Analyzing the relationship between English church appointments and population of English settlements in 1575-1800

Iana Fedorchuk *

Data Analysis course paper

Volkswirtschaft Fakultät

Ludwig-Maximilians-Universität München

Betreuer: Mathias Bühler, PhD

München, 26.08.2022

*Matrikelnummer: 12148235; Iana.Fedorchuk@campus.lmu.de

Contents

Introduction

The purpose of this paper is to describe and present results of the project: ”Analyzing the relationship between English church appointments and population of settlements in 1575-1800.”. Also to define, is the relationship exist at all, and what else does one need to do to find it out. The goal of the project is to analyze the relationship between English church appointments and population of the English settlements in the past, to see if any conclusions about significance of relationship can be made, to provide ideas for extension of the project and discussion of how the results of this or extended project can be helpful for other historical data based papers, or even for research that is based on the recent data observations. Also, to propose further research ideas that somehow correlate with the topic. To understand how to look at the bigger picture and to see more potential effects inside and outside this project.

The H0 hypothesis of this paper is: ”The English church appointments effected on population of English settlements in 1575-1800”.

To test the H0 hypothesis was created a data set that included: 970 cities in England,

English church appointments for each city where two or more people were present. The time frame is 10 periods between 1575 and 1800 with a step of 25 years. The database stores data up to year 1835, but after working with the data (grouping and cleaning) the time frame was fixed to 1575-1800. The total amount of appointments counted in the final dataset is 45777. Average yearly count of appointments is 10.2. Not all the appointments were selected for the further estimation. Only appointments that had multiple people present, got in to the final dataset and statistics. I explain it the following way: as meeting a person that serves in church for most of the visitors is not the first experience of meeting them, thus this kind of appointments can not tell us anything, but also potentially add noise to the data.

The data source for appointments is The Clergy Database (CCED or Clergy of the Church

of England database). The database includes English church appointments for each unique location that is indicated as CCED id and year of the appointment. CCED id indicates a part of an

English settlement. Appointments in CCED are classified by the type of appointment. And for each appointment one can find a unique person id that was present.

The paper is organized as follows. Section 1 presents the overview data preparation process for further estimation and testing of the H0 hypothesis. Section 2 describes the results of

estimated panel OLS models and regression models estimated with python machine learning library scikit-learn.

1 Data

This section describes the steps that were made to prepare the data set for testing H0 hypothesis.

The initial data contains two parts:

1) CCED appointments data. It includes unique location id (CCED id), year of appointment,

type of appointment and person id and name that attended the appointment.

2) Population data. It includes unique location (CCED id), city id (C ID), year, latitude, longitude and population.

At first as data with appointments and complete location for an appointment list were in

different files, I created a loop that merges each pair of appointment list and location.

Next, as the goal for the time frame was to have data grouped by year groups 1575-1825 with a step 25 years, at first I assigned to all appointments and population records years a year group. The algorithm was created the way that each year gets the closest year group assigned. For example, year 1823 was mapped to year group with value 1825 and year 1861 to year group with value 1850.

The next step was to fill missing values in population records table. At this step the missing values were filled with an average value of population between the preceding period and later period (for which data was available).

After, I merged the population records with appointments and their locations, based on the unique location id (CCED id), and later grouped the final table based on the year group, city, count of appointments with multiple people present. Only appointments that had multiple people present, got in to the final data set and statistics. As not meeting anybody but the person who serves in the church would hardly affect population rate. And likely the attendee of the appointment already familiar with the person from the church. Thus, in the data set were only included the appointments where at least two people were present (not counting the people, who serve in the church).

With all the described steps and more minor ones that can be found in the code, I received the final data set, which I already could use in empirical estimation.

The final data set included in total: 970 cities in England and 45777 appointments observed in 10 periods between 1575-1800 with step 25 a years. Average amount of appointments per year was 10.2.

On the Figure 1, is shown the relationship between population and English church appointments. The relationship is barely seen.

The correlation between church appointments and population overall is 0.0085, which is very low.

The annual correlation provides a slightly different results, and also shows that the corre-

Figure 1 — Relationship between Population and English church appointments

lation change by time, which could be true, or caused by the a lot of missing data. Starting at the beginning of observed period with low negative correlation, and later changing to a small positive correlation.

year	correlation between population and appointments
1575	-0.06
1600	-0.1
1625	-1.0
1650	-0.07
1675	-0.05
1700	-0.06
1725	0.03
1750	-0.07
1775	0.2
1800	0.01

2 Models and Results

In this section presented the results of empirical estimation for testing the H0 hypothesis

of the project for panel OLS estimation and scikit-learn regressor models.

For the first step of estimation were panel OLS models with Fixed Effects clustering and

without FE clustering. The models were estimated witha help fro module PanelOLS from

Python library linearmodels. Two specification of panel models were tested: normal regression and regression in logarithms

For estimating linear panel OLS in Python, the data should have MultiIndex that is created

by combining city id and year group as a tuple for each row.

