Solutions to Homework 2

Question 1: 10 points.

Problem Statement. An experiment is conducted on 36 specimens to determine the tensile yield strength of `A36` steel. Write a Python program that will:

1. Read the experimental test results from a file `steelA36.csv` into a Pandas dataframe.
2. Compute and print the maximum, minimum, and average tensile strengths.
3. Use the Pandas cut function (i.e., google `pd.cut()`) to organize the data into intervals covering the ranges: 36-38, 38-40, 40-42, 42-44, 44-46, 46-48.
4. Generate a histogram of "observations" versus "tensile yield stress."
5. Construct a stair-step graph of "cumulative frequency" versus "yield stress."

Note. The average value of the experimental results can be computed using Python's buildin functions. The "cumulative frequency" versus "yield stress" is given by

$$\text{Cumulative frequency(y)} = \int_0^y p(x)dx \tag{1}$$

where $p(x)$ is the probability distribution of tensile yield strengths. The matplotlib functions `plt.hist(.)` and `plt.step(.)` create histogram and stair-step graphs.

Python Source Code:

```
# ================================================================================
# TestDataProcessingA36Steel.py: Read, process and visualize data from data/steelA36.csv.
#
# Written by: Mark Austin                                                February, 2024
# ================================================================================

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import axes3d

from pandas import DataFrame
from pandas import read_csv

# ============================================================
# Main function ...
# ============================================================

def main():
    print("--- Enter TestDataProcessingA36Steel.main() ... ");
    print("--- ============================================ ... ");
    print("");

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load A36steel data file ... ");
    print("--- ");

    df = pd.read_csv('data/steelA36.csv')
    print(df)

    # Dataframe info and shape ...

    print( df.info() )
    print( df.shape )

    print("--- ");
    print("--- Part 02: Transform dataframe to numpy array ... ");
    print("--- ");

    strength = np.array ( df['Strength'].values )

    print("--- Raw data array ...");
    print( strength )

    print("--- Transform data array to list ...");
    print( strength.tolist() )

    strength_sorted = np.sort( strength )

    print("--- Sort array ...");
    print( strength_sorted )

    print("--- ");
    print("--- Part 03: Compute basic statistics ... ");
    print("--- ");

    print("--- Min strength     = {:5.2f} ...".format(min(strength)) );
    print("--- Max strength     = {:5.2f} ...".format(max(strength)) );
    print("--- Average strength = {:5.2f} ...".format(sum(strength)/len(strength)) );

    print("--- ");
    print("--- Part 03: Organize strength data into intervals: 32-36, 36-40, 40-44, 44-48  ... ");
```

```
    print("--- ");

    sinterval = [ "36-38", "38-40", "40-42", "42-44", "44-46", "46-48" ]
    steel_strength_intervals = pd.cut( strength_sorted, [ 36, 38, 40, 42, 44, 46, 48 ], labels = sint

    # Retrieve interval categories and codes ...

    labels     = steel_strength_intervals.codes
    categories = steel_strength_intervals.categories

    # Systematically print the interval for each category ...

    for index in range(len(strength_sorted)):
        label_index = labels[index]
        print( strength_sorted[index], label_index, categories[label_index] )

    print("--- ");
    print("--- Part 04: Create histogram of steel strengths ... ");
    print("--- ");

    nbins = 20;
    plt.hist( strength_sorted, nbins );
    plt.title('Tensile Yield Strength of A36 Steel');
    plt.xlabel('Tensile Yield Strength');
    plt.ylabel('No Observations');
    plt.grid()
    plt.show()

    print("--- ");
    print("--- Part 05: Generate cumulative frequency data and graph ... ");
    print("--- ");

    # Generate cumulative probability distribution ....

    npoints = len( strength_sorted );
    print("--- No data points = {:d} ...".format( npoints ));
    cumulative_probability = np.linspace( 0.0, 1.0, npoints, endpoint=True );

    # Step plot of cumulative probability vs yield strength ...

    plt.step( strength_sorted, cumulative_probability );
    plt.title('Cumulative probability distribution for Yield Strength of A36 Steel');
    plt.xlabel('Tensile Yield Strength');
    plt.ylabel('Cumulative Probability');
    plt.grid()
    plt.show()

    print("--- ============================================== ... ");
    print("--- Leave TestDataProcessingA36Steel.main() ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Program Output:

```
--- Enter TestDataProcessingA36Steel.main() ...
--- ============================================== ...
---
--- Part 01: Load A36steel data file ...
---
    Sample  Strength
0        1      42.3
1        2      42.1

... lines of data removed ...

34      35      46.3
35      36      39.5
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Sample    36 non-null     int64
 1   Strength  36 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 708.0 bytes
None
(36, 2)

--- Part 02: Transform dataframe to numpy array ...

--- Raw data array ...
[42.3 42.1 41.8 42.4 47.7 41.4 40.5 38.7 40.8 39.6 42.4 37.5 39.9 45.3
 41.6 36.8 45.4 44.8 39.2 40.7 38.5 40.1 42.8 42.5 43.1 36.2 46.2 41.5
 38.3 40.2 41.9 40.4 39.1 38.6 46.3 39.5]
--- Transform data array to list ...
[42.3, 42.1, 41.8, 42.4, 47.7, 41.4, 40.5, 38.7, 40.8, 39.6, 42.4, 37.5,
 39.9, 45.3, 41.6, 36.8, 45.4, 44.8, 39.2, 40.7, 38.5, 40.1, 42.8, 42.5,
 43.1, 36.2, 46.2, 41.5, 38.3, 40.2, 41.9, 40.4, 39.1, 38.6, 46.3, 39.5]
--- Sort array ...
[36.2 36.8 37.5 38.3 38.5 38.6 38.7 39.1 39.2 39.5 39.6 39.9 40.1 40.2
 40.4 40.5 40.7 40.8 41.4 41.5 41.6 41.8 41.9 42.1 42.3 42.4 42.4 42.5
 42.8 43.1 44.8 45.3 45.4 46.2 46.3 47.7]

--- Part 03: Compute basic statistics ...

--- Min strength     = 36.20 ...
--- Max strength     = 47.70 ...
--- Average strength = 41.28 ...

--- Part 03: Organize strength data into intervals: 32-36, 36-40, 40-44, 44-48  ...

36.2 0 36-38
36.8 0 36-38
37.5 0 36-38
38.3 1 38-40
```

```
.... lines of output removed ...

45.4 4 44-46
46.2 5 46-48
46.3 5 46-48
47.7 5 46-48

--- Part 04: Create histogram of steel strengths ...

--- Part 05: Generate cumulative frequency data and graph ...

--- No data points = 36 ...
--- ============================================== ...
--- Leave TestDataProcessingA36Steel.main() ...
```

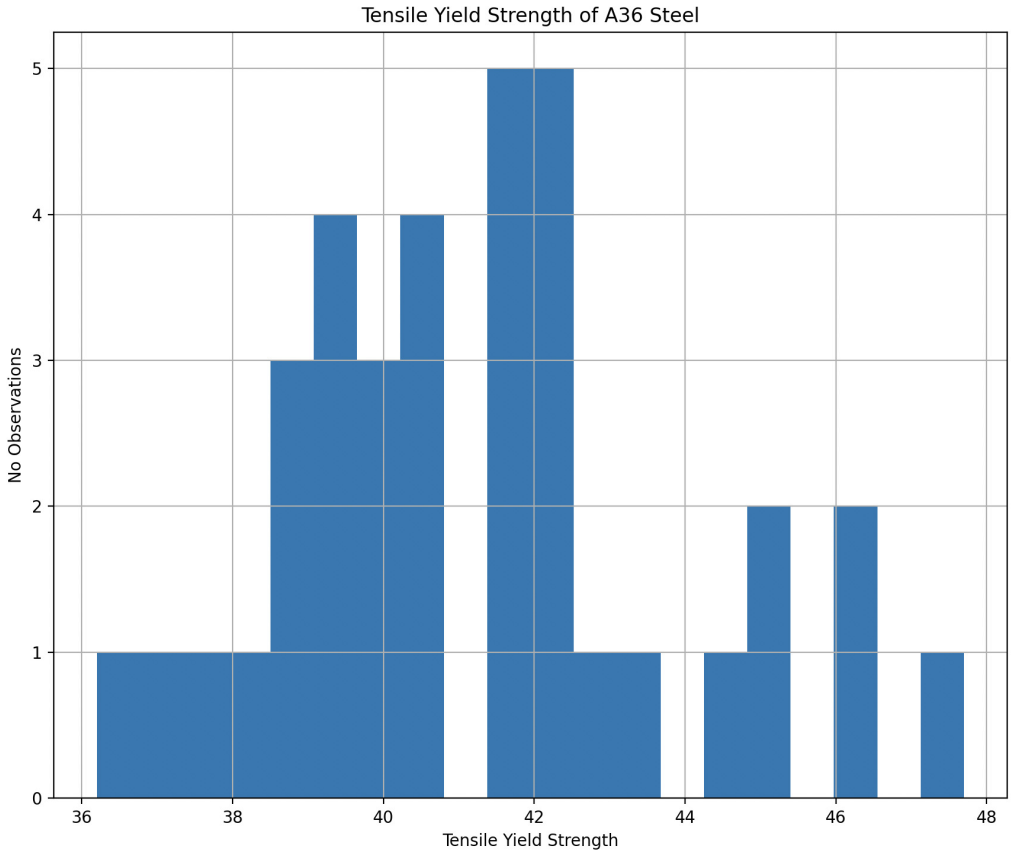

Figure 1: Tensile yield strength of A36 Steel.

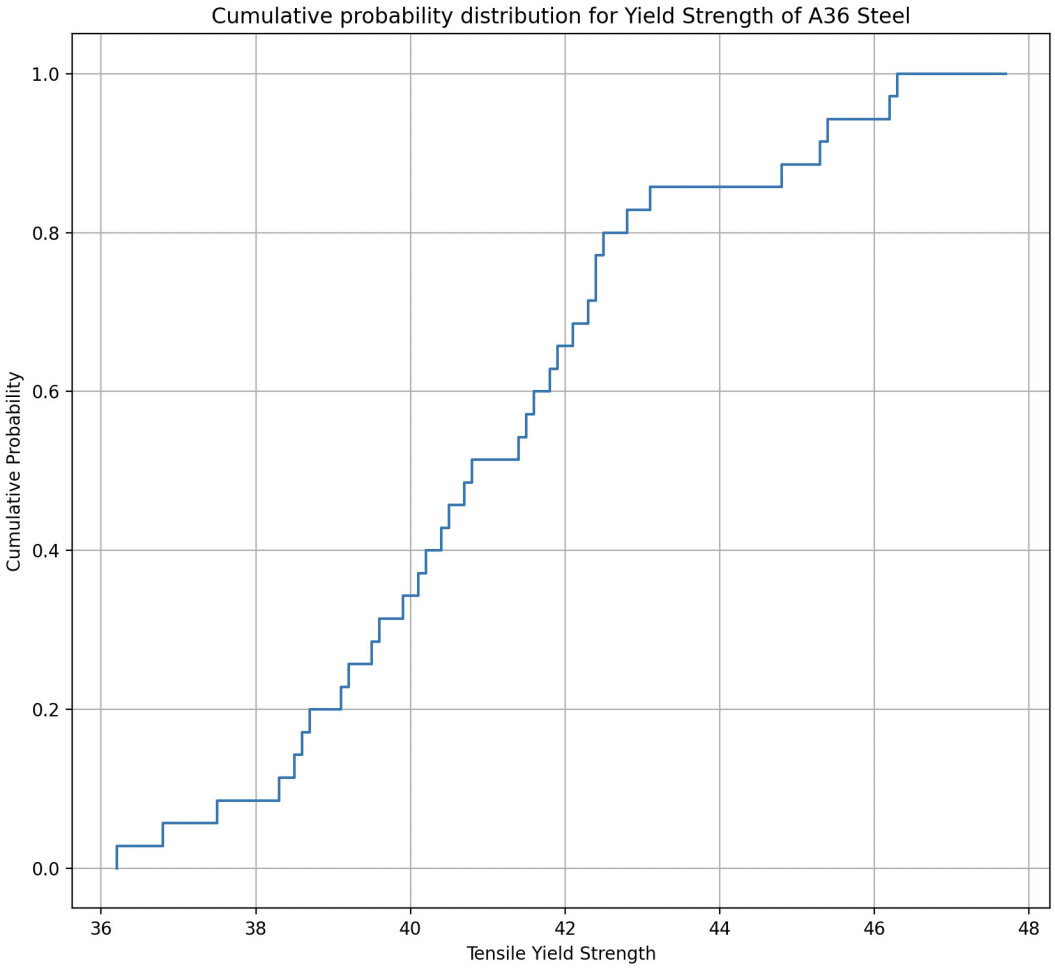

Figure 2: Cumulative probability distribution for tensile yield strength of A36 Steel.

Question 2: 10 points.

Problem Statement: Let dx be a floating point number whose magnitude is very small compared to 1. Write a Python program that will systematically evaluate the expression

$$f(dx) = \sqrt{1+dx} \; - \; \sqrt{1-dx} \tag{2}$$

for $|dx| \rightarrow 0$. Demonstrate that errors due to subtractive cancellation can be avoided by rewriting equation 2 as

$$g(dx) = \left[\frac{2dx}{\sqrt{1+dx} + \sqrt{1-dx}} \right]. \tag{3}$$

You should use math function `math.sqrt()` for the square root evaluations.

Note. I suggest that you set dx = 1, and then evaluate equations 2 and 3. Then, decrease dx by a factor of 10 and repeat the experiment. You should find that equations 2 and 3 will evaluate to the same value until the limits of double precision floating point storage are reached. And then, whereas evaluation of equation 2 will truncate to zero, equation 3 will evaluate to dx. It is relatively straight forward to show via a Taylor series expansion that dx is correct.

Python Source Code:

