app

January 15, 2023

[500]:
```
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
 ↪f1_score
from scipy.signal import butter, lfilter, find_peaks_cwt, find_peaks,␣
 ↪periodogram
from scipy.stats import kurtosis, skew
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, KFold, cross_val_score,␣
 ↪cross_validate
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import make_scorer
from sklearn.preprocessing import normalize
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import learning_curve

import pickle
import numpy as np
import matplotlib.pyplot as plt

import antropy as ent

import random
import os
```

[501]:
```
SEED = 57
SAMPLING_FREQUENCY = 173.6
```

[502]:
```
random.seed(SEED)
os.environ['PYTHONHASHSEED'] = str(SEED)
np.random.seed(SEED)
```

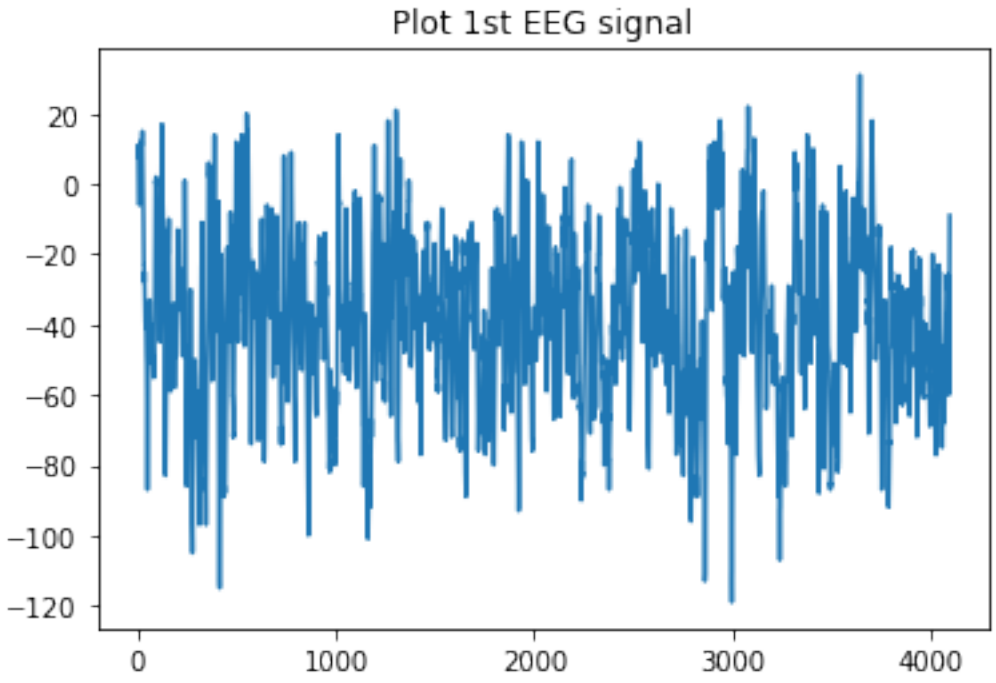
[503]:
```
x = pickle.load(open('../data/x.pkl', 'rb'))
y = pickle.load(open('../data/y.pkl', 'rb'))
```

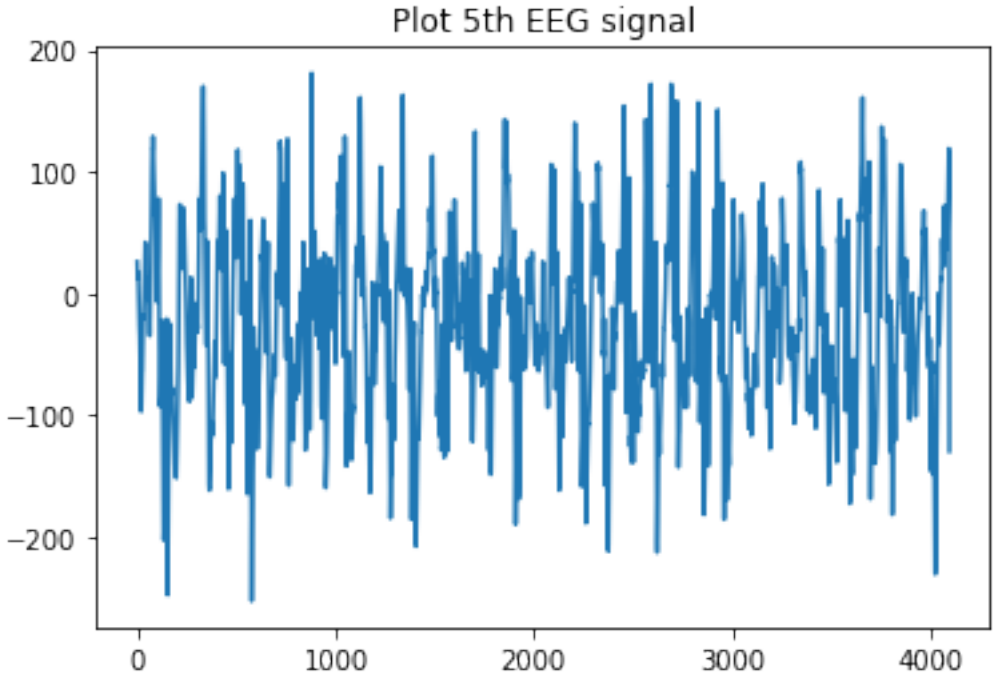
[504]:
```
x_normal = np.concatenate((x[:300], x[400:]), axis=0)
x_seizure = x[300:400]

print(x_normal.shape)
print(x_seizure.shape)

b, a = butter(3, [0.5,40], btype='bandpass',fs=SAMPLING_FREQUENCY)

x_normal_filtered = np.array([lfilter(b,a,x_normal[index,:]) for index in␣
 ↪range(x_normal.shape[0])])
x_seizure_filtered = np.array([lfilter(b,a,x_seizure[index,:]) for index in␣
 ↪range(x_seizure.shape[0])])

print(x_normal.shape)
print(x_seizure.shape)

x = np.concatenate((x_normal, x_seizure))
y = np.concatenate((np.zeros((400, 1)), np.ones((100, 1))))

print(x.shape)
print(y.shape)
```

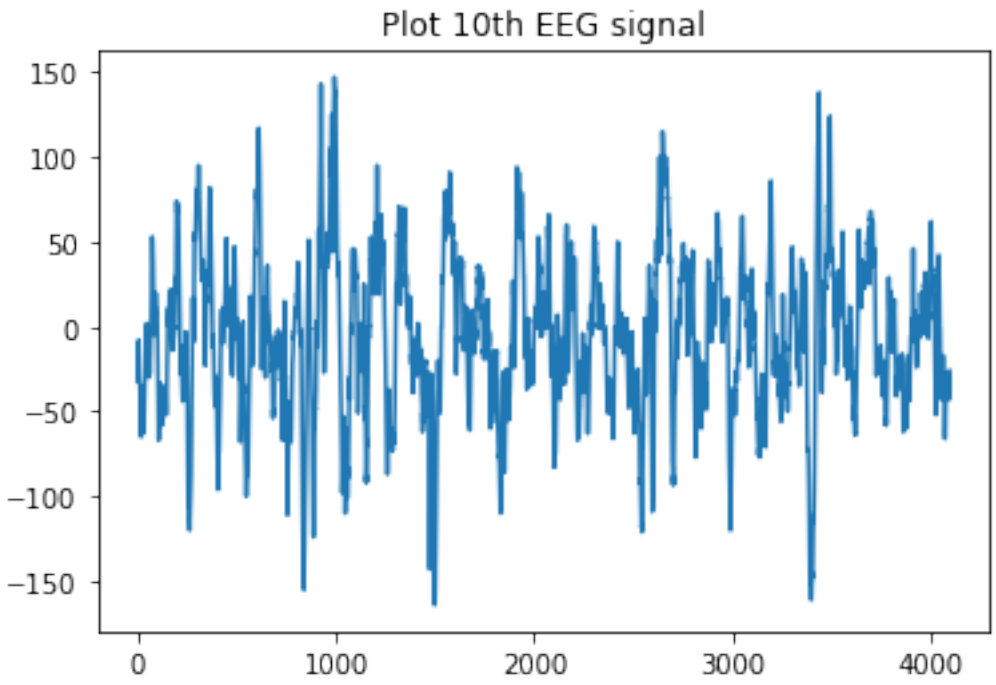
```
(400, 4097)
(100, 4097)
(400, 4097)
(100, 4097)
(500, 4097)
(500, 1)
```

[505]:
```
# plot some of the signals and their y values
plt.title('Plot 1st EEG signal')
plt.plot(x[0])
plt.show()

plt.title('Plot 5th EEG signal')
plt.plot(x[5])
plt.show()

plt.title('Plot 10th EEG signal')
plt.plot(x[10])
```

```
plt.show()
```


Plot 1st EEG signal
20
0
−20
−40
−60
−80
−100
−120
0
1000
2000
3000
4000


Plot 5th EEG signal
200
100
0
−100
−200
0
1000
2000
3000
4000



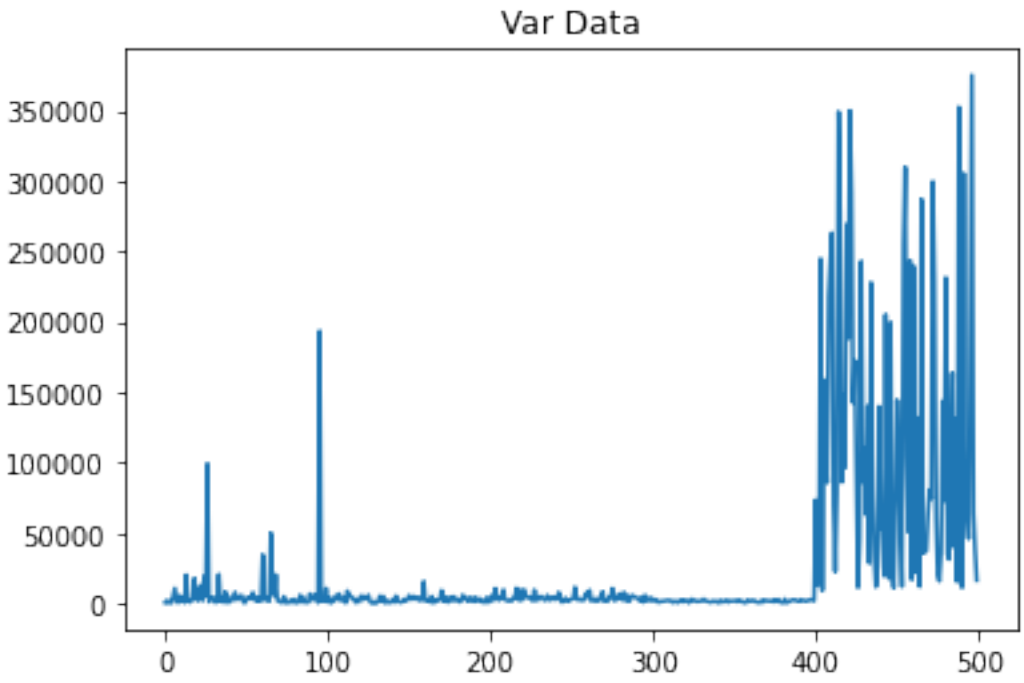
0.1 Statistical analysis

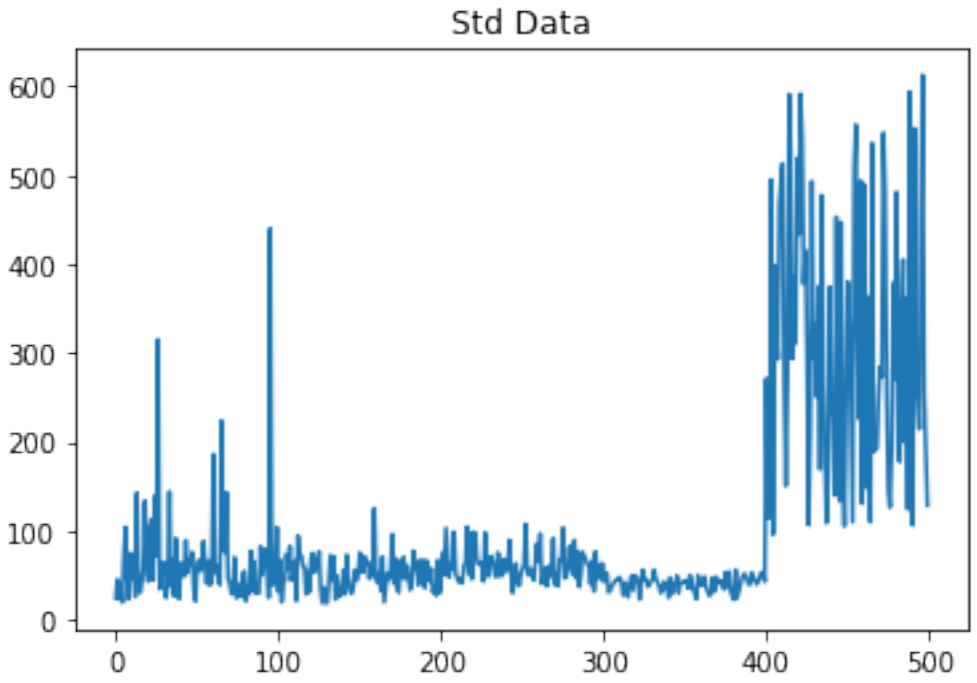
In order to visualize out data, we begin with the statistical analysis of our data, which we perform with the help of NumPy. Our statistical data are as follows: - Mean - Median - Max - Min - Std - Var - Sqrt - Arg Min - Arg Max

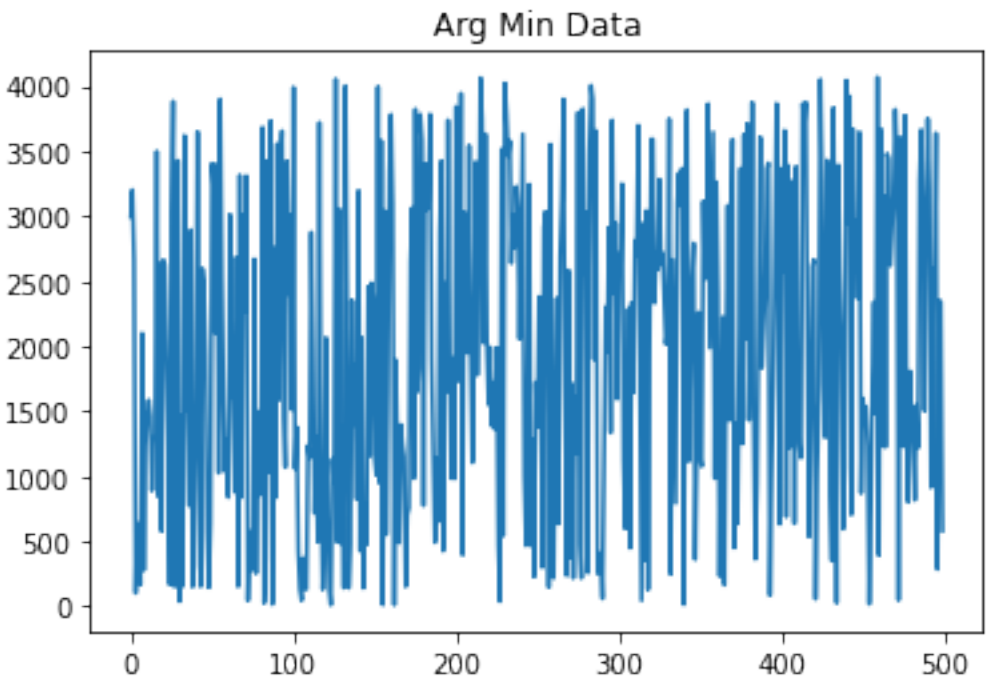
[506]:
```
def mean(data):
    return np.mean(data, axis=1)

def median(data):
    return np.median(data, axis=1)

def std(data):
    return np.std(data, axis=1)

def var(data):
    return np.var(data, axis=1)

def minimum(data):
    return np.min(data, axis=1)

def maximum(data):
    return np.max(data, axis=1)

def arg_min(data):
```

```
    return np.argmin(data, axis=1)

def arg_max(data):
    return np.argmax(data, axis= 1)

def sqrt(data):
    return np.sqrt(np.mean(data**2, axis=-1))

var = var(x)
std = std(x)
mean = mean(x)
median = median(x)
maximum = maximum(x)
minimum = minimum(x)
arg_min = arg_min(x)
arg_max = arg_max(x)
sqrt = sqrt(x)
```

[507]:
```
plt.title("Var Data")
plt.plot(var)
plt.show()

plt.title("Std Data")
plt.plot(std)
plt.show()

plt.title("Arg Min Data")
plt.plot(arg_min)
plt.show()
```


Var Data
350000
300000
250000
200000
150000
100000
50000
0
0
100
200
300
400
500


Std Data
600
500
400
300
200
100
0
0
100
200
300
400
500



Let's build our new x based on statistical features calculated before.

[508]:
```
var = var.reshape(-1, 1)
std = std.reshape(-1, 1)
maximum = maximum.reshape(-1, 1)
minimum = minimum.reshape(-1, 1)
mean = mean.reshape(-1, 1)
median = median.reshape(-1, 1)
sqrt = sqrt.reshape(-1, 1)
arg_min = arg_min.reshape(-1, 1)
arg_max = arg_max.reshape(-1, 1)

new_x = np.concatenate((var, std, maximum, minimum, mean, median, sqrt,␣
 ↪arg_max, arg_min), axis=1)
new_x.shape
```

[508]:
```
(500, 9)
```

[509]:
```
x_train, x_test, y_train ,y_test = train_test_split(new_x, y,␣
 ↪random_state=SEED, test_size=0.2)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(400, 9)
(100, 9)
(400, 1)
(100, 1)
```

[510]:
```
clf = SVC(kernel='linear', max_iter=30000000)
clf.fit(x_train, y_train)

y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.93
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\svm\_base.py:301: ConvergenceWarning: Solver terminated early
(max_iter=30000000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

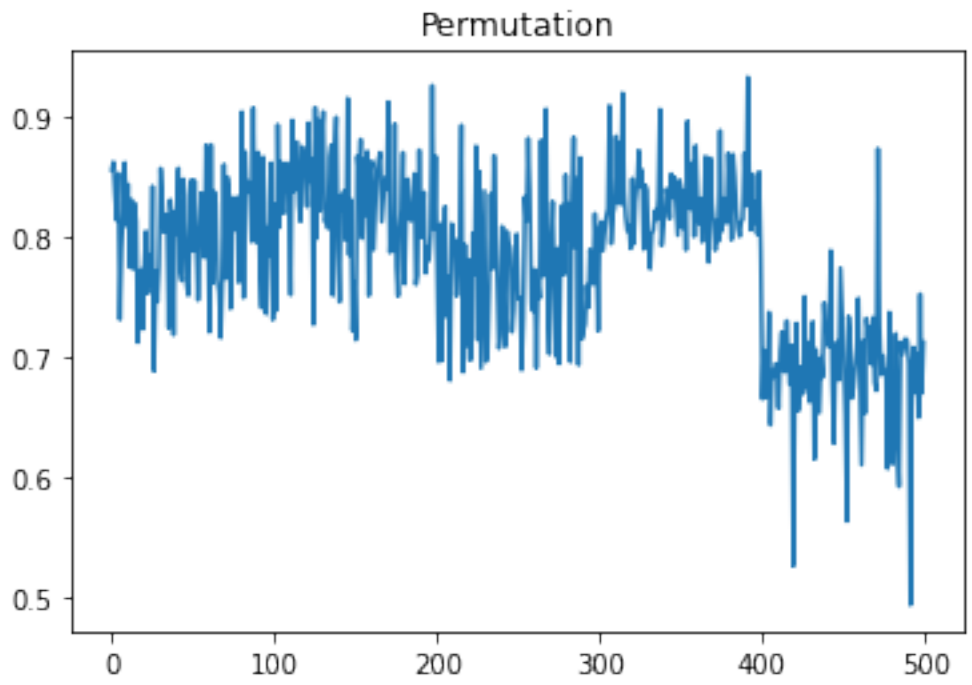
0.2 Entropy analysis

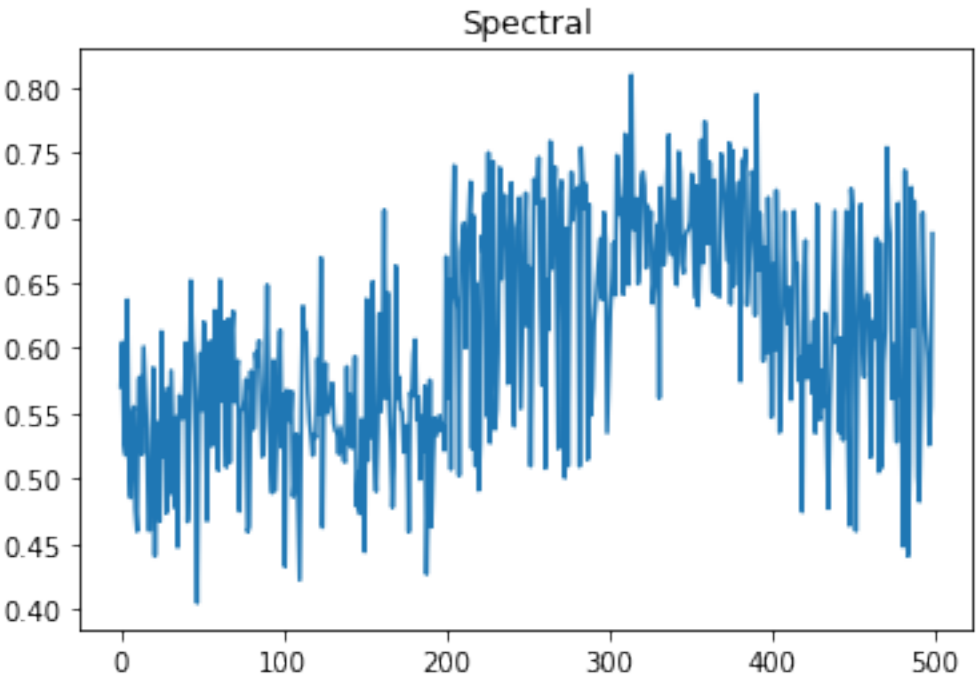
Entropy is proportional to the degree of disorder in a thermodynamic process. The higher the degree of disorder, the higher the entropy. We continue with the entropy analysis of our data, which we perform with the help of AntroPy. Our entropic data are as follows: - Permutation - Specular - Single Value Decomposition - Approximate - Sample - Lempel Ziv

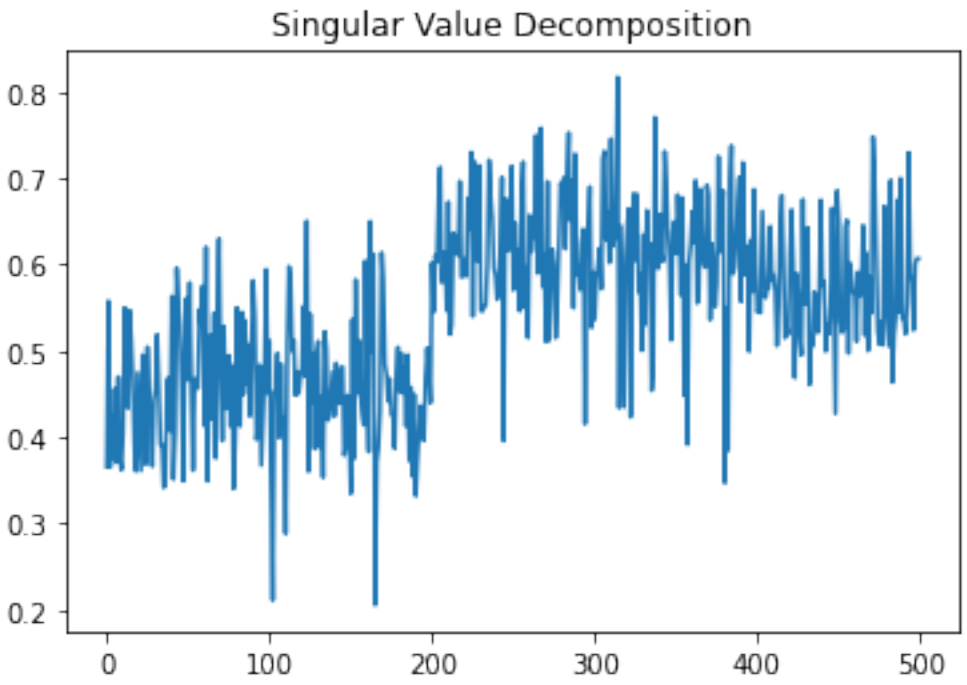
[511]:
```
# Permutation entropy
def permutation(data):
    return ent.perm_entropy(data, normalize=True)

