Discussion 5 Solutions

NOTE: All of these solutions are available on practice.dsc80.com — this is just so you can see everything on one place.

FA23 Midterm Problem 4

Problem

To generate a single sample under his null hypothesis, Alan should

A. Resample 744 donkeys with replacement from `donkeys`.

B. Resample 372 donkeys with replacement from `donkeys` with `'BCS'` < 3, and another 372 donkeys with `'BCS'` >= 3.

C. Randomly permute the `'Weight'` column.

Solution

Answer: C

The null hypothesis is "Donkeys with `'BCS'` >= 3 have the **same** `'Weight'` values on average compared to donkeys that have `'BCS'` < 3". Under the null hypothesis, we should have similar results with a shuffled dataset.

Options A and B shuffle with replacement (bootstrapping), while option C shuffles without replacement (permutation is done without replacement). Bootstrapping is generally used to estimate confidence intervals, while permutation tests are a kind of hypothesis test. In this case, we are performing a hypothesis test, so we want to permute the `'Weight'` column.

Problem

Doris wants to use multiple imputation to fill in the missing values in `'WeightAlt'`. She knows that `'WeightAlt'` is MAR conditional on `'BCS'` and `'Age'`, so she will perform multiple imputation conditional on `'BCS'` and `'Age'` - each missing value will be filled in with values from a random `'WeightAlt'` value **from a donkey with the same `'BCS'` and `'Age'`**. Assume that all `'BCS'` and `'Age'` combinations have observed `WeightAlt` values. Fill in the blanks in the code below to estimate the median of `'WeightAlt'` using multiple imputation conditional on `'BCS'` and `'Age'` with 100 repetitions. A function `impute` is also partially filled in for you, and you should use it in your answer.

Solution

```
def impute(col):
    col = col.copy()
    n = col.isna().sum()
    fill = np.random.choice(col.dropna(), n)
    col[col.isna()] = fill
    return col
results = []
for i in range(100):
    imputed = (donkeys.groupby(['BCS', 'Age'])['WeightAlt'].transform(impu
    results.append(imputed.median())
```

We start with the bottom five blanks as we are not sure what the parameter of `impute(col)` is until we write the function call first. We see that we are using a loop, and seeing that we are doing multiple imputation with 100 reputations, we can fill in `range(100)`. We then define the variable
`imputed`, which we can see from the last line of code that calls `imputed.median()` should be a list of `'WeightAlt'` that has imputed values. Since we want to make
our imputation conditional on
`'BCS'` and `'Age'`, we can fill in the next blank with a `groupby` method and pass in the list of columns we want - `['BCS', 'Age']`. We can see we have then selected the `'WeightAlt'` column in the problem, and so we need to use our `impute` function on that series. We can do so with a `transform` method and then pass in `impute`. Note this can also be done with `apply` and receive credit, but this is our solution.

Now, we can define the `impute` function to impute missing values from `col`. Since we have already aggregated on `['BCS', 'Age']`, we know that our given `col` has samples all of the same `'BCS'` and `'Age'` values. Therefore, to impute as defined in the question, we just need to fill in `NaN` values with any other value from `col`, chosen at random. We can see we will use

`np.random.choice`, which takes in its first parameter possible choices in a list, and in its second parameter the number of choices to make. The number of choices to make we can define as `n`, which is the number of `NaN` values. This is found with `col.isna().sum()`. Then our possible choices are any non-`NaN` values in `col`, which we can use `col.dropna()` to find. Finally, we fill in the `NaN` values in `col` by masking for the `NaN` indices with `col[col.isna()]`, and set it equal to our `fill` values. That will successfully impute values into our `col` and we can then return it.

WI23 Final Problem 1

Problem

The DataFrame `sat` contains one row for **most** combinations of `"Year"` and `"State"`, where `"Year"` ranges between `2005` and `2015` and `"State"` is one of the 50 states (not including the District of Columbia).

The other columns are as follows:

- `"# Students"` contains the number of students who took the SAT in that state in that year.
- `"Math"` contains the mean math section score among all students who took the SAT in that state in that year. This ranges from 200 to 800.
- `"Verbal"` contains the mean verbal section score among all students who took the SAT in that state in that year. This ranges from 200 to 800. (This is now known as the "Critical Reading" section.)

The first few rows of `sat` are shown below (though `sat` has many more rows than are pictured here).

	Year	State	# Students	Math	Verbal
0	2014	Washington	41277	519	510
1	2013	Arizona	22283	529	522
2	2006	Kansas	2545	591	582
3	2011	North Dakota	219	612	586
4	2009	New Mexico	2209	548	553

The data description stated that there is one row in `sat` for ***most*** combinations of `"Year"` (between `2005` and `2015`, inclusive) and `"State"`. This means that for most states, there are 11 rows in `sat` — one for each year between 2005 and 2015, inclusive.