```
# ============================================================
# TestSubtractiveCancellation02.py: Demonstratve subtractive
# cancellation on a simple formula:
#
#     f(dx) = math.sqrt(1+dx) - math.sqrt(1-dx)
#
# An equivalent formula that avoids subtractive cancellation is:
#
#     g(dx) = 2dx/(math.sqrt(1+dx) + math.sqrt(1-dx))
#
# Written by: Mark Austin                          February 2024
# ============================================================

import math

# Function f(dx) ....

def f(dx):
   return math.sqrt(1+dx) - math.sqrt(1-dx)

# Function g(dx) ....

def g(dx):
```

```
   return 2*dx/(math.sqrt(1+dx) + math.sqrt(1-dx))

# Main function ...

def main():
    print("--- Enter TestSubtractiveCancellation02.main() ... ");
    print("--- ======================================== ... ");
    print("");

    # Let x = 1 and dx become progressively smaller ...

    print("Subtractive Cancellation Experiment ...")
    print("-------------------------------------------------  ")
    print("               dx           f(dx)            g(dx)  ")
    print("=================================================  ")

    dx = 1.0;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.1;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.01;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.0001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.00001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.0000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.00000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.0000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.00000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.0000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.00000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.000000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.0000000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.00000000000000001;
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))
    dx = 0.000000000000000001;
```

```
    print("--- {:13.6e}  {:11.8e}   {:11.8e} ".format( dx, f(dx), g(dx)))

    print("================================================  ")

    print("--- ========================================== ... ");
    print("--- Leave TestSubtractiveCancellation02.main() ... ");

# call the main method ...

main()
```

Program Output: The textual output is:

```
--- Enter TestSubtractiveCancellation02.main() ...
--- ========================================== ...

Subtractive Cancellation Experiment ...
-------------------------------------------------
               dx           f(dx)             g(dx)
=================================================
---  1.000000e+00  1.41421356e+00   1.41421356e+00
---  1.000000e-01  1.00125550e-01   1.00125550e-01
---  1.000000e-02  1.00001250e-02   1.00001250e-02
---  1.000000e-03  1.00000013e-03   1.00000013e-03
---  1.000000e-04  1.00000000e-04   1.00000000e-04
---  1.000000e-05  1.00000000e-05   1.00000000e-05
---  1.000000e-06  1.00000000e-06   1.00000000e-06
---  1.000000e-07  1.00000000e-07   1.00000000e-07
---  1.000000e-08  1.00000001e-08   1.00000000e-08
---  1.000000e-09  1.00000008e-09   1.00000000e-09
---  1.000000e-10  1.00000008e-10   1.00000000e-10
---  1.000000e-11  1.00000008e-11   1.00000000e-11
---  1.000000e-12  1.00008890e-12   1.00000000e-12
---  1.000000e-13  1.00031095e-13   1.00000000e-13
---  1.000000e-14  9.88098492e-15   1.00000000e-14
---  1.000000e-15  9.99200722e-16   1.00000000e-15
---  1.000000e-16  1.11022302e-16   1.00000000e-16
---  1.000000e-17  0.00000000e+00   1.00000000e-17
---  1.000000e-18  0.00000000e+00   1.00000000e-18
---  1.000000e-19  0.00000000e+00   1.00000000e-19
=================================================

--- ========================================== ...
--- Leave TestSubtractiveCancellation02.main() ...
```

Question 3: 10 points.

Problem Statement: Figure 4 is a schematic of public-use airports located in Maryland.

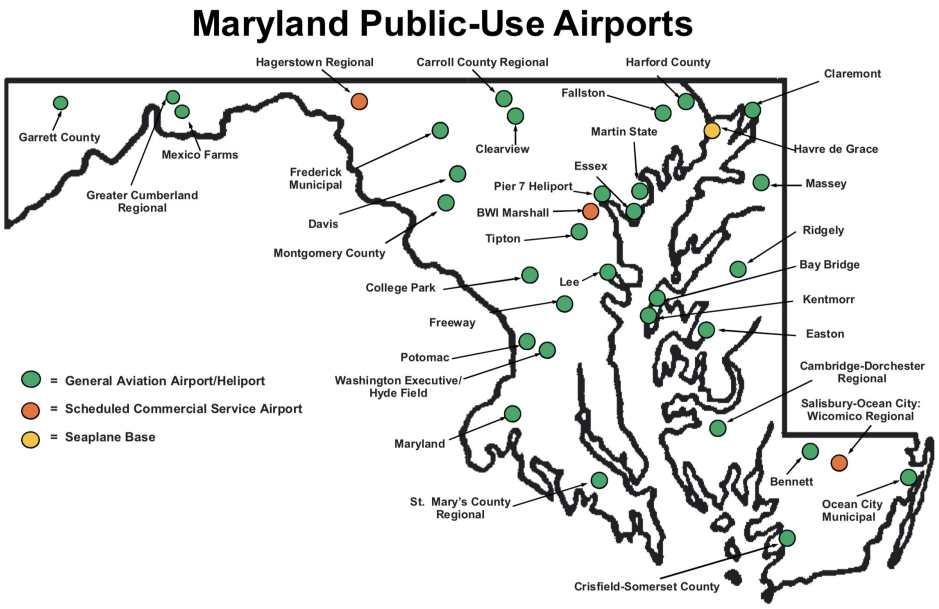

Figure 3: Maryland public-use airports (source: MD Airport Administration).

Write a Python program to assemble a **similar graphic** by pulling together – reading, filtering – and visualizing data sources for: (1) airports in the US, (2) boundary data for the state of MD, and (3) boundary data for the coastline.

You should read and store the airport data in a Pandas dataframe; then, employ a filter to isolate airports located in Maryland. You should use Pandas to store the airport data. The MD boundary and coastline data can be read directly into GeoPandas.

Note: The data file `python-code.d/data/airports-small.csv` contains a listing of 3,300 airports in the US. See `python-code.d/data/geography/maryland/` for boundary data for MD and the Chesapeake Bay. Finally, see the test examples in `python-code.d/geopandas/` for guidance on structuring your Python code.

Python Source Code:

```
# ================================================================================
# TestDataProcessingAirports01.py: Two purposes:
#
# 1. Read, process and visualize data from data/airports-small.csv (3,300 entries) ...
# 2. Use geopandas to display airports in Maryland ...
```

```
#
# Written by: Mark Austin                                                    February, 2024
# ============================================================================================

import numpy as np
import pandas as pd
import geopandas

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

from pandas import DataFrame
from pandas import Series
from pandas import read_csv

# ============================================================
# Main function ...
# ============================================================

def main():
    print("--- Enter TestDataProcessingAirports01.main()         ... ");
    print("--- ================================================= ... ");
    print("");

