pulsar6

November 4, 2022

1 Pulsar Emission Data Analysis

2 All Imports that may or may not be needed and used for the notebook

[ ]:
```
#currently including any and all Imports that maybe needed for the project.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.feature_selection import RFE
import datetime as dt
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree
from scipy.spatial.distance import pdist
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.dates as mdates
from scipy.stats import pearsonr
from scipy import stats
import statistics
import math
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.tsatools import lagmat
from numpy import array
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Bidirectional
```

```python
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics
from scipy.stats import binned_statistic
```

3 Section for extracting from a tar file.

Currently implemented for original TAR File structure.

[ ]:
```python
#This is also found in the main file under tarunzip.py
import tarfile
import os
import sys

#tar = tarfile.open("pulseTarFile.tar")
#tar.extractall('./Data')
#tar.close()
```

[ ]:
```python
#How to remove outlier data for these datasets.
#pulsar6npcleaned = pulsar6[(np.abs(stats.zscore(pulsar6["Brightness"])) <3)]
#pulsar6npcleaned
```

3.1 Beginning of Exploration

3.1.1 Examining the data

In this section we are determining the total integrity of the data to determine if further comprehensive data cleaning and uniforming processes are needed.

[ ]:
```python
colnames = ['Pulse Number', 'Brightness', 'Uncertainty']
pulsar = pd.read_csv("Data/J1644-4559.pulses", sep = ' ', header = None, names␣
 ↪= colnames)
```

[ ]:
```python
pulsar.shape
```

[ ]:
```
(698, 3)
```

[ ]:
```python
pulsar.head(25)
```

[ ]:

	Pulse Number	Brightness	Uncertainty
0	1	0.634671	0.002761
1	2	0.736945	0.005207
2	3	0.693834	0.002706
3	4	1.021866	0.010184
4	5	0.673845	0.006236
5	6	0.676883	0.004763
6	7	0.527039	0.002422
7	8	0.673417	0.003174
8	9	0.357076	0.002848
9	10	0.661704	0.005588

```
10             11    0.545564     0.003835
11             12    0.494655     0.003145
12             13    0.804260     0.005258
13             14    0.513362     0.005700
14             15    0.477025     0.002945
15             16    0.399571     0.004712
16             17    0.188069     0.002452
17             18    0.748592     0.005468
18             19    0.723437     0.004548
19             20    0.960154     0.006765
20             21    0.707715     0.006011
21             22    1.074550     0.006831
22             23    0.961340     0.006617
23             24    0.754457     0.004117
24             25    0.773151     0.004920
```

[ ]:
```
pulsar.describe()
```

[ ]:
```
       Pulse Number  Brightness  Uncertainty
count     698.00000  698.000000   698.000000
mean      349.50000    0.654319     0.004445
std       201.63953    0.163945     0.001855
min         1.00000    0.007642     0.002129
25%       175.25000    0.555267     0.003086
50%       349.50000    0.658295     0.003951
75%       523.75000    0.753396     0.005349
max       698.00000    1.159334     0.016097
```

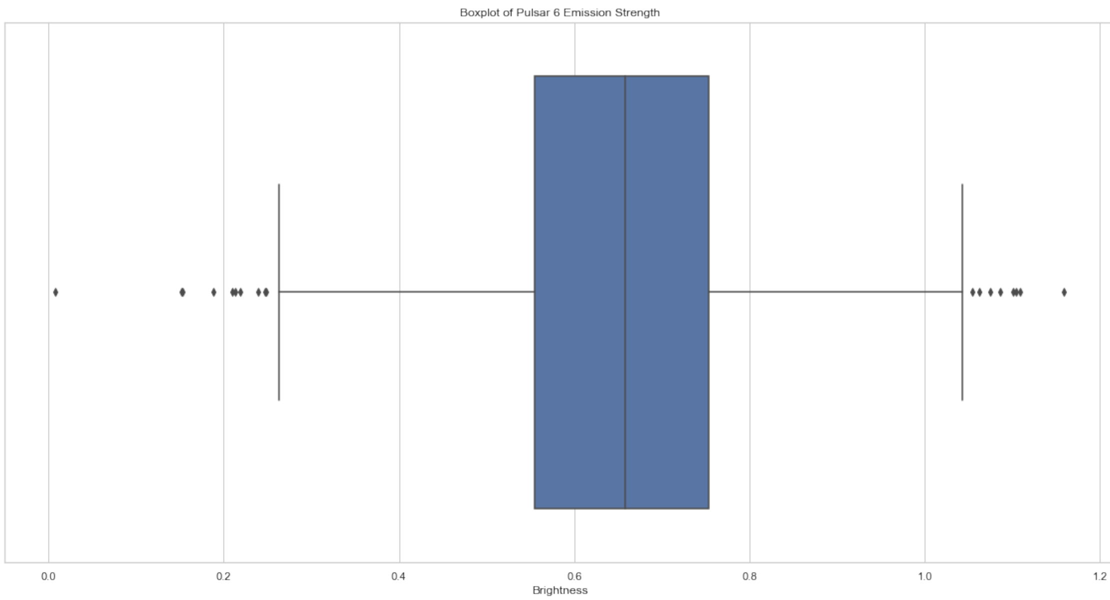
[ ]:
```
nullBoolBrightness = pd.isnull(pulsar["Brightness"])

pulsar[nullBoolBrightness]
```

[ ]:
```
Empty DataFrame
Columns: [Pulse Number, Brightness, Uncertainty]
Index: []
```

[ ]:
```
pulsar["Brightness"].describe()
```

[ ]:
```
count    698.000000
mean       0.654319
std        0.163945
min        0.007642
25%        0.555267
50%        0.658295
75%        0.753396
max        1.159334
Name: Brightness, dtype: float64
```

[ ]:
```python
plt.figure(figsize=(20,10))
sns.set_theme(style="whitegrid")
ax = sns.boxplot(x=pulsar["Brightness"]).set_title("Boxplot of Pulsar 6␣
↪Emission Strength")
```



[ ]:
```python
medianpulse6 = pulsar["Brightness"].median()
print("Median of Pulsar6: ", medianpulse6)
pulsar['Binary'] = np.where(pulsar['Brightness'] > medianpulse6, 1, 0)
```

```
Median of Pulsar6:  0.65829515
```

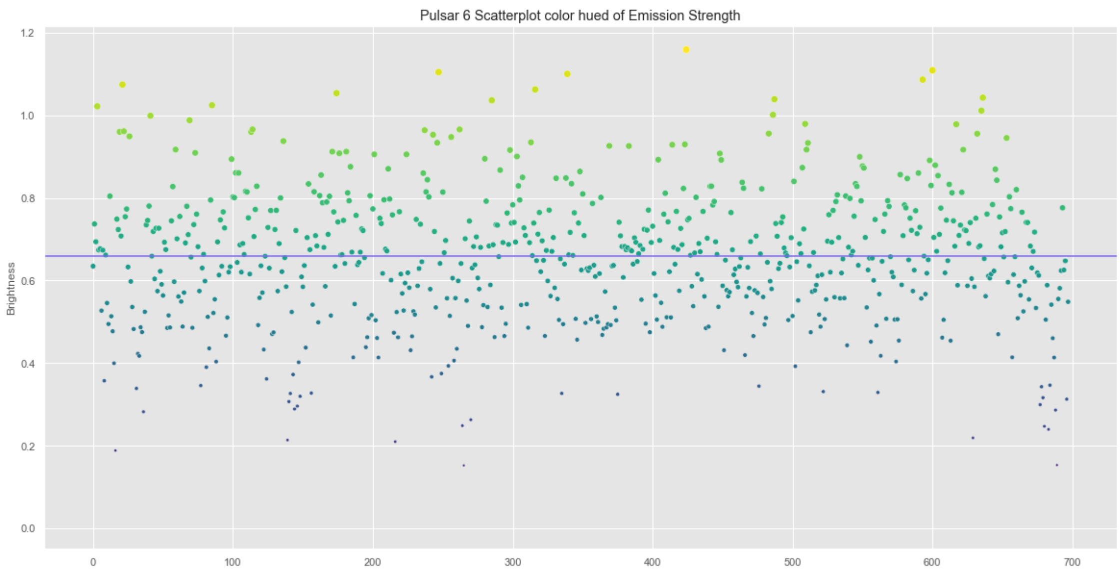
[ ]:
```python
pulsar
```

[ ]:

