**Exam II (Data Ingestion & Inspection, Exploratory data analysis and Visualization)**

**Due date: Dec 7th 2019 (Saturday), 11:59 pm**

Working with real-world weather and climate data, you will use pandas to manipulate the data into a usable form for analysis and systematically explore it using the techniques you’ve learned. Skeleton code has been provided to you with blank space. Copy the code in Jupyter notebook and fill all the blanks. Output for all exam portions need to be visible in the jupyter notebook. Please submit one jupyter notebook with your name on it.

There are three questions. Each question carries 20 points and has multiple parts. Your submitted Jupyter notebook needs to be clearly labeled with each question number

# **Question 1: Data Ingestion and Inspection**

# **1. a Reading in a data file**

First identify the method to use to read the data. The problem with real data such as this is that the files are almost never formatted in a convenient way. In this exercise, there are several problems to overcome in reading the file. First, there is no header, and thus the columns don't have labels. There is also no obvious index column, since none of the data columns contain a full date or time.

Your job is to read the file into a DataFrame using the default arguments. After inspecting it, you will re-read the file specifying that there are no headers supplied.

The CSV file has been provided for you as the variable data\_file

Instructions:

* Import pandas as pd.
* Read the file data\_file into a DataFrame called df.
* Print the output of df.head(). This has been done for you. Notice the formatting problems in df.
* Re-read the data using specifying the keyword argument header=None and assign it to df\_headers.
* Print the output of df\_headers.head(). This has already been done for you. Hit 'Submit Answer' and see how this resolves the formatting issues.

**Skeleton code**

# Import pandas

# Read in the data file: df

df = pd.read\_csv(\_\_\_\_)

# Print the output of df.head()

print(df.head())

# Read in the data file with header=None: df\_headers

df\_headers = pd.read\_csv(\_\_\_\_, \_\_\_\_=None)

# Print the output of df\_headers.head()

print(df\_headers.head())

# 1.b Re-assigning column names

After the initial step of reading in the data, the next step is to clean and tidy it so that it is easier to work with.

In this exercise, you will begin this cleaning process by re-assigning column names and dropping unnecessary columns.

pandas needs to be imported in the workspace as pd, and the file NOAA\_QCLCD\_2011\_hourly\_13904.txt has to be parsed and loaded into a DataFrame df. Create comma separated string of column names, column\_labels, and list of columns to drop, list\_to\_drop. You may visit the link at the bottom of slides to identify labels names or create your own label names.

##### Instructions

* Convert the comma separated string column\_labels to a list of strings using .split(','). Assign the result to column\_labels\_list.
* Reassign df.columns using the list of strings column\_labels\_list.
* Call df.drop() with list\_to\_drop and axis='columns'. Assign the result to df\_dropped.
* Print df\_dropped.head() to examine the result. This has already been done for you.

**Skeleton code**

# Split on the comma to create a list: column\_labels\_list

column\_labels\_list = \_\_\_\_

# Assign the new column labels to the DataFrame: df.columns

\_\_\_\_ = column\_labels\_list

# Remove the appropriate columns: df\_dropped

df\_dropped = \_\_\_\_

# Print the output of df\_dropped.head()

print(df\_dropped.head())

# 1.c Cleaning and tidying datetime data

In order to use the full power of pandas time series, you must construct a DatetimeIndex. To do so, it is necessary to clean and transform the date and time columns.

The DataFrame df\_dropped you created in the previous step is to be used and pandas has been imported as pd.

Your job is to clean up the date and Time columns and combine them into a datetime collection to be used as the Index.

##### Instructions

* Convert the 'date' column to a string with .astype(str) and assign to df\_dropped['date'].
* Add leading zeros to the 'Time' column. This has been done for you.
* Concatenate the new 'date' and 'Time' columns together. Assign to date\_string.
* Convert the date\_string Series to datetime values with pd.to\_datetime(). Specify the format parameter.
* Set the index of the df\_dropped DataFrame to be date\_times. Assign the result to df\_clean.

