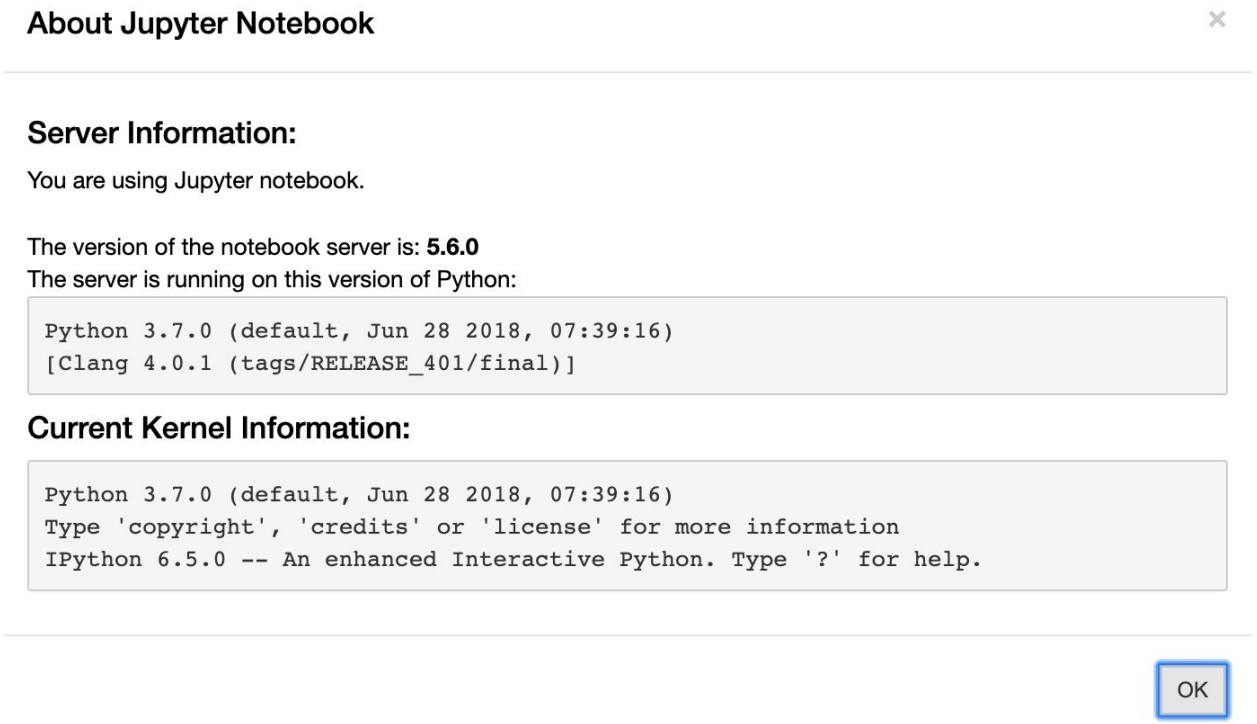
Bryce Reinhard
RTI
Data-scientist-exercise01 writeup
06/27/19

Technology:

Environment:



Software:

```
# imports
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
```

Cleaning data:

I was given a sqlite database that had 9 tables worth of data that was taken from the 1996 US census. These tables are:

workclasses
education_levels
marital_statues
occupations
relationships
races
sexes
countries
records

The majority of the data was stored in the records table and it had 48842 entries existing in it. The first problem was filling in all of the data in the records table to make it more easily understandable. In its raw form the table has a number of columns that list a foreign key that then points to the actual value. As an example, let's look at the sex_id column in the original database. This column only has two possible values: 1 or 2. So for the 48842 rows of the records table the sex_id column can only have a possible value of 1 or 2. If the sexes table is opened up it looks like this:

id	name
1	Female
2	Male

So in the sex_id column in records it gives you the id (1 or 2) of the sex that is actually stored there. The problem is that there are a number of foreign key ids being used to represent data and most weren't as simple as the sexes table. Here's another example to illustrate the point:

id	name
1	Divorced
2	Married-AF-spouse

3	Married-civ-spouse
4	Married-spouse-absent
5	Never-married
6	Seperated
7	Widowed

This table contains a lot more info than the sexes table. The marital_status_id column contains values in the [1,7] range compared to the [1,2] range of before. The way to fill in all the foreign key ids is to use SQL inner join query. Here's the query:

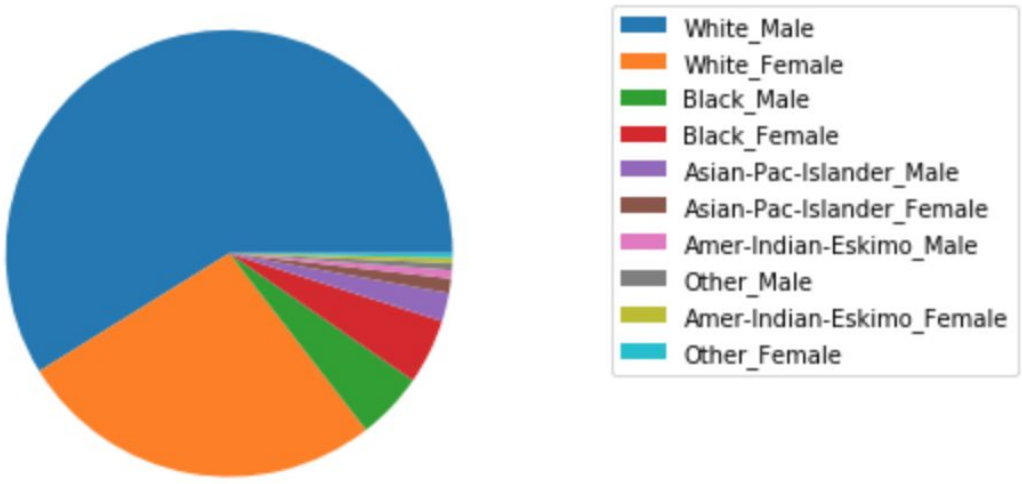
```
SELECT records.id, records.age, workclasses.name, education_levels.name,
records.education_num, marital_statuses.name,
occupations.name, relationships.name, races.name, sexes.name, records.capital_gain,
records.capital_loss,
records.hours_week, countries.name, records.over_50k
FROM records
INNER JOIN workclasses ON workclasses.id=records.workclass_id
INNER JOIN education_levels ON education_levels.id=records.education_level_id
INNER JOIN marital_statuses ON marital_statuses.id=records.marital_status_id
INNER JOIN occupations ON occupations.id=records.occupation_id
INNER JOIN relationships ON relationships.id=records.relationship_id
INNER JOIN races ON races.id=records.race_id
INNER JOIN sexes ON sexes.id=records.sex_id
INNER JOIN countries ON countries.id=records.country_id
```

The inner joins replace the ids of the foreign key ids with the name value. Now the data is human readable and after running the query to get the data it is stored in a pandas dataframe. I added two extra columns to this new dataframe to make life easier for me later on:
A profit column which is the capital_gain - capital_loss column summed together.
A race_sex column which concatenates the race and sex column together with an underscore (_) between them. After that I save the dataframe to a csv so as to avoid having to run the sql query too many times. I should mention that I had to change default jupyter settings in order to be able to run the sql query. More is explained in the comments of that particular cell.

Analyzing:

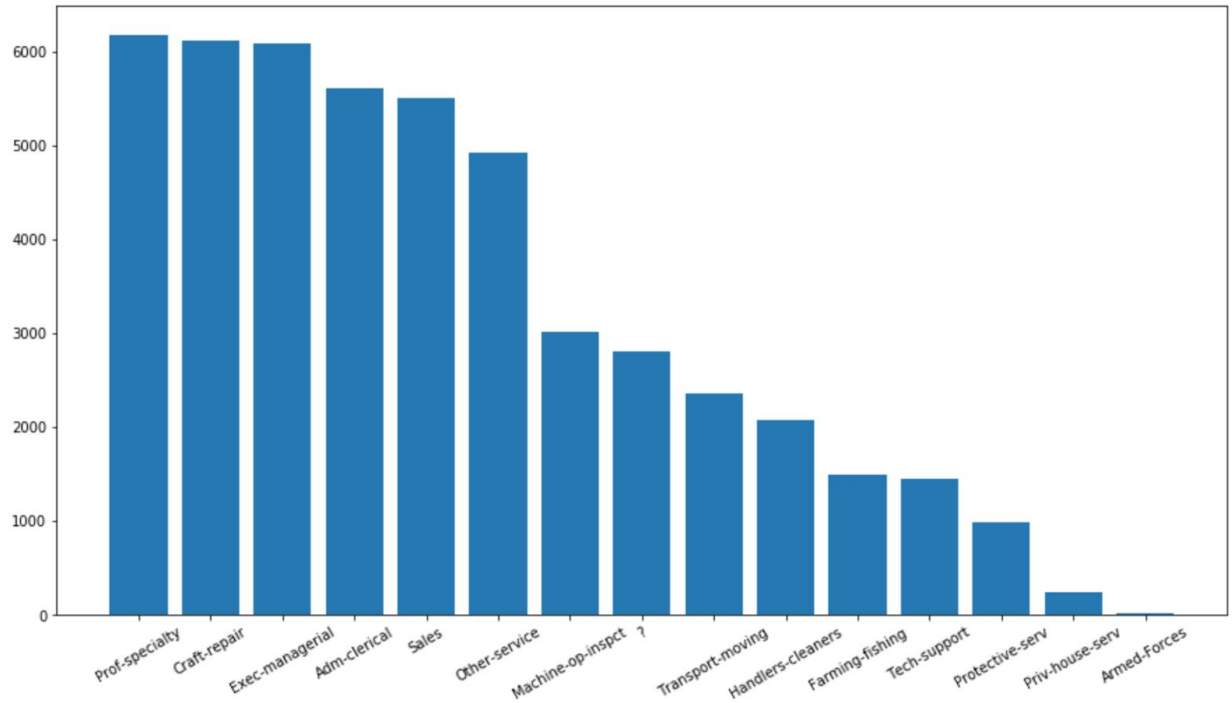
Now I have a csv I can easily import into a pandas dataframe to perform some analyzation on. At this stage I wasn't too familiar with the data yet so I decided to make a simple pie graph to try to better understand. What I graphed was the race_sex column I had made previously. Graph is here:

```
White_Male                   28735
White_Female                 13027
Black_Male                    2377
Black_Female                  2308
Asian-Pac-Islander_Male       1002
Asian-Pac-Islander_Female     517
Amer-Indian-Eskimo_Male       285
Other_Male                    251
Amer-Indian-Eskimo_Female     185
Other_Female                  155
Name: race_sex, dtype: int64
```



By creating a new column I was able to easily create a relationship between race and sex and understand how much of the data is made up of people of certain sexes and ethnicities.

The next type of chart I create is a bar chart and it also uses the race_sex column to more easily navigate the data. The bar graph shows for a certain race_sex what occupations they are most frequently working in. The chart explains it better than words can. Here:



This chart is a lot bigger than the pie plot so I wasn't able to fit all of it. Above the actual chart it lists the exact y value for each x to be more accurate. This graph is actually showing the total sum for each occupation. Other graphs show occupations for a specific race_sex. Then I go a bit into how relationships work for the different sexes and create a simple table.

```
Male
Married-civ-spouse       19899
Never-married             8899
Divorced                  2632
Separated                  599
Married-spouse-absent      324
Widowed                    285
Married-AF-spouse           12
Name: marital_status, dtype: int64

Female
Never-married            7218
Divorced                 4001
Married-civ-spouse       2480
Widowed                  1233
Separated                 931
Married-spouse-absent     304
Married-AF-spouse          25
Name: marital_status, dtype: int64
```

Model:

The next step was to create a model that would predict the odds of a person earning over 50k given a bunch of data. The model would have to be supervised since I had labeled data to train it with and it would have to be categorical, since it could only have two possible outcomes of yes this person will make over 50k or no they won't. I have done this type of problem before and used a CountVectorizer with a MultinomialNB model. I split all but the last 150 entries into a data set and used train_test_split from sklearn.model_selection to train the model. The accuracy:

```
print("Train accuracy is %.2f %%" % (model.score(X_train, y_train)*100))
print("Test accuracy is %.2f %%" % (model.score(X_test, y_test)*100))
```

```
Train accuracy is 94.49 %
Test accuracy is 94.31 %
```

After creating a function that would be able to give predictions of the data passed in I set up a cell that would be able to test the 150 entries I left out of the training data set. I didn't want to use data that the model had been trained on.

Future work:

Some future work that I would do if I had more time.
Replace the pie chart of race_sex with a donut chart.
Find out what factor increases the odds of getting over 50k the most.
More in depth analytic charts that explore more subtle relationships.
Try to get a higher degree of accuracy with the model.
Create a chart that uses the models predictive capabilities better.