	Pulse Number	Brightness	Uncertainty	Binary
0	1	0.634671	0.002761	0
1	2	0.736945	0.005207	1
2	3	0.693834	0.002706	1
3	4	1.021866	0.010184	1
4	5	0.673845	0.006236	1
..	…	…	…	…
693	694	0.776083	0.008928	1
694	695	0.625382	0.006018	0
695	696	0.647559	0.003765	0
696	697	0.312449	0.002901	0
697	698	0.548353	0.009056	0

[698 rows x 4 columns]

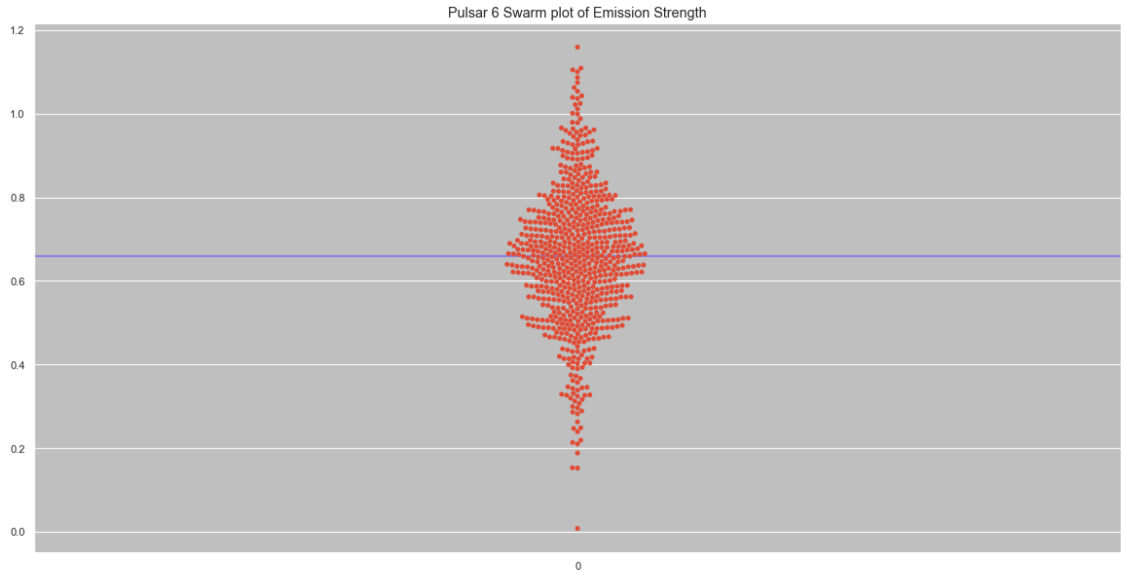
```
[ ]: plt.figure(figsize=(20,10))
     sns.set_style("darkgrid", {"axes.facecolor": ".75"})
     strength = pulsar.Brightness.values
     plt.style.use('ggplot')
     ax = sns.scatterplot(data=pulsar["Brightness"], s= strength*50, c=strength,␣
      ↪cmap="viridis", marker="o").set_title('Pulsar 6 Scatterplot color hued of␣
      ↪Emission Strength')
     ax= plt.axhline( y=0.65829515, ls='-',c='mediumslateblue')
```




```
[ ]: print(len(pulsar[(pulsar.Brightness > 0.6589028)]))
     print(len(pulsar[(pulsar.Brightness < 0.6589028)]))
```

348
350

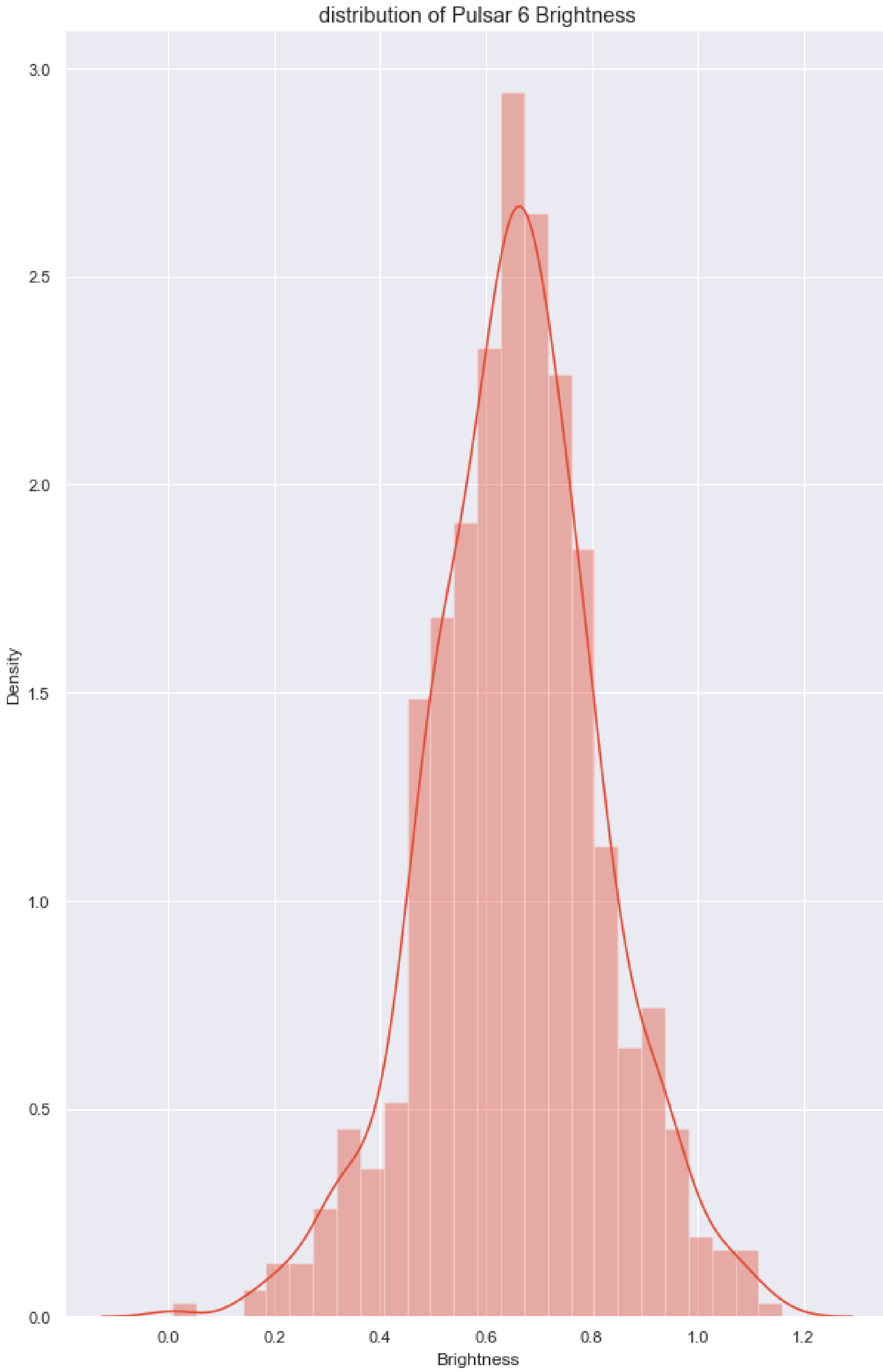
```
[ ]: plt.figure(figsize=(20,10))
     sns.set_style("darkgrid", {"axes.facecolor": ".75"})
     strength = pulsar.Brightness.values
     ax = plt.axhline( y=0.65829515, ls='-',c='mediumslateblue')
     ax = sns.swarmplot(data=pulsar["Brightness"], c="blue").set_title('Pulsar 6␣
      ↪Swarm plot of Emission Strength')
```



