Question 1a

- Answer

Count: 1001
Mean: 31.414585
Standard Deviation: 1.397672
Minimum: 26.3
25th Percentile: 30.4
Median: 31.5
75th Percentile: 32.4
Maximum: 35.4

- Command Prompt Output

```
#load necessary libraries
import pandas as pd
import numpy as np

#read in NormalSample.csv into df
df = pd.read_csv('/Users/tiffwong/Desktop/cs484/assignments/assignment 1/NormalSample.csv')

#%% question 1a
print('-- Question 1 --')
print(df.describe())
```

- Figure

```
-- Question 1 --
count    1001.000000
mean       31.414585
std         1.397672
min        26.300000
25%        30.400000
50%        31.500000
75%        32.400000
max        35.400000
```

- Explanation

I loaded the necessary libraries in and read in the NormalSample.csv file using pandas. Then I used the Pandas describe() function to find the items asked in 1a's question.

Question 1b

- Answer

The bin width recommended by the Izenman (1991) method, rounded to the nearest tenths, is h = 0.4.

- Command Prompt Output

```
-- Question 1b --
bin width using Izenman method is: 0.3998667554864774
```

- Explanation

The Izenman method to find bin width is $h = 2(IQR)N^{-1/3}$ where N is the number of observations and IQR is the Interquartile Range. Based off the five-number summary I got from #1a, I used the 25% quartile and 75% quartile to find the IQR value and since I know there's a count of 1001 values, I used that as the value of N. I coded the equation with the variables IQR and N, and I get that h = 0.3998667554864774.

Question 1c

- Answer

The recommended bin width is when d=1.0. At this value of d, C(d) is the lowest.

- Code/Calculations

```
#%% question 1c
print('-- Question 1c --')

Y = df['x']
def calcCD (Y, delta):
   maxY = np.max(Y)
   minY = np.min(Y)
   meanY = np.mean(Y)

   # Round the mean to integral multiples of delta
   middleY = delta * np.round(meanY / delta)

   # Determine the number of bins on both sides of the rounded mean
   nBinRight = np.ceil((maxY - middleY) / delta)
   nBinLeft = np.ceil((middleY - minY) / delta)
   lowY = middleY - nBinLeft * delta

   # Assign observations to bins starting from 0
   m = nBinLeft + nBinRight
   BIN_INDEX = 0;
   boundaryY = lowY
   for iBin in np.arange(m):
      boundaryY = boundaryY + delta
      BIN_INDEX = np.where(Y > boundaryY, iBin+1, BIN_INDEX)

   # Count the number of observations in each bins
   uBin, binFreq = np.unique(BIN_INDEX, return_counts = True)

   # Calculate the average frequency
   meanBinFreq = np.sum(binFreq) / m
   ssDevBinFreq = np.sum((binFreq - meanBinFreq)**2) / m
   CDelta = (2.0 * meanBinFreq - ssDevBinFreq) / (delta * delta)
   return(m, middleY, lowY, CDelta)

result = pd.DataFrame()
deltaList = [0.1, 0.2, 0.5, 1.0, 2.0, 5.0]

for d in deltaList:
   nBin, middleY, lowY, CDelta = calcCD(Y,d)
   highY = lowY + nBin * d
   result = result.append([[d, CDelta, lowY, middleY, highY, nBin]], ignore_index = True)

result = result.rename(columns = {0:'Delta', 1:'C(Delta)', 2:'Low Y', 3:'Middle Y',
                                  4:'High Y', 5:'N Bin'})
print(result)
```

- Command Prompt Output

```
-- Question 1c --
   Delta     C(Delta)  Low Y  Middle Y  High Y  N Bin
0    0.1 -6773.416311   26.2      31.4    35.4   92.0
1    0.2 -7964.759185   26.2      31.4    35.4   46.0
2    0.5 -8930.627205   26.0      31.5    35.5   19.0
3    1.0 -9538.290000   26.0      31.0    36.0   10.0
4    2.0 -7963.040000   26.0      32.0    36.0    5.0
5    5.0 -4946.862222   25.0      30.0    40.0    3.0
```

