Assignment 1 - Probability, Linear Algebra, Programming, and Git

January 21, 2020

Felipe Buchbinder
Netid: *fb91*

Instructions for all assignments can be found here, which is also linked to from the course syllabus.

1 Probability and Statistics Theory

Note: for all assignments, write out all equations and math using markdown and LaTeX. For this section of the assignment (Probability and Statistics Theory) show and type up ALL math work

1.1 Question 1

[3 points]

Let $f(x) = \begin{cases} 0 & x < 0 \\ \alpha x^2 & 0 \leq x \leq 2 \\ 0 & 2 < x \end{cases}$

For what value of α is $f(x)$ a valid probability density function?

ANSWER

In order for a function to be a valid probability density function, it must 1. be non-negative; and 2. integrate to unity over its entire domain.

The latter means that

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

In our case, this means

$$\int_0^2 \alpha x^2 dx = 1$$

which implies

$$\alpha = \frac{1}{\int_0^2 x^2 dx}$$

. Since $\int_0^2 x^2 dx = \frac{8}{3}$, it follows that

$$\alpha = \frac{3}{8}$$

Note that since $\alpha > 0$, the condition that $f(x)$ be non-negative is also guaranteed. Hence, for $\alpha = \frac{3}{8}$, $f(x)$ is a valid probability density function.

1.2 Question 2

[3 points] What is the cumulative distribution function (CDF) that corresponds to the following probability distribution function? Please state the value of the CDF for all possible values of x.

$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$

ANSWER

The CDF is, by definition,

$$F(x) = \int_{-\infty}^{x} f(s)ds = \int_{0}^{x} \frac{1}{3}ds = \frac{x}{3}$$

for $0 < x < 3$. More correctly, one should write the CDF as follows:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x}{3} & 0 < x \leq 3 \\ 1 & x > 3 \end{cases}$$

Note that $F(x)$ is never decreasing and has horizontal assymptotes $lim_{x\to-\infty}F(x) = 0$ and $lim_{x\to+\infty}F(x) = 1$, indicating that it is a valid CDF.

1.3 Question 3

[6 points] For the probability distribution function for the random variable X,

$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$

what is the (a) expected value and (b) variance of X. *Show all work*.

ANSWER

(a) Expected value

By definition,

$$E(X) = \int_{-\infty}^{+\infty} x \cdot f(x)dx$$

This implies

$$E(X) = \int_{0}^{3} x \cdot \frac{1}{3}dx = \frac{1}{3} \cdot \int_{0}^{3} xdx = \frac{1}{3} \cdot \frac{9}{2} = \frac{3}{2}$$

(b) Variance

We'll make use of the identity

$$V(X) = E(X^2) - [E(X)]^2$$

. We already know the value of $E(X)$ but we must still calculate the value of $E(X^2)$. It is:

$$E(X^2) = \int_{0}^{3} x^2 \cdot \frac{1}{3}dx = \frac{1}{3} \cdot \int_{0}^{3} x^2dx = \frac{1}{3} \cdot \frac{27}{3} = 3$$

This leads us to a variance equal to

$$V(X) = 3 - \left(\frac{3}{2}\right)^2 = 3 - \frac{9}{4} = \frac{3}{4}$$

Hence, to summarise,

$$E(X) = \frac{3}{2}$$

and

$$V(X) = \frac{3}{4}$$

1.4 Question 4

[6 points] Consider the following table of data that provides the values of a discrete data vector **x** of samples from the random variable X, where each entry in **x** is given as x_i.

Table 1. Dataset N=5 observations

	x_0	x_1	x_2	x_3	x_4
x	2	3	10	-1	-1

What is the (a) mean, (b) variance, and the of the data?

Show all work. Your answer should include the definition of mean, median, and variance in the context of discrete data.

ANSWER

(a) Mean

The mean is, by definition,

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

So, by merely plugging the values in the formula, one gets:

$$\bar{X} = \frac{2 + 3 + 10 - 1 - 1}{5} = \frac{13}{5} = 2.6$$

(b) Variance

The variance is defined as

$$s^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{X})^2}{N - 1}$$

So, again, by merely plugging in the values in the formula, we obtain

$$s^2 = \frac{(2 - 2.6)^2 + ... + (-1 - 2.6)^2}{5 - 1} = 20.3$$

1.5 Question 5

[8 points] Review of counting from probability theory.

(a) How many different 7-place license plates are possible if the first 3 places only contain letters and the last 4 only contain numbers?

(b) How many different batting orders are possible for a baseball team with 9 players?

(c) How many batting orders of 5 players are possible for a team with 9 players total?

(d) Let's assume this class has 26 students and we want to form project teams. How many unique teams of 3 are possible?

Hint: For each problem, determine if order matters, and if it should be calculated with or without replacement.

ANSWER

(a)

$$26^3 \cdot 10^4$$

In this case, the order matters and calculations should be made with replacement.

(b)

$$9!$$

In this case, the order matters and calculations should be made without replacement.

(c)

$$\frac{9!}{4!}$$

In this case, the order matters and calculations should be made without replacement.

(d)

$$\frac{26!}{3! \cdot 23!}$$

In this case, the order does not matter and calculations should be made without replacement.

2 Linear Algebra

2.1 Question 6

[7 points] Matrix manipulations and multiplication. Machine learning involves working with many matrices, so this exercise will provide you with the opportunity to practice those skills.

Let $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 4 \\ -3 \\ 6 \end{bmatrix}$, and $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Compute the following or indicate that it cannot be computed:

1. $\mathbf{AA}$
2. $\mathbf{AA}^T$
3. $\mathbf{Ab}$
4. $\mathbf{Ab}^T$
5. $\mathbf{bA}$
6. $\mathbf{b}^T\mathbf{A}$
7. $\mathbf{bb}$
8. $\mathbf{b}^T\mathbf{b}$
9. $\mathbf{bb}^T$
10. $\mathbf{b} + \mathbf{c}^T$
11. $\mathbf{b}^T\mathbf{b}^T$
12. $\mathbf{A}^{-1}\mathbf{b}$
13. $\mathbf{A} \circ \mathbf{A}$
14. $\mathbf{b} \circ \mathbf{c}$

Note: The element-wise (or Hadamard) product is the product of each element in one matrix with the corresponding element in another matrix, and is represented by the symbol "$\circ$".

ANSWER

[36]:
```python
import numpy as np
import numpy.linalg as la

#We begin by defining the matrices we will be using
A = np.matrix([[1,2,3],[2,4,5],[3,5,6]])
b = np.matrix([[-1, 3, 8]]).T
c = np.matrix([[4, -3, 6]]).T
I = np.diag([1,1,1])

#Now we define a function to multiply the matrices, if possible,
#or to explain to us why the multiplication cannot be done, if otherwise
def mmult(m1, m2):
    if m1.shape[1] == m2.shape[0]:
        return(m1 * m2)
    else:
        print("These two matrices cannot be multiplied, because the first one␣
↪has {} columns whereas the second has {} rows".format(m1.shape[1], m2.
↪shape[0]))
        return(None)
```

[37]:
```python
from IPython.display import Markdown, display

display(Markdown("1. $\mathbf{A}\mathbf{A}$"))
print(mmult(A,A))

display(Markdown("2. $\mathbf{A}\mathbf{A}^T$"))
print(mmult(A,A.T))

display(Markdown("3. $\mathbf{A}\mathbf{b}$"))
print(mmult(A, b))

display(Markdown("4. $\mathbf{A}\mathbf{b}^T$"))
print(mmult(A,b.T))

display(Markdown("5. $\mathbf{b}\mathbf{A}$"))
print(mmult(b,A))

display(Markdown("6. $\mathbf{b}^T\mathbf{A}$"))
print(mmult(b.T,A))

display(Markdown("7. $\mathbf{b}\mathbf{b}$"))
print(mmult(b,b))

display(Markdown("8. $\mathbf{b}^T\mathbf{b}$"))
print(mmult(b.T,b))

display(Markdown("9. $\mathbf{b}\mathbf{b}^T$"))
print(mmult(b,b.T))

display(Markdown("10. $\mathbf{b} + \mathbf{c}^T$"))
print(mmult(b,c.T))

display(Markdown("11. $\mathbf{b}^T\mathbf{b}^T$"))
print(mmult(b.T,b.T))

display(Markdown("12. $\mathbf{A}^{-1}\mathbf{b}$"))
print(mmult(la.inv(A),b))

display(Markdown("13. $\mathbf{A}\circ\mathbf{A}$"))
print(np.multiply(A,A))

display(Markdown("14. $\mathbf{b}\circ\mathbf{c}$"))
print(np.multiply(b,c))
```

1. $\mathbf{A}\mathbf{A}$

```
[[14 25 31]
 [25 45 56]
```

```
 [31 56 70]]
```

2. $\mathbf{AA}^T$

```
[[14 25 31]
 [25 45 56]
 [31 56 70]]
```

3. $\mathbf{Ab}$

```
[[29]
 [50]
 [60]]
```

4. $\mathbf{Ab}^T$

```
These two matrices cannot be multiplied, because the first one has 3 columns
whereas the second has 1 rows
None
```

5. $\mathbf{bA}$

```
These two matrices cannot be multiplied, because the first one has 1 columns
whereas the second has 3 rows
None
```

6. $\mathbf{b}^T\mathbf{A}$

```
[[29 50 60]]
```

7. $\mathbf{bb}$

```
These two matrices cannot be multiplied, because the first one has 1 columns
whereas the second has 3 rows
None
```

8. $\mathbf{b}^T\mathbf{b}$

```
[[74]]
```

9. $\mathbf{bb}^T$

```
[[ 1 -3 -8]
 [-3  9 24]
 [-8 24 64]]
```

10. $\mathbf{b} + \mathbf{c}^T$

```
[[ -4   3  -6]
 [ 12  -9  18]
 [ 32 -24  48]]
```

11. $\mathbf{b}^T\mathbf{b}^T$

```
These two matrices cannot be multiplied, because the first one has 3 columns
whereas the second has 1 rows
None
```

12. $\mathbf{A}^{-1}\mathbf{b}$

```
[[ 6.]
 [ 4.]
 [-5.]]
```

13. $\mathbf{A} \circ \mathbf{A}$

```
[[ 1  4  9]
 [ 4 16 25]
 [ 9 25 36]]
```

14. $\mathbf{b} \circ \mathbf{c}$

```
[[-4]
 [-9]
 [48]]
```

2.2 Question 7

[8 points] Eigenvectors and eigenvalues. Eigenvectors and eigenvalues are useful for some machine learning algorithms, but the concepts take time to solidly grasp. For an intuitive review of these concepts, explore this interactive website at Setosa.io. Also, the series of linear algebra videos by Grant Sanderson of 3Brown1Blue are excellent and can be viewed on youtube here.

1. Calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{A}$ above, from the last question.
2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. Also show that this relationship extends to higher orders: $\mathbf{A}\mathbf{A}\mathbf{v} = \lambda^2\mathbf{v}$
3. Show that the eigenvectors are orthogonal to one another (e.g. their inner product is zero). This is true for real, symmetric matrices.

ANSWER

[38]:
```
print("1.")
eigen = la.eig(A)
values = np.round(eigen[0], 3)
vectors = np.round(eigen[1], 3)

print("The eigenvalues of A are {}, {} and {},associated respectively to␣
 ↪eigenvectors {},{} and {}".format(values[0], values[1], values[2],␣
 ↪vectors[0], vectors[1], vectors[2]))

print("2.")
```

```
l = eigen[0][0]
v = eigen[1][:,0]
print("I'll choose the first eigenpair, which has eigenvalue {} and eigenvector␣
 ↪{}".format(l, v.T))
display(Markdown("Let's calculate $\mathbf{Av}$ and $\lambda \cdot v$ and show␣
 ↪that they're the same"))
display(Markdown("$\mathbf{Av}$"))
Av = mmult(A,v)
print(Av)
display(Markdown("$\lambda\mathbf{v}$"))
lv = l * v
print(lv)
print("They match!")

print("3.")
vectors = eigen[1]
display(Markdown("$\mathbf{v_0}\mathbf{v_1}$"))
print(vectors[0].dot(vectors[1].T))

display(Markdown("$\mathbf{v_0}\mathbf{v_2}$"))
print(vectors[0].dot(vectors[2].T))

display(Markdown("$\mathbf{v_1}\mathbf{v_2}$"))
print(vectors[1].dot(vectors[2].T))
```

```
1.
The eigenvalues of A are 11.345, -0.516 and 0.171,associated respectively to
eigenvectors [-0.328 -0.737  0.591],[-0.591 -0.328 -0.737] and [-0.737  0.591
0.328]
2.
I'll choose the first eigenpair, which has eigenvalue 11.344814282762082 and
eigenvector [[-0.32798528 -0.59100905 -0.73697623]]
```

Let's calculate **Av** and $\lambda \cdot v$ and show that they're the same

Av

```
[[-3.72093206]
 [-6.70488789]
 [-8.36085845]]
```

$\lambda\mathbf{v}$

```
[[-3.72093206]
 [-6.70488789]
 [-8.36085845]]
They match!
3.
```

$\mathbf{v}_0\mathbf{v}_1$

```
[[-4.99600361e-16]]
```

$\mathbf{v}_0\mathbf{v}_2$

```
[[5.55111512e-17]]
```

$\mathbf{v}_1\mathbf{v}_2$

```
[[3.60822483e-16]]
```

3 Numerical Programming

3.1 Question 8

[10 points] Loading data and gathering insights from a real dataset

Data. The data for this problem can be found in the `data` subfolder in the `assignments` folder on github. The filename is `egrid2016.xlsx`. This dataset is the Environmental Protection Agency's (EPA) Emissions & Generation Resource Integrated Database (eGRID) containing information about all power plants in the United States, the amount of generation they produce, what fuel they use, the location of the plant, and many more quantities. We'll be using a subset of those data.

The fields we'll be using include:

field	description
SEQPLT16	eGRID2016 Plant file sequence number (the index)
PSTATABB	Plant state abbreviation
PNAME	Plant name
LAT	Plant latitude
LON	Plant longitude
PLPRMFL	Plant primary fuel
CAPFAC	Plant capacity factor
NAMEPCAP	Plant nameplate capacity (Megawatts MW)
PLNGENAN	Plant annual net generation (Megawatt-hours MWh)
PLCO2EQA	Plant annual CO2 equivalent emissions (tons)

For more details on the data, you can refer to the eGrid technical documents. For example, you may want to review page 45 and the section "Plant Primary Fuel (PLPRMFL)", which gives the full names of the fuel types including WND for wind, NG for natural gas, BIT for Bituminous coal, etc.

There also are a couple of "gotchas" to watch out for with this dataset: - The headers are on the second row and you'll want to ignore the first row (they're more detailed descriptions of the headers). - NaN values represent blanks in the data. These will appear regularly in real-world data, so getting experience working with it will be important.

Your objective. For this dataset, your goal is answer the following questions about electricity generation in the United States:

(a) Which plant has generated the most energy (measured in MWh)?
(b) What is the name of the northern-most power plant in the United States?
(c) What is the state where the northern-most power plant in the United States is located?
(d) Plot a histogram of the amount of energy produced by each fuel for the plant.
(e) From the plot in (e), which fuel for generation produces the most energy (MWh) in the United States?

ANSWER

[39]:
```
import os
import pandas as pd
data = pd.read_excel("C:\\Users\\Felipe\\Desktop\\Duke MIDS\\Machine␣
↪Learning\\ids705\\assignments\\data\\egrid2016.xlsx",header = 1)
```

[40]:
```
data.head()
```

[40]:
```
   SEQPLT16 PSTATABB                            PNAME        LAT         LON  \
0         1       AK          7-Mile Ridge Wind Project  63.210689 -143.247156
1         2       AK   Agrium Kenai Nitrogen Operations  60.673200 -151.378400
2         3       AK                          Alakanuk  62.683300 -164.654400
3         4       AK                Allison Creek Hydro  61.084444 -146.353333
4         5       AK                             Ambler  67.087980 -157.856719

  PLPRMFL   CAPFAC  NAMEPCAP  PLNGENAN  PLCO2EQA
0     WND      NaN       1.8       NaN       NaN
1      NG      NaN      21.6       NaN       NaN
2     DFO  0.05326       2.6  1213.001  1049.863
3     WAT  0.01547       6.5   881.000     0.000
4     DFO  0.13657       1.1  1315.999  1087.881
```

(a)

[41]:
```
plant = data.sort_values("PLNGENAN",ascending = False).head(1)
name = plant.loc[:,"PNAME"].values[0]
state = plant.loc[:,"PSTATABB"].values[0]
generation = plant.loc[:,"PLNGENAN"].values[0]

print("The plant which generated the most energy (among all for which data is␣
↪available) was plant {} ({}), which generated {}  MWh".format(name, state,␣
↪generation))
```

```
The plant which generated the most energy (among all for which data is
available) was plant Palo Verde (AZ), which generated 32377476.999  MWh
```

(b)

[42]:
```
plant = data.sort_values("LAT",ascending = False).head(1)
name = plant.loc[:,"PNAME"].values[0]
state = plant.loc[:,"PSTATABB"].values[0]
lat = plant.loc[:,"LAT"].values[0]

print("The northernmost plan in the US is plant {} ({}), which is located at␣
↪latitude {}N".format(name, state, lat))
```

```
The northernmost plan in the US is plant Barrow (AK), which is located at
latitude 71.292N
```

(c)

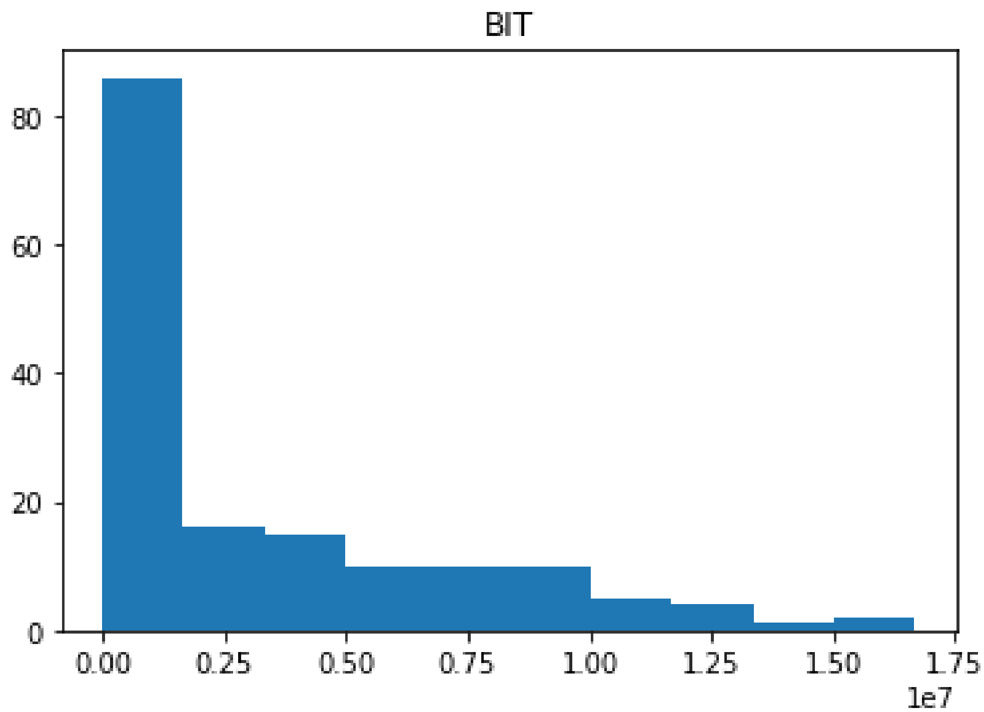
[43]:
```python
print(state)
```

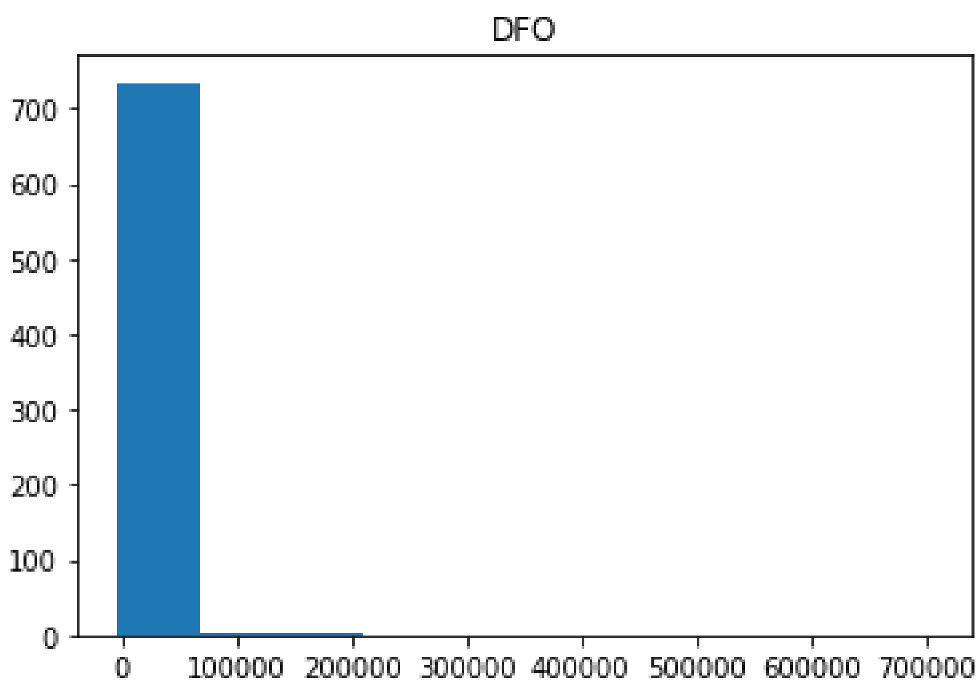
```
AK
```

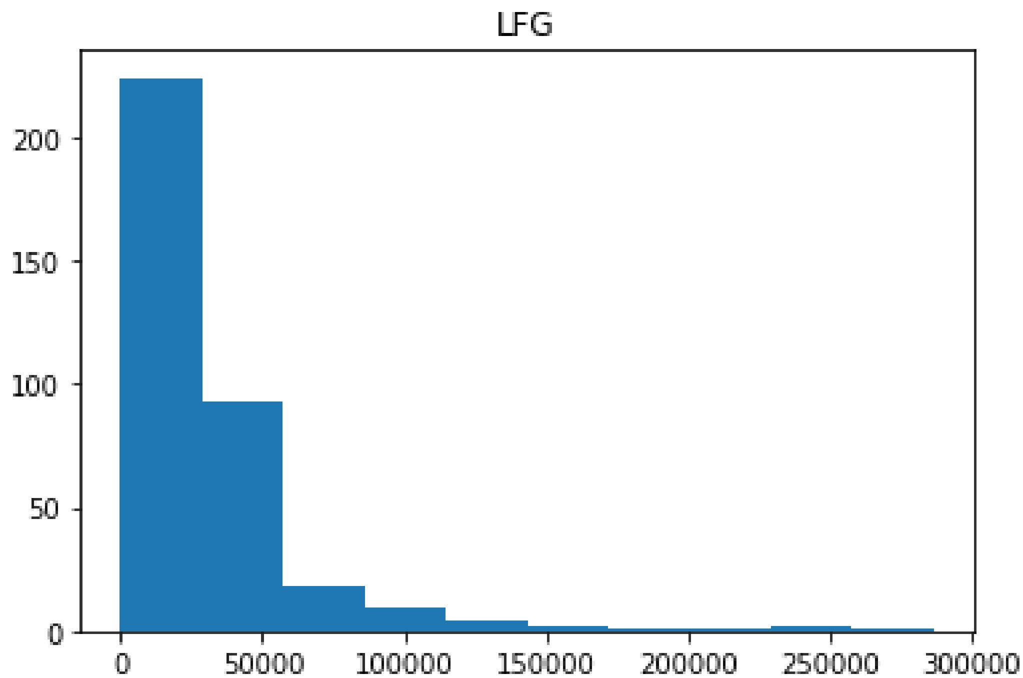
(d) (I'm not sure whate the question means when it says "for each fuel for a plant", so I'm assuming you want a histogram of energy generated by plants with each fuel type. However, to guarantee some meaning to these histograms, I'll only plot them for fuels which are used by at least 100 plants)

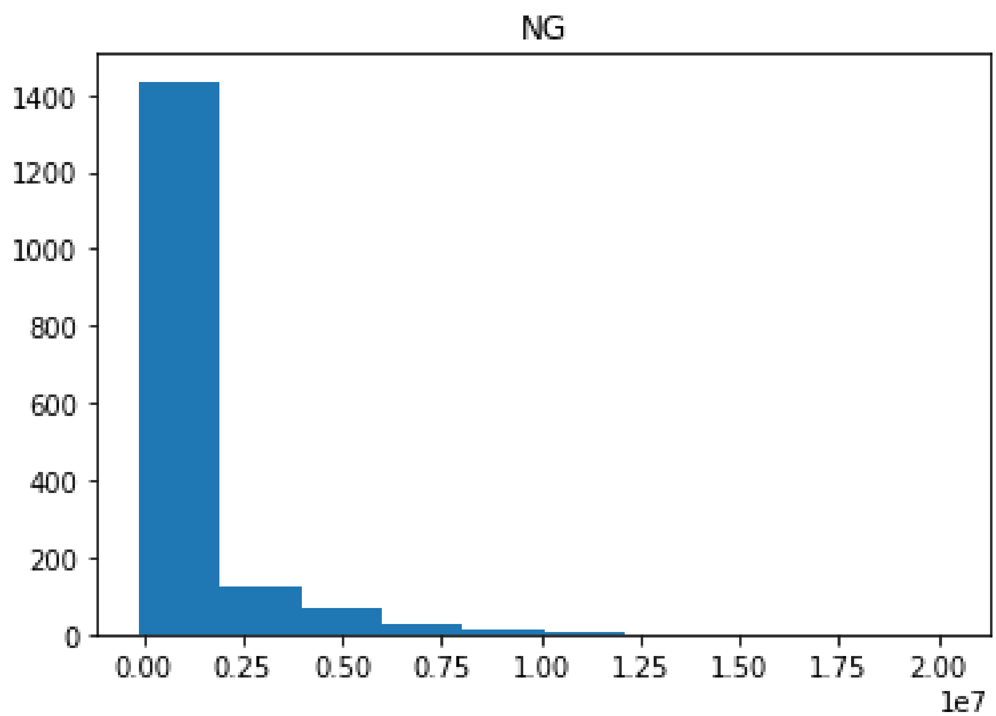
[44]:
```python
import matplotlib.pyplot as plt

gen_by_fuel = data.loc[:,["PLPRMFL","PLNGENAN"]]
fuel_types = list(gen_by_fuel.PLPRMFL.sort_values().unique())

for fuel in fuel_types:
    gen = gen_by_fuel.loc[gen_by_fuel.PLPRMFL == fuel,"PLNGENAN"]
    if len(gen) >= 100:
        plt.hist(gen)
        plt.title(fuel)
        plt.show()
```