5

2.1 Results of Panel OLS models

The first model specification equation looks as follows:

$population_{i,t} = appointments_{i,t} + city_i + \epsilon_{i,t}$ (1)

The second model specification equation looks as follows:

$ln(population_{i,t}) = ln(appointments_{i,t}) + city_i + \epsilon_{i,t}$ (2)

The only difference between the first and second specification is that the second equation

has log-log form.

The results of the first model specification showed: for the first model - with clustering city

FE, the coefficient in front of appointments is statistically significant at less then 10 percent level.

And for the second model without clustering city fixed effects, appointments coefficient has pvalue=0.0003 which means that coefficient in front of appointments is statistically significant on less then 1 percent level. The point estimate of the coefficient in front of appointments for clustering and not-clustering models is the same 16.4.

The second specification model result showed that coefficient in front of ln(appointments) is statistically significant at less then 1 percent level for both clustering and not-clustering methods (p-value=0.00). The point estimate for both methods is also the same and equal to around 0.11.

Based on this result I can not deny that there might be effect of English church appointments on population, I would say this matter need a deeper look, and digging dipper into historical data.

Here I included analysis with clustering Fixed Effects in case the data heteroscedasticity or has correlation inside of the clusters.

As one can see there were only difference in p-values for the first model specification. And estimating the model in log-log format, almost didn't have difference in p-values and in point estimate, which mean that using log-log form for equation, helped with the heteroscedasticity problem within the clusters.

That is a not surprising result, even though the city fixed effects were included, the results show that most probably estimate has a big positive bias, because we have a problem of omitted variable. Population is a very complex, and a lot of different factors can affect it. And of course quite a number of factors were not included into the equation.

2.2 Results of Regressor models from Scikit-learn Python library

Next part of the empirical analysis was focused on predicting inside the data frame by using different methods from scikit-learn, machine learning library, in Python. Results from the four following regression models were estimated: Linear Regressor, Random Forest Regressor, Lasso regressor, and LassoLars Regressor. Only regression models were used, as both population and appointments data represents continues quantity values. Thus, classification models would be irrelevant in this case, as they are used for predicting categorical (or class variables).

The results of the models were rated between each other based on typical metrics that is used for comparing regression models:

1) Mean Absolute Error (MAE)

2) Mean Squared Error(MSE)

3) Root Mean Squared Error (RSME)

After receiving the metrics, the easiest metrics to compare results with was Mean Absolute

Error. As the numbers are shorter from other metrics.
Listing of the MAE for the estimated models, sorted in ascending order. The lower the metrics the better. The same order of the scoring if one compares by MSE or RMSE in this specific case.

1) LassoLars Regressor, LARS stands for Least-Angle-Regression (MAE: 1601.82)
2) Lasso regressor (MAE: 1601.83)
3) Linear Regressor (MAE: 1602.86)
4) Random Forest Regressor (MAE: 1658.95)

The results show that the LassoLars model performed the best on predicting inside the data set. And Random Forest performed the worst from all tested models. Lasso and LassoLars MAE are very close. Scikit-learn library recommends to use linear or Lasso regression when the amount of rows in the data frame are lower than 100 thousand, which is the case for the data set from this research. The amount of rows in final dataset is 4394. Which pretty low for using advance methods like Random Forest. And the simpler methods like linear model and lasso model should manage to perform good on the final dataset from this research. LassoLars model is just a variation of Lasso model, that is why the results of Lasso and LassoLard are very close.

Conclusion

In conclusion, it would be good to say that for now with the current data available, one can not completely define the relationship between population and appointments. One can also argue that the appointments for the final data set should have been picked differently. But, the biggest challenge of this research goal that was just slightly tackled in this paper is the data has

too many missing values already, but also some explanatory variables that should be included in estimation equation is not available and never will be available, as the time period in the observed data was a long time ago. But still some of the explanatory variables can be added to the analysis. As a recommendation of expanding and advancing the analysis of church appointments effects on population, would be nice to do a deeper research of the literature and agglomeration and urban theory. New Economic Geography (NEG) is a theory that emerged in literature relatively recently suggests that agglomeration forces affect the growth of population in settlements. And one of the variables that is suggested by New Economic Geography is market potential of the cities. Also, to increase the precision of the analysis, one can look at the major events during the observable time period to confirm or deny if those events might affected the population in England. Also preceding the observable time period effect might have caused structural shift that still have effect that decreasing over time, for example. In this case even adding year fixed effects would not help. Another factors that can affect population are policy laws from the country or state level laws. Again, before doing any conclusions about relationship between English church appointments and population deeper and broader research and analysis should be made.