- Explanation

I created a function, calcCD(Y, delta), that took a dataframe and a value of delta imposed onto it. In it, I found the mean (n) and variance (v) of the number of observations using the summation formula for the two items, and I computed the formula for $C(d) = (2*n - v)/ (d^2)$. Then I printed the delta values along with their respective C(Delta) values to determine which value of delta was the best to use. I determined this by finding the d that minimizes C(d) because that would be the optimal bin width, which turned out to be d=1.0

Question 1d

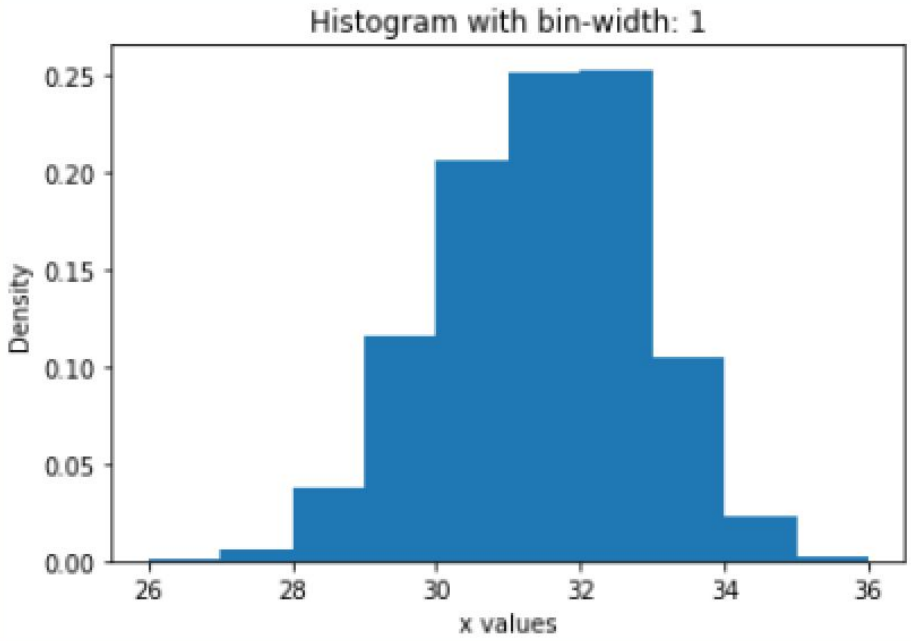
- Answer

The midpoints are: 26.5, 27.5, 28.5, 29.5, 30.5, 31.5, 32.5, 33.5, 34.5, 35.5 and the estimated density function values are 0.00100, 0.00599, 0.03796, 0.11588, 0.20579, 0.25075, 0.25275, 0.10490, 0.02298, 0.0020.

- Command Prompt Output

```
midpoints  estimated density function values
     26.5                                0.00100
     27.5                                0.00599
     28.5                                0.03796
     29.5                                0.11588
     30.5                                0.20579
     31.5                                0.25075
     32.5                                0.25275
     33.5                                0.10490
     34.5                                0.02298
     35.5                                0.00200
```

- Figure



- Explanation

Since my answer from 1c was that the recommended bin width should be d=1.0, the mid-points are: 26.5, 27.5, 28.5, 29.5, 30.5, 31.5, 32.5, 33.5, 34.5, 35.5. To find the estimated density function values, I applied the density estimate method from the slides and evaluated when density is true. The estimated density function values are: 0.00100, 0.00599, 0.03796, 0.11588, 0.20579, 0.25075, 0.25275, 0.10490, 0.02298, 0.0020. Then I used the histogram function in the matplotlib module to plot the density estimator as a vertical bar chart.