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load airports data file ... ");
    print("--- ");

    df = pd.read_csv('data/airports-small.csv')
    print(df)

    # Dataframe info and shape ...

    print( df.info() )
    print( df.shape )

    # Filter dataframe to keep only airports located in Maryland ...

    print("--- ");
    print("--- Part 02: Extract and print airports located in Maryland ... ");
    print("--- ");

    options = ['MD']
    dfMDairports = df [ df['state'].isin(options) ].copy()

    print(dfMDairports)

    # Dataframe info and shape ...

    print( dfMDairports.info() )
    print( dfMDairports.shape )

    print("--- ");
```

```
print("--- Part 03: Print individual rows of dfMD ... ");
print("--- ");

print("--- ");
print("--- Item  Iata  Aiport Name                                City          Latitude/Longitude
print("--- ================================================================================================

# Traverse rows of dataframe ...

i = 1
for index, row in dfMDairports.iterrows():
    iata   = str( row["iata"] );
    county = str( row["name"] );
    city   = str( row["city"] );
    lat    = row["latitude"];
    long   = row["longitude"];
    print("--- {:4d}:  {:3s}, {:39s}, {:13s} ({:f}, {:f}) ... ".format(i, iata, county, city, lat
    i = i + 1;

print("--- ================================================================================================
print("--- ");

print("--- ");
print("--- Part 04: Convert dfMD dataframe to list, then print ... ");
print("--- ");

# Convert dfMD dataframe to list ...

mdairportlist = dfMDairports.values.tolist();

# Traverse list, print details of individual airports ...

print("--- ");
print("--- Item  Iata  Aiport Name                                City          Latitude/Longitude
print("--- ================================================================================================

i = 1
for row in mdairportlist:
    iata   = str( row[0] );
    county = str( row[1] );
    city   = str( row[2] );
    lat    = row[5];
    long   = row[6];
    print("--- {:4d}:  {:3s}, {:39s}, {:13s} ({:f}, {:f}) ... ".format(i, iata, county, city, lat
    i = i + 1;

print("--- ================================================================================================
print("--- ");

print("--- ");
print("--- Part 05: Read Maryland boundary and coastline data files ... ")
print("--- ");

mdboundarydata = geopandas.read_file("data/geography/maryland/BNDY_StateBoundary_DoIT.shp")
mdboundarydata = mdboundarydata.to_crs(4326)
```

```
    mdcoastlinedata = geopandas.read_file("data/geography/maryland/BNDY_Shoreline_MGS.shp")
    mdcoastlinedata = mdcoastlinedata.to_crs(4326)

    print("--- MD boundary data ...");
    print(mdboundarydata.head())
    print(mdboundarydata.info())
    print(mdboundarydata.shape)

    print("--- MD coastline data ...");
    print(mdcoastlinedata.head())
    print(mdcoastlinedata.info())
    print(mdcoastlinedata.shape)

    print("--- ");
    print("--- Part 06: Define geopandas dataframes ... ")
    print("--- ");

    gdf01 = geopandas.GeoDataFrame(mdboundarydata)
    gdf02 = geopandas.GeoDataFrame(mdcoastlinedata)
    gdf03 = geopandas.GeoDataFrame( dfMDairports, geometry=geopandas.points_from_xy(dfMDairports.long

    print(gdf03)

    print("--- ");
    print("--- Part 07: Create boundary map for Maryland, then add airports ... ")
    print("--- ");

    # We can now plot our ``GeoDataFrame``.

    ax = gdf01.plot( color='white', edgecolor='black')
    ax.set_aspect('equal')
    ax.set_title("Airports in Maryland")

    gdf01.plot(ax=ax, color='white')
    gdf02.plot(ax=ax, edgecolor='green')
    gdf03.plot(ax=ax, color =  'red', markersize = 50, label= 'Airports')

    plt.legend('Airports:')
    plt.xlabel('longitude')
    plt.ylabel('latitude')
    plt.grid(True)
    plt.show()

