Biological data analysis

2020-12-14

Send solutions (ipynb and pdf or html) till 2020-12-19 23:55 to avoicikas@gmail.com

Fill in your name:

Jurga Jasinskaitė

In [20]:
```
import pathlib
import os
import re
import math
import random
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift
import pywt
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.signal as signal
import scipy.stats as stats
import scipy
import mne
import wave
from sklearn.decomposition import FastICA
sns.set()
```

Evaluation:

- Comments 25%
- Applied methods 25%
- Figures 25%
- Results 25%

> > **TASK**
> >
> > Microphones were placed in different locations. 8 recordings from different microphones placed in A7 directory.
> >
> > X1... X8.wav
> >
> > All microphones captured the same environment, but since they were in different locations the source-microphone distances are different.
> >
> > - Separate meaningful sources from the noise.
> > - Plot the signals and their frequency compositions.
> >
> > ---

To import sound `wavfile` from `scipy` can be used

In [165…
```
from scipy.io import wavfile
```

> First the analysis is done with first file only to see what are the parameters of recordings, what is the signal duration and rate, how does the signal look plotted.

In [166…
```
X1 = wave.open('X1.wav','r')
```

In [167…
```
X1.getparams()
```

Out[167…
```
_wave_params(nchannels=1, sampwidth=2, framerate=44100, nframes=930820, comptype='NO
NE', compname='not compressed')
```

> Signal duration in seconds:

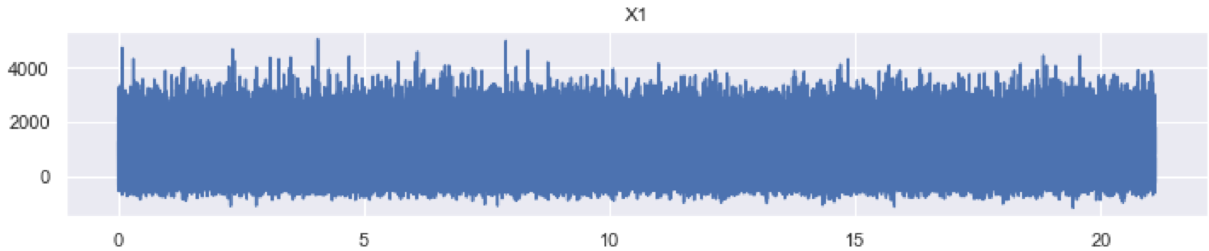
In [168…
```
930820/44100
```

Out[168…
```
21.10702947845805
```

In [169…
```
signal_1_raw = X1.readframes(-1)
signal_1 = np.fromstring(signal_1_raw, 'Int16')
```

```
<ipython-input-169-ffc5df673b11>:2: DeprecationWarning: Numeric-style type codes are
deprecated and will result in an error in the future.
  signal_1 = np.fromstring(signal_1_raw, 'Int16')
<ipython-input-169-ffc5df673b11>:2: DeprecationWarning: The binary mode of fromstrin
g is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instea
d
  signal_1 = np.fromstring(signal_1_raw, 'Int16')
```

In [170…
```
fs = X1.getframerate()
timing = np.linspace(0, len(signal_1)/fs, num=len(signal_1))

plt.figure(figsize=(12,2))
plt.title('X1')
plt.plot(timing,signal_1)
plt.show()
```



> Next analysis is repeated for all recodings seperatly, files are plotted.

In [171…
```
X2 = wave.open('X2.wav','r')
signal_raw_2 = X2.readframes(-1)
signal_2 = np.fromstring(signal_raw_2, 'Int16')

X3 = wave.open('X3.wav','r')
signal_raw_3 = X3.readframes(-1)
signal_3 = np.fromstring(signal_raw_3, 'Int16')

X4 = wave.open('X4.wav','r')
signal_raw_4 = X4.readframes(-1)
signal_4 = np.fromstring(signal_raw_4, 'Int16')

X5 = wave.open('X5.wav','r')
```

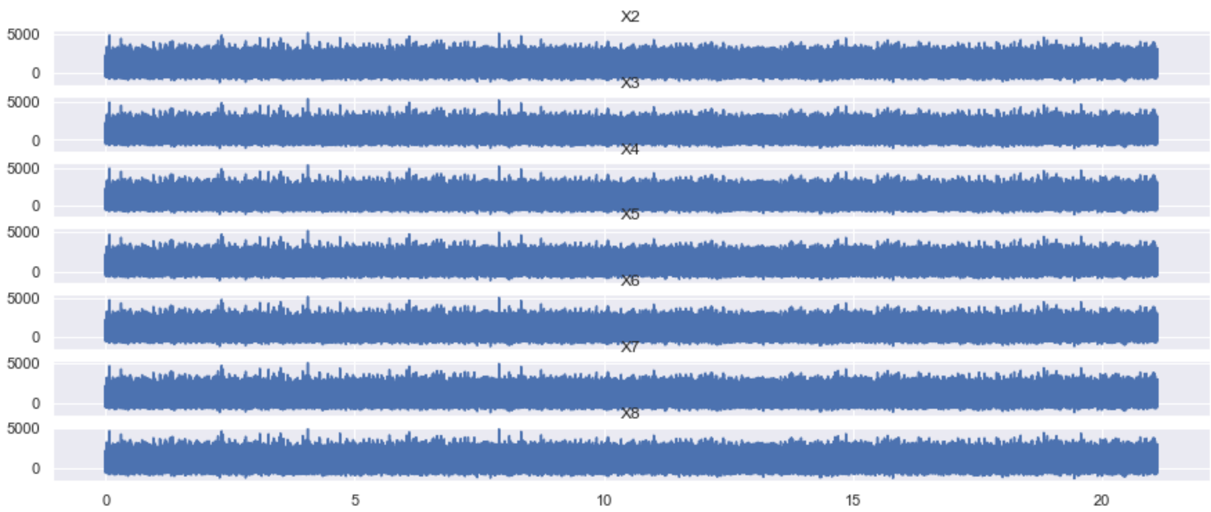
```
signal_raw_5 = X5.readframes(-1)
signal_5 = np.fromstring(signal_raw_5, 'Int16')

X6 = wave.open('X6.wav','r')
signal_raw_6 = X6.readframes(-1)
signal_6 = np.fromstring(signal_raw_6, 'Int16')

X7 = wave.open('X7.wav','r')
signal_raw_7 = X7.readframes(-1)
signal_7 = np.fromstring(signal_raw_7, 'Int16')

X8 = wave.open('X8.wav','r')
signal_raw_8 = X8.readframes(-1)
signal_8 = np.fromstring(signal_raw_8, 'Int16')
```

```
<ipython-input-171-3739528c2c79>:3: DeprecationWarning: Numeric-style type codes are
deprecated and will result in an error in the future.
  signal_2 = np.fromstring(signal_raw_2, 'Int16')
<ipython-input-171-3739528c2c79>:3: DeprecationWarning: The binary mode of fromstrin
g is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instea
d
  signal_2 = np.fromstring(signal_raw_2, 'Int16')
<ipython-input-171-3739528c2c79>:7: DeprecationWarning: Numeric-style type codes are
deprecated and will result in an error in the future.
  signal_3 = np.fromstring(signal_raw_3, 'Int16')
<ipython-input-171-3739528c2c79>:7: DeprecationWarning: The binary mode of fromstrin
g is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instea
d
  signal_3 = np.fromstring(signal_raw_3, 'Int16')
<ipython-input-171-3739528c2c79>:11: DeprecationWarning: Numeric-style type codes ar
e deprecated and will result in an error in the future.
  signal_4 = np.fromstring(signal_raw_4, 'Int16')
<ipython-input-171-3739528c2c79>:11: DeprecationWarning: The binary mode of fromstri
ng is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  signal_4 = np.fromstring(signal_raw_4, 'Int16')
<ipython-input-171-3739528c2c79>:15: DeprecationWarning: Numeric-style type codes ar
e deprecated and will result in an error in the future.
  signal_5 = np.fromstring(signal_raw_5, 'Int16')
<ipython-input-171-3739528c2c79>:15: DeprecationWarning: The binary mode of fromstri
ng is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  signal_5 = np.fromstring(signal_raw_5, 'Int16')
<ipython-input-171-3739528c2c79>:19: DeprecationWarning: Numeric-style type codes ar
e deprecated and will result in an error in the future.
  signal_6 = np.fromstring(signal_raw_6, 'Int16')
<ipython-input-171-3739528c2c79>:19: DeprecationWarning: The binary mode of fromstri
ng is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  signal_6 = np.fromstring(signal_raw_6, 'Int16')
<ipython-input-171-3739528c2c79>:23: DeprecationWarning: Numeric-style type codes ar
e deprecated and will result in an error in the future.
  signal_7 = np.fromstring(signal_raw_7, 'Int16')
<ipython-input-171-3739528c2c79>:23: DeprecationWarning: The binary mode of fromstri
ng is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  signal_7 = np.fromstring(signal_raw_7, 'Int16')
<ipython-input-171-3739528c2c79>:27: DeprecationWarning: Numeric-style type codes ar
e deprecated and will result in an error in the future.
  signal_8 = np.fromstring(signal_raw_8, 'Int16')
<ipython-input-171-3739528c2c79>:27: DeprecationWarning: The binary mode of fromstri
ng is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  signal_8 = np.fromstring(signal_raw_8, 'Int16')
```

In [360…

```
fig = plt.figure(figsize=(15, 6))

plt.subplot(7, 1, 1)
```

```
plt.title('X2')
plt.plot(timing,signal_2)

plt.subplot(7, 1, 2)
plt.title('X3')
plt.plot(timing,signal_4)

plt.subplot(7, 1, 3)
plt.title('X4')
plt.plot(timing,signal_4)

plt.subplot(7, 1, 4)
plt.title('X5')
plt.plot(timing,signal_5)

plt.subplot(7, 1, 5)
plt.title('X6')
plt.plot(timing,signal_7)

plt.subplot(7, 1, 6)
plt.title('X7')
plt.plot(timing,signal_7)

plt.subplot(7, 1, 7)
plt.title('X8')
plt.plot(timing,signal_8)

plt.show()
```



> Files are zipped into one file and Independent component analysis (ICA) ir performed on zipped list. ICA was chosen as an algorithm for signal and noise separation because it separates data into independant aaditive subcomponents than can be visualised and manipulated separatly later on (useful in this case since we have a problem simmilar to cocktail party problem). Then results are split again to corespond to each file that was read in the first step.

In [173…
```
X = list(zip(signal_1, signal_2, signal_3, signal_4, signal_5, signal_6, signal_7, s
```

In [174…
```
from sklearn.decomposition import FastICA
ica = FastICA(n_components=8, random_state=0)
ica_result= ica.fit_transform(X)
```

In [175…
```
ica_result.shape
```

Out[175…
```
(930820, 8)
```

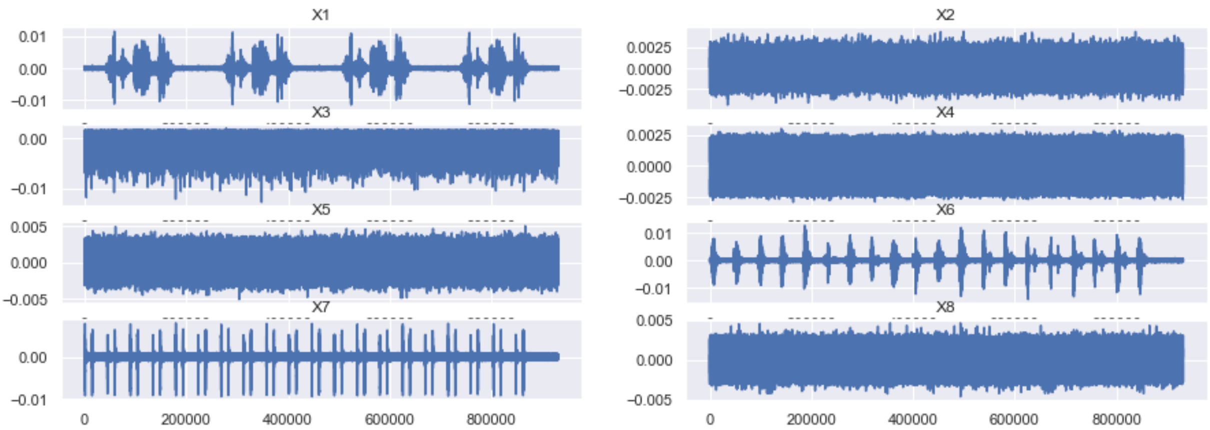
In [176...

```
result_signal_1 = ica_result[:,0]
result_signal_2 = ica_result[:,1]
result_signal_3 = ica_result[:,2]
result_signal_4 = ica_result[:,3]
result_signal_5 = ica_result[:,4]
result_signal_6 = ica_result[:,5]
result_signal_7 = ica_result[:,6]
result_signal_8 = ica_result[:,7]
```

> ### Files after ICA are plotted. It is clear that ICA worked, but only 3 sounds were exctracted and separated.

In [177...

```
fig = plt.figure(figsize=(15, 5))

plt.subplot(4, 2, 1)
plt.title('X1')
plt.plot(result_signal_1)

plt.subplot(4, 2, 2)
plt.title('X2')
plt.plot(result_signal_2)

plt.subplot(4, 2, 3)
plt.title('X3')
plt.plot(result_signal_3)

plt.subplot(4, 2, 4)
plt.title('X4')
plt.plot(result_signal_4)

plt.subplot(4, 2, 5)
plt.title('X5')
plt.plot(result_signal_5)

plt.subplot(4, 2, 6)
plt.title('X6')
plt.plot(result_signal_6)

plt.subplot(4, 2, 7)
plt.title('X7')
plt.plot(result_signal_7)

plt.subplot(4, 2, 8)
plt.title('X8')
plt.plot(result_signal_8)

plt.show()
```



> ### Files are convert to integer (so we can save as PCM 16-bit Wave

> files), mapped the appropriate range for int16 audio and volume is increased in order to better hear the recordings that are saved in the last step. After listening to records, three sounds are separated: a man counting from 1 to 20, birds chripping and heart beating.