Question 2a

- Answer

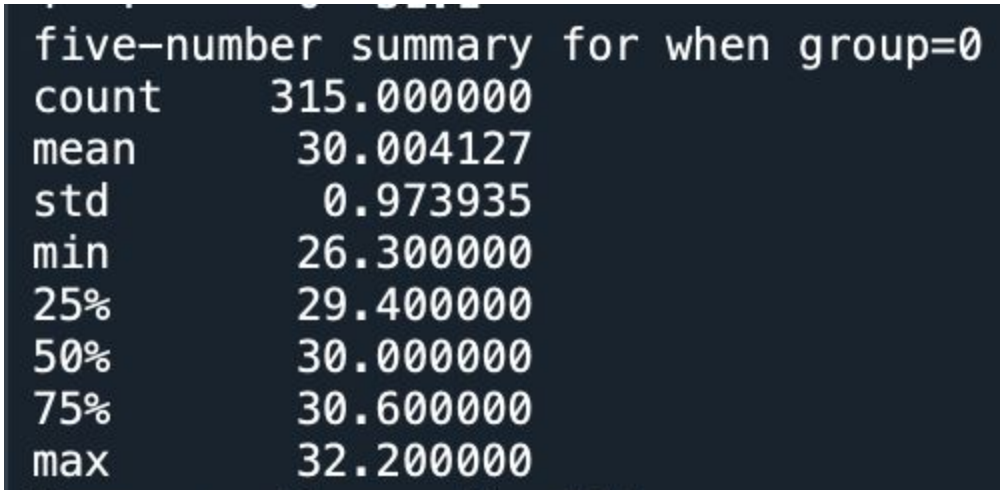
Category: group = 0

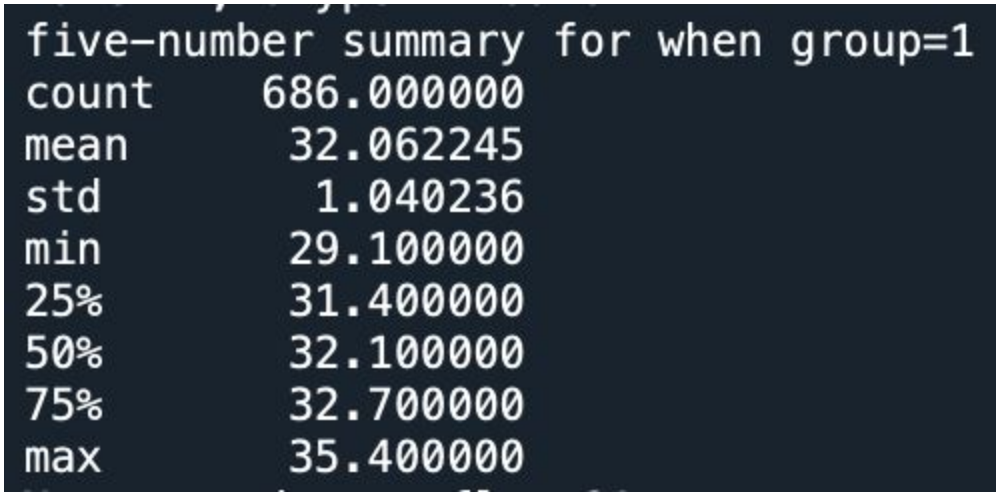
Minimum: 26.3
25th Percentile: 29.4
Median: 30.0
75th Percentile: 30.6
Maximum: 32.2
Upper whisker: 32
Lower whisker: 27

Category: group = 1

Minimum: 29.1
25th Percentile: 31.4
Median: 32.1
75th Percentile: 32.7
Maximum: 35.4
1.5IQR: 1.9500000000000064
Upper whisker: 34
Lower whisker: 29

- Command Prompt Output





- Explanation

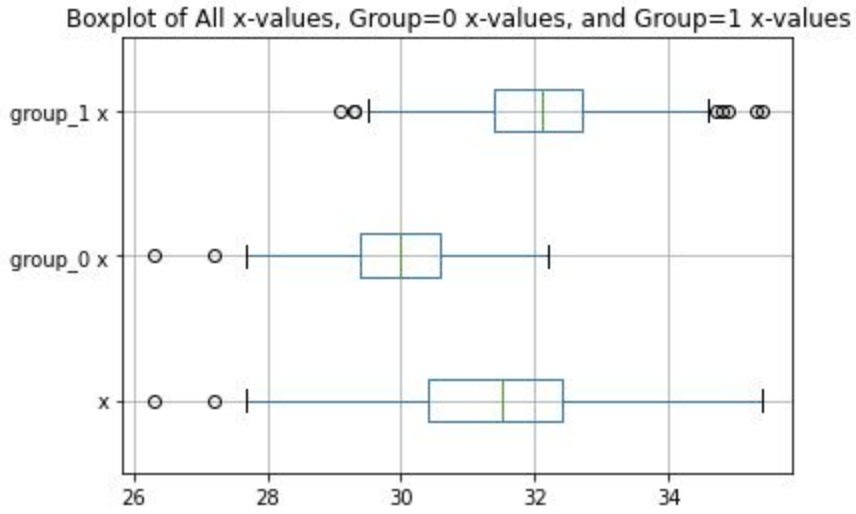
I filtered out the occasions when the NormalSample.csv file has group=0 and group=1. I grabbed the x values in those individual dataframes, and used the Pandas describe() function to get the five-number summary. After that, I used the 25th percentile and the 75th percentile to calculate the IQR for each of the two sets, and then used it to get the upper and lower whiskers.

