project

October 17, 2025

[1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
```

[2]:
```
df_demo_raw = pd.read_csv('NCHS_-_Death_rates_and_life_expectancy_at_birth.
 ↪csv', encoding='latin-1')
df_geo_raw = pd.read_csv('U.S.
 ↪_Life_Expectancy_at_Birth_by_State_and_Census_Tract_-_2010-2015.csv',␣
 ↪encoding='latin-1')
```

[3]:
```
df_demo_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1071 entries, 0 to 1070
Data columns (total 5 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Year                            1071 non-null   int64
 1   Race                            1071 non-null   object
 2   Sex                             1071 non-null   object
 3   Average Life Expectancy (Years)  1065 non-null   float64
 4   Age-adjusted Death Rate         1071 non-null   float64
dtypes: float64(2), int64(1), object(2)
memory usage: 42.0+ KB
```

[4]:
```
df_geo_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73121 entries, 0 to 73120
Data columns (total 6 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   State                       73121 non-null  object
 1   County                      73121 non-null  object
 2   CensusTract                 73070 non-null  float64
```

```
 3   LifeExpectancy               67199 non-null  float64
 4   LifeExpectancyRange          67199 non-null  object
 5   LifeExpectancyStandardError  67199 non-null  float64
dtypes: float64(3), object(3)
memory usage: 3.3+ MB
```

[6]:
```
df_geo_raw.head(10)
```

[6]:

	State	County	CensusTract	LifeExpectancy	LifeExpectancyRange	LifeExpectancyStandardError
0	Alabama	(blank)	NaN	75.5	75.2-77.5	0.0328
1	Alabama	Autauga County, AL	201.00	73.1	56.9-75.1	2.2348
2	Alabama	Autauga County, AL	202.00	76.9	75.2-77.5	3.3453
3	Alabama	Autauga County, AL	203.00	NaN	NaN	NaN
4	Alabama	Autauga County, AL	204.00	75.4	75.2-77.5	1.0216
5	Alabama	Autauga County, AL	205.00	79.4	77.6-79.5	1.1768
6	Alabama	Autauga County, AL	206.00	73.1	56.9-75.1	1.5519
7	Alabama	Autauga County, AL	207.00	NaN	NaN	NaN
8	Alabama	Autauga County, AL	208.01	78.3	77.6-79.5	2.3861
9	Alabama	Autauga County, AL	208.02	76.9	75.2-77.5	1.2628

[8]:
```
#get rid of lines like Alabama        (blank)        NaN        75.5        75.
 ↪2-77.5        0.0328
df_geo_cleaned = df_geo_raw[~df_geo_raw.eq('(blank)').any(axis=1)]
```

[9]:
```
#drop CensusTract as will only use counties as well as life expectancy range
df_geo_update = df_geo_cleaned.drop(columns=['CensusTract',␣
 ↪'LifeExpectancyRange'])
```

[14]:
```
#Check for null values
print(df_geo_update.head(10))
print(df_geo_update.isnull().sum().sum())
print(df_geo_update.isnull().any())
```

	State	County	LifeExpectancy	LifeExpectancyStandardError
1	Alabama	Autauga County, AL	73.1	2.2348
2	Alabama	Autauga County, AL	76.9	3.3453

```
3   Alabama  Autauga County, AL              NaN                          NaN
4   Alabama  Autauga County, AL             75.4                       1.0216
5   Alabama  Autauga County, AL             79.4                       1.1768
6   Alabama  Autauga County, AL             73.1                       1.5519
7   Alabama  Autauga County, AL              NaN                          NaN
8   Alabama  Autauga County, AL             78.3                       2.3861
9   Alabama  Autauga County, AL             76.9                       1.2628
10  Alabama  Autauga County, AL             73.9                       1.5923
11844
State                          False
County                         False
LifeExpectancy                  True
LifeExpectancyStandardError     True
dtype: bool
```

[15]:
```
#null values are only in floating point LifeExpectancy and␣
 ↪LifeExpectancyStandardError. Some simple analysis:
df_geo_update.describe().T
```

[15]:
```
                               count       mean       std      min      25%  \
LifeExpectancy               67148.0  78.309467  3.990279  56.9000  75.8000
LifeExpectancyStandardError  67148.0   1.855832  0.651331   0.4588   1.3797

                                50%        75%      max
LifeExpectancy              78.5000  81.000000  97.5000
LifeExpectancyStandardError  1.7067   2.186825   3.9998
```

[16]:
```
#Will replace the null values with mean in both columns
df_geo_update['LifeExpectancy'] = df_geo_update['LifeExpectancy'].
 ↪fillna(df_geo_update['LifeExpectancy'].mean())
df_geo_update['LifeExpectancyStandardError'] =␣
 ↪df_geo_update['LifeExpectancyStandardError'].
 ↪fillna(df_geo_update['LifeExpectancyStandardError'].mean())
df_geo_update.describe().T
```

[16]:
```
                               count       mean       std      min      25%  \
LifeExpectancy               73070.0  78.309467  3.825163  56.9000  76.100
LifeExpectancyStandardError  73070.0   1.855832  0.624379   0.4588   1.408

                                  50%      75%      max
LifeExpectancy              78.309467  80.8000  97.5000
LifeExpectancyStandardError  1.773550   2.1291   3.9998
```

[17]:
```
#Will strip the ",state" from "county,state" in county table
df_geo_update['County'] = df_geo_update['County'].str.split(',').str[0].str.
 ↪strip()
```

[19]:
```
print(df_geo_update.head(5))
print(df_geo_update.tail(5))
```

```
     State          County  LifeExpectancy  LifeExpectancyStandardError
1  Alabama  Autauga County       73.100000                     2.234800
2  Alabama  Autauga County       76.900000                     3.345300
3  Alabama  Autauga County       78.309467                     1.855832
4  Alabama  Autauga County       75.400000                     1.021600
5  Alabama  Autauga County       79.400000                     1.176800
         State           County  LifeExpectancy  LifeExpectancyStandardError
73116  Wyoming  Washakie County            80.1                       2.6916
73117  Wyoming  Washakie County            79.9                       2.8024
73118  Wyoming  Washakie County            81.8                       2.0776
73119  Wyoming    Weston County            79.0                       1.0697
73120  Wyoming    Weston County            78.6                       1.6093
```

[21]:
```
# Group by State and County, then average the numeric columns
df_geo = (
    df_geo_update.groupby(["State", "County"], as_index=False)
      .agg({
          "LifeExpectancy": "mean",
          "LifeExpectancyStandardError": "mean"
      })
)
df_geo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3144 entries, 0 to 3143
Data columns (total 4 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   State                        3144 non-null   object
 1   County                       3144 non-null   object
 2   LifeExpectancy               3144 non-null   float64
 3   LifeExpectancyStandardError  3144 non-null   float64
dtypes: float64(2), object(2)
memory usage: 98.4+ KB
```

[28]:
```
df_geo.to_csv('life_expectancy_by_geography.csv', index=False) # store cleaned␣
 ↪data
```

[24]:
```
#Life expectacy by geography dataset cleaned. Examine life expectacy by␣
 ↪demography dataset which needs less cleaning
print(df_demo_raw.head())
print(df_demo_raw.info())
print(df_demo_raw.describe().T)
```

```
   Year       Race         Sex  Average Life Expectancy (Years)  \
0  1900  All Races  Both Sexes                             47.3
```

```
1  1901  All Races  Both Sexes                              49.1
2  1902  All Races  Both Sexes                              51.5
3  1903  All Races  Both Sexes                              50.5
4  1904  All Races  Both Sexes                              47.6

   Age-adjusted Death Rate
0                   2518.0
1                   2473.1
2                   2301.3
3                   2379.0
4                   2502.5
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1071 entries, 0 to 1070
Data columns (total 5 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Year                              1071 non-null   int64
 1   Race                              1071 non-null   object
 2   Sex                               1071 non-null   object
 3   Average Life Expectancy (Years)   1065 non-null   float64
 4   Age-adjusted Death Rate           1071 non-null   float64
dtypes: float64(2), int64(1), object(2)
memory usage: 42.0+ KB
None
                                  count         mean         std     min  \
Year                             1071.0  1959.000000   34.367176  1900.0
Average Life Expectancy (Years)  1065.0    64.500188   11.843765    29.1
Age-adjusted Death Rate          1071.0  1593.061625  682.369379   611.3

                                     25%     50%      75%     max
Year                             1929.00  1959.0  1989.00  2018.0
Average Life Expectancy (Years)    57.10    66.8    73.90    81.4
Age-adjusted Death Rate          1012.95  1513.7  2057.15  3845.7
```

[25]:
```
#check for null values and where they appear
print(df_demo_raw.isnull().sum().any())
print(df_demo_raw.isnull().sum().sum())
print(df_demo_raw.isna().sum())
```

```
True
6
Year                              0
Race                              0
Sex                               0
Average Life Expectancy (Years)   6
Age-adjusted Death Rate           0
dtype: int64
```

[27]:
```
#all in Average Life Expectancy (Years) Column. Rename column and fill the NAs␣
 ↪with mean
df_demo = df_demo_raw.rename(columns={'Average Life Expectancy (Years)':␣
 ↪'LifeExpectancy', 'Age-adjusted Death Rate': 'AgeAdjustedDeathRate'})
df_demo['LifeExpectancy'] = df_demo['LifeExpectancy'].
 ↪fillna(df_demo['LifeExpectancy'].mean())
print(df_demo.info())
print(df_demo.describe().T)
print(df_demo.isna().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1071 entries, 0 to 1070
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Year                  1071 non-null   int64
 1   Race                  1071 non-null   object
 2   Sex                   1071 non-null   object
 3   LifeExpectancy        1071 non-null   float64
 4   AgeAdjustedDeathRate  1071 non-null   float64
dtypes: float64(2), int64(1), object(2)
memory usage: 42.0+ KB
None
                       count         mean         std     min      25%  \
Year                  1071.0  1959.000000   34.367176  1900.0  1929.00
LifeExpectancy        1071.0    64.500188   11.810512    29.1    57.15
AgeAdjustedDeathRate  1071.0  1593.061625  682.369379   611.3  1012.95

                         50%      75%     max
Year                  1959.0  1989.00  2018.0
LifeExpectancy          66.8    73.90    81.4
AgeAdjustedDeathRate  1513.7  2057.15  3845.7
Year                       0
Race                       0
Sex                        0
LifeExpectancy             0
AgeAdjustedDeathRate       0
dtype: int64
```

[29]:
```
#store cleaned demo data
df_demo.to_csv('life_expectancy_by_demography.csv', index=False)
```

[30]:
```
#we now have life expectancy data by demography and by geography. The data␣
 ↪varies too much to be combined into a single dataset
#solution: train and model each dataset and predict the outcome independently.␣
 ↪Average the two for a final prediction
```

[32]:
```
#start with geography. Issue: a lot of counties have common names in different␣
 ↪states which can skew the analysis.
#while it was necessary to treat them separately when cleaning the data, they␣
 ↪will be rejoined for analysis
df_geo['State_County'] = df_geo['State'] + '|' + df_geo['County']
```

[38]:
```
#prepare for training model
X_geo = ['State_County', 'LifeExpectancyStandardError']
y_geo = 'LifeExpectancy'
```

[39]:
```
#two part preprocessing
#part 1: encoding categorical State_County data and scaling␣
 ↪LifeExpectancyStandardError
#part 2: User will not enter LifeExpectancyStandardError. Will statistically␣
 ↪impute it and all other missing values
#State_County: categorical → impute most frequent → one-hot encode
#LifeExpectancyStandardError: numeric → impute median → scale
#https://www.geeksforgeeks.org/machine-learning/
 ↪what-is-exactly-sklearnpipelinepipeline/
geo_preprocessor = ColumnTransformer(transformers=[
    ('statecounty', Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ]), ['State_County']),
    ('stderr', Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]), ['LifeExpectancyStandardError'])
])
```

[40]:
```
# build model for predicting life expectancy based on geographic data
geo_model = Pipeline(steps=[
    ('preprocessor', geo_preprocessor),
    ('regressor', LinearRegression())
])
```

[41]:
```
# Train geographic life expectancy prediction model
geo_model.fit(df_geo[X_geo], df_geo[y_geo])
```

[41]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('statecounty',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['State_County']),
                                                 ('stderr',
```

```
                                                    Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                    ('scaler',
StandardScaler())]),
['LifeExpectancyStandardError'])])),
                ('regressor', LinearRegression())])
```

[42]:
```
#Repeat process with demography data
X_demo = ['Year', 'Race', 'Sex', 'AgeAdjustedDeathRate']
y_demo   = 'LifeExpectancy'
```

[43]:
```
# demography data preprocessing
#
#part 1: encoding categorical Race and Sex data and scaling year (birth year)␣
 ↪and death rate data
#part 2: User will not enter death rate. Will statistically impute it and all␣
 ↪other missing values
#Year, AgeAdjustedDeathRate: numeric → impute median → scale
#Race, Sex: categorical → impute most frequent → one-hot encode
demo_preprocessor = ColumnTransformer(transformers=[
    ('year', Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]), ['Year']),
    ('race', Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ]), ['Race']),
    ('sex', Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ]), ['Sex']),
    ('deathrate', Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]), ['AgeAdjustedDeathRate'])
])
```

[44]:
```
# build model for predicting life expectancy based on demographic data
demo_model = Pipeline(steps=[
    ('preprocessor', demo_preprocessor),
    ('regressor', LinearRegression())
])
```

[45]:
```
#Train demography model
demo_model.fit(df_demo[X_demo], df_demo[y_demo])
```

[45]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('year',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['Year']),
                                                 ('race',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Race']),
                                                 ('sex',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Sex']),
                                                 ('deathrate',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['AgeAdjustedDeathRate'])])),
                ('regressor', LinearRegression())])
```

[51]:
```
#Will be user-entered through web page
state = ["Idaho","Ohio", "Missouri"]
county = ["Elmore County","Allen County","Cass County"]
state_county = [f"{s}|{c}" for s, c in zip(state, county)]
year = [1999, 1942, 1970]
race = ["White", "All Races", "Black"]
sex = ["Female", "Both Sexes", "Male"]
#
geo_input = [
    pd.DataFrame([{'State_County': sc, 'LifeExpectancyStandardError': np.nan}])
    for sc in state_county
]
#
demo_input = [
    pd.DataFrame([{'Year': y, 'Race': r, 'Sex': s, 'AgeAdjustedDeathRate': np.
 ↪nan}])
    for y, r, s in zip(year, race, sex)
]
```

[52]:
```
#Predictions
p_geo = [geo_model.predict(df)[0] for df in geo_input]
p_demo = [demo_model.predict(df)[0] for df in demo_input]
```

[53]:
```
#Life expectancy predictions
life_expectancy = []
for g, d in zip(p_geo, p_demo):
    life_expectancy.append((g + d) / 2)
print(life_expectancy)
```

```
[np.float64(71.3283735113576), np.float64(71.68597236774232),
np.float64(72.20068775855448)]
```

[ ]: