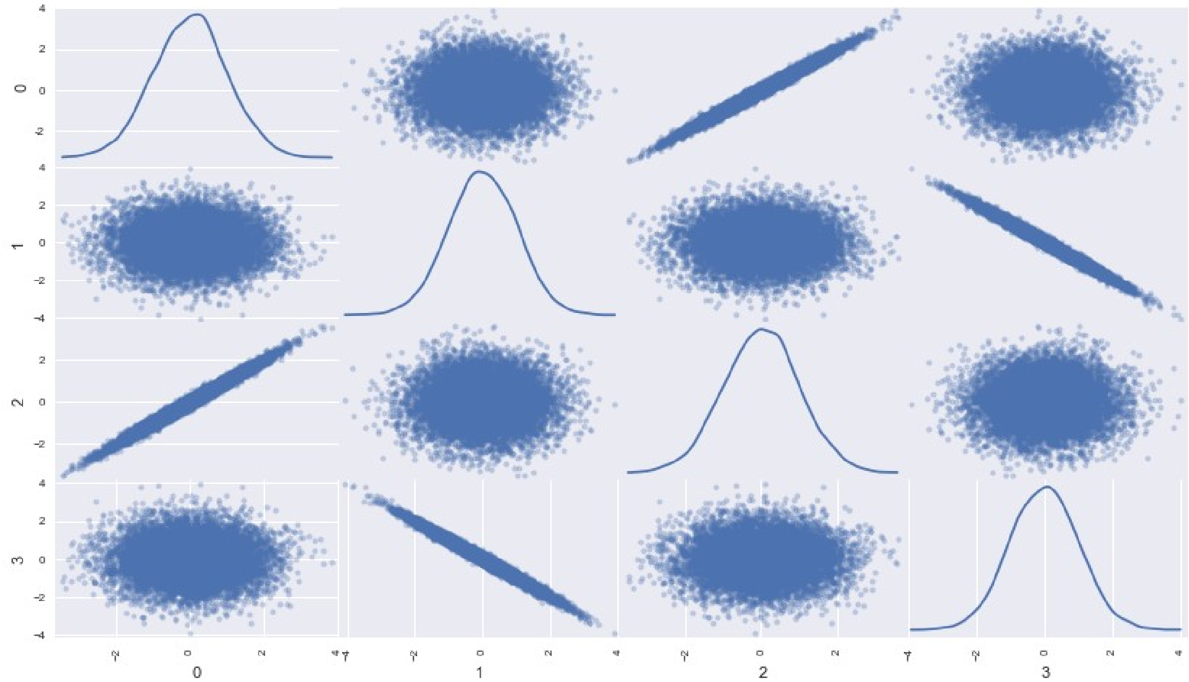
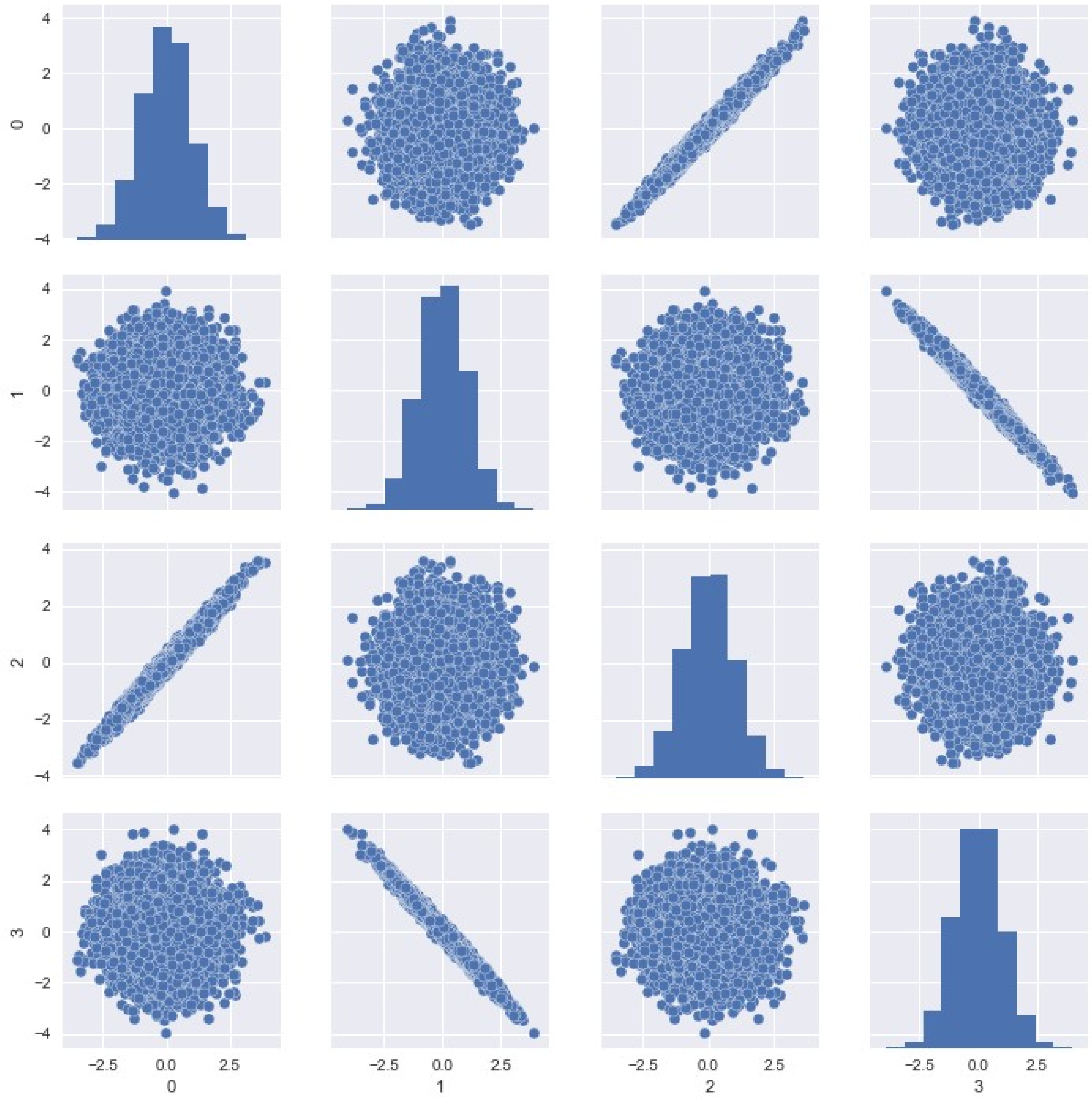
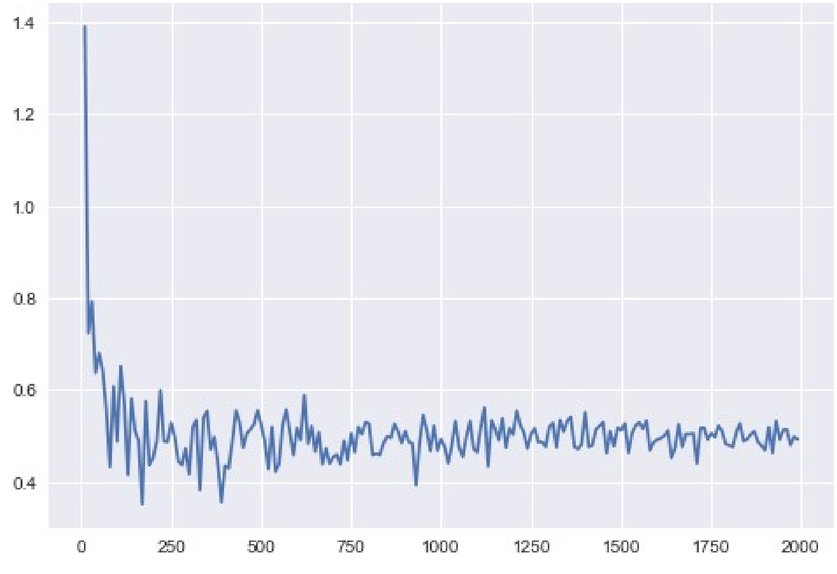
In [8]:

```
## Akaash Chikarmane, Sean Tremblay
## Programming Question 1

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn

df = pd.read_table('DF1', sep=',')#, header = None)
df = df.drop(df.columns[0], axis=1)
print(df)

#correlated_df = df.corr('pearson')

#print(correlated_df)

#This is part 2
covariance_df = np.cov(df, rowvar=False)
print(covariance_df)

print(" The numbers fit with the plot I got, because the closer the value is to bei
ng <= 1 the more correlated they are")

#Part1
pd.scatter_matrix(df, alpha = 0.3, figsize = (14,8), diagonal = 'kde')
seaborn.pairplot(df)

plt.show()

mean = [0,0,0]
cov =[[1,.2,0],[.2,1,.5],[0, .5, 1]]
#mean = [0, 0]
#cov = [[1, 0], [0, 100]]

#gives correct covariance matrix
#cov_array = np.cov(np.random.multivariate_normal(mean, cov, size=1000), rowvar=Fal
se)

#1000 is too short
covariance_x = range(10,2000,10)
covariance_y_array = []

# Going to check it versus third row second column ,
# should be .5
for i in covariance_x:
    covariance_y = np.cov(np.random.multivariate_normal(mean, cov, i) , rowvar=Fals
e)[2][1]
    covariance_y_array.append(covariance_y)

fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
ax2.plot(covariance_x, covariance_y_array)

plt.show()

print("Done")
```

```