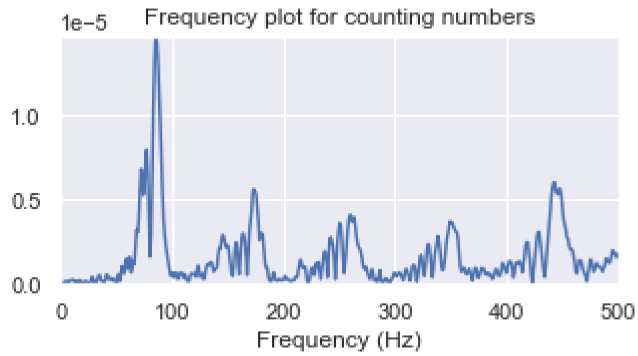
In [178…

```
result_signal_1_int = np.int16(result_signal_1*32767*100)
result_signal_2_int = np.int16(result_signal_2*32767*100)
result_signal_3_int = np.int16(result_signal_3*32767*100)
result_signal_4_int = np.int16(result_signal_4*32767*100)
result_signal_5_int = np.int16(result_signal_5*32767*100)
result_signal_6_int = np.int16(result_signal_6*32767*100)
result_signal_7_int = np.int16(result_signal_7*32767*100)
result_signal_8_int = np.int16(result_signal_8*32767*100)

# Write wave files to listen to them
wavfile.write("result_signal_1.wav", fs, result_signal_1_int)
wavfile.write("result_signal_2.wav", fs, result_signal_2_int)
wavfile.write("result_signal_3.wav", fs, result_signal_3_int)
wavfile.write("result_signal_4.wav", fs, result_signal_4_int)
wavfile.write("result_signal_5.wav", fs, result_signal_5_int)
wavfile.write("result_signal_6.wav", fs, result_signal_6_int)
wavfile.write("result_signal_7.wav", fs, result_signal_7_int)
wavfile.write("result_signal_8.wav", fs, result_signal_8_int)
```

> Below frequency plot for extracted sounds are ploted from clean data after ICA. Numbers counting sound has a reapeating patter in frequency. Heartbeat sound has clear peak in frequency at 50-150 Hz and birds chirping has no distinct freqency.

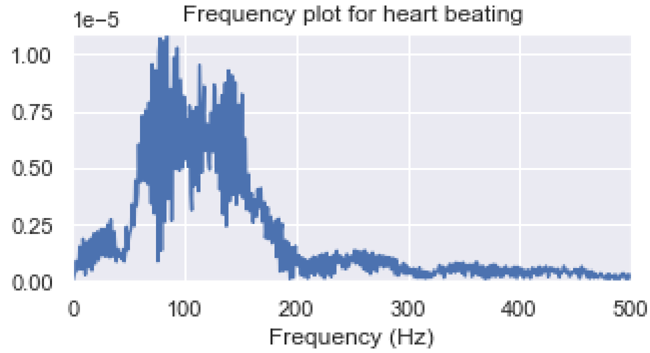
In [179…

```
sample_rate=44100
plt.figure(figsize=(5,5))
plt.subplot(212)
A = np.fft.fft(result_signal_6, sample_rate) / (len(result_signal_6)/2.0)
freq = np.linspace(sample_rate//2*-1, sample_rate//2, len(A))
response = np.abs(fftshift(A))
plt.title('Frequency plot for counting numbers')
plt.plot(freq, response)
plt.axis([0, 500, 0, max(response)])
plt.xlabel("Frequency (Hz)");
```



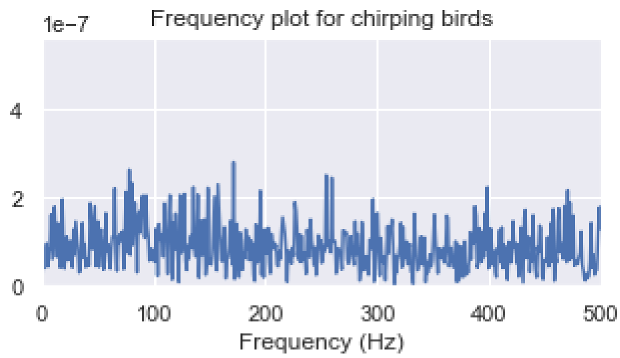
In [180…

```
sample_rate=44100
plt.figure(figsize=(5,5))
plt.subplot(212)
A = np.fft.fft(result_signal_7, sample_rate) / (len(result_signal_7)/2.0)
freq = np.linspace(sample_rate//2*-1, sample_rate//2, len(A))
response = np.abs(fftshift(A))
plt.title('Frequency plot for heart beating')
plt.plot(freq, response)
plt.axis([0, 500, 0, max(response)])
plt.xlabel("Frequency (Hz)");
```



In [181…

```
sample_rate=44100
plt.figure(figsize=(5,5))
plt.subplot(212)
A = np.fft.fft(result_signal_1, sample_rate) / (len(result_signal_1)/2.0)
freq = np.linspace(sample_rate//2*-1, sample_rate//2, len(A))
response = np.abs(fftshift(A))
plt.title('Frequency plot for chirping birds')
plt.plot(freq, response)
plt.axis([0, 500, 0, max(response)])
plt.xlabel("Frequency (Hz)");
```



TASK

EEG recordings capture post synaptic potentials generated by networks in the brain at different frequencies as well as noise from surroundings and other body parts.

![power]

The strength of each source in different EEG channels depend on their relative distance from each channel. Frontal channels are closer to eyes and we observe diminishing eyeblink amplitudes from frontal to ocipital channels. Ocipital channels contains more alpha (~12 Hz) brain activity. Line noise (50 Hz) is equally distributed accross all channels.

- Simulate EEG recording. At least 6 channels from different locations. Signal must contain eye blinks, line noise, and brain activity.
- Separate mixed simulated EEG recording back to sources.
- Plot simulated and separated signals.

Creating noise, alpha wave and blink signals separetly and joining them.

In [258...

```
np.random.seed(0)
dt = 0.02
Fs = 1 / dt
t = np.arange(0, 100, dt)
# noise:
nse = np.random.randn(len(t))
r = np.exp(-t / 0.05)
cnse = np.convolve(nse, r) * dt
cnse = cnse[:len(t)]

noise =(0.1 * np.sin(1 * np.pi * t) + cnse)*500
```

In [259...

```
F = 10
T = 1000/F
Fs = 50
Ts = 1./Fs
N = int(T/Ts)
t = np.linspace(0, T, N)
alpha_wave = np.sin(2*np.pi*F*t)
```

In [260...

```
noise_and_alpha = noise+alpha_wave
```

In [ ]:

In [260...

```
sample_rate = 5000
t1 = np.linspace(0, 35, sample_rate)
t2 = np.linspace(35, 37, sample_rate)
t3 = np.linspace(37, 45, sample_rate)
t4 = np.linspace(45, 47, sample_rate)
t5 = np.linspace(47, 65, sample_rate)
t6 = np.linspace(65, 69, sample_rate)
t7 = np.linspace(69, 100, sample_rate)
```

In [262...

```
def gen_wave(Hz, sample_rate, length_sec, Amp, phase):
    t = np.linspace(0, length_sec, length_sec * sample_rate, endpoint=False)
    x = Amp*(np.sin(Hz * 2 * np.pi * t + phase))
    return(x, t)
sample_rate = 5000
length_sec = 1
blink, t = np.array(gen_wave(1.2, sample_rate, length_sec, 200, -0.5))
```

Cretaing 6 channels of same data with eye blink being weaker i each one of them as seen from the plots below.

In [263...

```
channel1_1 = pd.DataFrame({'time': t1,
                   'signal': noise_and_alpha})
channel1_2 = pd.DataFrame({'time': t2,
                   'signal': (((noise_and_alpha*0.5)+blink))})
channel1_3 = pd.DataFrame({'time': t3,
                   'signal': noise_and_alpha})
channel1_4 = pd.DataFrame({'time': t4,
                   'signal': ((noise*0.2+blink))})
channel1_5 = pd.DataFrame({'time': t5,
                   'signal': noise_and_alpha})
channel1_6 = pd.DataFrame({'time': t6,
                   'signal': ((blink+(noise*0.5)))})
channel1_7 = pd.DataFrame({'time': t7,
                   'signal': noise_and_alpha})
```

```
channel1 = channel1_1.append(channel1_2, ignore_index=True).append(channel1_3, ignor
```

In [264…

```
channel2_1 = pd.DataFrame({'time': t1,
                    'signal': noise_and_alpha})
channel2_2 = pd.DataFrame({'time': t2,
                    'signal': (((noise_and_alpha*0.5)+blink)*0.9)})
channel2_3 = pd.DataFrame({'time': t3,
                    'signal': noise_and_alpha})
channel2_4 = pd.DataFrame({'time': t4,
                    'signal': ((noise*0.2+blink)*0.9)})
channel2_5 = pd.DataFrame({'time': t5,
                    'signal': noise_and_alpha})
channel2_6 = pd.DataFrame({'time': t6,
                    'signal': ((blink+(noise*0.5))*0.9)})
channel2_7 = pd.DataFrame({'time': t7,
                    'signal': noise_and_alpha})

channel2 = channel2_1.append(channel2_2, ignore_index=True).append(channel2_3, ignor
```

In [265…

```
channel3_1 = pd.DataFrame({'time': t1,
                    'signal': noise_and_alpha})
channel3_2 = pd.DataFrame({'time': t2,
                    'signal': (((noise_and_alpha*0.5)+blink)*0.8)})
channel3_3 = pd.DataFrame({'time': t3,
                    'signal': noise_and_alpha})
channel3_4 = pd.DataFrame({'time': t4,
                    'signal': ((noise*0.2+blink)*0.8)})
channel3_5 = pd.DataFrame({'time': t5,
                    'signal': noise_and_alpha})
channel3_6 = pd.DataFrame({'time': t6,
                    'signal': ((blink+(noise*0.5))*0.8)})
channel3_7 = pd.DataFrame({'time': t7,
                    'signal': noise_and_alpha})

channel3 = channel3_1.append(channel3_2, ignore_index=True).append(channel3_3, ignor
```

In [266…

```
channel4_1 = pd.DataFrame({'time': t1,
                    'signal': noise_and_alpha})
channel4_2 = pd.DataFrame({'time': t2,
                    'signal': (((noise_and_alpha*0.5)+blink)*0.7)})
channel4_3 = pd.DataFrame({'time': t3,
                    'signal': noise_and_alpha})
channel4_4 = pd.DataFrame({'time': t4,
                    'signal': ((noise*0.2+blink)*0.7)})
channel4_5 = pd.DataFrame({'time': t5,
                    'signal': noise_and_alpha})
channel4_6 = pd.DataFrame({'time': t6,
                    'signal': ((blink+(noise*0.5))*0.7)})
channel4_7 = pd.DataFrame({'time': t7,
                    'signal': noise_and_alpha})

channel4 = channel4_1.append(channel4_2, ignore_index=True).append(channel4_3, ignor
```

In [267…

```
channel5_1 = pd.DataFrame({'time': t1,
                    'signal': noise_and_alpha})
channel5_2 = pd.DataFrame({'time': t2,
                    'signal': (((noise_and_alpha*0.5)+blink)*0.6)})
channel5_3 = pd.DataFrame({'time': t3,
                    'signal': noise_and_alpha})
channel5_4 = pd.DataFrame({'time': t4,
                    'signal': ((noise*0.2+blink)*0.6)})
channel5_5 = pd.DataFrame({'time': t5,
```

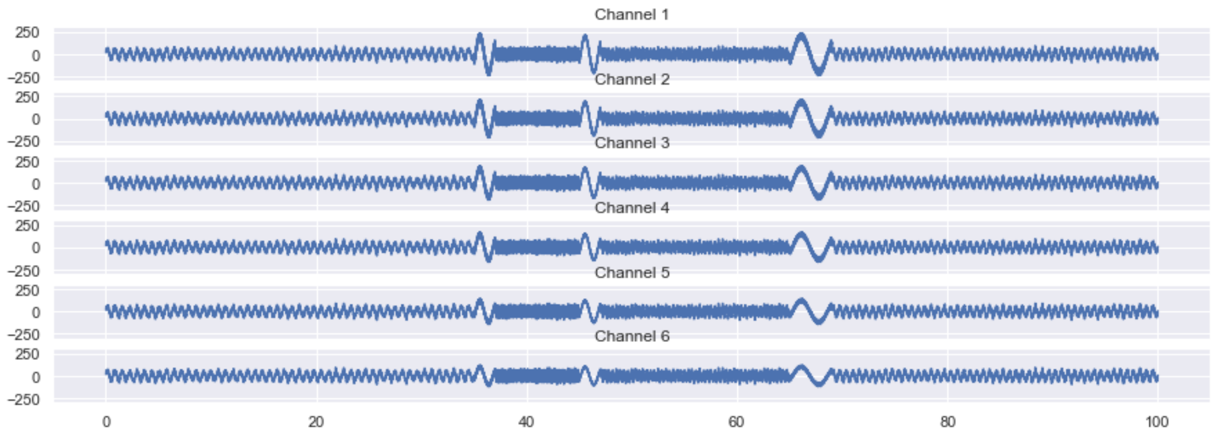
```
                   'signal': noise_and_alpha})
channel5_6 = pd.DataFrame({'time': t6,
                   'signal': ((blink+(noise*0.5))*0.6)})
channel5_7 = pd.DataFrame({'time': t7,
                   'signal': noise_and_alpha})

channel5 = channel5_1.append(channel5_2, ignore_index=True).append(channel5_3, ignor
```

In [268…

```
channel6_1 = pd.DataFrame({'time': t1,
                   'signal': noise_and_alpha})
channel6_2 = pd.DataFrame({'time': t2,
                   'signal': (((noise_and_alpha*0.5)+blink)*0.5)})
channel6_3 = pd.DataFrame({'time': t3,
                   'signal': noise_and_alpha})
channel6_4 = pd.DataFrame({'time': t4,
                   'signal': ((noise*0.2+blink)*0.5)})
channel6_5 = pd.DataFrame({'time': t5,
                   'signal': noise_and_alpha})
channel6_6 = pd.DataFrame({'time': t6,
                   'signal': ((blink+(noise*0.5))*0.5)})
channel6_7 = pd.DataFrame({'time': t7,
                   'signal': noise_and_alpha})
channel6 = channel6_1.append(channel6_2, ignore_index=True).append(channel6_3, ignor
```

In [269…

```
fig = plt.figure(figsize=(15, 5))

plt.subplot(6, 1, 1)
plt.title('Channel 1')
plt.plot(channel1.time, channel1.signal)
plt.ylim(-300,300)

plt.subplot(6, 1, 2)
plt.title('Channel 2')
plt.plot(channel2.time, channel2.signal)
plt.ylim(-300,300)

plt.subplot(6, 1, 3)
plt.title('Channel 3')
plt.plot(channel3.time, channel3.signal)
plt.ylim(-300,300)

plt.subplot(6, 1, 4)
plt.title('Channel 4')
plt.plot(channel4.time, channel4.signal)
plt.ylim(-300,300)

plt.subplot(6, 1, 5)
plt.title('Channel 5')
plt.plot(channel5.time, channel5.signal)
plt.ylim(-300,300)

plt.subplot(6, 1, 6)
plt.title('Channel 6')
plt.plot(channel6.time, channel6.signal)
plt.ylim(-300,300)

plt.show()
```



> Joining all data into one dataframe.

In [270…

```
signals = channel1
signals =signals.drop(['signal'], axis=1)
signals["channel1"]= channel1.signal
signals["channel2"] = channel2.signal
signals["channel3"] = channel3.signal
signals["channel4"] = channel4.signal
signals["channel5"] = channel5.signal
signals["channel6"] = channel6.signal
signals
```

Out[270…

	time	channel1	channel2	channel3	channel4	channel5	channel6
0	0.000000	17.640523	17.640523	17.640523	17.640523	17.640523	17.640523
1	0.007001	19.917029	19.917029	19.917029	19.917029	19.917029	19.917029
2	0.014003	27.250152	27.250152	27.250152	27.250152	27.250152	27.250152
3	0.021004	44.861525	44.861525	44.861525	44.861525	44.861525	44.861525
4	0.028006	54.345049	54.345049	54.345049	54.345049	54.345049	54.345049
...	...	...	...	...	...	...	...
34995	99.975195	-27.528126	-27.528126	-27.528126	-27.528126	-27.528126	-27.528126
34996	99.981396	-13.112386	-13.112386	-13.112386	-13.112386	-13.112386	-13.112386
34997	99.987598	-1.513443	-1.513443	-1.513443	-1.513443	-1.513443	-1.513443
34998	99.993799	0.735897	0.735897	0.735897	0.735897	0.735897	0.735897
34999	100.000000	6.336052	6.336052	6.336052	6.336052	6.336052	6.336052

35000 rows × 7 columns

> Running ICA analysis on dataframe above and choosing 3 components since that's how many I used to generate data, but 2 components exlain the data wuite well too.

In [271…

```
from sklearn.decomposition import FastICA
```

In [306…

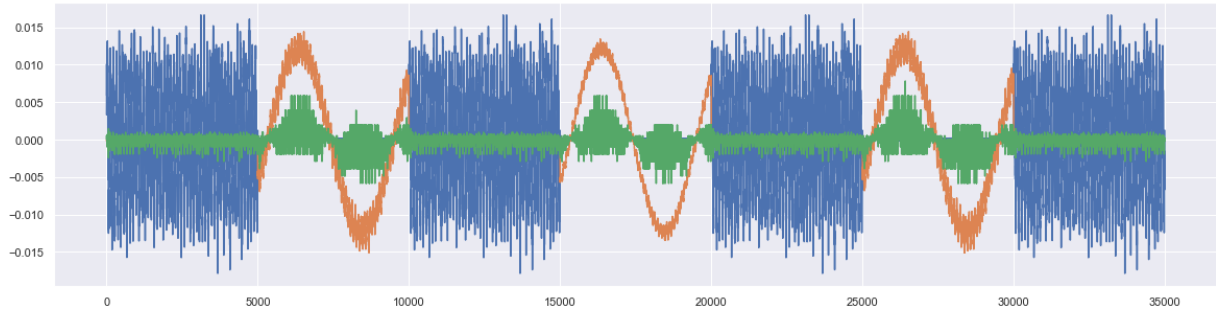
```
X = signals.drop(['time'], axis=1)
ica = FastICA(n_components=3)
X_ica= ica.fit_transform(X)
```

In [329…

```
fig = plt.figure(figsize=(20, 5))
plt.plot(X_ica)
```

Out[329…

```
[<matplotlib.lines.Line2D at 0x1c4100fb3a0>,
 <matplotlib.lines.Line2D at 0x1c4100fb460>,
 <matplotlib.lines.Line2D at 0x1c4100fb520>]
```



TASK

Flow cytometry allows researchers to identify, serparate and characterise different cell types. The detector and analog-to-digital conversion (ADC) system converts analog measurements of forward-scattered light (FSC) and side-scattered light (SSC) as well as dye-specific fluorescence signals into digital signals that can be processed by a computer.

flowcyt

cyto.csv contains data from flow cytometer.

FSCH/SSCH are measurements of the scattering. These parameters can be simplified as measures of a cell's size (FSCH) and a cell's internal complexity (SSCH).

FL1_H ... are flourescence parameters. Different fluorochromes were used to distinguish subpopulations.

Gate is the label given to each cell by the researcher. Two gates have been identified and labelled as 1 and 2. Noise labelled as -1.

- Explore the dataset
- Form a model to classify new data points as a particular gate.

