Mary

Denison CS-181/DA-210 Quiz 2

March 12, 2021

Instructions

- Quiz is to be completed synchronously and within a total of 50 minutes.
- **No** electronic resources, including, but not limited to Notebooks from this or any prior semester or class, online help or cheatsheets, web pages, stack overflow or tutorials, or **any execution environment** (cloud or local).
- You **are** permitted a **handwritten** 4x6 index card one-sided, with whatever notes you choose to include.
- If in class, you must hand write your answers on the provided hard copy of the quiz.
- If you are remote, you should be in the class Zoom and in a breakout room with your screen shared **for the duration of the quiz**.
- If you are remote, you will be given both a PDF and a text file containing Markdown for the test. You can do one of the following:
  - Use a PDF editor to add text blocks to annotate the PDF with your answers, submitting the annotated PDF to Notebowl, or
  - Print the PDF, and then hand write your answers on the hard copy. Submit by scanning or taking pictures of the quiz pages and uploading to Notebowl, or
  - Use **only** a text editor and the Markdown version of the quiz, and type your answers at the appropriate place after each of the questions, and submit by uploading the (saved) markdown file. In this last case, you are permitted to also display/read the PDF version of the quiz.
- Any student should **upload to Notebowl** evidence (i.e. a picture) showing their extra credit study group. This must be done by noon on the day of the quiz.

Q1: Pandas/Tabular Operations

6 by 3 points (18 points total) 16/18

The following series of questions will all use the `flights` data frame, that we saw earlier, along with the `planes` and the `airports` data frame. The `pandas` data frame is referred to with Python variable `flights`. On a separate page, I show the abbreviated subset of this data that serves as the source data frame for this question.

If I ask for an **expression**, then you do not need to perform an assignment, but only need to write the expression that demonstrates the appropriate operation.
Assignments should only be performed if the operation requires it.

These are intended to be "one liners" that can be expressed on a single line. If you can demonstrate the **same** functionality using multiple lines, you will receive the majority of credit.

1-A Write an expression that projects the columns `month`, `day`, `carrier`, and `flight`.

```
# Write answer here
flights['month', 'day', 'carrier', 'flight']]
```

3/3

1-B Write an expression that selects the rows in `flights` where the carrier is UA.

flights[flights.carrier == 'UA']

1½/3

```
# Write answer here
flights.loc['carrier', 'UA']
```

: to get all columns

Need boolean series

1-C Write an expression that selects the rows for which the arrival time is later than the scheduled arrival, and project the columns for carrier, flight, tailnum, origin, and destination.

Good and creative; should work. Just not an expression/one-liner

```
df = pd.DataFrame()
# Write answer here
for rowlabel, rowseries in flights.iterrows():
    if flights.loc[rowlabel,'arrival'] > flights.loc[rowlabel,'sched_arrival']:
        df.append(flights.loc[rowlabel,['carrier','flight','tailnum','origin',
```

2½/3

1-D Write an expression that obtains the **Series** of the `distance` column for the rows that are in June (the 6th month).

flights.loc[, 'distance']

```
# Write answer here
distance = pd.Series(flights['distance'][flights['month'] == 6])
```

2/3

↳ this gives Series, so no need to "left"

1E- Update the data frame with a column called `air_time` with values that reflect the time in the air between arrival and departure.

```
# Write answer here
flights[air_time] = flights['arrival'] - flights['departure']
```

3/3

1-F Write an expression that finds the mean and median for both the `distance` and the `air_time` columns of `flights`

```
# Write answer here
flights.agg({'distance':['mean','median'], 'air_time':['mean','median']})
```

3/3

1-EC Extra Credit one-liner: Perform a mutation that, in a single line, updates the `air_time` column for only the rows where the arrival is less than sched_arrival (i.e. the plane arrived ahead of the scheduled arrival time) to increase the air time by 15%.

Nice idea. No 'apply' that deals w/ a 2 column data frame, but idea is sound

```
# Write answer here
flights[air_time] = flights[['arrival','sched_arrival']].apply(lambda x,y:
    (x-y)*0.15 if x<y )
```

1/2

Q2: Group By and Aggregation

Part 1 (4 pts) Write code (in as few or as many steps as you need) to find, for each carrier, the number of non-missing entries and the median of the `distance` column and the mean and maximum of the `air_time` column.

Clearly state the **number of rows** and **number of columns** you expect in your result.

```
# Write answer here
flights.groupby('carrier').agg({'distance':['median'], 'air_time':['mean', 'max']})
```

'count'

3/4

Rows: 3 (not counting headers)
Cols: 2 you actually have 3, and I was expecting 4.

Part 2 (6 pts) This is a multiple step problem also based on the `flights` data frame. Start by adding two new columns to the data frame. The new `dist_category` column should have values of `"short"` for flights whose distance is less than 1000 miles and `"long"` otherwise. The new `arrival_delta` column should have values with the difference between a flight's (actual) arrival time and its scheduled arrival time.

We then want a two-by-two table where we give the mean for `air_time` and `arrival_delta` for each of the two distance categories of `long` and `short`. My own solution is four lines of code.

