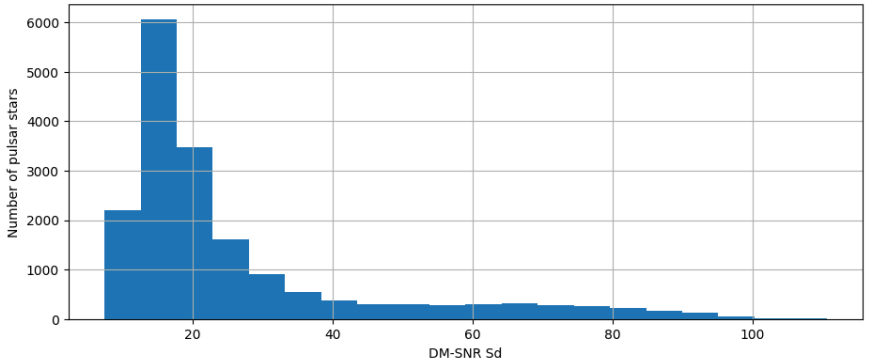

Number of pulsar stars
IP Mean

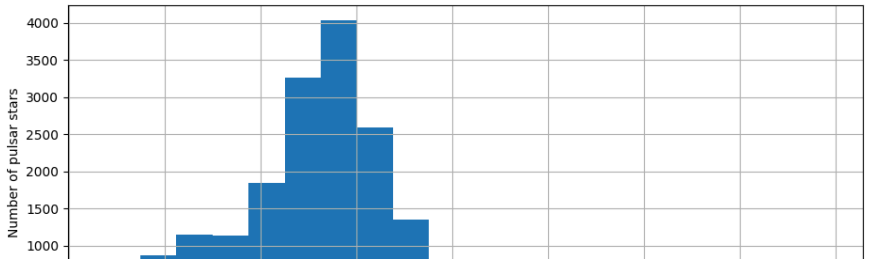

Number of pulsar stars
IP Sd

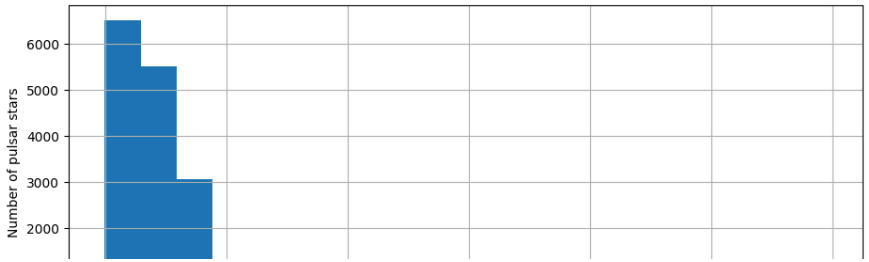

Number of pulsar stars
IP Kurtosis

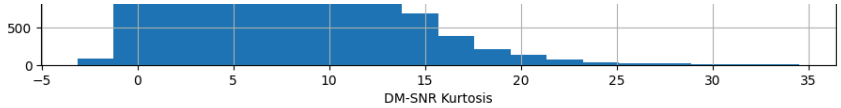

Number of pulsar stars
IP Skewness

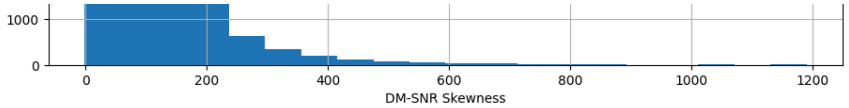

Number of pulsar stars
DM-SNR Mean


Number of pulsar stars
DM-SNR Sd


Number of pulsar stars


Number of pulsar stars





In [28]:
```
X = df.drop(['target_class'], axis=1)

y = df['target_class']
```

In [29]:
```
# 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, random_state = 0)
```

In [30]:
```
X_train.shape, X_test.shape
```

Out[30]: ((14318, 8), (3580, 8))

In [31]:
```
cols = X_train.columns
```

In [32]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [33]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [34]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [35]:
```
X_train.describe()
```

Out[35]:

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness
count	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04
mean	1.908113e-16	-6.550610e-16	1.042143e-17	3.870815e-17	-8.734147e-17	-1.617802e-16	-1.513588e-17	1.122785e-16
std	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00
min	-4.035499e+00	-3.181033e+00	-2.185946e+00	-5.744051e-01	-4.239001e-01	-9.733707e-01	-2.455649e+00	-1.003411e+00
25%	-3.896291e-01	-6.069473e-01	-4.256221e-01	-3.188054e-01	-3.664918e-01	-6.125457e-01	-5.641035e-01	-6.627590e-01
50%	1.587461e-01	5.846646e-02	-2.453172e-01	-2.578142e-01	-3.372294e-01	-4.067482e-01	3.170446e-02	-2.059136e-01
75%	6.267059e-01	6.501017e-01	-1.001238e-02	-1.419621e-01	-2.463724e-01	1.078934e-01	5.362759e-01	3.256217e-01
max	3.151882e+00	7.621116e+00	7.008906e+00	1.054430e+01	7.025568e+00	4.292181e+00	5.818557e+00	1.024613e+01

12. Run SVM with default hyperparameters

In [36]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc=SVC()

svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)

print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

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

In [37]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [38]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [39]:
```
# 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(accuracy_score(y_test, y_pred_test)))
```

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

In [40]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [41]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [42]:
```
y_pred_train = linear_svc.predict(X_train)

y_pred_train
```

Out[42]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

In [43]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

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

In [44]:
```
# 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
```

In [45]:
```
y_test.value_counts()
```

Out[45]:
```
target_class
0    3306
1     274
Name: count, dtype: int64
```

In [46]:
```
# check null accuracy score

null_accuracy = (3306/(3306+274))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.9235
```

In [47]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [48]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [49]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

In [50]:
```
# 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(accuracy_score(y_test, y_pred)))
```

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

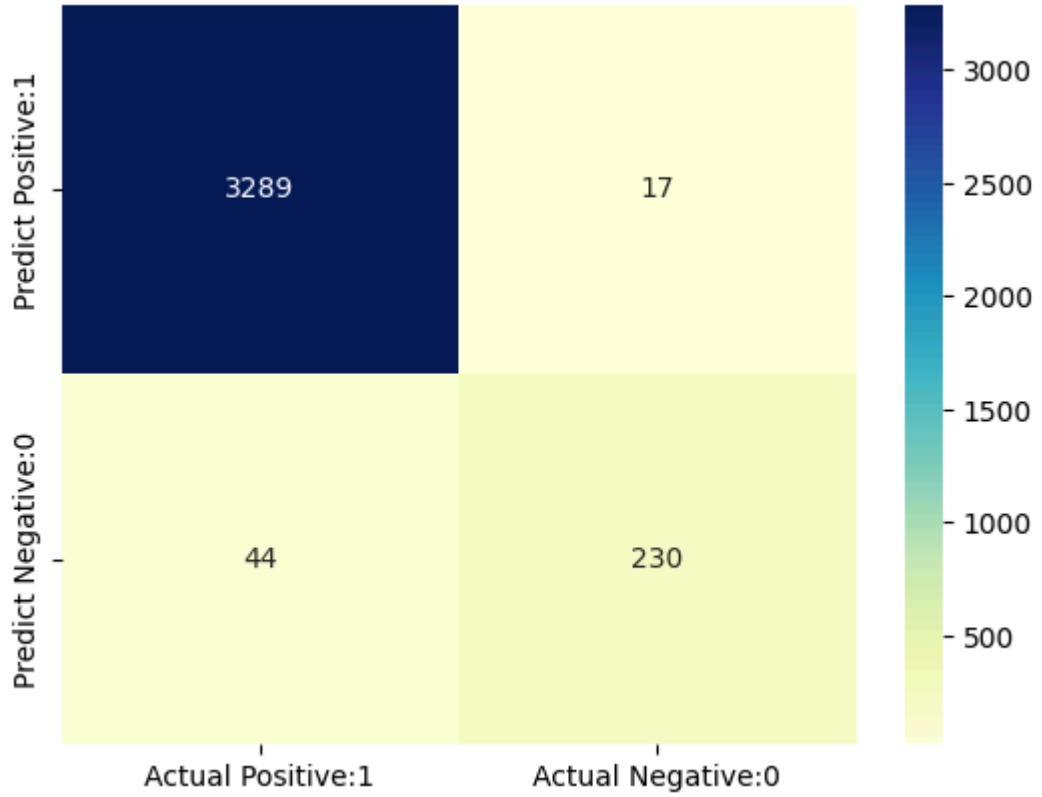
In [51]:
```
# Print the Confusion Matrix and slice it into four pieces

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 [52]:
```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

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

Out[52]: <Axes: >

In [53]:
```
plt.show()
```



In [54]:
```
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 [55]:
```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

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

In [57]:
```
# 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
```

In [58]:
```
# print precision score

precision = TP / float(TP + FP)


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

```
Precision : 0.9949
```

In [59]:
```
recall = TP / float(TP + FN)