Importing of data

In [274…

```
df = pd.read_csv ('cyto.csv')
df.head()
```

Out[274…

	FSC_H	SSC_H	FL1_H	FL2_H	FL3_H	FL1_A	FL1_W	Time	Gate
0	309	376	264	198	313	0	0	2	1
1	83	55	139	51	146	0	0	2	1
2	184	198	232	83	124	0	0	2	1
3	169	75	696	22	193	121	26	2	2
4	212	98	166	0	221	0	0	2	1

> **Data is passed into PCA analysis to reduce dimensions so the following analysis is easier to plot and understand.**

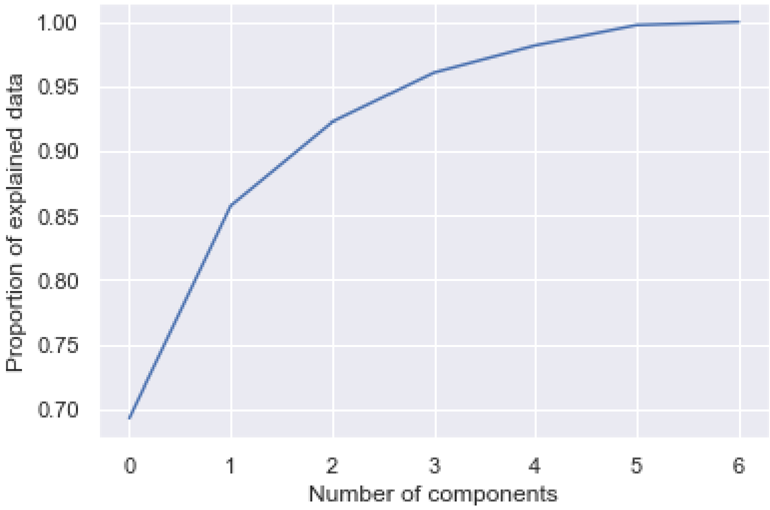
In [275…
```
from sklearn.decomposition import PCA
X = df.drop(['Gate', 'Time'], axis=1)
y = df.Gate
```

In [276…
```
pca = PCA(n_components=7,svd_solver='auto', random_state=1)
pca.fit(X)
```

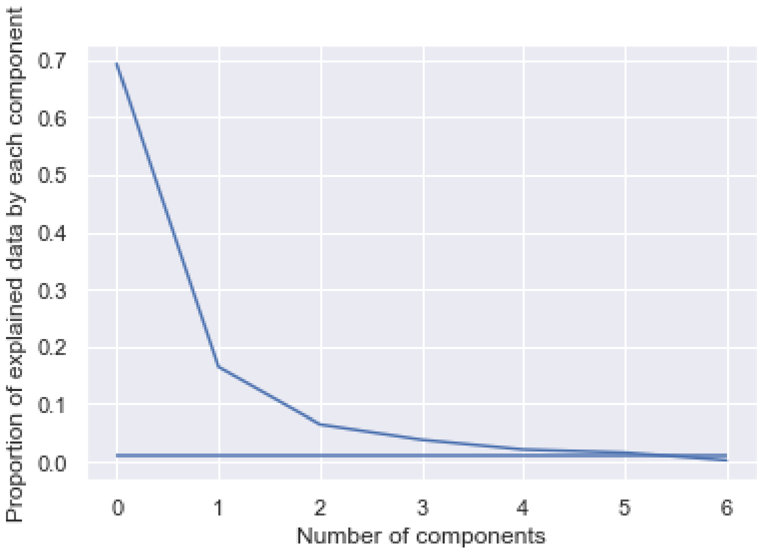
Out[276…
```
PCA(n_components=7, random_state=1)
```

> **As from graphs and calculations below, if 90% of data explained by model is enough, 3 components are chosen for further analysis.**

In [277…
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Proportion of explained data');
```



In [278…
```
plt.plot(pca.explained_variance_ratio_)
plt.hlines(1/100,0,6)
plt.xlabel('Number of components')
plt.ylabel('Proportion of explained data by each component');
```



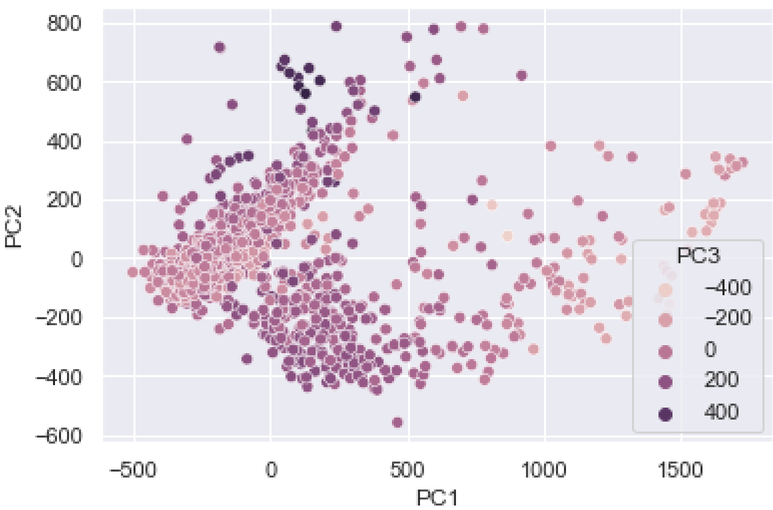

```
pca = PCA(0.90).fit(X)
```

In [279…
```
pca.n_components_
```
Out[279… 3

> ### After PCA data's dimensions are reduced from 9 to 3.

In [280…
```
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
df.shape
```
Out[280… (1545, 9)

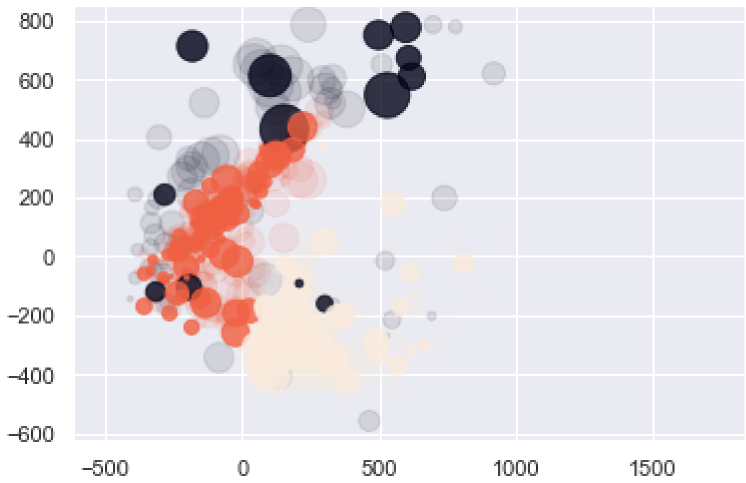
In [281…
```
X_pca.shape
```
Out[281… (1545, 3)

In [282…
```
X_pca_df = pd.DataFrame(X_pca, columns=['PC1','PC2','PC3'])
ax = sns.scatterplot('PC1','PC2', 'PC3', data=X_pca_df, hue = y)
```
```
C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fu
tureWarning: Pass the following variables as keyword args: x, y, hue. From version
0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```


> CLUSTERING

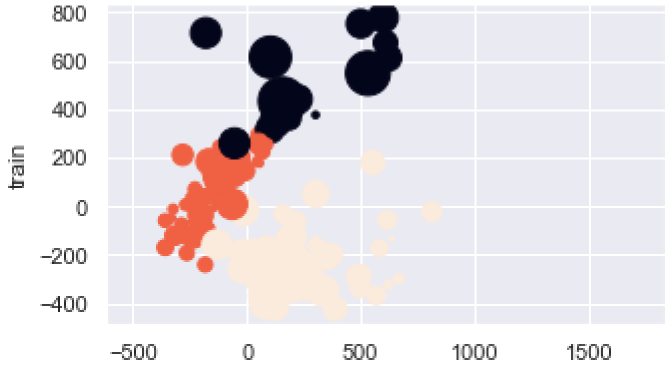
> ### First clusterring methon used is k-Nearest Neighbors, which gives a 0.959 score.