I understand the importance of the historical data analysis and making research about relationship between data that were might be present in the past. But as the last observation in CCED database is from year 1835. I can not help but wonder, how the current project can help us explain some relationship (if extended) now and in the future. What could be the equivalent or proxy to church appointments almost 2 centuries ago? Some counties still have a high portion of religious people, some countries have a large proportion of atheist people due to their different history. What nowadays the 'new religion' that would make people meet, start conversations and make new friends from it? How it will look like in future? And how often would people meet in future in close geographical proximity. What would be the next space where we are going to calculate distance? It would be an interesting research to find a similar kind of relationship but now, especially when more and more data is available every day. But interestingly before when doing research (not only scientific) people would carefully think about all the variables, data points, methods that they want to use, as each step of the research would take much longer time than now. While now people maybe put less thought into their data and methods, but the availability allows them to make mistakes and solve them faster.

References

1. Arts, Council) H. R. Clergy of the Church of England database. — URL: https : / / theclergydatabase.org.uk/jsp/search/index.jsp.

2. Hassink R., Gong H. New economic geography // The Wiley Blackwell Encyclopedia of Urban and Regional Studies. — 2016. — P. 513.

```
#IMPORTANT

#Code adapted from class, changes: instead of dowloading the output and location files to one folder,

#I saved them to two different folders, so I don't have to fish for them later with regular expression matching of the file names
```

```
#And the path to the source folder and output folders is also different from initial code

# Packages

from os import listdir

from os.path import isfile, join

# Try at home, using 'glob'

import pandas as pd

from bs4 import BeautifulSoup


# Startup code

files = [f for f in listdir('/home/iana/Downloads/Input/') if
isfile(join('/home/iana/Downloads/Input/',f))]

print(len(files))


# Processing if only one file: (dedent region)

#f = files[1]

# Processing if multiple files:

for f in files:

    print(f)

    with open('/home/iana/Downloads/Input/'+str(f),'rb') as file:

        soup = BeautifulSoup(file.read(),'html.parser')


    table = soup.find('div','tb s2')

    # classes of each table:

    for t in soup.find_all('table'):

        print(len(t))


    t1 = soup.find_all('table')

    if len(t1)>1:

        t2 = t1[1]


        df = pd.DataFrame(columns=['Names','PersonID','Year','Type','Full'])
    for row in t2.tbody.find_all('tr'):
        #print(row)
        columns = row.find_all('td')
        if len(columns)>0:
            #print(columns)
            names = columns[0].text.strip().replace("\r","").replace("\n","").replace("  "," ")
            year = columns[1].text.strip().replace("\r", "").replace("\n", "").replace("  ", " ")
            type = columns[2].text.strip().replace("\r", "").replace("\n", "").replace("  ", " ")
            office = columns[3].text.strip().replace("\r", "").replace("\n", "").replace("  ", " ")
            full = columns[4].text.strip().replace("\r", "").replace("\n", "").replace("  ", " ")
            c=columns[0].find('a',href=True)
            if c.__str__()!='None':
                persid = c['href'].replace('../persons/CreatePersonFrames.jsp?PersonID=','')
            else:
                persid = '0'


            df = df.append({'Names':names,'PersonID':persid,'Year':year,'Type':type,
'Office':office,'Full':full}, ignore_index=True)
    f1 = f.replace('.html','').replace('file','')
    df.to_csv('/home/iana/Downloads/Output/data'+str(f1)+'.csv',index=False)


    l1 = soup.find('ul',{'class':'s2'})
    cols = []
    for row in l1.find_all('li'):
        try:
            cols.append(row.label.text.replace('\xa0',' ').replace("\r", "").replace("\n", "").replace("  ", "
").replace(":", ""))
            last= row.label.text.replace('\xa0',' ').replace("\r", "").replace("\n", "").replace("  ", " ")
        except:
            cols.append(last)
    cols.append('parish')
```

```
        df = pd.DataFrame(cols)


        data = []
        for row in l1.find_all('li'):
            text = row.text.replace('\xa0',' ').replace("\r", "").replace("\n", "").replace("  ", " ")
            for c in range(0,len(cols)):
                text = text.replace(cols[c],"")
            data.append(text)


        l2 = soup.find('div',{'class':'ph'})
        data.append(l2.text.replace('\n','').replace('\r',''))
        df2 = pd.DataFrame(data)
        frame = [df,df2]
        final = pd.concat(frame,axis=1)
        final = final.transpose()
        final.to_csv('/home/iana/Downloads/Locations/Location'+str(f1)+'.csv',index=False)
#MY CODE STARTS HERE
#creating a list of ids -- CCED_IDS for files like data*.csv and Location*.csv
#I do this to be able to iterate through them later
#I import library -- glob, it is used for dealing with paths to files
import glob
#here I create a list of all the files in folder Output, which I mentioned earlier contains only files like data2.csv, data3.csv
#as the ids -- CCED_IDS for output files and location files are the same, we need to do this procedure only one time -- getting CCED_ids