Question 2b

- Answer

For the boxplot on the group=1 x-values, there are 5 outliers above the upper whisker and 2 outliers below the lower whisker.
For the boxplot on the group=0 x-values, there are 2 outliers below the lower whisker.
For the boxplot on the actual x-values from the NormalSample.csv, there are 2 outliers below the lower whisker.

- Figure



- Explanation

I used the series of group=0 and group=1's x-values I had from question 2a, then I turned them into dataframes. Next, I used the rename function to rename the columns of 'x' into something more specific to differentiate them from the 'x' column in the actual csv file. Next, I used the concat function to combine all three data frames into 1. Finally, I used the boxplot function to plot the three columns into three boxplots on the same graph, and made sure to make it horizontal.

Question 3a

- Answer

19.9497% of investigations are found to be frauds.

- Command Prompt Output

- Explanation

I read in the Fraud.csv file to a dataframe, then I filtered out the number of instances where the column FRAUD=1, because that's when the case is a fraud according to question 3's prompt. Then I divided the length of the fraud=1 data frame by the length of the original data frame, which got me 19.949664429530202%, so I rounded it to the 4th decimal place.

Question 3b

- Answer

After orthonormalizing the interval variables, I checked if the eigenvalues are greater than 1, which they all are, shown below.

i. There are 6 dimensions used, one for each interval variable: TOTAL_SPEND, DOCTOR_VISITS, NUM_CLAIMS, MEMBER_DURATION, OPTOM_PRESC, and NUM_MEMBERS.

ii. To check whether the orthonormalized columns are actually orthonormal, I transposed the transformed matrix and multiplied it with the actual transformed matrix itself. If the orthonormalized columns are actually orthonormal, then the result of this matrix multiplication should result in an identity matrix. The orthonormalized columns are indeed orthonormal as shown below when the result is an identity matrix.

- Command Prompt Output

Eigenvalues and Eigenvectors

```
Eigenvalues of x =
 [6.84728061e+03 8.38798104e+03 1.80639631e+04 3.15839942e+05
 8.44539131e+07 2.81233324e+12]
Eigenvectors of x =
 [[-5.37750046e-06 -2.20900379e-05  3.62806809e-05 -1.36298664e-04
  -7.26453432e-03  9.99973603e-01]
 [ 6.05433402e-03 -2.69942162e-02  1.27528313e-02  9.99013423e-01
   3.23120126e-02  3.69879256e-04]
 [-9.82198935e-01  1.56454700e-01 -1.03312781e-01  1.14463687e-02
   1.62110700e-03  1.52596881e-05]
 [ 1.59310591e-04 -4.91894718e-03  3.11864824e-03 -3.25018102e-02
   9.99428355e-01  7.25592222e-03]
 [ 6.90939783e-02 -2.10615119e-01 -9.75101628e-01  6.26672294e-03
   2.19857585e-03  4.79234486e-05]
 [ 1.74569737e-01  9.64577791e-01 -1.95782843e-01  2.73038995e-02
   6.21788707e-03  7.82430481e-05]]
checking if eigenvalues are >1: [ True  True  True  True  True  True]
```

