Lab2_Combined

September 11, 2020

1 Lab 2

Mayank Shouche, ms73656
Daniel Li, ddl933
Sunny Kharel, sk37963

1.1 Question 1

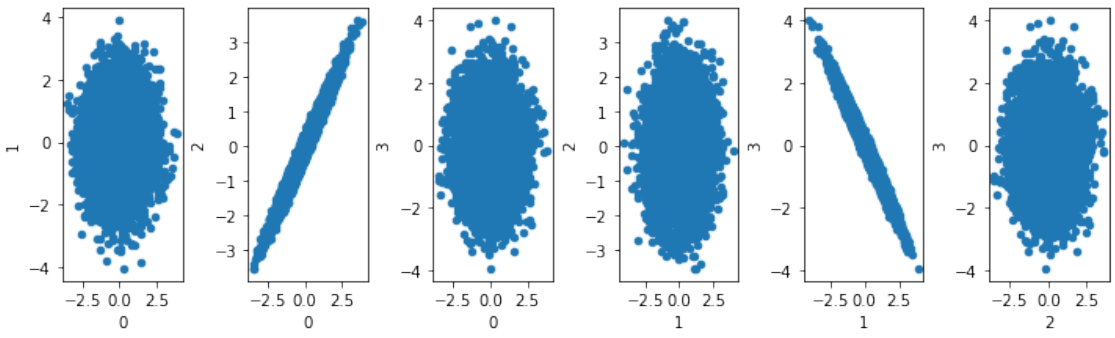
- Which columns are (pairwise) correlated? Figure out how to do this with Pandas, and also how to do this with Seaborn.

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('DF1', usecols=range(1, 5))
```

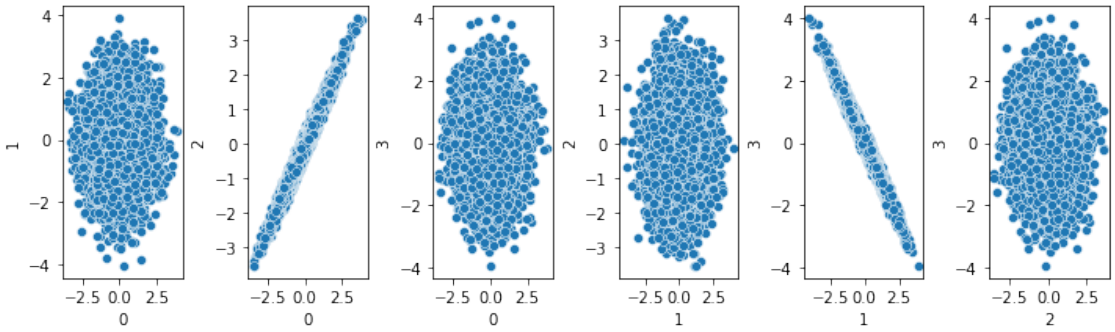
[3]:
```
# with pandas
fig, axs = plt.subplots(nrows=1,ncols=6,figsize=(10, 3))
fig.tight_layout()

counter = 0
for i in range(0, 4):
    for j in range(i + 1, 4):
        data.plot.scatter(x=i, y=j, ax=axs[counter])
        counter += 1
```



[4]:
```
# with seaborn
fig, axs = plt.subplots(nrows=1,ncols=6,figsize=(10, 3))
fig.tight_layout()