             0         1         2         3
0     1.038502  0.899865  0.835053 -0.971528
1     0.320455 -0.647459  0.149079  0.352593
2     0.055480  2.234771  0.271672 -2.108739
3    -0.007260 -0.524299 -0.126550  0.670827
4    -1.237390 -1.377017 -1.049932  1.342079
5     0.477841  0.032660  0.336723 -0.171675
6    -0.486923 -1.128336 -0.459850  1.113013
7     0.313020  0.677323  0.123082 -0.617958
8     0.919790 -0.539665  0.956577  0.821389
9     0.574238 -1.024339  0.471622  1.006623
10   -0.745211  1.117401 -0.955933 -1.128786
11   -0.472249  1.819872 -0.660452 -1.782977
12    0.426001  1.501646  0.275335 -1.466056
13    1.529169  1.964452  1.485045 -1.950166
14    0.454290 -0.643795  0.417083  0.623628
15    2.225789 -0.015177  2.123942  0.125185
16    0.325455 -0.679482  0.589621  0.849904
17   -0.620078 -0.260013 -0.631947  0.314077
18   -0.968355 -0.576313 -0.852067  0.608772
19   -0.497867 -0.826118 -0.474271  0.877328
20    0.138424  2.346433  0.246833 -2.328777
21   -0.612432  0.867661 -0.426946 -1.218642
22   -3.201179 -0.295933 -3.000997  0.163514
23   -0.703764 -0.895308 -0.399714  0.728313
24    1.601584 -0.255763  1.724664  0.133922
25    0.191843 -0.728643  0.266744  0.769094
26    0.125723  0.471644  0.053597 -0.459898
27    0.221168  0.606079  0.386691 -0.897673
28   -0.231375 -2.708069 -0.003476  2.633477
29    0.536865 -0.276776  0.394512  0.031890
...        ...       ...       ...       ...