Transformation Matrix:

```
Transformation Matrix =
 [[-6.49862374e-08 -2.41194689e-07  2.69941036e-07 -2.42525871e-07
  -7.90492750e-07  5.96286732e-07]
 [ 7.31656633e-05 -2.94741983e-04  9.48855536e-05  1.77761538e-03
   3.51604254e-06  2.20559915e-10]
 [-1.18697179e-02  1.70828329e-03 -7.68683456e-04  2.03673350e-05
   1.76401304e-07  9.09938972e-12]
 [ 1.92524315e-06 -5.37085514e-05  2.32038406e-05 -5.78327741e-05
   1.08753133e-04  4.32672436e-09]
 [ 8.34989734e-04 -2.29964514e-03 -7.25509934e-03  1.11508242e-05
   2.39238772e-07  2.85768709e-11]
 [ 2.10964750e-03  1.05319439e-02 -1.45669326e-03  4.85837631e-05
   6.76601477e-07  4.66565230e-11]]
```

The Transformed x:

```
The Transformed x =
 [[ 5.96859502e-03  1.02081629e-02 -6.64664861e-03  1.39590283e-02
   9.39352141e-03  6.56324665e-04]
 [-2.09672310e-02  5.01932025e-03  8.51930607e-04  5.16174400e-03
   1.22658834e-02  7.75702220e-04]
 [ 7.64597676e-03  1.97528525e-02 -7.38335310e-03 -1.71350853e-03
   1.50348109e-02  8.95075830e-04]
 ...
 [-7.18408819e-05 -1.62580211e-02  2.75078514e-02 -7.13245766e-03
  -4.74021952e-02  5.31896971e-02]
 [-1.80147801e-04 -1.62154130e-02  2.76213381e-02 -9.17125411e-03
  -4.76625006e-02  5.35474776e-02]
 [-2.21157680e-03 -2.73884697e-02  2.93391341e-02 -7.81347172e-03
  -4.70861917e-02  5.36071324e-02]]
```

Expect an Identity Matrix:

```
Expect an Identity Matrix =
 [[ 1.00000000e+00 -2.87703888e-15  1.90299165e-15  7.06552872e-15
   1.16226473e-15 -1.35308431e-16]
 [-2.87703888e-15  1.00000000e+00 -1.37216627e-15 -1.98244199e-14
  -6.59194921e-16  7.21644966e-16]
 [ 1.90299165e-15 -1.37216627e-15  1.00000000e+00  4.96366728e-15
  -6.24500451e-17 -1.17961196e-16]
 [ 7.06552872e-15 -1.98244199e-14  4.96366728e-15  1.00000000e+00
   1.10432496e-14 -4.20496971e-15]
 [ 1.16226473e-15 -6.59194921e-16 -6.24500451e-17  1.10432496e-14
   1.00000000e+00 -6.66133815e-16]
 [-1.35308431e-16  7.21644966e-16 -1.17961196e-16 -4.20496971e-15
  -6.66133815e-16  1.00000000e+00]]
```

- Explanation

Since CASE_ID and FRAUD is not an interval variable, I can remove those columns when putting the csv file data into a dataframe. Then I transposed the matrix, and used the Linalg module from the Numpy library to find the eigenvalues and eigenvectors of the transposed matrix. Next, I checked if the eigenvalues of the interval variables are greater than 1. Then I created the transformation matrix by finding the reciprocal of the square-rooted eigenvalues and then multiplying those values by the eigenvectors from earlier. The transformed x matrix is found by multiplying the transformation matrix by the original matrix from the fraud csv file. After that, I checked if the orthonormalized columns are really orthonormal by multiplying the transpose of the transformed matrix by the transformed matrix itself and got back a identity matrix.

Question 3c

- Answer

i. According to the scikit-learn website, the score function's purpose is to "return the mean accuracy on the given test data and labels". In this problem's context, the purpose of the score function is to find the accuracy of the Fraud dataset.
ii. The score function gives me a result of 0.8778523489932886. This means that the model I created with the NearestNeighbors module based on the transformation matrix and the FRAUD column of the original dataset is around 87.79% accurate on training data.

