Population Prediction

January 30, 2022

Luis Garduno

Dataset: **International Database (IDB)**

Question Of Interest: Predict the population of earth in 2122.

1 Data Understanding

1.1 Data Description

[1]:
```
import numpy as np
import pandas as pd

# Load dataset into dataframe
df = pd.read_csv('https://raw.githubusercontent.com/luisegarduno/
↪MachineLearning_Projects/master/data/idb5yr.all', delimiter='|',␣
↪encoding='ISO-8859-1')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34237 entries, 0 to 34236
Data columns (total 99 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   #YR        34237 non-null  int64
 1   TFR        26163 non-null  float64
 2   SRB        26163 non-null  float64
 3   RNI        26172 non-null  float64
 4   POP95_99   26171 non-null  float64
 5   POP90_94   26171 non-null  float64
 6   POP85_89   26171 non-null  float64
 7   POP80_84   26171 non-null  float64
 8   POP75_79   26171 non-null  float64
 9   POP70_74   26171 non-null  float64
 10  POP65_69   26171 non-null  float64
 11  POP60_64   26171 non-null  float64
 12  POP5_9     26171 non-null  float64
 13  POP55_59   26171 non-null  float64
```

```
14  POP50_54   26171 non-null  float64
15  POP45_49   26171 non-null  float64
16  POP40_44   26171 non-null  float64
17  POP35_39   26171 non-null  float64
18  POP30_34   26171 non-null  float64
19  POP25_29   26171 non-null  float64
20  POP20_24   26171 non-null  float64
21  POP15_19   26171 non-null  float64
22  POP10_14   26171 non-null  float64
23  POP100_    26171 non-null  float64
24  POP0_4     26171 non-null  float64
25  POP        34237 non-null  int64
26  NMR        26172 non-null  float64
27  NAME       34237 non-null  object
28  MR1_4      26163 non-null  float64
29  MR0_4      26163 non-null  float64
30  MPOP95_99  26171 non-null  float64
31  MPOP90_94  26171 non-null  float64
32  MPOP85_89  26171 non-null  float64
33  MPOP80_84  26171 non-null  float64
34  MPOP75_79  26171 non-null  float64
35  MPOP70_74  26171 non-null  float64
36  MPOP65_69  26171 non-null  float64
37  MPOP60_64  26171 non-null  float64
38  MPOP5_9    26171 non-null  float64
39  MPOP55_59  26171 non-null  float64
40  MPOP50_54  26171 non-null  float64
41  MPOP45_49  26171 non-null  float64
42  MPOP40_44  26171 non-null  float64
43  MPOP35_39  26171 non-null  float64
44  MPOP30_34  26171 non-null  float64
45  MPOP25_29  26171 non-null  float64
46  MPOP20_24  26171 non-null  float64
47  MPOP15_19  26171 non-null  float64
48  MPOP10_14  26171 non-null  float64
49  MPOP100_   26171 non-null  float64
50  MPOP0_4    26171 non-null  float64
51  MPOP       26171 non-null  float64
52  MMR1_4     26163 non-null  float64
53  MMR0_4     26163 non-null  float64
54  IMR_M      26163 non-null  float64
55  IMR_F      26163 non-null  float64
56  IMR        26163 non-null  float64
57  GRR        26163 non-null  float64
58  GR         26172 non-null  float64
59  FPOP95_99  26171 non-null  float64
60  FPOP90_94  26171 non-null  float64
61  FPOP85_89  26171 non-null  float64
```

```
 62  FPOP80_84  26171 non-null  float64
 63  FPOP75_79  26171 non-null  float64
 64  FPOP70_74  26171 non-null  float64
 65  FPOP65_69  26171 non-null  float64
 66  FPOP60_64  26171 non-null  float64
 67  FPOP5_9    26171 non-null  float64
 68  FPOP55_59  26171 non-null  float64
 69  FPOP50_54  26171 non-null  float64
 70  FPOP45_49  26171 non-null  float64
 71  FPOP40_44  26171 non-null  float64
 72  FPOP35_39  26171 non-null  float64
 73  FPOP30_34  26171 non-null  float64
 74  FPOP25_29  26171 non-null  float64
 75  FPOP20_24  26171 non-null  float64
 76  FPOP15_19  26171 non-null  float64
 77  FPOP10_14  26171 non-null  float64
 78  FPOP100_   26171 non-null  float64
 79  FPOP0_4    26171 non-null  float64
 80  FPOP       26171 non-null  float64
 81  FMR1_4     26163 non-null  float64
 82  FMR0_4     26163 non-null  float64
 83  GENC       34086 non-null  object
 84  FIPS       34237 non-null  object
 85  E0_M       26163 non-null  float64
 86  E0_F       26163 non-null  float64
 87  E0         26163 non-null  float64
 88  CDR        26172 non-null  float64
 89  CBR        26172 non-null  float64
 90  ASFR45_49  26173 non-null  float64
 91  ASFR40_44  26173 non-null  float64
 92  ASFR35_39  26173 non-null  float64
 93  ASFR30_34  26173 non-null  float64
 94  ASFR25_29  26173 non-null  float64
 95  ASFR20_24  26173 non-null  float64
 96  ASFR15_19  26173 non-null  float64
 97  AREA_KM2   34237 non-null  int64
 98  POP_DENS   34237 non-null  float64
dtypes: float64(93), int64(3), object(3)
memory usage: 25.9+ MB
```

[2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Make year column easier to understand
df.rename(columns={'#YR':'YEAR'}, inplace=True)
```

```python
# Remove every column except for year & population
for col in df.columns.values:
    if col != 'YEAR' and col != 'POP':
        df.drop(col, axis=1, inplace=True)

df.describe()
```

[2]:

	YEAR	POP
count	34237.000000	3.423700e+04
mean	2024.935158	3.306017e+07
std	43.571971	1.253471e+08
min	1950.000000	2.028000e+03
25%	1987.000000	4.304230e+05
50%	2025.000000	4.831844e+06
75%	2063.000000	2.070922e+07
max	2100.000000	1.647894e+09

1.2 Data Quality

[3]:
```python
import missingno as mn

mn.matrix(df)

# Count unique values in column 'gameId' of the dataframe
print('Number of unique values in column "YEAR" : ', df['YEAR'].nunique())
```

```
Number of unique values in column "YEAR" :  151
```

1.3 Clearning the Dataset

[4]:
```python
# Group by year & get sum
df_yr = df.groupby(by='YEAR')
df_yr = df_yr['POP'].sum()

# Create a new dataframe with new data (1951 - 2100)
pop_sum = []
for i in range(1951, 2023):
    pop_sum.append(df_yr[i])
df_pop = pd.DataFrame({'YEAR': list(range(1951, 2023)), 'POP': pop_sum})
df = df_pop

print(f'\n--> Current Population (2021): {df["POP"][70]:,d}\n')
df.tail(5)
```

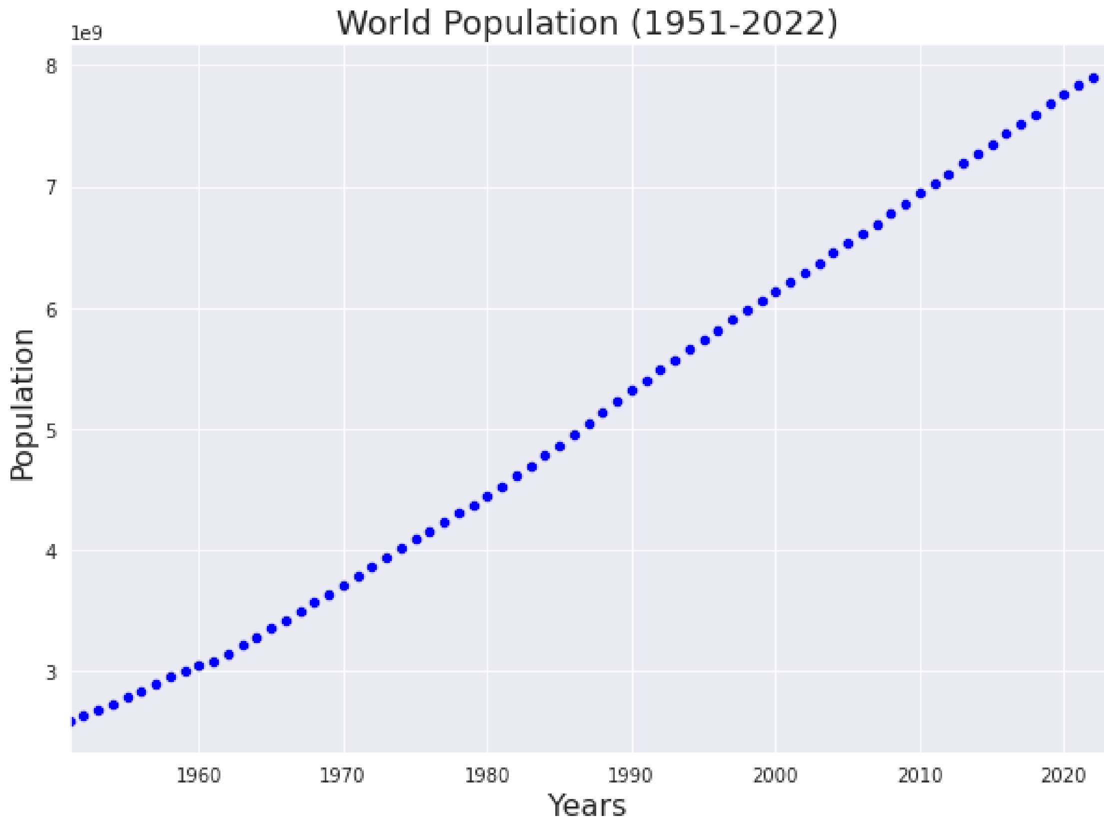
```
--> Current Population (2021): 7,831,718,605
```

[4]:

	YEAR	POP
67	2018	7597066210
68	2019	7676686052
69	2020	7756873419
70	2021	7831718605
71	2022	7905336896

[5]:
```python
sns.set_style("darkgrid")
plt.subplots(figsize=(10,7))
ax = sns.scatterplot(data=df, x='YEAR', y='POP', color='blue')
ax.set_xlabel('Years', fontsize=16)
ax.set_ylabel('Population', fontsize=16)
ax.set_title('World Population (1951-2022)', fontsize=18)
plt.xlim(1951, 2023)

plt.show()
```



[6]:
```
# Define X & Y
if 'POP' in df_pop:
    y = df_pop['POP'].values
    del df_pop['POP']
    X = df_pop.to_numpy()
```

---

2 Modeling

Derived the formula for calculating the optimal values of the regression weights:

$$w = (X^T X)^{-1} X^T y$$

where X is the matrix of values with a bias column of ones appended onto it. For the population dataset one could construct this X matrix by stacking a column of ones onto the `df_pop.YEAR` matrix.

$$X = \begin{bmatrix} & \vdots & & 1 \\ \cdots & \text{ds.data} & \cdots & \vdots \\ & \vdots & & 1 \end{bmatrix}$$

[7]:
```
# Create a matrix full of ones & stack 2 matrices horizontally
X = np.hstack((np.ones((len(X), 1)), X))

# Calculate optimal values of the regression weights
w = np.linalg.inv(X.T @ X) @ X.T @ y

print("\n++++++++++++++ WEIGHTS +++++++++++++++++\n", pd.DataFrame(data=w))
diff = np.round(( (y - (abs(np.dot(X,w) - y))) / y ) * 100, 2)
print("\n============= TARGET PERCENT ACCURACY ===============\n", pd.
↪DataFrame(data=diff))
```

```
++++++++++++++ WEIGHTS +++++++++++++++++
               0
0 -1.494551e+11
1  7.779070e+07

============= TARGET PERCENT ACCURACY ===============
        0
0   89.20
1   90.73
2   92.10
3   93.32
4   94.38
..    …
67  99.07
68  99.06
69  99.04
70  99.08
71  99.14

[72 rows x 1 columns]
```

To predict the output from our model, $\hat{y}$,from w and X we need to use

$\hat{y} = w^T X^T$, for row vector $\hat{y}$

[8]:
```
yHat_np = w.T @ X.T         # Shape : (1,72)
yHat_np = yHat_np.ravel()   # Shape : (72,)

MSE_np = (np.square(y - yHat_np)).mean()
```

```
print(f'MSE: {round(MSE_np):,d}')
```

MSE: 7,615,844,338,429,437

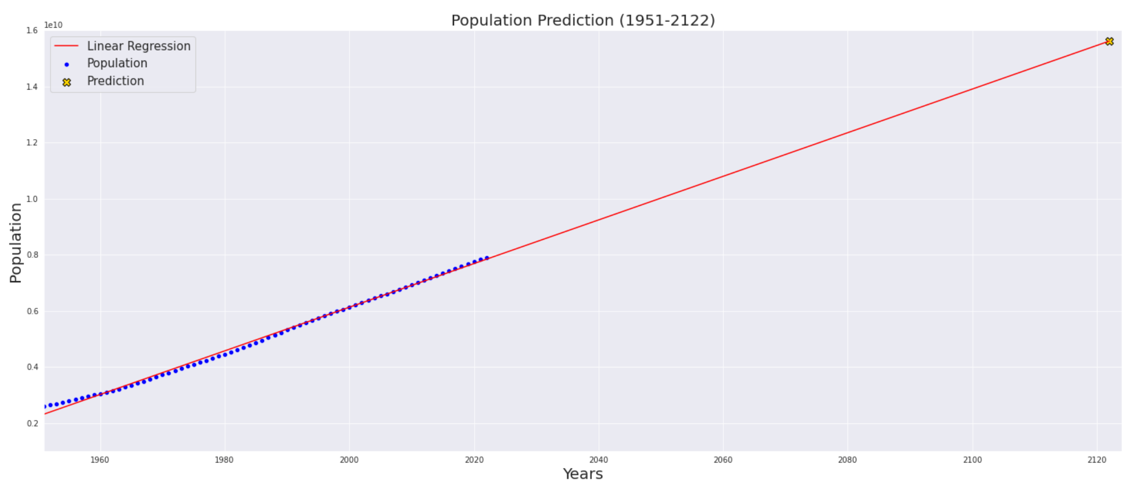
[9]:
```
X_new = np.array([[0], [2122]])
X_test = np.c_[np.ones((len(X_new), 1)), X_new]
y_test = X_test.dot(w)
print(f'\n--> ~Population (2122): {round(y_test[1]):,d}\n')

sns.set_style("darkgrid")
plt.subplots(figsize=(25,10))

plt.plot(X_new, y_test, "r-", color='red')
sns.scatterplot(data=df, x='YEAR', y=y, color='blue')
sns.scatterplot(x=X_new[1], y=y_test[1], s=100, marker="X", linewidth=1,␣
↪edgecolor='k', color='gold')

plt.xlabel('Years', fontsize=20)
plt.ylabel('Population', fontsize=20)
plt.title('Population Prediction (1951-2122)', fontsize=20)
plt.axis([1951, 2124, 1000000000, 16000000000])
plt.legend(["Linear Regression", "Population", "Prediction"], prop={'size': 15})
plt.show()
```

--> ~Population (2122): 15,616,786,279



3 Comparing Performance

[10]:
```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X, y)
MSE_sk = mean_squared_error(y, reg.predict(X))
y_testsk = reg.predict(X_test)

print("******** Linear Equation ********")
print("[Numpy] \th =", round(w[0],3), "* x + (" + str(round(w[1],5)) + ")")
print("[Sklearn]\th =", round(reg.intercept_,2), "* x + (" + str(round(reg.
↪coef_[1],5)) + ")\n")

print("******** Mean Squared Error ********")
print(f'[Numpy] \tMSE: {round(MSE_np):,d}')
print(f'[Sklearn]\tMSE: {round(MSE_sk):,d}\n')

print("******** Population Prediction - 2122 ********")
print(f'[Numpy] \t {round(y_test[1]):,d}')
print(f'[Sklearn]\t {round(y_testsk[1]):,d}\n')
```

```
******** Linear Equation ********
[Numpy]         h = -149455085136.29 * x + (77790702.83494)
[Sklearn]       h = -149455085136.39 * x + (77790702.83499)

******** Mean Squared Error ********
[Numpy]         MSE: 7,615,844,338,429,437
[Sklearn]       MSE: 7,615,844,338,429,399

******** Population Prediction - 2122 ********
[Numpy]          15,616,786,279
[Sklearn]        15,616,786,279
```

References Census. International Database (IDB). https://www.census.gov/data-tools/demo/idb/#/country?COUNTRY_YEAR=2022&COUNTRY_YR_ANIM=2022 (Accessed 01-22-2022)