

imputation

December 7, 2017

CS109 A (Group 18) - Blake Barr | Lekshmi Santhosh

```
In [1]: import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
sns.set_style("whitegrid")
%matplotlib inline
```

```
In [2]: final_df = pd.read_json("output/final_only11.json")
```

We first decided that a variable needed to be there > 75% of the time for us to impute it. The threshold of 75% or greater was set to obtain realistic, reliable and complete imputations. Heursitics and the findings from EDA drove the design of our imputation strategy. We thought that the best way to impute data is to use **a simple linear interpolation by MSA**. This approach attempts to take advantage of the fact that a variable should be highly related to itself in the past and future within the MSA.

We designed two approaches and evaluated the efficacy using the downstream performance -

0.0.1 Approach 1

This equation is what will be used to impute a variable denoted Y_t for year t

$$Y_t = B_0 + B_1 Y_{t-1} + B_2 \frac{Pop_t}{Pop_{t-1}} + B_3 MSA_i$$

We will leverage the variable value from the previous year (if available) and iteratively fill Y_t until the variable is completely filled. For example - Consider a situation where a variable has missing values for both 2008 and 2009. We first predict the variable value for 2008 using 2007. The imputed value of 2008 is then used to estimate the missing value of 2009.

If the variable has missing values for all past years before year t , the abovesaid strategy won't work. In such instances, we use the observation for future years (i.e. the years after t) for imputation.

$$Y_t = Y_{t+1} * \frac{Pop_t}{Pop_{t+1}}$$

0.0.2 Approach 2

$$Y_t = Y_{t-n/t+n} * \frac{Pop_t}{Pop_{t-n/t+n}}$$

In this second approach, we will do simple population adjustment using the nearest available year. Here, n represents the number of missing periods we have to go to find the first non-missing data. First, we will go to first lag period (one year before). If that is missing, we will go to year after. If data is missing for the nearest years (both lag and lead), we then identify two years before the year in question. If that is missing, we attempt an interpolation using the two years after, etc.

0.0.3 Imputation of Carson City Data

Carson City MSA had no crime data for the largest city in its MSA. But, since Carson city MSA is just Carson city, we used the MSA wide variables as its city-wide crime variables. Additionally, it was missing unemployment data, so we just grabbed annual unemployment data from the Bureau of Labor Statistics and filled the missings with those.

```
In [3]: ##### SET IMPUTATION STRATEGY HERE
        # This variable serves as a toggle to switch between the two imputation approaches
        imputation_strategy = 1

In [4]: final_df = final_df.sort_values(['join_key', 'year'])##### Generating Population Change

In [5]: # Get List of Missing Variables
        miss_vars = []
        for v in final_df.columns:
            miss_pct = np.sum(final_df[v].isnull()) / final_df.shape[0]
            if miss_pct > 0 and miss_pct < 0.25 and v not in ['largest_city']:
                miss_vars.append(v)
            if miss_pct > 0.25:
                del final_df[v]
```

Generating Population Changes

```
In [6]: # Get Population Rate Change
        # Generate lag given varname and number of periods
        gen_lags = lambda var, num: final_df.groupby(['join_key'])[var].shift(num)
        # Get Lag Year Difference
        final_df = final_df.sort_values(['join_key', 'year'])
        final_df['lag_year_diff'] = final_df.year - gen_lags('year', 1)
        print(final_df['lag_year_diff'].value_counts(dropna=False))
        # Get Lag Population
        final_df['lag1_pop_change'] = final_df['msa_pop'] / gen_lags('msa_pop', 1)

        # Get Lead Year Difference
        final_df['lead_year_diff'] = final_df.year - gen_lags('year', -1)
        print(final_df['lead_year_diff'].value_counts(dropna=False))
        # Lead 1 Population
        final_df['lead1_pop_change'] = final_df['msa_pop'] / gen_lags('msa_pop', -1)
```

```

del final_df['lead_year_diff']
del final_df['lag_year_diff']

1.0    2110
NaN     211
Name: lag_year_diff, dtype: int64
-1.0    2110
NaN     211
Name: lead_year_diff, dtype: int64

```

Imputation Strategy # 1

```

In [7]: if imputation_strategy == 1:
    print("Executing Strategy 1")
    # Imputation Approach # 1
    for v in miss_vars:
        lag_v = 'lag_%s' %v
        #####
        # While current variable has missing value
        #####
        # Using a counter so it does not go infinite loop
        # If missing all 11 values, won't be able to impute
        i = 0
        while np.sum(final_df.loc[:, v].isnull()) > 0:
            # Grab previous period value (either year before or two years)
            final_df[lag_v] = gen_lags(v, 1)
            # Using Lag variable and MSA FE
            x_vars = [v for v in final_df.columns if 'MSA_' in v] + [lag_v]

            # train - lag not missing v not missing
            # test - lag not missing. v missing
            lag_notnull = final_df[lag_v].notnull()
            train = (final_df[v].notnull()) & (lag_notnull)
            test = (final_df[v].isnull()) & (lag_notnull)

            # If there are situations where v is missing
            # but lag is also missing for all. We cannot do anything
            # we will have to use future data
            if np.sum(test) == 0 or i == 11:
                break
            # Only have to run regression once -
            # Then iteratively fill using coefficients
            if i == 0:
                lin_reg = LinearRegression().fit(final_df.loc[train, x_vars],
                                                  final_df.loc[train, v])