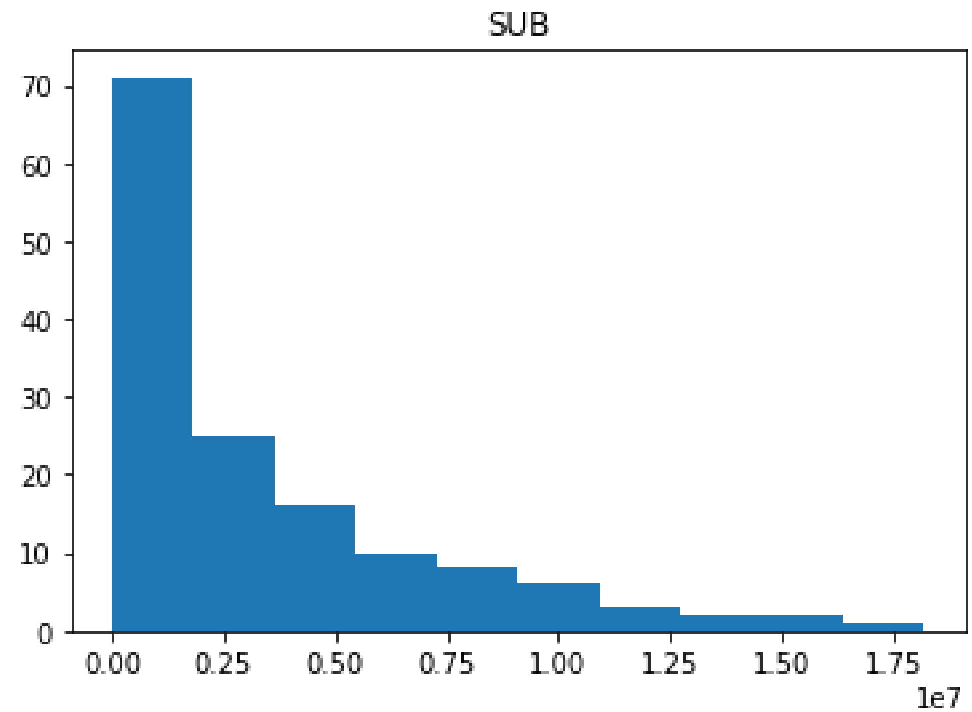
```
C:\Users\Felipe\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824:
RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\Felipe\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825:
RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

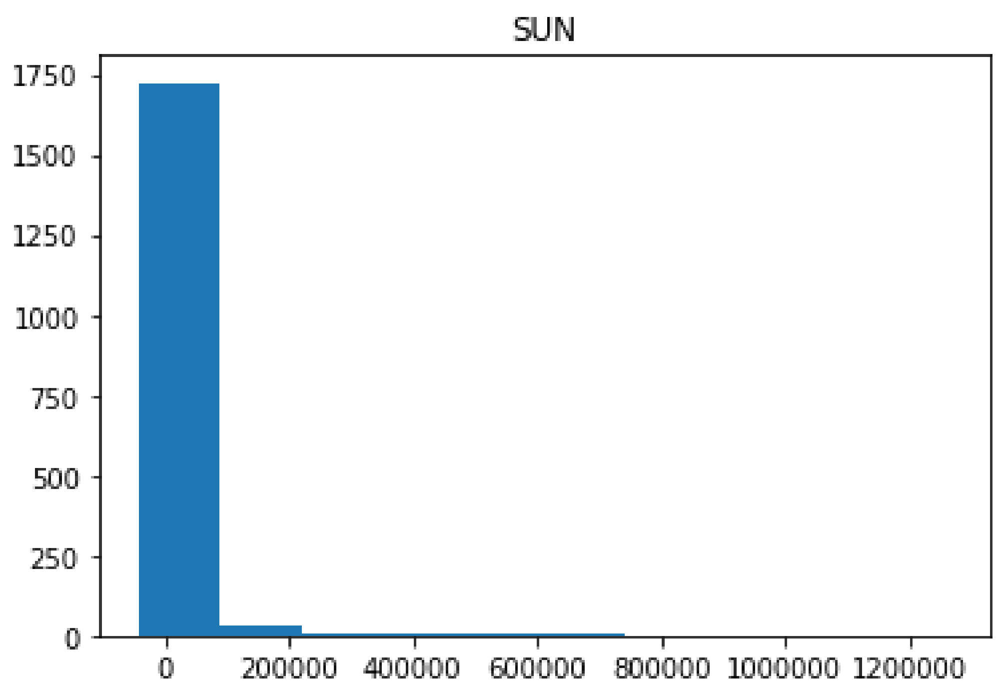

BIT
80
60
40
20
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e7

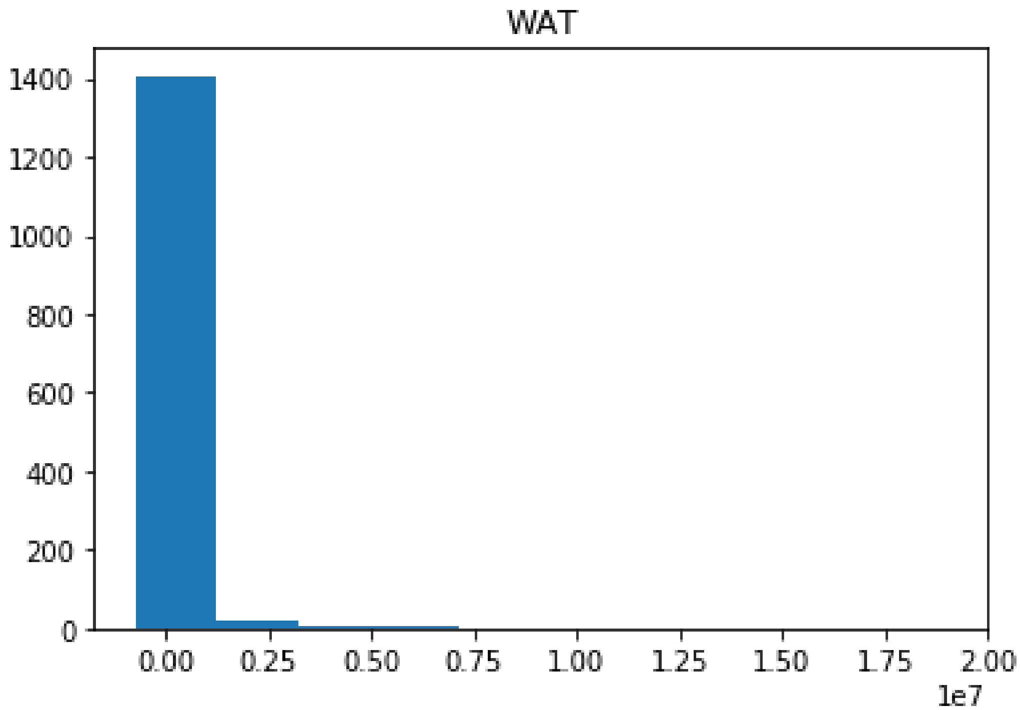

DFO
700
600
500
400
300
200
100
0
0
100000
200000
300000
400000
500000
600000
700000

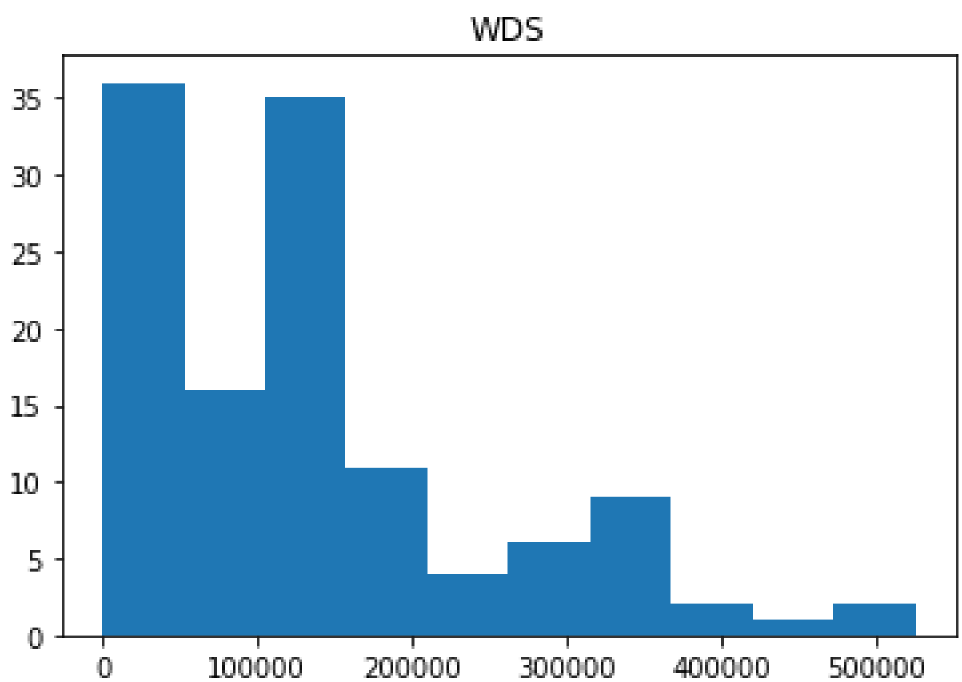

LFG
200
150
100
50
0
0
50000
100000
150000
200000
250000
300000

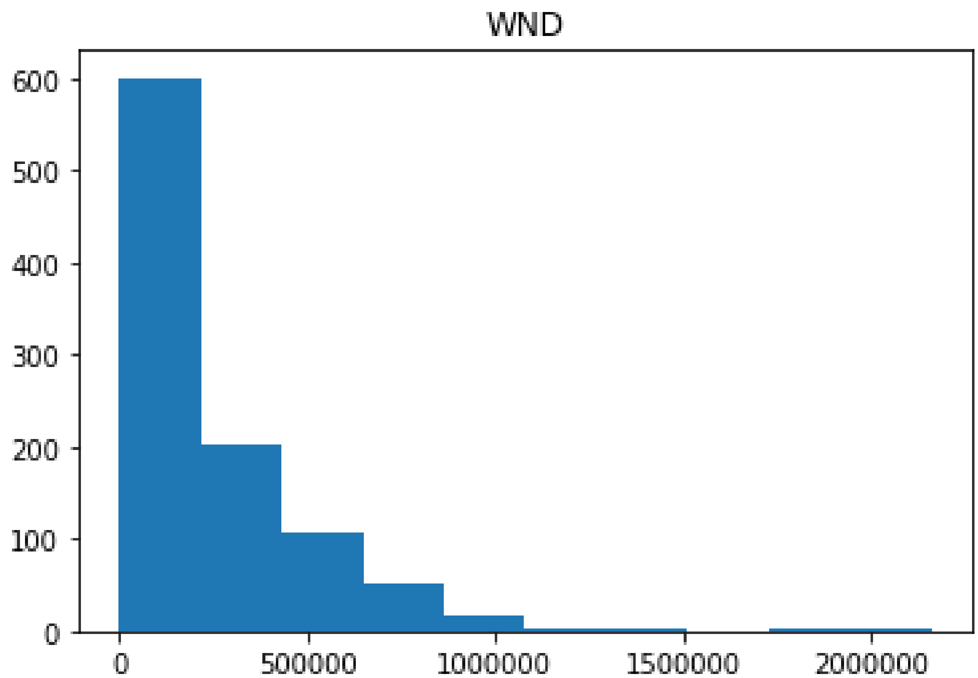

NG
1400
1200
1000
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7


SUB
70
60
50
40
30
20
10
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e7


SUN
1750
1500
1250
1000
750
500
250
0
0
200000
400000
600000
800000
1000000
1200000


WAT
1400
1200
1000
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7


WDS
35
30
25
20
15
10
5
0
0
100000
200000
300000
400000
500000



(e) This question can be most easily answered by calculating the total output for each kind of fuel and seeing which fuel type ranks highest. The answer we get is **Natural gas**.

[45]:
```
data.loc[:,["PLPRMFL","PLNGENAN"]].groupby("PLPRMFL").sum().
↪sort_values("PLNGENAN",ascending = False).head(5)
```

[45]:
```
              PLNGENAN
PLPRMFL
NG        1.314956e+09
NUC       8.124758e+08
BIT       5.049193e+08
SUB       4.716984e+08
WAT       2.607785e+08
```

3.2 Question 9

[8 points] Speed comparison between vectorized and non-vectorized code. Begin by creating an array of 10 million random numbers using the numpy random.randn module. Compute the sum of the squares first in a for loop, then using Numpy's `dot` module. Time how long it takes to compute each and report the results and report the output. How many times faster is the vectorized code than the for loop approach?

*Note: all code should be well commented, properly formatted, and your answers should be output using the `print()` function as follows (where the # represents your answers, to a reasonable precision):

```
Time [sec] (non-vectorized): ######
Time [sec] (vectorized):     ######
```

```
The vectorized code is ##### times faster than the vectorized code
```

ANSWER

[46]:
```python
import numpy as np
import time

# Generate the random samples

x = np.random.randn(10**7)

# Compute the sum of squares the non-vectorized way (using a for loop)
start_time_loop = time.time()
ssq = 0
for i in x:
    ssq += i**2
elapsed_time_loop = time.time() - start_time_loop

# Compute the sum of squares the vectorized way (using numpy)
start_time_vector = time.time()
x.dot(x)
elapsed_time_vector = time.time() - start_time_vector


# Print the results
print("Time [sec] (non-vectorized): {:.2f}".format(elapsed_time_loop))
print("Time [sec] (vectorized): {:.2f}".format(elapsed_time_vector))
print("The vectorized code is {:.2f} times faster than the non-vectorized code".
 ↪format(elapsed_time_loop/elapsed_time_vector))
```

```
Time [sec] (non-vectorized): 10.43
Time [sec] (vectorized): 0.01
The vectorized code is 1040.94 times faster than the non-vectorized code
```

3.3 Question 10

[10 points] One popular Agile development framework is Scrum (a paradigm recommended for data science projects). It emphasizes the continual evolution of code for projects, becoming progressively better, but starting with a quickly developed minimum viable product. This often means that code written early on is not optimized, and that's a good thing - it's best to get it to work first before optimizing. Imagine that you wrote the following code during a sprint towards getting an end-to-end system working. Vectorize the following code and show the difference in speed between the current implementation and a vectorized version.

The function below computes the function $f(x, y) = x^2 - 2y^2$ and determines whether this quantity is above or below a given threshold, `thresh=0`. This is done for $x, y \in \{-4, 4\}$, over a 2,000-by-2,000 grid covering that domain.

(a) Vectorize this code and demonstrate (as in the last exercise) the speed increase through vectorization and (b) plot the resulting data - both the function $f(x, y)$ and the thresholded output - using `imshow` from `matplotlib`.

Hint: look at the `numpy` `meshgrid` *documentation*

ANSWER

[47]:
```python
import numpy as np
import time
import matplotlib.pyplot as plt

# Initialize variables for this exerise
mesh_size = 2000 #Set to 2000 when running for real. Set it to smaller value␣
 ↪when testing code
x_values = np.arange(-4,4, 1.0/mesh_size)
y_values = np.arange(-4,4, 1.0/mesh_size)
x, y = np.meshgrid(x_values, y_values, sparse = True)