    print("--- ================================================ ... ");
    print("--- Leave TestDataProcessingAirports01.main()         ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Program Output: The textual output is:

```
--- Enter TestDataProcessingAirports01.main()         ...
--- ================================================ ...
---
--- Part 01: Load airports data file ...
---

     iata                        name  ...   latitude   longitude
0     00M                     Thigpen  ...  31.953765  -89.234505
1     00R       Livingston Municipal  ...  30.685861  -95.017928
2     00V                 Meadow Lake  ...  38.945749 -104.569893
3     01G                Perry-Warsaw  ...  42.741347  -78.052081
4     01J           Hilliard Airpark  ...  30.688012  -81.905944
...   ...                         ...  ...        ...         ...
3371  ZEF            Elkin Municipal  ...  36.280024  -80.786069
3372  ZER  Schuylkill Cty/Joe Zerbey  ...  40.706449  -76.373147
3373  ZPH      Zephyrhills Municipal  ...  28.228065  -82.155916
3374  ZUN                Black Rock  ...  35.083227 -108.791777
3375  ZZV       Zanesville Municipal  ...  39.944458  -81.892105

[3376 rows x 7 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3376 entries, 0 to 3375
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   iata       3376 non-null   object
 1   name       3376 non-null   object
 2   city       3364 non-null   object
 3   state      3364 non-null   object
 4   country    3376 non-null   object
 5   latitude   3376 non-null   float64
 6   longitude  3376 non-null   float64
dtypes: float64(2), object(5)
memory usage: 184.8+ KB
None
(3376, 7)

---
--- Part 02: Extract and print airports located in Maryland ...
---
     iata                                    name  ...   latitude  longitude
250   2G4                         Garrett County  ...  39.580278 -79.339417
289   2W5                               Maryland  ...  38.600537 -77.072969
290   2W6     Captain Walter Francis Duke Regional  ...  38.315361 -76.550111
1030  BWI       Baltimore-Washington International  ...  39.175402 -76.668198
1070  CBE                   Cumberland Regional  ...  39.615417 -78.760864
1103  CGE                 Cambridge-Dorchester  ...  38.539306 -76.030361
1106  CGS                         College Park  ...  38.980583 -76.922306
1286  DMW                      Carroll County  ...  39.608278 -77.007667
1418  ESN                      Easton /Newnam  ...  38.804167 -76.069000
1498  FDK                   Frederick Municipal  ...  39.417583 -77.374306
1525  FME                               Tipton  ...  39.085387 -76.759414
1576  GAI                 Montgomery Co Airpark  ...  39.168336 -77.166000
1712  HGR       Hagerstown Regional-Richard Henson  ...  39.707944 -77.729500
2334  MTN                         Martin State  ...  39.325663 -76.413766
```

```
2559  OXB                                Ocean City  ...  38.310444 -75.123972
2896  SBY  Salisbury-Ocean City: Wicomico Regional  ...  38.340526 -75.510288
3269  W29                    Bay Bridge Industrial  ...  38.976389 -76.329639
3273  W41                      Crisfield Municipal  ...  38.016790 -75.828821

[18 rows x 7 columns]
<class 'pandas.core.frame.DataFrame'>
Index: 18 entries, 250 to 3273
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   iata       18 non-null     object
 1   name       18 non-null     object
 2   city       18 non-null     object
 3   state      18 non-null     object
 4   country    18 non-null     object
 5   latitude   18 non-null     float64
 6   longitude  18 non-null     float64
dtypes: float64(2), object(5)
memory usage: 1.1+ KB
None
(18, 7)

---
--- Part 03: Print individual rows of dfMD ...
---

--- Item  Iata  Aiport Name                                  City          Latitude/Longitude      ...
--- ================================================================================================ ...
---    1:  2G4, Garrett County                             , Oakland       (39.580278, -79.339417) ...
---    2:  2W5, Maryland                                   , Indian Head   (38.600537, -77.072969) ...
---    3:  2W6, Captain Walter Francis Duke Regional       , Leonardtown   (38.315361, -76.550111) ...
---    4:  BWI, Baltimore-Washington International         , Baltimore     (39.175402, -76.668198) ...
---    5:  CBE, Cumberland Regional                        , Cumberland    (39.615417, -78.760864) ...
---    6:  CGE, Cambridge-Dorchester                       , Cambridge     (38.539306, -76.030361) ...
---    7:  CGS, College Park                               , College Park  (38.980583, -76.922306) ...
---    8:  DMW, Carroll County                             , Westminster   (39.608278, -77.007667) ...
---    9:  ESN, Easton /Newnam                             , Easton        (38.804167, -76.069000) ...
---   10:  FDK, Frederick Municipal                        , Frederick     (39.417583, -77.374306) ...
---   11:  FME, Tipton                                     , Odenton       (39.085387, -76.759414) ...
---   12:  GAI, Montgomery Co Airpark                      , Gaithersburg  (39.168336, -77.166000) ...
---   13:  HGR, Hagerstown Regional-Richard Henson         , Hagerstown    (39.707944, -77.729500) ...
---   14:  MTN, Martin State                               , Baltimore     (39.325663, -76.413766) ...
---   15:  OXB, Ocean City                                 , Ocean City    (38.310444, -75.123972) ...
---   16:  SBY, Salisbury-Ocean City: Wicomico Regional, Salisbury     (38.340526, -75.510288) ...
---   17:  W29, Bay Bridge Industrial                      , Stevensville  (38.976389, -76.329639) ...
---   18:  W41, Crisfield Municipal                        , Crisfield     (38.016790, -75.828821) ...
--- ================================================================================================ ...

---
--- Part 04: Convert dfMD dataframe to list, then print ...
---

--- Item  Iata  Aiport Name                                  City          Latitude/Longitude      ...
--- ================================================================================================ ...
```

```
---    1:  2G4, Garrett County                          , Oakland       (39.580278, -79.339417) ...
---    2:  2W5, Maryland                                , Indian Head   (38.600537, -77.072969) ...
---    3:  2W6, Captain Walter Francis Duke Regional    , Leonardtown   (38.315361, -76.550111) ...
---    4:  BWI, Baltimore-Washington International      , Baltimore     (39.175402, -76.668198) ...
---    5:  CBE, Cumberland Regional                     , Cumberland    (39.615417, -78.760864) ...
---    6:  CGE, Cambridge-Dorchester                    , Cambridge     (38.539306, -76.030361) ...
---    7:  CGS, College Park                            , College Park  (38.980583, -76.922306) ...
---    8:  DMW, Carroll County                          , Westminster   (39.608278, -77.007667) ...
---    9:  ESN, Easton /Newnam                          , Easton        (38.804167, -76.069000) ...
---   10:  FDK, Frederick Municipal                     , Frederick     (39.417583, -77.374306) ...
---   11:  FME, Tipton                                  , Odenton       (39.085387, -76.759414) ...
---   12:  GAI, Montgomery Co Airpark                   , Gaithersburg  (39.168336, -77.166000) ...
---   13:  HGR, Hagerstown Regional-Richard Henson      , Hagerstown    (39.707944, -77.729500) ...
---   14:  MTN, Martin State                            , Baltimore     (39.325663, -76.413766) ...
---   15:  OXB, Ocean City                              , Ocean City    (38.310444, -75.123972) ...
---   16:  SBY, Salisbury-Ocean City: Wicomico Regional, Salisbury     (38.340526, -75.510288) ...
---   17:  W29, Bay Bridge Industrial                   , Stevensville  (38.976389, -76.329639) ...
---   18:  W41, Crisfield Municipal                     , Crisfield     (38.016790, -75.828821) ...
--- ========================================================================================== ...

---
--- Part 05: Read Maryland boundary and coastline data files ...
---
--- MD boundary data ...
   OBJECTID  ...                                           geometry
0         1  ...  POLYGON ((-76.23660 37.88653, -76.23714 37.887...

[1 rows x 5 columns]
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    1 non-null      int64
 1   State       1 non-null      object
 2   Shape_Leng  1 non-null      float64
 3   Shape_Area  1 non-null      float64
 4   geometry    1 non-null      geometry
dtypes: float64(2), geometry(1), int64(1), object(1)
memory usage: 172.0+ bytes
None
(1, 5)
--- MD coastline data ...
   OBJECTID  ...                                           geometry
0         1  ...  MULTIPOLYGON (((-76.12498 39.69505, -76.12495 ...

[1 rows x 5 columns]
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    1 non-null      int64
 1   Id          1 non-null      int64
 2   Shape_Leng  1 non-null      float64
```

```
 3   Shape_Area  1 non-null      float64
 4   geometry    1 non-null      geometry
dtypes: float64(2), geometry(1), int64(2)
memory usage: 172.0 bytes
None
(1, 5)
---
--- Part 06: Define geopandas dataframes ...
---
     iata  ...                       geometry
250   2G4  ...  POINT (-79.33942 39.58028)
289   2W5  ...  POINT (-77.07297 38.60054)
290   2W6  ...  POINT (-76.55011 38.31536)

... lines of output removed ...

3269  W29  ...  POINT (-76.32964 38.97639)
3273  W41  ...  POINT (-75.82882 38.01679)

[18 rows x 8 columns]
---
--- Part 07: Create boundary map for Maryland, then add airports ...
--- ================================================ ...
--- Leave TestDataProcessingAirports01.main()         ...
```

