1
```
# import libraries
import seaborn as sns
import pandas as pd
import xml.etree.ElementTree as et
import sys

#import data
xtree = et.parse("hw2-patients.xml")
xroot = xtree.getroot()
```

2
```
# organizing dataset into dataframe
col_name = ["name", "age", "gender", "full_age"]
patient_info = []
for node in xroot[2]:
    pname = node.attrib.get("name")
    pfull_age = float(node.attrib.get("age"))
    page = round(pfull_age,1)
    pgender = node.attrib.get("gender")
    patient_info.append({"name":pname, "age": page, "gender":pgender, "full_age":pfull_age})

df = pd.DataFrame(patient_info, columns=col_name)
df.head()
```

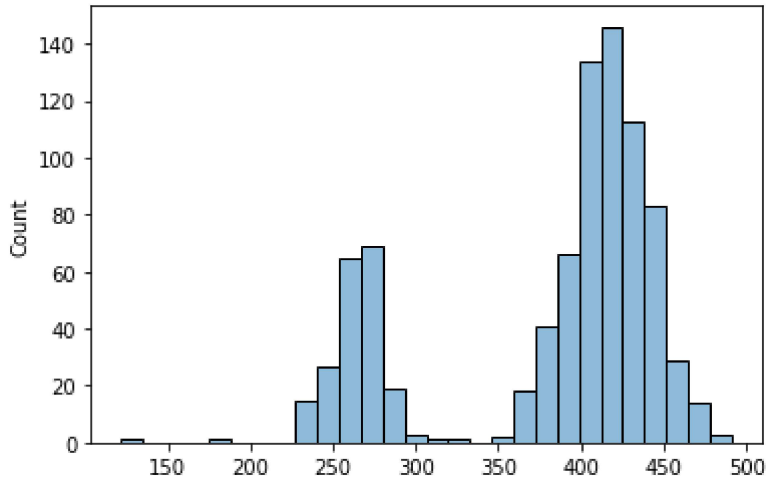
2

| | name | age | gender | full_age |
|---|---|---|---|---|
| **0** | Tammy Martin | 19.5 | female | 19.529988 |
| **1** | Lucy Stribley | 1.6 | female | 1.602197 |
| **2** | Albert Trevino | 19.3 | male | 19.317023 |
| **3** | Troy Armour | 79.4 | male | 79.441208 |
| **4** | Jose Masseria | 71.2 | male | 71.203863 |

3
```
# organize dataset by age
ages = df["age"].value_counts()
age_df = pd.DataFrame(ages)
age_df.reset_index()

# plot histogram of ages
sns.histplot(data=age_df, legend=False)
```

3
```
<AxesSubplot:ylabel='Count'>
```



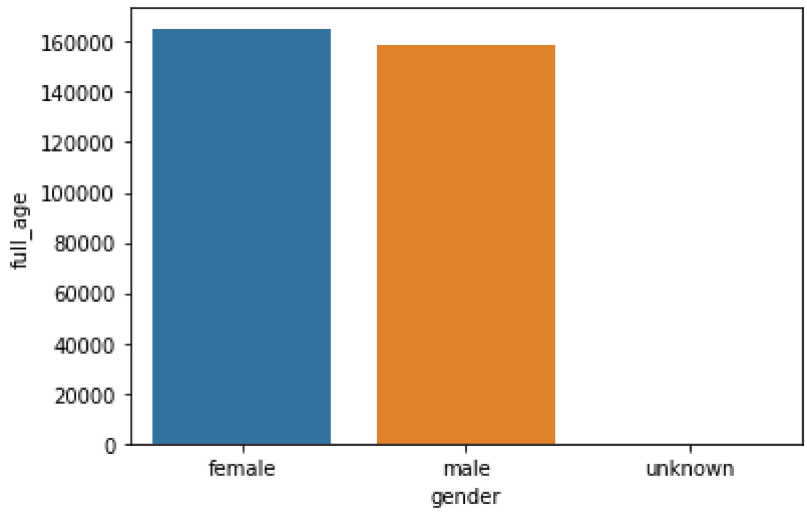

4
```
# find whether any patients share the exact same age
df["unique_exact_age"] = df["full_age"].duplicated(keep=False)
is_same = df[df["unique_exact_age"] == True]
duplicates = is_same['full_age'].count() != 0
print("The statement that any two or more patients share the same exact age is", duplicates)
```

```
The statement that any two or more patients share the same exact age is False
```

5
```
# find the distribution of genders
genders = df.groupby("gender")["full_age"].count().reset_index()
genders
print("Provider encoded gender as female, male, and unknown.")
```

```
Provider encoded gender as female, male, and unknown.
```

6
```
# plot genders
gender_plot = sns.barplot(data=genders, x='gender', y='full_age')
```




7
```
# sort ages
sorted_ages = sorted(df["full_age"])
print("The oldest patient is", sorted_ages[-1], "years old.")
```

```
The oldest patient is 84.99855742449432 years old.
```

8
```
# code for finding second oldest person in O(n) time in unsorted list
def find_second_oldest(list):
```

```
    oldest = 0
    second_oldest = 0
    for i in list:
       if i > oldest: oldest = i
       elif i > second_oldest: second_oldest = i
    return second_oldest
print("Second oldest is", find_second_oldest(df['full_age']))
```

```
Second oldest is 84.9982928781625
```

```
# find patient who is 41.5 years old
length = len(df.index)
p = length//2
target = df.iloc[p]['age']
while target != 41.5:
    if target > 41.5: p = p//2
    elif target < 41.5: p = (length - p)//2
    target = df.iloc[p]['age']
matched_age_patient = df.iloc[p]
print(matched_age_patient['name'])
```

```
James Robertson
```

```
# find number of patients at least 41.5 years old
sorted_rounded_ages = sorted(df['age'])
index_of_age = sorted_rounded_ages.index(41.5)
print("There are", length - index_of_age + 1, "people who are at least 41.5 years old in the dataset")
```

```
There are 150677 people who are at least 41.5 years old in the dataset
```

```
# generalized function returning number of patients who are
# at least low_age years old but strictly less than high_age years old
# in sorted list lst
def num_patients_in_age_range(lst, low_age, high_age):
    ph = len(lst)//2
    pl = 0
    pl_found = False
    ph_found = False
    carry = 0

    # base cases
    if len(lst) == 0: return 0
    if(lst[-1] < low_age): return 0
    if lst[0] >= low_age:
        pl_found = True
        pl = 0
    if lst[-1] <= high_age:
        ph_found = True
        ph = len(lst) - 1
        while lst[ph-1] == lst[ph]: ph -= 1
        if lst[ph] < high_age: carry = 1
    if(pl_found and ph_found): return ph-pl+carry

    # non-base case
    # find upper limit
    while not ph_found:
        # increment/decrement higher pointer
        if lst[ph] > high_age: ph = ph//2
        elif lst[ph] < high_age:
            # if value does not exist return index of lowest value higher than high_age
            if ph >= len(lst) - 1: ph_found = True
            elif lst[ph+1] > high_age: ph_found = True
            else: ph += (len(lst)-ph)//2
```

```
        # if value matched high_age return value
        if lst[ph] == high_age:
            ph_found = True

    # find lower limit
    while not pl_found :
        # increment/decrement lower pointer
        if lst[pl] > low_age:
            # if value does not exist return index of highest value lower than low_age
            if lst[pl-1] < low_age: pl_found = True
            else: pl = pl//2
        elif lst[pl] < low_age: pl += (ph - pl)//2

        # if value matched low_age return value
        if lst[pl] == low_age: pl_found = True

    if pl_found and pl == 0: carry = 1
    return ph-pl+carry

# test function
test_1_lst = [1, 2, 3, 4, 5, 6, 7]
test_2_lst = [0, 0, 0, 1, 3, 5, 7, 8]
test_3_lst = sorted_rounded_ages[1820:1830]
test_4_lst = [0, 1, 3, 5, 7, 8, 8]
test_5_lst = [0, 1]
test_6_lst = [1,2,3]
print("Test 1: testing consecutive numbers \n", num_patients_in_age_range(test_1_lst, 2, 5), "\n Correct
print("Test 2: testing duplicate numbers and high_age > lst[-1] \n", num_patients_in_age_range(test_2_lst
print("Test 3: testing subset of df \n", num_patients_in_age_range(test_3_lst, 0.3, 0.4), "\n Correct num
print("Test 4: testing duplicate high_age \n", num_patients_in_age_range(test_4_lst, 3, 8), "\n Correct n
print("Test 5: testing list of two elements \n", num_patients_in_age_range(test_5_lst, 0, 1), "\n Correct
print("Test 6: testing range not in list \n", num_patients_in_age_range(test_6_lst, 8, 10), "\n Correct n
```

```
Test 1: testing consecutive numbers
 3
 Correct number should be 3

Test 2: testing duplicate numbers and high_age > lst[-1]
 4
 Correct number should be 4

Test 3: testing subset of df
 2
 Correct number should be 2

Test 4: testing duplicate high_age
 3
 Correct number should be 3

Test 5: testing list of two elements
 1
 Correct number should be 1

Test 6: testing range not in list
 0
 Correct number should be 0
```

13

```
# return total number of patients and males in age range
def num_patients_males_in_age_range(dataframe, low_age, high_age):
    sorted_df = dataframe.sort_values(by=['age'])
    lst = sorted_df["age"].reset_index(drop=True)
    ph = len(lst)//2
    pl = 1
```

```
    pl_found = False
    ph_found = False
    carry = 0

    # base cases
    if (len(lst) == 0) or (lst[len(lst) - 1] < low_age):
        print("0 patients match description")
        sys.exit("Base case was reached, program terminated")
    if lst[1] >= low_age:
        pl_found = True
        pl = 0
    if lst[len(lst) - 1] <= high_age:
        ph_found = True
        ph = len(lst) - 1
        while lst[ph-1] == lst[ph]: ph -= 1
        if lst[ph] < high_age: carry = 1
    if(pl_found and ph_found): return ph-pl+carry

    # non-base case
    # find upper limit
    while not ph_found:
        # increment/decrement higher pointer
        if lst[ph] > high_age: ph = ph//2
        elif lst[ph] < high_age:
            # if value does not exist return index of lowest value higher than high_age
            if ph >= len(lst) - 1: ph_found = True
            elif lst[ph+1] > high_age: ph_found = True
            else: ph += (len(lst)-ph)//2

        # if value matched high_age return value
        if lst[ph] == high_age:
            ph_found = True

    # find lower limit
    while not pl_found :
        # increment/decrement lower pointer
        if lst[pl] > low_age:
            # if value does not exist return index of highest value lower than low_age
            if lst[pl-1] < low_age: pl_found = True
            else: pl = pl//2
        elif lst[pl] < low_age: pl += (ph - pl)//2

        # if value matched low_age return value
        if lst[pl] == low_age: pl_found = True

    if pl_found and pl == 0: carry = 1
    print("Total number of patients in age range", pl, "to", ph, "is", ph-pl+carry)
    range_df = sorted_df.iloc[pl-1:ph-1]
    males_in_range = range_df[range_df['gender']=='male']
    print("Total number of males in age range is", males_in_range['gender'].count())

# test function
print("Testing using first 10 rows of dataframe df: \n")
print("Dataset 1: \n", df.iloc[0:10].reset_index(drop=True))
num_patients_males_in_age_range(df.iloc[1:11], 0, 10)

print("Testing using last 10 rows of dataframe df: \n")
print("Dataset 2: \n", df.iloc[-11:-1].reset_index(drop=True))
num_patients_males_in_age_range(df.iloc[-11:-1], 30, 50)

print("Testing range not in list: \n")
print("Dataset 3: \n", df.iloc[-11:-1].reset_index(drop=True))
num_patients_males_in_age_range(df.iloc[-11:-1], 80, 90)

Testing using first 10 rows of dataframe df:
```

```
Dataset 1:
               name   age  gender   full_age  unique_exact_age
0      Tammy Martin  19.5  female  19.529988             False
1     Lucy Stribley   1.6  female   1.602197             False
2    Albert Trevino  19.3    male  19.317023             False
3       Troy Armour  79.4    male  79.441208             False
4     Jose Masseria  71.2    male  71.203863             False
5   Ethel Ferdinand  77.3  female  77.302707             False
6  Elizabeth Heflin   4.2  female   4.246844             False
7    Jesica Gautier  24.3  female  24.334514             False
8       Marta Allen  26.5  female  26.513040             False
9      Thomas Hagan  32.3    male  32.326220             False
Total number of patients in age range 0 to 1 is 2
Total number of males in age range is 0
Testing using last 10 rows of dataframe df:

Dataset 2:
                 name   age  gender   full_age  unique_exact_age
0    Patricia Edwards  21.7  female  21.690118             False
1      Kathy Streeter  48.9  female  48.935217             False
2  Bernice Sturdivant  11.3  female  11.347592             False
3         Percy Whitt  39.1    male  39.128485             False
4       Marcus Atkins  29.9    male  29.935480             False
5       Lillian Greig  56.2  female  56.241336             False
6        Jeremy Brode  61.0    male  60.955355             False
7         Lynda Brown  22.7  female  22.676277             False
8        Joyce Adkins  64.5  female  64.466378             False
9       Kevin Hensley  56.8    male  56.770128             False
Total number of patients in age range 4 to 5 is 1
Total number of males in age range is 1
Testing range not in list:

Dataset 3:
                 name   age  gender   full_age  unique_exact_age
0    Patricia Edwards  21.7  female  21.690118             False
1      Kathy Streeter  48.9  female  48.935217             False
2  Bernice Sturdivant  11.3  female  11.347592             False
3         Percy Whitt  39.1    male  39.128485             False
4       Marcus Atkins  29.9    male  29.935480             False
5       Lillian Greig  56.2  female  56.241336             False
6        Jeremy Brode  61.0    male  60.955355             False
7         Lynda Brown  22.7  female  22.676277             False
8        Joyce Adkins  64.5  female  64.466378             False
9       Kevin Hensley  56.8    male  56.770128             False
0 patients match description

An exception has occurred, use %tb to see the full traceback.

SystemExit: Base case was reached, program terminated

C:\Users\olina\AppData\Local\Programs\Python\Python39\lib\site-packages\IPython\core\interactiveshell.py:
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

Question 2 below this page

```
83  # import libraries
    import seaborn as sbs
    import pandas as pd

    # import data using code given
    from Bio import SeqIO
    human_genome = SeqIO.parse("GCA_000001405.28_GRCh38.p13_genomic.fna", "fasta")
```

```
57  # used code given to help loop through the entire chromosome across
    # all 15-mers and find the total number of subsequences with more than 2 N's
    less_than_two_n = 0
    for chromosome in human_genome:
        if chromosome.name == "CM000664.2":
            sequence = str(chromosome.seq).lower().encode('utf8', errors='ignore')
            s_len = len(sequence)

            # if length of sequence <= 15, then do not enter while loop
            if s_len <= 15 and sequence.count('n'.encode('utf8')) < 3:
                less_than_two_n += 1
                continue
            # if length of sequence > 15, evaluate each 15-char substring
            i = 0
            while i + 15 <= s_len:
                subseq = sequence[i:i+15]
                if subseq.count('n'.encode('utf8')) < 3: less_than_two_n += 1
                i += 1
    print(less_than_two_n)
```

```
    240548031
```

```
58  # estimate the number of distinct 15-mers in the reference genome's chromosome 2
    # using method discussed in class
    # using 100 hash functions from the given family in a single pass through the sequences
    p = 2_549_536_629_329
    bits_48 = 2 ** 48 - 1
    scale = 0x07fffffff
    from hashlib import sha256
    def get_ath_hash(a):
        def my_hash(subseq):
            return (((int(sha256(subseq.encode("utf8")).hexdigest(), 16) % bits_48) * a) % p) & scale
        return my_hash
```

```
*   def distinct_vals(seq, a):
        if len(seq) < 15: return 0
        i = 0
        hashes = set()
        h_num = get_ath_hash(a)
        while i + 15 <= len(seq):
            subseq = seq[i:i+15]
            i += 1
            # print(subseq, i)
            hashes.add(h_num(subseq))
        return len(hashes)

    # test on simple data
    data_1 = "abcdefghijklmnopqabcdefghijklmno"
    data_2 = "ctagctagctagctagctagcccccccccccc"
    data_3 = "acacacacacacacacacacacacacacacac"
    print("test 1: Correct answer should be 17")
    print("Observed answer is", distinct_vals(data_1, 10), "\n")
```

```
print("test 2: Correct answer should be 15")
print("Observed answer is", distinct_vals(data_2, 10), "\n")
print("test 3: Correct answer should be 2")
print("Observed answer is", distinct_vals(data_3, 10), "\n")

# test on chromosome
for chromosome in human_genome:
    if chromosome.name == "CM000664.2":
        sequence = str(chromosome.seq).lower()
        # note that 1 billion is approxiamtely 2^30
        print("test 4: trying algorithm on chromosome 2")
        print(distinct_vals(sequence, 30))
```

```
test 1: Correct answer should be 17
Observed answer is 17

test 2: Correct answer should be 15
Observed answer is 15

test 3: Correct answer should be 2
Observed answer is 2

test 4: trying algorithm on chromosome 2
144698064
```