Aditya Patadia
netID: ajp433
Section 05

Final Project Report

For my final project, I decided to use a csv file that contained pertinent information about government employees and their salaries. This csv file is officially maintained by the government and updated fairly regularly, making it both accurate and a good way to conduct my analysis on. I chose this specific database because I find that salary is an often "hidden" part of American society in the sense that people not often quick to disclose to others. On the other hand, those that do reveal their income publically on places such as social media are often outliers, and do not give an accurate representation of the general public. This database, though partially skewed due to government relation, provides a good example of salaries for the average American.

Another reason for picking this database was the relevance of salaries to our lives. For something this important, it's important to understand what exactly a "common salary is", especially when factoring in years of experience, position, and the different companies one person can work for. While databases can only give so much information, it can provide a good overview of relevant features that may be able to predict specific variables, such as salary in this case. In this case, it is a very practical thing for companies to want to keep track of - especially compared to other databases I considered using such as one that involved walkability indexes for various cities. I also found that the database's size to be very interesting - it dates to records from over 10 years ago, which makes it a very aged database.

Accessing the Database and Visualization

The main part of the project came from looking through various parts of the database and visualizing it to get a clear picture. This database was around 256,000 records in length, and there were 15 columns in total - Calendar Year, Calendar Quarter, As Of Date, Full Name, Authority Name, Salary Hourly Rate, Master Section Desc, Master Title Desc, Employee Relations Group, Compensation Method, Regular Pay, Overtime Payments, All Other Payments, YTD Earnings, and UniqueID. Most of the columns are self explanatory, but Salary Hourly Rate was more of a general salary, not just pay per hour, and I used that for most of my project.

The first thing I did was attempt to clean the data. Luckily for me, there was not much to clean up - the data was naturally precise and did not contain many nulls.. The only thing I had to fix was the "Employee Relations Group" column, which contained a total of 1500 null values. Since

this column was not particularly replaceable, I ended up dropping all rows with a null there, which was only a small part of the database.

Next, I looked up some trivial information by seeing the most common first names, last names, and first and last names together. Although this wasn't statistically significant, it was fun to see as a random piece of trivia in a way. I created a new column in the dataframe for first name and another for last name by taking the full name and splitting it by the comma. Then, I simply used .value_counts() and took the top 3 for each set of data. The top 3 first names ended up being Michael -> John -> Robert. The top 3 last names ended up being Williams -> Smith -> Johnson. The top 3 full names were Smith,James -> Perez,Luis -> Brown,Michael. Surprisingly, the top first name and top last name together was not as common as expected.

Moving onto some more relevant data, I looked at the date of joining to see if it related to the salary. For this database, there were 13 unique values for the joining date, and upon further checking, it was set to the december of each year since 2010. This meant that it was effectively grouped by year for convience, and so I looked at the average salary for each year. I expected the salaries to correspond directly to the years due to inflation. Surprisingly, this was not the case completely. From 2023-2015 it was in order, but 2024 came after that, likely because 2024 was based solely on new hires which might have a generally lower rate. More shocking was that 2016 came after 2014. My guess is that there was either some sort of economic downturn at the time, or there was some sort of issue in the government that caused the significant drop. Regardless, aside from this bump in the data, I think the ordering of the years was fairly consistent to the average salary of the year.

Next, I checked the types of compensation methods, which was also astonishing due to how many more types there were other than "full time" and "part time" - there was other, seasonal, hourly, T, RFT, INT, RPT, TPT, Active, Terminated, and Leave. I wanted to see how much of each type constituted the database, so I took the values as percentage counts and sorted them in order. The result was more or less what I expected - full time made up a large portion of the database, with around 88%, although I expected it to be a bit less than that. Part time made up around 8.5%, Seasonal made up 1.6%, and Other made up 1.4%, while the remaining categories filled up the rest.

With this in mind, I figured it was best to split the data into one solely with full time employment, and the other solely with part time employment, which was as simple as just using a boolean check df[df['COMPENSATION_METHOD'] == 'FULL TIME'], and .head() to double check.