a=list(glob.glob("/home/iana/Downloads/Output/*"))
#as the list that we just created contains not only file name but also the whole path to it in my computer,
#i need to remove two parts from it path and .csv
#it is done in the following two lines
a_stripped=[s.strip('/home/iana/Downloads/Output/') for s in a]
```

```
a_clean=[s.strip('.csv') for s in a_stripped]

#in order to be able to concatenate number with a string data or Location, we need to convert the ids to a string format

list_ids=[str(x) for x in a_clean]

#creating our strings to concatenate it with ids -- CCED_IDS in the next lines

data_string = 'data'

locaton_string = 'Location'

#concatenating strings with ids, so we can use the list of resulted strings in following loop to create dataframes with the names according to file names

data_names = [data_string + x for x in list_ids]

location_names = [locaton_string + x for x in list_ids]

#checking if it worked

print(data_names[88])

print(location_names[0])

#creating dataframes for each data and Location files, as we need to merge data and Location files according to their ids

data_path = '/home/iana/Downloads/Output/'


#this loop creates dataframes for each data file from folder output, with the names aka reading all the CSVs and converting them to dataframes

#each created dataframe is named according to the file it was read from

for name in data_names:

   globals()[name] = pd.read_csv(data_path+name+'.csv')

#at this step we have already all the data files dataframes in Kernel

#now I nead to read and create dataframes from all the location files, dataframes again have the name according to the file they were read from

location_path = '/home/iana/Downloads/Locations/'


#these location files had header as a first row, so we need to make python read the header from the first row

for name in location_names:

   globals()[name] = pd.read_csv(location_path+name+'.csv', header=[1])

#at this step we have created the dataframes from: all data and location files
```

```
#in previous steps I created list with location file names, and these names now stored in the list as strings

#but if i want to perform actions with the frames, i need to change the type of the variables in the list from string to object

#after this step python can understand that elements of the list not just strings, but dataframes that we created earlier

location_dfs = [eval(x) for x in location_names]

#now for each location dataframe I need to add CCED_ID as a column,

#so after I create a dataset with all appointments, I can differentiate between locations

#this loop assigns a new column with CCED_ID to each location dataframe,

for location_df, location_name in zip(location_dfs, location_names):

    location_df['cced_id'] = location_df['cced_id']=location_name

#now I want to merge each data and location dataframes together according to their CCED_ID

#creating string to form from it names of new merged dataframes

merged_string='merged'

merged_names = [merged_string + x for x in list_ids]

#after this step python can understand that elements of the list not just strings, but dataframes that we created earlier

data_dfs = [eval(x) for x in data_names]

location_dfs = [eval(x) for x in location_names]

#merging data and location dataframes, filling missing location values, as it was only one row, with forward fill method

for d,l,m in zip(data_dfs,location_dfs,merged_names):

    globals()[m] = pd.concat([d, l], axis=1).fillna(method='ffill')

#creating a list of merged dataframes from merged dataframes' names

merged_dfs = [eval(x) for x in merged_names]

#concatenating all merged dataframes together

df_all=pd.concat(merged_dfs)

print(df_all.columns)

#removing not needed columns

df_all=df_all.drop(['Contains',

       'Contains:', 'Contains:.1', 'Contains:.2', 'Contains:.3',

       'Contains:.4', 'Contains:.5', 'Contains:.6', 'Contains:.7',
       'Contains:.8', 'Contains:.9', 'Contains:.10', 'Contains:.11',
       'Contains:.12', 'Contains:.13', 'Contains:.14', 'Contains:.15',
       'Contains:.16', 'Contains:.17', 'Contains:.18', 'Contains:.19',
       'Contains:.20', 'Contains:.21'], axis=1)
#saving progress to protect from dead Kernel
df_all.to_csv('mathias_all.csv')
df_all.to_excel('mathias_all.xlsx')
df_all=pd.read_csv('mathias_all.csv')
#count of unique people in dataframe
print(df_all['PersonID'].nunique())
#removing string Location from column cced_id
df_all['cced_id'] = df_all['cced_id'].str.replace('Location', '')
#adding population dataframe to Kernel
pop=pd.read_stata('Population_year.dta')
#unique years in population dataframe
print(pop.year.unique())
#changing the type of data in Year column, to be able to filter by string lenght after
df_all["Year"]=df_all["Year"].values.astype('str')
#keeping in dataframe only year values that are 4 characters long
df_appointments=df_all[df_all['Year'].str.len()==4]
#changing type of values in Year column back to integers
df_appointments["Year"]=df_appointments["Year"].astype('int')
#checking all the unique Year values
appointments_years=df_appointments['Year'].unique().tolist()
appointments_years.sort()
print(appointments_years)
#1525(25)1850 is the goal for the dataframe, thus filtering appointments dataframe accordingly
df_appointments_years=df_appointments[(df_appointments['Year']>=1523)&(df_appointments['Ye
ar']<=1854)]
