Solutions to Homework 2

Question 1: 10 points.

Problem Statement: A laboratory experiment is conducted on 1030 specimens to determine the compressive strength (MPa) of concrete as a function of age (days) and various ingredients (e.g., cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate).

The experimental data (see python-code.d/data/materials/concrete-strength-data.csv) comprises eight (quantitative) input parameters:

```
Parameter             Description
-----------------------------------------------
Cement                kg in a m3 mixture.
Blast Furnace Slag    kg in a m3 mixture.
Fly Ash               kg in a m3 mixture.
Water                 kg in a m3 mixture.
Superplasticizer      kg in a m3 mixture.
Coarse Aggregate      kg in a m3 mixture.
Fine Aggregate        kg in a m3 mixture.
Age                   day (1 -- 365).
-----------------------------------------------
```

and one output:

```
Parameter             Description
-----------------------------------------------------------
Concrete strength     Concrete compressive strength (MPa).
-----------------------------------------------------------
```

What to do? Write a Python program that will:

1. Read the experimental test results from a file `concrete-strength-data.csv` into a Pandas dataframe.
2. Extract numpy arrays from the dataframe for age (days) and concrete compressive strength (MPa).
3. Compute and print the range of parameter values for each input (e.g., Cement content, Blast Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate and Age.

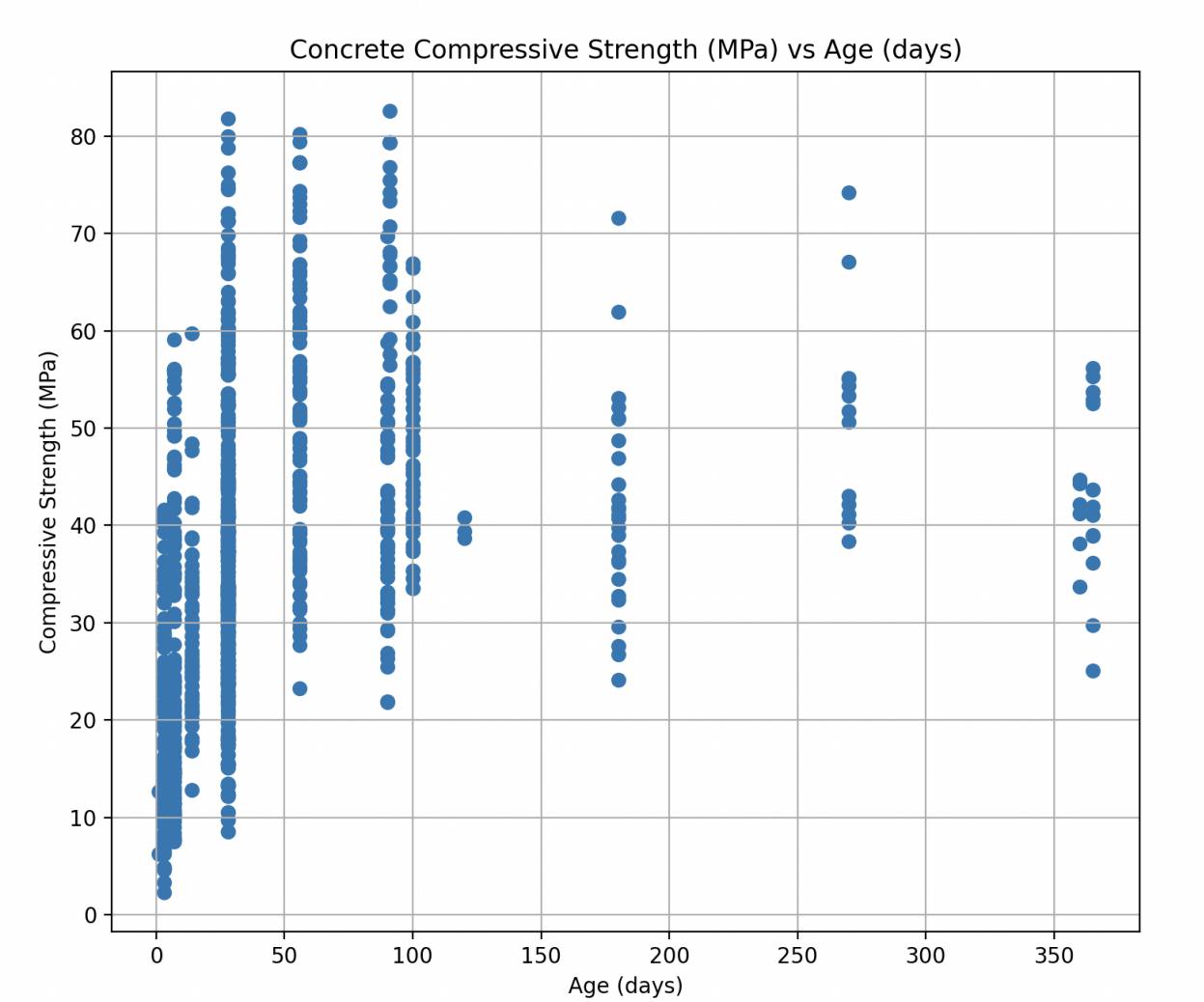

Figure 1: Scatter chart of concrete compressive strength (MPa) vs age (days).

4. Compute and print the maximum, minimum, and average concrete compressive strengths.

5. Create a scatter chart of concrete strength (MPa) vs age (days). See Figure 1.

6. Use the Pandas cut function (i.e., google `pd.cut()`) to organize the strength data into intervals: 0–25, 25–50, 50–75, and 75–100.

7. Generate a histogram of concrete compressive strength (MPa).

8. Generate the cumulative probability distribution for concrete compressive strength, and then create a stair-step graph of "cumulative frequency" versus "compressive strength."

Note 1. The average value of the experimental results can be computed using Python's buildin functions. The "cumulative frequency" versus "compressive strength" is given by

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

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

Python Source Code:

```
# ======================================================================
# TestMaterialsConcreteStrength02.py: Read, process and visualize experimental
# data on compressive strength of concrete.
#
# For details, see: data/material/concrete-strength-readme.txt
#
# Written by: Mark Austin                                    September 2025
# ======================================================================

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 TestMaterialsConcreteStrength02.main()     ... ");
    print("--- ================================================ ... ");
```

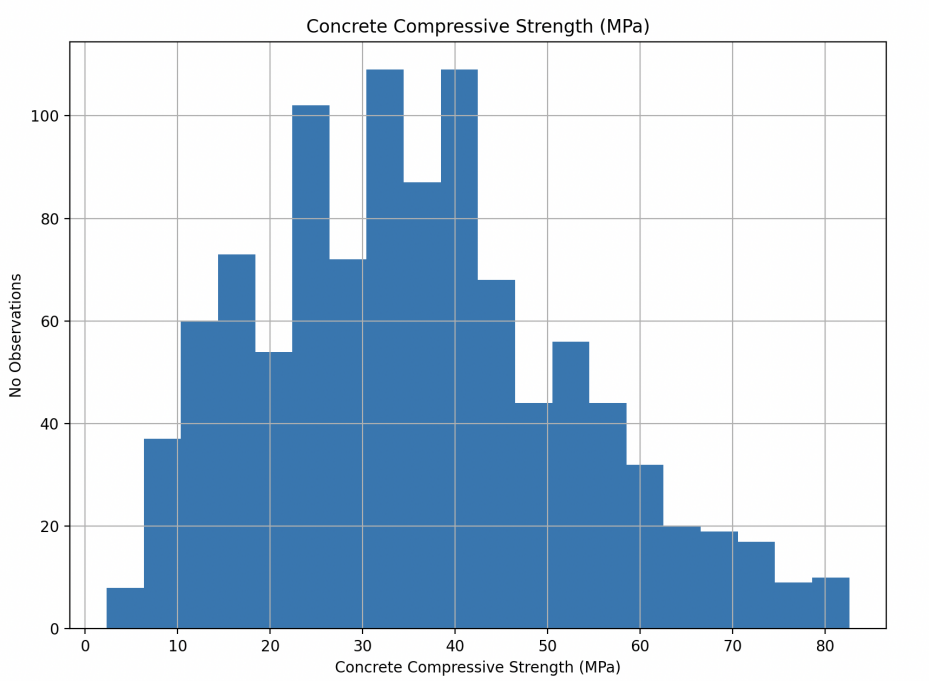

Figure 2: Histogram of concrete compressive strength (MPa).