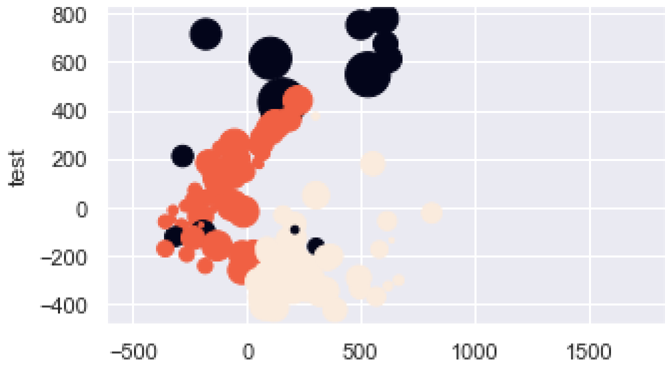
In [283…
```
from sklearn.neighbors import KNeighborsClassifier
```
In [284…
```
X = X_pca_df
y = df.Gate
```
In [285…
```
from sklearn.model_selection import train_test_split
##Splitting data into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```
In [286…
```
X_train.shape
```
Out[286… (1158, 3)

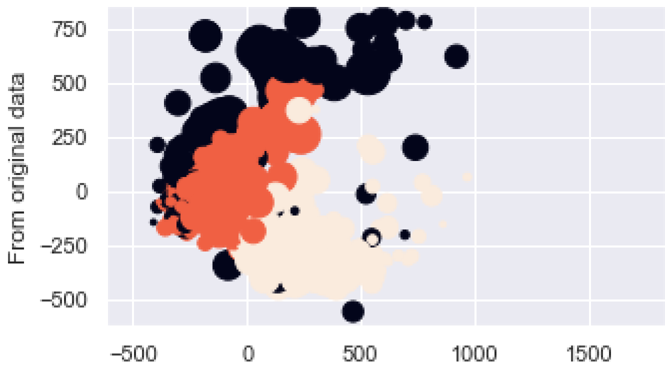
In [287…
```
X_test.shape
```
Out[287…
```
(387, 3)
```
In [288…
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```
Out[288…
```
0.958656330749354
```
In [289…
```
plt.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], X_train.iloc[:, 2], alpha=0.1, c
plt.scatter(X_test.iloc[:,0], X_test.iloc[:,1], X_test.iloc[:,2], alpha=0.8, c=y_tes
```
```
C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\matplotlib\collections.py:92
2: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```


> ### Next I tried Bayes classification method which have a score of 0.912.

In [290…
```
from sklearn.naive_bayes import GaussianNB
```
In [291…
```
from sklearn.metrics import accuracy_score
model = GaussianNB()
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=1)
model.fit(Xtrain, ytrain)
ynew = model.predict(Xtest)
accuracy_score(ytest, ynew)
```
Out[291…
```
0.9121447028423773
```
In [292…
```
fig, ax = plt.subplots(3, 1, figsize=(5,10))
ax[0].scatter(Xtest.iloc[:, 0], Xtest.iloc[:, 1], Xtest.iloc[:, 2],c=ynew)
ax[0].set_ylabel("train")
ax[1].scatter(Xtest.iloc[:, 0], Xtest.iloc[:, 1], Xtest.iloc[:, 2],c=ytest)
ax[1].set_ylabel("test");
ax[2].scatter(X_pca_df.iloc[:, 0], X_pca_df.iloc[:, 1], X_pca_df.iloc[:, 2],alpha=1,
ax[2].set_ylabel("From original data");
```
```
C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\matplotlib\collections.py:92
2: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```







> ### Next method is KMeans which clustered the results quite well despite not having the "Gate" column available.

In [293…

```
from sklearn.cluster import KMeans
```

In [294…

```
kmeans = KMeans(n_clusters=3, n_init=10)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
y_kmeans
```

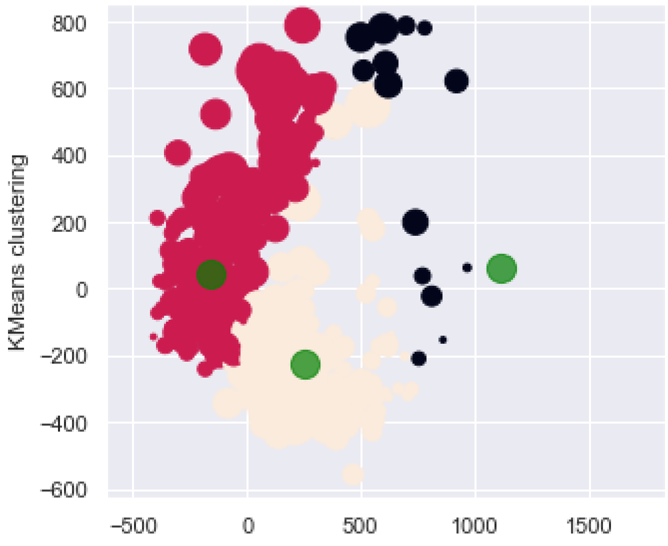
Out[294…

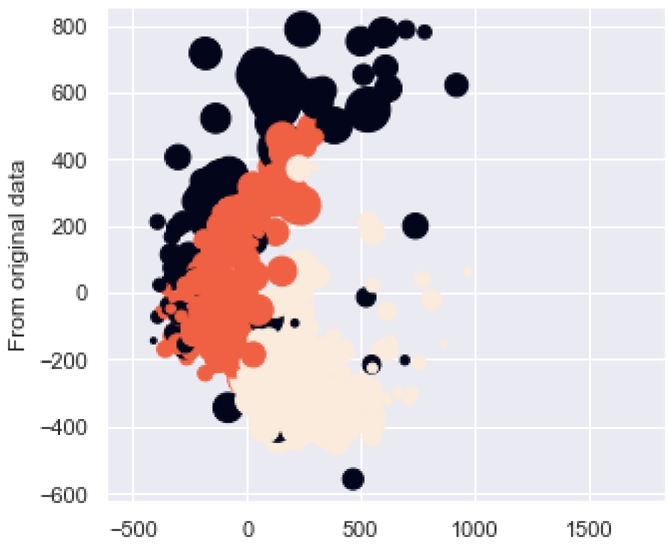
```
array([1, 1, 1, ..., 1, 0, 1])
```

In [295…