#Define functions
def f(x,y):
    return(x**2 - 2 * y **2)

def g(x, threshold = 0.0):
    '''Returns True if x > threshold and False '''
    return(x > threshold)
```

[48]:
```python
# Nonvectorized implementation
start_time_nonvec = time.time()
f_nonvec = []
g_nonvec = []
for i in x_values:
    for j in y_values:
        new_f = f(i,j)
        new_g = g(new_f)
        f_nonvec.append(new_f)
        g_nonvec.append(new_g)
elapsed_time_nonvec = time.time() - start_time_nonvec
```

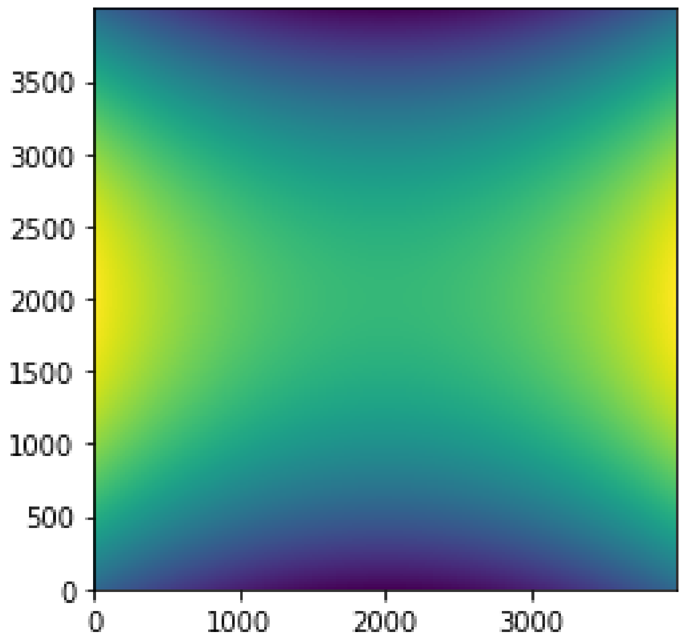
[49]:
```python
# Vectorized implementation
start_time_vec = time.time()
f_vec = f(x, y)
g_vec = g(f_vec)
elapsed_time_vec = time.time() - start_time_vec
```

[50]:
```python
# Print the time for each and the speed increase
print("Elapsed time (non-vectorized) : {:.2f}".format(elapsed_time_nonvec))
print("Elapsed time (vectorized) : {:.2f}".format(elapsed_time_vec))
print("The vectorized code is {:.2f} times faster than the non-vectorized code".
 ↪format(elapsed_time_nonvec/elapsed_time_vec))
```

```
Elapsed time (non-vectorized) : 42.38
Elapsed time (vectorized) : 0.13
The vectorized code is 333.93 times faster than the non-vectorized code
```

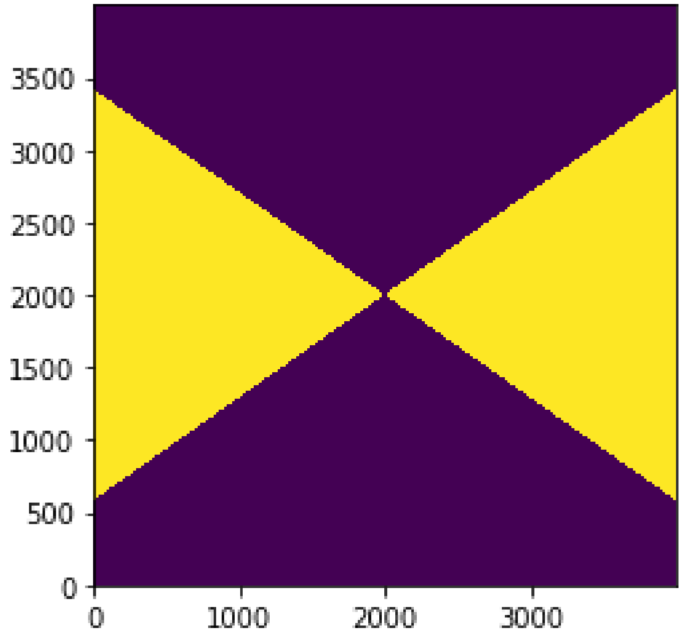
[51]:
```
#Plot the result -f(x,y)
plt.imshow(f_vec, origin = 'lower')
```

[51]: `<matplotlib.image.AxesImage at 0x1cfb2f66a20>`



[52]:
```
#Plot the result - g(x,y)
plt.imshow(g_vec, origin = 'lower')
```

[52]: `<matplotlib.image.AxesImage at 0x1cfb2fb2e48>`



3.4 Question 11

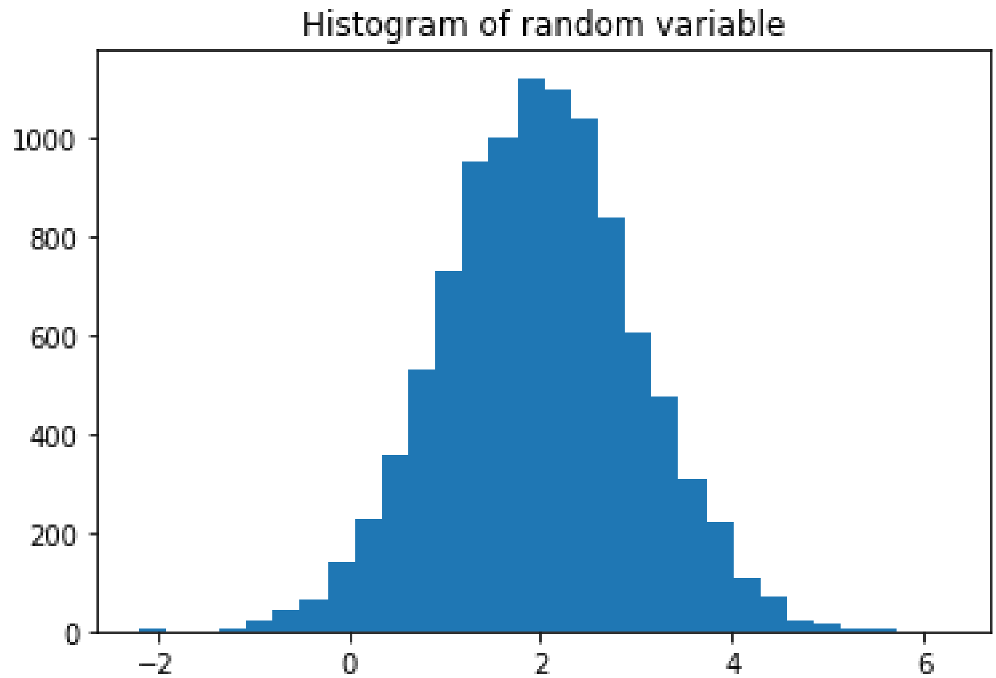
[10 points] This exercise will walk through some basic numerical programming exercises. 1. Synthesize $n = 10^4$ normally distributed data points with mean $\mu = 2$ and a standard deviation of $\sigma = 1$. Call these observations from a random variable X, and call the vector of observations that you generate, **x**. 2. Calculate the mean and standard deviation of **x** to validate (1) and provide the result to a precision of four significant figures. 3. Plot a histogram of the data in **x** with 30 bins 4. What is the 90th percentile of **x**? The 90th percentile is the value below which 90% of observations can be found. 5. What is the 99th percentile of **x**? 6. Now synthesize $n = 10^4$ normally distributed data points with mean $\mu = 0$ and a standard deviation of $\sigma = 3$. Call these observations from a random variable Y, and call the vector of observations that you generate, **y**. 7. Create a new figure and plot the histogram of the data in **y** on the same axes with the histogram of **x**, so that both histograms can be seen and compared. 8. Using the observations from **x** and **y**, estimate $E[XY]$

ANSWER

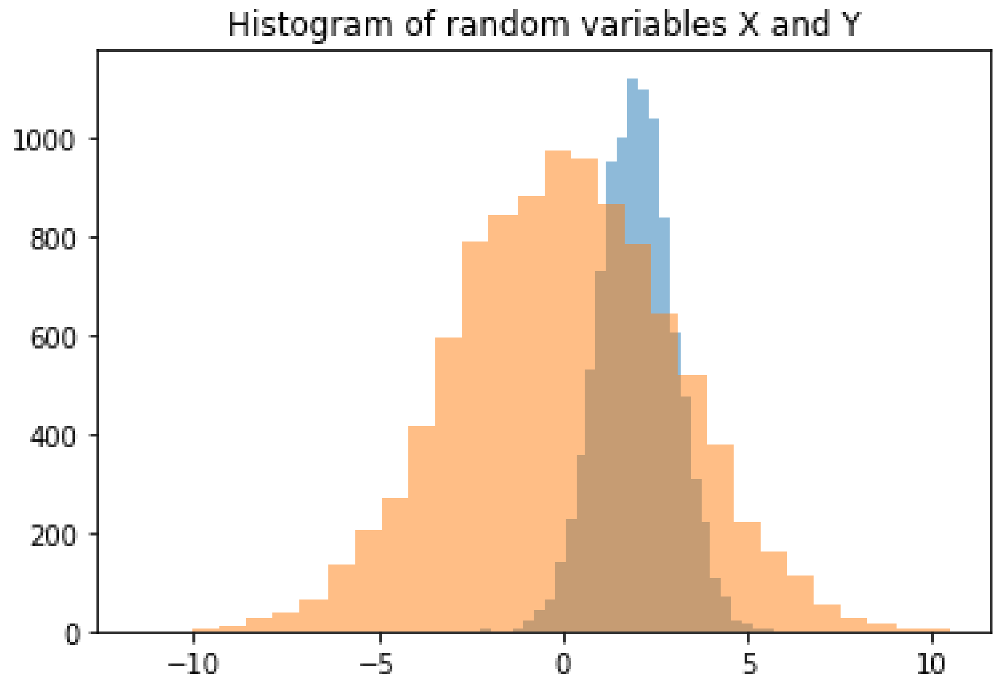
[53]:
```python
# 1

print("1.")
x = np.random.normal(2, 1, size = 10**4)
print("Random vector X generated")

# 2

print("2.")
print("Mean = {:.4f}".format(np.mean(x)))
print("St.Dev. = {:.4f}".format(np.std(x)))
```

```python
# 3

print("3.")
plt.hist(x, bins = 30)
plt.title("Histogram of random variable")
plt.show()

# 4

print("4.")
print("90th percentile = {:.4f}".format(np.percentile(x, 90)))

# 5

print("5.")
print("99th percentile = {:.4f}".format(np.percentile(x, 99)))

# 6

print("6.")
y = np.random.normal(0, 3, size = 10**4)
print("Random vector Y generated")

# 7

print("7.")
plt.hist(x, bins = 30, alpha = 0.5)
plt.hist(y, bins = 30, alpha = 0.5)
plt.title("Histogram of random variables X and Y")
plt.show()

# 8
print("8.")
EXY = x.dot(y) / 10**4
print("E(XY) is estimated to be {:.4f}".format(EXY))
```

```
1.
Random vector X generated
2.
Mean = 1.9876
St.Dev. = 1.0070
3.
```




```
4.
90th percentile = 3.2798
5.
99th percentile = 4.3318
6.
Random vector Y generated
7.
```




