Does a Country's Economic Freedom Level Influence the Fertility Rate of its Population

By: Christian Prather

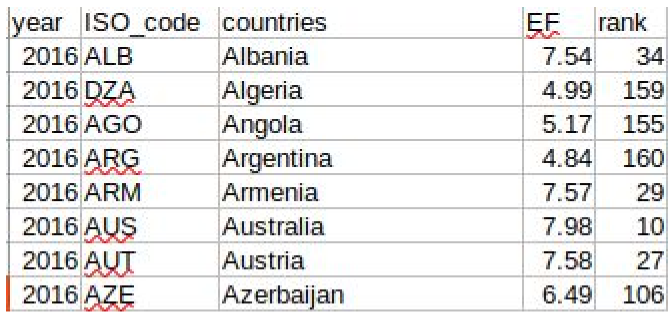
Data:

Data selection became a very critical and informative step in this project, I began not sure what I wanted to do beyond knowing that I wanted to see if I could use machine learning (ML) to find a correlation between two data sets that otherwise a human could not see easily. I began by shifting through datasets posted on a known set kaggel.com. Kaggel is a large repository of dataset able to be utilized by the public and often targeted as a starting point for many Machine Learning applications.

Looking through data for a while I discovered a dataset of economic freedom for countries across the world, this data contains per country data on a countries economic freedom index based on size of government, property rights, sound money, freedom to trade, and regulation. Each country had input for multiple years spanning from 1970 - 2016, it also contains a countries iso code. This would be important later as it would be a primary method to joining the two datasets I collected. This got me thinking that I wanted to see if I could correlate a country's “x” result from its economic health. I began looking through datasets filtering by any that had countries as a column so I could check for correlations per country. I eventually found a dataset for infant mortality, fertility, and a country's adjusted net national income indexed by iso codes, this would be perfect. I decided to look for a correlation between a country’s calculated economic freedom (EF) and its fertility. By conjecture you may guess that as EF rises so will fertility but I wanted a defined model to project this as you could then apply it to fluctuating economic states due to such events as pandemics, and natural disasters. The economic freedom has an unspecified license however as it is only being utilized in an educational project I felt it okay to use. The fertility dataset falls under CC BY-SA 4.0 license so it is good to use as is.

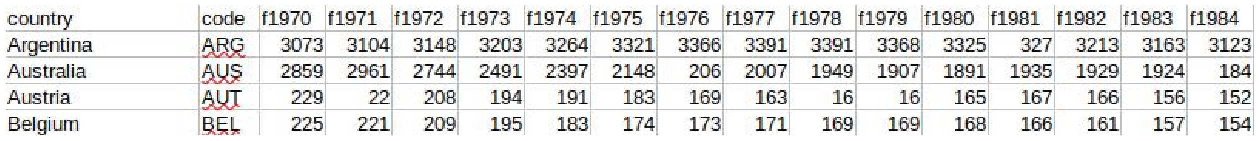
Parsing usable data from these sets was definitely not trivial. The economic data contained columns beyond just the iso and EF, I had to think intuitively as to what features would prove the most important. I filtered out multiple finally landing on country, iso and ef. For the fertility dataset its primary problem was in how it was formatted, each row contained a country, iso, and fertility, mortality, income, for each year spanning 1970 - 2016. This produced a row for each country containing around 40 columns when only (iso, fertility, income of a specified year) is needed.

Freedom:



year	ISO_code	countries	EF	rank
2016	ALB	Albania	7.54	34
2016	DZA	Algeria	4.99	159
2016	AGO	Angola	5.17	155
2016	ARG	Argentina	4.84	160
2016	ARM	Armenia	7.57	29
2016	AUS	Australia	7.98	10
2016	AUT	Austria	7.58	27
2016	AZE	Azerbaijan	6.49	106

Fertility:

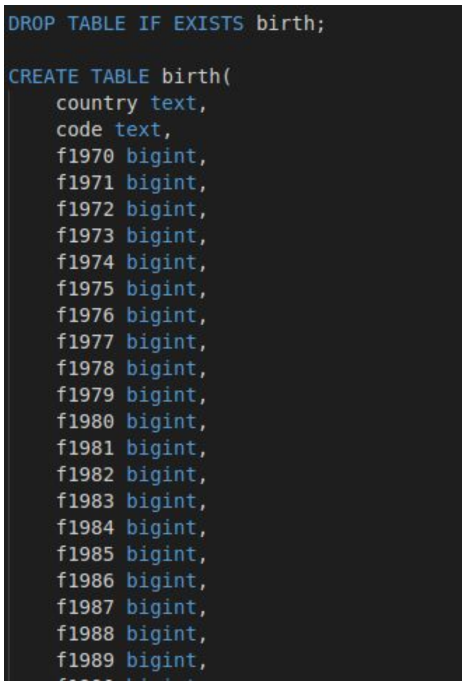


country	code	f1970	f1971	f1972	f1973	f1974	f1975	f1976	f1977	f1978	f1979	f1980	f1981	f1982	f1983	f1984
Argentina	ARG	3073	3104	3148	3203	3264	3321	3366	3391	3391	3368	3325	327	3213	3163	3123
Australia	AUS	2859	2961	2744	2491	2397	2148	206	2007	1949	1907	1891	1935	1929	1924	184
Austria	AUT	229	22	208	194	191	183	169	163	16	16	165	167	166	156	152
Belgium	BEL	225	221	209	195	183	174	173	171	169	169	168	166	161	157	154

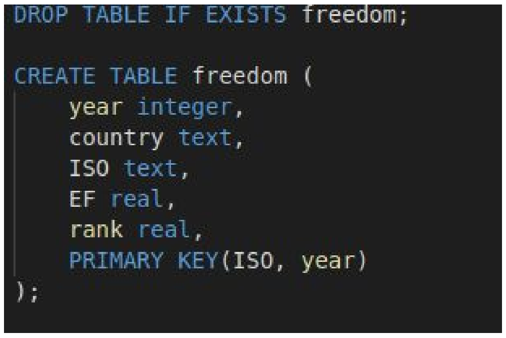
To clean the data I decided to write a basic python script to work directly on the csv files before they got loaded into the database. This script loaded the datasets from csv files into pandas dataframes and removed any Nan's / mortality data while saving back out to filtered csv files. See attached page for clean-up.py.

The database was then created with a basic sql script and using a copy command to load data into two distinct tables. One table for each dataset each with a birth with a primary key on iso codes and freedom with a primary key on year and iso as each country had multiple year entries. I choose not to set the data up in a format needed for ML prior to uploading rather than opting for preprocessing in the training script. This was done because I felt in practice it makes more sense to store the data in a db in a format that is more universal than targeted and allow the application to format the pulled data to a way it would need. Once loaded to the db I could begin the explore the ML aspect of the project.

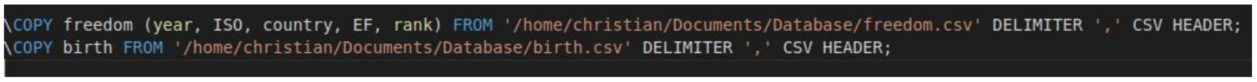
Data_load.sql:



```sql
DROP TABLE IF EXISTS birth;

CREATE TABLE birth(
    country text,
    code text,
    f1970 bigint,
    f1971 bigint,
    f1972 bigint,
    f1973 bigint,
    f1974 bigint,
    f1975 bigint,
    f1976 bigint,
    f1977 bigint,
    f1978 bigint,
    f1979 bigint,
    f1980 bigint,
    f1981 bigint,
    f1982 bigint,
    f1983 bigint,
    f1984 bigint,
    f1985 bigint,
    f1986 bigint,
    f1987 bigint,
    f1988 bigint,
    f1989 bigint,
```



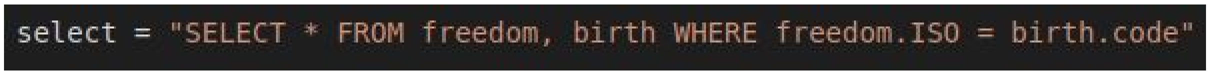
```sql
DROP TABLE IF EXISTS freedom;

CREATE TABLE freedom (
    year integer,
    country text,
    ISO text,
    EF real,
    rank real,
    PRIMARY KEY(ISO, year)
);
```



```sql
\COPY freedom (year, ISO, country, EF, rank) FROM '/home/christian/Documents/Database/freedom.csv' DELIMITER ',' CSV HEADER;
\COPY birth FROM '/home/christian/Documents/Database/birth.csv' DELIMITER ',' CSV HEADER;
```

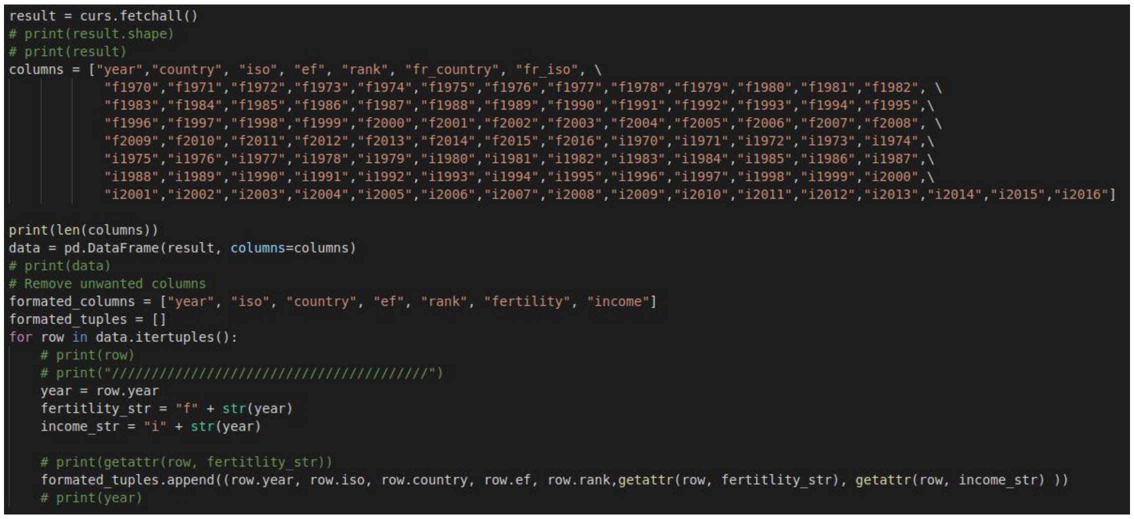
ML:

I choose to explore two different ML models, a simple linear regression method and a neural network. As this was a regression problem (fertility is an integer not corresponding to a class) I was able to look up a baseline for current models that typically work well with regression problems and use them as a starting point. Linear regression and neural net models seemed to be the consensus. The data was pulled from the DB using a simple select on the two tables with an inner join on iso code.



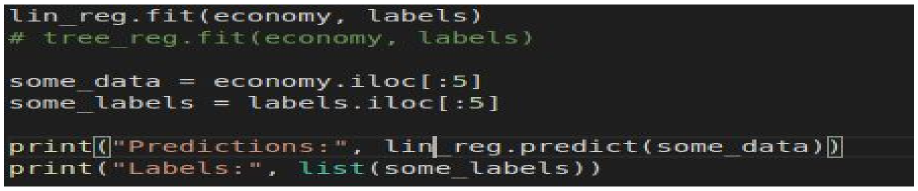
```
select = "SELECT * FROM freedom, birth WHERE freedom.ISO = birth.code"
```

I then loaded this unfiltered data into a dataframe dictating new column names that I could then use as keys to filter the data. I then iterated over every entry and threw out any columns that were not (“year", "iso", "country", "ef", "rank", "fertility", "income") with fertility and income being just for the specified year.

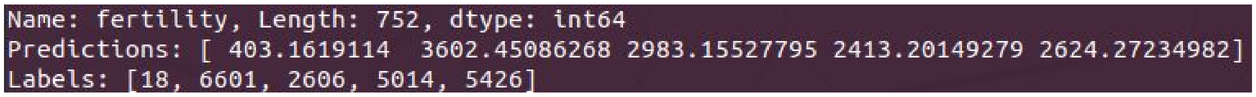


```
result = curs.fetchall()
# print(result.shape)
# print(result)
columns = ["year","country", "iso", "ef", "rank", "fr_country", "fr_iso", \
            "f1970","f1971","f1972","f1973","f1974","f1975","f1976","f1977","f1978","f1979","f1980","f1981","f1982", \
            "f1983","f1984","f1985","f1986","f1987","f1988","f1989","f1990","f1991","f1992","f1993","f1994","f1995",\
            "f1996","f1997","f1998","f1999","f2000","f2001","f2002","f2003","f2004","f2005","f2006","f2007","f2008", \
            "f2009","f2010","f2011","f2012","f2013","f2014","f2015","f2016","i1970","i1971","i1972","i1973","i1974",\
            "i1975","i1976","i1977","i1978","i1979","i1980","i1981","i1982","i1983","i1984","i1985","i1986","i1987",\
            "i1988","i1989","i1990","i1991","i1992","i1993","i1994","i1995","i1996","i1997","i1998","i1999","i2000",\
            "i2001","i2002","i2003","i2004","i2005","i2006","i2007","i2008","i2009","i2010","i2011","i2012","i2013","i2014","i2015","i2016"]

print(len(columns))
data = pd.DataFrame(result, columns=columns)
# print(data)
# Remove unwanted columns
formated_columns = ["year", "iso", "country", "ef", "rank", "fertility", "income"]
formated_tuples = []
for row in data.itertuples():
    # print(row)
    # print("/////////////////////////////////////////")
    year = row.year
    fertitlity_str = "f" + str(year)
    income_str = "i" + str(year)

    # print(getattr(row, fertitlity_str))
    formated_tuples.append((row.year, row.iso, row.country, row.ef, row.rank,getattr(row, fertitlity_str), getattr(row, income_str) ))
    # print(year)
```

This was one of the tricker parts to get right. I then utilized Scikit-learn to handle splitting the data into a training and testing set. From both sets the input data had its fertility column removed and for the labels I only kept the fertility column. A simple linear regression model was fit to the data with Scikit-learn and evaluated against 5 test entries. This proved to be too complex a space for a simple linear model to accurately represent as the values varied by thousands.

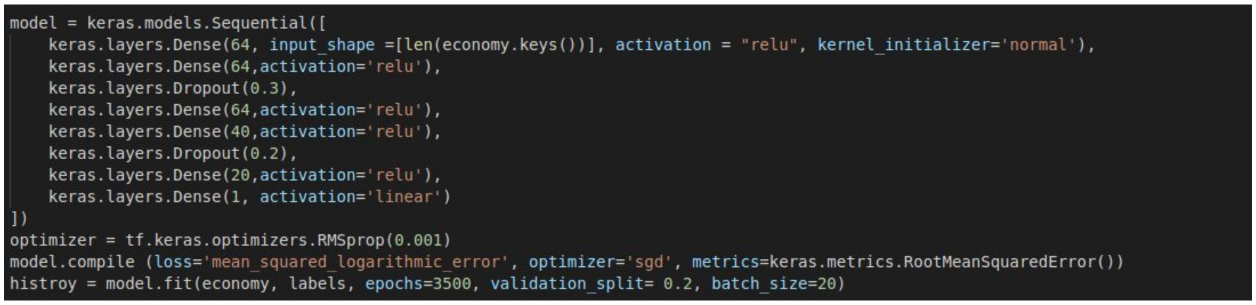


```
lin_reg.fit(economy, labels)
# tree_reg.fit(economy, labels)

some_data = economy.iloc[:5]
some_labels = labels.iloc[:5]

print("Predictions:", lin_reg.predict(some_data))
print("Labels:", list(some_labels))
```



```
Name: fertility, Length: 752, dtype: int64
Predictions: [ 403.1619114  3602.45086268 2983.15527795 2413.20149279 2624.27234982]
Labels: [18, 6601, 2606, 5014, 5426]
```

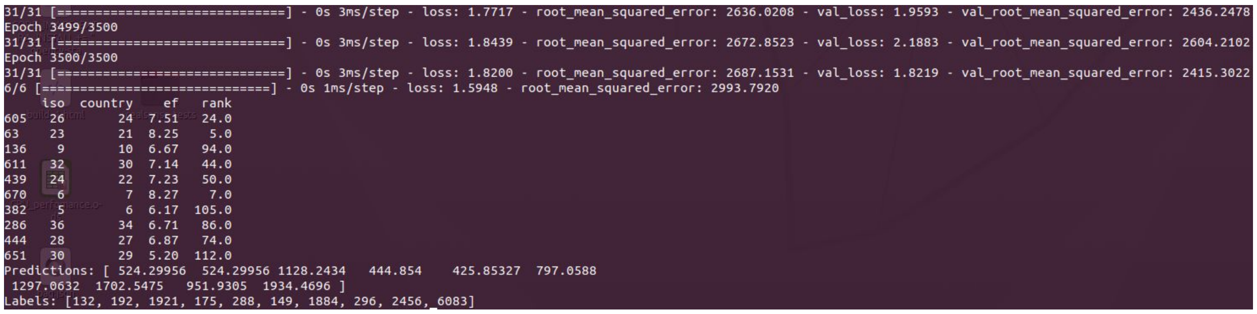
I then shifted my attention to a neural net as they tend to do a good job at recognizing deep patterns. To do this I used Tensorflow, an open ML library from Google. My model went through several cycles of tuning hyperparameters until I settled on the final one consisting of 4 hidden dense layers and two hidden dropout layers.



```
model = keras.models.Sequential([
    keras.layers.Dense(64, input_shape =[len(economy.keys())], activation = "relu", kernel_initializer='normal'),
    keras.layers.Dense(64,activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(64,activation='relu'),
    keras.layers.Dense(40,activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(20,activation='relu'),
    keras.layers.Dense(1, activation='linear')
])
optimizer = tf.keras.optimizers.RMSprop(0.001)
model.compile (loss='mean_squared_logarithmic_error', optimizer='sgd', metrics=keras.metrics.RootMeanSquaredError())
histroy = model.fit(economy, labels, epochs=3500, validation_split= 0.2, batch_size=20)
```

It was evaluated using a mean squared logarithmic error which is simply selected from documentation on tensorflow about regression models. This model was then trained for 3500 epochs at 19 steps per epoch for a total training time of around 15 minutes on my desktop machine. At first the results were terrible, early on the training process (low epochs) the predicted values were all over the place off my orders of magnitude at their worst. I was not confident that there was going to be a correlation at this point. I let it continue training however which resulted in a clue as to what was happening, by the time I would finish training all predicted values would be the same. It was as if the model had some feature that was completely swaying the result. I then realized that two of my utilized columns are probably not important and are over biased, (year and income). This was because the year really should have nothing to do with fertility as EF already represents that year and is really the feature we care more about and income was dropped because its range was from millions to billions. That range is never a good idea to pass into a model for training. I could have utilized the income field had I normalized the data to a more conceivable range however I felt it best just to drop it all together. Once both of these features were dropped from the data frame the model really took off. After 3500 epochs I was able to get values within 3x the original, which while not necessarily accurate did show a direct pattern and correlation.

Neural Net Results:



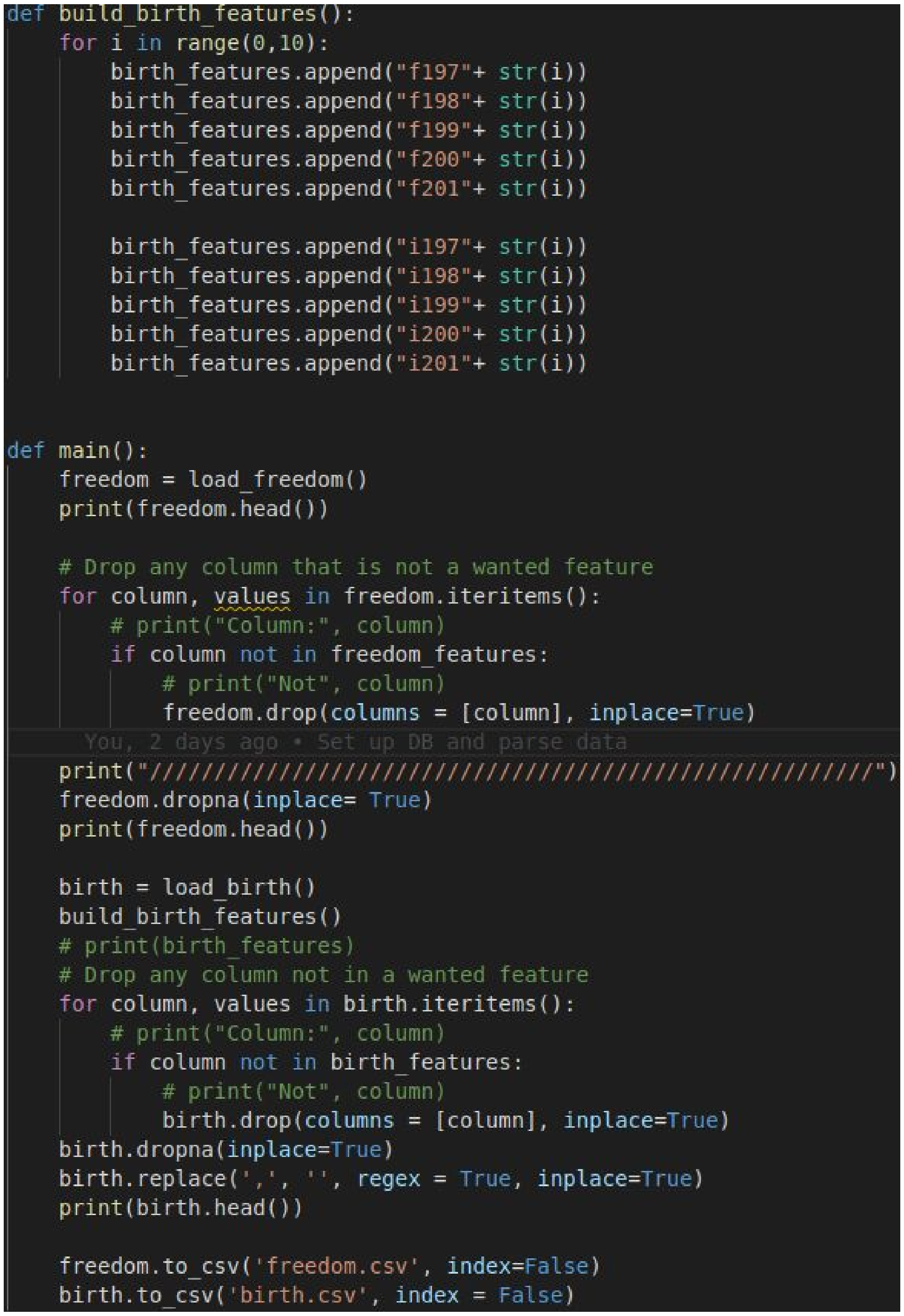
```
31/31 [==============================] - 0s 3ms/step - loss: 1.7717 - root_mean_squared_error: 2636.0208 - val_loss: 1.9593 - val_root_mean_squared_error: 2436.2478
Epoch 3499/3500
31/31 [==============================] - 0s 3ms/step - loss: 1.8439 - root_mean_squared_error: 2672.8523 - val_loss: 2.1883 - val_root_mean_squared_error: 2604.2102
Epoch 3500/3500
31/31 [==============================] - 0s 3ms/step - loss: 1.8200 - root_mean_squared_error: 2687.1531 - val_loss: 1.8219 - val_root_mean_squared_error: 2415.3022
6/6 [==============================] - 0s 1ms/step - loss: 1.5948 - root_mean_squared_error: 2993.7920
     iso  country    ef   rank
605   26       24  7.51   24.0
63    23       21  8.25    5.0
136    9       10  6.67   94.0
611   32       30  7.14   44.0
439   24       22  7.23   50.0
670    6        7  8.27    7.0
382    5        6  6.17  105.0
286   36       34  6.71   86.0
444   28       27  6.87   74.0
651   30       29  5.20  112.0
Predictions: [ 524.29956  524.29956 1128.2434   444.854    425.85327  797.0588
 1297.0632  1702.5475   951.9305  1934.4696 ]
Labels: [132, 192, 1921, 175, 288, 149, 1884, 296, 2456, 6083]
```

Conclusion:

So in conclusion it appears that there is infact a correlation between countries' economic freedom and their fertility rate. It would be interesting to see if this model could be refined further to get a more predictive model. I believe this could be done with some more advanced preprocessing on the data including normalization. It appears that the data is quite useful and personally it was very nice to see how easy it is to go from start to finish with a ML project utilizing a postgresql database as that will often be a source of data for future projects.

Code: https://github.com/Christian-Prather/database-final

clean-up.py



```python
def build_birth_features():
    for i in range(0,10):
        birth_features.append("f197"+ str(i))
        birth_features.append("f198"+ str(i))
        birth_features.append("f199"+ str(i))
        birth_features.append("f200"+ str(i))
        birth_features.append("f201"+ str(i))

        birth_features.append("i197"+ str(i))
        birth_features.append("i198"+ str(i))
        birth_features.append("i199"+ str(i))
        birth_features.append("i200"+ str(i))
        birth_features.append("i201"+ str(i))


def main():
    freedom = load_freedom()
    print(freedom.head())

    # Drop any column that is not a wanted feature
    for column, values in freedom.iteritems():
        # print("Column:", column)
        if column not in freedom_features:
            # print("Not", column)
            freedom.drop(columns = [column], inplace=True)
      You, 2 days ago • Set up DB and parse data
    print("///////////////////////////////////////////////////////////")
    freedom.dropna(inplace= True)
    print(freedom.head())

    birth = load_birth()
    build_birth_features()
    # print(birth_features)
    # Drop any column not in a wanted feature
    for column, values in birth.iteritems():
        # print("Column:", column)
        if column not in birth_features:
            # print("Not", column)
            birth.drop(columns = [column], inplace=True)
    birth.dropna(inplace=True)
    birth.replace(',', '', regex = True, inplace=True)
    print(birth.head())

    freedom.to_csv('freedom.csv', index=False)
    birth.to_csv('birth.csv', index = False)
```