counter = 0
for i in range(0, 4):
    for j in range(i + 1, 4):
        sns.scatterplot(data=data, x=f'{i}', y=f'{j}', ax=axs[counter])
        counter += 1
```



From the above plots, it can be seen that columns **0** and **2** & columns **1** & **3** are pairwise correlated, as they form a strong positive or negative linear figure when plotted. The others seem to have no correlation pairwise.

- Compute the covariance matrix of the data. Write the explicit expression for what this is, and then use any command you like (e.g., np.cov) to compute the 4×4 matrix. Explain why the numbers that you get fit with the plots you got.

The formula to calculate the covariance matrix is as follows:

$\mathbf{C} = \frac{1}{n-1}\mathbf{D}_{cent}^{T}\mathbf{D}_{cent}$

where n is the number of samples, and $\mathbf{D}_{cent}$ is the sample matrix, with each column centered about the mean.

[5]:
```
np.cov(np.transpose(data.to_numpy()))
```

[5]:
```
array([[ 1.00155793, -0.00401176,  0.99162409,  0.00412485],
       [-0.00401176,  1.00537841, -0.00409877, -0.99545662],
       [ 0.99162409, -0.00409877,  1.00158867,  0.00408108],
       [ 0.00412485, -0.99545662,  0.00408108,  1.00516828]])
```

The covariance array for this dataset appears to show a strong covariance between columns **0** and **2** & columns **1** & **3**. This is the same as the results from the plots, as correlation is just a scalar multiple of covariance, so columns that are correlated to a high degree should show a relative higher

covariance than other columns. Furthermore, the correlation numbers in this case should strongly match the covariance numbers, as the data was drawn from gaussians with $\sigma = 1$, so the multiplier for correlation is just 1.

- The above problem in reverse. Generate a zero-mean multivariate Gaussian random variable in 3 dimensions,$Z = (X1, X2, X3)$ so that $(X1, X2)$ and $(X1, X3)$ are uncorrelated, but $(X2, X3)$ are correlated. Specifically: choose a covariance matrix that has theabove correlations structure, and write this down. Then find a way to generate samples from this Gaussian. Choose one of the non-zero covariance terms (C_{ij}, if C denotes your covariance matrix) and plot it vs the estimated covariance term, as the number of samples you use scales. The goal is to get a visual representation of how the empirical covariance converges to the true (or family) covariance.

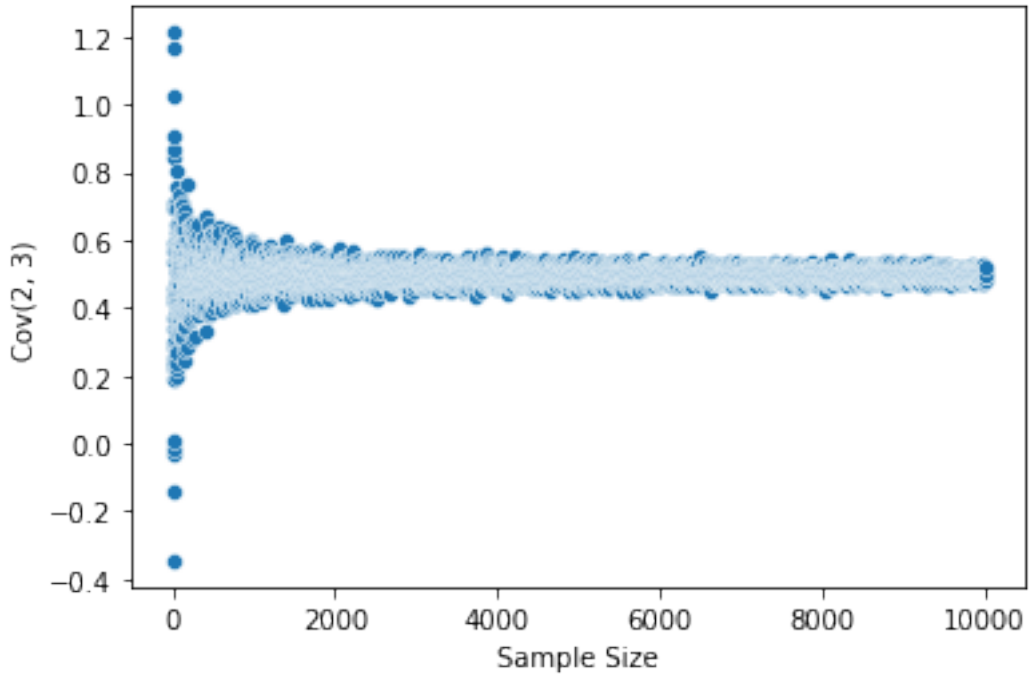
[6]:
```
mean = np.array([0, 0, 0])
cov = np.array([[1, 0, 0], [0, 1, 0.50], [0, 0.50, 1]])

sample_sizes = list(range(2, 10000))
covariance_terms = []

for i in sample_sizes:
    norm = np.random.multivariate_normal(mean, cov, i)
    covariance_terms.append(np.cov(np.transpose(norm))[1][2])