# Spectral entropy
def spectral(data):
    return ent.spectral_entropy(data, sf=100, method='welch', normalize=True)

# Singular value decomposition entropy
def singular_value_decomposition(data):
    return ent.svd_entropy(data, normalize=True)

# Approximate entropy
def approximate(data):
    return ent.app_entropy(data)
```

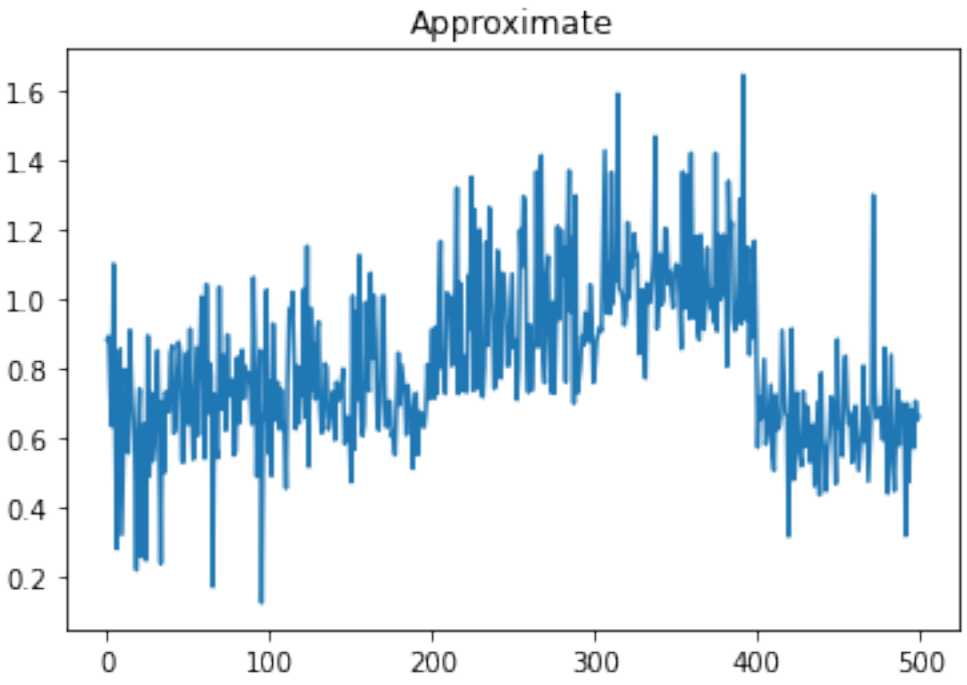
```
# Sample entropy
def sample(data):
    return ent.sample_entropy(data)

# Lempel-Ziv complexity
def lempel_ziv(data):
    return ent.lziv_complexity('01111000011001', normalize=True)

permutation = np.array([permutation(x[i, :]) for i in range(x.shape[0])])
spectral = np.array([spectral(x[i, :]) for i in range(x.shape[0])])
singular_value_decomposition = np.array([singular_value_decomposition(x[i, :])␣
 ↪for i in range(x.shape[0])])
approximate = np.array([approximate(x[i, :]) for i in range(x.shape[0])])
lempel_ziv = np.array([lempel_ziv(x[i, :]) for i in range(x.shape[0])])
sample = np.array([sample(x[i, :]) for i in range(x.shape[0])])
```

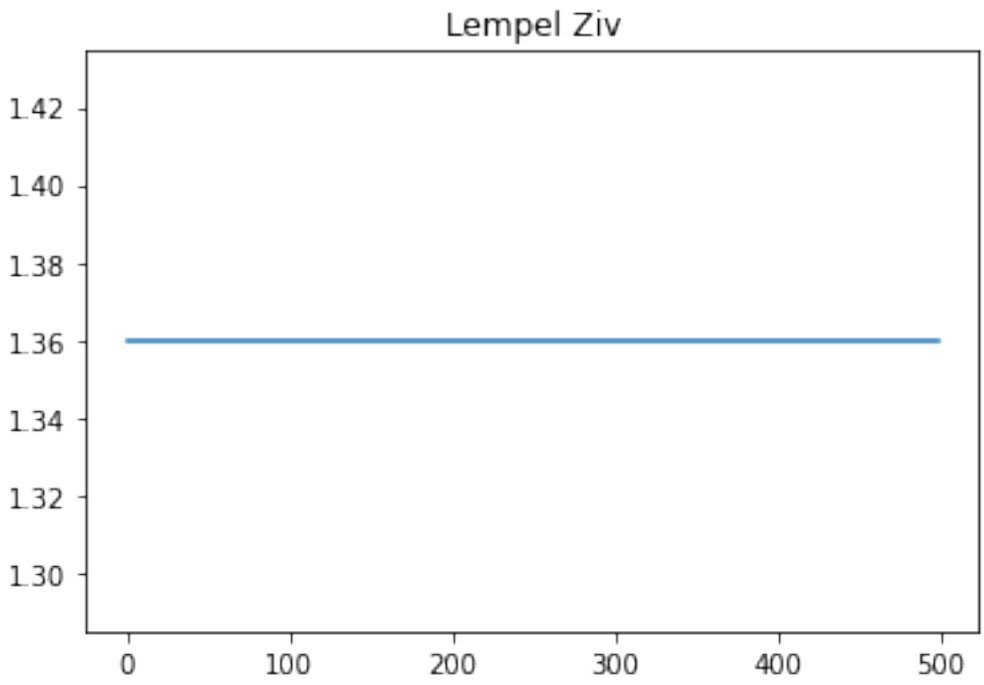
[512]:
```
plt.title('Permutation')
plt.plot(permutation)
plt.show()

plt.title('Spectral')
plt.plot(spectral)
plt.show()

plt.title('Singular Value Decomposition')
plt.plot(singular_value_decomposition)
plt.show()

plt.title('Approximate')
plt.plot(approximate)
plt.show()

plt.title('Lempel Ziv')
plt.plot(lempel_ziv)
plt.show()

plt.title('Sample')
plt.plot(sample)
plt.show()
```

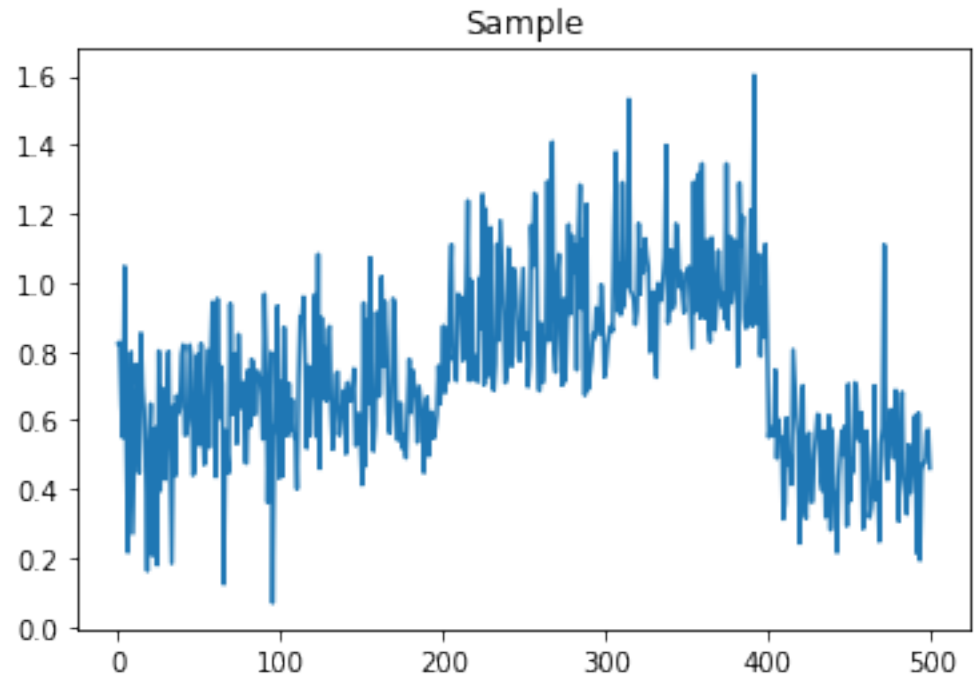

Permutation
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400
500


Spectral
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
100
200
300
400
500


Singular Value Decomposition
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500


Approximate
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
100
200
300
400
500





Let's build our new x based on fast fourier transform features calculated before.

[513]:
```
permutation = permutation.reshape(-1, 1)
spectral = spectral.reshape(-1, 1)
approximate = approximate.reshape(-1, 1)
singular_value_decomposition = singular_value_decomposition.reshape(-1, 1)
lempel_ziv = lempel_ziv.reshape(-1, 1)
sample = sample.reshape(-1, 1)

new_x = np.concatenate((permutation, spectral, approximate,␣
 ↪singular_value_decomposition, lempel_ziv, sample), axis=1)
new_x.shape
```

[513]:
```
(500, 6)
```

[514]:
```
x_train, x_test, y_train ,y_test = train_test_split(new_x, y,␣
 ↪random_state=SEED, test_size=0.2)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(400, 6)
(100, 6)
(400, 1)
(100, 1)
```

[515]:
```
clf = SVC(kernel='linear', max_iter=20000000)
clf.fit(x_train, y_train)

y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
```

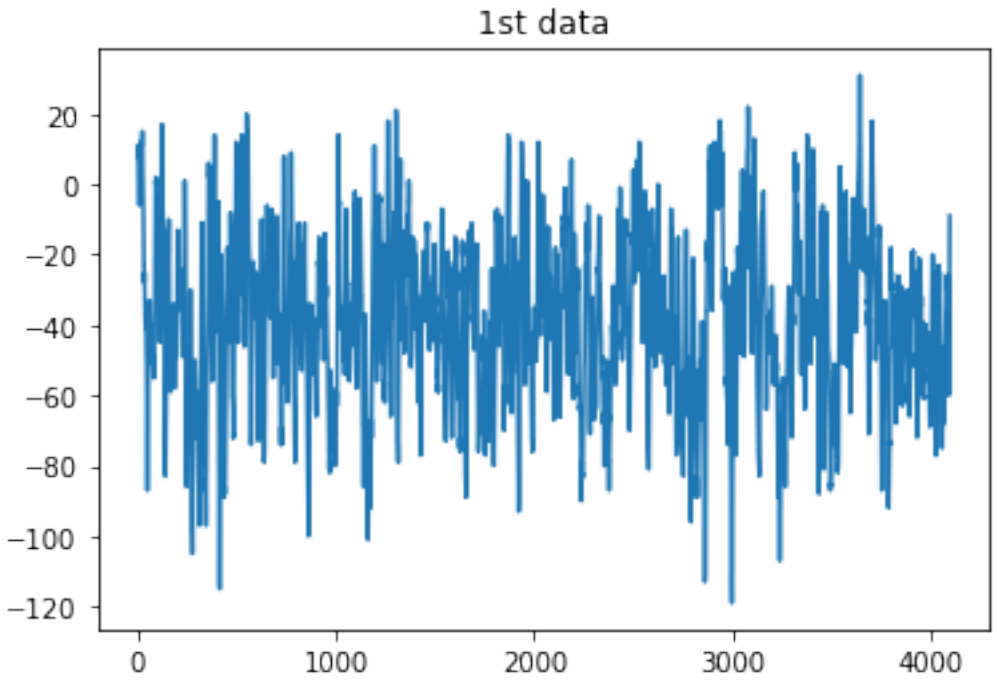
```
0.91
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

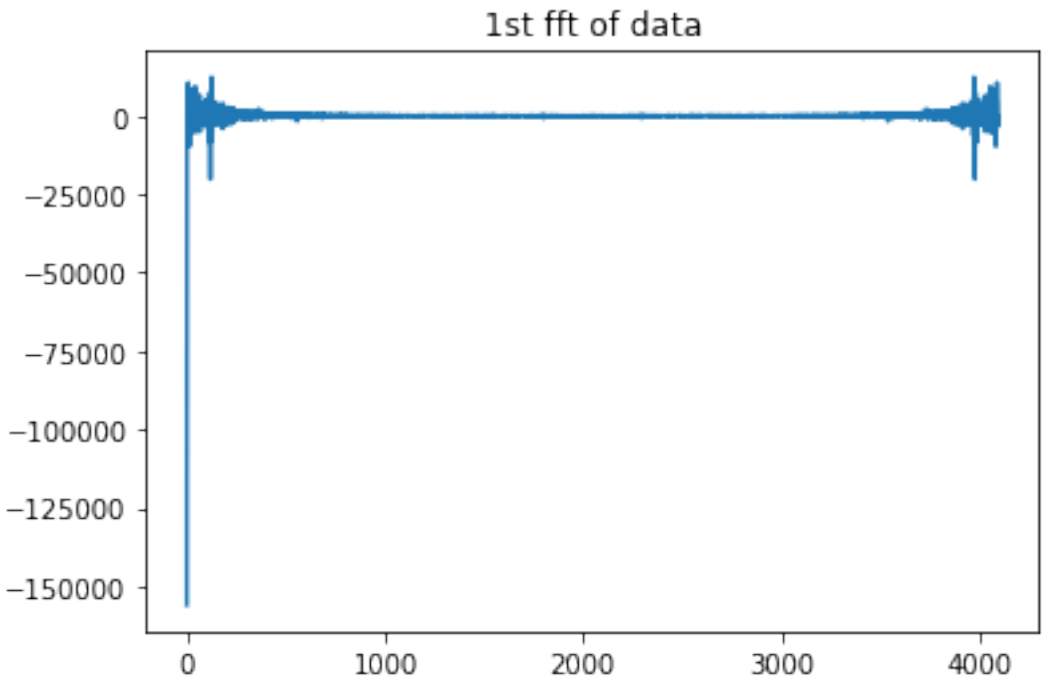
0.3 What is FFT?

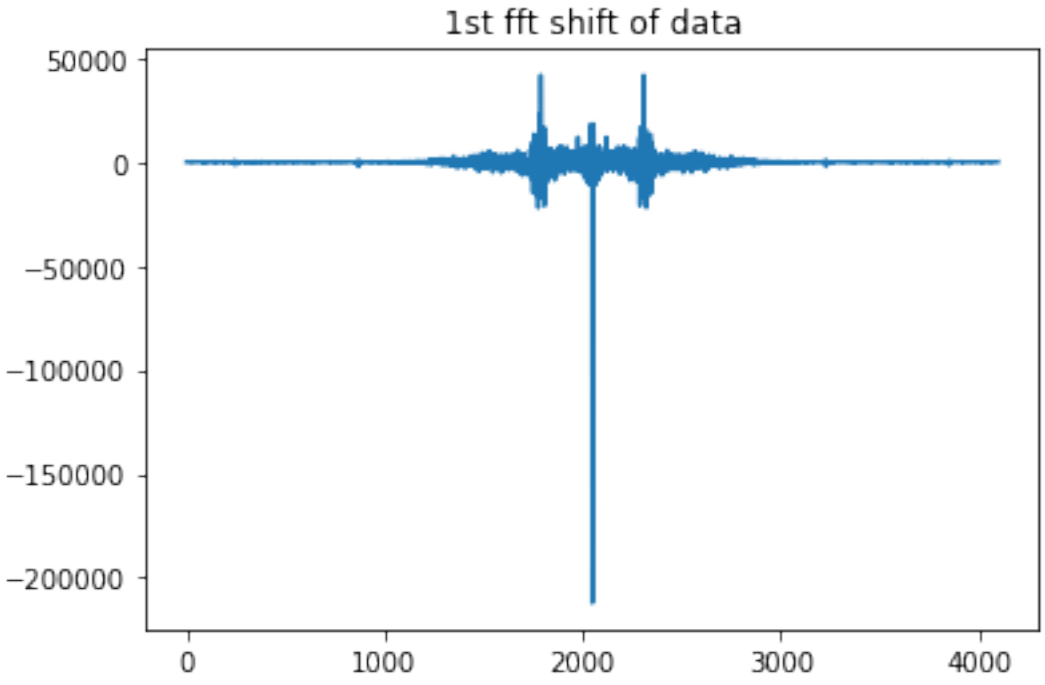
A fast Fourier transform is an algorithm that computes the discrete Fourier transform of a sequence, or its inverse. Fourier analysis converts a signal from its original domain to a representation in the frequency domain and vice versa.

[516]:
```
fft = np.fft.fft(x)
fft_shift = np.fft.fftshift(fft)

plt.title('1st data')
plt.plot(x[0])
plt.show()

plt.title('1st fft of data')
plt.plot(fft[0])
plt.show()

plt.title('1st fft shift of data')
plt.plot(fft_shift[0])
plt.show()
```