It turns out that there are 11 rows in `sat` for all 50 states, except for one state. Fill in the blanks below so that `missing_years` evaluates to an **array**, sorted in any order, containing the years for which that one state does not appear in `sat`.

Solution

```
state_only = sat.groupby("State").filter(lambda df: df.shape[0] < 11)
merged = sat["Year"].value_counts().to_frame().merge(
    state_only, left_index=True, right_on='Year', how='left'
) # an outer merge also works!
missing_years = merged[merged['# Students'].isna()]['Year'].to_numpy()
```

Blank A

The initial step (in the `state_only` variable) involves identifying the state that has fewer than 11 records in the dataset. This is achieved by the lambda function `lambda df: df.shape[0] < 11`, leaving us with records from only the state that has missing data for certain years.

Blank B

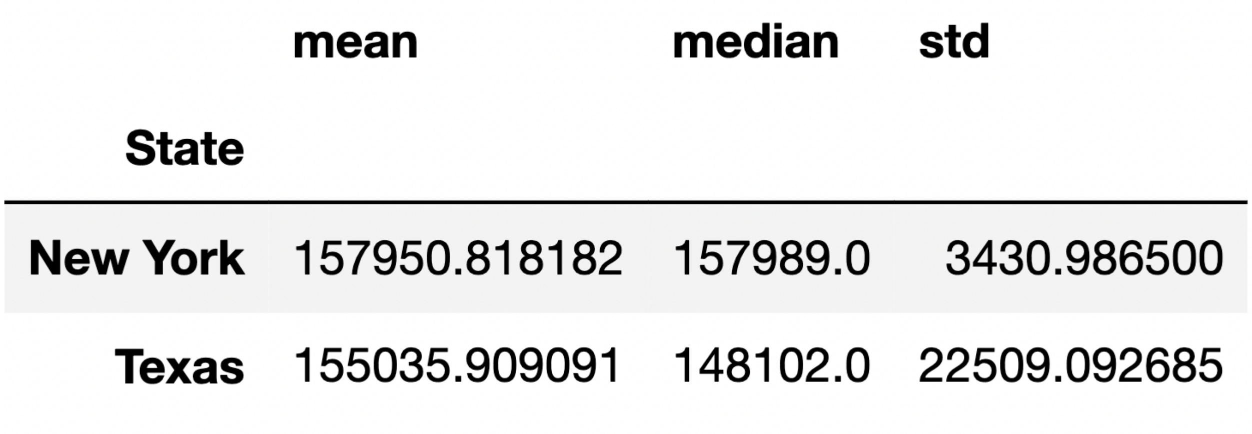
Next, applying `.value_counts()` to `sat["Year"]` produces a Series that enumerates the total occurrences of each year from 2005 to 2015. Converting this Series to a DataFrame with `.to_frame()`, we then merge it with the `state_only` DataFrame. This merging results in a DataFrame (merged) where the years lacking corresponding entries in `state_only` are marked as NaN.

Blank C

Finally, the expression `merged[merged['# Students'].isna()]['Year']` in `missing_years` identifies the specific years that are absent for the one state in the sat dataset. This is determined by selecting years in the merged DataFrame where the `"# Students"` column has NaN values, indicating missing data for those years.

Problem

The following DataFrame contains summary statistics for all SAT takers in New York and Texas from 2005 to 2015. Suppose we want to run a statistical test to assess whether the distributions of the number of students between 2005 and 2015 in New York and Texas are significantly different.



	mean	median	std
State			
New York	157950.818182	157989.0	3430.986500
Texas	155035.909091	148102.0	22509.092685

Given the information in the above DataFrame, which test statistic is **most likely** to yield a significant difference?

- $mean\,number\;of\;students\;in\,Texas\;-\;mean\,number\;of\;students\;in\;New\;York$
- $|\,mean\,number\;of\;students\;in\,Texas\;-\;mean\,number\;of\;students\;in\;New\;York\,|$
- $|\,median\,number\;of\;students\;in\,Texas\;-$
  $median\,number\;of\;students\;in\;New\;York\,|$
- The Kolmogorov-Smirnov statistic

Solution

Answer: The Kolmogorov-Smirnov statistic

Here, the means and medians of the two samples are similar, so their observed difference in means and observed difference in medians are both small. This means that a permutation test using either one of those as a test statistic will likely fail to yield a significant difference. However, the standard deviations of both distributions are quite different, which means the shapes of the distributions are quite different. The Kolmogorov-Smirnov statistic measures the distance between two distributions by considering their entire shape, and since these distributions have very different shapes, they will likely have a larger Kolmogorov-Smirnov statistic than expected under the null.