sns.scatterplot(x=sample_sizes, y=covariance_terms)
plt.xlabel('Sample Size')
plt.ylabel('Cov(2, 3)')
```

[6]:
```
Text(0, 0.5, 'Cov(2, 3)')
```



The graph above plots C_{23} versus sample size. The chosen covariance matrix was $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.5 \\ 0 & 0.5 & 1 \end{bmatrix}$, so the true value of C_{23} was 0.5. As can be seen in the graph above, this is the value point estimates of C_{23} approach at larger sample sizes.

1.2 Question 2 - Outliers

Consider the two-dimensional data in DF2 in Lab2 Data.zip. Look at a scatter plot of the data. It contains two points that look like potential outliers. Which one is "more" outlying? Propose a transformation of the data that makes it clear that the point at $(-1, 1)$ is more outlying than the point at $(5.5, 5)$, even though the latter point is"farther away" from the nearest points. Plot the data again after performing this transformation. Provide discussion as appropriate to justify your choice of transformation.

Hint: if y comes from a standard Gaussian in two dimensions (i.e., with covariance equal to the two by two identity matrix), and $Q = \begin{pmatrix} 2 & \frac{1}{2} \\ \frac{1}{2} & 2 \end{pmatrix}$

what is the covariance matrix of the random variable z = Qy? If you are given z, how would you create a random Gaussian vector with covariance equal to the identity, using z?

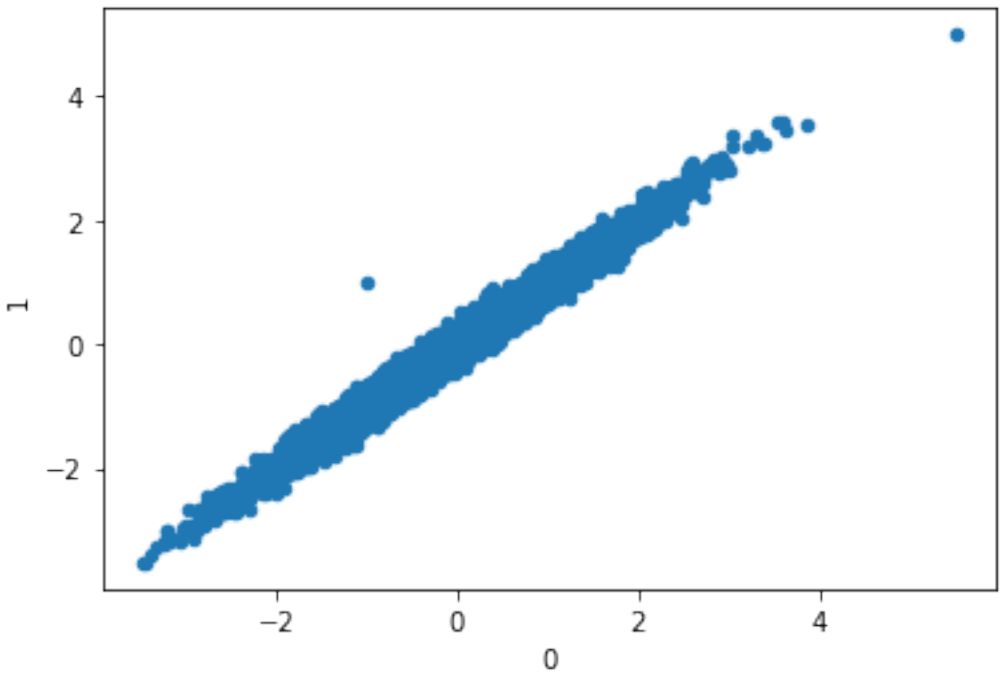
[1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

[2]:
```
# Reading data and making the dataframe
data = pd.read_csv('DF2')
testdata = pd.read_csv('DF2')
df = pd.DataFrame(data)

# From looking at the dataframe, the first column seemed to just count num. of␣
 ↪data entries. Second and third columns are actual data.
print(list(df), '\n')