9970 -2.730439 -1.547178 -2.701821  1.708031
9971  0.688554 -0.370272  0.725510  0.236869
9972 -0.884338  0.889492 -0.785772 -0.823868
9973 -2.400426  0.117176 -2.372839 -0.025592
9974 -1.598474  0.268953 -1.623537 -0.362582
9975 -0.535834  0.012192 -0.441412  0.095823
9976 -1.100836  2.479797 -1.014950 -2.628093
9977 -0.008764 -0.075486  0.009927 -0.219688
9978  0.683233 -0.140395  0.633649 -0.088128
9979 -1.406525 -0.908348 -1.271741  0.882756
9980 -0.173609 -0.185395 -0.343300  0.256917
9981  0.960713 -0.128936  0.722877  0.139783
9982 -0.226978  1.026923 -0.154546 -1.041752
9983  0.550183  1.805770  0.256797 -1.447349
9984  0.447656 -0.905489  0.640393  0.796162
9985  0.165993  0.222795  0.151294 -0.023453
9986 -0.814842 -1.089027 -0.902200  1.057734
9987  0.207744  1.102626  0.116032 -1.273074
9988 -0.452244  0.129441 -0.517421 -0.328953
9989 -0.493284  0.222432 -0.717655 -0.267048
9990 -0.423431 -0.415418 -0.465615  0.494101
9991 -1.684161 -0.182262 -1.538363  0.004538
9992 -1.780461  0.910024 -1.597428 -0.772670
9993 -0.941366  0.055372 -0.657820  0.192450
9994  0.790076  1.339041  0.717082 -1.282713
9995 -0.632309 -0.145873 -0.797517  0.436184
9996  0.679417 -0.530216  0.526470  0.439397
9997  0.890697 -2.210855  1.072751  2.285372
9998  0.475293  0.490971  0.536909 -0.195772
9999  1.207406  0.819239  1.230797 -0.752397

[10000 rows x 4 columns]
```


0
1
2
3
4
2
0
-2
-4
-2
0
2
4

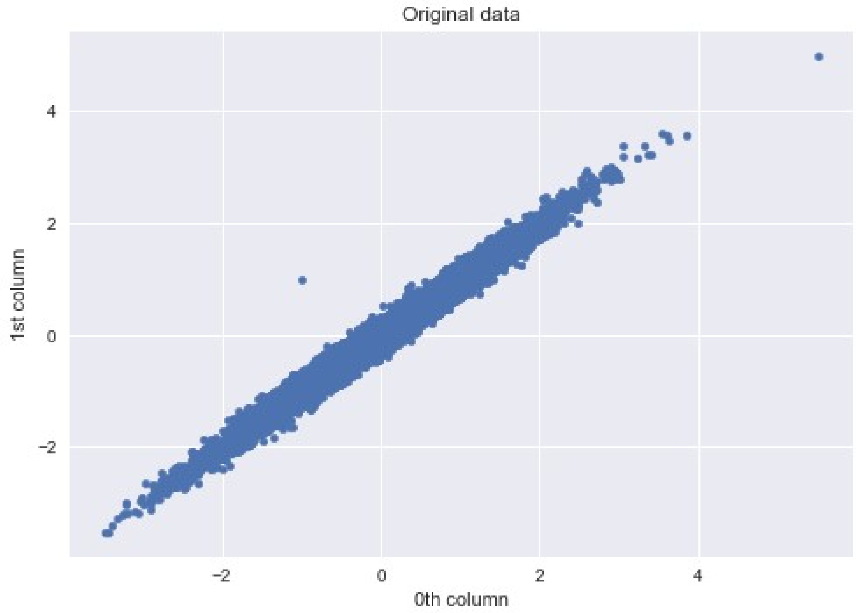

0
1
2
3
4
2
0
−2
−4
−2.5
0.0
2.5


1.4
1.2
1.0
0.8
0.6
0.4
0
250
500
750
1000
1250
1500
1750
2000

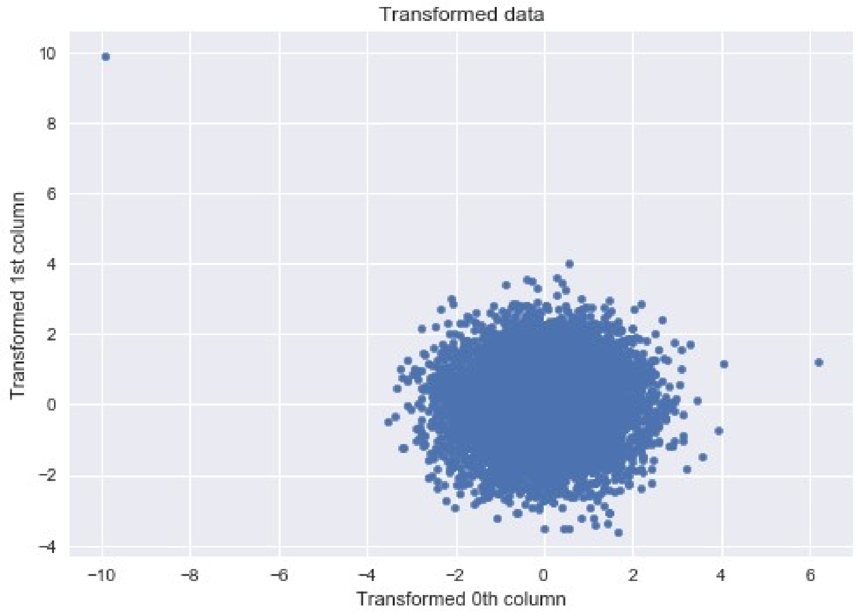
```
Done
```

In [10]:

```
## Programming Question 2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as splin

## Questions
# Which one is more outlying?
# I would say the point at (-1, 1) is more of an outlier because it is farther away
from the best fit line that the rest of the data seems to follow

# Propose a transformation:
# (sqrt(cov(input=Y)))^-1 <- transformation matrix
# Printed below

# Justify your choice of transformation
# Y = QZ <- we're after Z
# Y ~ N(mu, covariance matrix)
# var(Y) = var(QZ) = QZQ^T = covariance matrix
# If we say Z = identity matrix, we can just take the square root of the covariance
matrix of Y to get Q
# (Q^-1)Y = Z

# Initial scatter plot
df = pd.read_table('DF2', sep=',')
df.drop('Unnamed: 0', axis=1, inplace=True)
df.plot.scatter(x=0, y=1)
plt.xlabel("0th column")
plt.ylabel("1st column")
plt.title("Original data")
plt.show()

# Transformed scatter plot to show distance of each point from center. Shows that p
oint at (-1, 1) is more of an outlier than (5.5, 5)
yT = df.transpose()
outliersT = np.array([[-1, 5.5], [1, 5]])
covariance = np.cov(m=yT, rowvar=True)
Q = splin.sqrtm(A=covariance)
Q_inverse = splin.inv(a=Q)
print("Transformation Q = ", Q_inverse)
outliers_transform = np.dot(Q_inverse, outliersT)
outliersData_transform = pd.DataFrame(outliers_transform.transpose())
zT = np.dot(Q_inverse, yT)
z = pd.DataFrame(zT.transpose())
z.plot.scatter(x=0, y=1)
plt.xlabel("Transformed 0th column")
plt.ylabel("Transformed 1st column")
plt.title("Transformed data")
plt.show()

print("Below are the two outlying points. The 0th row is formerly (-1, 1). The 1st
row is formerly (5.5, 5)")
outliersData_transform.head()
```



```
Transformation Q =  [[ 5.31270488 -4.6064893 ]
 [-4.6064893   5.31496647]]