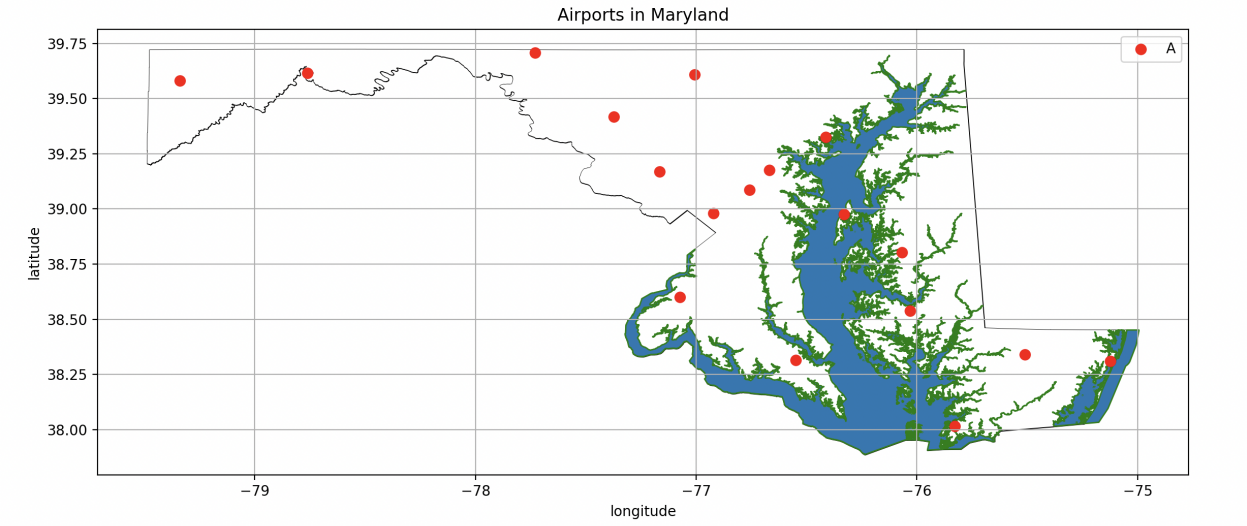

Figure 4: Using geopandas to model and visualize airports in Maryland.

Question 4: 10 points.

Problem Statement: The data file `python-code.d/data/titanic.csv` contains information on 887 of the passengers and their attributes, including:

```
--- Survived: 1 means passenger survived; 0 for victims.
--- Pclass:   1, 2 and 3 for first, second and third class.
--- Name:     Master/miss first name, family name.
--- Sex:      male or female.
--- Age:      covers the range 0 to 80.
--- Siblings/Spouses Aboard
--- Parents/Children Aboard
--- Fare:     First class (1) tickets are the most expensive.
```

Write a Python program that will read `titanic.csv` into a Pandas dataframe, and then systematically analyze the content from a variety of perspectives. As noted above, the goal is to understand: Who survived, and why? Things to do:

1. Read `titanic.csv` into a Pandas dataframe.

2. Separate the data into two categories: passengers that survived, passengers that drowned. For each category compute the relevant statistics (e.g., how many people, ratio of males and females, number of passengers in each passenger class).

3. Generate histograms for the distribution of age among the survivors and victims.

In Cameron's movie, women and children were given priority to board a lifeboat, and hence survived.

4. Is this part of the story supported by the `titanic.csv` data, or not?

5. Is there any evidence in the data that first class passengers (class 1) were given priority in boarding a lifeboat?

Solution Strategy: In addition to the tasks mentioned above, it seems that we should identify the number of children among the survivors/victims. In the early 1900s a child would be someone younger than say 10 or 12 (not 18). And you'd expect children and their mothers would be given access to the lifeboats, regarless of their gender.

So, a reasonable strategy is: isolate lists for each of these categories and compute appropriate percentages.