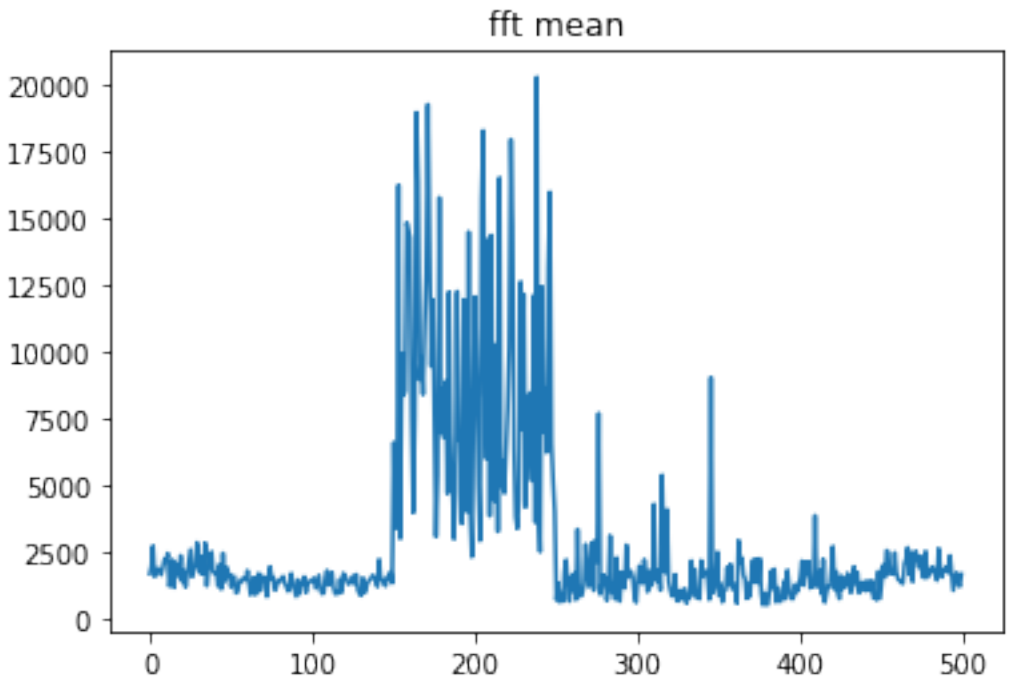

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\matplotlib\cbook\__init__.py:1298: ComplexWarning: Casting complex
values to real discards the imaginary part
  return np.asarray(x, float)
```

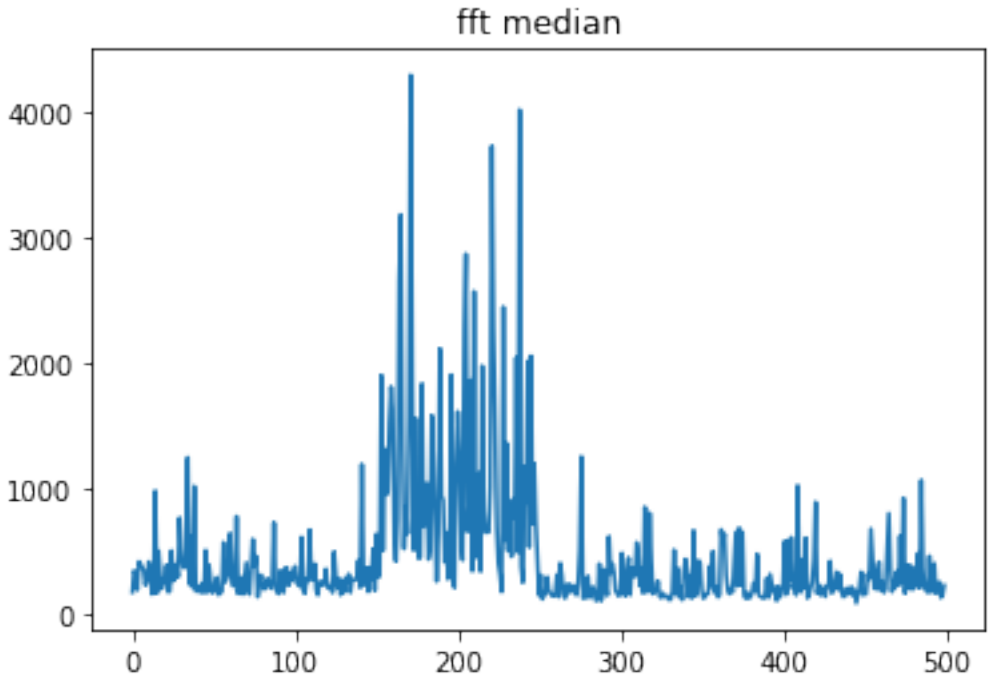


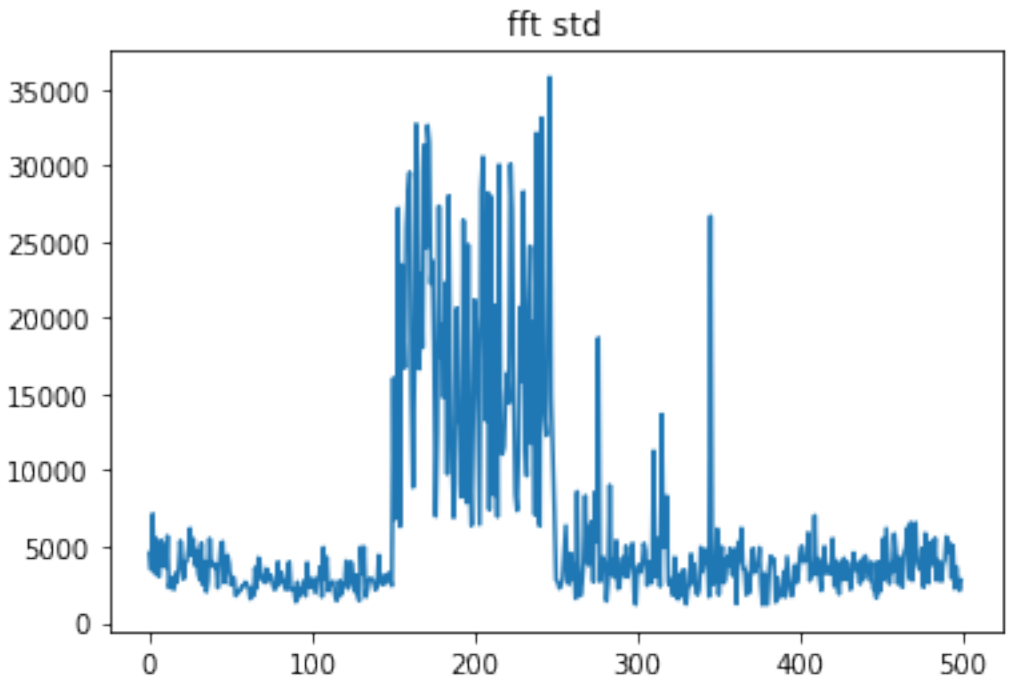


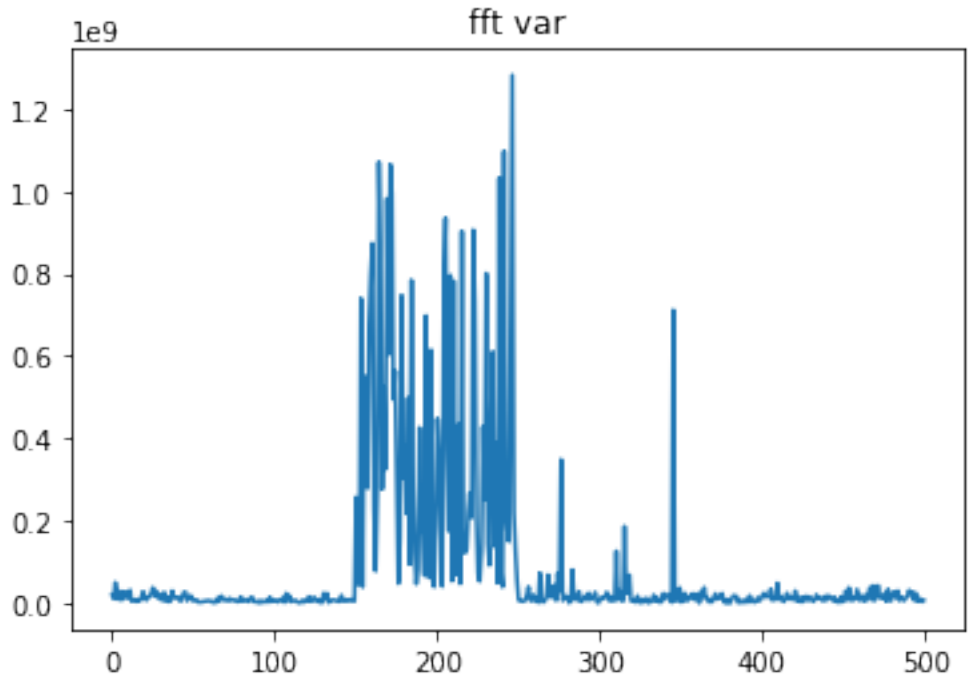
[517]:
```
fft_abs = np.abs(fft_shift)
fft_mean = np.mean(fft_abs, axis=1)
```

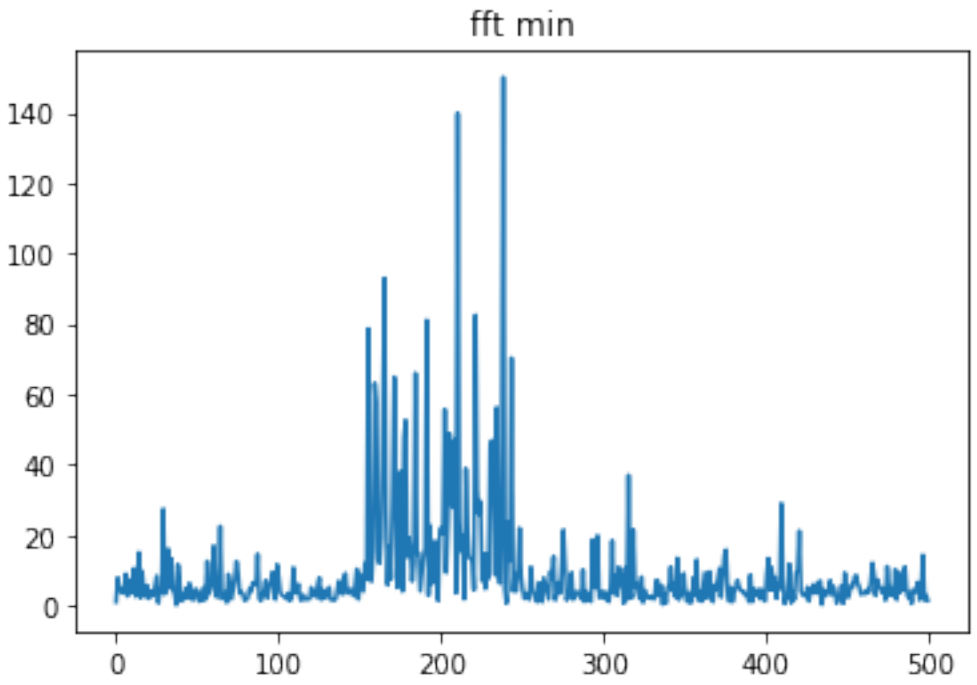
```
fft_var = np.var(fft_abs, axis=1)
fft_median = np.median(fft_abs, axis=1)
fft_std = np.std(fft_abs, axis=1)
fft_max = np.max(fft_abs, axis=1)
fft_min = np.min(fft_abs, axis=1)

plt.title('fft mean')
plt.plot(fft_mean)
plt.show()

plt.title('fft median')
plt.plot(fft_median)
plt.show()

plt.title('fft std')
plt.plot(fft_std)
plt.show()

plt.title('fft var')
plt.plot(fft_var)
plt.show()

plt.title('fft min')
plt.plot(fft_min)
plt.show()

plt.title('fft max')
plt.plot(fft_max)
plt.show()
```

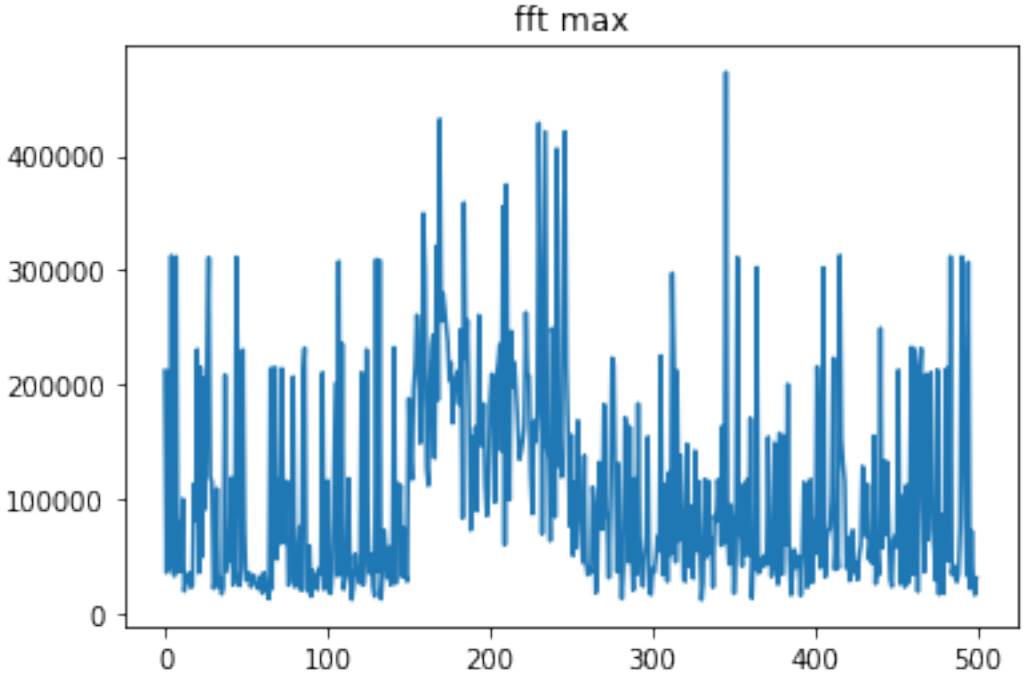

fft mean
20000
17500
15000
12500
10000
7500
5000
2500
0
0
100
200
300
400
500


fft median
4000
3000
2000
1000
0
0
100
200
300
400
500


fft std
35000
30000
25000
20000
15000
10000
5000
0
0
100
200
300
400
500


fft var
1e9
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500





Let's build our new x based on fast fourier transform features calculated before.

[518]:
```
fft_mean = fft_mean.reshape(-1, 1)
fft_median = fft_median.reshape(-1, 1)
fft_var = fft_var.reshape(-1, 1)
fft_std = fft_std.reshape(-1, 1)
fft_max = fft_max.reshape(-1, 1)
fft_min = fft_min.reshape(-1, 1)

new_x = np.concatenate((fft_mean, fft_median, fft_std, fft_var, fft_max,␣
 ↪fft_min), axis=1)
print(new_x.shape)
```

```
(500, 6)
```

[519]:
```
x_train, x_test, y_train ,y_test = train_test_split(new_x, y,␣
 ↪random_state=SEED, test_size=0.2)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(400, 6)
(100, 6)
(400, 1)
(100, 1)
```

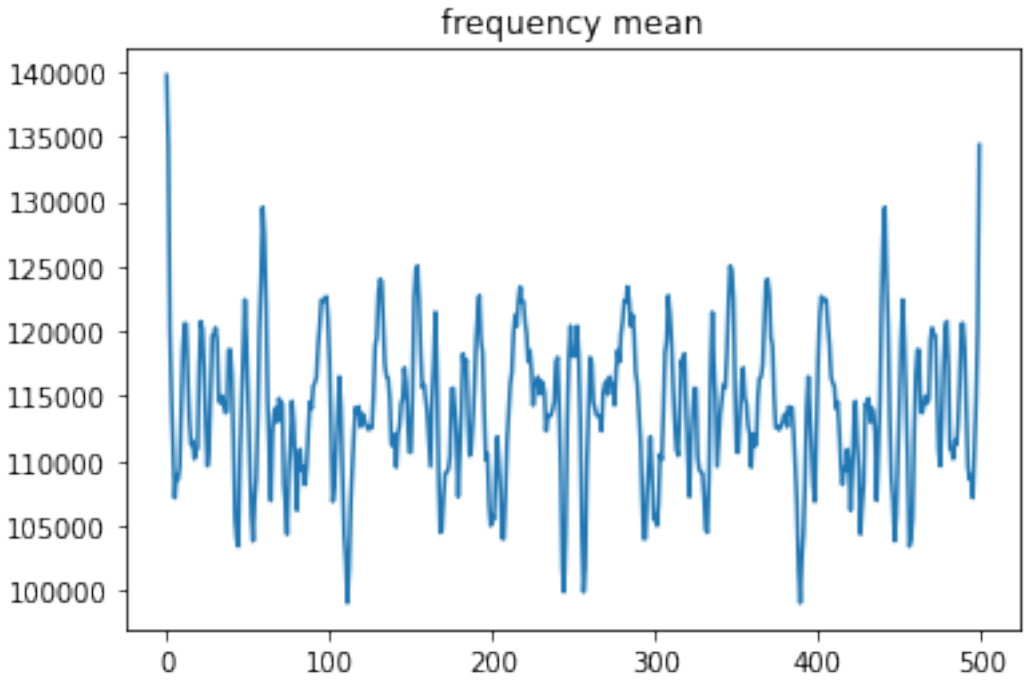
[520]:
```
clf = SVC(kernel='linear', max_iter=20000000)
clf.fit(x_train, y_train)

y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
```

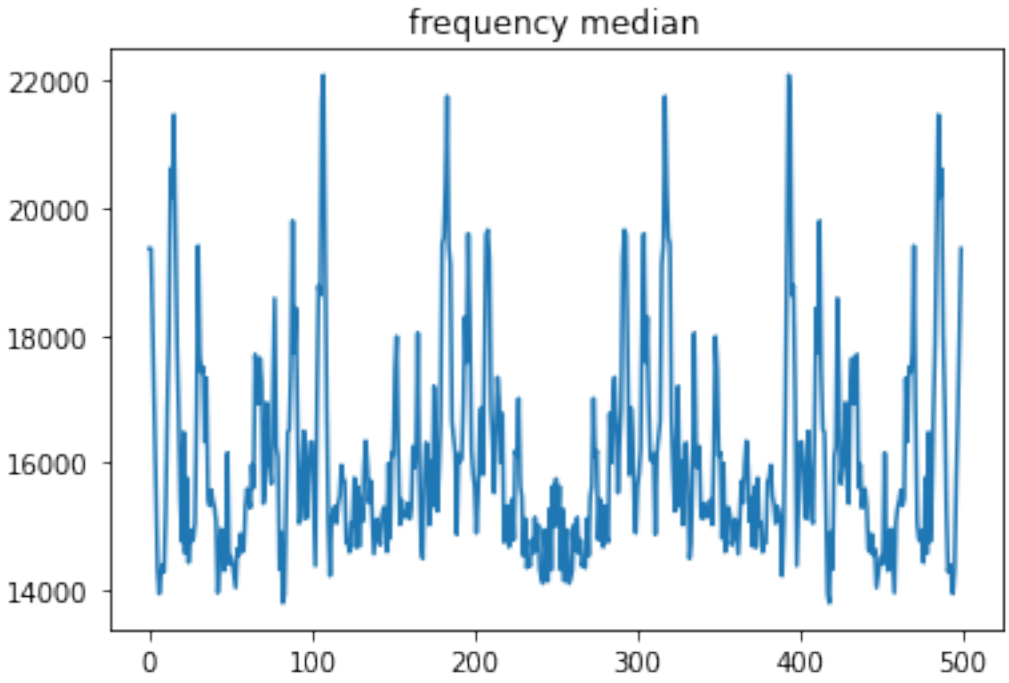
```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

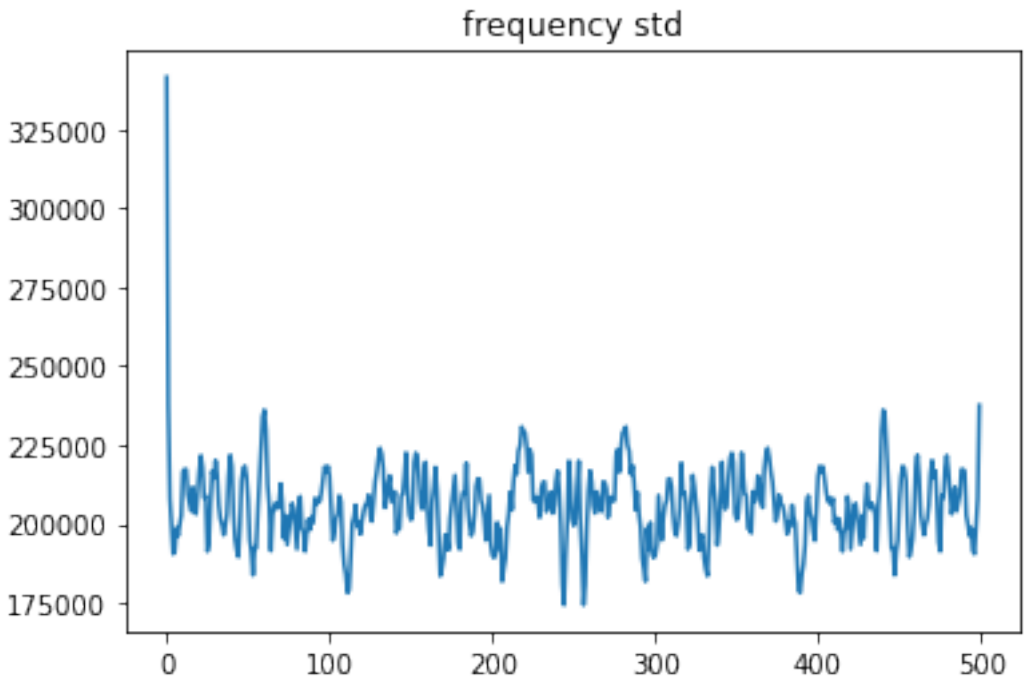
```
0.74
```

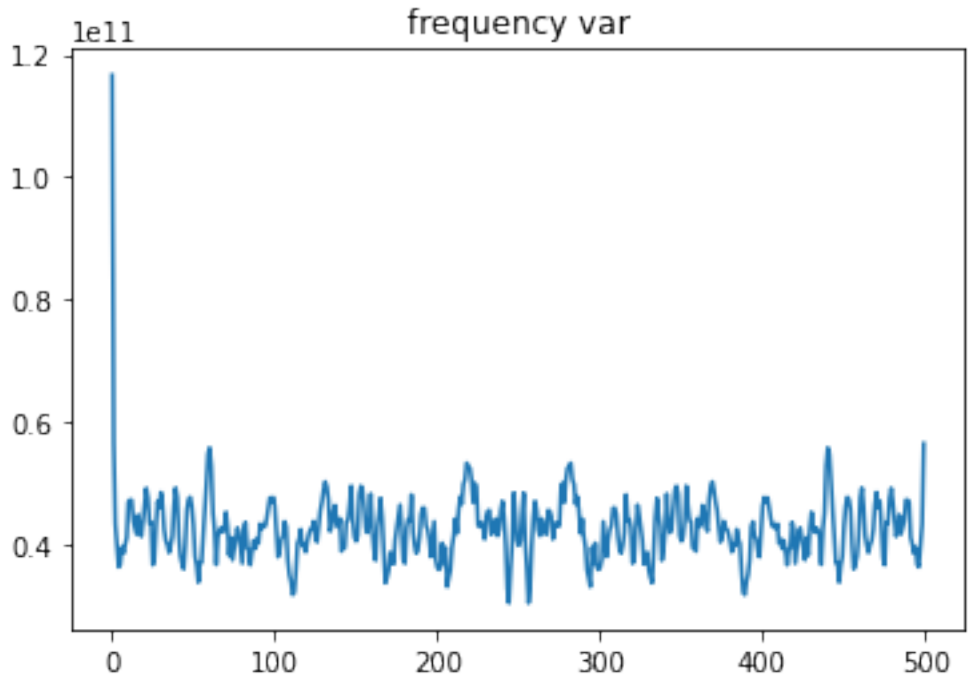
```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\svm\_base.py:301: ConvergenceWarning: Solver terminated early
(max_iter=20000000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

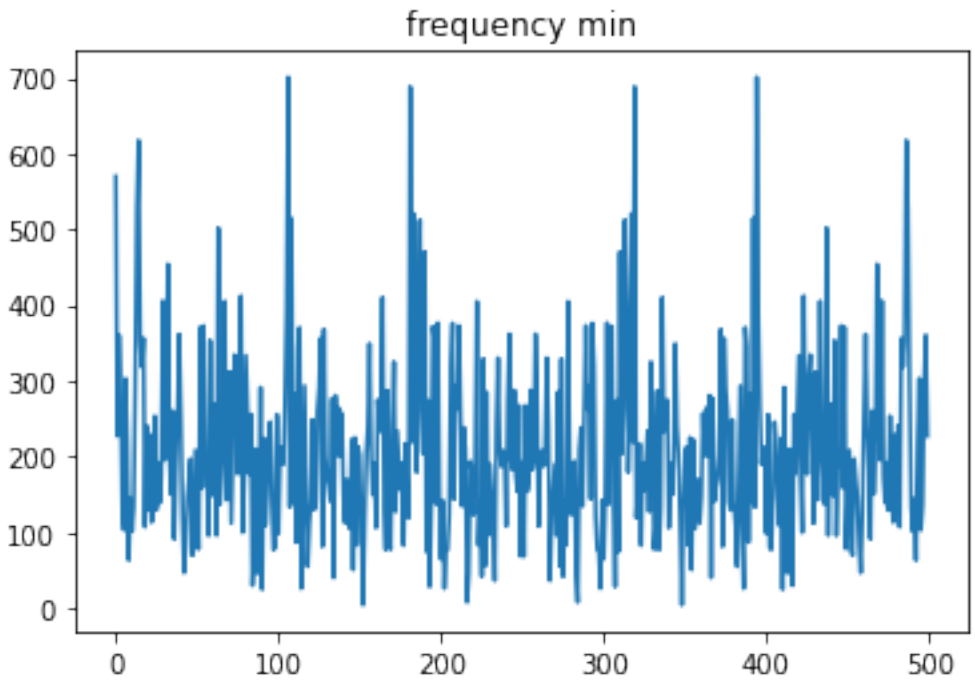
[521]:
```
frequency = np.fft.fft2(x)
frequency_abs = np.abs(frequency)
frequency_mean = np.mean(frequency_abs, axis=1)
frequency_median = np.median(frequency_abs, axis=1)
frequency_var = np.var(frequency_abs, axis=1)
frequency_std = np.std(frequency_abs, axis=1)
frequency_min = np.min(frequency_abs, axis=1)
frequency_max = np.max(frequency_abs, axis=1)

plt.title('frequency mean')
plt.plot(frequency_mean)
plt.show()

plt.title('frequency median')
plt.plot(frequency_median)
plt.show()

plt.title('frequency std')
plt.plot(frequency_std)
plt.show()

plt.title('frequency var')
plt.plot(frequency_var)
plt.show()

plt.title('frequency min')
plt.plot(frequency_min)
plt.show()

plt.title('frequency max')
plt.plot(frequency_max)
plt.show()
```

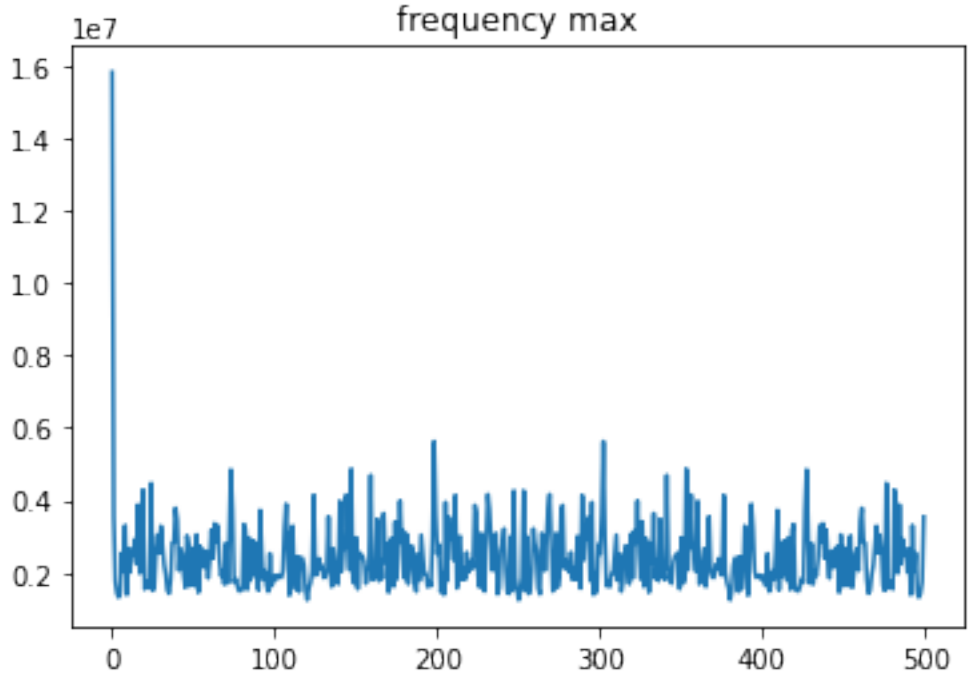

frequency mean
140000
135000
130000
125000
120000
115000
110000
105000
100000
0
100
200
300
400
500


frequency median
22000
20000
18000
16000
14000
0
100
200
300
400
500


frequency std
325000
300000
275000
250000
225000
200000
175000
0
100
200
300
400
500


frequency var
1e11
1.2
1.0
0.8
0.6
0.4
0
100
200
300
400
500





Let's build our new x based on fast fourier transform features calculated before.

[522]:
```
frequency_mean = frequency_mean.reshape(-1, 1)
frequency_median = frequency_median.reshape(-1, 1)
frequency_var = frequency_var.reshape(-1, 1)
frequency_std = frequency_std.reshape(-1, 1)
frequency_max = frequency_max.reshape(-1, 1)
frequency_min = frequency_min.reshape(-1, 1)

new_x = np.concatenate((frequency_mean, frequency_median, frequency_std,␣
 ↪frequency_var, frequency_max, frequency_min), axis=1)
print(new_x.shape)
```