**Skeleton code**

# Convert the date column to string: df\_dropped['date']

df\_dropped['date'] = \_\_\_\_

# Pad leading zeros to the Time column: df\_dropped['Time']

df\_dropped['Time'] = df\_dropped['Time'].apply(lambda x:'{:0>4}'.format(x))

# Concatenate the new date and Time columns: date\_string

date\_string = \_\_\_\_

# Convert the date\_string Series to datetime: date\_times

date\_times = pd.\_\_\_\_(date\_string, \_\_\_\_='%Y%m%d%H%M')

# Set the index to be the new date\_times container: df\_clean

df\_clean = \_\_\_\_

# Print the output of df\_clean.head()

print(df\_clean.head())

# 1.d Cleaning the numeric columns

The numeric columns contain missing values labeled as 'M'. In this part, your job is to transform these columns such that they contain only numeric values and interpret missing data as NaN.

The pandas function pd.to\_numeric() is ideal for this purpose: It converts a Series of values to floating-point values. Furthermore, by specifying the keyword argument errors='coerce', you can force strings like 'M' to be interpreted as NaN.

A DataFrame df\_clean is to be used, and as usual, pandas has been imported as pd.

##### Instructions

* Print the 'dry\_bulb\_faren' temperature between 8 AM and 9 AM on June 20, 2011.
* Convert the 'dry\_bulb\_faren' column to numeric values with pd.to\_numeric(). Specify errors='coerce'.
* Print the transformed dry\_bulb\_faren temperature between 8 AM and 9 AM on June 20, 2011.
* Convert the 'wind\_speed' and 'dew\_point\_faren' columns to numeric values with pd.to\_numeric(). Again, specify errors='coerce'.

**Skeleton code**

# Print the dry\_bulb\_faren temperature between 8 AM and 9 AM on June 20, 2011

print(df\_clean.loc[\_\_\_\_:\_\_\_\_, \_\_\_\_])

# Convert the dry\_bulb\_faren column to numeric values: df\_clean['dry\_bulb\_faren']

df\_clean['dry\_bulb\_faren'] = \_\_\_\_(df\_clean['dry\_bulb\_faren'], \_\_\_\_=\_\_\_\_)

# Print the transformed dry\_bulb\_faren temperature between 8 AM and 9 AM on June 20, 2011

print(df\_clean.\_\_\_\_[\_\_\_\_:\_\_\_\_, \_\_\_\_])

# Convert the wind\_speed and dew\_point\_faren columns to numeric values

df\_clean['wind\_speed'] = pd.to\_numeric(\_\_\_\_, \_\_\_\_=\_\_\_\_)

df\_clean['dew\_point\_faren'] = pd.to\_numeric(\_\_\_\_, \_\_\_\_=\_\_\_\_)

# **Question 2: Exploratory data analysis (EDA)**

# 2.a Signal min, max, median

Now that you have the data read and cleaned, you can begin with EDA. First, you will analyze the 2011 Auburn weather data.

Your job in this exercise is to analyze the 'dry\_bulb\_faren' column and print the median temperatures for specific time ranges. You can do this using partial datetime string selection.

The cleaned dataframe is to be used as df\_clean.

##### Instructions

* Select the 'dry\_bulb\_faren' column and print the output of .median().
* Use .loc[] to select the range '2011-Apr':'2011-Jun' from dry\_bulb\_faren' and print the output of .median().
* Use .loc[] to select the month '2011-Jan' from 'dry\_bulb\_faren' and print the output of .median().

**Skeleton Code**

# Print the median of the dry\_bulb\_faren column

print(\_\_\_\_)

# Print the median of the dry\_bulb\_faren column for the time range '2011-Apr':'2011-Jun'

print(df\_clean.loc[\_\_\_\_:\_\_\_\_, 'dry\_bulb\_faren'].\_\_\_\_)

# Print the median of the dry\_bulb\_faren column for the month of January

print(df\_clean.\_\_\_\_[\_\_\_\_, \_\_\_\_].\_\_\_\_)

# 2.b Signal variance

You're now ready to compare the 2011 weather data with the 30-year normals reported in 2010. Questions to explore is, on average, how much hotter was every day in 2011 than expected from the 30-year average?

The DataFrames df\_clean and df\_climate from previous steps are to be made available in the workspace.

Your job is to first resample df\_clean and df\_climate by day and aggregate the mean temperatures. You will then extract the temperature related columns from each - 'dry\_bulb\_faren' in df\_clean, and 'Temperature' in df\_climate - as NumPy arrays and compute the difference.

Notice that the indexes of df\_clean and df\_climate are not aligned - df\_clean has dates in 2011, while df\_climate has dates in 2010. This is why you extract the temperature columns as NumPy arrays. An alternative approach is to use the pandas .reset\_index() method to make sure the Series align properly. You will practice this approach as well.

##### Instructions

* Downsample df\_clean with daily frequency and aggregate by the mean. Store the result as daily\_mean\_2011.
* Extract the 'dry\_bulb\_faren' column from daily\_mean\_2011 as a NumPy array using .values. Store the result as daily\_temp\_2011. Note: .values is an attribute, not a method, so you don't have to use ().
* Downsample df\_climate with daily frequency and aggregate by the mean. Store the result as daily\_climate.
* Reset the index of daily\_climate and extract the Temperature column. To do this, first reset the index of daily\_climate using the .reset\_index() method, and then use bracket slicing to access 'Temperature'. Store the result as daily\_temp\_climate.

**Skeleton code**

# Downsample df\_clean by day and aggregate by mean: daily\_mean\_2011

daily\_mean\_2011 = \_\_\_\_

# Extract the dry\_bulb\_faren column from daily\_mean\_2011 using .values: daily\_temp\_2011

daily\_temp\_2011 = \_\_\_\_

# Downsample df\_climate by day and aggregate by mean: daily\_climate

daily\_climate = \_\_\_\_

# Extract the Temperature column from daily\_climate using .reset\_index(): daily\_temp\_climate

daily\_temp\_climate = \_\_\_\_

# Compute the difference between the two arrays and print the mean difference

difference = daily\_temp\_2011 - daily\_temp\_climate

print(difference.mean())

# 2.c Sunny or cloudy

On average, how much hotter is it when the sun is shining? In this exercise, you will compare temperatures on sunny days against temperatures on overcast days.

Your job is to use Boolean selection to filter for sunny and overcast days, and then compute the difference of the mean daily maximum temperatures between each type of day.

The DataFrame df\_clean from previous step is to be used. The column 'sky\_condition' provides information about whether the day was sunny ('CLR') or overcast ('OVC').

##### Instructions

* Get the cases in df\_clean where the sky is clear. That is, when 'sky\_condition' equals 'CLR', assigning to is\_sky\_clear.
* Use .loc[] to filter df\_clean by is\_sky\_clear, assigning to sunny.
* Resample sunny by day ('D'), and take the max to find the maximum daily temperature.

**Skeleton code**

# Using df\_clean, when is sky\_condition 'CLR'?

is\_sky\_clear = df\_clean['sky\_condition']=='CLR'

# Filter df\_clean using is\_sky\_clear

sunny = df\_clean.loc[is\_sky\_clear]

# Resample sunny by day then calculate the max

sunny\_daily\_max = sunny.resample('D').max()

# See the result

sunny\_daily\_max.head()

# **Question 3: Visual Exploratory Data Analysis**

# 3.a Weekly average temperature and visibility

Is there a correlation between temperature and visibility? Let's find out.

In this part, your job is to plot the weekly average temperature and visibility as subplots. To do this, you need to first select the appropriate columns and then resample by week, aggregating the mean.

In addition to creating the subplots, you will compute the Pearson correlation coefficient using .corr(). The Pearson correlation coefficient, known also as Pearson's r, ranges from -1 (indicating total negative linear correlation) to 1 (indicating total positive linear correlation). A value close to 1 here would indicate that there is a strong correlation between temperature and visibility.

The DataFrame df\_clean has to be used.

##### Instructions

* Import matplotlib.pyplot as plt.
* Select the 'visibility' and 'dry\_bulb\_faren' columns and resample them by week, aggregating the mean. Assign the result to weekly\_mean.
* Print the output of weekly\_mean.corr().
* Plot the weekly\_mean dataframe with .plot(), specifying subplots=True.

**Skeleton code**

# Import matplotlib.pyplot as plt

# Select the visibility and dry\_bulb\_faren columns and resample them: weekly\_mean

weekly\_mean = \_\_\_\_

# Print the output of weekly\_mean.corr()

print(\_\_\_\_)

# Plot weekly\_mean with subplots=True

weekly\_mean.\_\_\_\_(\_\_\_\_=\_\_\_\_)

plt.show()

# 3.b Daily hours of clear sky

In the previous step, you analyzed the 'sky\_condition' column to explore the difference in temperature on sunny days compared to overcast days. Recall that a 'sky\_condition' of 'CLR' represents a sunny day. In this part, you will explore sunny days in greater detail. Specifically, you will use a box plot to visualize the fraction of days that are sunny.

The 'sky\_condition' column is recorded hourly. Your job is to resample this column appropriately such that you can extract the number of sunny hours in a day and the number of total hours. Then, you can divide the number of sunny hours by the number of total hours, and generate a box plot of the resulting fraction.

As before, df\_clean is to be made available in the workspace.

##### Instructions

* Get the cases in df\_clean where the sky is clear. That is, when 'sky\_condition' equals 'CLR', assigning the result to is\_sky\_clear.
* Resample is\_sky\_clear by day, assigning to resampled.

**Skeleton code**

# Using df\_clean, when is sky\_condition 'CLR'?

is\_sky\_clear = \_\_\_\_

# Resample is\_sky\_clear by day

resampled = \_\_\_\_

# See the result

resampled

# 3.c Heat or humidity

Dew point is a measure of relative humidity based on pressure and temperature. A dew point above 65 is considered uncomfortable while a temperature above 90 is also considered uncomfortable.

In this exercise, you will explore the maximum temperature and dew point of each month. The columns of interest are 'dew\_point\_faren' and 'dry\_bulb\_faren'. After resampling them appropriately to get the maximum temperature and dew point in each month, generate a histogram of these values as subplots. Uncomfortably, you will notice that the maximum dew point is above 65 every month!

df\_clean has been pre-loaded for you.

##### Instructions

* Select the 'dew\_point\_faren' and 'dry\_bulb\_faren' columns (in that order). Resample by month and aggregate the maximum monthly temperatures. Assign the result to monthly\_max.
* Plot a histogram of the resampled data with bins=8, alpha=0.5, and subplots=True.

**Skeleton code**

# Resample dew\_point\_faren and dry\_bulb\_faren by Month, aggregating the maximum values: monthly\_max

monthly\_max = \_\_\_\_

# Generate a histogram with bins=8, alpha=0.5, subplots=True

\_\_\_\_

# Show the plot

plt.show()

# 3.d Probability of high temperatures

We already know that 2011 was hotter than the climate normals for the previous thirty years. In this final step, you will compare the maximum temperature in August 2011 against that of the August 2010 climate normals. More specifically, you will use a CDF plot to determine the probability of the 2011 daily maximum temperature in August being above the 2010 climate normal value. To do this, you will leverage the data manipulation, filtering, resampling, and visualization skills you have acquired throughout this course.

The two DataFrames df\_clean and df\_climate are available in the workspace. Your job is to select the maximum temperature in August in df\_climate, and then maximum daily temperatures in August 2011. You will then filter to keep only the days in August 2011 that were above the August 2010 maximum, and use this to construct a CDF plot.

Once you've generated the CDF, notice how it shows that there was a 50% probability of the 2011 daily maximum temperature in August being 5 degrees above the 2010 climate normal value!

##### Instructions

* From df\_climate, extract the maximum temperature observed in August 2010. The relevant column here is 'Temperature'. You can select the rows corresponding to August 2010 in multiple ways. For example, df\_climate.loc['2011-Feb'] selects all rows corresponding to February 2011, while df\_climate.loc['2009-09', 'Pressure'] selects the rows corresponding to September 2009 from the 'Pressure' column.
* From df\_clean, select the August 2011 temperature data from the 'dry\_bulb\_faren'. Resample this data by day and aggregate the maximum value. Store the result in august\_2011.
* Filter rows of august\_2011 to keep days where the value exceeded august\_max. Store the result in august\_2011\_high.
* Construct a CDF of august\_2011\_high using 25 bins. Remember to specify the kind, normed, and cumulative parameters in addition to bins.

**Skeleton Code**

# Extract the maximum temperature in August 2010 from df\_climate: august\_max

august\_max = df\_climate.loc['2010-Aug','Temperature'].max()

print(august\_max)

# Resample August 2011 temps in df\_clean by day & aggregate the max value: august\_2011

august\_2011 = df\_clean.loc['2011-Aug','dry\_bulb\_faren'].resample('D').max()

# Filter for days in august\_2011 where the value exceeds august\_max: august\_2011\_high

august\_2011\_high = august\_2011.loc[august\_2011 > august\_max]

# Construct a CDF of august\_2011\_high

august\_2011\_high.plot(kind='hist', normed=True, cumulative=True, bins=25)

# Display the plot

plt.show()