```
8.
E(XY) is estimated to be 0.0254
```

4 Version Control via Git

4.1 Question 12

[1 point] You will need to use Git to submit assignments and in the course projects and is generally a version control and collaboration tool. You can even use some Git repositories (e.g. Github) as hosts for website, such as with the course website.

Complete the Atlassian Git tutorial, specifically the following listed sections. Try each concept that's presented. For this tutorial, instead of using BitBucket as your remote repository host, you may use your preferred platform such as Github or Duke's Gitlab. 1. What is version control 2. What is Git 3. Install Git 4. Setting up a repository 5. Saving changes 6. Inspecting a repository 7. Undoing changes 8. Rewriting history 9. Syncing 10. Making a pull request 11. Using branches 12. Comparing workflows

I also have created two videos on the topic to help you understand some of these concepts: Git basics and a step-by-step tutorial.

For your answer, affirm that you *either* completed the tutorial or have previous experience with all of the concepts above. Do this by typing your name below and selecting the situation that applies from the two options in brackets.

ANSWER

I, Felipe Buchbinder, affirm that I have previous experience that covers all the content in this tutorial

5 Exploratory Data Analysis

5.1 Question 13

[20 points] Here you'll bring together some of the individual skills that you demonstrated above and create a Jupyter notebook based blog post on data analysis.

1. Find a dataset that interests you and relates to a question or problem that you find intriguing
2. Using a Jupyter notebook, describe the dataset, the source of the data, and the reason the dataset was of interest.
3. Check the data and see if they need to be cleaned: are there missing values? Are there clearly erroneous values? Do two tables need to be merged together? Clean the data so it can be visualized.
4. Plot the data, demonstrating interesting features that you discover. Are there any relationships between variables that were surprising or patterns that emerged? Please exercise creativity and curiosity in your plots.
5. What insights are you able to take away from exploring the data? Is there a reason why analyzing the dataset you chose is particularly interesting or important? Summarize this as if your target audience was the readership of a major news organization - boil down your findings in a way that is accessible, but still accurate.

Here your analysis will evaluated based on: 1. Data cleaning: did you look for and work to resolve issues in the data? 2. Quality of data exploration: did you provide plots demonstrating interesting aspects of the data? 3. Interpretation: Did you clearly explain your insights? Restating the data, alone, is not interpretation. 5. Professionalism: Was this work done in a way that exhibits professionalism through clarity, organization, high quality figures and plots, and meaningful descriptions?

ANSWER

As a signalling of my interest in Energy Research, I'll be using data on fuel prices in Brazil.

I want to understand the determinants of price markup on gas stations, and how has this evolved through time. There are, indeed, many studies on gasoline prices and its evolution. Some studies have shown that prices offered to the final consumer would go up when the barrel of petroleum went up, but wouldn't go down when the price of the barrel of petroleum dropped. So I'm interested in studying the market power of gas stations, by developing a regression (possibly geospatial) model to predict gas station's markup.

To do this, I have a series of datasets with the prices which every gas station in Brazil bought or sold fuel, every week, from 2004 to 2019 (first semester). It contains information on Gasoline, Ethanol, Diesel and Liquified Petroleum Gas. Such data is collected by the Brazilian Petroleum Agency, and can be found here.

But there's a problem.

There's lots of missing data regarding the price paid by gas stations to buy gasoline.

If this data is missing at random, I can still do inference. If it is not, than I cannot.

My goal in this exercise will be to investigate if data on the price paid by gas stations to buy gasoline is missing at random.

I consider only prices for gasoline in the first semester of 2019.

[54]:
```
#Load the data
```

```python
data = pd.read_csv("C:\\Users\\Felipe\\Desktop\\Duke MIDS\\Machine␣
↪Learning\\Machine Learning\\gasoline prices\\2019-1_CA.csv", encoding =␣
↪"utf-16", sep = "\t", decimal = ",")

#Change column names to english
data.columns = ['Region',␣
↪'State','City','Gas_Station_Name','Gas_Station_id','Fuel_Type','Date', \
                'Sell_Price','Buy_Price','Unit','Fuel_Source']

#Let's take a look at our data
data.sample(5)
```

[54]:

```
        Region State                 City  \
323776      SE    RJ  CAMPOS DOS GOYTACAZES
436276      SE    SP             PAULINIA
15471       CO    GO            ITUMBIARA
450165      SE    SP                SALTO
326827      SE    RJ             ITABORAI

                                   Gas_Station_Name  Gas_Station_id Fuel_Type  \
323776  S.G.A. INTERLAGOS DE CAMPOS COMB. LTDA - ME  39238340000190    ETANOL
436276         POSTO JARDIM EUROPA DA PAULINIA LTDA   2804930000123  GASOLINA
15471                          AUTO POSTO KEOPS LTDA   5783134000140    ETANOL
450165                      POSTO DOIS MIL SALTO LTDA  24173688000170    DIESEL
326827                             POSTO NOTA 10 LTDA   7473553000100       GNV

              Date  Sell_Price  Buy_Price        Unit  \
323776  16/01/2019       3.190     2.6799  R$ / litro
436276  06/05/2019       4.199     3.9612  R$ / litro
15471   03/04/2019       3.089        NaN  R$ / litro
450165  01/05/2019       3.499        NaN  R$ / litro
326827  15/04/2019       3.099        NaN  R$ / litro

                          Fuel_Source
323776                         BRANCA
436276  PETROBRAS DISTRIBUIDORA S.A.
15471                          BRANCA
450165                         RAIZEN
326827                         BRANCA
```

[55]:

```python
#Filter gasoline
data = data.loc[data.Fuel_Type == 'GASOLINA']

assert (data.Fuel_Type == 'GASOLINA').all()

#Drop column on Fuel_Type
data = data.drop('Fuel_Type', axis = 1)
```

```
#Let's check if all sell and buy prices are in the same unit
assert len(data.Unit.unique()) == 1

#Since all sell and buy prices are in the same unit, let's drop it
#Let's also drop the Gas_Station_Name, since we can already identify it with
 ↪it' id.
data = data.drop(['Unit','Gas_Station_Name'], axis = 1)

#Which variables do we have?
for variable in data.columns:
    print(variable)
```

```
Region
State
City
Gas_Station_id
Date
Sell_Price
Buy_Price
Fuel_Source
```

[56]:
```
#How much data do we have?
data.shape
```

[56]: (150984, 8)

[57]:
```
#How much missing data do we have in each variable?
#Let's give our answer as a percentage of the number of rows
round(100 * data.isnull().sum() / data.shape[0], 2)
```

[57]:
```
Region             0.00
State              0.00
City               0.00
Gas_Station_id     0.00
Date               0.00
Sell_Price         0.00
Buy_Price         58.14
Fuel_Source        0.00
dtype: float64
```

Our data seems pretty much complete, except for information on the Buy Price, where 58% of the dara is missing.

This is unfortunate, since I would love to study determinants of Gas Station's markups... I can still do that if I show that missing values are MAR, so I'll create a new variable which indicates if the Buy_Price is missing (1 if it's missing, 0 if not).

My goal is to investigate this variable and hopefully I'll find that it's missing at random. Most likely not. But I need to test this at some point, so I might as well use this exercise as an opportunity.

[58]:
```python
#Create indicator variable.
#1 if Buy_Price is missing.
#0 otherwise
data.loc[:,"is_missing_buy_price"] = data.Buy_Price.isnull()

#Check if our variable was created with no errors
#This means two things:
#1. All values with missing Buy_Price must be evaluated to True;
#2. All values with non-missing Buy_Price must be evaluated to False;
assert (data.loc[data.Buy_Price.isnull(),"is_missing_buy_price"] == True).all()
assert (data.loc[data.Buy_Price.notnull(),"is_missing_buy_price"] == False).
 ↪all()

#Discard Buy_Price
data = data.drop('Buy_Price', axis = 1)
```

[59]:
```python
#Let's take care of the Date variable and treat it as a date, not as a string
data.loc[:,"Date"] = pd.to_datetime(data.Date,format = "%d/%m/%Y")

#Let's also assign each week the number it corresponds in the year
data.loc[:,"Week"] = data.loc[:,"Date"].dt.week

#Let's take a look
data.sample(5)
```

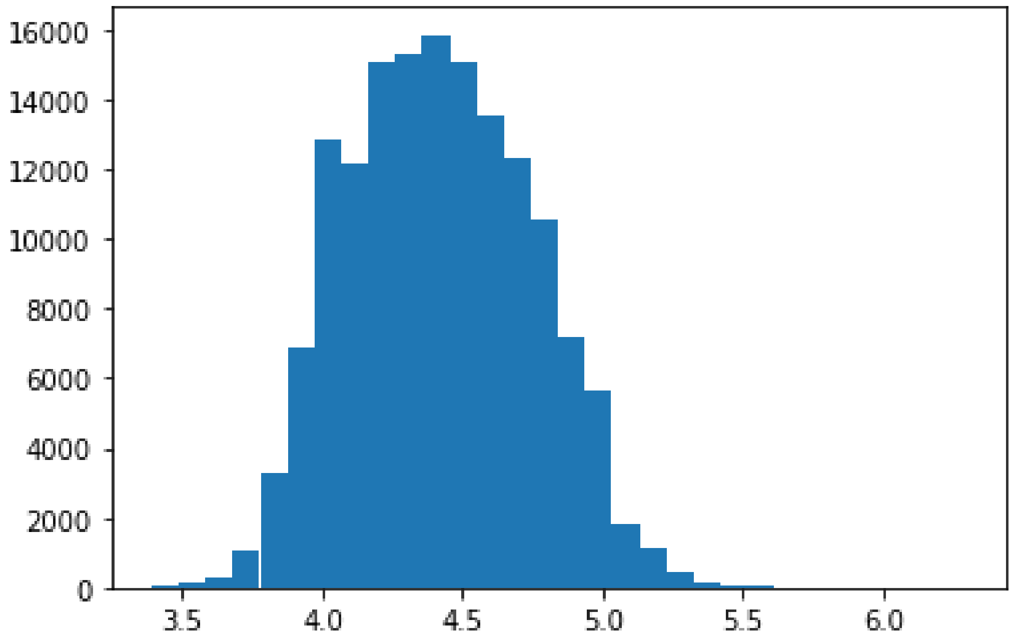
[59]:
```
       Region State                 City  Gas_Station_id       Date  \
281265     SE    MG  GOVERNADOR VALADARES  13569064002012 2019-05-20
156586     NE    PE             SALGUEIRO   8279379000122 2019-05-22
211902      S    RS         CAXIAS DO SUL  68821172000160 2019-05-20
110726     NE    BA               VALENCA   8925345000168 2019-06-05
202184      S    PR   SAO JOSE DOS PINHAIS  79082087000143 2019-05-06

        Sell_Price                   Fuel_Source  is_missing_buy_price  Week
281265       4.550  PETROBRAS DISTRIBUIDORA S.A.                  True    21
156586       4.849                        BRANCA                 False    21
211902       4.939                        RAIZEN                 False    21
110726       4.889  PETROBRAS DISTRIBUIDORA S.A.                 False    23
202184       4.299                        RAIZEN                  True    19
```

[60]:
```python
#Are selling prices reasonable? Any outliers?
data.loc[:,['Sell_Price','Week']].describe()
```

[60]:
```
          Sell_Price           Week
count  150984.000000  150984.000000
mean        4.411974      13.498530
std         0.330756       7.502443
min         3.390000       1.000000
25%         4.179000       7.000000
50%         4.399000      13.000000
```

```
75%          4.659000      20.000000
max          6.290000      26.000000
```

[61]:
```python
plt.hist(data.Sell_Price, bins = 30)
plt.show()
```



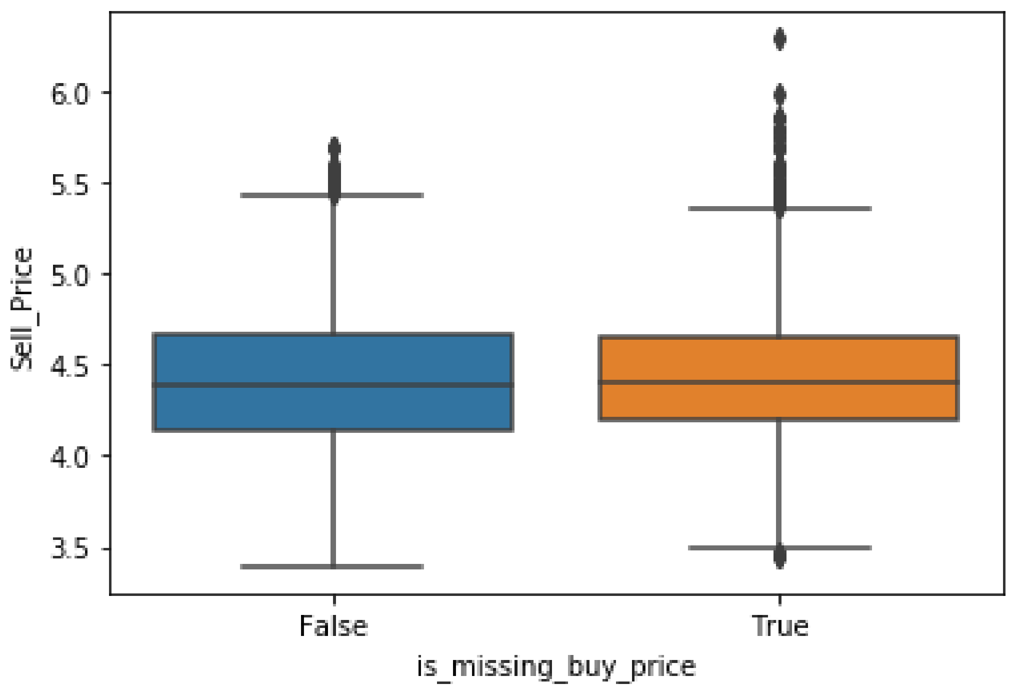
Apparently there are no outliers. Good.

Now let's do some feature engineering.

Let's now try to establish relationships between our variable of interest and the remaining variables. Hopefully, we won't find anything, which would help us make our claim that the variable is missing at random.

[62]:
```python
import seaborn as sns
sns.boxplot(y='Sell_Price', x='is_missing_buy_price', data = data)
```

[62]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cfb4cd57f0>
```



The selling price does not seem to be different between gas stations for which the buy price is missing or not missing. This is good news for us.

Now let's see if the geographical location of the gas station can predict it.

[63]:
```
data.loc[:,["Region","is_missing_buy_price"]].groupby("Region").mean().
 ↪sort_values('is_missing_buy_price',ascending = False)
```

[63]:
```
        is_missing_buy_price
Region
CO                  0.715929
S                   0.647655
N                   0.578926
NE                  0.574756
SE                  0.535518
```

Apparently the probability of a missing value varies substantially by region. It seems much more likely to have missing data in the Center-West (CO) region, than in the South-East (SE).

If we wish to perform the same analysis at the city level, we must take care not to include cities which have a very small number of gas stations. So let's take a look at the distribution of gas stations per city:

[64]:
```
nb_gas_stations = data.loc[:,['City','Region','Gas_Station_id']].
 ↪drop_duplicates().groupby(['City','Region']).count()
nb_gas_stations.columns = ['Number_of_Gas_Stations']

#The cities with the highest number of gas stations should be the countries'␣
 ↪largest cities.
```

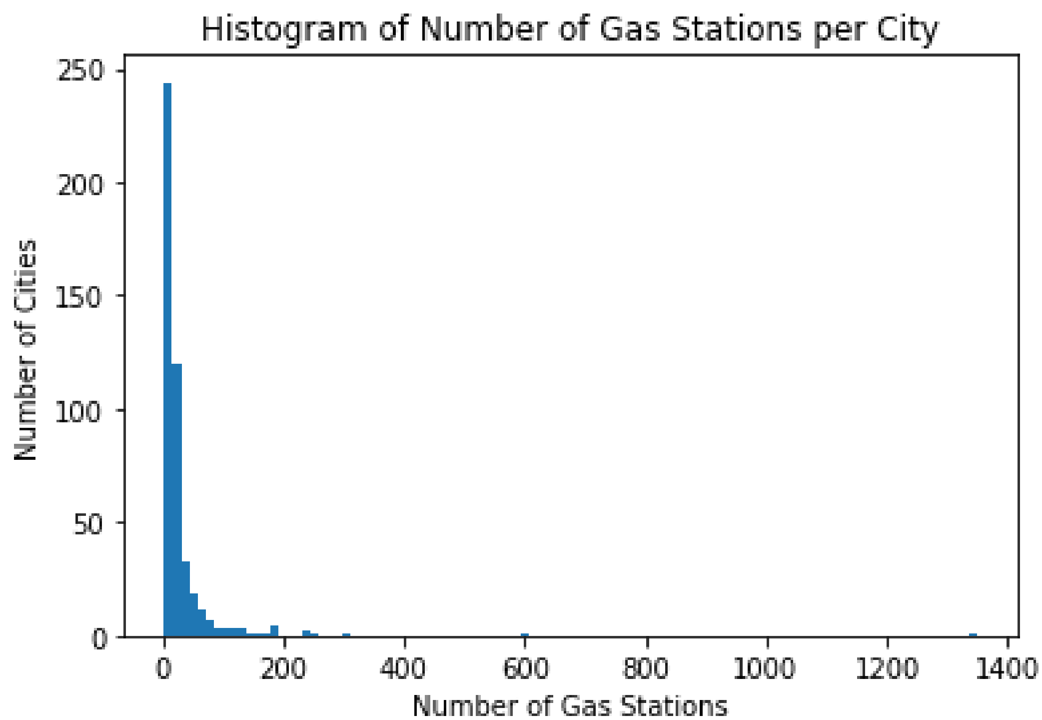
```python
#Indeed, they are.
print(nb_gas_stations.sort_values('Number_of_Gas_Stations', ascending = False).
 ↪head(4))
```

```
                         Number_of_Gas_Stations
City           Region
SAO PAULO      SE                          1349
RIO DE JANEIRO SE                           604
FORTALEZA      NE                           303
GOIANIA        CO                           248
```

[65]:
```python
#So let's take a look at the distribution of gas stations throughout the␣
 ↪country
nb_gas_stations.describe()
```

[65]:
```
       Number_of_Gas_Stations
count              459.000000
mean                31.461874
std                 75.912183
min                  4.000000
25%                 11.000000
50%                 16.000000
75%                 27.500000
max               1349.000000
```

[66]:
```python
plt.hist(nb_gas_stations.Number_of_Gas_Stations, bins = 100)
plt.title("Histogram of Number of Gas Stations per City")
plt.xlabel("Number of Gas Stations")
plt.ylabel("Number of Cities")
plt.show()
```



Note that the minimum of our distribution is 4. This means that the city with the least number of gas stations has 4 gas stations.

I confess that I find this result strange. It made me wonder weather the dataset really has all Brazilian pumps, as I thought it had. So I'll investigate that.

[67]:
```
print("Brazil has about 5,000 cities, but our data has {} different cities".
↪format(len(data.City.unique())))
```

```
Brazil has about 5,000 cities, but our data has 458 different cities
```

These numbers are very different.

It is unliquely that there are no cities with a single gas station.

In other words, it is unliquely that only 458 cities have gas stations.

More likely, data was only collected for these major cities.

This may suggest some bias in our data.

It is also interesting to note that, if only 458 cities have 4+ gas stations, this means that the *vast* majority of Brazilian cities have at most 3 gas station.

This means we get a false impression of this distribution by merely looking at our dataset. The quartiles are actually much closer to zero than shown in our previous descriptive statistics analysis.

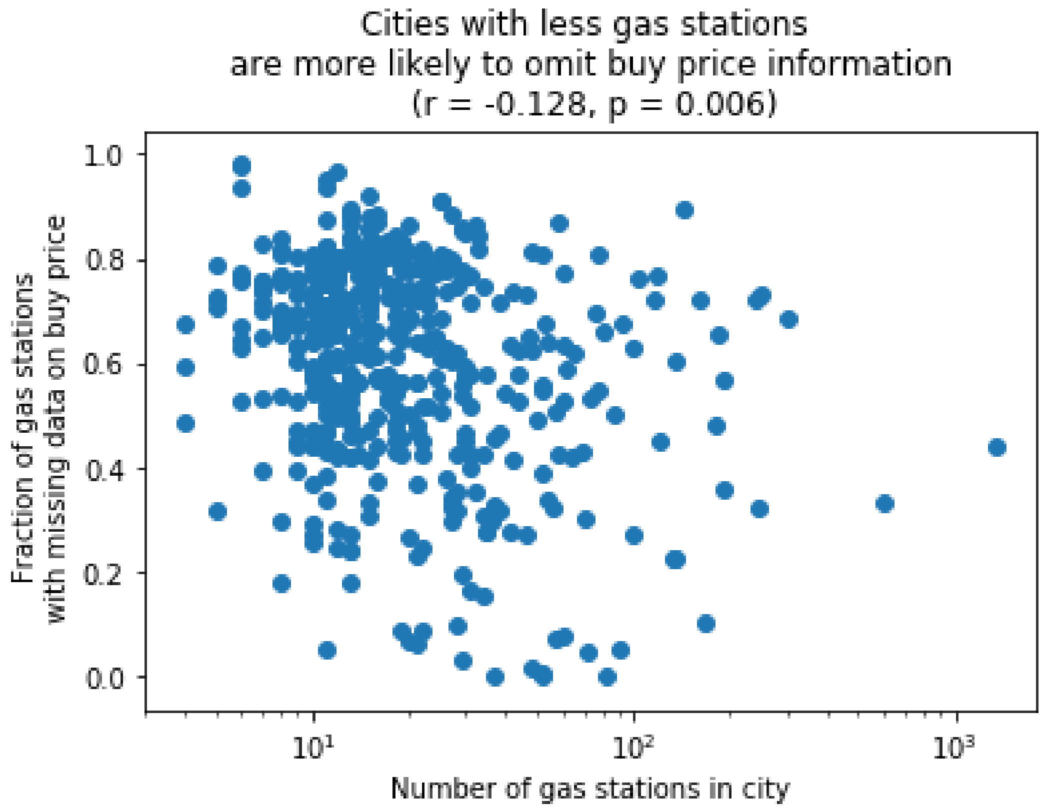
In addition, gas stations in cities which have 1, 2 or 3 gas stations operate in a near-monopoly market. Their dynamics is arguably different than what we observe in the remaining 458 cities, where structuring a stable cartel is much more complex. This impacts the gas station's markup directly, as markups are arguably smaller the more competition one encounters.

This raises the hypothesis that cities with less gas stations may have more market power *not* to reveal the price which they paid for gasoline. Let's see if there's some evidence to this hypothesis:

[68]:
```
#Fraction of gas stations with missing buy price in each city
missing_by_city = data.loc[:,["City","Region","is_missing_buy_price"]].
 ↪groupby(["City","Region"]).mean().
 ↪sort_values('is_missing_buy_price',ascending = False)

#Let's merge it with the number of gas stations per city
missing_by_city = missing_by_city.merge(nb_gas_stations, how = 'inner',on =␣
 ↪('City','Region'), validate = '1:1')
```

[69]:
```
#Correlation
from scipy.stats import pearsonr
r_aux_obj = pearsonr(missing_by_city.Number_of_Gas_Stations, missing_by_city.
 ↪is_missing_buy_price)
r = r_aux_obj[0] #Pearson Correlation Coefficient
p = r_aux_obj[1] #p-value

#Let's make these things more printable
r = str(round(r, 3))
p = str(round(p, 3))
```

[70]:
```
#Plot relationship between number of gas stations and fraction of them which␣
 ↪omit buy price
plt.scatter(missing_by_city.Number_of_Gas_Stations, missing_by_city.
 ↪is_missing_buy_price)
plt.xscale("log")
plt.title("Cities with less gas stations \n are more likely to omit buy price␣
 ↪information \n (r = " + str(r) + ", p = " + p + ")")
plt.xlabel("Number of gas stations in city")
plt.ylabel("Fraction of gas stations \n with missing data on buy price")
plt.show()
```



There is a week, albeit significant relationship between the number of gas stations in a city and the fraction of gas stations which omit buy price.

This confirms our hypothesis on the existence of such relationship, although we can not be sure that this relationship exists for the reason which led us to think of it in the first place, namely, that gas stations have more market power when there are fewer of them. Alternatively, it may be that the government is more effective in collecting data in larger cities, where there are more gas stations.

Either way, the relationship exists.

This is bad news for me, as it is further evidence that missing data on buy price is *not* missing at random (MAR). Hence, so far, we have two evidences that missing data on buy price is not MAR:

1. The incidence of missing values is not homogeneous among the country's regions;
2. Gas stations are more likely to omit information on their buy price in cities where there are few gas stations;

This imposes some limitations to the paper I wished to write on gas station's markpus.

On the other hand, if missing data is contingent on variables within my dataset, I can still have hopes of overcoming this problem using multiple imputation methods. Moreover, we must acknowledge that this dataset is possibly the closest we can get to populational data. In addition, it is also likely to be the most reliable data we can get.

So let's proceed with our analysis. We wished to see if different cities could have different probabilities of omission for the buyer price variable.

In some sense, we know this answer to be affirmative, as we know it to be related to the number of gas stations in the city. Still, let's look at the fraction of missing data for different cities. We consider only cities with at least 10 different gas stations, to avoid taking averages of very small numbers.

[71]:
```
#Filter
min_lim = 10 #At least min_lim gas stations per city
missing_by_city_filtered = missing_by_city[missing_by_city.
 ↪Number_of_Gas_Stations >= min_lim]

#Order
missing_by_city_filtered = missing_by_city_filtered.
 ↪sort_values('is_missing_buy_price', ascending = False)

#Print the cities with the highest probability of missing data
print("The cities with the highest probability of missing data")
print(missing_by_city_filtered.head(5))

#Print the cities with the lowesr probability of missing data
print("The cities with the lowest probability of missing data")
print(missing_by_city_filtered.tail(5))
```

```
The cities with the highest probability of missing data
                              is_missing_buy_price  Number_of_Gas_Stations
City                 Region
VIDEIRA              S                    0.967033                      12
PRESIDENTE VENCESLAU SE                   0.949495                      11
ITABAIANA            NE                   0.937824                      11
SANTA ROSA           S                    0.922222                      15
JI-PARANA            N                    0.912536                      25
The cities with the lowest probability of missing data
                               is_missing_buy_price  Number_of_Gas_Stations
City                  Region
NITEROI               SE                   0.015534                      48
SAO GONCALO           SE                   0.006787                      52
PIRACICABA            SE                   0.004057                      37
BOA VISTA             N                    0.002959                      52
CAMPOS DOS GOYTACAZES SE                   0.000000                      82
```

As we can see, there is substantial variation in the fraction of missing data accross cities. While part of this is explained by the number of gas stations in the city, a substantial amount of the variability remains to be explained. Possibly part of this variation is due to some fixed effect on the city-level, but this is something we will not investigate here. Rather, we shall now move on the consider the role of time.

Namely, is the fraction of omission constant throughout time

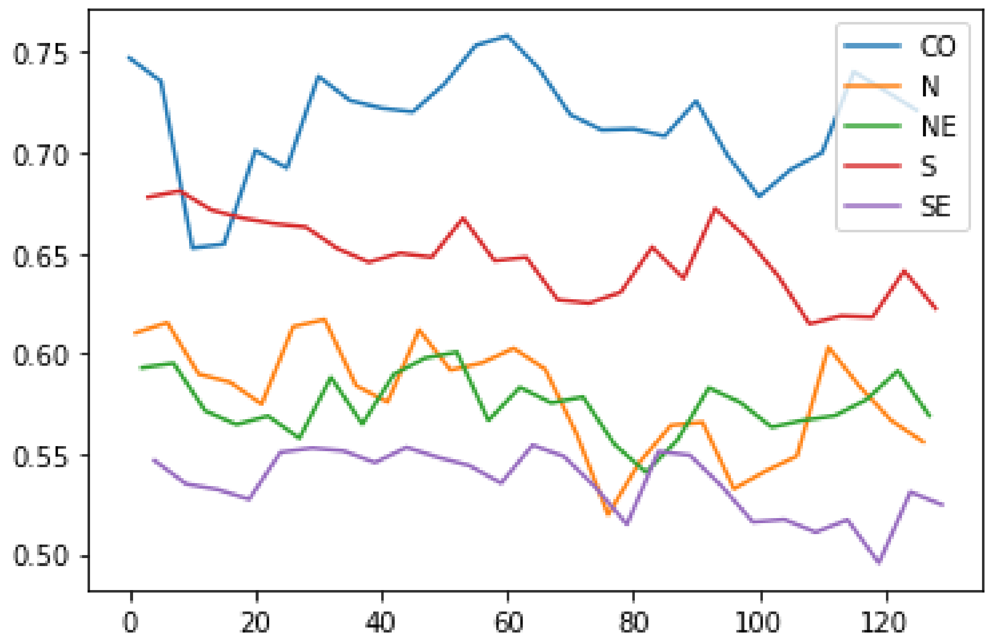
[72]:
```
#Define series
time_series = data.loc[:,["Week","is_missing_buy_price"]].groupby("Week").mean()
```

```
#Plot
plt.plot(time_series)
plt.title("Incidence of missing data through time")
plt.ylabel("Fraction of gas stations \n with missing data on buy price")
plt.xlabel("Week of year")
plt.show()
```

The incidence of missing data seems to be going down, which is a good thing! This may be because government is becoming more efficient collecting data, or because gas stations are losing the power to omit such information, or yet because of some other reason. But it's both interesting and positive that information is becoming more and more complete.

Let's split the overall trend by region.

[73]:
```
#Define series
time_series = data.loc[:,["Week","Region","is_missing_buy_price"]].
 ↪groupby(["Week","Region"], as_index = False).mean()

regions = time_series.Region.unique()
for region in regions:
    plt.plot(time_series.loc[time_series.Region == region,␣
 ↪"is_missing_buy_price"])
plt.legend(regions)
plt.show()
```



The decreasing trend is less clearer when we look at each region separately.

The Center-West (CO), where we had previously seen that is where missing data is most proeminent, does not exhibit any decreasing trend.

Such trend is found mostly on the South and South East, the richest portions of the country, where probably most of the gas stations are. Let's see if this is true:

[74]:
```
nb_gas_stations.groupby("Region").sum().sort_values("Number_of_Gas_Stations",␣
    ↪ascending = False)
```

[74]:
```
        Number_of_Gas_Stations
Region
SE                        7435
NE                        2716
S                         2164
CO                        1234
N                          892
```

The regions with the most number of gas stations are the South-East (SE), the North-East (NE) and the South (S). Two of these exhibit a clear negative trend.

The Center-West (CO) does not exhibit such a negative trend, but it is much smaller then the previously cited regions, so it doesn't weight that much on the overall trend.

Either way, this graph is interesting, as it shows us that the previous graph has to be interpreted with caution. One should not interpret that our dataset has become increasingly more complete, as we were tempted to interpret it. Rather, one should interpret that our dataset has become increasingly more complete *in some regions*, namely, in the regions were it was the most complete to begin with.

This means that, if we are to consider a multiple imputation model to account for the missing data in Buy_Price, we should consider an interaction term between the region where the gas station is located and a time component.

Let's summarize our findings.

My goal was to verify if missing data on variable *Buy_Price* was missing at random or not.

Our answer to this question is that *Buy_Price* is **missing NOT at random**. In particular, our exploratory data analysis showed us that:

1. The incidence of missing values is not homogeneous among the country's regions;
2. Gas stations are more likely to omit information on their buy price in cities where there are few gas stations;
3. The incidence of missing values has been reducing over time, but only in some regions. Since these regions account for the majority of the country's gas stations, this is reflected on the national trend. However, the regions where missing values are the most frequent are not being part of this downward trend.

```
[ ]:
```