```
(500, 6)
```

[523]:
```
x_train, x_test, y_train ,y_test = train_test_split(new_x, y,␣
 ↪random_state=SEED, test_size=0.2)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(400, 6)
(100, 6)
(400, 1)
(100, 1)
```

[524]:
```
clf = SVC(kernel='linear', max_iter=30000000)
clf.fit(x_train, y_train)

y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

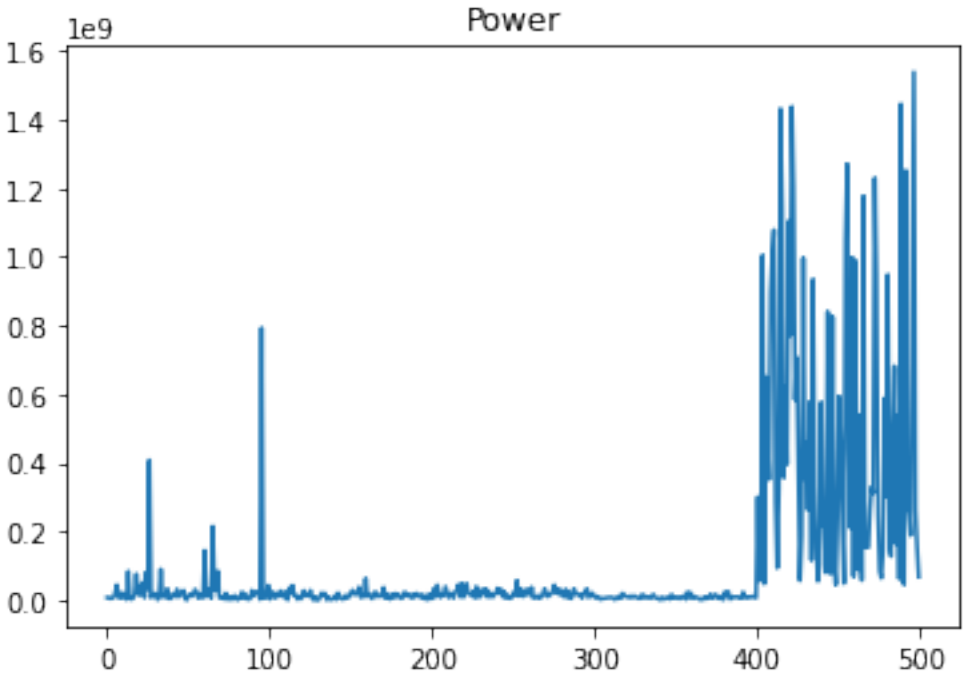
```
0.26
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\svm\_base.py:301: ConvergenceWarning: Solver terminated early
(max_iter=30000000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[525]:
```
#calculate the energy of each signal
energy = np.sum(x**2, axis=1)
print(energy.shape)
```

```
(500,)
```

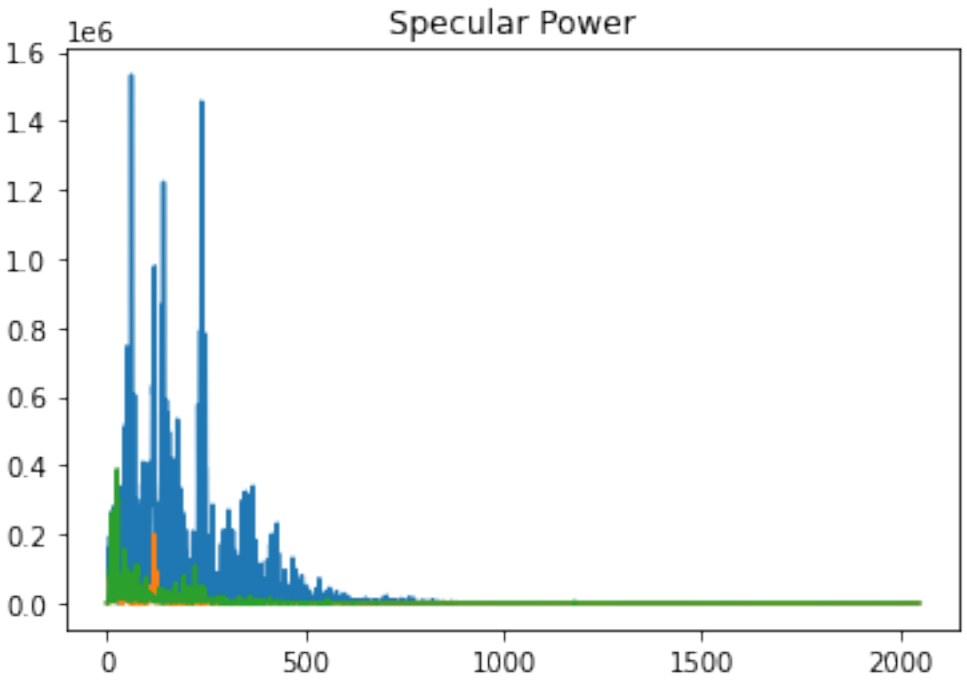
[526]:
```
plt.title('Power')
plt.plot(energy)
plt.show()

energy = energy.reshape(-1, 1)
```



[527]:
```
# calculate power spectral density for each signal
power_specular = np.array([periodogram(x[ind, :])[1] for ind in range(x.
  ↪shape[0])])
print(power_specular.shape)
```

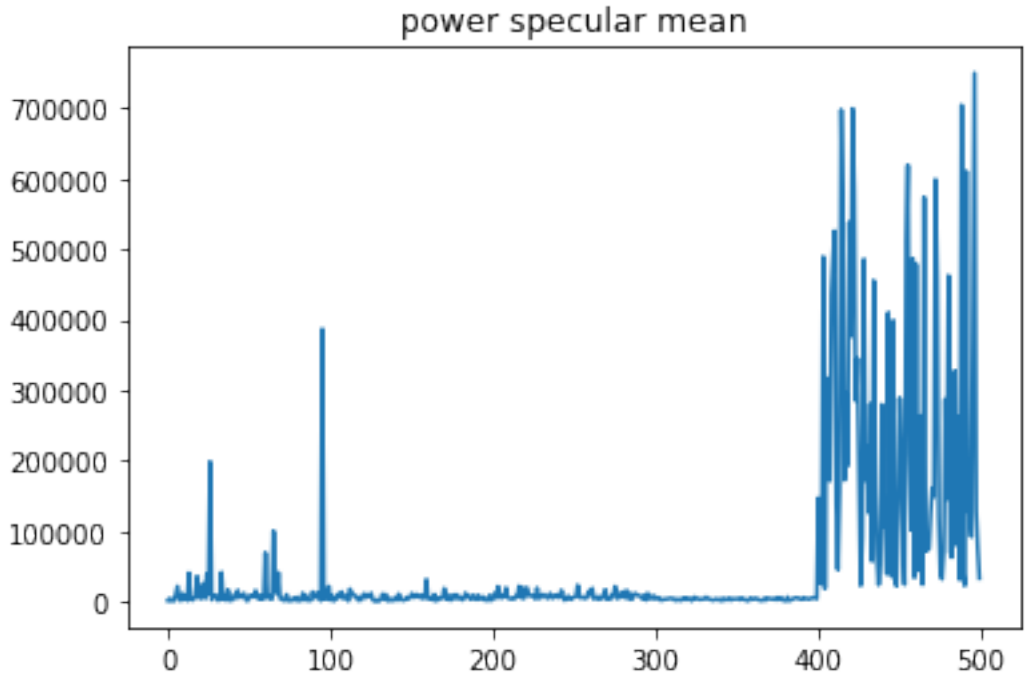
```
(500, 2049)
```

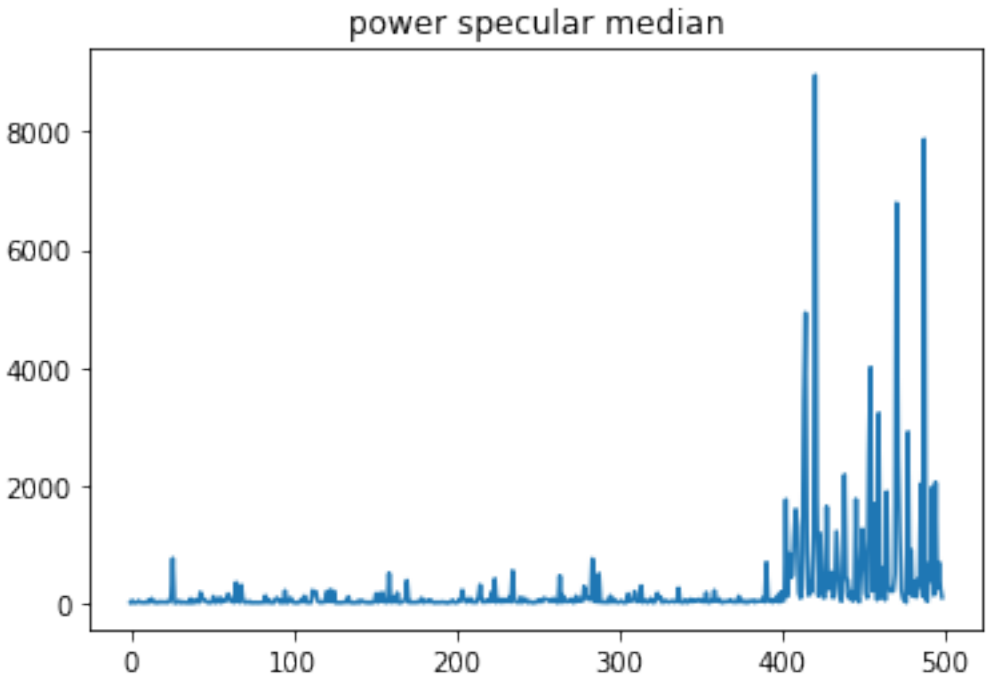
[528]:
```
plt.title('Specular Power')
plt.plot(power_specular[-1])
plt.plot(power_specular[0])
plt.plot(power_specular[1])
plt.show()
```

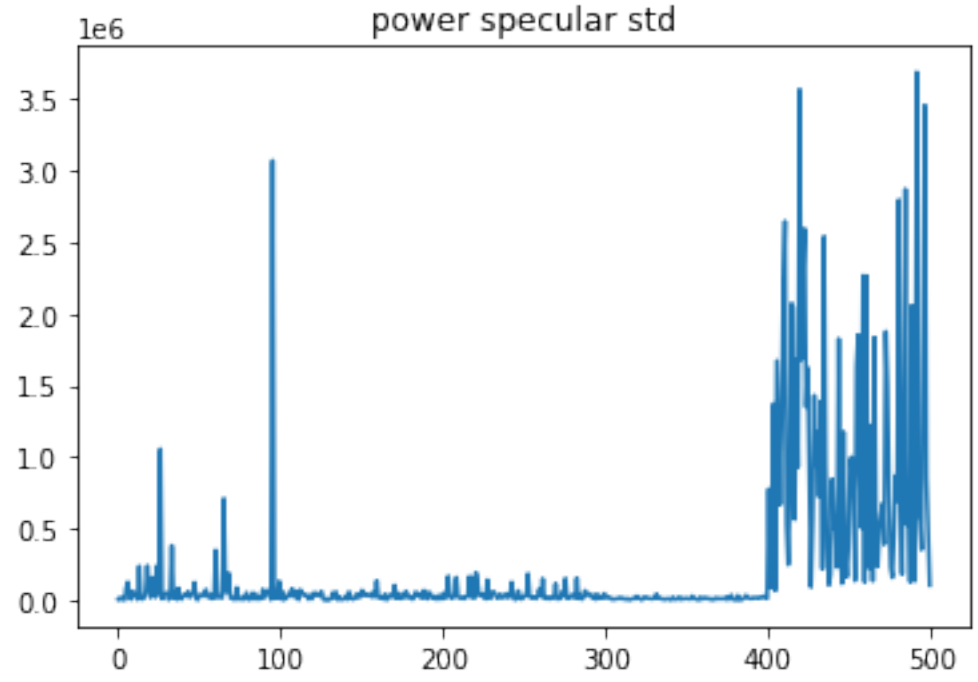


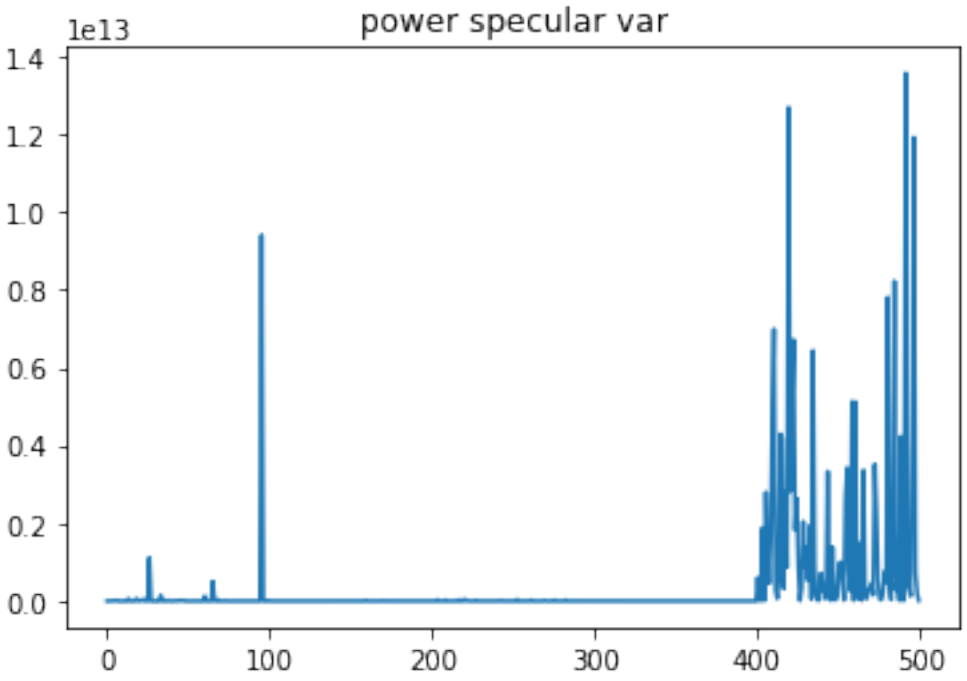
[529]:
```
power_specular_mean = np.mean(power_specular, axis=1)
power_specular_median = np.median(power_specular, axis=1)
power_specular_std = np.std(power_specular, axis=1)
power_specular_var = np.var(power_specular, axis=1)
power_specular_min = np.min(power_specular, axis=1)
power_specular_max = np.max(power_specular, axis=1)

plt.title('power specular mean')
plt.plot(power_specular_mean)
plt.show()

plt.title('power specular median')
plt.plot(power_specular_median)
plt.show()

plt.title('power specular std')
plt.plot(power_specular_std)
plt.show()

plt.title('power specular var')
plt.plot(power_specular_var)
plt.show()

plt.title('power specular min')
```

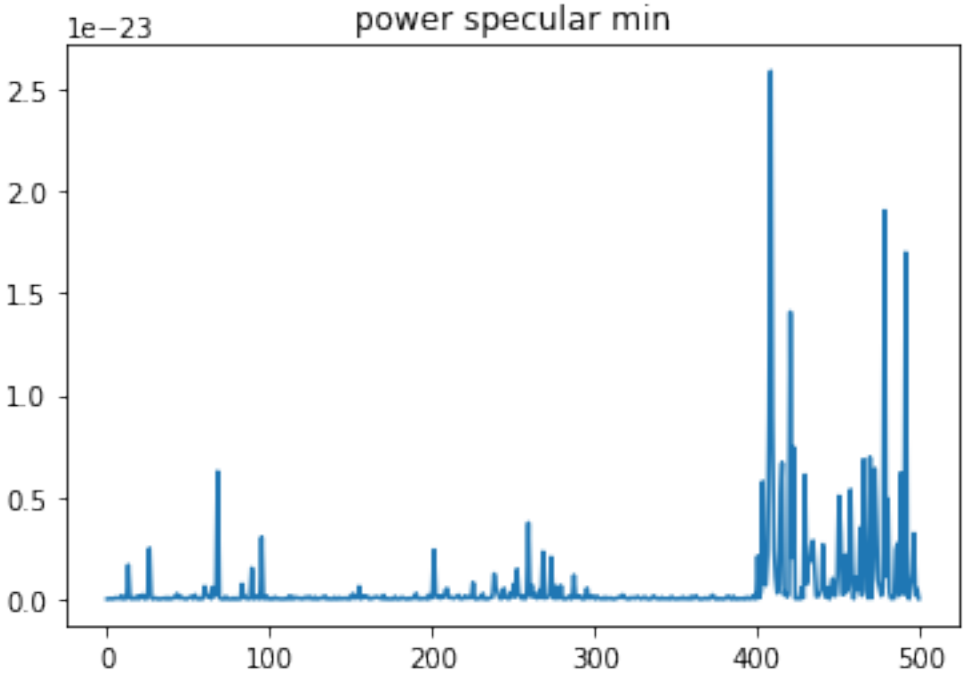
```
plt.plot(power_specular_min)
plt.show()

plt.title('power specular max')
plt.plot(power_specular_max)
plt.show()

power_specular_mean = power_specular_mean.reshape(-1, 1)
power_specular_median = power_specular_median.reshape(-1, 1)
power_specular_std = power_specular_std.reshape(-1, 1)
power_specular_var = power_specular_var.reshape(-1, 1)
power_specular_min = power_specular_min.reshape(-1, 1)
power_specular_max = power_specular_max.reshape(-1, 1)
```

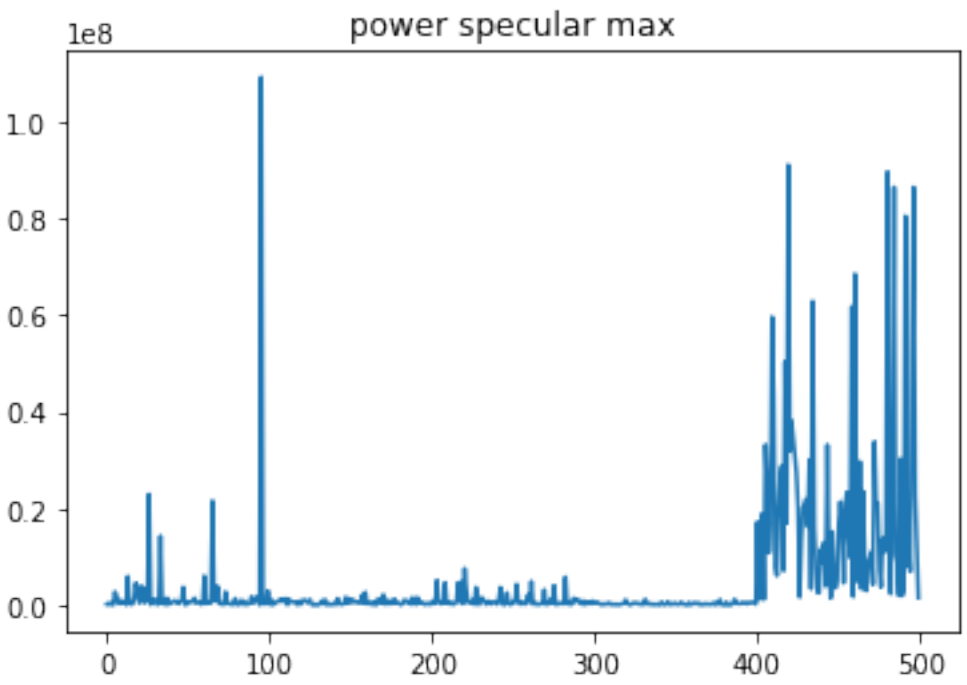



power specular median
8000
6000
4000
2000
0
0
100
200
300
400
500

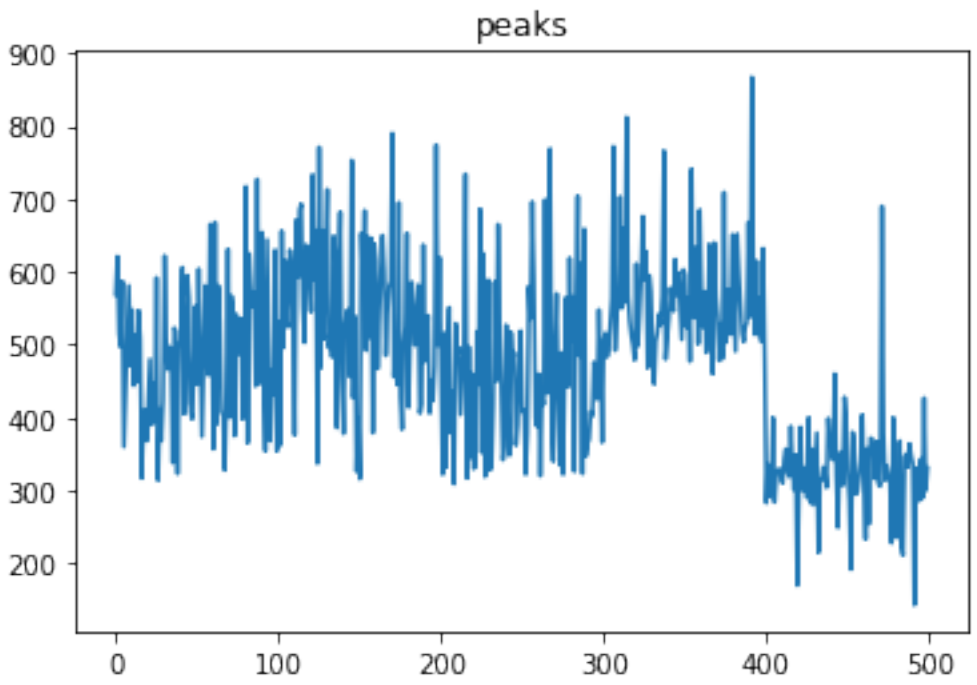

power specular std
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500


power specular var
1e13
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

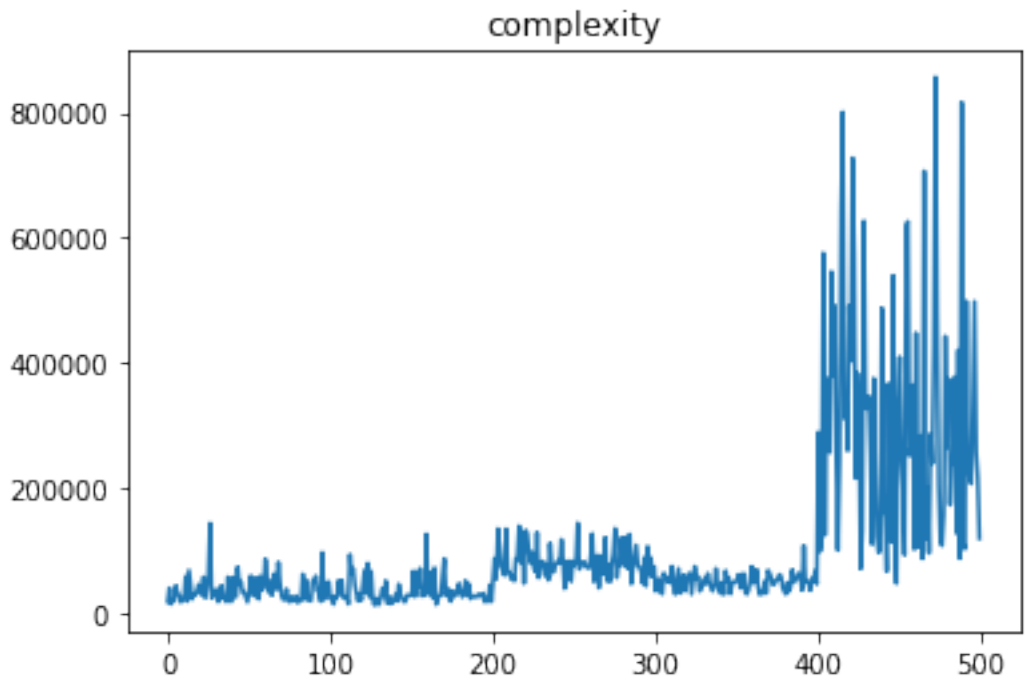

power specular min
1e−23
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500



[530]:
```
#detect peaks
peaks = np.array([find_peaks(x[i, :])[0].shape[0] for i in range(x.shape[0])])

plt.title('peaks')
plt.plot(peaks)
plt.show()

peaks = peaks.reshape(-1, 1)
```



[531]:
```
#calculate complexity
complexity = np.array([np.sum(np.abs(np.diff(x[i, :]))) for i in range(x.
 ↪shape[0])])

plt.title('complexity')
plt.plot(complexity)
plt.show()

complexity = complexity.reshape(-1, 1)
```

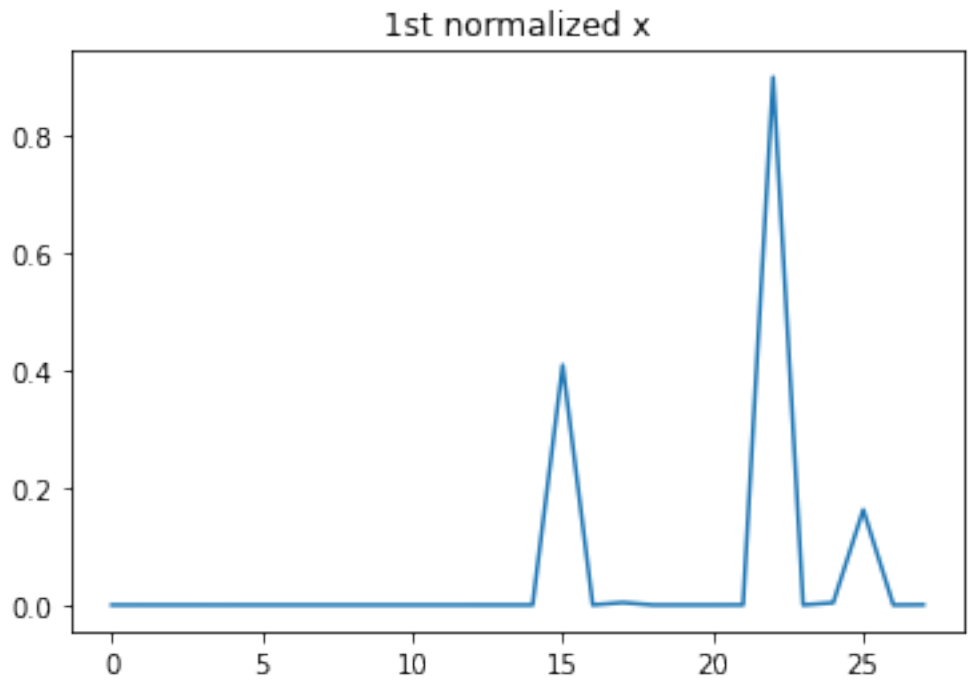


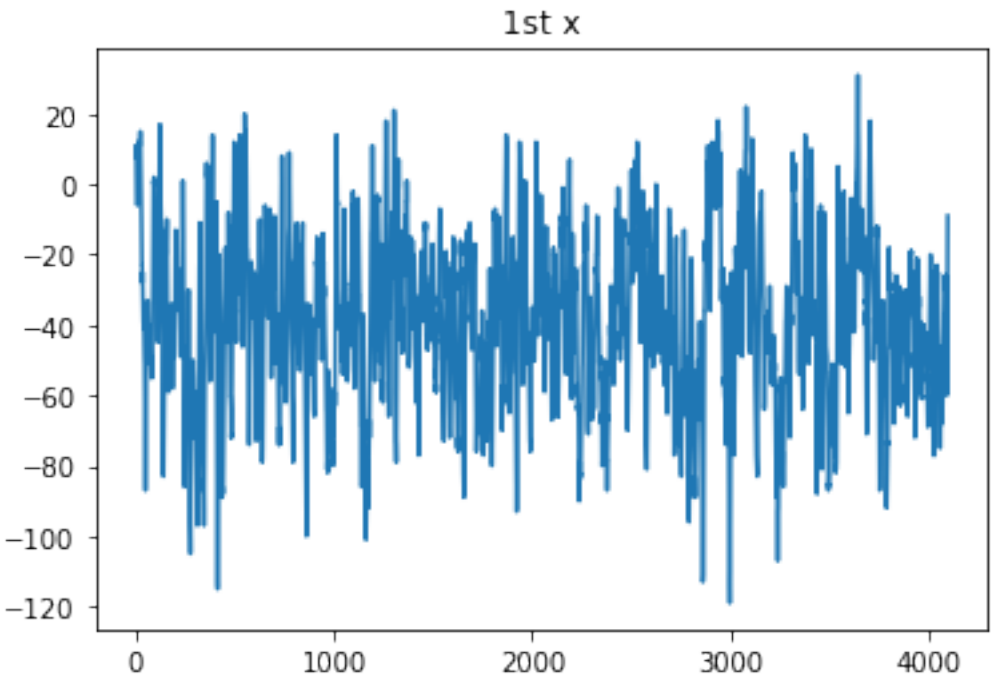
[532]:
```
new_x = np.concatenate((var,
                        mean,
                        maximum,
                        minimum,
                        median,
                        std,
                        var,
                        permutation,
                        spectral,
                        singular_value_decomposition,
                        approximate,
                        sample,
                        lempel_ziv,
                        fft_mean,
                        fft_median,
                        fft_var,
                        fft_std,
                        fft_max,
                        fft_min,
                        power_specular_mean,
                        power_specular_median,
                        power_specular_std,
                        power_specular_var,
                        power_specular_min,
```

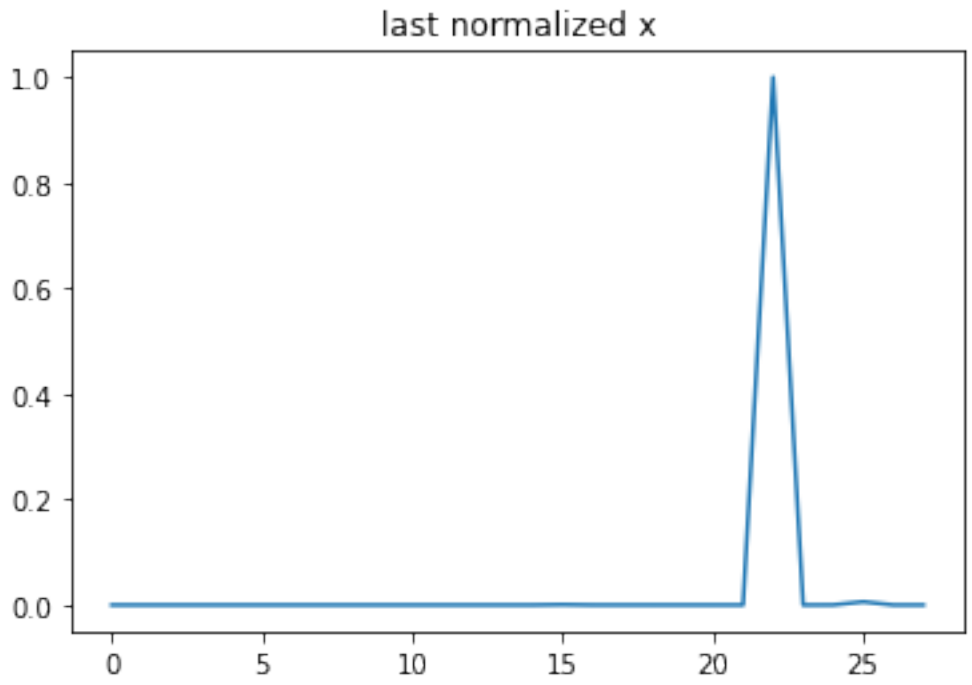
```
                        power_specular_max,
                        energy,
                        peaks,
                        complexity), axis=1)

x_train, x_test, y_train, y_test = train_test_split(new_x, y,␣
 ↪random_state=SEED, test_size=0.2)
print(new_x.shape)
print(new_x)
```

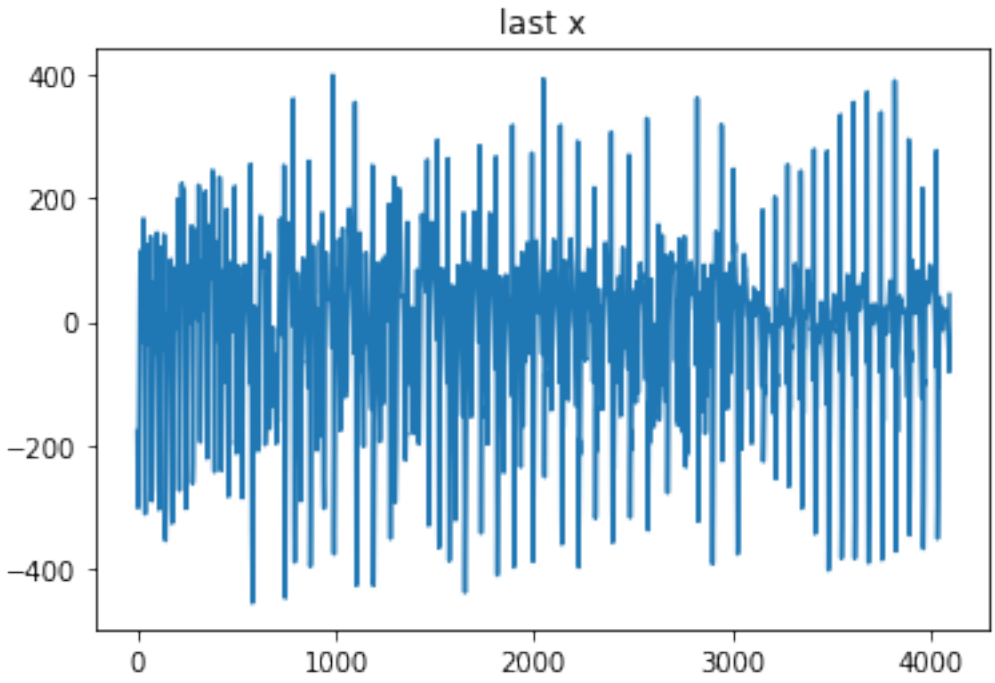
```
(500, 28)
[[ 5.85240101e+02 -3.80785941e+01  3.10000000e+01 …  8.33829400e+06
   5.69000000e+02  1.86870000e+04]
 [ 2.04747837e+03 -1.23775934e+01  1.92000000e+02 …  9.01619900e+06
   6.21000000e+02  4.03000000e+04]
 [ 5.58204044e+02  2.78664877e+01  9.20000000e+01 …  5.46845100e+06
   5.15000000e+02  1.48300000e+04]
 …
 [ 6.41967333e+04 -5.22089334e-01  6.83000000e+02 …  2.63015133e+08
   4.27000000e+02  2.57480000e+05]
 [ 3.73167473e+04 -1.50078106e+01  5.55000000e+02 …  1.53809499e+08
   3.00000000e+02  2.16483000e+05]
 [ 1.64935065e+04 -1.86465707e+01  3.99000000e+02 …  6.89984010e+07
   3.31000000e+02  1.19561000e+05]]
```

[533]:
```
normalized_x = normalize(new_x)

plt.title('1st normalized x')
plt.plot(normalized_x[0])
plt.show()

plt.title('1st x')
plt.plot(x[0])
plt.show()

plt.title('last normalized x')
plt.plot(normalized_x[-1])
plt.show()

plt.title('last x')
plt.plot(x[-1])
plt.show()
```


1st normalized x
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25


1st x
20
0
−20
−40
−60
−80
−100
−120
0
1000
2000
3000
4000


last normalized x
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25


last x
400
200
0
−200
−400
0
1000
2000
3000
4000

0.4 RANDOM FOREST CLASSIFIER

- n_estimators: The number of trees in the forest.
- max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_split: The minimum number of samples required to split an internal node: If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- random_state: Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features < n_features). See Glossary for details.

[534]:
```
clf = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)
clf.fit(x_train, y_train)
y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
print(recall_score(y_test, y_prediction))
print(precision_score(y_test, y_prediction))
```

```
0.98
0.9230769230769231
1.0
```

```
C:\Users\SQ-PC\AppData\Local\Temp\ipykernel_6596\1168402667.py:2:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  clf.fit(x_train, y_train)
```

[535]:
```
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'recall': make_scorer(recall_score),
    'precision': make_scorer(precision_score),
    'f1': make_scorer(f1_score),
}

cv = KFold(n_splits=5, random_state=SEED, shuffle=True)
cross_validate = cross_validate(estimator=clf, X=new_x, y=y, cv=cv)
cross_validate
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

[535]:
```
{'fit_time': array([0.10861278, 0.12804103, 0.11365056, 0.10465026,
0.10864949]),
 'score_time': array([0.0080092 , 0.01299882, 0.0079565 , 0.00800133,
0.00700212]),
 'test_score': array([0.98, 0.95, 1.  , 0.98, 0.97])}
```

[536]:
```
scores = cross_val_score(clf, new_x, y, cv=5)
print("k-fold cross validation accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),␣
  ↪scores.std() * 2))
```

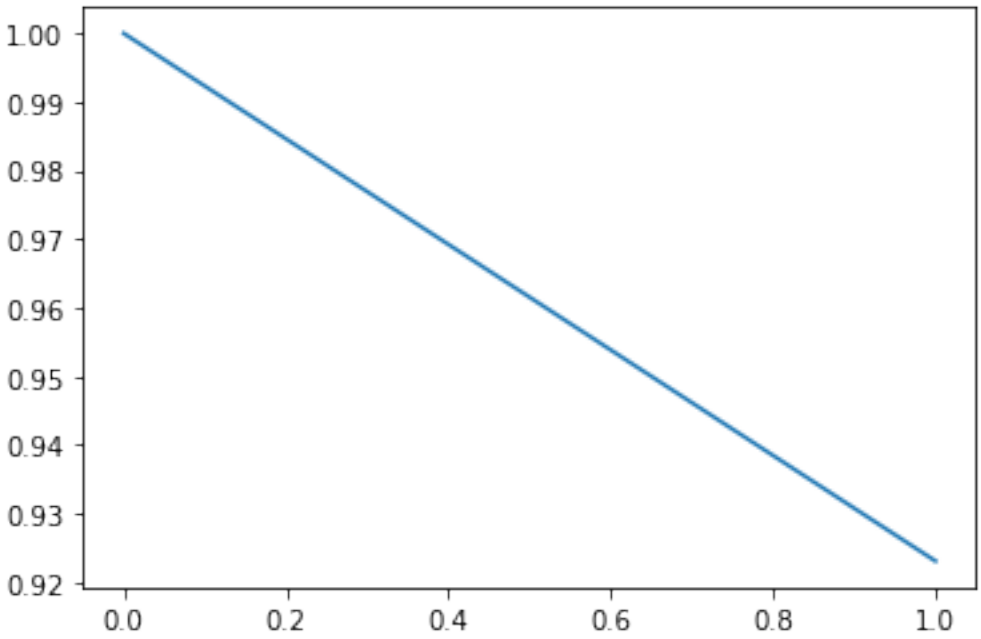
```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
```

```
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

```
k-fold cross validation accuracy: 0.97 (+/- 0.07)
```

[537]:
```
#recall plot
plt.plot(recall_score(y_test, y_prediction, average=None))
```

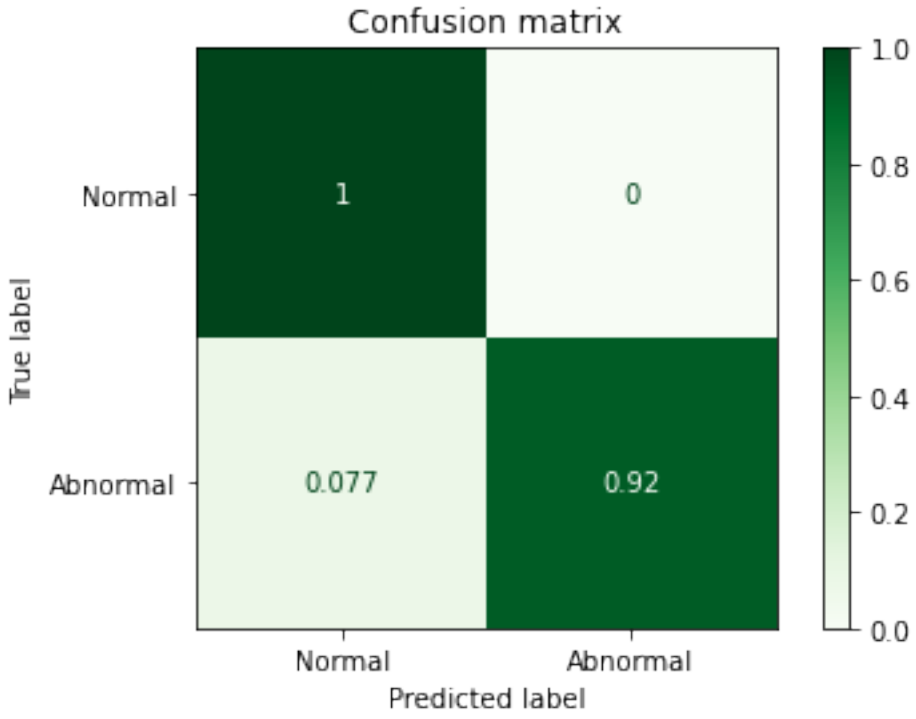
[537]:
```
[<matplotlib.lines.Line2D at 0x1bd06e05220>]
```



[538]:
```
#confusion matrix plot
display = plot_confusion_matrix(clf, x_test, y_test, display_labels=['Normal',␣
 ↪'Abnormal'], cmap=plt.cm.Greens, normalize='true')
display.ax_.set_title('Confusion matrix')

plt.show()

print(confusion_matrix(y_test, y_prediction))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\deprecation.py:87: FutureWarning: Function
```

```
plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
[[74  0]
 [ 2 24]]
```

0.5 K Nearest Neighbors Classifier

n_neighbors: n_neighbors is the number of neighbors that we are going to use to classify our data which here is 2 - weights: weights is the weight function used in prediction. possible values are: - uniform: uniform weights. all points in each neighborhood are weighted equally. - distance : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

[539]:
```
clf = KNeighborsClassifier(n_neighbors=2,
                           weights='distance',
                           algorithm='auto',
                           leaf_size=30,
                           p=2,
                           metric='minkowski',
                           metric_params=None,
```

```
                                n_jobs=None)
clf.fit(x_train, y_train)
y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
print(recall_score(y_test, y_prediction))
print(precision_score(y_test, y_prediction))
```

```
0.93
0.7307692307692307
1.0
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

[540]:
```
# scoring = {
#     'accuracy': make_scorer(accuracy_score),
#     'recall': make_scorer(recall_score),
#     'precision': make_scorer(precision_score),
#     'f1': make_scorer(f1_score),
# }

# cv = KFold(n_splits=5, random_state=SEED, shuffle=True)
# cross_validate = cross_validate(estimator=clf, X=new_x, y=y, cv=cv)
# cross_validate
```

[541]:
```
scores = cross_val_score(clf, new_x, y, cv=5)
print("k-fold cross validation accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),␣
 ↪scores.std() * 2))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

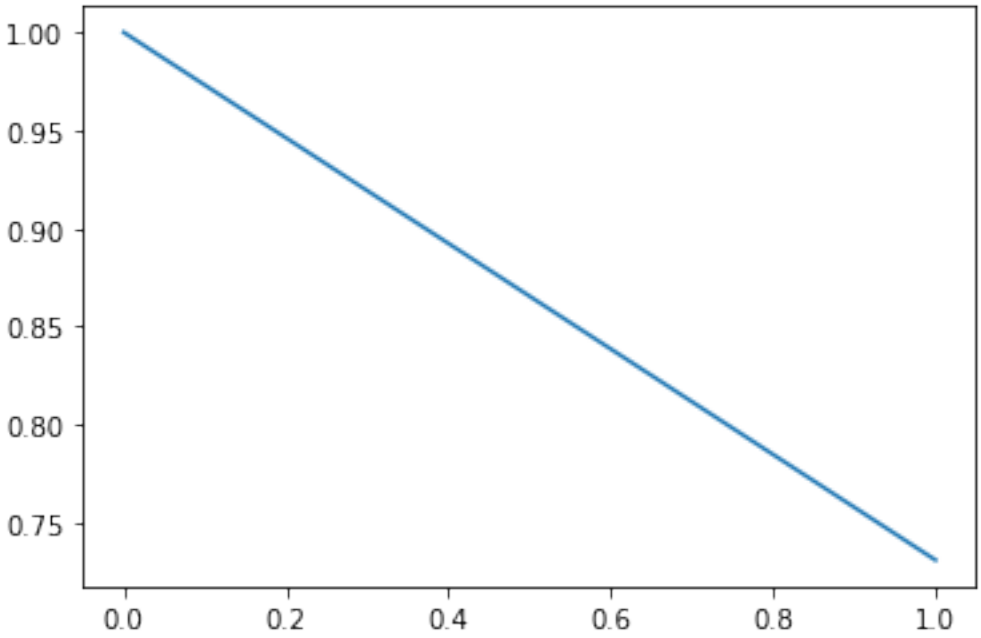
```
k-fold cross validation accuracy: 0.93 (+/- 0.08)
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
```

```
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neighbors\_classification.py:200: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

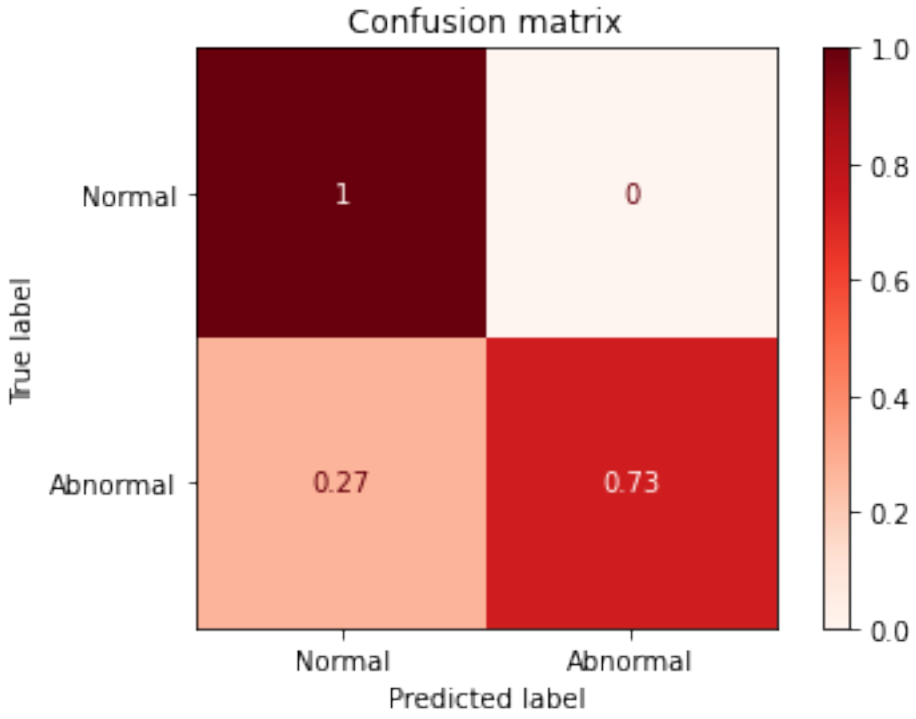
[542]:
```
#recall plot
plt.plot(recall_score(y_test, y_prediction, average=None))
```

[542]:
```
[<matplotlib.lines.Line2D at 0x1bd1119b430>]
```



[543]:
```
#confusion matrix plot
display = plot_confusion_matrix(clf, x_test, y_test, display_labels=['Normal',␣
 ↪'Abnormal'], cmap=plt.cm.Reds, normalize='true')
display.ax_.set_title('Confusion matrix')

plt.show()

print(confusion_matrix(y_test, y_prediction))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\deprecation.py:87: FutureWarning: Function
plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
[[74  0]
 [ 7 19]]
```

0.6 SVM

kernel: kernel type to be used in the algorithm. it must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. if none is given, 'rbf' will be used. if a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

[544]:
```
#make svc faster
clf = SVC(kernel='rbf', max_iter=2000000000, C=10, random_state=0)
clf.fit(x_train, y_train)
y_prediction = clf.predict(x_test)

print(accuracy_score(y_test, y_prediction))
```

```python
print(recall_score(y_test, y_prediction))
print(precision_score(y_test, y_prediction))
```

```
0.92
0.6923076923076923
1.0
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[545]:

```python
# scoring = {
#     'accuracy': make_scorer(accuracy_score),
#     'recall': make_scorer(recall_score),
#     'precision': make_scorer(precision_score),
#     'f1': make_scorer(f1_score),
# }

# cv = KFold(n_splits=5, random_state=SEED, shuffle=True)
# cross_validate = cross_validate(estimator=clf, X=new_x, y=y, cv=cv)
# cross_validate
```

[546]:

```python
scores = cross_val_score(clf, new_x, y, cv=5)
print("k-fold cross validation accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),␣
 ↪scores.std() * 2))
```

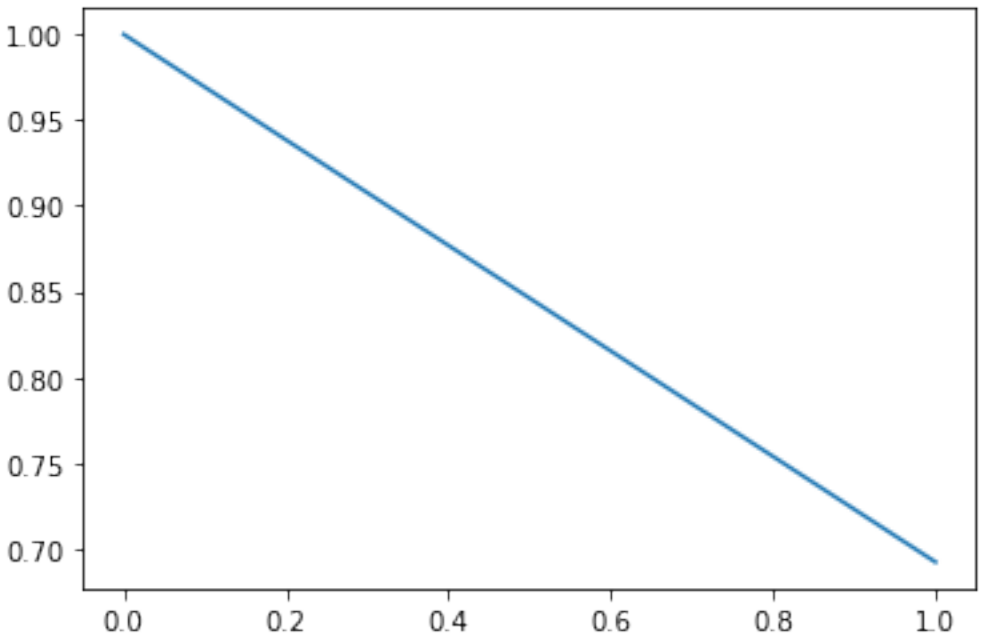
```
k-fold cross validation accuracy: 0.94 (+/- 0.02)
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
```

```
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[547]:
```
#recall plot
plt.plot(recall_score(y_test, y_prediction, average=None))
```

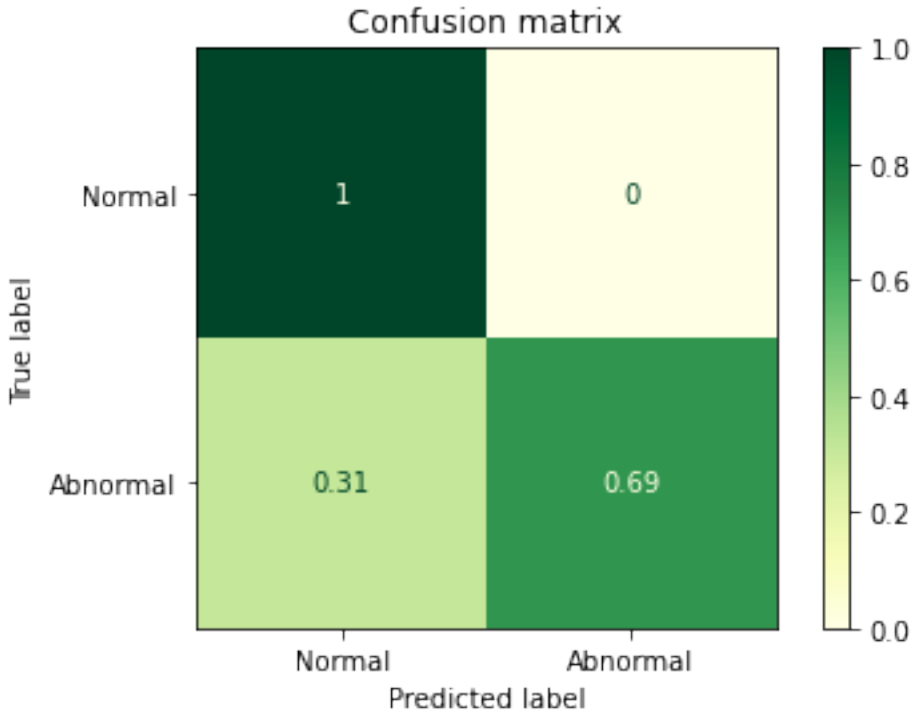
[547]:
```
[<matplotlib.lines.Line2D at 0x1bd0a089490>]
```



[548]:
```
#confusion matrix plot
display = plot_confusion_matrix(clf, x_test, y_test, display_labels=['Normal',␣
 ↪'Abnormal'], cmap=plt.cm.YlGn, normalize='true')
display.ax_.set_title('Confusion matrix')

plt.show()

print(confusion_matrix(y_test, y_prediction))
```

```
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\deprecation.py:87: FutureWarning: Function
plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
```

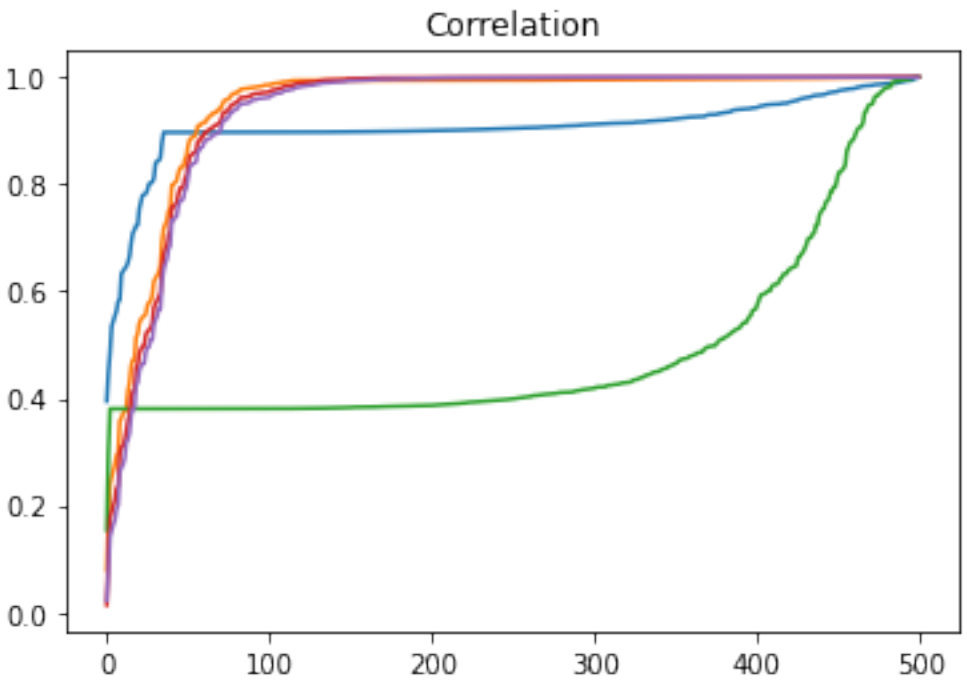
```
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
[[74  0]
 [ 8 18]]
```

[549]:
```
#calculate correlation of each feature and compare it with other features
correlation = np.corrcoef(new_x)

#compare each feature correlation with other features correlation and sort them␣
 ↪to get the most correlated features
correlation = np.abs(correlation)
correlation = np.sort(correlation, axis=1)
```

[550]:
```
# ploy random correlations after sorting

plt.title("Correlation")

for i in range(0, correlation.shape[1], 100):
    plt.plot(correlation[i])

plt.show()
```



[551]:
```
the_best_features = []
BEST_SCORE = 0.7

for i in range(0, new_x.shape[1]):
    x = new_x[:,i].reshape(-1, 1);
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=SEED)

    clf = DecisionTreeClassifier()
    clf.fit(x_train, y_train)

    y_prediction = clf.predict(x_test)

    model_accuracy = accuracy_score(y_test, y_prediction)
    model_recall = recall_score(y_test, y_prediction)
    model_precision = precision_score(y_test, y_prediction)

    print(f'feature #{i} with accuracy: {model_accuracy}, recall:␣
 ↪{model_recall}, precision: {model_precision}')

    if model_accuracy > BEST_SCORE and model_recall > BEST_SCORE and␣
 ↪model_precision > BEST_SCORE:
        the_best_features.append(i)
```

```
feature #0 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
```

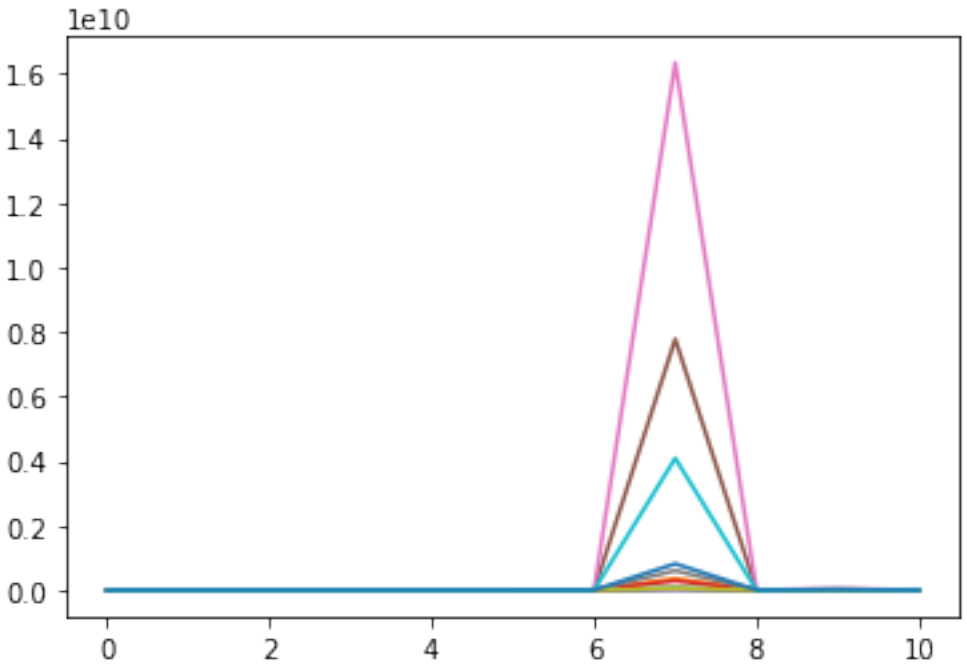
```
feature #1 with accuracy: 0.72, recall: 0.34615384615384615, precision: 0.45
feature #2 with accuracy: 0.92, recall: 0.7692307692307693, precision:
0.9090909090909091
feature #3 with accuracy: 0.91, recall: 0.7307692307692307, precision:
0.9047619047619048
feature #4 with accuracy: 0.89, recall: 0.6153846153846154, precision:
0.9411764705882353
feature #5 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
feature #6 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
feature #7 with accuracy: 0.84, recall: 0.5384615384615384, precision:
0.7777777777777778
feature #8 with accuracy: 0.7, recall: 0.46153846153846156, precision:
0.42857142857142855
feature #9 with accuracy: 0.71, recall: 0.23076923076923078, precision: 0.4
feature #10 with accuracy: 0.7, recall: 0.3076923076923077, precision: 0.4
feature #11 with accuracy: 0.77, recall: 0.5384615384615384, precision: 0.56
feature #12 with accuracy: 0.74, recall: 0.0, precision: 0.0
feature #13 with accuracy: 0.66, recall: 0.19230769230769232, precision:
0.2777777777777778
feature #14 with accuracy: 0.65, recall: 0.2692307692307692, precision:
0.30434782608695654
feature #15 with accuracy: 0.74, recall: 0.2692307692307692, precision: 0.5
feature #16 with accuracy: 0.74, recall: 0.2692307692307692, precision: 0.5
feature #17 with accuracy: 0.62, recall: 0.15384615384615385, precision: 0.2
feature #18 with accuracy: 0.66, recall: 0.19230769230769232, precision:
0.2777777777777778
feature #19 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
feature #20 with accuracy: 0.83, recall: 0.6153846153846154, precision:
0.6956521739130435
feature #21 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
feature #22 with accuracy: 0.93, recall: 0.7307692307692307, precision: 1.0
feature #23 with accuracy: 0.74, recall: 0.0, precision: 0.0
feature #24 with accuracy: 0.93, recall: 0.8076923076923077, precision:
0.9130434782608695
feature #25 with accuracy: 0.95, recall: 0.8076923076923077, precision: 1.0
feature #26 with accuracy: 0.79, recall: 0.46153846153846156, precision:
0.631578947368421
feature #27 with accuracy: 0.92, recall: 0.8076923076923077, precision: 0.875

c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\SQ-PC\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
```

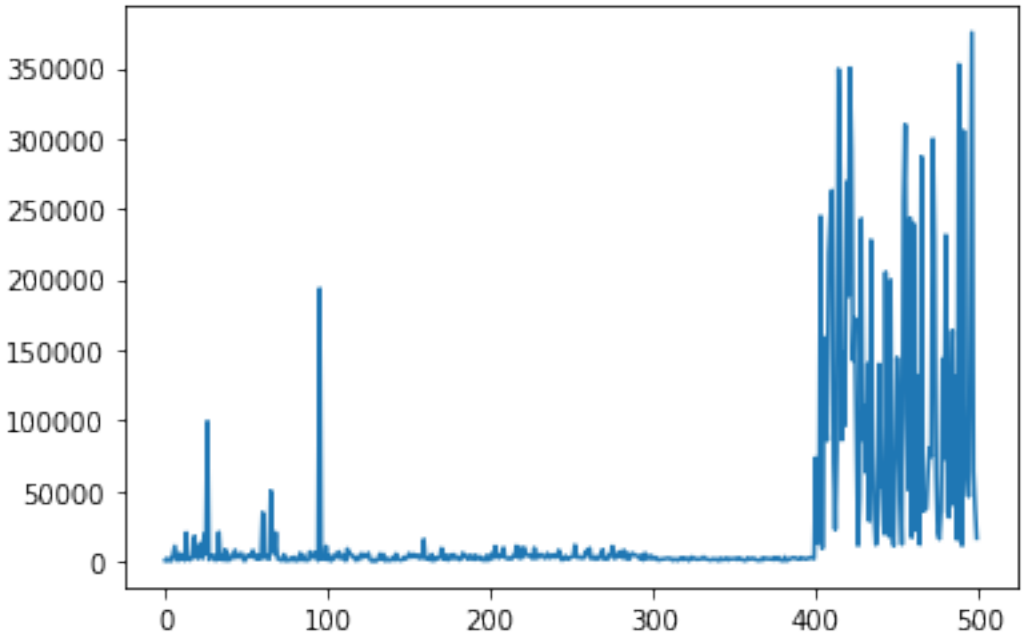
[552]:
```
optimized_x = new_x[:, the_best_features]
print(optimized_x.shape)
```

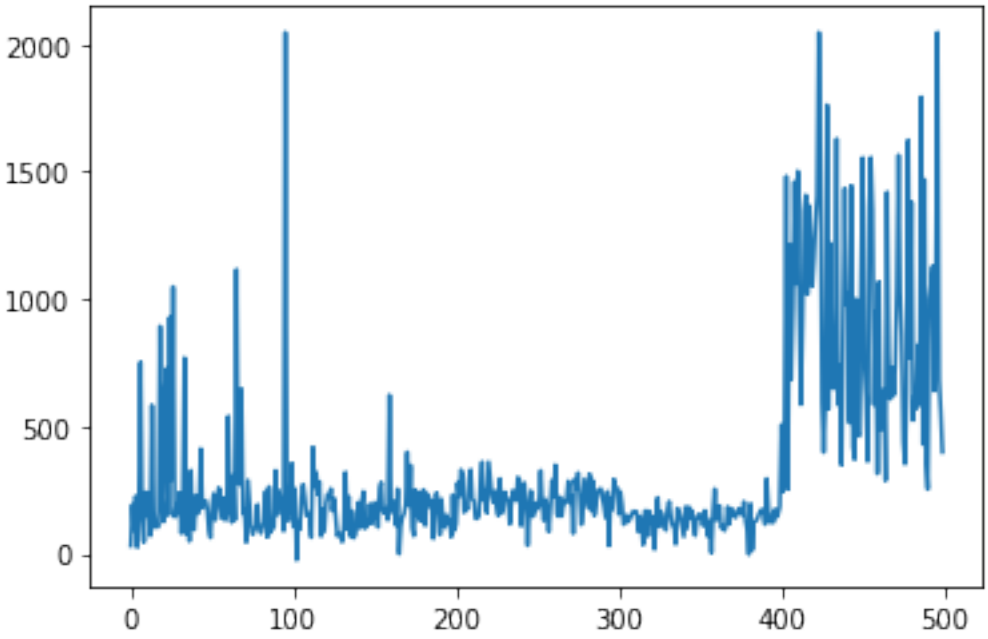
```
(500, 11)
```

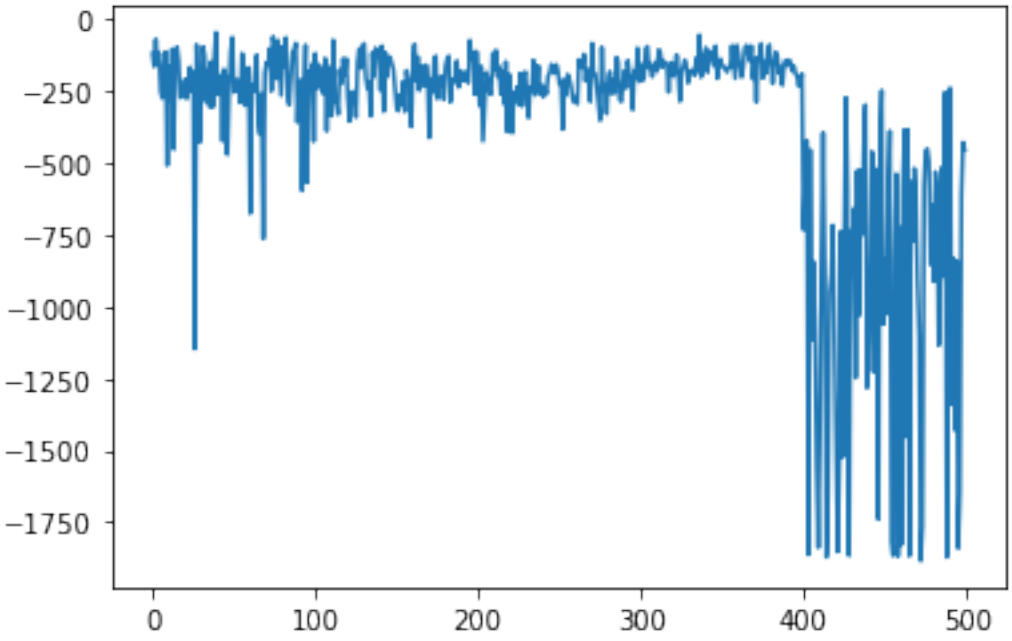
[553]:
```
for i in range(optimized_x.shape[1]):
    plt.plot(optimized_x[i])

plt.show()
```

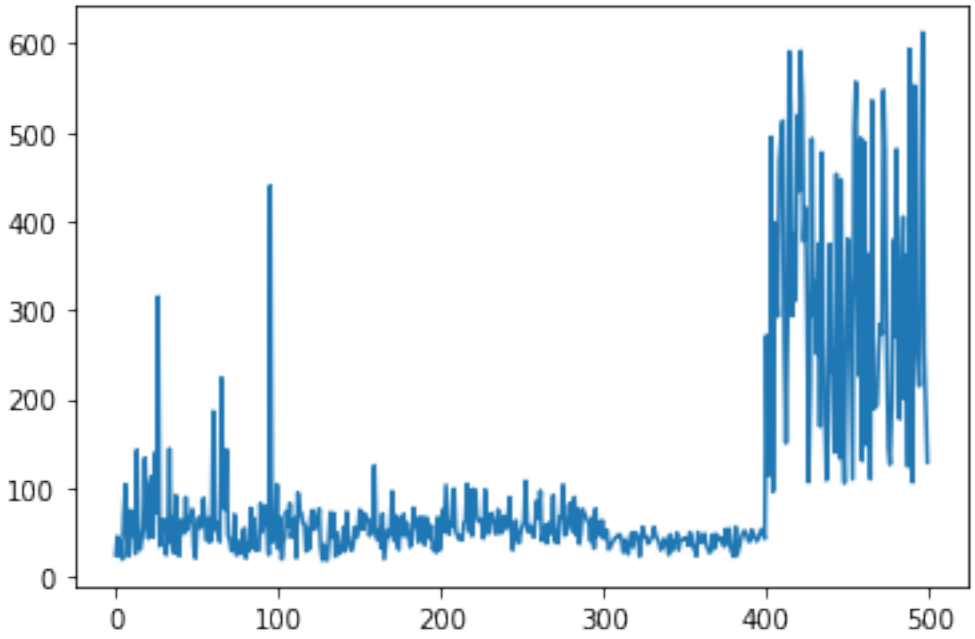


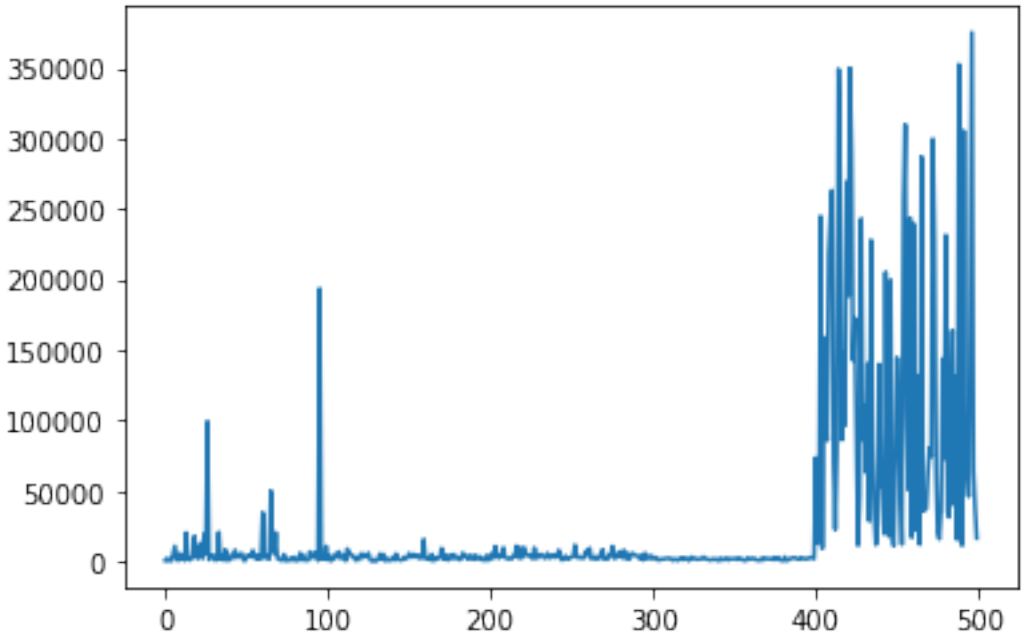
[554]:
```
for feature in optimized_x.T:
    plt.plot(feature)
    plt.show()
```

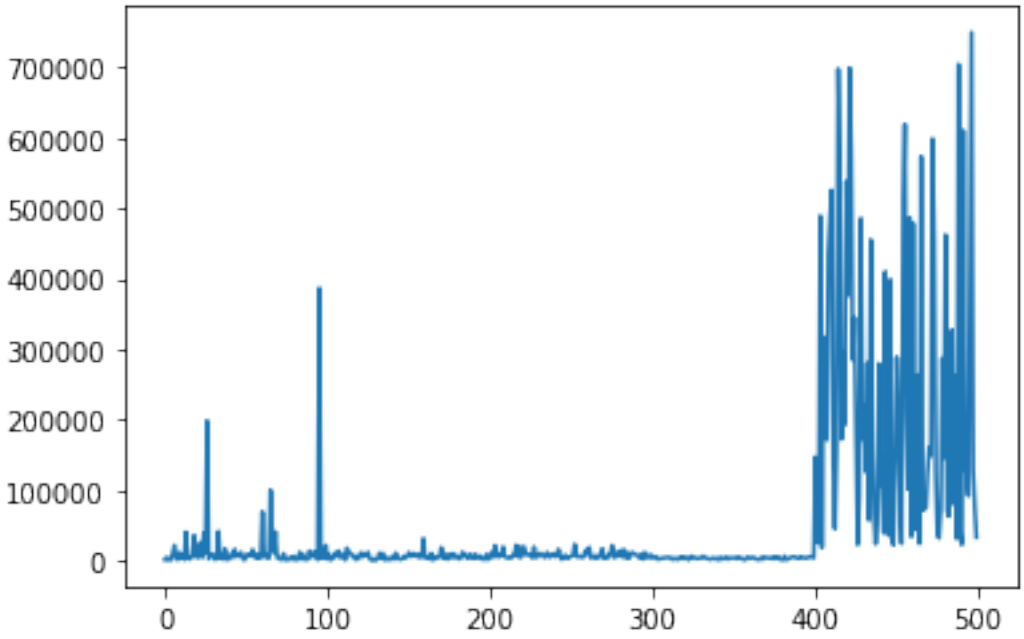

350000
300000
250000
200000
150000
100000
50000
0
0
100
200
300
400
500

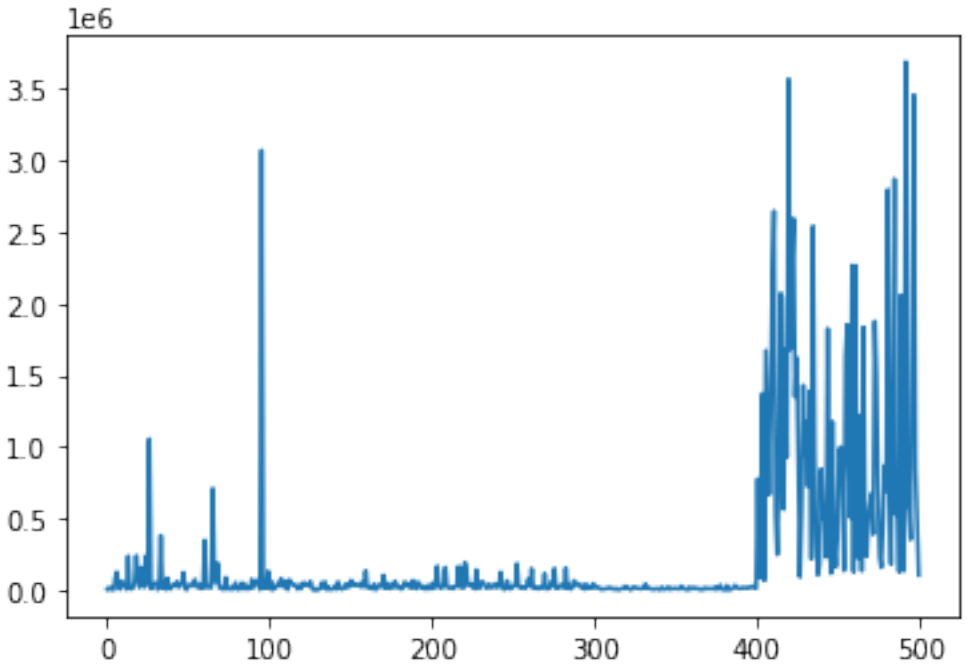

2000
1500
1000
500
0
0
100
200
300
400
500

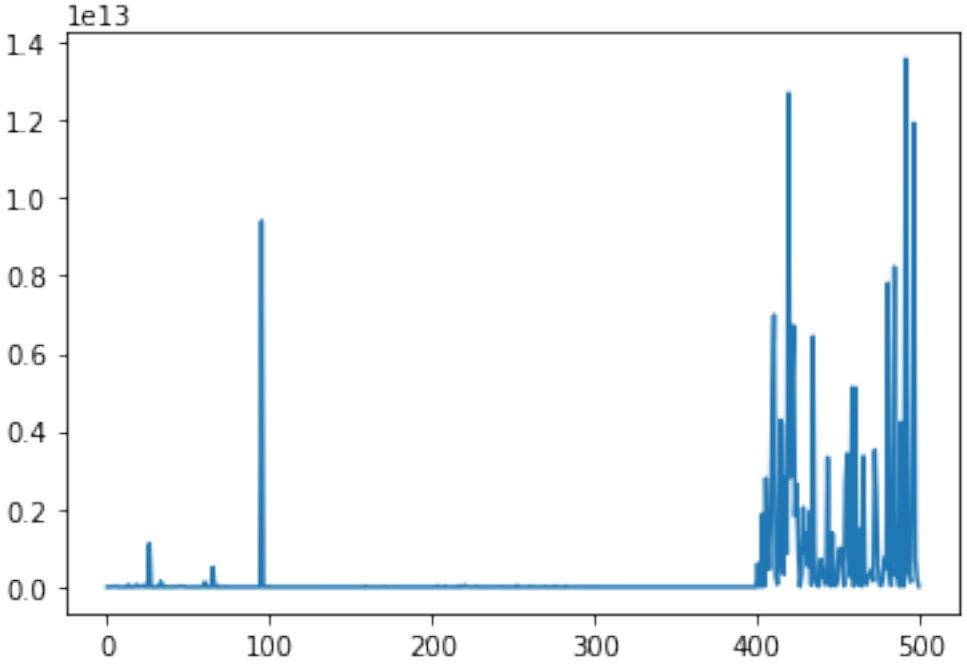

0
−250
−500
−750
−1000
−1250
−1500
−1750
0
100
200
300
400
500

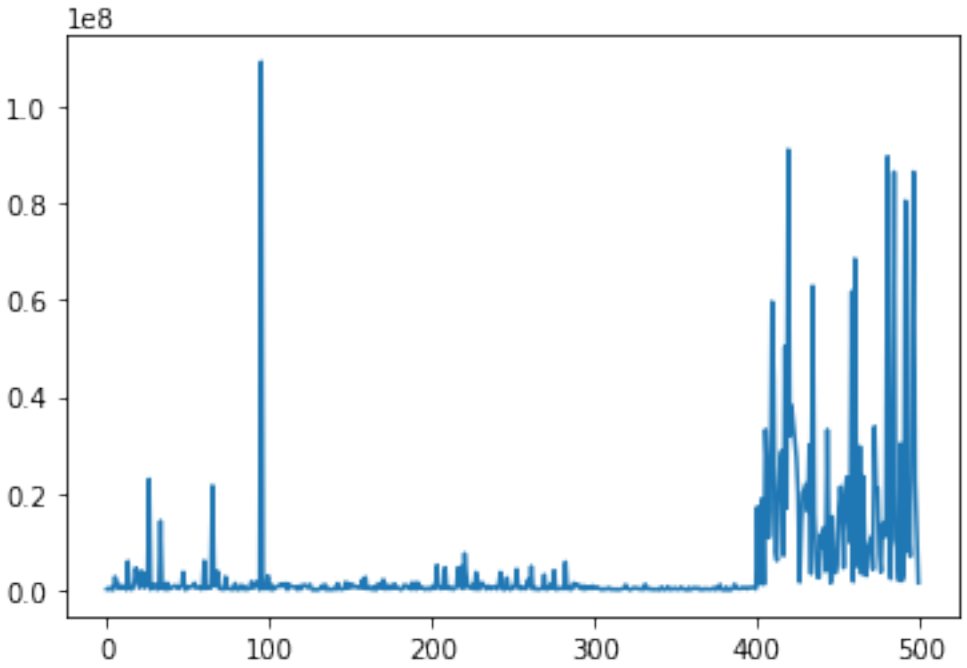

600
500
400
300
200
100
0
0
100
200
300
400
500

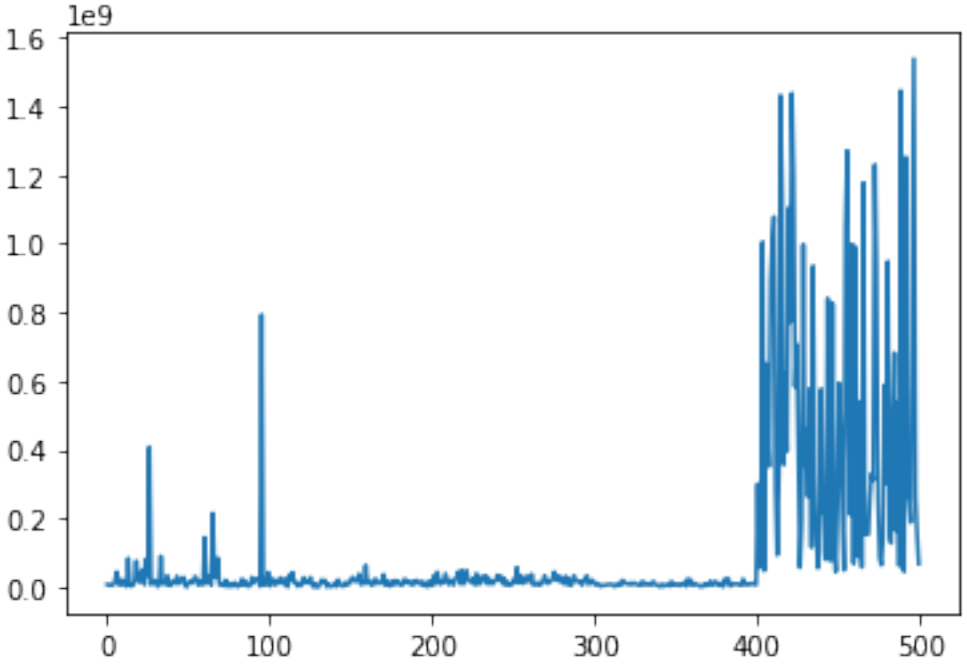

350000
300000
250000
200000
150000
100000
50000
0
0
100
200
300
400
500

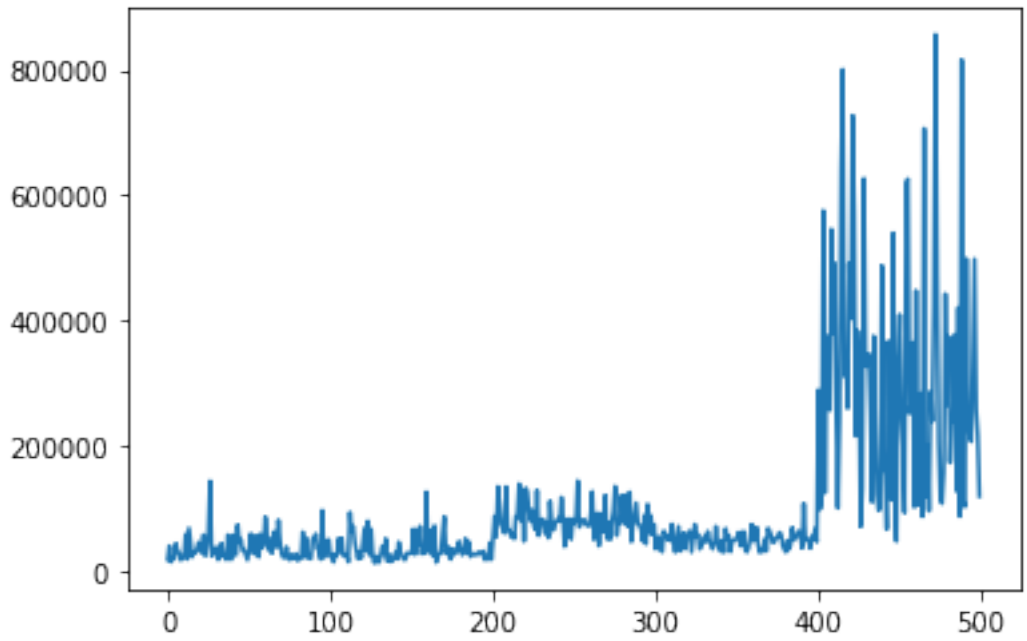

700000
600000
500000
400000
300000
200000
100000
0
0
100
200
300
400
500


1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500


1e13
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


1e8
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500



[555]:
```
# def calculate_best_features(dataset):
#     the_best_features = []
#     old_x = dataset
#     for i in range(0, old_x.shape[1]):
#         features_without_i = np.delete(new_x, i, axis=1)
#         features_with_i = dataset[:,i].reshape(-1, 1)

#         x_train, x_test, y_train, y_test = train_test_split(features_without_i, y, test_size=0.2, random_state=SEED)

#         clf = DecisionTreeClassifier(random_state=0)
#         clf.fit(x_train, y_train)
#         y_prediction = clf.predict(x_test)

#         model_accuracy_without_i = accuracy_score(y_test, y_prediction)
#         model_recall_without_i = recall_score(y_test, y_prediction)
#         model_precision_without_i = precision_score(y_test, y_prediction)

#         print('feature with out', i, '(accuracy, recall, precision):', model_accuracy_without_i, model_recall_without_i, model_precision_without_i)

#         x_train, x_test, y_train, y_test = train_test_split(features_with_i, y, test_size=0.2, random_state=SEED)

#         clf = DecisionTreeClassifier(random_state=0)
```

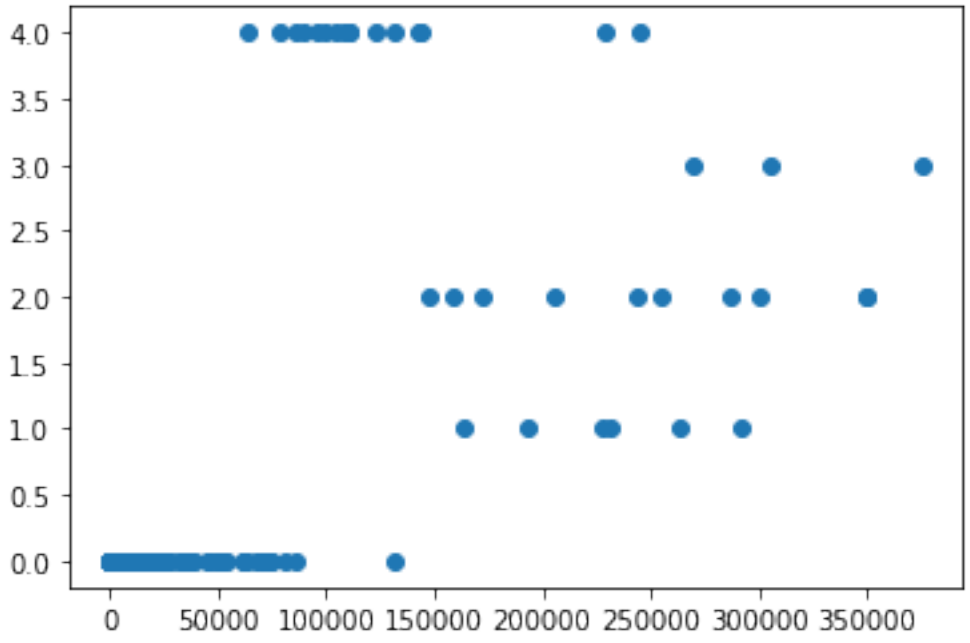
```
#         clf.fit(x_train, y_train)
#         y_prediction = clf.predict(x_test)

#         model_accuracy_with_i = accuracy_score(y_test, y_prediction)
#         model_recall_with_i = recall_score(y_test, y_prediction)
#         model_precision_with_i = precision_score(y_test, y_prediction)

#         print('feature with', i, '(accuracy, recall, precision):',␣
 ↪model_accuracy_with_i, model_recall_with_i, model_precision_with_i)

#         # if model_accuracy_with_i > model_accuracy_without_i and␣
 ↪model_recall_with_i > model_recall_without_i and model_precision_with_i >␣
 ↪model_precision_without_i:
#         #     the_best_features.append(i)

#     return the_best_features
```