#changing type of year column's values from population dataframe to integer
pop['year']=pop['year'].astype('int')
```

```
a=pop['year'].unique().tolist()

a.sort()

#1525(25)1850 is the goal for the dataframe, thus filtering population dataframe accordingly

pop_years=pop[(pop['year']>=1550)&(pop['year']<=1850)]

#creating appointment years list

a_list=df_appointments_years.Year.tolist()

#creating a list of year groups to meet the dataframe requirements 1525(25)1850

group_list=list(range(1550, 1850, 25))

#for each year from appointments finding the index of group_list that the year is closest to

L1 = a_list

L2 = group_list



def find_closest(x):

   l2_diffs = [abs(x - y) for y in L2]

   return l2_diffs.index(min(l2_diffs))


#returns index of the year in the list and the index of the closest group as a tuple

k = [(i, find_closest(x)) for i, x in enumerate(L1)]


print(k)

#returns only second elements of tuples

#creating mask to match the group to years later

mask = [tup[1] for tup in k]


print(mask)

#creating dataframe from mask list

df_mask = pd.DataFrame(mask, columns =['mask_id'])

#creating a list of tuple for matching mask ids with year groups

matching = [(0, 1550), (1, 1575), (2, 1600), (3, 1625), (4, 1650), (5, 1675), (6, 1700), (7, 1725), (8,
1750), (9, 1775), (10, 1800), (11, 1825)]
```

```
df_mask["year_group"] = df_mask["mask_id"].map(dict(matching))

#adding to mask dataframe original years

df_mask['year']=list(df_appointments_years['Year'])

#removing duplicates from mask dataframe

df_mask_new = df_mask.drop_duplicates(keep = 'last').reset_index(drop = True)

#merging mask dataframe and appointments dataframe to assign year groups to appointments

appointments = pd.merge(df_appointments_years, df_mask_new, how="outer", left_on=["Year"], right_on=["year"])

#saving progress to protect from dead Kernel

appointments.to_csv('appointments_with_locations_and_year_groups.csv')

#creating a list of unique values from year column from population dataframe

pop_year_list=list(pop_years.year.unique())

#for each unique year from population dataframe finding the index of group_list that the year is closest to

L1 = pop_year_list

L2 = group_list



def find_closest(x):

   l2_diffs = [abs(x - y) for y in L2]

   return l2_diffs.index(min(l2_diffs))


#returns index of the year in the list and the index of the closest group as a tuple

k_new = [(i, find_closest(x)) for i, x in enumerate(L1)]


print(k_new)

#returns only second elements of tuples

#creating mask to match the group to years later

mask_new = [tup[1] for tup in k_new]


print(mask_new)

#creating dataframe from mask list
```

```
df_mask_new = pd.DataFrame(mask_new, columns =['mask_id'])

#creating a list of tuple for matching mask ids with year groups

matching = [(0, 1550), (1, 1575), (2, 1600), (3, 1625), (4, 1650), (5, 1675), (6, 1700), (7, 1725), (8, 1750), (9, 1775), (10, 1800), (11, 1825)]

df_mask_new["year_group"] = df_mask_new["mask_id"].map(dict(matching))

#adding to mask dataframe original years

df_mask_new['Year']=pop_year_list

#removing duplicates from mask dataframe

df_mask_new1 = df_mask_new.drop_duplicates(keep = 'last').reset_index(drop = True)

#merging mask dataframe and appointments dataframe to assign year groups to appointments

pop_year_groups_new = pd.merge(pop_years, df_mask_new1, how="outer", left_on=["year"], right_on=["Year"])

#saving progress to protect from dead kernel

pop_year_groups_new.to_csv('pop_year_groups_new.csv')

#reading the appointments dataframe that we assigned locations and year groups previously

appointments=pd.read_csv('appointments_with_locations_and_year_groups.csv')

#cleaning all the white spaces from appointment's types, so we don't have duplicates based on different amount of spaces

appointments['type_clean']=appointments['Type'].str.replace(" ","")

#removing not needed columns from appointments dataframe

appointments=appointments.drop(['Unnamed: 0.1','year', 'mask_id'], axis=1)

#reading population dataframe to which we assigned year groups previously

pop_year_groups_new=pd.read_csv('pop_year_groups_new.csv')

#now values of population in the table are floats, changing their type to integer

pop_year_groups_new['Population'] = pop_year_groups_new['Population'].astype(int)

#replacing missing values marked as 0 to numpy NaN values

pop_year_groups_new['Population']=pop_year_groups_new['Population'].replace(0, np.nan)

#in population dataframe there are missing values for population records, and I want to fill them later

#to fill missing population we don't need cced_id, we need c_id, pop, year, year group

pop_small = pop_year_groups_new.drop_duplicates(
  subset = ['C_ID', 'year_group', 'year'],
  keep = 'last').reset_index(drop = True)

#soring values in grouped population dataframe by 3 columns

pop_s = pop_small.sort_values(by=['C_ID', 'year_group', 'year'])

print (pop_s.head())

#filling missing population value with the average of nearest observations for each city -- C_ID: the
nearest previuos and nearest post observation mean

pop_s.loc[:,'Population_clean'] = pop_s.groupby('C_ID')['Population'].apply(lambda group:
group.interpolate(method='linear') if np.count_nonzero(np.isnan(group)) < (len(group) - 1) else
group)

pop_s.loc[:,'Population_clean'] = pop_s.groupby('C_ID').apply(lambda group:
group.interpolate(method='linear', limit_area='outside', limit_direction='both'))

#removing duplicates

pop_s_unique_year_group = pop_s.drop_duplicates(