- Code

```
#%% question 3c

from sklearn.neighbors import KNeighborsClassifier as kNC

#specify the target: FRAUD = 1, not FRAUD = 0
target = df['FRAUD']

#use kNN module and specify that neighbors=5
neigh = kNC(n_neighbors = 5, algorithm = 'brute', metric = 'euclidean')

#training the model
nbrs = neigh.fit(transf_x, target)

print("The score function gives me: ", nbrs.score(transf_x, target))
```

- Command Prompt Output

```
The score function gives me:  0.8778523489932886
```

- Explanation

First I specified that the target is the FRAUD column in the original dataset, and then I used the kNeighborsClassifier module to specify the number of neighbors to 5, given in the problem statement. Next, I trained the model using the fit function with the transformed matrix and the target matrix from earlier. Finally, I used the score function to judge the accuracy of the model, which gave me 87.79%.

Question 3d

- Answer

Based on the given input variables, the five neighbor values are [588, 2897, 1199, 1246, 886]. The input variables are when TOTAL_SPEND = 7500, DOCTOR_VISITS = 15, NUM_CLAIMS = 3, MEMBER_DURATION = 127, OPTOM_PRESC = 2, and NUM_MEMBERS = 2 equal those specified values. The target variable is FRAUD=1 at those input variable values.

- Code

```
#%% question 3d
print("-- QUESTION 3d --")

# Find the nearest neighbors of these focal observations
focal = [[7500, 15, 3, 127, 2, 2]]

#multiply focal matrix by the transformation matrix
t_focal = focal * transf;

##use kNN module and specify that neighbors=5
myNeighbors = nbrs.kneighbors(t_focal, n_neighbors = 5, return_distance = False)
print("My Neighbors = \n", myNeighbors)
```

- Command Prompt Output

```
-- QUESTION 3d --
My Neighbors =
 [[ 588 2897 1199 1246  886]]
```

- Explanation

First I set a matrix, focal, with the specified input variable values in the problem statement. Then I multiplied the focal matrix by the transformation matrix from before. Next, I used the kneighbors() function to use the model from question 3c to find its 5 neighbors, which is [588, 2897, 1199, 1246, 886].

Question 3e

- Answer

The predicted probability of fraud (when FRAUD=1) is 100% of the time. If the predicted probability is greater than or equal to the answer in question 3a, then the observation will be classified as a fraud, otherwise, not a fraud. Based on this criterion, the observation is not misclassified since 100%>19.49%.

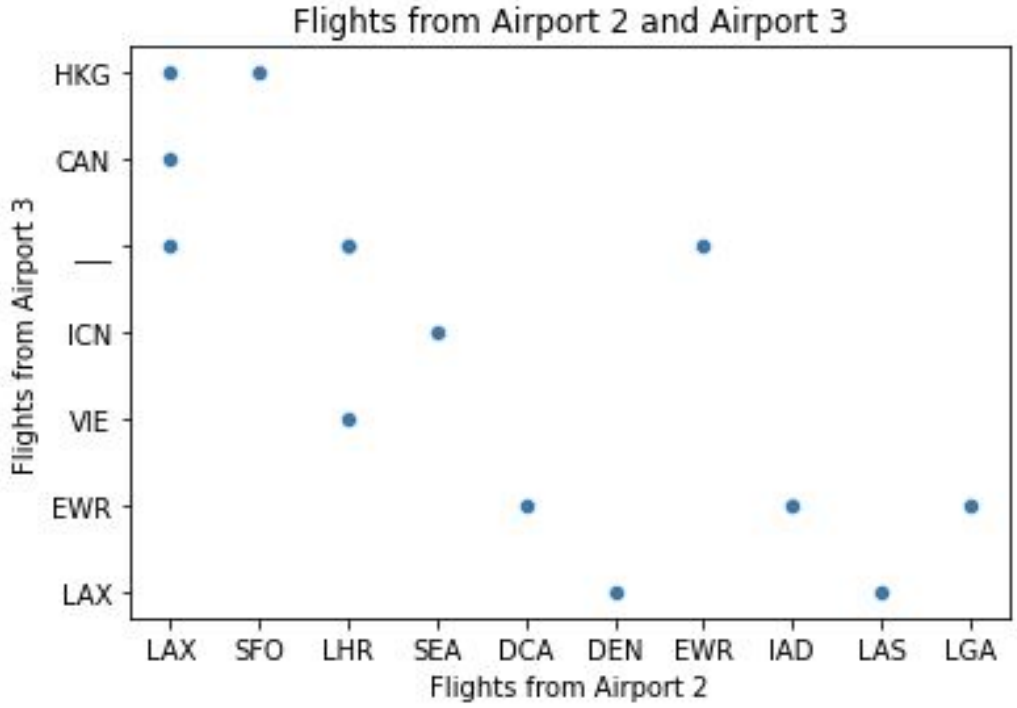
- Command Prompt Output

- Explanation

I used the function predict_proba() to pass the transformed focal matrix in and predict it based on the mode lI made in question 3c. I get that the predicted probability of fraud is [0 1], this means that there's a 0% probability that FRAUD=0 and a 100% probability that FRAUD=1.

