Data quality plan

Summary of data quality.

Features with high data quality
There are some features here with a small percentage of missing values but all features here have missing value percentage of less than 5% so there is no need to impute them.
case_month
state_fips_code
age_group
current_status
sex
death_yn

Features with poor data quality (Will impute these features)
Even though some of these features are missing a lot of data. They all have more than 50% of the data and so we can impute the rest of the data using the 50% or more that we have. Also if I dropped these features in addition to the features that follow in the next section I will have dropped most of the dataset.
county_fips_code
race
ethnicity
symptom_status
hosp_yn
case_positive_specimen_interval.

Features with very poor data quality (will drop these features)
I will drop these features because they are missing most of the data and so even imputing the values would not be very informative about these features considering most of the data is missing.
process
exposure_yn
icu_yn
underlying_conditions_yn
case onset_interval

Data quality plan
Note that before beginning I already made a few decisions earlier to clean the dataset. I dropped the res_state and res_county features since these were effectively duplicates of the county_fips_code and the state_fips_code features. I also changed values that took the form

“Missing” to NaN values so that the true missing values % would appear in my descriptive statistics since some of the missing rows were already NaN and others took the form “Missing”. The code I used to implement this is available in the jupyter notebook. I will print it below also.

```
# I am dropping the columns res_state and the column res_county since we already have fips codes for both of these!
fr = fr.drop('res_state',  axis=1)
fr = fr.drop('res_county',  axis=1)
```

```
#### as you can see many of the features have values that take the value Missing
### which we are going to replace with NaN

fr['sex'].replace('Missing',  float('nan'),  inplace=True)
fr['age_group'].replace('Missing',  float('nan'),  inplace=True)
fr['race'].replace('Missing',  float('nan'),  inplace=True)
fr['ethnicity'].replace('Missing',  float('nan'),  inplace=True)
fr['process'].replace('Missing',  float('nan'),  inplace=True)
fr['exposure_yn'].replace('Missing',  float('nan'),  inplace=True)
fr['symptom_status'].replace('Missing',  float('nan'),  inplace=True)
fr['hosp_yn'].replace('Missing',  float('nan'),  inplace=True)
fr['icu_yn'].replace('Missing',  float('nan'),  inplace=True)
```

I am planning to drop the following set of features because these features have more than 50% of their data missing. When most of the data is missing imputing the data can lead to bias and inaccurate results. It can also introduce artificial correlations between the variables that can cause model overfitting. For these reasons dropping the features entirely is more desirable than imputing them.

process
exposure_yn
icu_yn
underlying_conditions_yn
case_onset_interval

In order to drop the features I will use the following python code

Import pandas as p
dataframe = dataframe.drop('feature_name', axis=1).

I will then check if I have dropped the features using

print(dataframe.dtypes)

Refer to the jupyter notebook to see the implementation.

```
In [398]: fr = fr.drop('process',  axis=1)
          fr = fr.drop('exposure_yn',  axis=1)
          fr = fr.drop('icu_yn',  axis=1)
          fr = fr.drop('underlying_conditions_yn',  axis=1)
          fr = fr.drop('case_onset_interval',  axis=1)

          print(fr.dtypes)
```

```
case_month                        category
state_fips_code                   category
county_fips_code                  category
age_group                         category
sex                               category
race                              category
ethnicity                         category
case_positive_specimen_interval    float64
current_status                    category
symptom_status                    category
hosp_yn                           category
death_yn                          category
dtype: object
```

The features have been dropped. Note that res_state and res_county were already dropped earlier for reasons explained in the jupyter notebook.

Imputing missing data for features with poor data quality
county_fips_code
race
ethnicity
symptom_status
hosp_yn
case_positive_specimen_interval.

Categorical features will be imputed with the modal value and continuous features will be imputed with the mean or median whichever is more relevant to the feature.

The following python code will impute missing values with the mode for a categorical feature.

```
    import numpy as np


import numpy as np


Mode = fr['county_fips_code'].mode().iloc[0]
fr['county_fips_code'] = fr['county_fips_code'].fillna(Mode)


Mode = fr['race'].mode().iloc[0]
fr['race'] = fr['race'].fillna(Mode)

Mode = fr['ethnicity'].mode().iloc[0]
fr['ethnicity'] = fr['ethnicity'].fillna(Mode)

Mode = fr['symptom_status'].mode().iloc[0]
fr['symptom_status'] = fr['symptom_status'].fillna(Mode)

Mode = fr['hosp_yn'].mode().iloc[0]
fr['hosp_yn'] = fr['hosp_yn'].fillna(Mode)
```

Now to check we succeeded

```
y= fr['county_fips_code'].isna().sum()
print(y)
```

```
4]: y= fr['county_fips_code'].isna().sum()
    print(y)

    0
```

Answer is 0 in the notebook so all NaN values were replaced.

Next I will replace all negative values in case_positive_specimen interval with NaN values.
This is because it is logically incoherent for the positive result date to precede the earliest date for covid.

import numpy as np
fr['case_positive_specimen_interval'] = np.where(fr['case_positive_specimen_interval'] < 0, np.nan, fr['case_positive_specimen_interval'])

Next I will impute the missing values for the case_positive_specimen_interval feature.
I will impute the missing rows with the mean value. First I will calculate the mean and then use the .fillna method to replace the NaN values.

Code
import numpy as np
fr['case_positive_specimen_interval'] =
fr['case_positive_specimen_interval'].fillna(fr['case_positive_specimen_interval'].mean()

Test for success

y= fr['case_positive_specimen_interval'].isna().sum()
print(y)

```
In [469]: y= fr['case_positive_specimen_interval'].isna().sum(
          print(y)

          0
```

Since y = 0 it means all the NaN values were converted.