```



Below are the two outlying points. The 0th row is formerly (-1, 1). The 1st row is formerly (5.5, 5)

Out[10]:

	0	1
0	-9.919194	9.921456
1	6.187430	1.239141

In [11]:

```
## Programming Question 3

print("YEAH G")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
print("YEAH G")
from sklearn import linear_model
#from sklearn import datasets ## imports datasets from scikit-learn
#data = datasets.load_boston() ## loads Boston dataset from datasets library
import math
def standard_error_for_different_error(error_value):

    Beta_values = []
    # Akaash here's the website: https://towardsdatascience.com/simple-and-multiple
-linear-regression-in-python-c928425168f9

    lm = linear_model.LinearRegression()
    for i in range( 0 , 1000):


        Gaussian_Normal_x = np.random.randn(error_value)

        #print(Gaussian_Normal_x)
        Gaussian_Normal_e = np.random.randn(error_value)

        #print(Gaussian_Normal_e)
        # Don't need x since it s just 0's
        y_values = -3 + Gaussian_Normal_e

        # Get a column vector of x, y
        #model = lm.fit(X,y)

        lm.fit(Gaussian_Normal_x.reshape(error_value, 1), y_values)
        # SANITY CHECK:
        #lm.coef_
        #will give an output like:

        #array([ -1.07170557e-01,   4.63952195e-02,   2.08602])
        Beta_values.append(lm.coef_[0])

    standard_dev_d = np.std(Beta_values)

    return standard_dev_d

try:
    beta_hat = None

    error_range = range(5,506, 100)
    error_range.append(150)
    error_range.sort()

    empirical_standard_dev_of_error = []
    #empirical_standard_dev_of_error.append(standard_error_for_different_error(150)
)

    for error_value in error_range:

        empirical_standard_dev_of_error.append(standard_error_for_different_error(e
rror_value))

    python_plot = plt.plot(error_range, empirical_standard_dev_of_error)
    plt.show()
```

```
YEAH G
YEAH G
YEAH G'range' object has no attribute 'append'
```