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

```
Recall or Sensitivity : 0.9868
```

True positive Rate

In [60]:
```
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 [61]:
```
false_positive_rate = FP / float(FP + TN)


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

```
False Positive Rate : 0.0688
```

specificity

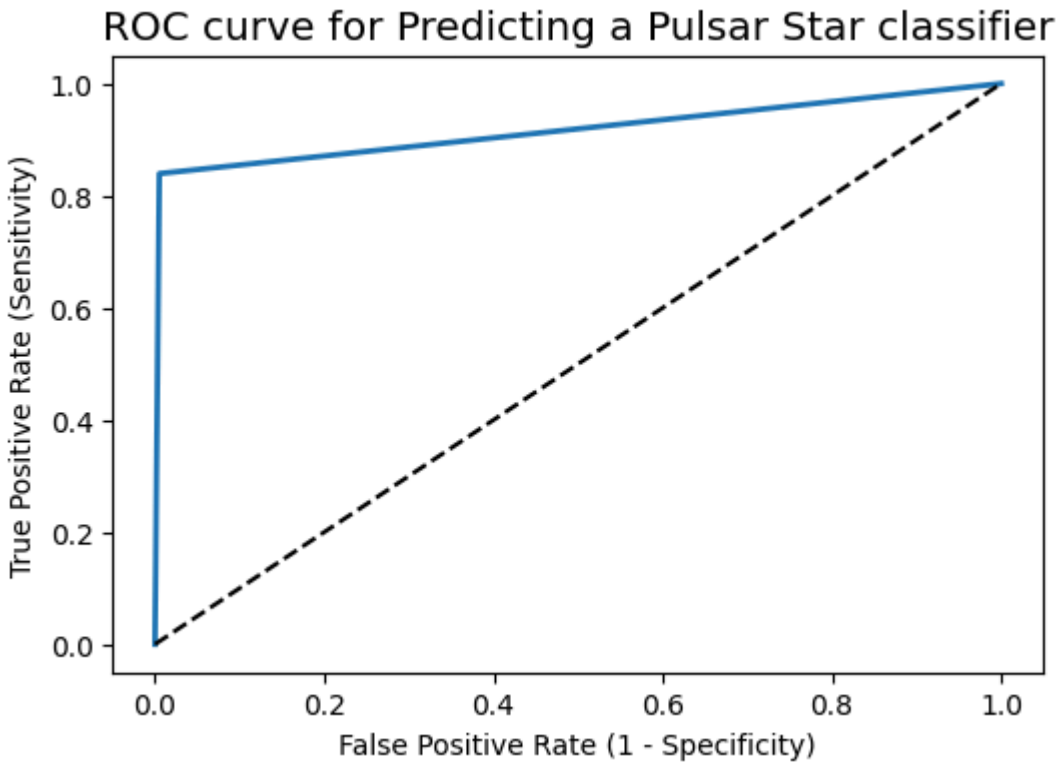
In [62]:
```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.9312
```

18. ROC-AUC

In [63]:
```
# 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 [64]:
```
# 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 [65]:
```
# 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, scoring='roc_auc').mean()

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

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

19. Stratified k-fold Cross Validation with shuffle split

In [66]:
```
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 [67]:
```
# print cross-validation scores with linear kernel

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

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

[0.98296089 0.97458101 0.97988827 0.97876502 0.97848561]
```

In [68]:
```
# print average cross-validation score with linear kernel

print('Average stratified cross-validation score with linear kernel:{:.4f}'.format(linear_scores.mean()))
```

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

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

In [69]:
```
rbf_svc=SVC(kernel='rbf')


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

In [70]:
```
# print cross-validation scores with rbf kernel

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

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

[0.97849162 0.97011173 0.97318436 0.9709416  0.96982397]
```

In [71]:
```
# print average cross-validation score with rbf kernel

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

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

20. Hyperparameter Optimization using GridSearch CV

Table of Contents

In [ ]:
```
# import GridSearchCV
from sklearn.model_selection import GridSearchCV


```

```
# import SVC classifier
from sklearn.svm import SVC


# instantiate classifier with default hyperparameters with kernel=rbf, C=1.0 and gamma=auto
svc=SVC()



# declare parameters for hyperparameter tuning
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]}
              ]




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


grid_search.fit(X_train, y_train)
```

In [ ]:

```
# 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_estimator_))
```

In [ ]:
```
# calculate GridSearch CV score on test set

print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

The Complated

In [ ]:
```
# 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_estimator_))
```

In [ ]:
```

```