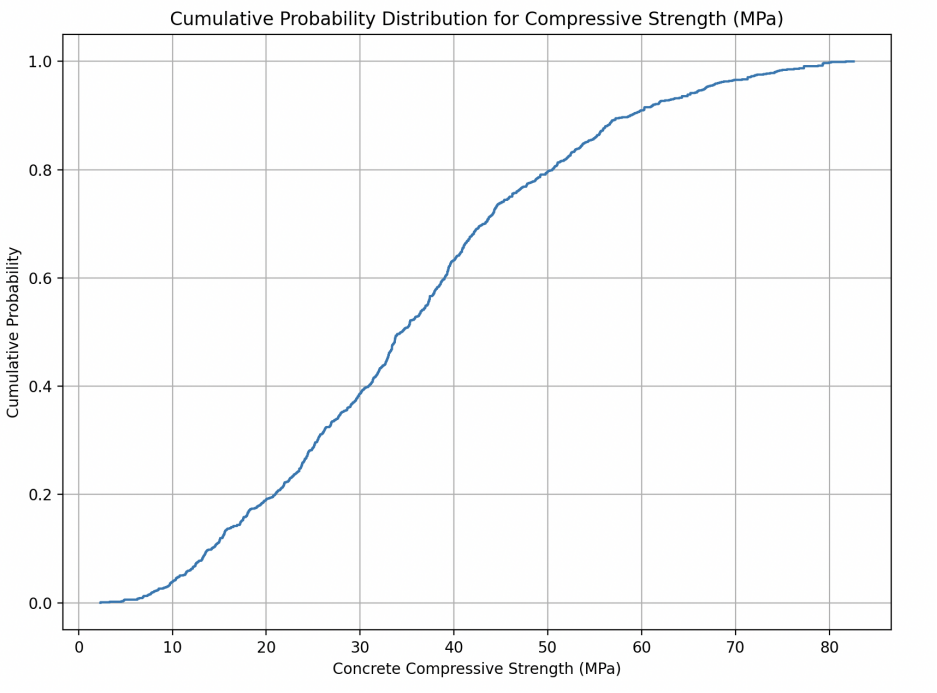

Figure 3: Cumulative distribution of concrete compressive strength (MPa).

```
print("");

# Load and print dataset

print("--- ");
print("--- Part 01: Load data/materials/concrete-strength-data.csv into Pandas ... ");
print("--- ");

df = pd.read_csv('data/materials/concrete-strength-data.csv')
print(df)

# Dataframe info and shape ...

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

print("--- ");
print("--- Part 02: Extract numpy arrays from dataframe columns ... ");
print("--- ");

# Input parameters ...

age      = np.array ( df['Age'].values )

cement   = np.array ( df['Cement'].values )
blastf   = np.array ( df['Blast Furnace Slag'].values )
flyash   = np.array ( df['Fly Ash'].values )
water    = np.array ( df['Water'].values )
superp   = np.array ( df['Superplasticizer'].values )
coarseag = np.array ( df['Coarse Aggregate'].values )
fineag   = np.array ( df['Fine Aggregate'].values )

# Output parameter ...

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

print("--- ");
print("--- Part 03: Range of Input/Output Parameter Values ...");
print("--- ");

print("--- Input: Age: ");
print("---       Min = {:5.2f} days ...".format(min(age)) );
print("---       Max = {:5.2f} days ...".format(max(age)) );

print("--- Input: Cement: ");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(cement)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(cement)) );

print("--- Input: Blast Furnace Slag: ");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(blastf)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(blastf)) );

print("--- Input: Fly Ash:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(flyash)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(flyash)) );
```

```
print("--- Input: Water:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(water)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(water)) );

print("--- Input: Superplasticizer:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(superp)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(superp)) );

print("--- Input: Coarse Aggregate:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(coarseag)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(coarseag)) );

print("--- Input: Fine Aggregate:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(fineag)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(fineag)) );

print("--- Output: Concrete Compressive Strength: ");
print("---       Min = {:5.2f} MPa ...".format(min(strength)) );
print("---       Max = {:5.2f} MPa ...".format(max(strength)) );

print("--- ");
print("--- Part 04: Basic statistics on concrete age/compressive strength ...");
print("--- ");

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

print("--- ");
print("--- Part 05: Transform data arrays into lists ...");
print("--- ");

age.tolist()
strength.tolist()

print("--- ");
print("--- Part 06: Scatter chart of concrete strength (MPa) vs age (days) ...");
print("--- ");

plt.scatter(age, strength )
plt.xlabel('Age (days)');
plt.ylabel('Compressive Strength (MPa)');
plt.title('Concrete Compressive Strength (MPa) vs Age (days)')
plt.grid()
plt.show()

print("--- ");
print("--- Part 07: Organize strength data into intervals: 0-25, 25-50, 50-75, 75-100 ... ");
print("--- ");
```

```
    age_sorted = np.sort( age )
    strength_sorted = np.sort( strength )

    sinterval = [ "0-20", "20-40", "40-60", "60-80", "80-100" ]
    steel_strength_intervals = pd.cut( strength_sorted, [ 0, 20, 40, 60, 80, 100 ], labels = sinterva

    # 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 08: Create histogram of Concrete Compressive Strengths (MPa) ... ");
    print("--- ");

    nbins = 20;
    plt.hist( strength_sorted, nbins );
    plt.title('Concrete Compressive Strength (MPa)');
    plt.xlabel('Concrete Compressive Strength (MPa)');
    plt.ylabel('No Observations');
    plt.grid()
    plt.show()

    print("--- ");
    print("--- Part 09: 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 Compressive Strength (MPa)');
    plt.xlabel('Concrete Compressive Strength (MPa)');
    plt.ylabel('Cumulative Probability');
    plt.grid()
    plt.show()

    print("--- ============================================== ... ");
    print("--- Leave TestMaterialsConcreteStrength02.main()   ... ");

# call the main method ...

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

Program Output: The abbreviated textual output is:

```
--- Enter TestMaterialsConcreteStrength02.main()     ...
--- ================================================ ...
---
--- Part 01: Load data/materials/concrete-strength-data.csv into Pandas ...
---
      Cement  Blast Furnace Slag  ...  Age  Concrete Compressive Strength
0      540.0                 0.0  ...   28                          79.99
1      540.0                 0.0  ...   28                          61.89
2      332.5               142.5  ...  270                          40.27
3      332.5               142.5  ...  365                          41.05
4      198.6               132.4  ...  360                          44.30
...      ...                 ...  ...  ...                            ...
1025   276.4               116.0  ...   28                          44.28
1026   322.2                 0.0  ...   28                          31.18
1027   148.5               139.4  ...   28                          23.70
1028   159.1               186.7  ...   28                          32.77
1029   260.9               100.5  ...   28                          32.40

[1030 rows x 9 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Cement                         1030 non-null   float64
 1   Blast Furnace Slag             1030 non-null   float64
 2   Fly Ash                        1030 non-null   float64
 3   Water                          1030 non-null   float64
 4   Superplasticizer               1030 non-null   float64
 5   Coarse Aggregate               1030 non-null   float64
 6   Fine Aggregate                 1030 non-null   float64
 7   Age                            1030 non-null   int64
 8   Concrete Compressive Strength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
None
(1030, 9)
---
--- Part 02: Extract numpy arrays from dataframe columns ...
---
--- Part 03: Range of Input/Output Parameter Values ...
---
--- Input: Age:
---       Min =  1.00 days ...
---       Max = 365.00 days ...
--- Input: Cement:
---       Min = 102.00 kg in m3 mixture ...
---       Max = 540.00 kg in m3 mixture ...
--- Input: Blast Furnace Slag:
---       Min =  0.00 kg in m3 mixture ...
---       Max = 359.40 kg in m3 mixture ...
--- Input: Fly Ash:
---       Min =  0.00 kg in m3 mixture ...
```

```
---       Max = 200.10 kg in m3 mixture ...
--- Input: Water:
---       Min = 121.80 kg in m3 mixture ...
---       Max = 247.00 kg in m3 mixture ...
--- Input: Superplasticizer:
---       Min =  0.00 kg in m3 mixture ...
---       Max = 32.20 kg in m3 mixture ...
--- Input: Coarse Aggregate:
---       Min = 801.00 kg in m3 mixture ...
---       Max = 1145.00 kg in m3 mixture ...
--- Input: Fine Aggregate:
---       Min = 594.00 kg in m3 mixture ...
---       Max = 992.60 kg in m3 mixture ...
--- Output: Concrete Compressive Strength:
---       Min =  2.33 MPa ...
---       Max = 82.60 MPa ...
---
--- Part 04: Basic statistics on concrete age/compressive strength ...
---
--- Min age     =  1.00 days ...
--- Max age     = 365.00 days ...
--- Average age = 45.66 days ...
---
--- Min strength     =  2.33 MPa ...
--- Max strength     = 82.60 MPa ...
--- Average strength = 35.82 MPa ...
---
--- Part 05: Transform data arrays into lists ...
---
--- Part 06: Scatter chart of concrete strength (MPa) vs age (days) ...
---
--- Part 07: Organize strength data into intervals: 0-25, 25-50, 50-75, 75-100 ...
---
2.33 0 0-20
3.32 0 0-20
4.57 0 0-20
4.78 0 0-20

.... lines of output removed ...

79.3 3 60-80
79.4 3 60-80
79.99 3 60-80
80.2 4 80-100
81.75 4 80-100
82.6 4 80-100
---
--- Part 08: Create histogram of Concrete Compressive Strengths (MPa) ...
--- Part 09: Generate cumulative frequency data and graph ...
--- No data points = 1030 ...
--- ============================================== ...
--- Leave TestMaterialsConcreteStrength02.main()   ...
```

Clearly, by itself age (days) is **not a good predictor** of concrete compressive strength (MPa).

Question 2: 20 points.

Problem Statement: The purpose of this question is to take a first step in understanding the spatial and temporal nature of air traffic from BWI to international destinations, 1990 through 2020. Solutions to this problem are complicated by the constantly evolving nature of resources at BWI – for our purposes, it is important to note that Southwest Airlines did not have a presence at BWI until the mid 1990s, and Concourse A/B only opened in 2005 (source: Google).

International Air Traffic at BWI. Compared to volumes of domestic traffic to/from Baltimore-Washington International (BWI) Airport (175,000 flights in 2015 alone; 70% of domestic flights are Southwest), the number of international flights is small. Figure 6 shows, for example, a draft visualization of only 6,935 international flights to/from BWI, 1990 through 2020.

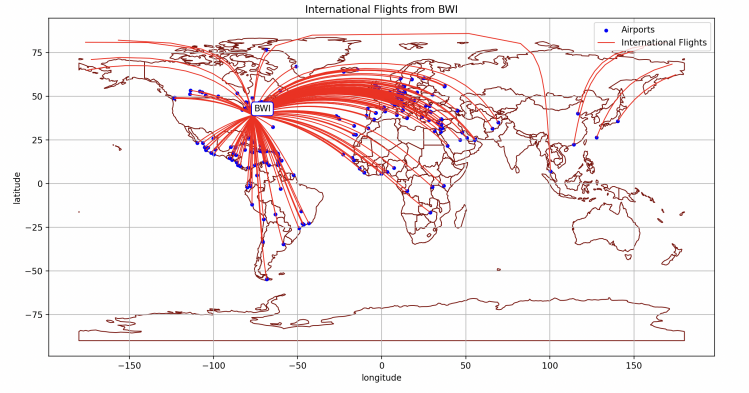

Figure 4: Draft visualization for international flights from BWI, 1990 through 2020.

A preliminary analysis of the flight data (see python-code.d/data/transportation/air/flights-international-bwi1990-2020.csv) indicates that planes departed BWI to 246 destinations, the most popular routes being:

- BWI to CUN (Cancún International Airport, Mexico) $\longrightarrow$ 803 flights.
- BWI to YYZ (Lester Pearson International Airport, Toronto) $\longrightarrow$ 527 flights.
- BWI to NAS (Lynden Pindling International Airport, Nassau, Bahamas) $\longrightarrow$ 509 flights.
- BWI to MBJ (Sangster International Airport, Jamaica) $\longrightarrow$ 484 flights.
- BWI to PUJ (Punta Cana International Airport, Dominican Republic) $\longrightarrow$ 388 flights.
- BWI to KEF (Keflavik International Airport, Iceland) $\longrightarrow$ 355 flights.
- BWI to LHR (London Heathrow Airport, England) $\longrightarrow$ 266 flights.
- BWI to AUA (Queen Beatrix International Airport, Aruba) $\longrightarrow$ 265 flights.

At the other end of the spectrum, approximately half (135) of the flight destinations report a flight count of three or less. For example, during the past thirty years (1990-2020) there is only one recorded flight from BWI to YYJ (Victoria International Airport, Victoria, Canada).

Four Sources of Data: To understand the nature of international flights from BWI, we will gather data from four sources:

Data Source 1: The data file data/transportation/air/airline-iata-codes-small.csv contains iata codes and countries of origin for the main airlines using BWI, IAD and DCA. The columns of data are:

```
   Code -- IATA code for airline.
Airline -- Name of ailine.
Country -- Ownership of the airline ...
```

For example, the first entry is: AA, American Airlines, United States.

Data Source 2: The data file data/airports/airports-worldwide-large.csv is a listing of 57,422 airports worldwide. For our purposes, the columns of interest are:

```
Col  1   ident         --
Col  2   type          -- Type of airport (e.g., large_airport, heliport, closed).
Col  3   name          -- Name of airport.
Col  4   elevation_ft  -- Airport elevation (ft).
Col  5   continent     --
Col  6   iso_country   -- Code for country of airport (e.g., CA for Canada).
Col  7   iso_region    -- Maryland, British Columbia, etc ...
Col  8   municipality  --
Col  9   gps_code      --
Col 10   iata_code     -- International Air Transportation Code (e.g., BWI, YYZ).
Col 11   local_code    --
Col 12   coordinates   -- Airport longitude and latitude.
```

Data Source 3: The data file data/transportation/air/flights-international-bwi1990-2020.csv contains details of 6,935 flight departures from BWI to international destinations. The data is organized into 16 columns, namely:

```
Col  1   data_dte    --
Col  2   Year        --
Col  3   Month       --
Col  4   usg_apt_id  --
Col  5   usg_apt     --
Col  6   usg_wac     --
Col  7   fg_apt_id   --
```

```
Col  8   fg_apt      --
Col  9   fg_wac      --
Col 10   airlineid   --
Col 11   carrier     --
Col 12   carriergroup --
Col 13   type        --
Col 14   Scheduled   --
Col 15   Charter     --
Col 16   Total       --
```

Data Source 4: The shape data file data/geography/world/ne_110m_admin_0_countries.shp provides details on a world map.

What to do?

Write a Python program that will:

1. Load the contents of airline-iata-codes-small.csv into a Pandas dataframe. Create and print a dictionary of airline code, i.e.,

   ```
   Dictionary: Airlines:
   ---------------------------------------------------
   key: AA --> value: [' American Airlines', ' United States'] ...
   key: AC --> value: [' Air Canada', ' Canada'] ...
   key: AF --> value: [' Air France', ' France'] ...

   ... etc, etc, etc ...
   ---------------------------------------------------
   ```

   For a hint, see the pandas method `to_dict(..)`.

2. Load the contents of airports-worldwide-large.csv into a Pandas dataframe. Remove all entries that do not have a valid `iata` code. Create and print a dictionary of airports that can be accessed by their iata code. A sample of text output might look like:

   ```
   key: CUN --> value: ['Cancún International Airport', nan, 'MX', '-86.87709, 21.03650'] ...
   ```

3. Load the contents of flights-international-bwi1990-2020.csv into a Pandas dataframe. Create and print a dictionary of airport destinations (i.e., the key) along with a count of the number of flights (i.e., the value). Sort this dictionary by flight count, then print details of the flight destination such as the city, country, and coordinates. For example, textual output for the BWI (Baltimore) to CUN (Cancun) flight might look like:

```
--- Flight: BWI to CUN     --> count: 803 ...
--- City: Cancún International Airport ...
--- Region: nan ...
--- Country: MX ...
--- Coordinates: (long, lat) = (-86.87709,  21.03650) ...
```

4. Replicate Figure 6, and then customize the visualization to highlight the most frequently traveled international routes from BWI (listed at the top of the question).

5. Create a bar chart or histogram of flights from BWI to CUN (Cancun, MX), 1990 to 2020.

Python Source Code: Parts 1 through 4.

```
# ============================================================================
# TestAirTransportationBWI02.py. Analysis and visualization of flights from BWI
# to International destinations.

# --- Only plot flight trajectories where destination count > 260.
#
# Written by: Mark Austin                                          October 2025
# ============================================================================

from geographiclib.geodesic import Geodesic
from geojson import MultiLineString

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

import numpy as np
import pandas as pd
import geopandas

import matplotlib.pyplot as plt

import shapely
from shapely.geometry import Point, LineString, Polygon
from shapely import wkt

from collections import OrderedDict

# ============================================================
# Print dictionary ...
# ============================================================

def printDictionary( title, dictionary, sort = False ):

    print("")
    print("Dictionary: {:s}:".format(title))
    print("-------------------------------------------------- ")
```

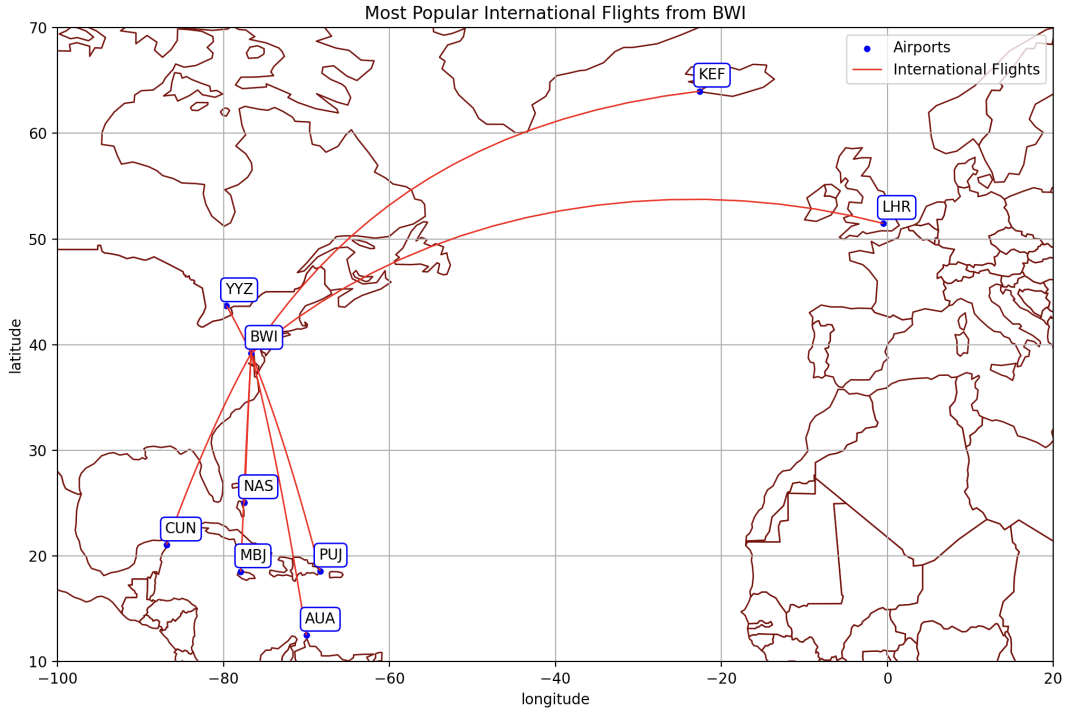

Figure 5: Most popular flights from BWI to international destinations, 1990 through 2020.

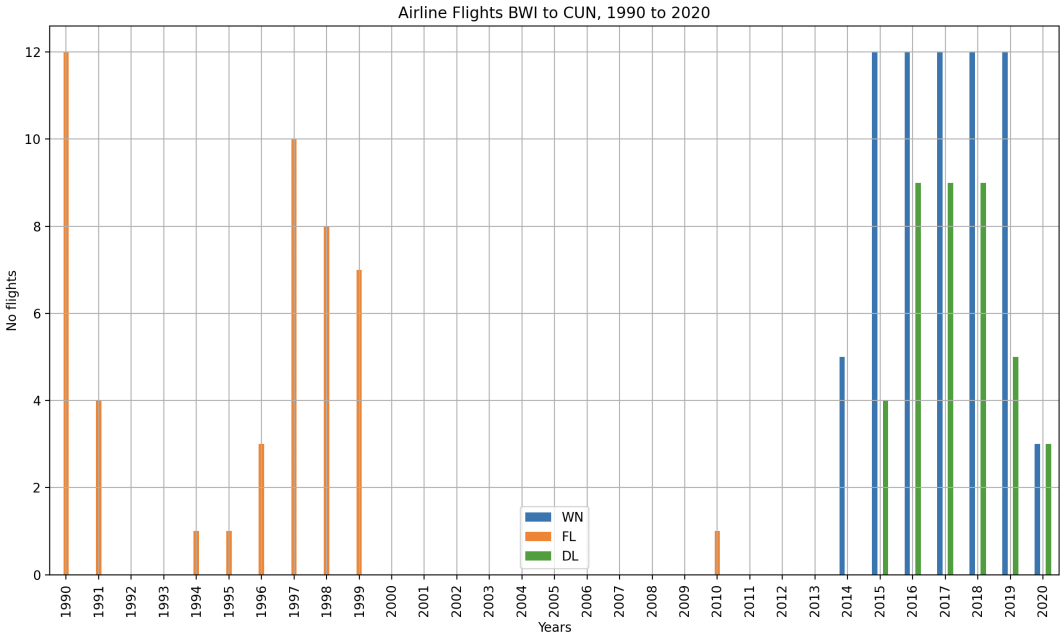

Figure 6: Barchart of flights from BWI to CUN, 1990 through 2020.

```
    if sort == True:
       sortedDictionary = OrderedDict( sorted( dictionary.items() ) )
       for key, value in sortedDictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))
    else:
       for key, value in dictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))

    print("-------------------------------------------------- ")

# ============================================================
# Geodesic calculation ...
# ============================================================

def geodesic(lat1, lon1, lat2, lon2, steps):
    inverse = Geodesic.WGS84.Inverse(lat1, lon1, lat2, lon2)
    linestrings = []
    coordinates = []

    for i in range(0, steps + 1):
        direct = Geodesic.WGS84.Direct(inverse['lat1'], inverse['lon1'], inverse['azi1'],
                                       (i / float(steps)) * inverse['s12'])

        if len(coordinates) > 0:
            expression01 = (coordinates[-1][0] < -90 and direct['lon2'] >  90);
            expression02 = (coordinates[-1][0] >  90 and direct['lon2'] < -90);
            if (expression01 or expression02):
                linestrings.append(coordinates)
                coordinates = []
        coordinates.append((direct['lon2'], direct['lat2']))

    linestrings.append(coordinates)
    geojson = MultiLineString(linestrings)
    return geojson

# ==============================
# main method ...
# ==============================

def main():
    print("--- Enter TestAirTransportationBWI02.main()   ... ");
    print("--- ======================================== ... ");

    print("");
    print("--- Part 01: Load ../air/airlines-iata-codes-small.csv ... ");
    print("--- ================================================== ... ");
    print("");

    # Read data/transportation/air/airline-iata-codes-small.csv ...

    datapath01 = "data/transportation/air/airline-iata-codes-small.csv"
    dfAirlines = pd.read_csv( datapath01 )
    airlinecompact01    = dfAirlines.set_index('Code').T
    airlinedictionary01 = airlinecompact01.to_dict('list')
```

```
# Sort and print dictionary ...

printDictionary( "Airlines", airlinedictionary01, sort = True );

print("");
print("--- Part 02: Load data/airports/airports-worldwide-large.csv data file ... ");
print("--- ================================================================ ... ");
print("");

datapath02 = "data/airports/airports-worldwide-large.csv";
dfAirports = pd.read_csv( datapath02 )

# Remove rows that do not have a valid iata code ...

dfAirports = dfAirports [ ( dfAirports['iata_code'].notnull() ) ]

# Extract essential columns ...

items = [ 'name', 'continent', 'iso_country', 'iata_code', 'coordinates' ]
dfAirportscompact01 = dfAirports.filter( items )

dfAirportscompact01 = dfAirportscompact01.set_index('iata_code').T
airportdictionary01 = dfAirportscompact01.to_dict('list')

# Sort and print dictionary ...

printDictionary( "Airports", airportdictionary01, sort = True );

print("")
print("--- Part 03: Load flights-international-bwi1990-2020.csv ... ");
print("--- ================================================== ... ");
print("")

datapath03 = "data/transportation/air/flights-international-bwi1990-2020.csv"
dfFlights  = pd.read_csv(datapath03)

# Create compact representation of columns ...

items = [ 'Year', 'Month', 'fg_apt', 'carrier' ]
flightscompact01 = dfFlights.filter( items )

# Print heading ....
# print( flightscompact01.head() );

print( flightscompact01 );

print("");
print("--- Part 04: Create dictionary of flight destinations + counts ... ");
print("--- ============================================================ ... ");
print("");

# Dictionary of flight destinations ...

flightDestination = {}
for i in flightscompact01['fg_apt'].tolist():
```

```
   if i in flightDestination:
      flightDestination[i] += 1
   else:
      flightDestination[i] = 1

# Sort items, then print ...

print ("")
print ("--- International Flight Destinations ...")

ij = 0;
for city in sorted(flightDestination, key = flightDestination.get):
    destinationcount = flightDestination[ city ];
    ij = ij + 1;
    print("--- ")
    print("--- Destination {:3d}: BWI to {:7s} --> count: {:2d} ...".format( ij, city, destinatio
    dest = airportdictionary01.get( city )
    if dest is not None:
       city01      = dest[0];
       region01    = dest[1];
       country01   = dest[2];
       coords01    = dest[3];
       longitude01 = str( coords01.split(",")[0] )
       latitude01  = str( coords01.split(",")[1] )
       print("--- City: {:s} ...".format( str( city01 ) ) )
       print("--- Region: {:s} ...".format( str( dest[1] ) ) )
       print("--- Country: {:s} ...".format( str( country01 ) ) )
       print("--- Coordinates: (long, lat) = ({:s}, {:s}) ...".format( longitude01, latitude01 )

print("");
print("--- Part 05: Draw USA boundary map ... ");
print("--- ============================== ... ");
print("");

# Read world boundary shp file into geopandas ...

filepathWorld = "data/geography/world/ne_110m_admin_0_countries.shp";
worldboundary = geopandas.read_file( filepathWorld )
gdf01         = geopandas.GeoDataFrame(worldboundary)

print("");
print("--- Part 06: International Flights from BWI ... ");
print("--- ===================================== ... ");
print("");

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

print("--- plot geodataframe ...");

ax = gdf01.plot( color='white', lw = 1, edgecolor='maroon')
ax.set_aspect('equal')
ax.set_title("Most Popular International Flights from BWI")

# Add BWI to list of airports ...
```

```
print("--- add bwi to list of airports ...");

bwi = airportdictionary01[ "BWI" ]
coords01  = bwi[3];
longitude = str( coords01.split(",")[0] )
latitude  = str( coords01.split(",")[1] )

bwiairport01 = Point( longitude, latitude )

airports = []
airports.append( bwiairport01 )

print("");
print("--- Part 07: Create list of destination airports ... ");
print("--- ========================================== ... ");
print("");

for city in sorted(flightDestination, key = flightDestination.get):
    destinationcount = flightDestination[ city ];
    dest             = airportdictionary01.get( city )
    destinationcount = flightDestination[ city ];
    if dest is not None and destinationcount > 260:

       print("--- ")
       print("--- Flight: BWI to {:7s}: flight count = {:d} ...".format( city, destinationcount )

       coords01    = dest[3];
       longitude01 = str( coords01.split(",")[0] )
       latitude01  = str( coords01.split(",")[1] )

       # Create point for airport, then add to list of airports  ...

       airport01 = Point( longitude01, latitude01 )
       airports.append( airport01 )

print("");
print("--- Part 08: Draw airports and geodesic flight paths on map ... ");
print("--- ========================================================= ... ");
print("");

# Draw airports ...

gdf02 = geopandas.GeoDataFrame( geometry = airports )
gdf02.set_crs("EPSG:4326", inplace=True)

gdf02.plot(ax=ax, color = 'blue', markersize = 12, label = 'Airports' )

# Draw geodesic flight paths ...

source = airportdictionary01.get("BWI")   # <-- Baltimore ...

coords01    = source[3];
sourcelongitude01 = str( coords01.split(",")[0] )
sourcelatitude01  = str( coords01.split(",")[1] )
```

```
flightpaths = []
for key, value in flightDestination.items():
    destination      = airportdictionary01.get( key )   # <-- Destination ...
    destinationcount = flightDestination[ key ]
    if destinationcount > 260 and destination is not None:
       coords01    = destination[3];
       destlongitude01 = str( coords01.split(",")[0] )
       destlatitude01  = str( coords01.split(",")[1] )

       # Geodesic calculation for source --> destination flight ...

       linestrings02 = []
       i = 1;
       nosegments = 20;
       for linestring in geodesic( float( sourcelatitude01) , float( sourcelongitude01),
                                   float( destlatitude01), float( destlongitude01 ), nosegments ):
           linestrings02.append(linestring)
           i = i + 1;

       # Textual description of flightpath segements ...

       line02 = MultiLineString(linestrings02)
       flightpaths.append( line02 )
    else:
       print("---   airport key: {:s} not found ...". format(key) );

# Draw flightpaths on map ...

gdf03 = geopandas.GeoDataFrame( geometry = flightpaths )
gdf03.set_crs("EPSG:4326", inplace=True)

gdf03.plot(ax=ax, lw =  1, edgecolor = 'red', label = 'International Flights' )

# Annotation for BWI Airport ...

print("");
print("--- Part 09: Add airport annotations + plot labels ...");
print("--- ============================================== ... ");
print("");

# Annotation for BWI ...

iata_code = "BWI";
airport   = airportdictionary01[ iata_code ]
coords01  = source[3];
bwilongitude01 = str( coords01.split(",")[0] )
bwilatitude01  = str( coords01.split(",")[1] )

dlong = -0.7; dlat = 0.7;
ax.annotate( iata_code, xy=( float (bwilongitude01) + dlong, float(bwilatitude01) + dlat ),
       xytext=(3, 3), textcoords="offset points",
       bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="blue", lw=1))

# Annotations for destination airports
```

```
    for key, value in flightDestination.items():
        iata_code = key;
        print("--- iata_code --> {:s} ...".format( iata_code ))
        destination      = airportdictionary01.get( iata_code )  # <-- Destination ...
        destinationcount = flightDestination[ key ]
        if destinationcount > 260 and destination is not None:
           coords01    = destination[3];
           destlongitude01 = str( coords01.split(",")[0] )
           destlatitude01  = str( coords01.split(",")[1] )

           dlong = -0.7; dlat = 0.7;
           ax.annotate( iata_code, xy=( float (destlongitude01) + dlong, float(destlatitude01) + dlat
                        xytext=(3, 3), textcoords="offset points",
                        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="blue", lw=1))

    # Add labels to main plot, then show display ...

    plt.legend(loc='best')
    plt.xlabel('longitude')
    plt.ylabel('latitude')

    # Window for continental US ...

    plt.xlim( [-100,  20 ] )
    plt.ylim( [  10,  70 ] )

    plt.grid(True)
    plt.show()

    print("--- ");
    print("--- ======================================== ... ");
    print("--- Finished TestAirTransportationBWI02.main() ... ");

# call the main method ...

main()
```

Abbreviated Program Output: Parts 1 through 4.

```
--- Part 01: Load ../air/airlines-iata-codes-small.csv ...
--- ================================================ ...

Dictionary: Airlines:
-------------------------------------------------
key: AA --> value: [' American Airlines', ' United States'] ...
key: AC --> value: [' Air Canada', ' Canada'] ...
key: AF --> value: [' Air France', ' France'] ...
key: AI --> value: [' Air India', ' India'] ...

... lines of output removed ...

key: VS --> value: [' Virgin Atlantic Airways', ' United Kingdom'] ...
key: VX --> value: [' Virgin America', ' United States'] ...
```

```
key: WN --> value: [' Southwest Airlines', ' United States'] ...
key: WS --> value: [' Westjet', ' Canada'] ...
-------------------------------------------------

--- Part 02: Load data/airports/airports-worldwide-large.csv data file ...
--- ================================================================== ...

Dictionary: Airports:
-------------------------------------------------
key: - --> value: ['Vejrø', 'EU', 'DK', '11.375, 55.035'] ...
key: 0 --> value: ['Corixá Airport', 'SA', 'BR', '-58.31719970703125, -16.38360023498535'] ...
key: AAA --> value: ['Anaa Airport', 'OC', 'PF', '-145.50999450683594, -17.35260009765625'] ...

... lines of output removed ...

key: BWI --> value: ['Baltimore/Washington International Thurgood Marshall Airport', nan, 'US', '-76.

... lines of output removed ...

key: CUN --> value: ['Cancún International Airport', nan, 'MX', '-86.8770980835, 21.036500930800003']

... lines of output removed ...

key: ZZO --> value: ['Zonalnoye Airport', 'AS', 'RU', '142.761001587, 50.6692008972'] ...
key: ZZU --> value: ['Mzuzu Airport', 'AF', 'MW', '34.01179885864258, -11.444700241088867'] ...
key: ZZV --> value: ['Zanesville Municipal Airport', nan, 'US', '-81.89209747310001, 39.9444007874']
-------------------------------------------------

--- Part 03: Load flights-international-bwi1990-2020.csv ...
--- ================================================== ...

      Year  Month fg_apt carrier
0     2007     10    LEJ      WO
1     2009     10    KEF      GL
2     2005      7    MSE      WO
3     2002      7    YHM      CO
4     2003      8    PUJ      U5
...    ...    ...    ...     ...
6930  2010      8    YYZ      QK
6931  2007      7    YYZ      QK
6932  2002     12    YYZ      QK
6933  2003      7    YYZ      QK
6934  2002     11    YYZ      QK

[6935 rows x 4 columns]

--- Part 04: Create dictionary of flight destinations + counts ...
--- ========================================================== ...

--- International Flight Destinations ...
---
--- Destination   1: BWI to YYJ     --> count:  1 ...
--- City: Victoria International Airport ...
--- Region: nan ...
--- Country: CA ...
```

```
--- Coordinates: (long, lat) = (-123.426002502,  48.646900177) ...

... lines of output removed ...

---
--- Destination 238: BWI to FPO     --> count: 259 ...
--- City: Grand Bahama International Airport ...
--- Region: nan ...
--- Country: BS ...
--- Coordinates: (long, lat) = (-78.695602417,  26.5587005615) ...
---
--- Destination 239: BWI to AUA     --> count: 265 ...
--- City: Queen Beatrix International Airport ...
--- Region: nan ...
--- Country: AW ...
--- Coordinates: (long, lat) = (-70.015198,  12.5014) ...
---
--- Destination 240: BWI to LHR     --> count: 266 ...
--- City: London Heathrow Airport ...
--- Region: EU ...
--- Country: GB ...
--- Coordinates: (long, lat) = (-0.461941,  51.4706) ...
---
--- Destination 241: BWI to KEF     --> count: 355 ...
--- City: Keflavik International Airport ...
--- Region: EU ...
--- Country: IS ...
--- Coordinates: (long, lat) = (-22.6056,  63.985001) ...
---
--- Destination 242: BWI to PUJ     --> count: 388 ...
--- City: Punta Cana International Airport ...
--- Region: nan ...
--- Country: DO ...
--- Coordinates: (long, lat) = (-68.36340332030001,  18.567399978599997) ...
---
--- Destination 243: BWI to MBJ     --> count: 484 ...
--- City: Sangster International Airport ...
--- Region: nan ...
--- Country: JM ...
--- Coordinates: (long, lat) = (-77.91339874267578,  18.503700256347656) ...
---
--- Destination 244: BWI to NAS     --> count: 509 ...
--- City: Lynden Pindling International Airport ...
--- Region: nan ...
--- Country: BS ...
--- Coordinates: (long, lat) = (-77.466202,  25.039) ...
---
--- Destination 245: BWI to YYZ     --> count: 527 ...
--- City: Lester B. Pearson International Airport ...
--- Region: nan ...
--- Country: CA ...
--- Coordinates: (long, lat) = (-79.63059997559999,  43.6772003174) ...
---
--- Destination 246: BWI to CUN     --> count: 803 ...
--- City: Cancún International Airport ...
```

```
--- Region: nan ...
--- Country: MX ...
--- Coordinates: (long, lat) = (-86.8770980835,  21.036500930800003) ...

--- Part 05: Draw USA boundary map ...
--- ============================== ...

--- Part 06: International Flights from BWI ...
--- ===================================== ...

--- plot geodataframe ...
--- add bwi to list of airports ...

--- Part 07: Create list of destination airports ...
--- ========================================== ...

---
--- Flight: BWI to AUA    : flight count = 265 ...
--- Flight: BWI to LHR    : flight count = 266 ...
--- Flight: BWI to KEF    : flight count = 355 ...
--- Flight: BWI to PUJ    : flight count = 388 ...
--- Flight: BWI to MBJ    : flight count = 484 ...
--- Flight: BWI to NAS    : flight count = 509 ...
--- Flight: BWI to YYZ    : flight count = 527 ...
--- Flight: BWI to CUN    : flight count = 803 ...

--- Part 08: Draw airports and geodesic flight paths on map ...
--- ======================================================== ...

--- Part 09: Add airport annotations + plot labels ...
--- ============================================ ...

--- ======================================== ...
--- Finished TestAirTransportationBWI02.main() ...
```

Abbreviated Python Source Code: Part 5.

```
# ================================================================================
# TestAirTransportationBWI03.py. International flights from BWI to Cancun, Mexico.
#
# Written by: Mark Austin                                              September 2025
# ================================================================================

from geographiclib.geodesic import Geodesic
from geojson import MultiLineString

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

import numpy as np
import pandas as pd
```

```
import geopandas

import matplotlib.pyplot as plt

import shapely
from shapely.geometry import Point, LineString, Polygon
from shapely import wkt

from collections import OrderedDict

# ============================================================
# Print dictionary ...
# ============================================================

def printDictionary( title, dictionary, sort = False ):

    print("")
    print("Dictionary: {:s}:".format(title))
    print("-------------------------------------------------- ")

    if sort == True:
       sortedDictionary = OrderedDict( sorted( dictionary.items() ) )
       for key, value in sortedDictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))
    else:
       for key, value in dictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))

    print("-------------------------------------------------- ")

# main method ...

def main():
    print("--- Enter TestAirTransportationBWI03.main()   ... ");
    print("--- ======================================== ... ");

    print("");
    print("--- Part 01: Load ../air/airlines-iata-codes-small.csv ... ");
    print("--- ================================================ ... ");
    print("");

    # Read data/transportation/air/airline-iata-codes-small.csv ...

    datapath01 = "data/transportation/air/airline-iata-codes-small.csv"
    dfAirlines = pd.read_csv( datapath01 )
    airlinecompact01    = dfAirlines.set_index('Code').T
    airlinedictionary01 = airlinecompact01.to_dict('list')

    # Sort and print dictionary ...

    printDictionary( "Airlines", airlinedictionary01, sort = True );

    print("");
    print("--- Part 02: Load data/airports/airports-worldwide-large.csv data file ... ");
    print("--- ============================================================== ... ");
```

```
print("");

datapath02 = "data/airports/airports-worldwide-large.csv";
dfAirports = pd.read_csv( datapath02 )

# Remove rows that do not have a valid iata code ...

dfAirports = dfAirports [ ( dfAirports['iata_code'].notnull() ) ]

# Extract essential columns ...

items = [ 'name', 'continent', 'iso_country', 'iata_code', 'coordinates' ]
dfAirportscompact01 = dfAirports.filter( items )

dfAirportscompact01 = dfAirportscompact01.set_index('iata_code').T
airportdictionary01 = dfAirportscompact01.to_dict('list')

# Sort and print dictionary ...

printDictionary( "Airports", airportdictionary01, sort = True );

print("")
print("--- Part 03: Load flights-international-bwi1990-2020.csv ... ");
print("--- ================================================== ... ");
print("")

datapath03 = "data/transportation/air/flights-international-bwi1990-2020.csv"
dfFlights  = pd.read_csv(datapath03)

print("--- Dataset shape: {:s} ...".format( str(dfFlights.shape) ))
print("--- Column names: {:s} ...".format( str(dfFlights.columns.tolist()) ))

print("")
print("--- Dataset info and description:")
print("")

print( dfFlights.info() )
print( dfFlights.describe() )

print("")
print("--- Part 04: Create compact representation of columns ... ");
print("--- ================================================== ... ");
print("")

# Create compact representation of columns ...

items = [ 'Year', 'Month', 'fg_apt', 'carrier' ]
dfCompact01 = dfFlights.filter( items )

# Print heading ....

print( dfCompact01.head() );

print("")
print("--- Compact dataset info:")
```

```
print("")

print( dfCompact01.info() );

print("");
print("--- Part 05: Filter data to focus on flights to Cancun (CUN) ... ");
print("--- ======================================================== ... ");
print("");

dfCancun01 = dfCompact01[ dfCompact01['fg_apt'] == 'CUN' ]
print( dfCancun01 )

dfCancun01Sorted = dfCancun01.sort_values(by='Year')
print("--- DataFrame sorted by 'Year' in ascending order:")
print( dfCancun01Sorted )

print("");
print("--- WN: Southwest flights sorted by year ...")
print("");

dfCancun02Sorted = dfCancun01Sorted[ dfCancun01Sorted['carrier'] == 'WN' ]
dfCancun02Sorted = dfCancun02Sorted.groupby(['Year'])['Year'].count()

print( dfCancun02Sorted )

print("");
print("--- FL: Breeze Airways flights sorted by year ...")
print("");

dfCancun03Sorted = dfCancun01Sorted[ dfCancun01Sorted['carrier'] == 'MX' ]
dfCancun03Sorted = dfCancun03Sorted.groupby(['Year'])['Year'].count()

print( dfCancun03Sorted )

print("");
print("--- DL: Delta Airlines flights sorted by year ...")
print("");

dfCancun04Sorted = dfCancun01Sorted[ dfCancun01Sorted['carrier'] == 'DL' ]
dfCancun04Sorted = dfCancun04Sorted.groupby(['Year'])['Year'].count()

print( dfCancun04Sorted )

print("");
print("--- Part 06: Count no flights to Cancun by each airline ... ");
print("--- ================================================== ... ");
print("");

dfgroup01 = dfCancun01.groupby(['carrier'])['carrier'].count()

print( dfgroup01.info() );

print("Flights to Cancun: ")

for index, value in dfgroup01.items():
```

```
    print("--- Airline: {:4s} ---> No flights: {:3d}".format( index, value) )

print("");
print("--- Part 07: Aggregate flights into a single dataframe ... ");
print("--- ================================================ ... ");
print("");

# Manually generate lists for dataframe ...

years     = [ 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999,
              2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,
              2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019,
              2020 ];

flflights = [   12,    4,    0,    0,    1,    1,    3,   10,    8,    7,
                 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 0 ];

dlflights = [    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 0,    0,    0,    0,    0,    4,    9,    9,    9,    5,
                 3 ];

wnflights = [    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                 0,    0,    0,    0,    5,   12,   12,   12,   12,   12,
                 3 ];

# Create dictionary from lists ...

dict01 = {'Years': years,
          'FL': flflights,
          'DL': dlflights,
          'WN': wnflights }

flights02 = pd.DataFrame(dict01);
print( flights02 );

print("");
print("--- Part 08: Generate bar chart    ... ");
print("--- ============================== ... ");
print("");

flights02.plot(x="Years", y= ["WN", "FL", "DL"], kind = "bar" );

plt.ylabel("No flights");
plt.title("Airline Flights BWI to CUN, 1990 to 2020");
plt.grid(True);
plt.show()

print("--- ");
print("--- ======================================== ... ");
print("--- Finished TestAirTransportationBWI03.main() ... ");
```

```
# call the main method ...

main()
```

Abbreviated Program Output: Part 5.

```
--- Enter TestAirTransportationBWI03.main()   ...
--- ========================================= ...

--- Part 01: Load ../air/airlines-iata-codes-small.csv ...
--- ================================================== ...

Dictionary: Airlines:
-------------------------------------------------

... lines of output removed ...

-------------------------------------------------

--- Part 02: Load data/airports/airports-worldwide-large.csv data file ...
--- ================================================================== ...

Dictionary: Airports:
-------------------------------------------------

... lines of output removed ...

-------------------------------------------------

--- Part 03: Load flights-international-bwi1990-2020.csv ...
--- ================================================== ...

--- Dataset shape: (6935, 16) ...
--- Dataset info and description:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6935 entries, 0 to 6934
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   data_dte      6935 non-null   object
 1   Year          6935 non-null   int64
 2   Month         6935 non-null   int64
 3   usg_apt_id    6935 non-null   int64
 4   usg_apt       6935 non-null   object
 5   usg_wac       6935 non-null   int64
 6   fg_apt_id     6935 non-null   int64
 7   fg_apt        6935 non-null   object
 8   fg_wac        6935 non-null   int64
 9   airlineid     6935 non-null   int64
 10  carrier       6880 non-null   object
 11  carriergroup  6935 non-null   int64
 12  type          6935 non-null   object
```

```
 13  Scheduled     6935 non-null   int64
 14  Charter       6935 non-null   int64
 15  Total         6935 non-null   int64
dtypes: int64(11), object(5)
memory usage: 867.0+ KB
None
              Year        Month  ...      Charter        Total
count  6935.000000  6935.000000  ...  6935.000000  6935.000000
mean   2004.605912     6.331363  ...     2.362221    29.507426
std       8.717963     3.353307  ...     6.284410    45.196241
min    1990.000000     1.000000  ...     0.000000     1.000000
25%    1997.000000     3.000000  ...     0.000000     1.000000
50%    2004.000000     6.000000  ...     0.000000     8.000000
75%    2012.000000     9.000000  ...     1.000000    53.000000
max    2020.000000    12.000000  ...    98.000000   326.000000

[8 rows x 11 columns]

--- Part 04: Create compact representation of columns ...
--- ================================================== ...

   Year  Month fg_apt carrier
0  2007     10    LEJ      WO
1  2009     10    KEF      GL
2  2005      7    MSE      WO
3  2002      7    YHM      CO
4  2003      8    PUJ      U5

--- Compact dataset info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6935 entries, 0 to 6934
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Year     6935 non-null   int64
 1   Month    6935 non-null   int64
 2   fg_apt   6935 non-null   object
 3   carrier  6880 non-null   object
dtypes: int64(2), object(2)
memory usage: 216.8+ KB
None

--- Part 05: Filter data to focus on flights to Cancun (CUN) ...
--- ======================================================== ...

--- DataFrame sorted by 'Year' in ascending order:
      Year  Month fg_apt carrier
5836  1990      1    CUN      MX
4862  1990     11    CUN      MX
4867  1990      7    CUN      MX
1917  1990      2    CUN     ACQ
4868  1990      8    CUN      MX
...    ...    ...    ...     ...
4180  2020      2    CUN      DL
```

```
6497  2020      3    CUN      WN
6683  2020      2    CUN      WN
6679  2020      1    CUN      WN
3931  2020      1    CUN      DL

[803 rows x 4 columns]

--- WN: Southwest flights sorted by year ...

Year
2014     5
2015    12
2016    12
2017    12
2018    12
2019    12
2020     3
Name: Year, dtype: int64

--- FL: Breeze Airways flights sorted by year ...

Year
1990    12
1991     4
1994     1
1995     1
1996     3
1997    10
1998     8
1999     7
2010     1
Name: Year, dtype: int64

--- DL: Delta Airlines flights sorted by year ...

Year
2015    4
2016    9
2017    9
2018    9
2019    5
2020    3
Name: Year, dtype: int64

--- Part 06: Count no flights to Cancun by each airline ...
--- ================================================== ...

Flights to Cancun:

--- Airline: 1I   ---> No flights:  10
--- Airline: 23Q  ---> No flights:   1
--- Airline: 2KQ  ---> No flights:   1
--- Airline: 5J   ---> No flights:  13
--- Airline: 5Y   ---> No flights:   1
--- Airline: AA   ---> No flights:   3
```

```
--- Airline: ACQ  ---> No flights:  23
--- Airline: AM   ---> No flights:  13
--- Airline: AS   ---> No flights:  28
--- Airline: B6   ---> No flights:   4
--- Airline: BFQ  ---> No flights:   1
--- Airline: CDQ  ---> No flights:   1
--- Airline: CO   ---> No flights:  16
--- Airline: CPQ  ---> No flights:   8
--- Airline: DL   ---> No flights:  39
--- Airline: FCQ  ---> No flights:   4
--- Airline: FF   ---> No flights:   2
--- Airline: FI   ---> No flights:   1
--- Airline: FL   ---> No flights:  66
--- Airline: FQ   ---> No flights:   2
--- Airline: G4   ---> No flights:   1
--- Airline: GD   ---> No flights:  34
--- Airline: GL   ---> No flights:  13
--- Airline: HCQ  ---> No flights:   3
--- Airline: JKQ  ---> No flights:   3
--- Airline: KEQ  ---> No flights:   5
--- Airline: KP   ---> No flights:   1
--- Airline: KW   ---> No flights:  16
--- Airline: LGQ  ---> No flights:  31
--- Airline: LUQ  ---> No flights:   1
--- Airline: MG   ---> No flights:  13
--- Airline: MMQ  ---> No flights:  21
--- Airline: MX   ---> No flights:  47
--- Airline: NK   ---> No flights:  29
--- Airline: OI   ---> No flights:   1
--- Airline: PA   ---> No flights:   2
--- Airline: PCQ  ---> No flights:  29
--- Airline: PJQ  ---> No flights:   6
--- Airline: PLQ  ---> No flights:   6
--- Airline: PNQ  ---> No flights:   1
--- Airline: QTQ  ---> No flights:   1
--- Airline: RD   ---> No flights:   9
--- Airline: RYQ  ---> No flights:  17
--- Airline: SM   ---> No flights:   5
--- Airline: SPQ  ---> No flights:  11
--- Airline: SX   ---> No flights:   1
--- Airline: SY   ---> No flights:   1
--- Airline: SYQ  ---> No flights:   1
--- Airline: T9   ---> No flights:  13
--- Airline: TEQ  ---> No flights:   9
--- Airline: TW   ---> No flights:   2
--- Airline: TZ   ---> No flights:  10
--- Airline: U5   ---> No flights:  84
--- Airline: UA   ---> No flights:  11
--- Airline: US   ---> No flights:  25
--- Airline: USQ  ---> No flights:   6
--- Airline: VB   ---> No flights:   7
--- Airline: VIQ  ---> No flights:   4
--- Airline: WA   ---> No flights:   1
--- Airline: WN   ---> No flights:  68
--- Airline: WO   ---> No flights:   2
```

```
--- Airline: X9   ---> No flights:   2
--- Airline: XG   ---> No flights:   2
--- Airline: XP   ---> No flights:   8
--- Airline: ZN   ---> No flights:   3

--- Part 07: Aggregate flights into a single dataframe ...
--- ================================================ ...
```

	Years	FL	DL	WN
0	1990	12	0	0
1	1991	4	0	0
2	1992	0	0	0
3	1993	0	0	0
4	1994	1	0	0
5	1995	1	0	0
6	1996	3	0	0
7	1997	10	0	0
8	1998	8	0	0
9	1999	7	0	0
10	2000	0	0	0
11	2001	0	0	0
12	2002	0	0	0
13	2003	0	0	0
14	2004	0	0	0
15	2005	0	0	0
16	2006	0	0	0
17	2007	0	0	0
18	2008	0	0	0
19	2009	0	0	0
20	2010	1	0	0
21	2011	0	0	0
22	2012	0	0	0
23	2013	0	0	0
24	2014	0	0	5
25	2015	0	4	12
26	2016	0	9	12
27	2017	0	9	12
28	2018	0	9	12
29	2019	0	5	12
30	2020	0	3	3

```
--- Part 08: Generate bar chart    ...
--- ============================== ...

--- ======================================== ...
--- Finished TestAirTransportationBWI03.main() ...
```

Question 3: 20 points.

Problem Statement: The purpose of this question is to give you **practice at using Pandas** to work with – i.e., filter, transform, group, query, – a datafile that reports on 5.8 million flights across in US during 2015. There are many questions one might ask: How many airlines and planes operate within the domestic US? How many planes does each airline have? Which airlines have the most flights per year? What are the busiest airports? With respect to individual flights, what are the shortest and longest flights? How many flights (on average) does a plane make during a year? What proportion of the day is a plane actually flying? How far does a plane travel during a year?

Data Source: The data file python-code.d/data/transportation/air/flights-usa-large2015.csv contains data on 5.8 million flights across in US during 2015. The data is organized into 31 columns, namely:

```
Cols 1-3   YEAR, MONTH, DAY    -- Date of the flight.
Col    4   DAY_OF_WEEK         -- Monday (1), Tuesday (2), etc ...
Col    5   AIRLINE             -- For example, UA = United Airlines ...
Col    6   FLIGHT_NUMBER       --
Col    7   TAIL_NUMBER         --

... lines of output removed ...

Col   26   CANCELLATION_REASON --
Col   27   AIR_SYSTEM_DELAY    --
Col   28   SECURITY_DELAY      --
Col   29   AIRLINE_DELAY       --
Col   30   LATE_AIRCRAFT_DELAY --
Col   31   WEATHER_DELAY       --
```

Notice that this file makes reference to the origin and destination airports and the airline providing the flight, so analysis of its contents will require the support of airline-iata-codes-small.csv and airports-worldwide-large.csv (covered in Question 2).

What to do? Write a Python program that will:

1. Load the contents of airports-worldwide-large.csv into a Pandas dataframe. Remove all entries that do not have a valid `iata` code. Create and print a dictionary of airports. For a hint, see the pandas method `to_dict(..)`.

2. Load the contents of python-code.d/data/transportation/air/flights-usa-large2015.csv into a Pandas dataframe.

   Preliminary analysis indicates that while the overall quality of this datafile is very high, it's not perfect. Identify and print the number of missing values for each of the 31 columns. What areas of the datafile have the lowest overall quality?

3. Compute and print the total number of flights for each of the participating airlines (i.e., AA, WN, etc).

4. The flights-usa-large2015.csv contains flights from 930 airports, but by design, numbers of airline flights are concentrated at hubs (this is the well known hub-and-spoke model).

Identify and print the number of flights leaving the five busiest airports in the US. You should find that the five busiest airports account for more than 20% of all flight departures in the US.

5. Create a dictionary that contains a list of the planes operated by each airline. Compute and print a summary of each airline and the size of its aircraft fleet.

6. For all flights in the US, compute: (1) the total air time (it's more than 600 million minutes), (2) the minimum and maximum flight durations, (3) the total distance flown (think billions of miles), and (4) the average flight time.

7. At this point you should see that the way United Airlines uses its planes is completely different to Southwest Airlines.

Choose a representative plane from each airline and compute and print: (1) the total number of flights during 2015, (2) the minimum, maximum and average flight times, (3) the total air time, and (4) the total distance traveled during 2015.

8. Briefly summarize your conclusions in a few bullet points + text.

Python Source Code:

```
# ============================================================================
# TestAirplaneFlights01.py. Analyse 5.8 million flights within the US, 2015.
#
# Written by: Mark Austin                                        September 2025
# ============================================================================

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

import numpy as np
import pandas as pd
import geopandas

import matplotlib.pyplot as plt

import shapely
from shapely.geometry import Point, LineString, Polygon
from shapely import wkt

from collections import OrderedDict

# ============================================================
# Print dictionary ...
# ============================================================
```

```
def printDictionary( title, dictionary, sort = False ):
    print("Dictionary: {:s}:".format(title))
    print("------------------------------------------------- ")

    if sort == True:
       sortedDictionary = OrderedDict( sorted( dictionary.items() ) )
       for key, value in sortedDictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))
    else:
       for key, value in dictionary.items():
          print("key: {:s} --> value: {:s} ...".format( key, str(value) ))

    print("------------------------------------------------- ")

# main method ...

def main():
    print("--- Enter TestAriplaneFlights01.main()        ... ");
    print("--- ======================================== ... ");

    print("");
    print("--- Part 01: Load data/airports/airports-usa.csv data file ... ");
    print("--- ======================================================== ... ");
    print("");

    dfFlights = pd.read_csv('data/transportation/air/flights-usa-large2015.csv')

    print("--- Dataset shape: {:s} ...".format( str(dfFlights.shape) ))
    print("--- Column names: {:s} ...".format( str(dfFlights.columns.tolist()) ))

    print("")
    print("--- Dataset info:")
    print("")

    print( dfFlights.info() )

    print("")
    print("--- Dataset description:")
    print("")

    print( dfFlights.describe() )

    print("");
    print("--- Part 02: Preliminary Data Analysis ... ");
    print("--- ======================================================== ... ");
    print("");

    print("")
    print("--- Sum and print missing values per column: ")
    print("")

    print( dfFlights.isnull().sum(), "\n")

    print("");
```

```
print("--- Count number of unique values ... ");
print("");

no_airlines = dfFlights['AIRLINE'].nunique()

print("--- No airlines = {:d} ...".format(no_airlines) );

print("");
print("--- Frequency of airlines in flight data ... ");
print("");

no_value_counts = dfFlights['AIRLINE'].value_counts( normalize=True )
print( no_value_counts )

no_origin_airports = dfFlights['ORIGIN_AIRPORT'].nunique()

print("--- No origin airports = {:d} ...".format(no_origin_airports) );

print("");
print("--- Count of origin airports ... ");
print("");

no_value_counts = dfFlights['ORIGIN_AIRPORT'].value_counts()
print( no_value_counts )

print("");
print("--- Count of destination airports ... ");
print("");

no_value_counts = dfFlights['DESTINATION_AIRPORT'].value_counts()

print( no_value_counts )

no_value_counts = dfFlights['DESTINATION_AIRPORT'].value_counts( normalize=True )

print("");
print( no_value_counts )

print("");
print("--- Distance metrics ... ");
print("");

distance_metrics = dfFlights['DISTANCE']
print( distance_metrics )

print("");
print("--- Summary of airline flights ... ");
print("");

airline_counts = dfFlights.groupby('AIRLINE').size()
print(airline_counts)

print("");
print("--- Part 03: Create compact representation of flights ... ");
print("--- ================================================ ... ");
```

```
print("");

items = [ 'AIRLINE', 'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'AIR_TIME', 'DESTINATION_AIRPORT', 'DISTANC
df01 = dfFlights.filter( items )

print(df01)

# Basic statistics ...

print("");
print("--- Part 04: Total No flights for each airline ... ");
print("--- ======================================== ... ");

print( df01['AIRLINE'].value_counts() );

print("");
print("--- Part 05: ComOrganize data by airline then plane tail number ... ");
print("--- ========================================================= ... ");
print("");

# Organized data by airline then plane tail number ....

dfgroup01 = df01.groupby(['AIRLINE'])['TAIL_NUMBER'].apply(list)
print(dfgroup01)

# For each airline, print total number of plane visits

dfgroup01 = df01.groupby("AIRLINE").agg(list)

print("");
print("AIRLINE        TOTAL NO FLIGHTS");
print("================================");

for index, row in dfgroup01.iterrows():
    print(" {:s} --> {:10d} flights ...".format( index, len( row['TAIL_NUMBER'] ) ) )

# For each airline, aggregate planes into a list ...

df02 = df01.groupby("AIRLINE").agg(list)

# Loop over rows in dataframe ...

print("");
print("AIRLINE               NO PLANES");
print("================================");

for index, row in df02.iterrows():
    print(" {:s} --> {:6d} ...".format( index, len( set( row['TAIL_NUMBER'] )) ))

print("");
print("--- Part 06; Flight statistics ... ");
print("--- ========================= ... ");
print("");

total_airtime   = dfFlights['AIR_TIME'].sum()
```

```
maximum_airtime = dfFlights['AIR_TIME'].max()
minimum_airtime = dfFlights['AIR_TIME'].min()
average_airtime = dfFlights['AIR_TIME'].mean()

print("--- Total air time        = {:.2f} minutes ...".format( total_airtime ) );
print("--- Minimum flight time   = {:.2f} minutes ...".format( minimum_airtime ) );
print("--- Maximum flight time   = {:.2f} minutes ...".format( maximum_airtime ) );
print("--- Average flight time   = {:.2f} minutes ...".format( average_airtime ) );

total_distance = df01['DISTANCE'].sum()

print("--- Total flight distance = {:,d} miles ...".format( total_distance ) );

# Filter data to only keep flights associated with tail number: N3KUAA

print("");
print("--- Part 07: United Airlines Planes: [ N38459, N845UA, N451UA, N479UA, N440UA ] ... ");
print("--- ================================================================================ ... ");

uaplanes = [ 'N38459', 'N845UA', 'N451UA', 'N479UA', 'N440UA' ];
for tailnumber in uaplanes:
    print("---");
    print("--- Tail number: {:s} ...".format( tailnumber ));

    dfplane01 = df01 [ ( df01['TAIL_NUMBER'] == tailnumber ) ]

    # Minimum, maximum, average flight times, total airtime  ...

    total_airtime   = dfplane01['AIR_TIME'].sum()
    maximum_airtime = dfplane01['AIR_TIME'].max()
    minimum_airtime = dfplane01['AIR_TIME'].min()
    average_airtime = dfplane01['AIR_TIME'].mean()
    no_flights      = len( dfplane01['AIR_TIME'] )

    print("---   Total no flights      = {:,.2f} ...".format( no_flights ) );
    print("---   No flights/day        = {:,.2f} ...".format( no_flights/365 ) );
    print("---   Total air time        = {:,.2f} minutes ...".format( total_airtime ) );
    print("---   Minimum flight time   = {:,.2f} minutes ...".format( minimum_airtime ) );
    print("---   Maximum flight time   = {:,.2f} minutes ...".format( maximum_airtime ) );
    print("---   Average flight time   = {:,.2f} minutes ...".format( average_airtime ) );

    total_distance = dfplane01['DISTANCE'].sum()
    print("---   Total flight distance = {:,d} miles ...".format( total_distance ) );

print("");
print("--- Part 08: Southwest Planes: [N486WN, N293WN, N8641B, N8303R, N7707C ] ... ");
print("--- ================================================================== ... ");

wnplanes = [ 'N486WN', 'N293WN', 'N8641B', 'N8303R', 'N7707C' ]
for tailnumber in wnplanes:
    print("---");
    print("--- Tail number: {:s} ...".format( tailnumber ));

    dfplane01 = df01 [ ( df01['TAIL_NUMBER'] == tailnumber ) ]
```

```
        # Minimum, maximum, average flight times, total airtime  ...

        total_airtime   = dfplane01['AIR_TIME'].sum()
        maximum_airtime = dfplane01['AIR_TIME'].max()
        minimum_airtime = dfplane01['AIR_TIME'].min()
        average_airtime = dfplane01['AIR_TIME'].mean()
        no_flights      = len( dfplane01['AIR_TIME'] )

        print("---   Total no flights      = {:,.2f} ...".format( no_flights ) );
        print("---   No flights/day        = {:,.2f} ...".format( no_flights/365 ) );
        print("---   Total air time        = {:,.2f} minutes ...".format( total_airtime ) );
        print("---   Minimum flight time   = {:,.2f} minutes ...".format( minimum_airtime ) );
        print("---   Maximum flight time   = {:,.2f} minutes ...".format( maximum_airtime ) );
        print("---   Average flight time   = {:,.2f} minutes ...".format( average_airtime ) );

        total_distance = dfplane01['DISTANCE'].sum()
        print("---   Total flight distance = {:,d} miles ...".format( total_distance ) );

    print("--- ");
    print("--- ======================================== ... ");
    print("--- Leave TestAirplaneFlights01.main()       ... ");

# call the main method ...

main()
```

Abbreviated Program Output:

```
--- Part 01: Load data/airports/airports-usa.csv data file ...
--- ================================================== ...

--- Dataset info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5819079 entries, 0 to 5819078
Data columns (total 31 columns):
 #   Column               Dtype
---  ------               -----
 0   YEAR                 int64
 1   MONTH                int64
 2   DAY                  int64
 3   DAY_OF_WEEK          int64
 4   AIRLINE              object
 5   FLIGHT_NUMBER        int64
 6   TAIL_NUMBER          object
 7   ORIGIN_AIRPORT       object
 8   DESTINATION_AIRPORT  object
 9   SCHEDULED_DEPARTURE  int64
 10  DEPARTURE_TIME       float64
 11  DEPARTURE_DELAY      float64
 12  TAXI_OUT             float64
 13  WHEELS_OFF           float64
 14  SCHEDULED_TIME       float64
```

```
 15  ELAPSED_TIME         float64
 16  AIR_TIME             float64
 17  DISTANCE             int64
 18  WHEELS_ON            float64
 19  TAXI_IN              float64
 20  SCHEDULED_ARRIVAL    int64
 21  ARRIVAL_TIME         float64
 22  ARRIVAL_DELAY        float64
 23  DIVERTED             int64
 24  CANCELLED            int64
 25  CANCELLATION_REASON  object
 26  AIR_SYSTEM_DELAY     float64
 27  SECURITY_DELAY       float64
 28  AIRLINE_DELAY        float64
 29  LATE_AIRCRAFT_DELAY  float64
 30  WEATHER_DELAY        float64
dtypes: float64(16), int64(10), object(5)
memory usage: 1.3+ GB
None

--- Dataset description:

            YEAR         MONTH  ...  LATE_AIRCRAFT_DELAY  WEATHER_DELAY
count  5819079.0  5.819079e+06  ...         1.063439e+06   1.063439e+06
mean      2015.0  6.524085e+00  ...         2.347284e+01   2.915290e+00
std          0.0  3.405137e+00  ...         4.319702e+01   2.043334e+01
min       2015.0  1.000000e+00  ...         0.000000e+00   0.000000e+00
25%       2015.0  4.000000e+00  ...         0.000000e+00   0.000000e+00
50%       2015.0  7.000000e+00  ...         3.000000e+00   0.000000e+00
75%       2015.0  9.000000e+00  ...         2.900000e+01   0.000000e+00
max       2015.0  1.200000e+01  ...         1.331000e+03   1.211000e+03

[8 rows x 26 columns]

--- Part 02: Preliminary Data Analysis ...
--- ================================================== ...

--- Sum and print missing values per column:

YEAR                       0
MONTH                      0
DAY                        0
DAY_OF_WEEK                0
AIRLINE                    0
FLIGHT_NUMBER              0
TAIL_NUMBER            14721
ORIGIN_AIRPORT             0
DESTINATION_AIRPORT        0
SCHEDULED_DEPARTURE        0
DEPARTURE_TIME         86153
DEPARTURE_DELAY        86153
TAXI_OUT               89047
WHEELS_OFF             89047
SCHEDULED_TIME             6
ELAPSED_TIME          105071
```

```
AIR_TIME                105071
DISTANCE                     0
WHEELS_ON                92513
TAXI_IN                  92513
SCHEDULED_ARRIVAL            0
ARRIVAL_TIME             92513
ARRIVAL_DELAY           105071
DIVERTED                     0
CANCELLED                    0
CANCELLATION_REASON    5729195
AIR_SYSTEM_DELAY       4755640
SECURITY_DELAY         4755640
AIRLINE_DELAY          4755640
LATE_AIRCRAFT_DELAY    4755640
WEATHER_DELAY          4755640
dtype: int64

--- Count number of unique values ...

--- No airlines = 14 ...
--- Frequency of airlines in flight data ...

AIRLINE
WN    0.216848
DL    0.150519
AA    0.124759
OO    0.101108
EV    0.098293
UA    0.088626
MQ    0.050632
B6    0.045892
US    0.034149
AS    0.029647
NK    0.020171
F9    0.015610
HA    0.013107
VX    0.010638
Name: proportion, dtype: float64

--- No origin airports = 930 ...

--- Count of origin and destination airports ...

ORIGIN_AIRPORT
ATL      346836
ORD      285884
DFW      239551
DEN      196055
LAX      194673
          ...
11503         1
14222         1
13964         1
13459         1
12265         1
```

```
DESTINATION_AIRPORT
ATL      346904
ORD      285906
DFW      239582
DEN      196010
LAX      194696
          ...
13964         1
14025         1
15497         1
12265         1
10666         1

DESTINATION_AIRPORT
ATL      5.961493e-02
ORD      4.913252e-02
DFW      4.117181e-02
DEN      3.368402e-02
LAX      3.345822e-02
             ...
13964    1.718485e-07
14025    1.718485e-07
15497    1.718485e-07
12265    1.718485e-07
10666    1.718485e-07

--- Summary of airline flights ...

AIRLINE
AA     725984
AS     172521
B6     267048
DL     875881
EV     571977
F9      90836
HA      76272
MQ     294632
NK     117379
OO     588353
UA     515723
US     198715
VX      61903
WN    1261855

--- Part 03: Create compact representation of flights ...
--- ================================================ ...

        AIRLINE TAIL_NUMBER  ... DESTINATION_AIRPORT  DISTANCE
0            AS      N407AS  ...                 SEA      1448
1            AA      N3KUAA  ...                 PBI      2330
2            US      N171US  ...                 CLT      2296
3            AA      N3HYAA  ...                 MIA      2342
4            AS      N527AS  ...                 ANC      1448
...         ...         ...  ...                 ...       ...
```

```
5819074      B6      N657JB  ...                 BOS      2611
5819075      B6      N828JB  ...                 PSE      1617
5819076      B6      N913JB  ...                 SJU      1598
5819077      B6      N527JB  ...                 SJU      1189
5819078      B6      N534JB  ...                 BQN      1576

[5819079 rows x 6 columns]

--- Part 04: Total No flights for each airline ...
--- ======================================== ...

AIRLINE
WN    1261855
DL     875881
AA     725984
OO     588353
EV     571977
UA     515723
MQ     294632
B6     267048
US     198715
AS     172521
NK     117379
F9      90836
HA      76272
VX      61903

--- Part 05: ComOrganize data by airline then plane tail number ...
--- ======================================================== ...

AIRLINE
AA    [N3KUAA, N3HYAA, N3LAAA, N853AA, N3GXAA, N3CTA...
AS    [N407AS, N527AS, N309AS, N413AS, N457AS, N431A...
B6    [N607JB, N597JB, N653JB, N239JB, N627JB, N307J...
DL    [N3730B, N826DN, N958DN, N547US, N3751B, N651D...
EV    [N12142, N26545, N882AS, N720EV, N832AS, N1296...
F9    [N918FR, N905FR, N218FR, N933FR, nan, nan, N21...
HA    [N389HA, N492HA, N476HA, N484HA, N477HA, N485H...
MQ    [N660MQ, N932MQ, N925MQ, N5PBMQ, N514MQ, N504M...
NK    [N635NK, N525NK, N632NK, N528NK, N629NK, N514N...
OO    [N701BR, N8982A, N445SW, N746SK, N824AS, N432S...
UA    [N78448, N76517, N76519, N37293, N30401, N7952...
US    [N171US, N584UW, N571UW, N804AW, N172US, N197U...
VX    [N361VA, N852VA, N845VA, N842VA, N621VA, N525V...
WN    [N742SW, N486WN, N293WN, N249WN, N630WN, N910W...
Name: TAIL_NUMBER, dtype: object

AIRLINE        TOTAL NO FLIGHTS
==============================
 AA -->     725984 flights ...
 AS -->     172521 flights ...
 B6 -->     267048 flights ...
 DL -->     875881 flights ...
 EV -->     571977 flights ...
 F9 -->      90836 flights ...
```

```
 HA -->      76272 flights ...
 MQ -->     294632 flights ...
 NK -->     117379 flights ...
 OO -->     588353 flights ...
 UA -->     515723 flights ...
 US -->     198715 flights ...
 VX -->      61903 flights ...
 WN -->    1261855 flights ...

AIRLINE               NO PLANES
==============================
 AA -->   1045 ...
 AS -->    147 ...
 B6 -->    215 ...
 DL -->    829 ...
 EV -->    391 ...
 F9 -->     63 ...
 HA -->     51 ...
 MQ -->    204 ...
 NK -->     79 ...
 OO -->    384 ...
 UA -->    722 ...
 US -->    352 ...
 VX -->     57 ...
 WN -->    705 ...

--- Part 06; Flight statistics ...
--- ========================== ...

--- Total air time        = 648606351.00 minutes ...
--- Minimum flight time   = 7.00 minutes ...
--- Maximum flight time   = 690.00 minutes ...
--- Average flight time   = 113.51 minutes ...
--- Total flight distance = 4,785,357,409 miles ...

--- Part 07: United Airlines Planes: [ N38459, N845UA, N451UA, N479UA, N440UA ] ...
--- ============================================================================ ...
---
--- Tail number: N38459 ...
---   Total no flights      = 943.00 ...
---   No flights/day        = 2.58 ...
---   Total air time        = 167,989.00 minutes ...
---   Minimum flight time   = 28.00 minutes ...
---   Maximum flight time   = 395.00 minutes ...
---   Average flight time   = 178.33 minutes ...
---   Total flight distance = 1,305,776 miles ...
---
--- Tail number: N845UA ...
---   Total no flights      = 1,223.00 ...
---   No flights/day        = 3.35 ...
---   Total air time        = 144,213.00 minutes ...
---   Minimum flight time   = 24.00 minutes ...
---   Maximum flight time   = 374.00 minutes ...
---   Average flight time   = 118.01 minutes ...
---   Total flight distance = 1,046,779 miles ...
```

```
---
--- Tail number: N451UA ...
---   Total no flights      = 1,116.00 ...
---   No flights/day        = 3.06 ...
---   Total air time        = 155,715.00 minutes ...
---   Minimum flight time   = 25.00 minutes ...
---   Maximum flight time   = 353.00 minutes ...
---   Average flight time   = 140.03 minutes ...
---   Total flight distance = 1,175,511 miles ...
---
--- Tail number: N479UA ...
---   Total no flights      = 1,121.00 ...
---   No flights/day        = 3.07 ...
---   Total air time        = 155,846.00 minutes ...
---   Minimum flight time   = 23.00 minutes ...
---   Maximum flight time   = 371.00 minutes ...
---   Average flight time   = 139.52 minutes ...
---   Total flight distance = 1,174,372 miles ...
---
--- Tail number: N440UA ...
---   Total no flights      = 1,167.00 ...
---   No flights/day        = 3.20 ...
---   Total air time        = 164,729.00 minutes ...
---   Minimum flight time   = 27.00 minutes ...
---   Maximum flight time   = 350.00 minutes ...
---   Average flight time   = 141.89 minutes ...
---   Total flight distance = 1,254,513 miles ...

--- Part 08: Southwest Planes: [N486WN, N293WN, N8641B, N8303R, N7707C ] ...
--- ================================================================== ...
---
--- Tail number: N486WN ...
---   Total no flights      = 1,981.00 ...
---   No flights/day        = 5.43 ...
---   Total air time        = 202,706.00 minutes ...
---   Minimum flight time   = 29.00 minutes ...
---   Maximum flight time   = 323.00 minutes ...
---   Average flight time   = 104.27 minutes ...
---   Total flight distance = 1,485,489 miles ...
---
--- Tail number: N293WN ...
---   Total no flights      = 1,883.00 ...
---   No flights/day        = 5.16 ...
---   Total air time        = 193,334.00 minutes ...
---   Minimum flight time   = 28.00 minutes ...
---   Maximum flight time   = 325.00 minutes ...
---   Average flight time   = 104.11 minutes ...
---   Total flight distance = 1,408,200 miles ...
---
--- Tail number: N8641B ...
---   Total no flights      = 1,577.00 ...
---   No flights/day        = 4.32 ...
---   Total air time        = 219,737.00 minutes ...
---   Minimum flight time   = 31.00 minutes ...
---   Maximum flight time   = 349.00 minutes ...
```

```
---   Average flight time   = 140.23 minutes ...
---   Total flight distance = 1,675,402 miles ...
---
--- Tail number: N8303R ...
---   Total no flights      = 1,489.00 ...
---   No flights/day        = 4.08 ...
---   Total air time        = 204,930.00 minutes ...
---   Minimum flight time   = 29.00 minutes ...
---   Maximum flight time   = 360.00 minutes ...
---   Average flight time   = 139.22 minutes ...
---   Total flight distance = 1,570,895 miles ...
---
--- Tail number: N7707C ...
---   Total no flights      = 1,962.00 ...
---   No flights/day        = 5.38 ...
---   Total air time        = 203,462.00 minutes ...
---   Minimum flight time   = 27.00 minutes ...
---   Maximum flight time   = 340.00 minutes ...
---   Average flight time   = 105.42 minutes ...
---   Total flight distance = 1,498,521 miles ...
```

Points to Note:

- Overall, the poorest quality of data is associated with delays – delays in departure time, air systems delays, security delays, late aircraft delays, and delays caused by weather. Our analysis is very rough and does not take into account how measurement of delays might have changed over time – presumably it's a lot better today than say thirty years ago.
- Flight times range from 7 mins up to 690 mins (11.5 hours).
- United Airlines (722 planes) and Southwest Airlines (705 planes) have roughly number of planes, but mix of plane types and the way in which these planes are used is quite different. Southwest is well known for its exclusive use of Boeing 737 planes. United Airlines has a mix of plane types suitable for commuter-type and long-haul flights.
- Over the course of a year, Southwest planes spend significantly more time in the air flying than United. On average, Southwest planes take 4-5 trips per day, with average flight durations slightly longer than 100 minutes. United planes take 2.5 - 3.5 flights per day, with average flight durations 118 - 180 mins.