6/6

```
# Write answer here
flights['dist_category'] = flights['distance'].apply(lambda x: 'short' if x < 1000 'long')
flights['arrival_delta'] = flights['arrival'] - flights['sched_arrival']
flights.groupby('dist_category').agg({'air_time':['mean'], 'arrival_delta',['mean']})
```

Q3: Combining Tables

5 points

Suppose, in addition to `flights`, you have the table `planes` whose index is the tail number (`tailnum`) of each plane. (See the reference sheet for this quiz.) Further suppose you want to combine `flights` and `planes` to obtain a single table with the columns: `month`, `day`, `carrier`, `flight`, `tailnum`, and `seats` for every flight in the flights table.

not a great idea in general, since a plane could be in multiple flights

```
# Write answer here
df2 = flights.set_index('tailnum')
df3 = pd.join(df2, Planes, how='inner')
df3 = df3.reset_index()
df4 = df3[['month', 'day', 'carrier', 'flight', 'tailnum', 'seats']]
df4
```

left

4/5

Solution

In combining tables, we have three choices: `concat`, `join`, and `merge`. We need columns from both tables.

- `concat` is not suitable, since we do not have the same rows
- The "matching" condition is when the `tailumn` in the `flights` table has the same value as the `tailnum` in the `planes` table, so
  - `join` is not suitable, as it needs the matching condition to be an index or part of an index in both tables
  - ... so `merge`, with an `on` of `tailnum` is the correct operation.

We need a `left` merge with the flights table as the left table, since we do not want any flights to disappear if there is not a corresponding entry in the `planes` table. (Note the "for every flight" in the wording of the problem.)

```
combined = pd.merge(flights, planes, on="tailnum", how="left")
combined[['month', 'day', 'carrier', 'flight', 'tailnum', 'seats']]
```

Q4: Tidy Data

10 points

Consider the following data frame, containing data from multiple weather stations with values of high and low temperatures seen at the station over various months of the year. For brevity, we are only showing columns for Jan through Apr, but you can assume the columns would continue through Dec. The data set could also, reasonably, have data for many more weather stations than just the two shown.

	StationID	StationName	Var	Jan	Feb	Mar	Apr
0	CMH	John Glenn Airport	high	34	37	49	61
1	CMH	John Glenn Airport	low	21	24	34	43
2	WDUB	Denison Slayter	high	36	40	51	63
3	WDUB	Denison Slayter	low	20	21	31	39

= date

1. Give the functional dependency or functional dependencies entailed in the data set. This will also give me your determination of independent and dependent variables.
2. Give, in English, the set of steps required to obtain one or more tidy tables that conform to your functional dependencies.
   - It may be helpful to name a table created in a particular step.
   - Your steps need not give parameters, but they **must** be clear and unambiguous, so that someone who knows a data frame package (like pandas) could translate your steps into actual code.
   - While not required, it can help you to receive partial credit if you tell me the columns resulting after any transformation or mutation step.

7½/10

StationID → StationName missing

StationID, month → high, low

Answer:

~~1.)~~ StationID is the functional dependency

An FD has <ind_vars> → <dep vars>

2.) melt with StationID, StationName, and Var as id_vars and the months as vars naming that column 'month'. Name variables to temp.

~~3.)~~ Split Var column into two new cols high and low then delete/drop var. → Needs a pivot to get into cols for correct

... how are rows affected in this "split"?

4.) Pivot w/ StationID and month as index, ~~StationName~~, high and low as cols, temp as values.

Var as pivot column we do not have a notion of multiple columns as the pivot col

Solution

Observations and Red Flags

1. We see the time series of months with month value across columns. If more data is added, we are adding columns; both are red flags.
2. We are inclined to aggegate *across a row*; also a red flag.
3. If we think about high and low temperatures (whether by month or by any other time period), these are **distinct measures**, and so are **variables**, not values of a categorical variable.
   - The fact that we have a column `Var`, whose entries are the names of variables is a red flag
4. We see repeated data for `StationID` and `StationName`, which seem to track one-for-one, and are not dependent on a month or on whether the measure is for a high or low temperature.

Functional Dependencies

From observation 4 above, we see that

$$\texttt{StationID} \to \texttt{StationName}$$

With the conclusion that high and low are variables, distinct measures, and the measures are performed at a particular station over a particular month, we get that station and month determine high and low:

$$\texttt{StationID, Month} \to \texttt{high, low}$$

Steps

A: Station Table from first FD

1. Project `StationID` and `StationName` from `df`, giving `stations`
2. Remove duplicates from `stations`

B: Temperature Table from second FD

1. Copy `df` to `df0`, dropping `StationName`
   - Columns in `df0`: `StationID`, `Var`, `Jan`, ...
2. Melt `df0` with the `StationID` and `Var` as the non-melt columns, and the months as the melt columns, naming the new var column `Month`, obtaining `df1`
   - Columns in `df1`: `StationID`, `Var`, `Month`, and `value`
3. Pivot `df1` using `Var` as the pivot column and `StationID` and `Month` as the "Index", copying to result of `temperatures`
   - Columns in `temperatures`: `StationID`, `Month`, `high`, `low`