# Plotting scatter plot of the data
scatterPlot = df.plot.scatter(x='0', y='1', c = None)
```

```
['Unnamed: 0', '0', '1']
```



From scatter plot, two data points seem to be outliers. (-1, 1) and (5.5, 5). We can also that most data points are ploted around the line x = y.

[3]:
```
np.cov(df['0'], df['1'])
```

[3]:
```
array([[1.00464777, 0.9942424 ],
       [0.9942424 , 1.00415964]])
```

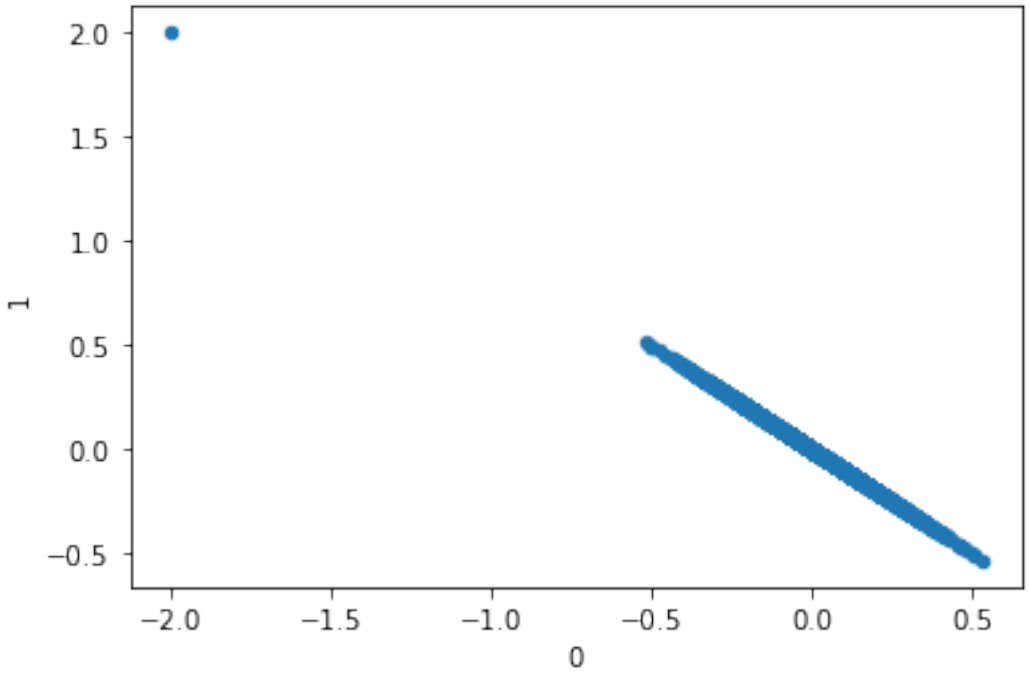
[4]:
```
# Transformation Matrix
q = np.asarray([[1, -1],
                [-1, 1]])
```

[5]:
```
# Transforming all data by z = Qy where z is output data, y is input data (data␣
↪graphed above), Q is transformation matrix, Qy is dot product
x = np.asarray(df['0'])
y = np.asarray(df['1'])
z = np.zeros(len(x))
for i in range(0, len(x)):
    z = [x[i], y[i]]
    c = np.dot(q, z)
    x[i] = c[0]
    y[i] = c[1]
```

[6]:
```
# Graph new matrix and show covariance matrix.
scatterPlot = df.plot.scatter(x= '0', y= '1', c = None)
```

```
cov_matrix = np.cov(df['0'], df['1'])
cov_matrix
```

[6]:
```
array([[ 0.02032261, -0.02032261],
       [-0.02032261,  0.02032261]])
```



[7]:
```
# Outlier we want to separate
a = [-1, 1]
c = np.dot(q, a)
c
```

[7]:
```
array([-2,  2])
```

The transformation matrix $Q = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ was picked because points around the x=y line would all be mapped aroumed the origin as it would subtract x and y allowing new x and y to be 0 since the original x and y's are around x=y. Meanwhile the outlier since it is (-1, 1) would have its new coordinates at (-2, 2).

1.3 Question 3

Generate data as follows: x_i $N(0,1), e_i$ $N(0,1)$. Generate y by $y_i = {}_0 + x_i + e_i$, where ${}_0 = -3$ and $= 0$.

[1]:
```
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt

def generate_data(size):
    e_i = np.random.normal(0, 1, size)
    x_i = np.random.normal(0, 1, size)
    return (np.column_stack((x_i, np.ones(size))), -3 + (0 * x_i) + e_i)
```

- By creating fresh data and each time computing $\hat{\beta}$ and recording $\hat{\beta} - \beta$, compute the empirical standard deviation of the error for $n = 150$. By running a linear regression of y vs. noise, we find $\hat{\beta} = -0.15$. Given your empirical computation of the standard deviation of the error, how significant is the value -0.15?

[2]:
```
errs = []

for i in range(2000): # get 2000 samples of the error
    x, y = generate_data(150)
    beta = np.linalg.inv(x.T @ x) @ x.T @ y
    errs.append(beta[0] - 0) # true beta is 0, so error is just value of␣
 ↪beta_hat[0]

std_err = np.std(errs, ddof=1)
print('Standard deviation of error:', std_err) # ddof = 1 to divide by n-1␣
 ↪instead of n (sample vs population std. dev)

delta = -0.15/std_err
print(f'-0.15 is {np.abs(delta)} standard deviations away from the true value␣
 ↪of beta = 0')
p_value = norm.cdf(delta) - norm.cdf(-delta)
print(f'The probability this occured with our given noise model is␣
 ↪{abs(p_value)}')
```

```
Standard deviation of error: 0.08281099973591191
-0.15 is 1.8113535699165195 standard deviations away from the true value of beta
= 0
The probability this occured with our given noise model is 0.9299138617630667
```

A $\hat{\beta}$ of -0.15 is, according to our empirical value, approximately 1.81 standard distributions away from the mean, and has an approximately 93% chance of originating from our noise model (assuming normally-distributed errors), which indicates it is not very statistically significant, as there is a very high chance our model is correct.

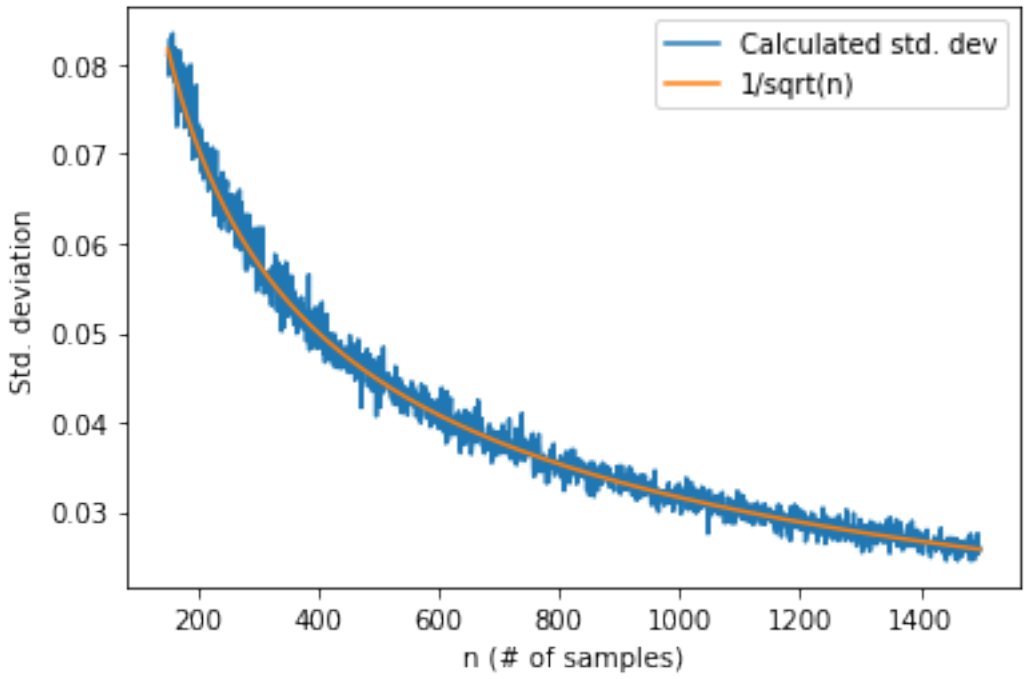
- Now repeat the above experiment for different values of n. Plot these values, and on the same plot, plot $\frac{1}{\sqrt{n}}$. How is the fit?

[551]:
```
std_devs = []
```