[ ]:
```
plt.figure(figsize=(10, 16))
with sns.axes_style('darkgrid'):
    sns.distplot(pulsar.Brightness)
plt.title("distribution of Pulsar 6 Brightness")
```

```
C:\Users\tajki\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

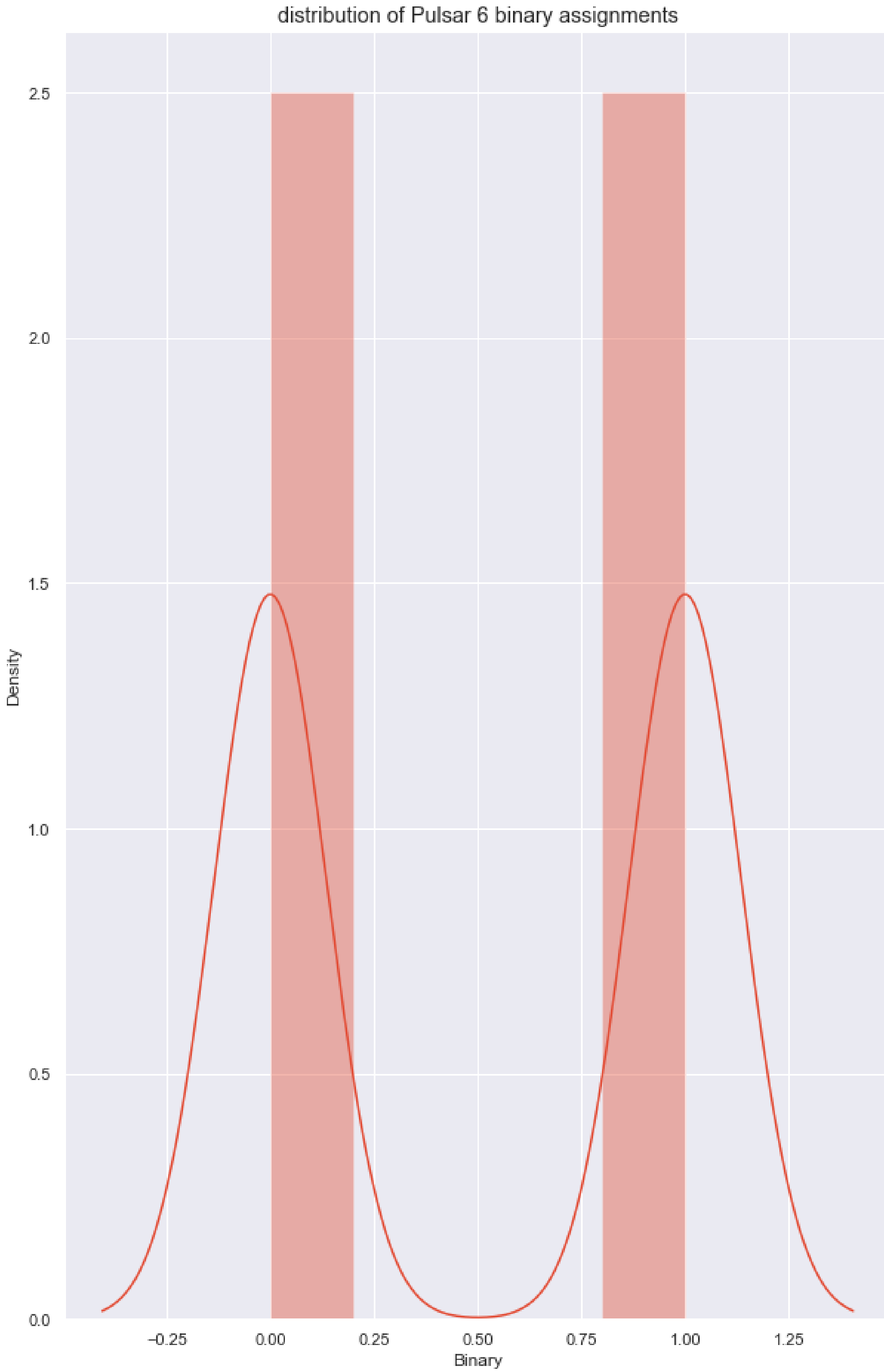
[ ]:
```
Text(0.5, 1.0, 'distribution of Pulsar 6 Brightness')
```


distribution of Pulsar 6 Brightness
Density
Brightness
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2

[ ]:
```
plt.figure(figsize=(10, 16))
with sns.axes_style('darkgrid'):
    sns.distplot(pulsar.Binary)
plt.title("distribution of Pulsar 6 binary assignments")
```

```
C:\Users\tajki\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[ ]:
```
Text(0.5, 1.0, 'distribution of Pulsar 6 binary assignments')
```

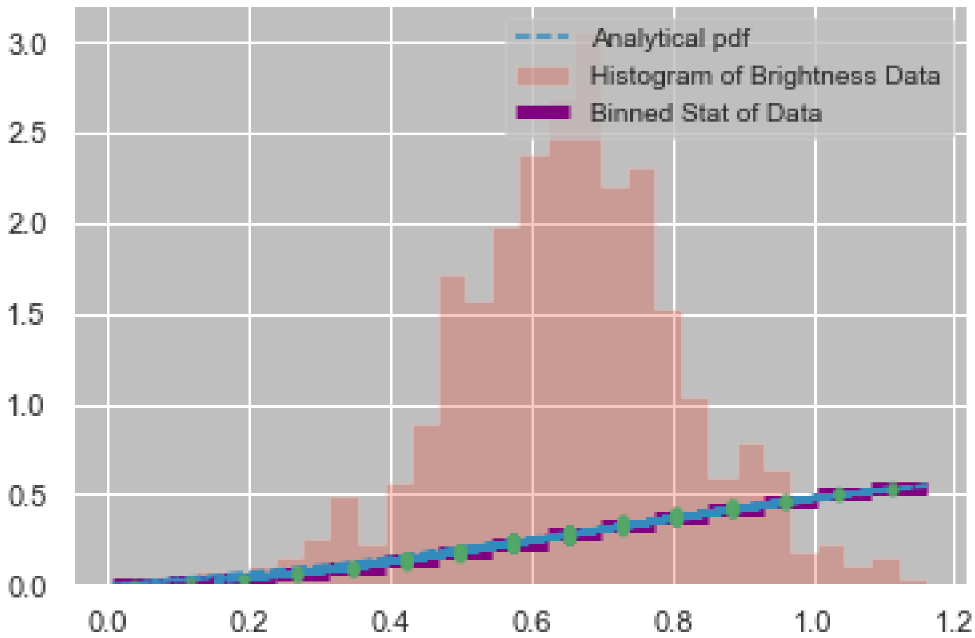

distribution of Pulsar 6 binary assignments
Density
Binary
2.5
2.0
1.5
1.0
0.5
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25

4 Rolling Medians, Rolling Means, Binned Medians and Binned Mean analysis.

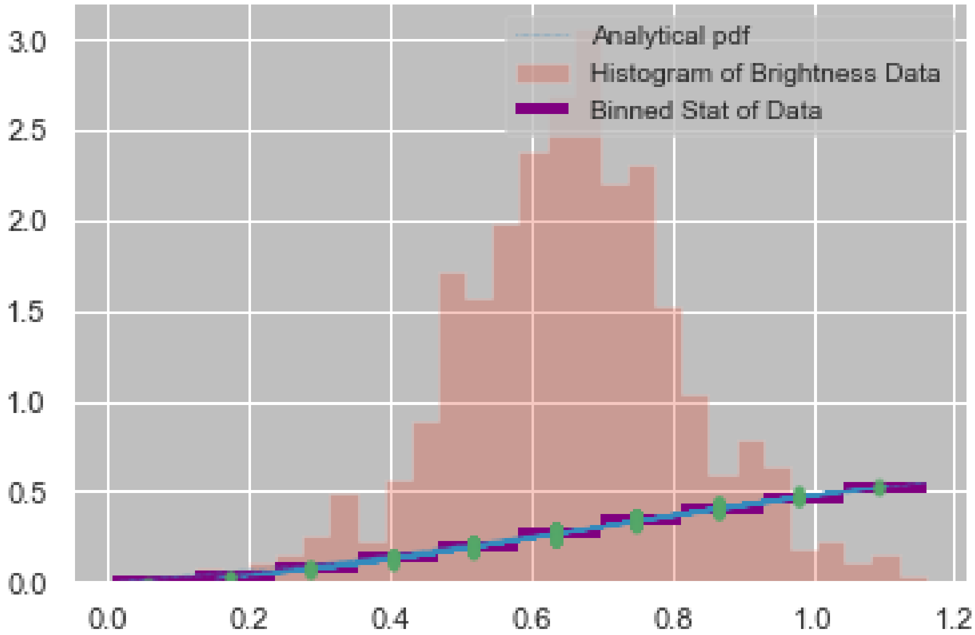
[ ]:
```
data = pulsar["Brightness"]
data
```

[ ]:
```
0      0.634671
1      0.736945
2      0.693834
3      1.021866
4      0.673845
         …
693    0.776083
694    0.625382
695    0.647559
696    0.312449
697    0.548353
Name: Brightness, Length: 698, dtype: float64
```

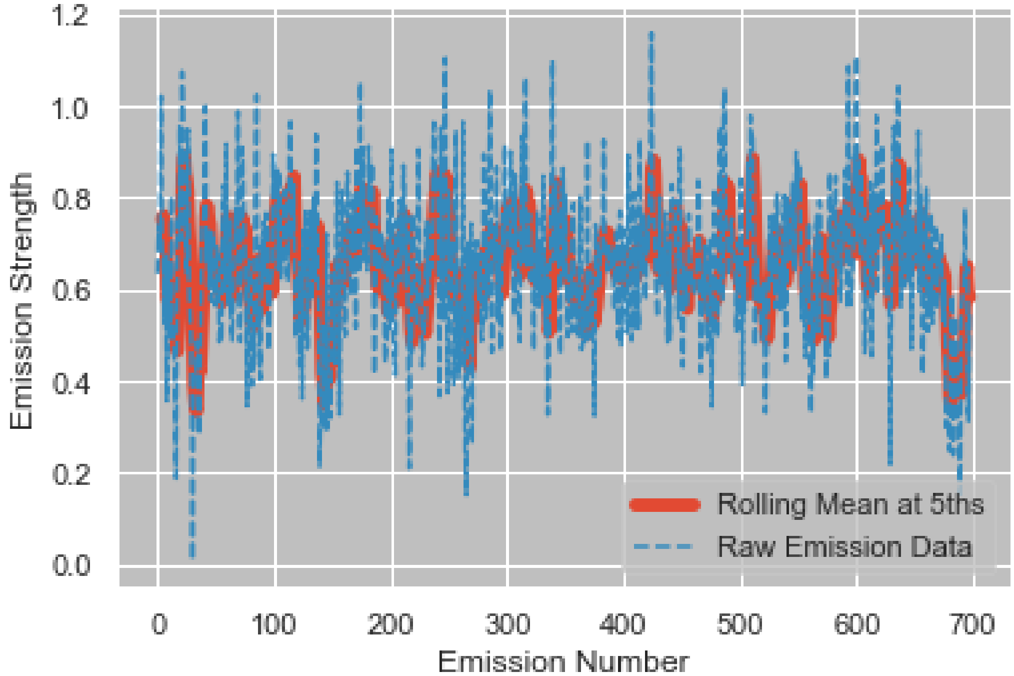
[ ]:
```
dataPDF = stats.maxwell.pdf(data)
bin_means, bin_edges, binnumber = stats.binned_statistic(data, dataPDF,
        statistic='mean', bins=15)
bin_width = (bin_edges[1] - bin_edges[0])
bin_centers = bin_edges[1:] - bin_width/2

plt.figure()
plt.hist(data, bins=30, density=True, histtype='stepfilled', alpha=0.3,␣
 ↪label='Histogram of Brightness Data')
plt.plot(data, dataPDF, '--', label = "Analytical pdf")
plt.hlines(bin_means, bin_edges[:-1], bin_edges[1:], colors='purple', lw=5,␣
 ↪label='Binned Stat of Data')
plt.plot((binnumber - 0.5) * bin_width, dataPDF, 'g.', alpha=0.5)
plt.legend(fontsize=10)
plt.show()
```



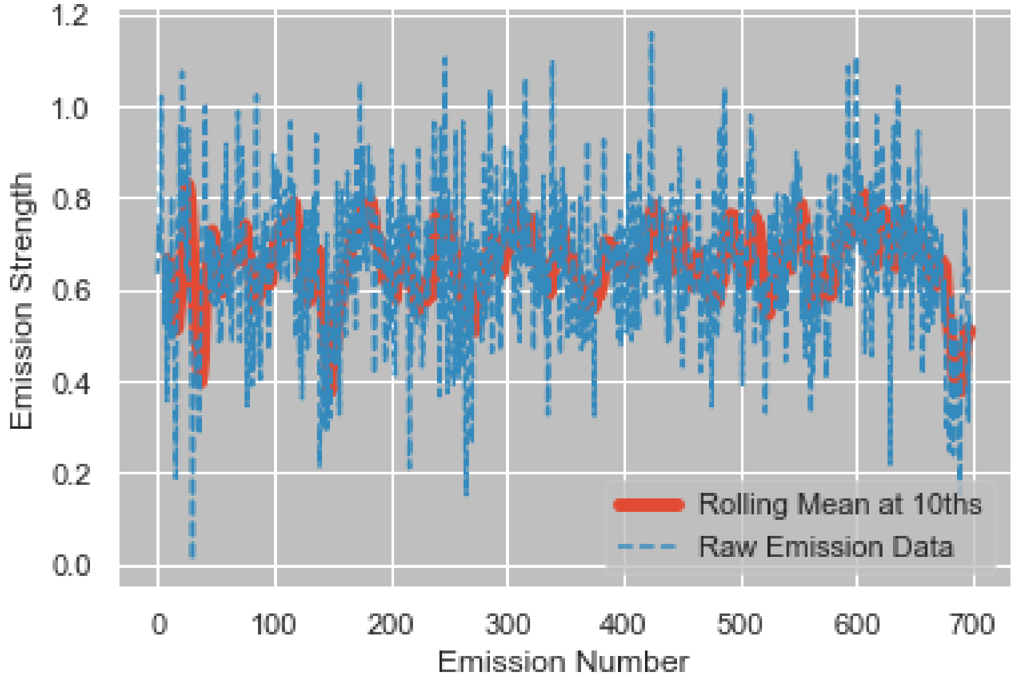
[ ]:
```
bin_median, bin_edges, binnumber = stats.binned_statistic(data, dataPDF,
        statistic='median', bins=10)
bin_width = (bin_edges[1] - bin_edges[0])
bin_centers = bin_edges[1:] - bin_width/2

plt.figure()
plt.hist(data, bins=30, density=True, histtype='stepfilled', alpha=0.3,␣
 ↪label='Histogram of Brightness Data')
plt.plot(data, dataPDF, ':', label = "Analytical pdf", lw=0.5)
plt.hlines(bin_median, bin_edges[:-1], bin_edges[1:], colors='purple', lw=4,␣
 ↪label='Binned Stat of Data')
plt.plot((binnumber - 0.5) * bin_width, dataPDF, 'g.', alpha=0.5)
plt.legend(fontsize=10)
plt.show()
```



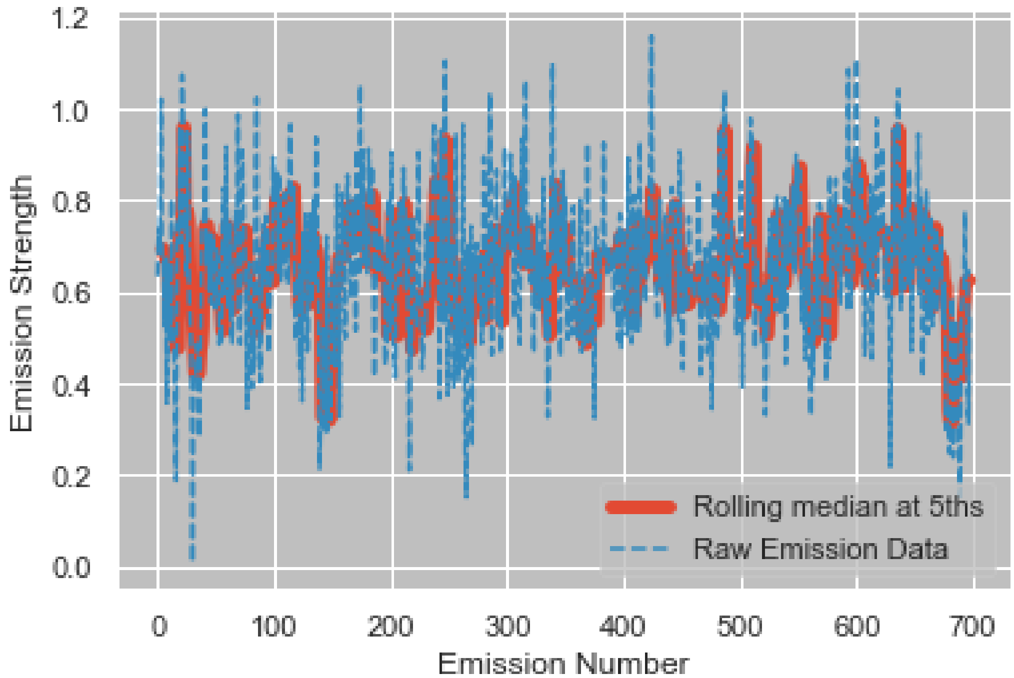
[ ]:
```
pulsar['RollingMeanEmissions5ths'] = pulsar["Brightness"].rolling(5).mean()

plt.plot(pulsar['RollingMeanEmissions5ths'], label="Rolling Mean at 5ths", lw=5)
plt.plot(pulsar['Brightness'], label= "Raw Emission Data", linestyle='--')
plt.legend()
plt.ylabel('Emission Strength')
plt.xlabel('Emission Number')
plt.show()
```



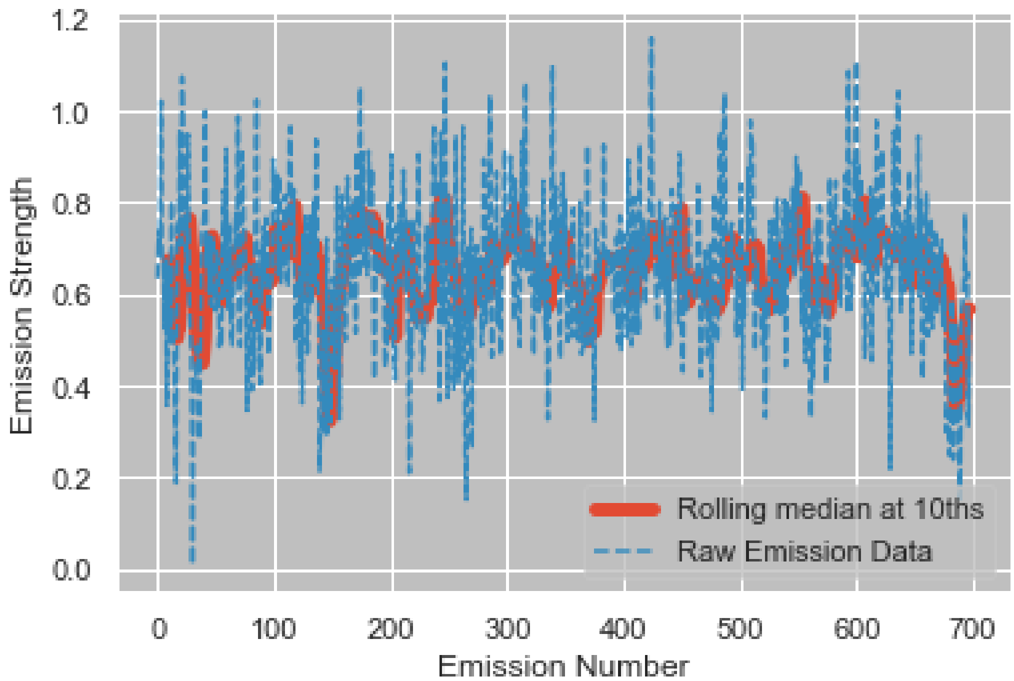
[ ]:
```
pulsar['RollingMeanEmissions10ths'] = pulsar["Brightness"].rolling(10).mean()

plt.plot(pulsar['RollingMeanEmissions10ths'], label="Rolling Mean at 10ths",␣
 ↪lw=5)
plt.plot(pulsar['Brightness'], label= "Raw Emission Data", linestyle='--')
plt.legend()
plt.ylabel('Emission Strength')
plt.xlabel('Emission Number')
plt.show()
```




```
[ ]: pulsar['RollingMedianEmissions5ths'] = pulsar["Brightness"].rolling(5).median()

plt.plot(pulsar['RollingMedianEmissions5ths'], label="Rolling median at 5ths",␣
 ↪lw=5)
plt.plot(pulsar['Brightness'], label= "Raw Emission Data", linestyle='--')
plt.legend()
plt.ylabel('Emission Strength')
plt.xlabel('Emission Number')
plt.show()
```




```
[ ]: pulsar['RollingMedianEmissions10ths'] = pulsar["Brightness"].rolling(10).
     ↪median()

     plt.plot(pulsar['RollingMedianEmissions10ths'], label="Rolling median at␣
     ↪10ths", lw=5)
     plt.plot(pulsar['Brightness'], label= "Raw Emission Data", linestyle='--')
     plt.legend()
     plt.ylabel('Emission Strength')
     plt.xlabel('Emission Number')
     plt.show()
```



[ ]:
```
pulsar.head(25)
```

[ ]:

	Pulse Number	Brightness	Uncertainty	Binary	RollingMeanEmissions5ths	\
0	1	0.634671	0.002761	0	NaN	
1	2	0.736945	0.005207	1	NaN	
2	3	0.693834	0.002706	1	NaN	
3	4	1.021866	0.010184	1	NaN	
4	5	0.673845	0.006236	1	0.752232	
5	6	0.676883	0.004763	1	0.760675	
6	7	0.527039	0.002422	0	0.718693	
7	8	0.673417	0.003174	1	0.714610	
8	9	0.357076	0.002848	0	0.581652	
9	10	0.661704	0.005588	1	0.579224	
10	11	0.545564	0.003835	0	0.552960	
11	12	0.494655	0.003145	0	0.546483	
12	13	0.804260	0.005258	1	0.572651	
13	14	0.513362	0.005700	0	0.603909	
14	15	0.477025	0.002945	0	0.566973	
15	16	0.399571	0.004712	0	0.537775	
16	17	0.188069	0.002452	0	0.476457	
17	18	0.748592	0.005468	1	0.465324	
18	19	0.723437	0.004548	1	0.507339	
19	20	0.960154	0.006765	1	0.603965	
20	21	0.707715	0.006011	1	0.665593	

21	22	1.074550	0.006831	1	0.842890
22	23	0.961340	0.006617	1	0.885439
23	24	0.754457	0.004117	1	0.891643
24	25	0.773151	0.004920	1	0.854242

	RollingMeanEmissions10ths	RollingMedianEmissions5ths \
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	0.693834
5	NaN	0.693834
6	NaN	0.676883
7	NaN	0.673845
8	NaN	0.673417
9	0.665728	0.661704
10	0.656817	0.545564
11	0.632588	0.545564
12	0.643631	0.545564
13	0.592780	0.545564
14	0.573098	0.513362
15	0.545367	0.494655
16	0.511470	0.477025
17	0.518988	0.477025
18	0.555624	0.477025
19	0.585469	0.723437
20	0.601684	0.723437
21	0.659673	0.748592
22	0.675381	0.960154
23	0.699491	0.960154
24	0.729103	0.773151

	RollingMedianEmissions10ths
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	0.673631
10	0.673631
11	0.667561
12	0.667561
13	0.603634

```
14                       0.536301
15                       0.520201
16                       0.504008
17                       0.504008
18                       0.529463
19                       0.529463
20                       0.610538
21                       0.715576
22                       0.715576
23                       0.736015
24                       0.751524
```

5 Kalman Filter to go here

[ ]:

```

```

5.1 Binary Classification

[ ]:

```
X = pulsar[['Brightness', 'Uncertainty']]
y = pulsar['Binary']
```

[ ]:

```
X.head()
```

[ ]:

```
   Brightness  Uncertainty
0    0.634671     0.002761
1    0.736945     0.005207
2    0.693834     0.002706
3    1.021866     0.010184
4    0.673845     0.006236
```

[ ]:

```
y.head()
```

[ ]:

```
0    0
1    1
2    1
3    1
4    1
Name: Binary, dtype: int32
```

[ ]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.20)
```

[ ]:

```
from sklearn.preprocessing import StandardScaler

train_scaler = StandardScaler()
X_train = train_scaler.fit_transform(X_train)
```

```
test_scaler = StandardScaler()
X_test = test_scaler.fit_transform(X_test)
```

[ ]:
```
model = LogisticRegression()

model.fit(X_train, y_train)
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
predictions = model.predict(X_test)
```

[ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)

TN, FP, FN, TP = confusion_matrix(y_test, predictions).ravel()

print('True Positive(TP)  = ', TP)
print('False Positive(FP) = ', FP)
print('True Negative(TN)  = ', TN)
print('False Negative(FN) = ', FN)
```

```
True Positive(TP)  =  67
False Positive(FP) =  0
True Negative(TN)  =  71
False Negative(FN) =  2
```

[ ]:
```
accuracy =  (TP + TN) / (TP + FP + TN + FN)

print("Accuracy of the model is ", accuracy)
```

```
Accuracy of the model is  0.9857142857142858
```

5.2 Bidirectional LSTM Model

[ ]:
```
# making a list with the brightness and uncertainty values
values_list = pulsar[['Brightness', 'Uncertainty']].values.tolist()
values_list[:10]
```

[ ]:
```
[[0.6346714, 0.002760888],
 [0.7369454, 0.005207055],
 [0.6938341, 0.0027059],
 [1.021866, 0.01018372],
 [0.6738453, 0.006235539],
 [0.6768825, 0.004762893],
 [0.5270392, 0.002422239],
 [0.6734173, 0.003174072],
```

```
 [0.3570756, 0.00284815],
 [0.6617037, 0.005587867]]
```

[ ]:
```python
from sklearn import preprocessing

# normalizing the values
values_list = preprocessing.normalize(values_list)
```

[ ]:
```python
# function for spliting  list in a format we can use in the model
def split_list(blist, steps):
    X, y = list(), list()
    for i in range(len(blist)):
        end_ix = i + steps
        if end_ix > len(blist)-1:
            break
        list_x, list_y = blist[i:end_ix], blist[end_ix][0]
        X.append(list_x)
        y.append(list_y)
    return array(X), array(y)
```

[ ]:
```python
# splitting the list
X, y = split_list(values_list, 100)

# reshaping the list to feed the model
X = X.reshape((X.shape[0], X.shape[1], 2))
```

[ ]:
```python
# splitting the list into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.20)
```

[ ]:
```python
# setting the parameters for the lstm model and compiling it
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(100, 2)))
model.add(Dense(25, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[ ]:
```python
# training the model
history = model.fit(X_train, y_train, epochs=50, verbose=1,␣
 ↪batch_size=(int(X_train.shape[0]/50)))
```

```
Epoch 1/50
54/54 [==============================] - 6s 41ms/step - loss: 0.6672 - accuracy:
0.0000e+00
Epoch 2/50
54/54 [==============================] - 2s 41ms/step - loss: 147.9319 -
```

```
accuracy: 0.0000e+00
Epoch 3/50
54/54 [==============================] - 2s 40ms/step - loss: 0.3789 - accuracy:
0.0000e+00
Epoch 4/50
54/54 [==============================] - 2s 41ms/step - loss: 0.3077 - accuracy:
0.0000e+00
Epoch 5/50
54/54 [==============================] - 2s 41ms/step - loss: 0.2549 - accuracy:
0.0000e+00
Epoch 6/50
54/54 [==============================] - 2s 40ms/step - loss: 0.2139 - accuracy:
0.0000e+00
Epoch 7/50
54/54 [==============================] - 2s 41ms/step - loss: 0.1816 - accuracy:
0.0000e+00
Epoch 8/50
54/54 [==============================] - 2s 41ms/step - loss: 0.1556 - accuracy:
0.0000e+00
Epoch 9/50
54/54 [==============================] - 2s 40ms/step - loss: 0.1344 - accuracy:
0.0000e+00
Epoch 10/50
54/54 [==============================] - 2s 41ms/step - loss: 0.1170 - accuracy:
0.0000e+00
Epoch 11/50
54/54 [==============================] - 2s 42ms/step - loss: 0.1025 - accuracy:
0.0000e+00
Epoch 12/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0904 - accuracy:
0.0000e+00
Epoch 13/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0800 - accuracy:
0.0000e+00
Epoch 14/50
54/54 [==============================] - 2s 37ms/step - loss: 0.0713 - accuracy:
0.0000e+00
Epoch 15/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0637 - accuracy:
0.0000e+00
Epoch 16/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0571 - accuracy:
0.0000e+00
Epoch 17/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0515 - accuracy:
0.0000e+00
Epoch 18/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0465 - accuracy:
```

```
0.0000e+00
Epoch 19/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0421 - accuracy:
0.0000e+00
Epoch 20/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0382 - accuracy:
0.0000e+00
Epoch 21/50
54/54 [==============================] - 2s 38ms/step - loss: 0.0348 - accuracy:
0.0000e+00
Epoch 22/50
54/54 [==============================] - 2s 37ms/step - loss: 0.0317 - accuracy:
0.0000e+00
Epoch 23/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0289 - accuracy:
0.0000e+00
Epoch 24/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0265 - accuracy:
0.0000e+00
Epoch 25/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0242 - accuracy:
0.0000e+00
Epoch 26/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0222 - accuracy:
0.0000e+00
Epoch 27/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0204 - accuracy:
0.0000e+00
Epoch 28/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0188 - accuracy:
0.0000e+00
Epoch 29/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0173 - accuracy:
0.0000e+00
Epoch 30/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0159 - accuracy:
0.0000e+00
Epoch 31/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0147 - accuracy:
0.0000e+00
Epoch 32/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0136 - accuracy:
0.0000e+00
Epoch 33/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0125 - accuracy:
0.0000e+00
Epoch 34/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0115 - accuracy:
```

```
0.0000e+00
Epoch 35/50
54/54 [==============================] - 2s 43ms/step - loss: 0.0107 - accuracy:
0.0000e+00
Epoch 36/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0098 - accuracy:
0.0000e+00
Epoch 37/50
54/54 [==============================] - 2s 44ms/step - loss: 0.0091 - accuracy:
0.0000e+00
Epoch 38/50
54/54 [==============================] - 2s 43ms/step - loss: 0.0084 - accuracy:
0.0000e+00
Epoch 39/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0077 - accuracy:
0.0000e+00
Epoch 40/50
54/54 [==============================] - 2s 39ms/step - loss: 0.0071 - accuracy:
0.0000e+00
Epoch 41/50
54/54 [==============================] - 2s 40ms/step - loss: 0.0066 - accuracy:
0.0000e+00
Epoch 42/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0060 - accuracy:
0.0000e+00
Epoch 43/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0056 - accuracy:
0.0000e+00
Epoch 44/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0051 - accuracy:
0.0000e+00
Epoch 45/50
54/54 [==============================] - 2s 41ms/step - loss: 0.0047 - accuracy:
0.0000e+00
Epoch 46/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0043 - accuracy:
0.0000e+00
Epoch 47/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0039 - accuracy:
0.0000e+00
Epoch 48/50
54/54 [==============================] - 2s 42ms/step - loss: 0.0035 - accuracy:
0.0000e+00
Epoch 49/50
54/54 [==============================] - 2s 37ms/step - loss: 0.0032 - accuracy:
0.0000e+00
Epoch 50/50
54/54 [==============================] - 2s 35ms/step - loss: 0.0029 - accuracy:
```

```
0.0000e+00
```

```
[ ]: # predicting the y/brightness values for the test set
y_pred = model.predict(X_test, verbose=0)
y_pred[:10]
```

```
[ ]: array([[0.99742967],
        [0.9974283 ],
        [0.99742913],
        [0.9974295 ],
        [0.99742895],
        [0.9974301 ],
        [0.99742985],
        [0.99742836],
        [0.9974307 ],
        [0.99742925]], dtype=float32)
```

```
[ ]: # evaluating the model
model.evaluate(X_test, y_test)
```

```
4/4 [==============================] - 0s 12ms/step - loss: 0.0027 - accuracy:
0.0000e+00
```

