**Exercise 5**

**Task 3. Fill in missing values**

3a. Use DF.fillna()

popDF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = popDF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.fillna(\_\_\_\_\_\_)

**pop['Regionfilled'] = pop['Region'].fillna(-1)**

3b. Use DF.replace()

popDF\_\_\_\_\_\_\_\_\_\_\_\_.replace(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, inplace=True)

**pop['Regionfilled'].replace({-1:"NoReg"}, inplace=True)**

3c. Replace values by setting them equal to new value

popDF.loc[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_] = "DC"

popDF.loc[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_] = "PR"

**pop['RegionAll'] = pop['Region']**

**pop.loc[pop.index.isin(['Puerto Rico']), 'RegionAll'] = "PR"**

**pop.loc[pop.index.isin(['District of Columbia']), 'RegionAll'] = "DC"**

3d. Display the values of just the Region and newly created variables for the missing rows in original popDF.

popDF.loc[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_]

**pop.loc[pop.index.isin(['District of Columbia', 'Puerto Rico']), ['Region', 'Regionfilled', 'RegionAll']]**

**Task 4. Drop a column and view 5 random rows of data**

4a. Drop the column ‘Regionfilled’

popDF.drop(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**pop.drop('Regionfilled', axis=1, inplace=True, errors='ignore')**

**pop.head()**

4b. Display a random sample of 5 rows

popDF.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**pop.sample(5)**

**Task 5. Binning – Cuts and Quantiles**

5a. View the distribution of ‘y2019’

popDF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**pop['y2019'].describe()**

5b. Produce 5 equal size bins of ‘y2019.

pd.cut(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).head(10)

**pd.cut(pop.y2019, 5).head(10)**

5b.i. How many states/territories lie in each bin?

pd.cut(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**pd.cut(pop.y2019, 5).value\_counts()**

5c. Produce 5 quantiles of ‘y2019’

pd.qcut(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).tail(10)

**pd.qcut(pop.y2019, 5).tail(10)**

5c.i. How many states/territories lie in each quantile?

pd.qcut(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**pd.qcut(pop.y2019, 5).value\_counts()**

**Task 6. Create dummy variables (one hot encoding)**

6a. Create a DataFrame of dummies

dummiesDF = pd.get\_dummies (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**popDummies = pd.get\_dummies(pop.Region, prefix='R\_')**

**popDummies.head()**

6b. Join two DataFrames

popDF[\_\_\_\_\_\_\_\_\_\_\_\_\_\_].join(dummiesDF)

**pop[['Region']].join(popDummies)**

**Task 7. Create a new DF with subset of the columns and rename its index**

7a. Obtain subset of columns

popDF\_3y = popDF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**popDR\_3y = pop.loc[:,['Region', 'RegionAll','y2017', 'y2018', 'y2019']].copy()**

**popDR\_3y.head()**

7b. Rename its index

popDF\_3y.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_= 'State'

**popDR\_3y.index.rename('State', inplace=True)**

**popDR\_3y.head(3)**

**Task 8. Applying functions to DataFrame columns**

8a. Using Series unary and binary ufuncs to perform elementwise calculations

popDF\_3y['popchangepct'] = (\_\_\_\_\_\_\_\_\_\_\_\_\_ - \_\_\_\_\_\_\_\_\_\_\_\_\_) / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Write code to produce the following output from popDF\_3y for the top 5 highest popchangepct from 2018 to 2019.

popDF\_3y\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.sort\_values(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).\_\_\_\_\_\_\_\_\_\_\_\_

**popDR\_3y['popchangepct']=(popDR\_3y.y2019-popDR\_3y.y2018)/popDR\_3y.y2018**

**popDR\_3y.head()**

**popDR\_3y[['y2018', 'y2019', 'popchangepct']].sort\_values(ascending=False, by='popchangepct').head(5)**

8b. Using DF.applymap()

popDF\_3y.loc[\_\_\_\_\_\_\_\_\_\_\_\_].applymap(lambda x: np.round(\_\_\_\_\_\_\_\_\_\_,\_\_\_\_)).sort\_index().head()

popDF\_3y.loc[\_\_\_\_\_\_\_\_\_\_\_\_]\

.applymap(lambda x: np.round(\_\_\_\_\_\_\_\_\_\_,\_\_\_\_))\

.sort\_index()\

.head()