  subset = ['C_ID', 'year_group'],

  keep = 'last').reset_index(drop = True)

#merging filled with averages population values back to the population dataframe

pop_not_na=pd.merge(pop_year_groups_new,
pop_s_unique_year_group[["C_ID","year_group","Population_clean"]], how="left",
left_on=["C_ID","year_group"], right_on=["C_ID","year_group"])

#saving progress to protect from dead Kernel

pop_not_na.to_csv('pop_not_na.csv')

#I decided to take into account only appointments that had at least two people present

#Thus, as a first step, I need group by the appointments dataframe by unique location id -- cced_id,
year_group and type of the appointment

#and count amount of people present at each appointment, for each location for each type of the
appointment

appointments_g=appointments.groupby(['cced_id',
'year_group','type_clean'])['PersonID'].size().reset_index()

#now from counted people at appointments we can filter appointments by having at least two
people

appointments_multiple_people=appointments_g[appointments_g['PersonID']>=2]

#now for each unique location and year I count how many appointments with multiple people they
had

appointments_by_cced_id_and_year_group=appointments_multiple_people.groupby(['cced_id',
'year_group'])['PersonID'].count().reset_index(name='mult_appointments')
```

```
#creating a mapping dataframe for mapping unique location ids -- cced_id to the cities they belong
to

mapping = pop_not_na.drop_duplicates(

  subset = ['cced_id', 'C_ID'],

  keep = 'last').reset_index(drop = True)

#merging mapping for cced_ids to C_IDs with appointments dataframe

appo_with_town=pd.merge(appointments_by_cced_id_and_year_group,
mapping[["C_ID","cced_id"]], how="left", left_on=["cced_id"], right_on=["cced_id"])

#grouping appointments by city_id -- C_ID, and year group and summing multiple people
appointments

appo_year_group_and_city = appo_with_town.groupby(['C_ID',
'year_group'])['mult_appointments'].sum().reset_index(name='appointments')

#saving progress to protect from dead Kernel

appo_year_group_and_city.to_csv('appo_year_group_and_city.csv')

#removing duplicates from population dataframe, as we need for final dataset data to be grouped
by year group and city id

pop_city = pop_not_na.drop_duplicates(

  subset = ['C_ID','latitude','longitude','Population_clean','year_group'],

  keep = 'last').reset_index(drop = True)

#merging the population dataframe with appintments dataframe with left join, as previously I filled
missing values

#for population dataframe with averages of neighbor years for each location that had a missing
value in population column

appo_with_pop=pd.merge(pop_city, appo_year_group_and_city, how="left", left_on=["C_ID",
"year_group"], right_on=["C_ID", "year_group"])

#saving progress to protect from dead Kernel

appo_with_pop.to_csv('appo_with_pop.csv')

appo_with_pop=pd.read_csv('appo_with_pop.csv')

#sorting the final merged dataframe by city id and year group

appo_with_pop_sort = appo_with_pop.sort_values(by=['C_ID', 'year_group'])

print (appo_with_pop_sort.head())

#removing missing values from final dataframe

appo_with_pop_sort_not_na=appo_with_pop_sort[appo_with_pop_sort['appointments'].notna()]

#saving dataframe for visation of relationship between population and appointments
```

```
for_rel_graph=appo_with_pop_sort_not_na[['C_ID','year_group','Population_clean','appointments']]
]

for_rel_graph.to_csv('for_rel_graph.csv')

#correlation between population and appointments

cormat = appo_with_pop_sort_not_na[['Population_clean','appointments']].corr()

#the correlation is positive, but very small

#let's look at annual correlation between population and appointments

print(cormat)

#annual correlation between population and appointments

annual_cormat =
appo_with_pop_sort_not_na.groupby("year_group")[['Population_clean','appointments']].corr()

#with time correlation changes from negative to positive

print(annual_cormat)

#creating a list with all city ids

C_ID_list=list(appo_with_pop_sort_not_na['C_ID'])

#creating a list with all year groups

year_group_list=list(appo_with_pop_sort_not_na['year_group'])

#creating a list of city ids list and year group ids

arrays = [C_ID_list, year_group_list]

#constructing a new index -- MultiIndex consisting of city id and year group id in a tuple for each row
in the dataframe

new_index=pd.MultiIndex.from_arrays(arrays, names=('entity', 'time'))

#assigning new -- MultiIndex to dataframe

appo_multi=appo_with_pop_sort_not_na.set_index(new_index)

print(appo_multi.columns)

#creating a dataframe with variables of interest

df=appo_multi[['Population_clean','appointments']]

#I constructed the following panel

print(df)

#importing a library for building linear model for the panel

from linearmodels.panel import PanelOLS

#specifying the linear model, including the C_ID fixed effects
```

```
mod = PanelOLS(df.Population_clean, df.appointments, entity_effects=True)

#clustering the C_ID fixed effects

res = mod.fit(cov_type='clustered', cluster_entity=True)

print(res)

#importing a library for building linear model for the panel

from linearmodels.panel import PanelOLS

#specifying the linear model, including the C_ID fixed effects

mod = PanelOLS(df.Population_clean, df.appointments, entity_effects=True)

#not-clustering the C_ID fixed effects

res = mod.fit()

print(res)

# for the first model --with clustering city FE, the coefficient infront of appointments is statistically significant at less then 10 percent level

# and for the second model without clustering city fixed effects, appointments coefficient has p-value=0.0003

#which means that coefficient infront of appointments is

#statistically significant on less then 1 percent level

#adding natural logarithm of population to dataframe

appo_with_pop_sort_not_na['ln_pop']=np.log2(appo_with_pop_sort_not_na['Population_clean'])

#adding natural logarithm of appointments to dataframe

appo_with_pop_sort_not_na['ln_app']=np.log2(appo_with_pop_sort_not_na['appointments'])

#assigning MultiIndex of tuples C_ID and year group to dataframe

log_multi=appo_with_pop_sort_not_na.set_index(new_index)

print(log_multi)

#creating a dataframe with variables of interest

df_log=log_multi[['ln_pop','ln_app']]

from linearmodels.panel import PanelOLS

#specifying the linear model in logarithms, including the C_ID fixed effects

mod = PanelOLS(df_log.ln_pop, df_log.ln_app, entity_effects=True)

#clustering C_ID fixed effects

res = mod.fit(cov_type='clustered', cluster_entity=True)

print(res)
```

```
from linearmodels.panel import PanelOLS

#specifying the linear model in logarithms, including the C_ID fixed effects

mod = PanelOLS(df_log.ln_pop, df_log.ln_app, entity_effects=True)

#not-clustering C_ID fixed effects

res = mod.fit()

print(res)


# for both of the runs of model in logarithms --with clustering FE, and without clustering city fixed effects

#appointments coefficient has p-value=0 which means that coefficient infront of ln_appointments is

#statistically significant on less then 1 percent level

#importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

from sklearn.model_selection import GridSearchCV

#dropping rows with NaN values in columns of interest

appo_with_pop_sort_not_na1=appo_with_pop_sort_not_na.dropna(subset=['year_group','appointments','Population_clean'])

#getting dummi-variables for each year group

df_train=pd.get_dummies(appo_with_pop_sort_not_na1, columns=['year_group'])

df_train.info()

#changing type of variables of interest from string to integer

df_train['Population_clean']=df_train['Population_clean'].astype(int)

df_train['appointments']=df_train['appointments'].astype(int)

#prepairing the dataframe with X -- dependable variables
```

```
for_x=df_train[['year_group_1575',
'year_group_1600','year_group_1625','year_group_1650','year_group_1675','year_group_1700','year_group_1725','year_group_1750','year_group_1775','year_group_1800','year_group_1825','appointments']]

#assigning dependable variables and output variable

X=for_x

y=df_train.Population_clean

#splitting X and y for train and test datasets with proportion 75 o 25 percent, applying shuffling

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42,

                                    shuffle=True)

#fitting Random Forest Regression to the Training set

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)

#fiting the model

regressor.fit(X_train, y_train)

# Predicting the Test set results for Random Forest Regression

y_pred = regressor.predict(X_test)

# Evaluating the Random Forest Regression algorithm

from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

from sklearn.linear_model import LinearRegression

#building a model for linear regression prediction

regressor = LinearRegression()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the linear regression algorithm

from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
from sklearn import linear_model

#building a model for Lasso regression prediction

regressor = linear_model.Lasso(alpha=0.1, normalize=True)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the Lasso regression algorithm

from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

#building a model for LassoLars regression prediction

regressor = linear_model.LassoLars(alpha=0.1, normalize=True)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the LassoLars regression algorithm

from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

# I checked prediction MAE, MSE, and RSME for 4 regressor algorithms: Linear Model, RandomForest
Regressor, Lasso, and LassoLars

# The best result got LassoLars model

# on the second place Lasso model

# on the third place is Linear model

# and the last is RandomForest Regressor model

#the rating is based on MAE, MSE, and RSME for each model

print(df.appointments.sum())

print(df.appointments.mean())

#importing library for visualization

library(ggplot2)
```

```
# if you want to run this code on your machine please change the file path

df <- rio::import('C:/Users/ianaf/Downloads/for_rel_graph.csv')

ggplot(data = df, aes(x = appointments,y = Population_clean))+
  geom_point()+
  geom_smooth()
#filtering appointments with 95 percent quantile
df_filtered <- df[df$appointments<= quantile(df$appointments, 0.95),]
#plotting again
ggplot(data = df_filtered, aes(x = appointments,y = Population_clean))+
  geom_point()+
  geom_smooth()
ggplot(data = df_filtered, aes(x = appointments,y = Population_clean))+
  geom_boxplot(aes(group=appointments))+
  geom_smooth()+
  scale_y_log10()+
  ylab("Population, log scale")


ggplot(data = df_filtered, aes(x = appointments,y = Population_clean))+
  geom_boxplot(aes(group=appointments))+
  geom_smooth()+
  scale_y_log10()+
  ylab("Population, log scale")+
  facet_wrap(vars(year_group))
# on all the graphs the relationship is barely visible data_analysis
```