```
fig, ax = plt.subplots(2, 1, figsize=(5,10))
centers = kmeans.cluster_centers_
ax[0].scatter(X_pca_df.iloc[:, 0], X_pca_df.iloc[:, 1], X_pca_df.iloc[:, 2],alpha=1,
ax[0].scatter(centers[:, 0], centers[:, 1], c="green", s=200, alpha=0.7);
ax[0].set_ylabel("KMeans clustering")
ax[1].scatter(X_pca_df.iloc[:, 0], X_pca_df.iloc[:, 1], X_pca_df.iloc[:, 2],alpha=1,
ax[1].set_ylabel("From original data");
```

```
C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\matplotlib\collections.py:92
2: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```





Additional results:

Logistic Regression 91%

Support Vector Machine 84%

Random Forest 90%

In [296…
```
X1 = df.drop(['Gate', 'Time'], axis=1)
y1 = df.Gate
```

In [297…
```
import sklearn as sk
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(random_state=0, solver='lbfgs', multi_class='ovr').fit(X1, y
LR.predict(X1.iloc[460:,:])
round(LR.score(X1,y1), 4)
```

Out[297… 0.9081

In [298…
```
from sklearn import svm

SVM = svm.LinearSVC()
SVM.fit(X1, y1)
SVM.predict(X1.iloc[460:,:])
round(SVM.score(X1,y1), 4)
```

```
C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\sklearn\svm\_base.py:976: Con
```

```
vergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[298…

```
0.8777
```

In [299…

```python
import sklearn as sk
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)
RF.fit(X1, y1)
RF.predict(X1.iloc[460:,:])
round(RF.score(X1,y1), 4)
```

Out[299…

```
0.9042
```

> ## Trying to implement Binary classification algorithm but it is not working for some reason and I cannot troubleshoot it.

In [300…

```python
train_data, test_data, train_labels, test_labels = train_test_split(X, y, random_sta
```

In [301…

```python
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
x_train = np.asarray(train_labels).astype('float32')
x_test = np.asarray(test_labels).astype('float32')
```

In [302…

```python
from keras import layers, models
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_7 (Dense)              (None, 16)                272
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
_________________________________________________________________
```

In [303…

```python
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [304…

```python
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

In [305…

```python
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-305-90a50ee7d258> in <module>
----> 1 history = model.fit(partial_x_train,
      2                     partial_y_train,
      3                     epochs=20,
      4                     batch_size=512,
      5                     validation_data=(x_val, y_val))

~\Anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in fit(sel
f, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data,
 shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_st
eps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiproce
ssing)
   1098                 _r=1):
   1099               callbacks.on_train_batch_begin(step)
-> 1100               tmp_logs = self.train_function(iterator)
   1101               if data_handler.should_sync:
   1102                 context.async_wait()

~\Anaconda3\lib\site-packages\tensorflow\python\eager\def_function.py in __call__(se
lf, *args, **kwds)
    826     tracing_count = self.experimental_get_tracing_count()
    827     with trace.Trace(self._name) as tm:
--> 828       result = self._call(*args, **kwds)
    829       compiler = "xla" if self._experimental_compile else "nonXla"
    830       new_tracing_count = self.experimental_get_tracing_count()

~\Anaconda3\lib\site-packages\tensorflow\python\eager\def_function.py in _call(self,
*args, **kwds)
    869       # This is the first call of __call__, so we have to initialize.
    870       initializers = []
--> 871       self._initialize(args, kwds, add_initializers_to=initializers)
    872     finally:
    873       # At this point we know that the initialization is complete (or less

~\Anaconda3\lib\site-packages\tensorflow\python\eager\def_function.py in _initialize
(self, args, kwds, add_initializers_to)
    723     self._graph_deleter = FunctionDeleter(self._lifted_initializer_graph)
    724     self._concrete_stateful_fn = (
--> 725         self._stateful_fn._get_concrete_function_internal_garbage_collected(
# pylint: disable=protected-access
    726             *args, **kwds))
    727 

~\Anaconda3\lib\site-packages\tensorflow\python\eager\function.py in _get_concrete_f
unction_internal_garbage_collected(self, *args, **kwargs)
   2967       args, kwargs = None, None
   2968     with self._lock:
-> 2969       graph_function, _ = self._maybe_define_function(args, kwargs)
   2970     return graph_function
   2971 

~\Anaconda3\lib\site-packages\tensorflow\python\eager\function.py in _maybe_define_f
unction(self, args, kwargs)
   3359 
   3360           self._function_cache.missed.add(call_context_key)
-> 3361           graph_function = self._create_graph_function(args, kwargs)
   3362           self._function_cache.primary[cache_key] = graph_function
   3363 

~\Anaconda3\lib\site-packages\tensorflow\python\eager\function.py in _create_graph_f
unction(self, args, kwargs, override_flat_arg_shapes)
   3194     arg_names = base_arg_names + missing_arg_names
   3195     graph_function = ConcreteFunction(
-> 3196         func_graph_module.func_graph_from_py_func(
   3197             self._name,
   3198             self._python_function,

~\Anaconda3\lib\site-packages\tensorflow\python\framework\func_graph.py in func_grap
```

```
h_from_py_func(name, python_func, args, kwargs, signature, func_graph, autograph, au
tograph_options, add_control_dependencies, arg_names, op_return_value, collections,
 capture_by_value, override_flat_arg_shapes)
    988         _, original_func = tf_decorator.unwrap(python_func)
    989
--> 990       func_outputs = python_func(*func_args, **func_kwargs)
    991
    992       # invariant: `func_outputs` contains only Tensors, CompositeTensors,

~\Anaconda3\lib\site-packages\tensorflow\python\eager\def_function.py in wrapped_fn
(*args, **kwds)
    632           xla_context.Exit()
    633         else:
--> 634           out = weak_wrapped_fn().__wrapped__(*args, **kwds)
    635         return out
    636

~\Anaconda3\lib\site-packages\tensorflow\python\framework\func_graph.py in wrapper(*
args, **kwargs)
    975           except Exception as e:  # pylint:disable=broad-except
    976             if hasattr(e, "ag_error_metadata"):
--> 977               raise e.ag_error_metadata.to_exception(e)
    978             else:
    979               raise

ValueError: in user code:

    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\training.py:805 train_function  *
        return step_function(self, iterator)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\training.py:795 step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\distrib
ute\distribute_lib.py:1259 run
        return self._extended.call_for_each_replica(fn, args=args, kwargs=kwargs)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\distrib
ute\distribute_lib.py:2730 call_for_each_replica
        return self._call_for_each_replica(fn, args, kwargs)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\distrib
ute\distribute_lib.py:3417 _call_for_each_replica
        return fn(*args, **kwargs)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\training.py:788 run_step  **
        outputs = model.train_step(data)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\training.py:754 train_step
        y_pred = self(x, training=True)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\base_layer.py:998 __call__
        input_spec.assert_input_compatibility(self.input_spec, inputs, self.name)
    C:\Users\jurga.jasinskaite\Anaconda3\lib\site-packages\tensorflow\python\keras\e
ngine\input_spec.py:255 assert_input_compatibility
        raise ValueError(

    ValueError: Input 0 of layer sequential_2 is incompatible with the layer: expect
ed axis -1 of input shape to have value 10000 but received input with shape (None,
 1)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```