I moved on to some more checking by using the 3 main statistical ways of calculating averages - mean median and mode. To do this, it involved the same method, which was to use

df[‘SALARY_HOURLY_RATE’] with either .mean() .median() or .mode(), with the df either being df, full_df, or part_df. The only other thing to keep in consideration was that the .mode() returned a series of values, so I had to take [0] for the most common one. Lastly, I did some simple formatting as well as some comparison to see how much the part time values affected the mean salary for the dataframe with both types. Additionally, I used a box plot to visualize the data, because it makes the data much more clear visually and shows the quartiles, median, and any potential outliers. Once again, I was thrown for a loop when I saw a very suspicious high value to the far right. More specifically, this high value was in the part time batch, and was higher than any other value in the full time group, which was immediately offputting. I looked at the data itself again, using idxmax() in order to retrieve the particular record where this anomaly took place. When comparing it to other values in the field, it seemed that that particular field was very irregular. I checked the max for the full time dataframe just to see how the fields matched there, and sure enough, the money earned was more on par with other metrics of earning money such as ‘YTD_EARNING’ and ‘REGULAR_PAY’. Unless i am overlooking something that could give someone that much money for a part time job, I think that there must have been some sort of error in the database that was never resolved, and I dropped that piece of data - which made a $33.21 difference to the salary mean, with 21370 rows!

The next visualization I did and the most important one in the context of my intention with this project was to break down the data by the amount of money made. I checked the minimum salary (0) and the maximum salary, and created 8 equal divisions to see how many people belonged to each category. I started by using np.linspace to create the even categories from min to max, then I used pd.cut to separate the values by the ‘SALARY_HOURLY_RATE’ into the bins I created with linspace. I then followed the standard procedure to create a bar plot, with the exception of having custom labels with “k” to represent the amount of money in 1,000s while also making it legible. The data here was interesting - for 36k-73k, 73k-110k, 1k-36k, 110k-147k, 147k-184k, the results were 68.1%, 20.7%, 7.0%, 3.3%, and 0.7%. While only 7% belonged to a group that was working full time and making less than 36k, the vast majority of people were making somewhere between 36-73k per year. 20% were above average in capping out to 110k, but only a staggering 4% of employees could cross that threshold. Putting it into perspective, it definitely breaks the stigma that 100k is “normal” - in fact, it is well above the means of 80% of full time employees. I also found it interesting that 7% belonged to 1k-36k, while 3.3% belonged to 110k-147k, but I am inclined to believe that the data does not account for the entirety of their situations.

One of the last visualizations I did was to look at each job salary by the title. Of course, this was not feasible to do with every single job title out there, so I took the top 10 to try and see the most valued jobs in terms of salary. Doing this was as simple as grouping by the master_title_desc, taking the mean, and using .head(10). I used a horizontal bar chart, and the results were slightly different than I expected. A lot of the higher positions involved executive, chief, or senior

positions. However, I was pleasantly surprised to find that the the highest paid position on average was still being a doctor, which makes is the most reliable when it comes to anyone wanting to achieve a high salary.

Machine Learning Implementation

For this part of the assignment, I wanted to determine if there were some factors that could predict the salary someone would make, given the available data in the dataset. After doing some tests before, I realized there were a few columns that I should keep in mind - AS_OF_DATE, which seemed to have a positive correlation with salary (higher year = higher money), AUTHORITY_NAME, which naturally makes sense as different companies have different values, MASTER_TITLE_DESC, as the type of occupation impacted the salary as seen in the visualization above, and COMPENSATION_METHOD, because part time salaries will generally be lower than the compensation from full time jobs.

I used sklearn for the machine learning, which a good library that streamlines the process of machine learning down to a few simple methods. I made a "ml_df" that contained the relevant columns, and I split into X for training and y for the salary, which I wanted to predict. Originally, I operated with the full dataset, unaware of how truly large the dataset was. I let it run for around 40 minutes, and it was still running, and at that point I realized it might be better to reduce the dataset, so I used .sample(frac=0.1) to take a random sample of 10% of the database. Surely enough, this allowed the code to compile within a few seconds.

As for the actual implementation, I needed to extract the year and month (because of the rare case with September) as the day was unchanging for the set of values that AS_OF_DATE could be. This involved simply using pd.to_datetime() and formatting it to suit my needs. After getting the year and the month, I used one-hot-encoding for the categorical variables, dropped the original date column, and moved on to splitting the data via train_test_split. A test_size of 0.3 was too large, so I ended up trying 0.25 and 0.2, which ended up being better in the grand scheme of things. From there, I used the inbuilt models to check the MSE and R^2 score, which was way off the mark. I got errors such as 55452373656166978912098713 6 for MSE and -716811102302484992.00 for R^2, although it varied slightly based on randomness within the training data. Because of these absurdly large values, I realized I had to change something, and I figured the best approach to try would be to use standardization in case there was a lot of variance I was not accounting for. For this, I used an inbuilt method from sklearn called StandardScaler, and essentially transformed the train and test data with .transform() and .fit_transform(). This made the predictions much better - the error results had an absolute value between 0-10 for both scores.

This was still not very good. My next thought was to try and use polynomial regression in order to assign better weights, which I thought might have been the issue. I encountered the same issue

that I had when trying to originally run the code - the dataset was too big, even when taking a sample as small as 10% of the data. I had to shrink it to a very low amount, and that ended up being too volatile, and not very accurate overall.

Back to the next course of action, I had to figure out what to do with the big errors. Going off the R^2 score alone, this meant that the predictions were worse than simply calculating the mean of the data and predicting each value to be that. Although I had originally thought that it would be too big of a giveaway, I realized I needed to give more data to the model, and so I ended up adding the YTD_EARNINGS column, and repeated the same steps as above. At first, the MSE and R^2 score was too big as usual, but after standardizing the data, it ended up working out; the R^2 score turned out to be around 0.8, which was pretty good and meant that the model was predicting the salaries fairly well.

Database Component

While some similar observations could be made by using matplotlib, pandas, and numpy libraries, SQL was still an effective way of gaining some insight into particular metrics of the data. However, the immediate problem with this was that my data was in the form of a csv file, which is easier to export, import, and view, and thus more common when it came to obtaining datasets. In order to convert that to an actual database to use SQL on, I had to use sqlalchemy - I created the engine and made my database called "government_employees.db", then turned my dataframe into sql by using df.to_sql, with the table name being 'govinfo'. Afterwards, I used sqlite3 in order to serve as a lightweight processor for executing my queries. The first thing I did was make sure that my table was actually processed and ready to be queried on, which it was.

Before anything else, I wanted to see if it worked normally. I executed a query to check all the records where the full name was "SMITH,JAMES" which as you can recall earlier was the most common first name and last name combination. From there, it was a matter of creating a function to get rid of the redundancy in needing to call conn.execute() and cur.fetchall() for each query. I used two separate functions just in case I wanted to use the output rather than print it directly, but it made little difference.

Among these queries, one was similar to my visualization that I did earlier - I checked each salary starting from 10,000 and moved up. It was interesting to see how many people existed in each category, and I did the same with the overtime payments to see how much of a difference overtime made to some people. Another query involved looking at the top 5 employees by authority name (company), and a few others calculated averages of salaries with that in mind. Overall, the queries were not that useful unless it came to using functions like COUNT(*) or AVG() or GROUPBY() - there were simply too many records in my database to be able to see them all effectively. However, this is solely based on my use case, and may be different depending on the need.

Summary

The problem I was solving was trying to figure out the salary of a person from various attributes that relate to their employment in a company. The strategic aspects involved were using visualizations in order to learn more about the features and how they were connected, and using queries to gain more insights, with machine learning to actually make the predictions.

My project was important because of how important salaries are in the daily world. They essentially dictate your quality of life, so understanding what it is comprised of is useful to many people. In addition to being interested in this topic, I think it was the first time I have done a full database oriented project which made it a learning experience on manipulating analyzing and using data.

I used a dataset from the government, which I found from https://data.nj.gov/Government-Finance/YourMoney-Authority-Payroll/kiki-imre/about_data . I used matplotlib visualizations and I used linear regression from scikit. I experimented with each of the attributes and saw how much they correlated with the salary. My key finding was that salary is often multifaceted and does not depend on just the company, time, or even position. However, there can be assumptions made regarding average salaries of specific fields (like Doctors) or newly given salaries resulting in more money. This was mostly similar to my original hypothesis, but it was still cool to see.

The hard part about my approach was that there probably wasn't enough full and complete data on each and any individual. For example, I think it would have been much more yielding if I had access to something like the colleges each person went to beforehand, or something like how rich they were growing up. That being said, there was still enough data to make good statistical observations, and I believed more than anything it shows the true nature of how much people make in general, as opposed to the norms society pushes onto us.