Project: Analying the relationship between English church appointments and population of settlements in 1575-1800.

The goal of the project is to analyze the relationship between church appointments and population, to see if any conclutions about significance of relatonship can be made, to provide ideas for extension of the project and discussion of how the results of this or extended project can be helpful for other historical data based projects/researches, or even for researches that are based on the recent observations.

The H0 hypothesis: The English church appointments effected on population of English settlements in 1575-1800.

The time frame is 1575-1800 and has a step of 25 years. The databease has data till 1835, but after performing grouping, cleaning and removing missing values, I had only time-frame available from 1575-1800.

Data Source: The Clergy DataBase: https://theclergydatabase.org.uk/. The data base stores Enlish church appointments for each unique location that is indicated as CCED_id and year of the appintment. Appointments are differentiated by the Type. For example: Induction Mandate, Death, Marriage, Nomination, etc. For each appointment Type, year and unique location id -- CCED_id there is a person id and name.

Data: The final dataset included observations for 970 cities that totaled 45777 appoinments (with multiple people present). Average count of appointments for the cities from final data set per year is 10.2.

Appointments: I decided to count in only appointments where multiple people were present (at least two)

As having an appointment with only one person, would not affect the population, because besides the attendee and priest (or any other person that serves in church) there are no other people for the attendee to meet. I don't count as a meeting -- meeting the person who serves in church, as these attendee meets the church service people multiple times, and they are not unique with every appointment in the church.

Population: The initial dataset with population has issing values for some cities for some periods. I filled the missing values (as much as possible) for each city that had a missing papulation observation with an average value of population between the preceding period and later period -- for which data was available.

How: by collecting population data and church appintments data I constructed the dataset that includes (for each city, for each year) population and number of appintments that took place. Using the dataset I calculated the point estimates for two model specifications:

1. Population ~ appointments + City_Fixed_Effects

2. ln(Population) ~ ln(appointments) + City_Fixed_Effects

for each specification I run two models: with city fixed effects clustering and without.

To see the results of the estimation, please read the paper in this repository.

With the next step I tested four regression methods from scikit-learn library: Linear Regressor, Random Forest Regressor, Lasso regressor, and LassoLars regressor.

I compared the outputs of these four regression models to each other using Mean Absolute Error (MAE), Mean Squared Error(MSE), and Root Mean Squared Error (RSME) -- that are normally used for scoring regression models.

To see the results of comparison between 4 scikit-learn regression models, please read the paper in this repository.

How to run code from the repository:

1. Please first run the file with name data_analysis_Iana_Fedorchuk.ipynb. It includes code and comments for manipulating the data and constructing the final dataset, as well as, panel regression outputs description and results discussion. In the end of the file you will find te section with scikit-learn regressor models, the code for splitting data to to test and train, specifications and the results of fitting those models. It also includes a small discussion for comparing the results.

IMPORTANT! When I started working on the project, I asked our professor, Mathias Bühler, for the initial HTML files parsed from https://theclergydatabase.org.uk/ as the page was down. Then I applied the code for extracting the elements of the HTML files that was provided for us, with some minor changes.

2. For getting the plot for relationship between population and appointments please run data_analysis_Iana_Fedorchuk_visualization.r code. As an input file for visualization please use for_rel_graph.csv.

Please find the paper in the current repository with file name data_analysis_Iana_Fedorchuk.pdf

The file that includes code with comments, README, and paper please find also in this repository. Or in the folder where you found the README. The file name is characters_Iana_Fedorchuk

If you would like to see the whole GitHub repository, please click here https://github.com/yanafedorchuk/data_analysis