In [12]:

```
## Programming Question 4
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def retrieve_file(XXXX):
    X_string = str(XXXX)
    path = ".\\Names\\" + "yob" + X_string + ".txt"
    result = pd.read_table(path, sep=',', header=None)
    return result

# input k and XXXX, returns the top k names from year XXXX
def TopKNames(k, XXXX):
    df = retrieve_file(XXXX)
    df = df.groupby(0).agg(sum) #groups by name then aggregates/sums them
    sorted = df.sort_values(by=2, ascending=False) #sorts by frequency
    print(sorted[:k])
    return sorted.nlargest(n=k, columns=2)

# input Name returns the frequency for men and women of the name Name
def NameFreq(Name, XXXX):
    df = retrieve_file(XXXX)
    rows, columns = df.shape
    result = pd.DataFrame()
    for row in range(rows):
        if(df[0][row] == Name):
            print(Name + "(" + str(df[1][row]) + ")" + " has frequency " + str(df[2
][row]))
            result = result.append(df.iloc[row], ignore_index = True)
    return result

# input Name = name to search for, XXXX = year, bPrint = boolean print (true = prin
t, false = don't print) returns the relative frequency for men and women of the nam
e Name
def NameRelFreq(Name, XXXX, bPrint):
    df = retrieve_file(XXXX)
    rows, columns = df.shape
    result = pd.DataFrame()
    for row in range(rows):
        if(df[0][row] == Name):
            if(bPrint):
                print(Name + "(" + str(df[1][row]) + ")" + " has relative frequency
" + str(np.divide(df[2][row], rows)))
            result = result.append(df.iloc[row], ignore_index = True)
    result_rows, result_cols = result.shape
    if(not result.empty):
        extra = np.array([np.divide(result[2][0], rows)], dtype=float)
    if(result_rows > 1):
        extra = np.append(extra, [np.divide(result[2][1], rows)])
    if(not result.empty):
        result[3] = extra
        result.drop(2, axis=1, inplace=True)
        result.columns=[0, 1, 2]
        result = result.sort_values(2, ascending=False).reset_index(drop=True)
    if(bPrint):
        print(result)
    return result

# Outputs names that became more popular for the other gender in XXXX vs. YYYY. Doe
s not check YYYY vs. XXXX. XXXX and YYYY can be any year between and including 1880
and 2015
def PopularityShift(XXXX=1880, YYYY=2015):
    result = []
    result = np.array(result)
```

```
                 2
0
Michael      69219
Jennifer     58591
Christopher  49401
Jason        48479
David        42212
James        39583
Matthew      38054
Joshua       36283
Amanda       35905
John         35522
Michael(F) has frequency 546
Michael(M) has frequency 68673
Michael(F) has relative frequency 0.0280878646021
Michael(M) has relative frequency 3.53274345388
         0  1         2
0  Michael  M  3.532743
1  Michael  F  0.028088
Names that shifted gender popularity between 1881 and 2014:  ['Marion' 'Sidney'
'Leslie' 'Alva' 'Ollie' 'Jimmie' 'Lou' 'Artie' 'Alpha'
 'Jean' 'Guadalupe' 'Theo']
```

```
Out[12]: array(['Marion', 'Sidney', 'Leslie', 'Alva', 'Ollie', 'Jimmie', 'Lou',
                'Artie', 'Alpha', 'Jean', 'Guadalupe', 'Theo'],
               dtype='<U32')
```