Python Source Code:

```
# ================================================================================
```

```
# TestDataProcessingTitanic.py: Read, process and visualize data from data/titanic.csv.
#
# Written by: Mark Austin                                                    February, 2024
# ============================================================================================

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

from pandas import DataFrame
from pandas import read_csv

# ================================================================
# Main function ...
# ================================================================

def main():
    print("--- Enter TestDataProcessingTitanic.main()         ... ");
    print("--- ============================================== ... ");
    print("");

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load titanic data file ... ");
    print("--- ");

    df = pd.read_csv('data/titanic.csv')

    print("--- ");
    print("--- Part 02: Titanic dataframe description ... ");
    print("--- ");

    print( df.describe() )

    print("--- ");
    print("--- Part 03: Iterate over dataframe columns ... ");
    print("--- ");

    for col in df.columns:
       print(col)

    # Filter dataframe to separate survivors and victims ...

    print("--- ");
    print("--- Part 04: Filter dataframe to separate survivors and victims ... ");
    print("--- ");

    dfSurvived = df [ df['Survived'] > 0 ].sort_values ( by=['Age'] )
    print(dfSurvived)

    print( dfSurvived.info() )
    print( dfSurvived.shape )
```

```
dfVictim = df [ df['Survived'] == 0 ].sort_values ( by=['Age'] )
print(dfVictim)

print( dfVictim.info() )
print( dfVictim.shape )

print("--- ");
print("--- ==================================== ... ");
print("--- Part 05: Gather statistics of survivors ... ");
print("--- ==================================== ... ");
print("--- ");

print("--- ");
print("--- Passenger,    Age, Passenger Class, Sex,   Fare ...");
print("--- ============================================================================ ... ");

# Traverse rows of survivors dataframe ...

i = 1
noFemaleSurvivors = 0
noMaleSurvivors   = 0
noChildren05      = 0
noChildren10      = 0
noChildren15      = 0
noPClass01        = 0
noPClass02        = 0
noPClass03        = 0
for index, row in dfSurvived.iterrows():
    age    = row["Age"];
    pclass = row["Pclass"];
    sex    = str( row["Sex"] );
    fare   = row["Fare"];

    # Print details of survivors ...

    print("--- {:9d}:  {:5.1f}, {:15d}, {:10s}, {:5.1f} ... ".format(i, age, pclass, sex.rjust(10

    # Gather statistics ...

    if(sex == "male"):
       noMaleSurvivors = noMaleSurvivors + 1;
    if(sex == "female"):
       noFemaleSurvivors = noFemaleSurvivors + 1;

    if(age <= 5):
       noChildren05 = noChildren05 + 1;
    if(age <= 10):
       noChildren10 = noChildren10 + 1;
    if(age <= 15):
       noChildren15 = noChildren15 + 1;

    # Gather count for no passengers in each pclass ...

    match pclass:
```

```
          case 1:
               noPClass01 = noPClass01 + 1
          case 2:
               noPClass02 = noPClass02 + 1
          case 3:
               noPClass03 = noPClass03 + 1
          case _:
               print("--- pclass not defined ..." );

     i = i + 1;

print("--- ========================================================================== ... ");
print("--- ");
print("--- Summary of Statistics for Survivors:");
print("--- ");
print("--- No male survivors       = {:d} ...".format( noMaleSurvivors ));
print("--- No female survivors     = {:d} ...".format( noFemaleSurvivors ));
print("--- ");
print("--- No children (age <=  5) = {:d} ...".format( noChildren05 ));
print("--- No children (age <= 10) = {:d} ...".format( noChildren10 ));
print("--- No children (age <= 15) = {:d} ...".format( noChildren15 ));
print("--- ");
print("--- No passengers (pclass 1) = {:d} ...".format( noPClass01 ));
print("--- No passengers (pclass 2) = {:d} ...".format( noPClass02 ));
print("--- No passengers (pclass 3) = {:d} ...".format( noPClass03 ));

print("--- ");
print("--- ========================================== ... ");
print("--- Part 06: Gather details of passenger victims ... ");
print("--- ========================================== ... ");
print("--- ");

print("--- ");
print("--- Passenger,    Age, Passenger Class, Sex,   Fare ...");
print("--- ========================================================================== ... ");

# Traverse rows of victims dataframe ...

i = 1
noFemaleVictims = 0
noMaleVictims   = 0
noChildren05    = 0
noChildren10    = 0
noChildren15    = 0
noPClass01      = 0
noPClass02      = 0
noPClass03      = 0
for index, row in dfVictim.iterrows():
    age    = row["Age"];
    pclass = row["Pclass"];
    sex    = str( row["Sex"] );
    fare   = row["Fare"];

    # Print details of victims ...
```

```
    print("--- {:9d}:  {:5.1f}, {:15d}, {:10s}, {:5.1f} ... ".format(i, age, pclass, sex.rjust(10

    # Gather statistics ...

    if(sex == "male"):
       noMaleVictims = noMaleVictims + 1;
    if(sex == "female"):
       noFemaleVictims = noFemaleVictims + 1;

    if(age <= 5):
       noChildren05 = noChildren05 + 1;
    if(age <= 10):
       noChildren10 = noChildren10 + 1;
    if(age <= 15):
       noChildren15 = noChildren15 + 1;

    # Gather count for no passengers in each pclass ...

    match pclass:
         case 1:
              noPClass01 = noPClass01 + 1
         case 2:
              noPClass02 = noPClass02 + 1
         case 3:
              noPClass03 = noPClass03 + 1
         case _:
              print("--- pclass not defined ..." );

    i = i + 1;

print("--- ========================================================================== ... ");
print("--- ");
print("--- Summary of Statistics for Victims:");
print("--- ");
print("--- No male victims         = {:d} ...".format( noMaleVictims ));
print("--- No female victims       = {:d} ...".format( noFemaleVictims ));
print("--- ");
print("--- No children (age <=  5) = {:d} ...".format( noChildren05 ));
print("--- No children (age <= 10) = {:d} ...".format( noChildren10 ));
print("--- No children (age <= 15) = {:d} ...".format( noChildren15 ));
print("--- ");
print("--- No passengers (pclass 1) = {:d} ...".format( noPClass01 ));
print("--- No passengers (pclass 2) = {:d} ...".format( noPClass02 ));
print("--- No passengers (pclass 3) = {:d} ...".format( noPClass03 ));

print("--- ");
print("--- Part 04: Create histogram of age of Titanic Survivors/Victims ... ");
print("--- ");

# Extract array from dataframe ...

survivors = np.array ( dfSurvived['Age'].values )
victims   = np.array ( dfVictim['Age'].values )

# Create histograms ...
```

```
    fig, ((ax0,ax1)) = plt.subplots(nrows=1, ncols=2)

    nbins = 20;
    colors = [ 'red' ]
    ax0.hist( survivors, nbins, density=True, histtype='bar', color=colors, label=colors)
    ax0.set_title('Age of 233 Survivors');
    ax0.set( xlabel="Age", ylabel="Probability Density")
    ax0.set_xlim( [0,80] )
    ax0.set_ylim( [0,0.05] )
    ax0.grid()

    colors = [ 'blue' ]
    ax1.hist( victims, nbins, density=True, histtype='bar', color=colors, label=colors)
    ax1.set_title('Age of 464 Victims');
    ax1.set( xlabel="Age", ylabel="Probability Density")
    ax1.set_xlim( [0,80] )
    ax1.set_ylim( [0,0.05] )
    ax1.grid()

    plt.show()

    print("--- ============================================ ... ");
    print("--- Leave TestDataProcessingAirport.main()         ... ");

# call the main method ...

main()
```

Program Output: The textual output is:

```
--- Enter TestDataProcessingTitanic.main()         ...
--- ============================================ ...

--- Part 01: Load titanic data file ...

--- Part 02: Titanic dataframe description ...

         Survived      Pclass  ...  Parents/Children Aboard       Fare
count  887.000000  887.000000  ...               887.000000  887.00000
mean     0.385569    2.305524  ...                 0.383315   32.30542
std      0.487004    0.836662  ...                 0.807466   49.78204
min      0.000000    1.000000  ...                 0.000000    0.00000
25%      0.000000    2.000000  ...                 0.000000    7.92500
50%      0.000000    3.000000  ...                 0.000000   14.45420
75%      1.000000    3.000000  ...                 0.000000   31.13750
max      1.000000    3.000000  ...                 6.000000  512.32920

[8 rows x 6 columns]

--- Part 03: Iterate over dataframe columns ...

Survived
```

```
Pclass
Name
Sex
Age
Siblings/Spouses Aboard
Parents/Children Aboard
Fare

--- Part 04: Filter dataframe to separate survivors and victims ...

     Survived  Pclass  ... Parents/Children Aboard     Fare
799         1       3  ...                       1   8.5167
751         1       2  ...                       1  14.5000
641         1       3  ...                       1  19.2583
466         1       3  ...                       1  19.2583
77          1       2  ...                       2  29.0000
..        ...     ...  ...                     ...      ...
567         1       2  ...                       0  10.5000
825         1       1  ...                       0  80.0000
273         1       1  ...                       0  77.9583
480         1       3  ...                       0   9.5875
627         1       1  ...                       0  30.0000

[342 rows x 8 columns]
<class 'pandas.core.frame.DataFrame'>
Index: 342 entries, 799 to 627
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Survived                 342 non-null    int64
 1   Pclass                   342 non-null    int64
 2   Name                     342 non-null    object
 3   Sex                      342 non-null    object
 4   Age                      342 non-null    float64
 5   Siblings/Spouses Aboard  342 non-null    int64
 6   Parents/Children Aboard  342 non-null    int64
 7   Fare                     342 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 24.0+ KB
None
(342, 8)
     Survived  Pclass  ... Parents/Children Aboard     Fare
384         0       3  ...                       2  46.9000
163         0       3  ...                       1  39.6875
16          0       3  ...                       1  29.1250
204         0       3  ...                       1  10.4625
820         0       3  ...                       1  39.6875
..        ...     ...  ...                     ...      ...
669         0       2  ...                       0  10.5000
115         0       3  ...                       0   7.7500
490         0       1  ...                       0  49.5042
95          0       1  ...                       0  34.6542
847         0       3  ...                       0   7.7750

[545 rows x 8 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 545 entries, 384 to 847
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Survived                 545 non-null    int64
 1   Pclass                   545 non-null    int64
 2   Name                     545 non-null    object
 3   Sex                      545 non-null    object
 4   Age                      545 non-null    float64
 5   Siblings/Spouses Aboard  545 non-null    int64
 6   Parents/Children Aboard  545 non-null    int64
 7   Fare                     545 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 38.3+ KB
None
(545, 8)

--- ===================================== ...
--- Part 05: Gather statistics of survivors ...
--- ===================================== ...

---
--- Passenger,    Age, Passenger Class, Sex,   Fare ...
--- ============================================================================ ...
---         1:    0.4,               3,       male,   8.5 ...
---         2:    0.7,               2,       male,  14.5 ...
---         3:    0.8,               3,     female,  19.3 ...

... lines of output removed ...

---       340:   63.0,               1,     female,  78.0 ...
---       341:   63.0,               3,     female,   9.6 ...
---       342:   80.0,               1,       male,  30.0 ...
--- ============================================================================ ...
---
--- Summary of Statistics for Survivors:
---
--- No male survivors       = 109 ...
--- No female survivors     = 233 ...
---
--- No children (age <=  5) = 33 ...
--- No children (age <= 10) = 41 ...
--- No children (age <= 15) = 52 ...
---
--- No passengers (pclass 1) = 136 ...
--- No passengers (pclass 2) = 87 ...
--- No passengers (pclass 3) = 119 ...

--- ========================================= ...
--- Part 06: Gather details of passenger victims ...
--- ========================================= ...

--- Passenger,    Age, Passenger Class, Sex,   Fare ...
--- ============================================================================ ...
```

```
---         1:    1.0,               3,       male,  46.9 ...
---         2:    1.0,               3,       male,  39.7 ...
---         3:    2.0,               3,       male,  29.1 ...

... lines of output removed ...

---       543:   71.0,               1,       male,  49.5 ...
---       544:   71.0,               1,       male,  34.7 ...
---       545:   74.0,               3,       male,   7.8 ...
--- ========================================================================== ...

--- Summary of Statistics for Victims:
---
--- No male victims         = 464 ...
--- No female victims       = 81 ...
---
--- No children (age <=  5) = 16 ...
--- No children (age <= 10) = 32 ...
--- No children (age <= 15) = 42 ...
---
--- No passengers (pclass 1) = 80 ...
--- No passengers (pclass 2) = 97 ...
--- No passengers (pclass 3) = 368 ...

--- Part 04: Create histogram of age of Titanic Survivors/Victims ...

--- ============================================ ...
--- Leave TestDataProcessingAirport.main()         ...
```

Interpretation of Statistics. A side-by-side comparison of the statistics for survivors and victims reveals:

```
------------------------------------------------------------
                     Factor         Survivors       Victims
============================================================
                       Males:               109           464
                     Females:               233            81
                     ---------------------------------------
                       Total:               342           545
                     ---------------------------------------

Detailed Analysis:
------------------------------------------------------------

Young children (age <=  5):                33            16
Young children (age <= 10):                41            32
Young children (age <= 15):                52            42

   Passengers (pClass = 1):               136            80
   Passengers (pClass = 2):                87            97
   Passengers (pClass = 3):               119           368
============================================================
```

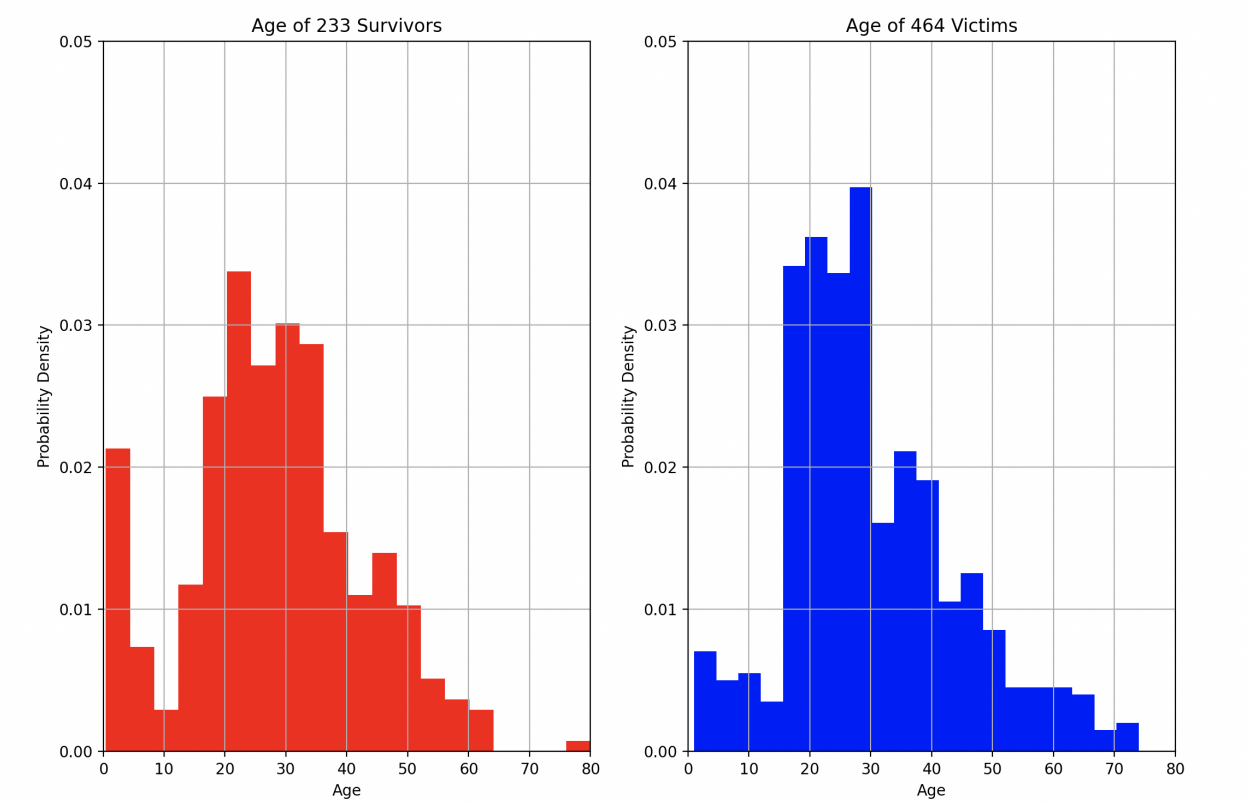

Figure 5: Histogram of probability density vs age for survivors and victims on the Titanic (blue + red areas sum to 1.0).

A few point to note:

1. The passenger list is comprised of 573 males and 314 females, so not a 50-50 split. The data indicates that on a percentage basis, females were much more likely to survive than males.
2. Two-thirds of young children (age ≤ 5) survived.
3. High paying passengers, passenger class 1, were much more likely to survive than those in passenger classes 2 and 3.