```
for n in range(150, 1500):
    errs = []
    for i in range(500):
        x, y = generate_data(n)
        beta = np.linalg.inv(x.T @ x) @ x.T @ y
        errs.append(beta[0] - 0) # true beta is 0, so error is just value of␣
 ↪beta_hat[0]
    std_devs.append(np.std(errs, ddof=1))
```

[550]:
```
plt.plot(range(150, 1500), std_devs, label='Calculated std. dev')
plt.plot(range(150, 1500), [1/np.sqrt(x) for x in range (150, 1500)], label='1/
 ↪sqrt(n)')
plt.legend()
plt.xlabel('n (# of samples)')
plt.ylabel('Std. deviation')
```

[550]:
```
Text(0, 0.5, 'Std. deviation')
```



The fit is very close to the graph of $\frac{1}{\sqrt{n}}$, as expected.

1.4 Question 4

- Write a program that on input k and XXXX, returns the top k names from year XXXX.

[9]:
```
import pandas as pd
import glob

def process_dataset(file):
    d = pd.read_csv(file, sep=",")
    d.columns=['Name', 'Gender', 'Frequency']
    d['Year'] = int(file.split('/')[-1][3:7])
    return d

allfiles = glob.glob('Names/*.txt')
data = pd.concat(process_dataset(file) for file in allfiles)
data.head()


def top_k_names_in_year(k, year):
    top_k = data.loc[data['Year']==year].nlargest(k, 'Frequency')
    result = {}
    for _, row in top_k.iterrows():
        result[row['Name']]= row['Frequency']
    return result

top_k_names_in_year(5, 1905)
```

[9]:
```
{'John': 8060, 'Helen': 6811, 'William': 6495, 'James': 6042, 'Margaret': 5690}
```

- Write a program that on input Name returns the frequency for men and women of the name Name.

[10]:
```
def gender_name_frequencies(name):
    return {
        'M':data.loc[(data['Name']==name) & (data['Gender']=='M')].Frequency.
 ↪sum(),
        'F':data.loc[(data['Name']==name) & (data['Gender']=='F')].Frequency.
 ↪sum()
    }

gender_name_frequencies('John')
```

[10]:
```
{'M': 5095674, 'F': 21657}
```

- It could be that names are more diverse now than they were in 1880, so that a name may be relatively the most popular, though its frequency may have been decreasing over the years. Modify the above to return the relative frequency. Note that in the next coming lectures we will learn how to quantify diversity using entropy.

[11]:
```
def top_k_names_in_year_relative_frequency(k, year):
    number_names = len(data.loc[data['Year']==year].Name.unique())
    top_k = data.loc[data['Year']==year].nlargest(k, 'Frequency')
```

```
    result = {}
    for _, row in top_k.iterrows():
        result[row['Name']]= row['Frequency']/number_names
    return result

top_k_names_in_year_relative_frequency(5, 1905)
```

[11]:
```
{'John': 2.419693785649955,
 'Helen': 2.044731311918343,
 'William': 1.9498649054338038,
 'James': 1.8138697087961573,
 'Margaret': 1.708195737015911}
```

- Find all the names that used to be more popular for one gender, but then became more popular for another gender.

[14]:
```
names = set()
def gender_name_frequency_in_year(name, year):
    m_cnt_vals = data.loc[(data['Name']==name) & (data['Gender']=='M') &␣
 ↪(data['Year']==year), 'Frequency']
    f_cnt_vals = data.loc[(data['Name']==name) & (data['Gender']=='F') &␣
 ↪(data['Year']==year), 'Frequency']
    m_cnt, f_cnt = 0,0
    if m_cnt_vals.size>0:
        m_cnt=m_cnt_vals[0]
    if f_cnt_vals.size>0:
        f_cnt=f_cnt_vals[0]
    if m_cnt>f_cnt:
        return 'M'
    else:
        return 'F'

# data.loc[(data['Name']=='John') & (data['Gender']=='M') &␣
 ↪(data['Year']==1905), 'Frequency']
def gender_popularity_shifted_names():
    #make a df of name, year male gender frequency, female gender frequency,␣
 ↪difference
    for name in data.Name.unique():
        prev = gender_name_frequency_in_year(name, 1880)
        for year in range(1881, 2016):
            curr = gender_name_frequency_in_year(name, year)
            if prev!=curr:
                names.add(name)
                break
            prev = curr
```