```
[ ]: [0.0027466383762657642, 0.0]
```

5.3 ML Evaluation.

5.3.1 Logistic Regression

Rewards no significant results for this type of analysis and is dropped for a LSTM attempt

5.3.2 Bidirectional LSTM

Loss is low so the model is performing well. But the accuracy is low therefore unable to obtain trend and therefore not rewarding any information. This means we cannot predict any of the values with confidence.

6 Preliminary runs test

6.0.1 Math Logic

$$Z = \frac{R - \tilde{R}}{s_R}$$

$$\tilde{R} = \frac{2_{n1n2}}{n1 + n2} + 1$$

$$s_R^2 = \frac{2_{n1n2}(2n1n2 - n1 - n2)}{(n1 + n2)^2(n1 + n2 - 1)}$$

link to resource: https://www.geeksforgeeks.org/runs-test-of-randomness-in-python/

$ Z_{critical} = 1.96 $ as the confidence interval level of 95% thus this is a 2 tailed test. If the probability as corrosponding to this confidence interval $ H_{null} $ will be rejected as it is not statistically significant as denoted by $|Z| > Z_{critical} $

There is also code attempting to change it from a z-score probability to a P-score for ease of understanding and clarity.

7 FUNCTION CODE FOR RUNS TEST

[ ]:
```
binaryData1 = pulsar['Binary'].tolist()
print("pulsar6 original: ",binaryData1)
```

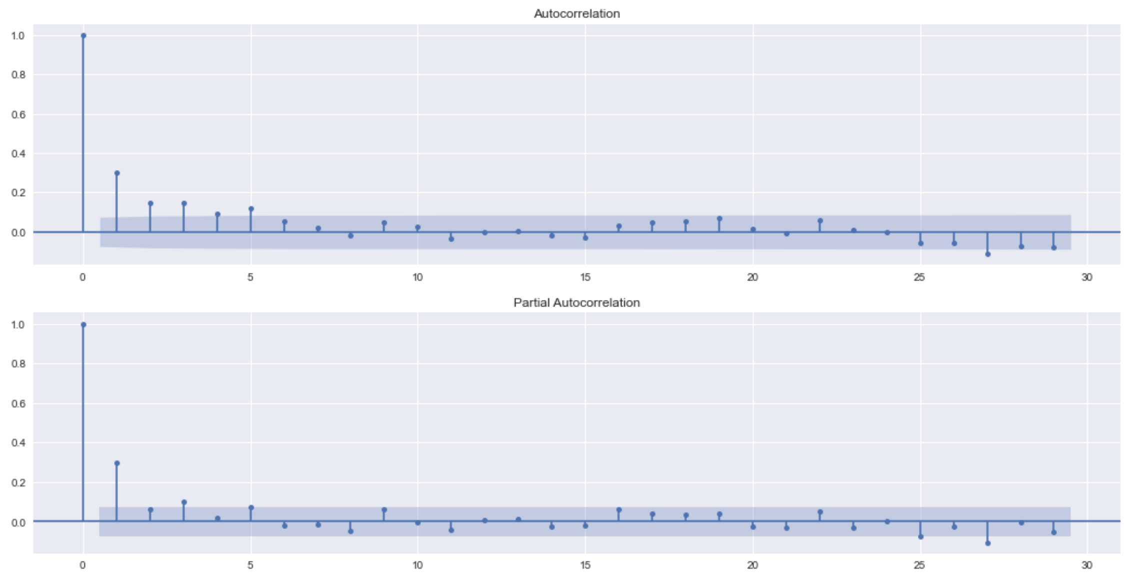
```
pulsar6 original:  [0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1,
1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0]
```

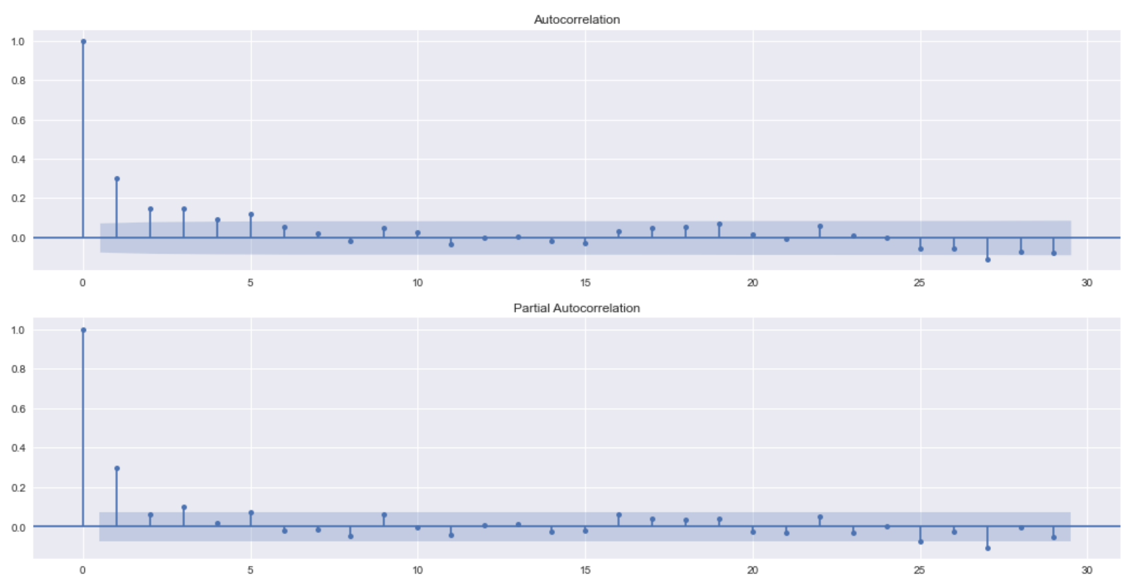
8 Below we begin autocorrelation and autocovariance analysis

To get started with this I am playing around with guide from: https://towardsdatascience.com/a-step-by-step-guide-to-calculating-autocorrelation-and-partial-autocorrelation-8c4342b784e8

[ ]:
```
plt.style.use("seaborn")
plt.rcParams["figure.figsize"] = (18, 9)

fig, ax = plt.subplots(2,1)

plot_acf(pulsar['Brightness'], ax=ax[0])
plot_pacf(pulsar['Brightness'], ax=ax[1], method="ols")
```

[ ]:





[ ]:
```
acf(pulsar['Brightness'], nlags=10)
```

```
C:\Users\tajki\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:667:
FutureWarning: fft=True will become the default after the release of the 0.12
release of statsmodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(
```

[ ]:
```
array([ 1.        ,  0.29929122,  0.14656878,  0.14948301,  0.09384681,
        0.11707783,  0.05493324,  0.02160374, -0.01711482,  0.04777   ,
        0.02563995])
```

[ ]:
```python
acfpulsar = pd.DataFrame()
for lag in range(0,11):
    acfpulsar[f"B_lag_{lag}"] = pulsar['Brightness'].shift(lag)


acfpulsar
```

[ ]:
```
       B_lag_0   B_lag_1   B_lag_2   B_lag_3   B_lag_4   B_lag_5   B_lag_6  \
0     0.634671       NaN       NaN       NaN       NaN       NaN       NaN
1     0.736945  0.634671       NaN       NaN       NaN       NaN       NaN
2     0.693834  0.736945  0.634671       NaN       NaN       NaN       NaN
3     1.021866  0.693834  0.736945  0.634671       NaN       NaN       NaN
4     0.673845  1.021866  0.693834  0.736945  0.634671       NaN       NaN
..         …         …         …         …         …         …         …
693   0.776083  0.623757  0.581248  0.555266  0.152886  0.286132  0.413354
694   0.625382  0.776083  0.623757  0.581248  0.555266  0.152886  0.286132
695   0.647559  0.625382  0.776083  0.623757  0.581248  0.555266  0.152886
696   0.312449  0.647559  0.625382  0.776083  0.623757  0.581248  0.555266
697   0.548353  0.312449  0.647559  0.625382  0.776083  0.623757  0.581248

       B_lag_7   B_lag_8   B_lag_9  B_lag_10
0          NaN       NaN       NaN       NaN
1          NaN       NaN       NaN       NaN
2          NaN       NaN       NaN       NaN
3          NaN       NaN       NaN       NaN
4          NaN       NaN       NaN       NaN
..         …         …         …         …
693   0.460095  0.541486  0.346502  0.239302
694   0.413354  0.460095  0.541486  0.346502
695   0.286132  0.413354  0.460095  0.541486
696   0.152886  0.286132  0.413354  0.460095
697   0.555266  0.152886  0.286132  0.413354

[698 rows x 11 columns]
```

[ ]:
```python
acfpulsar.corr()["B_lag_0"].values
```

[ ]:
```
array([ 1.        ,  0.29938402,  0.14710414,  0.15003691,  0.09455452,
        0.11800036,  0.05537751,  0.02179885, -0.01724535,  0.04863954,
```

```
0.02621294])
```

8.0.1 Getting every 5th as per the auto correlation

8.0.2 Creating a new set of discrete 100 sets and examining them specifically

8.0.3 Further Random testing to move into extensive testing

Getting every 5th as per the auto correlation

