Bellevue Almshouse dataset

- Data cleaning I
  - Converting data types
  - Duplicates and missing data
    - Frequency of values
- Renaming, deleting, and sorting columns
- Filtering/subsetting data

Tidy data structure

- Each variable is in a column
- Each observation is a row
- Each value is a cell

first_name	last_name	age
Mary	Gallagher	28.0
John	Sanin(?)	19.0
Anthony	Clark	60.0
Lawrence	Feeney	32.0

Converting data types

- Converting to date-time data type
  - data_frame[‘column_name’] = pd.to_datetime(data_frame[‘column_name’], format=’%Y%m%d)
    - e.g. bellevue_df['date_in'] = pd.to_datetime(bellevue_df['date_in'], format='%Y-%m%d')
- Another way to check data types
  - data_frame.dtypes

Dealing with duplicates

- .duplicated(keep = ‘first’/’last’/False):
  - Creates a True/False dataframe to check which rows in the original dataframe are duplicated
  - keep
    - first: considers the first entry in the dataframe as the unique entry
    - last: considers the last entry in the dataframe as the unique entry
    - False: considers all entry as duplicates
  - Default argument: keep = ‘first’

Dealing with duplicates

- df[df.duplicated(keep=False)]
  - Selects duplicated rows from the original dataframe that fulfills the True/False dataframe conditions
- .drop_duplicate(keep = ‘first’/’last’/False):
  - Drops all the duplicated rows and keeps the first entry, last entry, or none of the entries
  - Default argument: keep = ‘first’

Missing Data

- .isna() / .notna()
  - Creates True/False table for values with/out NA
    - dataframe_variable[‘column name'].notna()]
    - bellevue_df['professions'].notna()
  - Filters out NA values by comparing to original df
    - dataframe_variable[dataframe_variable[‘column name'].notna()]
    - e.g. bellevue_df[bellevue_df['professions'].notna()]

Missing Data

- .count()
  - count() method always excludes NaN values
  - To find the percentage of not blank data in every column:
    - bellevue_df.count() / len(bellevue_df)
- .fillna()
  - Fill the NaN values in the DataFrame with a different value by using the .fillna() method
    - bellevue_df['professions’].fillna('no profession information recorded')

Frequency: Most common items in a column

- df[“column_name”].value_counts()
    - To count the number of unique values in a column

Rename Columns

- .rename(columns={})
  - bellevue_df.rename(columns={‘professions': ‘jobs’})
  - To save the new column name to the dataframe, we need to overwrite the variable
    - bellevue_df = bellevue_df.rename(columns={‘professions': ‘jobs’})

Drop Columns

- .drop(columns=”column name”)
  - bellevue_df = bellevue_df.drop(columns="children”)

Sorting Columns

- .sort_values(by=’column_name’)
  - bellevue_df.sort_values(by='date_in', ascending=True)””)

Filter/Subset Data

- data_frame[‘column_name’] == ‘value’
  - Produces a True/False table based on condition
    - e.g. bellevue_df['profession'] == 'teacher’
- data_frame[data_frame[‘column_name’] == ‘value’
  - Filters out the rows from the original data frame that fits the condition
    - e.g. bellevue_df[bellevue_df['profession'] == 'teacher']

Groupby Columns

Allows us to group data and perform calculations on the groups

- Creates a groupby object
  - data_frame.groupby(‘column_name’)
    - bellevue_df.groupby('professions')

Groupby Columns

- Counting non-blank values in each column
  - data_frame.groupby(‘column_name’).count()
    - bellevue_df.groupby('professions').count()
- Isolating specifc column
  - data_frame.groupby(‘column_name’)[‘column2’].count()
    - bellevue_df.groupby('professions')[‘gender’].count()
- Stacking methods
  - data_frame.groupby(‘column_name’).count().sort()
    - bellevue_df.groupby('professions').count().sort(ascending=False)