In [13]:

```
## Programming Question 5
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import math
import seaborn


tweets = pd.read_csv("tweets.csv")
tweets.head()


def get_candidate(row):
    candidates = []
    text = row["text"].lower()
    if "clinton" in text or "hillary" in text:
        candidates.append("clinton")
    if "trump" in text or "donald" in text:
        candidates.append("trump")
    if "sanders" in text or "bernie" in text:
        candidates.append("sanders")
    return ",".join(candidates)

tweets["candidate"] = tweets.apply(get_candidate,axis=1)

counts = tweets["candidate"].value_counts()
plt.bar(range(len(counts)), counts)
plt.show()

print(counts)

from datetime import datetime

tweets["created"] = pd.to_datetime(tweets["created"])
tweets["user_created"] = pd.to_datetime(tweets["user_created"])

tweets["user_age"] = tweets["user_created"].apply(lambda x: (datetime.now() - x).to
tal_seconds() / 3600 / 24 / 365)
plt.hist(tweets["user_age"])
plt.show()

# user_location ,
plt.hist(tweets["user_age"])
plt.title("Tweets mentioning candidates")
plt.xlabel("Twitter account age in years")
plt.ylabel("# of tweets")
plt.show()


cl_tweets = tweets["user_age"][tweets["candidate"] == "clinton"]
sa_tweets = tweets["user_age"][tweets["candidate"] == "sanders"]
tr_tweets = tweets["user_age"][tweets["candidate"] == "trump"]
plt.hist([
        cl_tweets,
        sa_tweets,
        tr_tweets
    ],
    stacked=True,
    label=["clinton", "sanders", "trump"]
)
plt.legend()
plt.title("Tweets mentioning each candidate")
plt.xlabel("Twitter account age in years")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-13-fd78d2c0f4dc> in <module>()
      8 
      9 
---> 10 tweets = pd.read_csv("tweets.csv")
     11 tweets.head()
     12 

C:\ProgramData\Anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(file
path_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, pref
ix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skip
initialspace, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, s
kip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser,
dayfirst, iterator, chunksize, compression, thousands, decimal, lineterminator,
quotechar, quoting, escapechar, comment, encoding, dialect, tupleize_cols, error
_bad_lines, warn_bad_lines, skipfooter, skip_footer, doublequote, delim_whitespa
ce, as_recarray, compact_ints, use_unsigned, low_memory, buffer_lines, memory_ma
p, float_precision)
    644                     skip_blank_lines=skip_blank_lines)
    645 
--> 646         return _read(filepath_or_buffer, kwds)
    647 
    648     parser_f.__name__ = name

C:\ProgramData\Anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepat
h_or_buffer, kwds)
    387 
    388     # Create the parser.
--> 389     parser = TextFileReader(filepath_or_buffer, **kwds)
    390 
    391     if (nrows is not None) and (chunksize is not None):

C:\ProgramData\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self
, f, engine, **kwds)
    728             self.options['has_index_names'] = kwds['has_index_names']
    729 
--> 730         self._make_engine(self.engine)
    731 
    732     def close(self):

C:\ProgramData\Anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(
self, engine)
    921     def _make_engine(self, engine='c'):
    922         if engine == 'c':
--> 923             self._engine = CParserWrapper(self.f, **self.options)
    924         else:
    925             if engine == 'python':

C:\ProgramData\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self
, src, **kwds)
   1388         kwds['allow_leading_cols'] = self.index_col is not False
   1389 
-> 1390         self._reader = _parser.TextReader(src, **kwds)
   1391 
   1392         # XXX

pandas\parser.pyx in pandas.parser.TextReader.__cinit__ (pandas\parser.c:4184)()

pandas\parser.pyx in pandas.parser.TextReader._setup_parser_source (pandas\parse
r.c:8449)()

FileNotFoundError: File b'tweets.csv' does not exist
```

In [ ]: