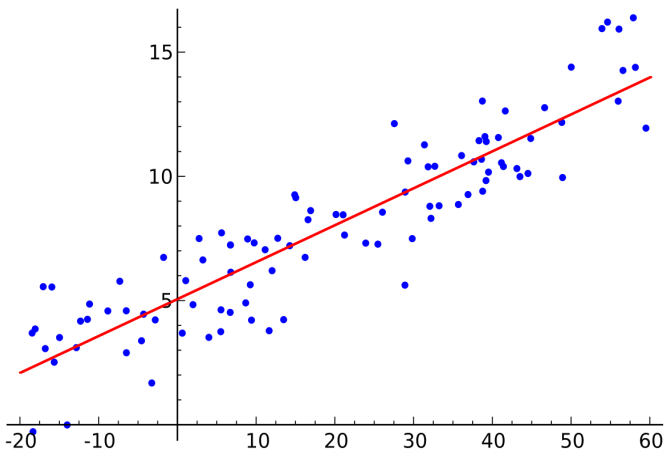
**Food for Thought**

- How does a linear regression algorithm work?
- What is the difference between Pearson's correlation coefficient and other types of correlation coefficients?
- Is it possible to accurately predict the birth and death rates of a country given other information about it?
- Are other supervised algorithms, such as random forest regressors, more efficient and/or more accurate than linear regression?
- What is the relationship, if any, between GDP per capita and birth rate?




**Introduction**

I will be working with a pre-cleaned version of the countries of the world dataset I used in last week's lesson. It contains information about population, area, GDP, birth and death rates, and region for 195 countries. I would like to use a linear regression model to predict the birth rate and the death rate of an unknown country, specifically of the three countries for which that data was absent originally.

**Step 1: Load the Data**

This may seem trivial, but it is always important before starting a machine learning project to make sure you have the data you want in the format you want. Here, I again have an Excel spreadsheet in the same folder as the Jupyter notebook in which I'm writing this code, so I can pull it in using xlrd and reformat it using pandas. (I'm also importing the numpy library because I'll most likely have to use it later.) This time, I use the head function from pandas mostly to make sure the data is in the format I want.

```
import pandas as pd
import numpy as np
import xlrd

xlsx = xlrd.open_workbook("CSC 357 Week 2 Lesson.xlsx", on_demand=True)
with pd.ExcelFile(xlsx) as wb:
    df = pd.read_excel(wb, "info")

df.head(5)
```

**Step 2: Remove Unnecessary Columns**

I'm actually going to be creating two different models, one to predict birth rate and one for death rate. In both cases I want to remove the remaining categorical variables, as well as the column containing values I want to predict using the *other* model. (While there might be some sort of relationship between birth rate and death rate, I'm assuming that neither value will be known.) I can do this two ways, but since I want to keep the majority of the columns, the more efficient method uses the drop function from pandas.

```
birth_data = df.drop(columns = ["Country", "Capital", "Death_rate"])
death_data = df.drop(columns = ["Country", "Capital", "Birth_rate"])
birth_data.head(5)
death_data.head(5)
```

**Step 3: Create a Training Set and Test Set**

Before we train a model, we need to separate our datasets into two groups: data with which to train the model and data with which to test it. We have no idea at the moment which of our variables will correlate most with birth rate or death rate, so we can simply sample countries randomly instead of using stratified sampling. Fortunately there is a method in the model_selection module in sklearn that we can use to split our data without much hassle.

```
from sklearn.model_selection import train_test_split
birth_train, birth_test = train_test_split(birth_data, test_size=0.2, random_state=42)
death_train, death_test = train_test_split(death_data, test_size=0.2, random_state=42)
```

**Step 4: Look for Correlations**

We want to know which variables in our dataset are more likely to be correlated with birth rate and death rate. The easiest way to figure this out is to use the corr method from pandas to compute a correlation matrix for all variables, then to extract just the column with relevant values.

```
birth_corr_matrix = birth_train.corr()
birth_corr_matrix["Birth_rate"].sort_values(ascending=False)
death_corr_matrix = death_train.corr()
death_corr_matrix["Death_rate"].sort_values(ascending=False)
```

**Step 5: Separate the Labels**

For any prediction algorithm, it is important to separate the values we want to predict (also called *labels*) from the values we will use to predict them. We can use the pandas library to copy the birth rate and death rate column into label variables and drop the corresponding columns from our datasets.

```
birth_train_labels = birth_train["Birth_rate"].copy()
birth_train = birth_train.drop("Birth_rate", axis=1)
death_train_labels = death_train["Death_rate"].copy()
death_train = death_train.drop("Death_rate", axis=1)
```

**Step 6: Scale the Data**

It turns out that most machine learning algorithms perform best when all the data is on the same scale. So in reality, our data was not as clean as we thought it was at the beginning, since we have columns like population which range from a few thousand to over a billion, as well as the region columns which only contain 0 and 1. We can use the StandardScaler class from sklearn to standardize the values of all of our variables.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
birth_train = pd.DataFrame(scaler.fit_transform(birth_train))
death_train = pd.DataFrame(scaler.fit_transform(death_train))
birth_train.columns = ["Africa", "Asia", "C_America", "Europe", "N_America",
                       "Oceania", "S_America", "Population", "Area", "Density",
                       "GDP_millions", "GDP_per_capita"]
death_train.columns = ["Africa", "Asia", "C_America", "Europe", "N_America",
                       "Oceania", "S_America", "Population", "Area", "Density",
                       "GDP_millions", "GDP_per_capita"]
birth_train.head(5)
death_train.head(5)
```

**Step 7: Train a Linear Regression Model**

Finally, all of the data is in the shape we want, so we can begin to use it for training. As with so many other machine learning tasks, the things we need to import come from the sklearn library.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
birth_lr = lin_reg.fit(birth_train, birth_train_labels)
death_lr = lin_reg.fit(death_train, death_train_labels)
```

**Step 8: Evaluate the Model**

It's super easy to train a model when the sklearn library already has predefined classes and methods for so many different algorithms. But perhaps more important than training the model is measuring whether the model works well. A fairly standard performance measurement for linear regression algorithms is the root mean squared error (RMSE), which takes the vertical distance from each point to the fitted line, squares them so they're all positive, adds them together, averages them, and takes the square root of that final quantity. We can find a method to calculate mean squared error (MSE) in the metrics module of sklearn, and we can derive RMSE from there.

```
from sklearn.metrics import mean_squared_error
birth_pred = birth_lr.predict(birth_train)
birth_mse = mean_squared_error(birth_train_labels, birth_pred)
birth_rmse = np.sqrt(birth_mse)
death_pred = death_lr.predict(death_train)
death_mse = mean_squared_error(death_train_labels, death_pred)
death_rmse = np.sqrt(death_mse)
print("RMSE for Birth Rate: ", birth_rmse, "\nRMSE for Death Rate: ", death_rmse)
```

**Conclusion**

The result of training a linear regression algorithm on this dataset is an RMSE of about 15.2 for birth rate (which ranges from 6.5 to 43.7 in the full dataset) and about 2.1 for death rate (which ranges from 1.6 to 15.1). There are probably several ways to improve these results: if region is important, we could classify countries into more than seven regions; we could gather more data on a different variable such as healthcare spending per capita; we could use a different type of algorithm, such as a decision tree regression algorithm; and we could use a grid search to find the optimal combination of hyperparameters.

**Notes**

- Keep in mind that not only is linear regression the simplest machine learning algorithm because it is one of the only ones with a closed form equation we can calculate by hand, but I used a small, relatively simple dataset. More data and more complex algorithms will generally lead to better results.
- I didn't evaluate the performance of the final linear regression algorithm on the test data, only the training data. It's entirely possible that the RMSE scores for the test data could be completely different.

**Resources**

Most of this code is adapted from the Jupyter notebook for Chapter 2 of our textbook, which can be accessed here: https://github.com/ageron/handson-ml/raw/master/02_end_to_end_machine_learning_project.ipynb
Documentation for the LinearRegression class:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
More on linear regression algorithms:
https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a