```
[ ]: held5ths = pulsar[pulsar.index % 5 == 0]
     held5ths
```

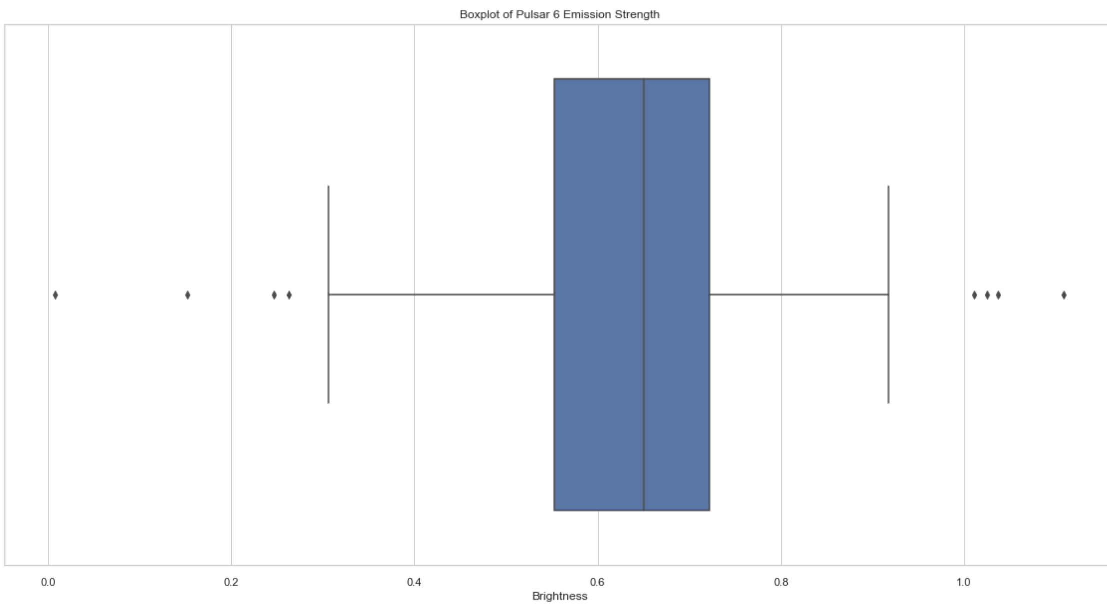
```
[ ]:      Pulse Number  Brightness  Uncertainty  Binary  RollingMeanEmissions5ths  \
     0               1    0.634671     0.002761       0                       NaN
     5               6    0.676883     0.004763       1                  0.760675
     10             11    0.545564     0.003835       0                  0.552960
     15             16    0.399571     0.004712       0                  0.537775
     20             21    0.707715     0.006011       1                  0.665593
     ..            …         …            …        …                      …
     675           676    0.618826     0.002507       0                  0.634092
     680           681    0.246916     0.004276       0                  0.363455
     685           686    0.541486     0.003149       0                  0.444185
     690           691    0.555266     0.003657       0                  0.373547
     695           696    0.647559     0.003765       0                  0.650806

          RollingMeanEmissions10ths  RollingMedianEmissions5ths  \
     0                          NaN                         NaN
     5                          NaN                    0.693834
     10                    0.656817                    0.545564
     15                    0.545367                    0.494655
     20                    0.601684                    0.723437
     ..                         …                           …
     675                   0.648694                    0.630414
     680                   0.498773                    0.316197
     685                   0.403820                    0.505084
     690                   0.408866                    0.413354
     695                   0.512176                    0.625382

          RollingMedianEmissions10ths
     0                            NaN
     5                            NaN
     10                      0.673631
     15                      0.520201
     20                      0.610538
     ..                           …
     675                     0.650434
     680                     0.572860
     685                     0.344559
```

```
690                        0.436725
695                        0.568257

[140 rows x 8 columns]
```

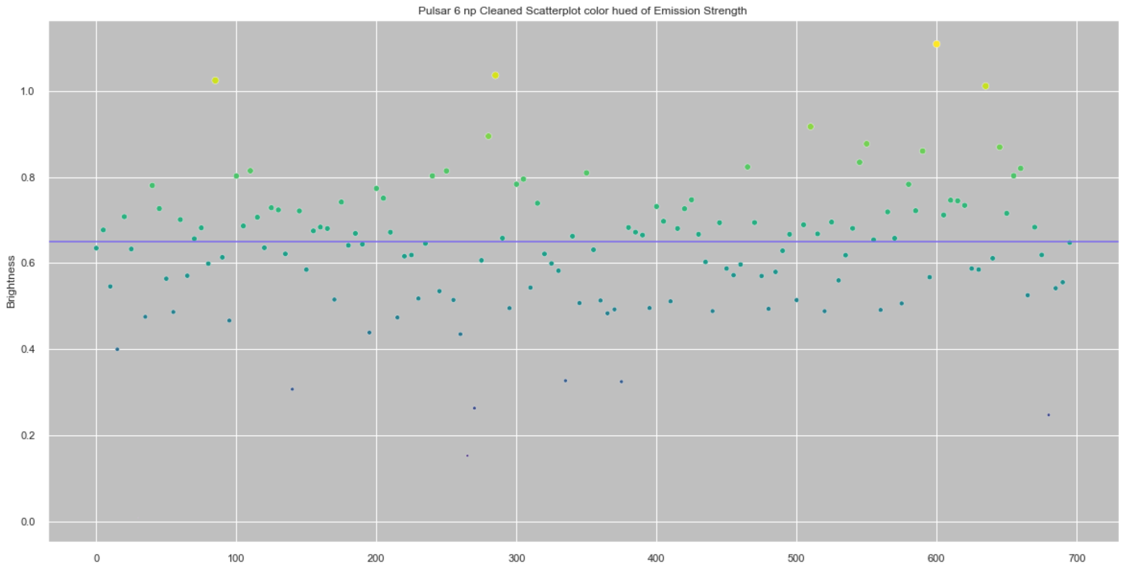
[ ]:
```
medianheld5ths = held5ths["Brightness"].median()
medianheld5ths
```

[ ]: 0.6508051

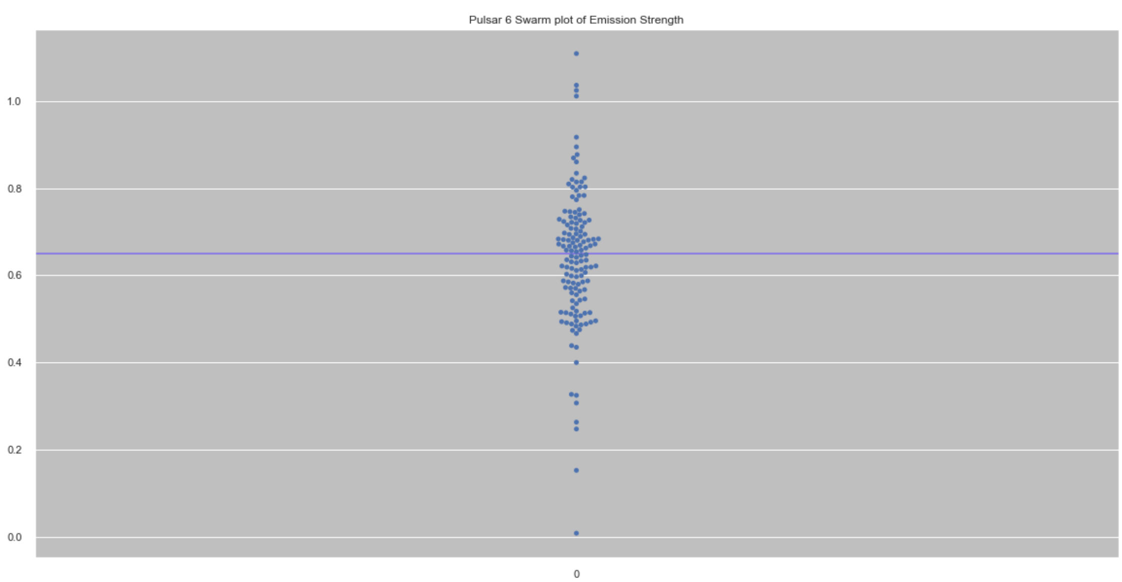
[ ]:
```
plt.figure(figsize=(20,10))
sns.set_theme(style="whitegrid")
ax = sns.boxplot(x=held5ths["Brightness"]).set_title("Boxplot of Pulsar 6␣
↪Emission Strength")
```



[ ]:
```
plt.figure(figsize=(20,10))
sns.set_style("darkgrid", {"axes.facecolor": ".75"})
strength = held5ths.Brightness.values
ax = sns.scatterplot(data=held5ths["Brightness"], s= strength*50, c=strength,␣
↪cmap="viridis", marker="o").set_title('Pulsar 6 np Cleaned Scatterplot color␣
↪hued of Emission Strength')
ax = plt.axhline( y=0.6508051, ls='-',c='mediumslateblue')
```



[ ]:
```
plt.figure(figsize=(20,10))
sns.set_style("darkgrid", {"axes.facecolor": ".75"})
strength = held5ths.Brightness.values
ax = plt.axhline( y=0.6508051, ls='-',c='mediumslateblue')
ax = sns.swarmplot(data=held5ths["Brightness"], c="blue").set_title('Pulsar 6␣
↪Swarm plot of Emission Strength')
```



[ ]:
```
print(len(held5ths[(held5ths.Brightness > 0.6508051)]))
print(len(held5ths[(held5ths.Brightness < 0.6508051)]))
```

```
70
70
```

Randomness testing

[ ]:
```
np.savetxt(r'every5thbinarypulsar6.txt', held5ths.Binary, fmt='%d',␣
 ↪delimiter='')
np.savetxt(r'allpulsar6.txt', pulsar.Binary, fmt='%d', delimiter='')
```

[ ]:
```
pulsar.Binary
```

[ ]:
```
0      0
1      1
2      1
3      1
4      1
      ..
693    1
694    0
695    0
696    0
697    0
Name: Binary, Length: 698, dtype: int32
```