**popDR\_3y.loc[:, ['y2017', 'y2018', 'y2019']].applymap(lambda x: np.round(int(x)/1000000,2)).sort\_index().head()**

**popDR\_3y.loc[:, ['y2017', 'y2018', 'y2019']]\**

**.applymap(lambda x: np.round(int(x)/1000000,2))\**

**.sort\_index()\**

**.head()**

8c. Using DF.apply()

***8c.i. Using apply() to access column values - Standardization of a variable***

popDF\_3y\_\_\_\_\_\_\_\_\_\_\_\_\_\_.apply(lambda x : (x - \_\_\_\_\_\_\_\_\_\_) / \_\_\_\_\_\_\_\_\_\_).describe()

**popDR\_3y[['y2017','y2018','y2019']].apply(lambda x: ( x - x.mean() ) / x.std() ).describe()**

***8c.ii. Using apply() to accces column values - Normalization of a variable***

popDF\_3y\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.apply(lambda x : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_).describe()

**popDR\_3y[['y2017','y2018','y2019']].apply(lambda x: ( x - x.min()) /(x.max()-x.min())).describe()**

***8c.iii. Using apply() to access row values***

popDF\_3y.loc[\_\_\_\_\_\_\_\_\_\_\_\_\_].apply(lambda row : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, axis =1).value\_counts()

**# I’m including some selection using .loc, but I don’t think it matters since we’re running .value\_counts() which is summarizing the view anyways, right?   
# popDR\_3y.loc[:,['y2018', 'y2019']].apply(lambda row: row['y2018'] > row['y2019'] , axis=1).value\_counts()**

**popDR\_3y.apply(lambda row: row['y2018'] > row['y2019'] , axis=1).value\_counts()**

**Task 9. Perform calculations over groups**

9a. Compute within-RegionAll statistics

popDF\_3y[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_].groupby(popDF\_3y[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_]).sum().sort\_index()

**popDR\_3y[['y2017', 'y2018', 'y2019']].groupby(popDR\_3y['RegionAll']).sum().sort\_index()**

9b. Compute standardized variables within each group

popDF\_3y\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\

.groupby(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)\

.apply(lambda x : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )\

.sort\_index().head()

**popDR\_3y[['y2017', 'y2018', 'y2019']]\**

**.groupby(popDR\_3y['RegionAll'])\**

**.apply(lambda x: ( x - x.mean() ) / x.std() )\**

**.sort\_index().head()**

9c. Creating an output of aggregate values of the same size as grouped DF using DF.transform()

popDF\_3y['y2019'].groupby(\_\_\_\_\_\_\_\_\_\_\_\_\_\_).transform('count')

OR

popDF\_3y['y2019'].groupby(\_\_\_\_\_\_\_\_\_\_\_\_\_\_).transform(lambda x : x.count())

**popDR\_3y['y2019'].groupby(popDR\_3y['RegionAll']).transform('count')**

**popDR\_3y['y2019'].groupby(popDR\_3y['RegionAll']).transform(lambda x: x.count())**

**Task 10. Re-shaping DataFrames**

10a. Stacking only 2 years – 2018 and 2019

popDF\_3y\_stack = popDF\_3y\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.stack()

**# I didn’t have this column, adding now  
popDR\_3y['RegionMean']= popDR\_3y['y2019'].groupby(popDR\_3y['RegionAll']).transform('mean')**

**popDR\_3y.head()**

**popDR\_3y\_stack = popDR\_3y[['y2018', 'y2019']].stack()**

**popDR\_3y\_stack.index**

10b. Unstacking

popDF\_3y\_stack.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**popDR\_3y\_stack.unstack()**