Question 4a

- Answer/Figure



- Explanation

First I made a csv file with the data given called airport.csv, then I loaded it in as a data frame. I used the Seaborn library to plot a scatter plot with the x-axis as Airport 2 data and the y-axis as Airport 3 data.

Question 4b

- Answer/Figure

	Count
LAX	5
___	5
LHR	4
EWR	4
HKG	2
IAD	1
SFO	1
DCA	1
VIE	1
DEN	1
ICN	1
LGA	1
SEA	1
LAS	1
CAN	1

- Explanation

First I pulled out columns Airport 2 and Airport 3 from the original dataframe I read the whole csv file into, and combined them into one dataframe itself. Next, I used the crosstab() function to find the frequency count of each airport code.

Question 4c

- Answer

The flights that most resemble this new flight are: A, C, F, J, M, and N. (explained further in 4d)

- Command Prompt Output

i - iii.

```
Flight  ...                                                 vectors
     A  ...  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
     B  ...  [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
     C  ...  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
     D  ...  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
     E  ...  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
     F  ...  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
     G  ...  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
     H  ...  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
     I  ...  [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
     J  ...  [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
     K  ...  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]
     L  ...  [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
     M  ...  [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
     N  ...  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
     O  ...  [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]
```

iv.

```
Flight  ... cosine
     A  ...    0.5
     B  ...    1.0
     C  ...    0.5
     D  ...    1.0
     E  ...    1.0
     F  ...    0.5
     G  ...    1.0
     H  ...    1.0
     I  ...    1.0
     J  ...    0.5
     K  ...    1.0
     L  ...    1.0
     M  ...    0.5
     N  ...    0.5
     O  ...    1.0
   new  ...    0.0
```

- Explanation

I got all the unique codes from the dataframe I made previously from 4b for Airports 2 and 3, created an empty vector in the original dataframe to store the vector of word counts in, and then I looped through the length of the dataframe and all the unique codes I had to fill in the vectors for each flight. Next, I typed in the CosineD() function given to us in the class slides, created a column to have 0s in it to store the Cosine distances, and then I used a for loop to calculate the cosine distances from each of the flight vectors to the new flight.

Question 4d

- Answer & Explanation

The flights that have the shortest Cosine Distance from the new flight are flights: A, C, F, J, M, and N. The Cosine distances between all the flights and the new flight ranged from 0.5 to 1, so the ones that most resembled the new flight are the ones with a distance of 0.5

- Command Prompt Output

```
Flight  ... cosine
     A  ...    0.5
     B  ...    1.0
     C  ...    0.5
     D  ...    1.0
     E  ...    1.0
     F  ...    0.5
     G  ...    1.0
     H  ...    1.0
     I  ...    1.0
     J  ...    0.5
     K  ...    1.0
     L  ...    1.0
     M  ...    0.5
     N  ...    0.5
     O  ...    1.0
   new  ...    0.0
```

- Code

```
#given Cosine function
def CosineD (x, y):
    normX = np.sqrt(np.dot(x, x))
    normY = np.sqrt(np.dot(y, y))
    if (normX > 0.0 and normY > 0.0):
        outDistance = 1.0 - np.dot(x, y) / normX / normY
    else:
        outDistance = np.NaN
    return (outDistance)

#create new array for the new flight
new_f = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

#create a column for cosine values to be put in
df['cosine'] = np.empty((len(df),0)).tolist()

#for loop to calcualte the cosine distance from vector to new_f and store in df
for i in range(len(df)):
    df['cosine'][i] = CosineD(df['vectors'][i], new_f)
```