```

```

        i = i + 1
        # Fill in missing data with predictions
        final_df.loc[test, v] = lin_reg.predict(final_df.loc[test, x_vars])
    # Get rid of lag var if you created one
    if lag_v in final_df.columns:
        del final_df[lag_v]
    #####
    # Still Missing - Use Future Data
    #####
    # Similiar counter.
    # Do not get stuck in infinite loop if all future also missing
    j = 0
    while np.sum(final_df.loc[:, v].isnull()) > 0:
        if j == 11:
            break
        lead_v = 'lead_%s' %v
        # Get future Value
        final_df[lead_v] = gen_lags(v, -1)
        # Need v to be missing and v in next period to be next missing
        null_v = (final_df[v].isnull()) & (final_df[lead_v].notnull())
        #  $Y_t = Y_{t+1} * (P_t / P_{t+1})$ 
        final_df.loc[null_v, v] = final_df.loc[null_v,
                                                lead_v] * final_df.loc[null_v,
                                                                        "lead1_pop_cl

        j = j + 1
        del final_df[lead_v]

```

Executing Strategy 1

0.0.4 Imputation Strategy # 2

```

In [8]: '''
        Function
        -----
        adjust_var

        Given lag/lead and period This function will return the variable in that lag/lead
        period times population change between current period and that period
        '''
    def adjust_var(df, var, period_num, lead=False):
        if lead:
            period_num = -1 * period_num
        shift_var = "shift_%s" %var
        df.loc[:, shift_var] = gen_lags(var, period_num)
        df.loc[:, 'shift_pop'] = df['msa_pop'] / gen_lags('msa_pop', period_num)
        miss = df[var].isnull()
        # Fill missing with adjustment

```

```

df.loc[miss, v] = df.loc[miss, shift_var] * df.loc[miss, 'shift_pop']
del df[shift_var]
del df['shift_pop']
return(df)

```

```

In [9]: if imputation_strategy == 2:
    print("Executing Strategy # 2")
    # This we will juse use lead and lag value with population adjustment
    get_missing = lambda v: np.sum(final_df.loc[:, v].isnull())
    # Switch off lag, lead, lag, lead
    for v in miss_vars:
        if get_missing(v) > 0:
            for period in range(1, 11):
                # Do Lags and then lead for each period
                for lead_bool in [False, True]:
                    final_df = adjust_var(final_df, v, period, lead=lead_bool)
                if get_missing(v) == 0:
                    break

```

0.0.5 Imputation for Carson City

Carson city was missing city crime statistics. Since this MSA just includes Carson city we decided to use the MSA wide crime stats as the city stats.

```

In [10]: carson = final_df.city_key.str.contains("Carson")
final_df.loc[carson, ['MSA', 'year']].head(13)
crime_vars = ['violent_crime', 'mur_mans', 'rape', 'robbery',
              'assault', 'property', 'burglary', 'larceny', 'mv_theft']
for v in crime_vars:
    final_df.loc[carson, 'city_%s' %v] = final_df.loc[carson, v]
# Fix population
final_df.loc[carson, 'city_pop'] = final_df.loc[carson, 'msa_pop']

In [11]: del final_df['lead1_pop_change']
del final_df['lag1_pop_change']

```

Still missing Carson City unemployment data. Going to use unemployment data from BLS to fill these in

```

In [12]: # Going to just use annual Carson City unemployment data for
# 3 unemployment vars missing for carson cityall variables
carson_labor = pd.read_excel("data/NVCARSOURN.xls", skiprows=10)
carson_labor['year'] = carson_labor["observation_date"].dt.year
carson_labor = carson_labor.loc[(carson_labor.year >= 2006) &
                                (carson_labor.year <= 2016), :]
annual_stats = list(carson_labor.groupby("year")['NVCARSOURN'].mean())

# Fill in Carson City Crime Stats using Annual Stats
for i, year in enumerate(range(2006, 2017)):

```

```

        final_df.loc[(final_df['join_key'] == 'Carson City-NV') &
                     (final_df.year == year),
                     ['unemp_16_19', 'unemp_16_ovr', 'unemp_female']] = annual_stats[i]

In [13]: # Check for remaining missing
for v in final_df.columns:
    if "MSA_" not in v:
        miss_pct = np.sum(final_df[v].isnull()) / final_df.shape[0]
        if miss_pct > 0:
            print(v)
            print("MSAs")
            print(final_df.loc[final_df[v].isnull(), 'join_key'].unique())

largest_city
MSAs
['Carson City-NV' 'Texarkana-AR-TX']

In [14]: # Save version based on imputation variables
final_df.to_json('output/final_imputed%i.json' %imputation_strategy)

```