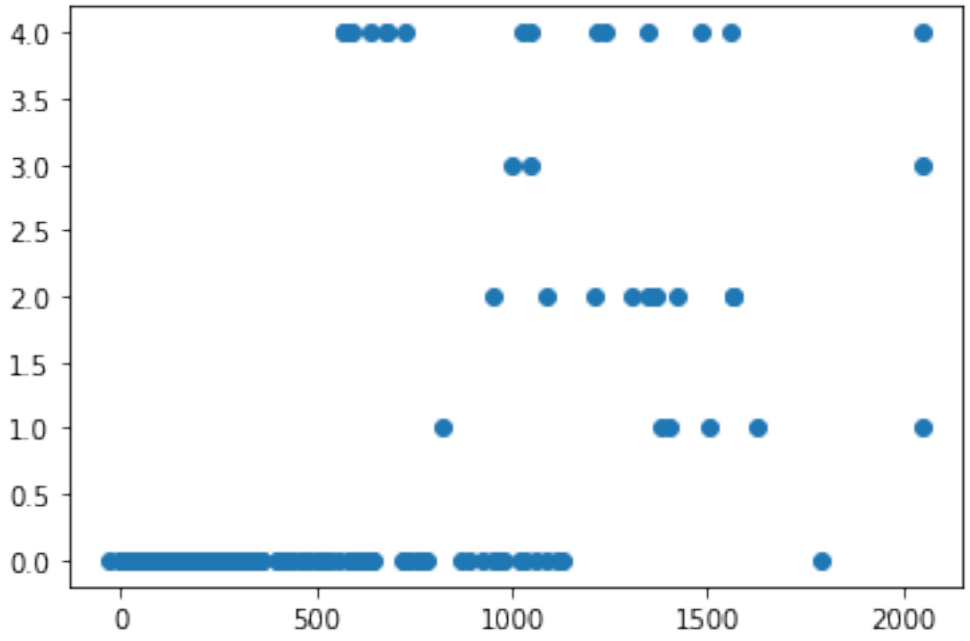
[556]:
```
# while True:
#     final_features = calculate_best_features(optimized_x)
#     if len(final_features) == 0:
#         optimized_x = np.delete(optimized_x, 0, axis=1)
#         if optimized_x.shape[1] == 0:
#             break
#         continue
#     else:
#         optimized_x = new_x[:,final_features]
#         break
```

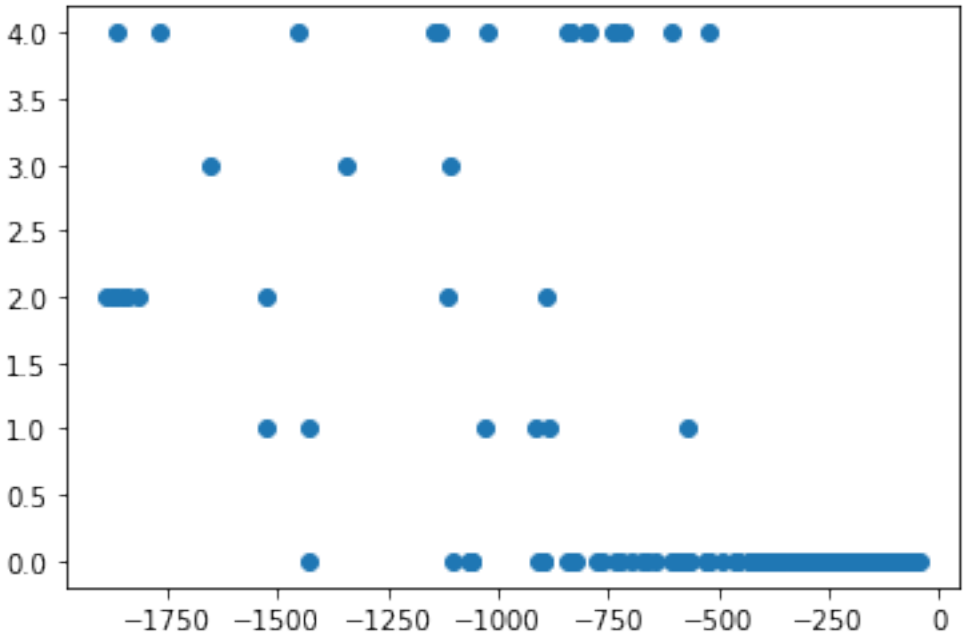
[561]:
```
#cluster data
x_train, x_test, y_train, y_test = train_test_split(optimized_x, y, test_size=0.
 ↪2, random_state=SEED, shuffle=True)
kmeans = KMeans(n_clusters=5, random_state=0).fit(x_train)
print(np.bincount(kmeans.labels_))
```

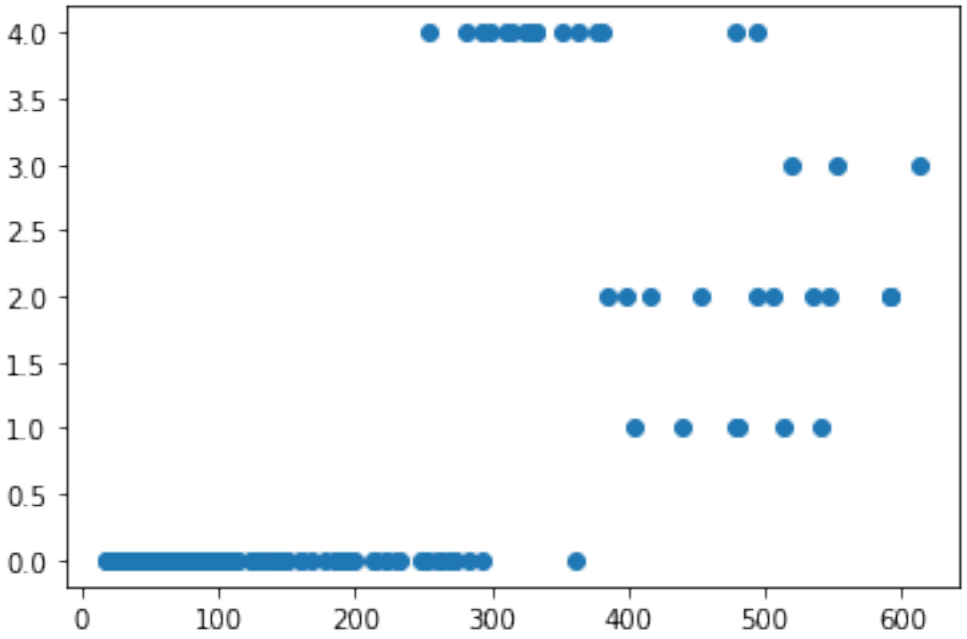
```
[365   6  10   3  16]
```

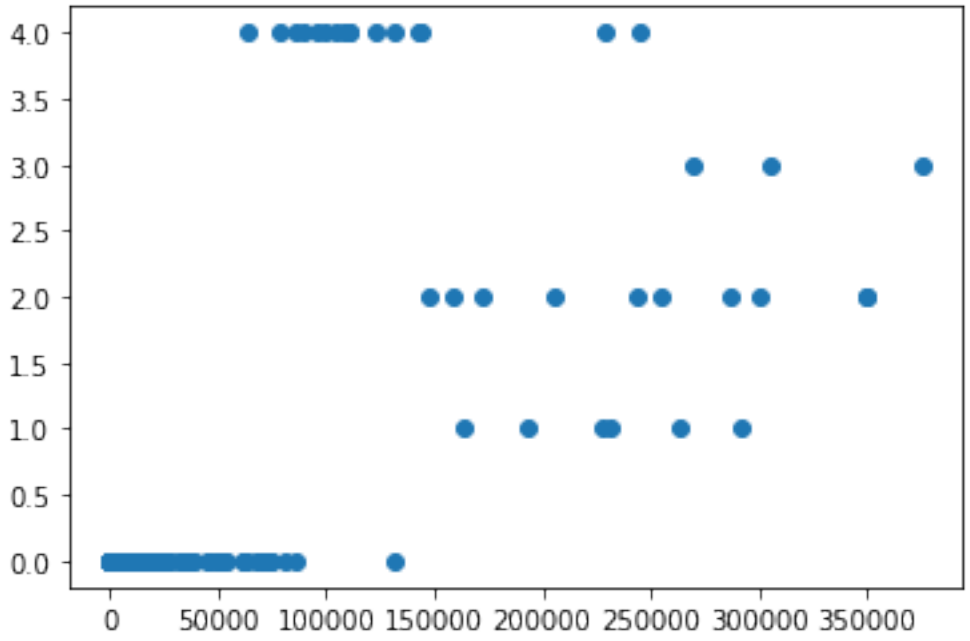
[562]:
```
#plot clusters
for i in range(0, x_train.shape[1]):
    plt.scatter(x_train[:,i], kmeans.labels_)
    plt.show()
```

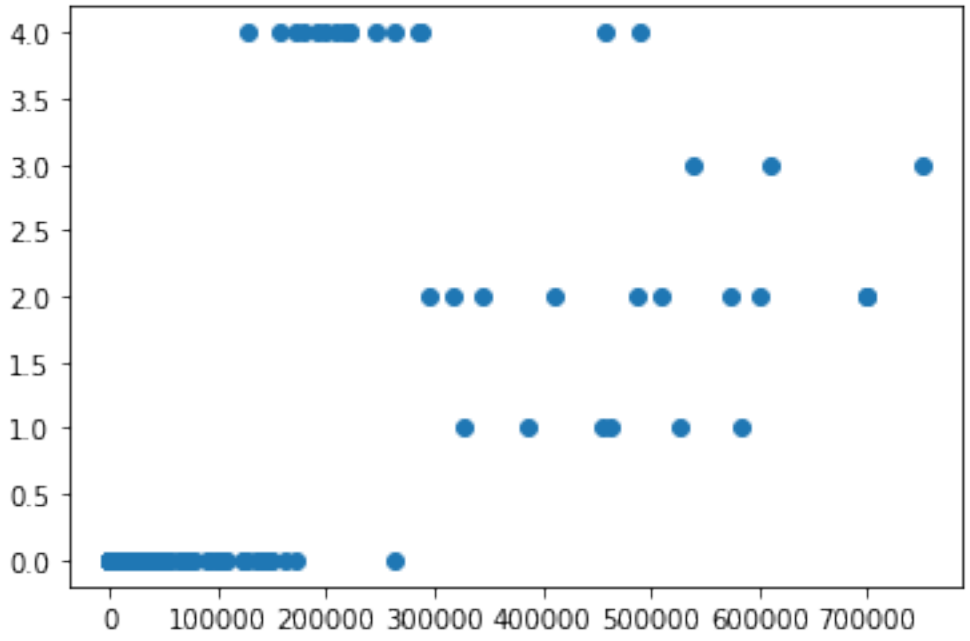

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
250000
300000
350000

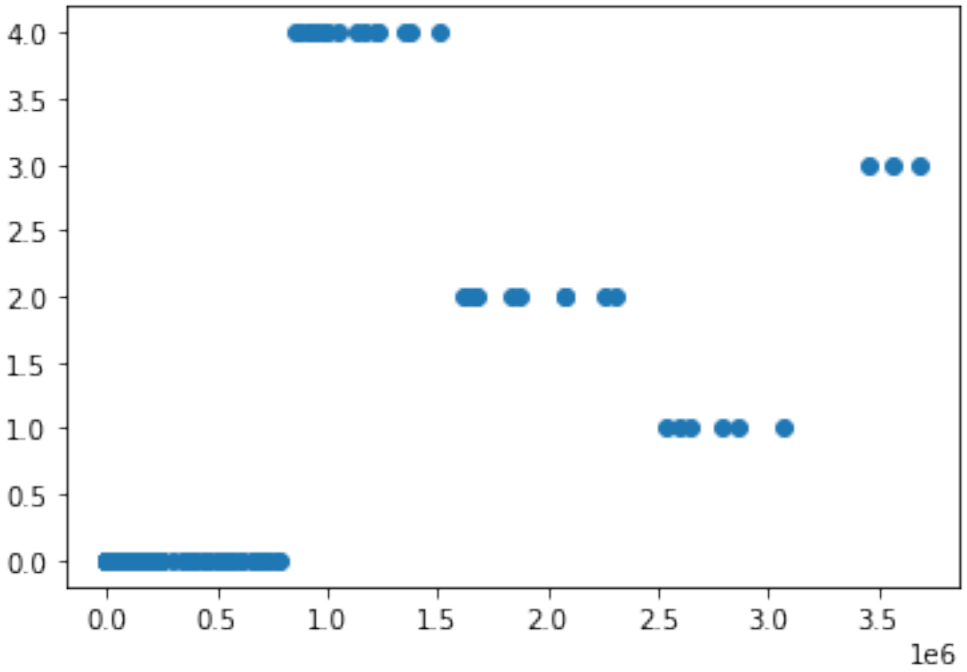

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000

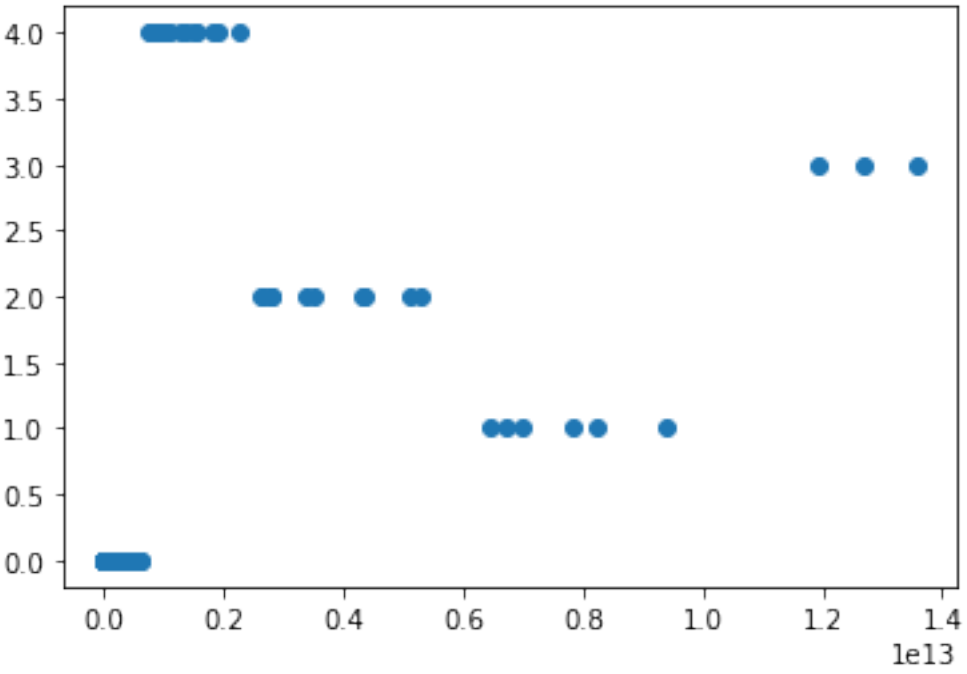

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−1750
−1500
−1250
−1000
−750
−500
−250
0

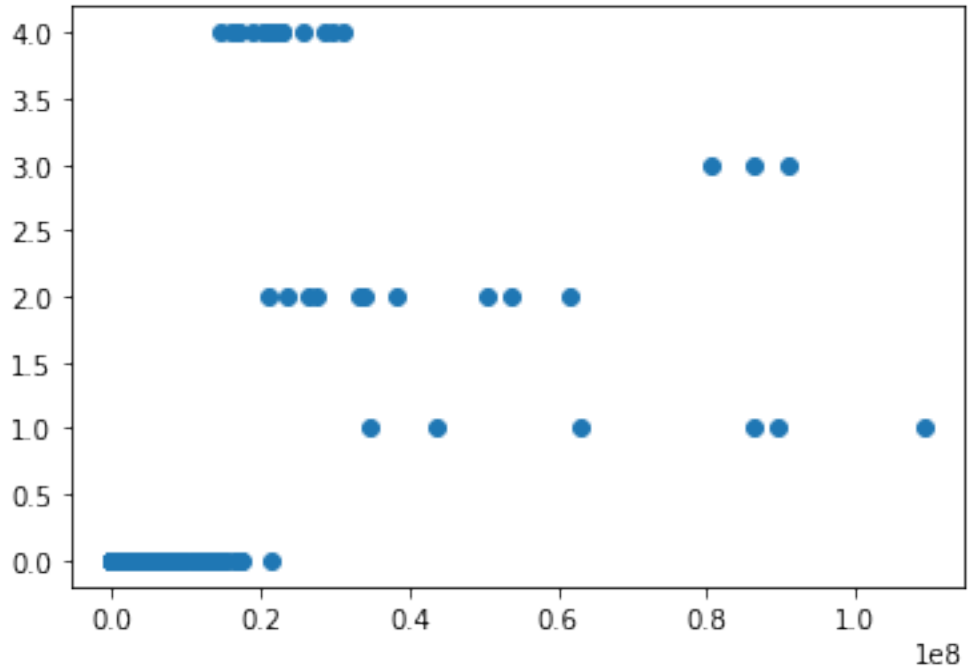

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
600

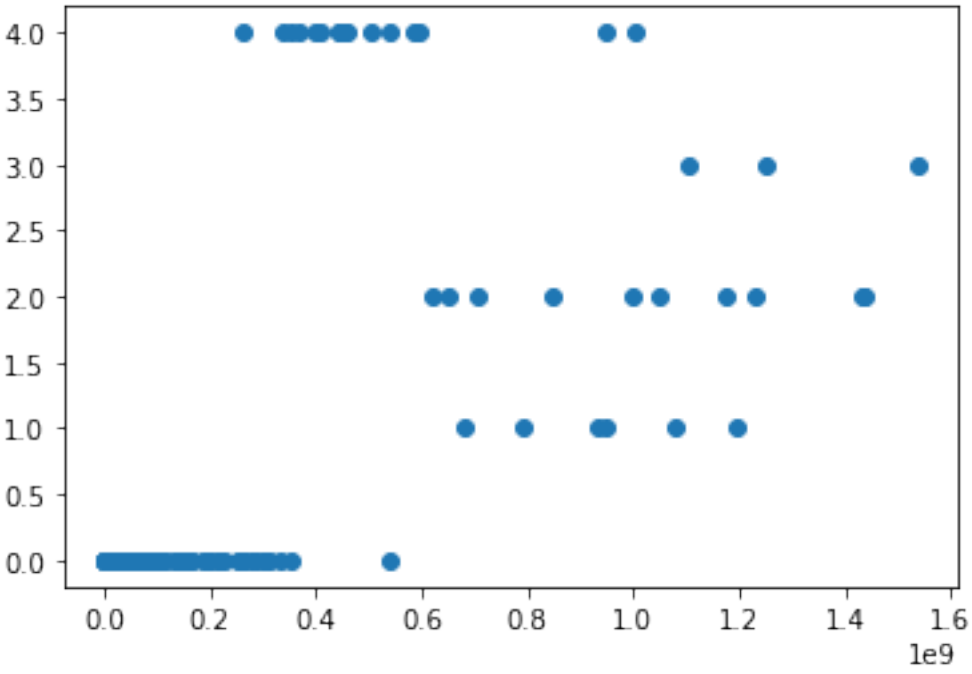

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
250000
300000
350000

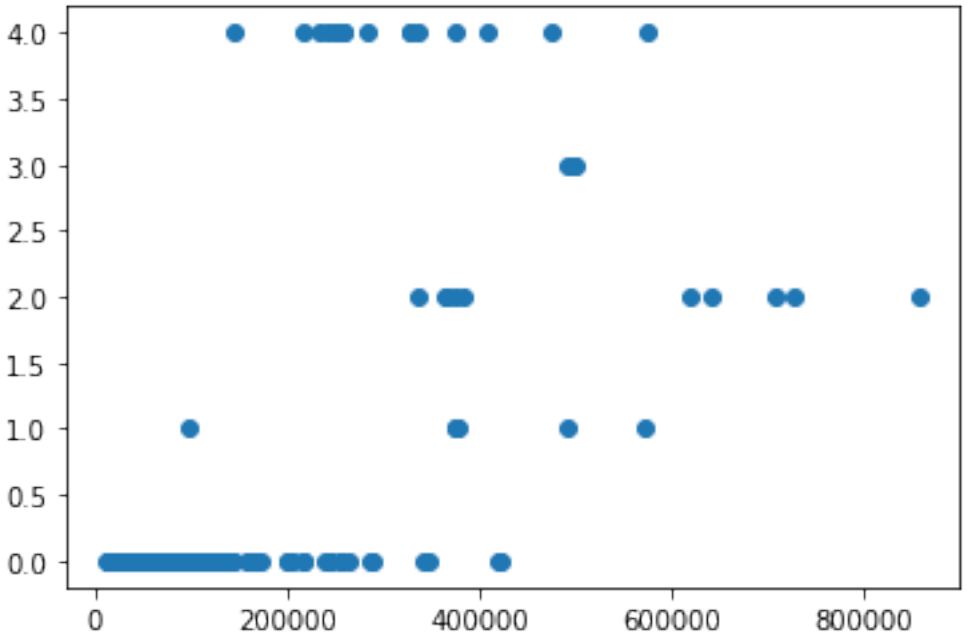

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100000
200000
300000
400000
500000
600000
700000


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e6


4.0
3.5
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
1.4
1e13


4.0
3.5
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
1e8


4.0
3.5
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
1.4
1.6
1e9



[563]:
```
#train model for each cluster
for i in range(0, 5):
    x_train_cluster = x_train[kmeans.labels_ == i]
    y_train_cluster = y_train[kmeans.labels_ == i]
    print(len(x_train_cluster))
    clf = DecisionTreeClassifier(random_state=0)
    clf.fit(x_train_cluster, y_train_cluster)
    y_prediction = clf.predict(x_test)
    model_accuracy = accuracy_score(y_test, y_prediction)
    model_recall = recall_score(y_test, y_prediction)
    model_precision = precision_score(y_test, y_prediction)
    print('cluster', i, '(accuracy, recall, precision):', model_accuracy,␣
 ↪model_recall, model_precision)
```

```
365
cluster 0 (accuracy, recall, precision): 0.97 0.8846153846153846 1.0
6
cluster 1 (accuracy, recall, precision): 0.9 0.6153846153846154 1.0
10
cluster 2 (accuracy, recall, precision): 0.26 1.0 0.26
3
cluster 3 (accuracy, recall, precision): 0.26 1.0 0.26
16
cluster 4 (accuracy, recall, precision): 0.91 0.6538461538461539 1.0
```