Machine Learning - Documentation

Data Science Job Salaries

1. Introduction - Exploratory Data Analysis (EDA) Documentation

Data Science is one of the fastest growing jobs in the IT sector. In our project we will dive deeper into data science and similar field jobs to get more understanding of what is happening. Since we are working with a lot of job listings related to Data Science, we want to come up with a few questions that can help us better understand the data we are working with and what our goal is.

1.1. Purpose

The initial goal for EDA is to extract as much information as possible from our data set to get both a written and a visual understanding of the potential data we can use, correlate, get rid of etc. To find our target value and implement it.

The final goal of EDA is to first formulate and then answer these questions with the codes we use to conclude what data we will be using exactly for our machine learning model to predict potentially on the test set.

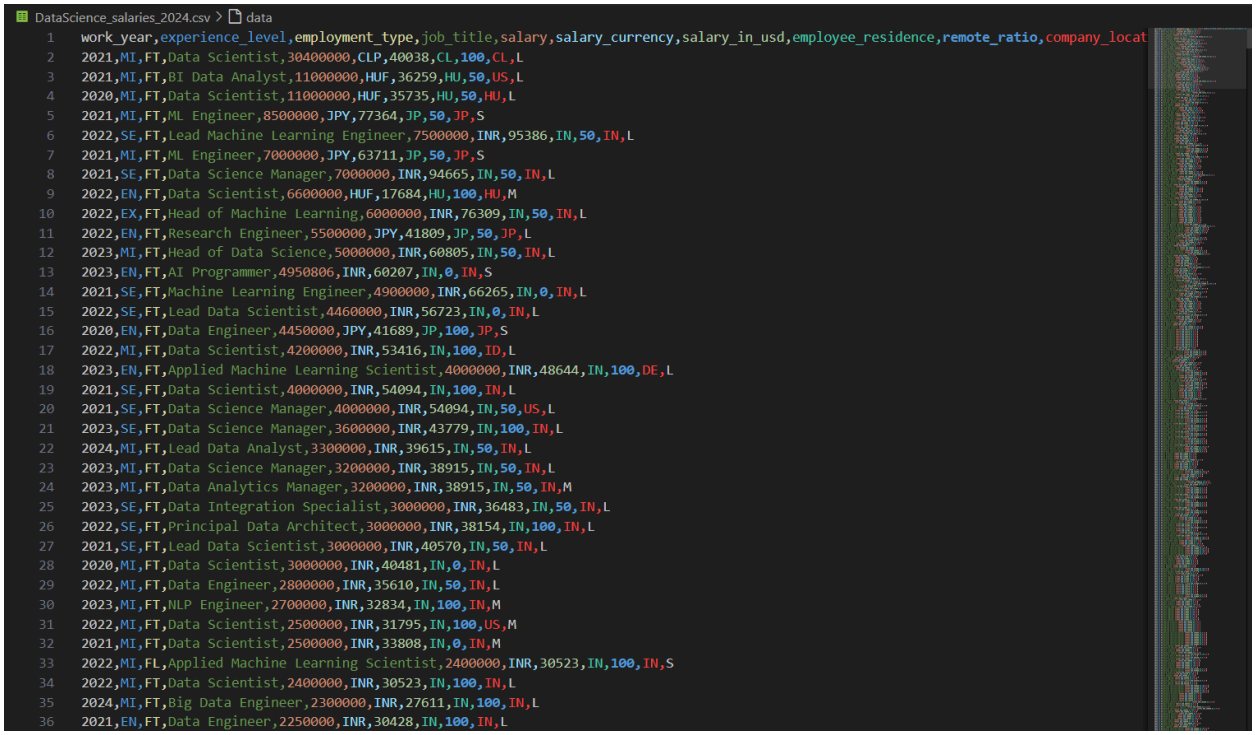
1. How many job listings are distributed per year, currency, experience level and employment type?

2. Are there any trends in salaries for experience level, company size, employee residence?

3. What is the distribution of experience levels, company location and employee residence and so on across the the dataset for salaries?

1.2. Dataset Description

We are working with VSCode so after downloading the dataset from the given website we had DataScience_salaries_2024.csv. Since the file was already in csv file, we just imported it to vscode to read. But the first problem was that the data was unreadable:



```
DataScience_salaries_2024.csv > data
work_year,experience_level,employment_type,job_title,salary,salary_currency,salary_in_usd,employee_residence,remote_ratio,company_locat
2021,MI,FT,Data Scientist,30400000,CLP,40038,CL,100,CL,L
2021,MI,FT,BI Data Analyst,11000000,HUF,36259,HU,50,US,L
2020,MI,FT,Data Scientist,11000000,HUF,35735,HU,50,HU,L
2021,MI,FT,ML Engineer,8500000,JPY,77364,JP,50,JP,S
2022,SE,FT,Lead Machine Learning Engineer,7500000,INR,95386,IN,50,IN,L
2021,MI,FT,ML Engineer,7000000,JPY,63711,JP,50,JP,S
2021,SE,FT,Data Science Manager,7000000,INR,94665,IN,50,IN,L
2022,EN,FT,Data Scientist,6600000,HUF,17684,HU,100,HU,M
2022,EX,FT,Head of Machine Learning,6000000,INR,76309,IN,50,IN,L
2022,EN,FT,Research Engineer,5500000,JPY,41809,JP,50,JP,L
2023,MI,FT,Head of Data Science,5000000,INR,60805,IN,50,IN,L
2023,EN,FT,AI Programmer,4950806,INR,60207,IN,0,IN,S
2021,SE,FT,Machine Learning Engineer,4900000,INR,66265,IN,0,IN,L
2022,SE,FT,Lead Data Scientist,4460000,INR,56723,IN,0,IN,L
2020,EN,FT,Data Engineer,4450000,JPY,41689,JP,100,JP,S
2022,MI,FT,Data Scientist,4200000,INR,53416,IN,100,ID,L
2023,EN,FT,Applied Machine Learning Scientist,4000000,INR,48644,IN,100,DE,L
2021,SE,FT,Data Scientist,4000000,INR,54094,IN,100,IN,L
2021,SE,FT,Data Science Manager,4000000,INR,54094,IN,50,US,L
2023,SE,FT,Data Science Manager,3600000,INR,43779,IN,100,IN,L
2024,MI,FT,Lead Data Analyst,3300000,INR,39615,IN,50,IN,L
2023,MI,FT,Data Science Manager,3200000,INR,38915,IN,50,IN,L
2023,MI,FT,Data Analytics Manager,3200000,INR,38915,IN,50,IN,M
2023,SE,FT,Data Integration Specialist,3000000,INR,36483,IN,50,IN,L
2022,SE,FT,Principal Data Architect,3000000,INR,38154,IN,100,IN,L
2021,SE,FT,Lead Data Scientist,3000000,INR,40570,IN,50,IN,L
2020,MI,FT,Data Scientist,3000000,INR,40481,IN,0,IN,L
2022,MI,FT,Data Engineer,2800000,INR,35610,IN,50,IN,L
2023,MI,FT,NLP Engineer,2700000,INR,32834,IN,100,IN,M
2022,MI,FT,Data Scientist,2500000,INR,31795,IN,100,US,M
2021,MI,FT,Data Scientist,2500000,INR,33808,IN,0,IN,M
2022,MI,FL,Applied Machine Learning Scientist,2400000,INR,30523,IN,100,IN,S
2022,MI,FT,Data Scientist,2400000,INR,30523,IN,100,IN,L
2024,MI,FT,Big Data Engineer,2300000,INR,27611,IN,100,IN,L
2021,EN,FT,Data Engineer,2250000,INR,30428,IN,100,IN,L
```

We imported pandas as we needed it to load the csv file
file_path = 'DataScience_salaries_2024.csv'
df = pd.read_csv(file_path)

Then we used `df.head()` to get a better visualization of first 5 brackets to understand the data we are working with:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	compa
0	2021	MI	FT	Data Scientist	30400000	CLP	40038	CL	100	CL	
1	2021	MI	FT	BI Data Analyst	11000000	HUF	36259	HU	50	US	
2	2020	MI	FT	Data Scientist	11000000	HUF	35735	HU	50	HU	
3	2021	MI	FT	ML Engineer	8500000	JPY	77364	JP	50	JP	
4	2022	SE	FT	Lead Machine Learning Engineer	7500000	INR	95386	IN	50	IN	

2. Data Understanding

From the first glance, we can see that there are not just 1 year but multiple years. The salary is the annual salary. There are 11 data types that we are working with: work_year, experience_level, employment_type, job_title etc

To further understand and visualize the data, we can look more into how the data collection and structure looks like

2.1. Data Handling and Cleaning

Since we did not get any errors when getting an overview of the data, we have discovered that there were no inconsistencies, errors or missing values inside the dataset. So this part was pretty easily to deal with as data preprocessing was not required

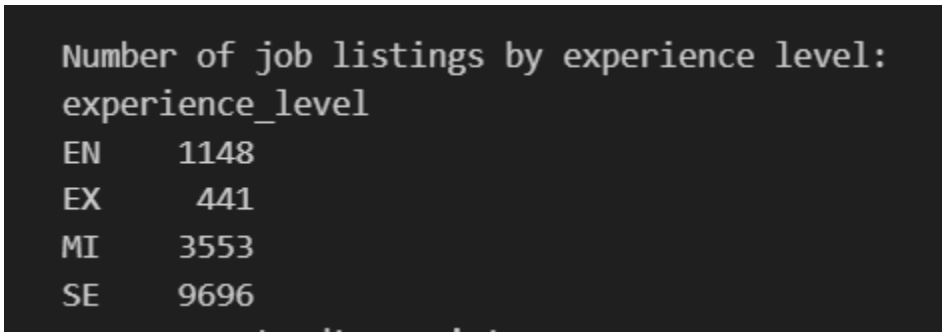
2.2. Data Collection and Structure

For this we have to create multiple dataframes first which will be useful for getting, average value, number of indexes, different plots depending on the dataframe.

job_listings_by_experience_level = df['experience_level'].value_counts().sort_index()print("\nNumber of job listings by experience level:")print(job_listings_by_experience_level)

For example, here we are counting the number of job listings in the DataFrame (df) based on different experience levels, then sorting the counts in ascending order and printing the result, showing the number of job listings for each experience level.

Which when running the code, will give this output:



From here we can see that there are 4 different experience levels. Entry, Mid, Senior and Executive. Even if you combine all other experience levels, it does not match up to how many senior experience required job listings there are. This is an indication that our average salary will be inflated to a degree due to this.

```
Total number of job listings: 14838

Average salary: $149874.72
```

As we can see, there are 14838 listings and due to aforementioned reason, our average salary is pretty high, standing at 149874.72 USD

The other data types we looked into first was number of job listings by employment type:

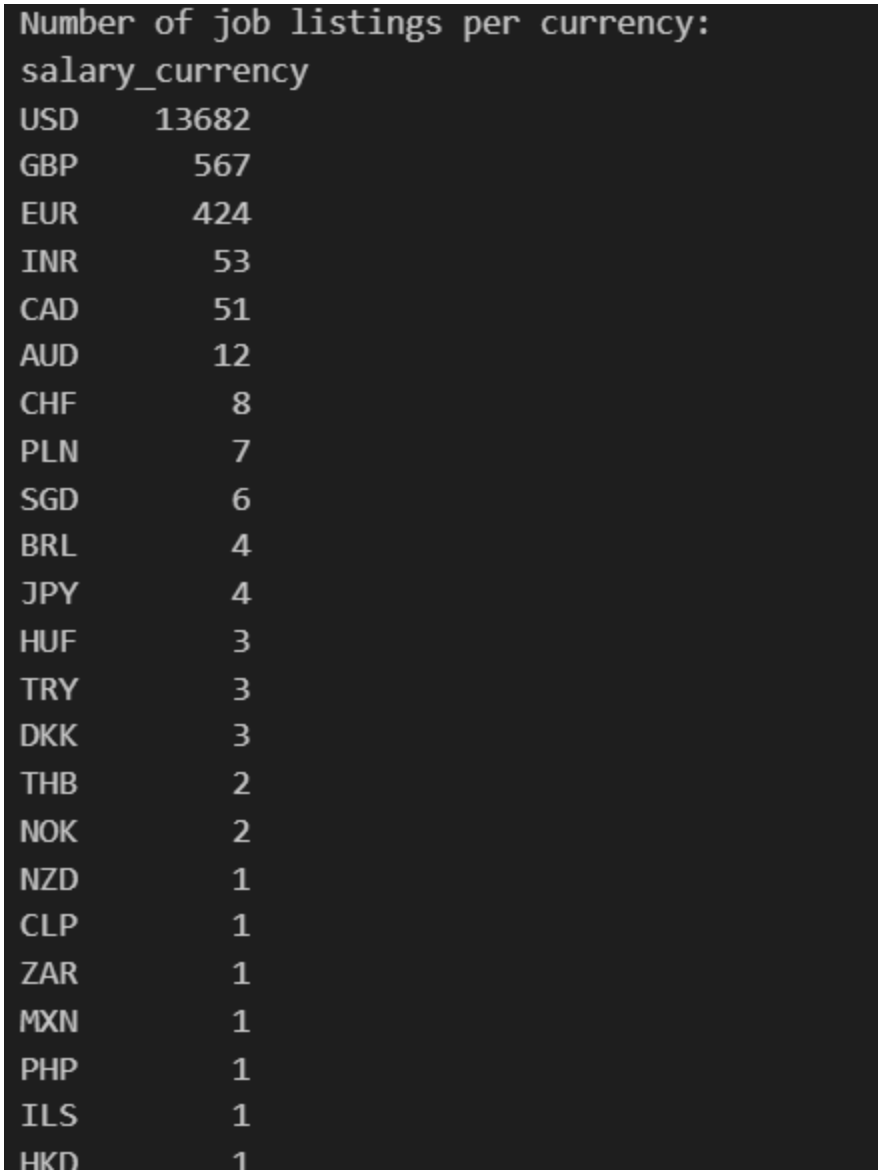
```
  Number of job listings by employment type:
  employment_type
  CT       26
  FL       13
  FT    14772
  PT       27
```

Here we have 4 different employment types.
Contract, Freelance, Full Time and Part Time. Due to the overwhelming majority of the employment types being full time. The price data is pretty representative with regards to full time.

Job listings by employee residence:

```
Number of job listings by employee residence:
employee_residence
AD        1
AE        5
AM        2
AR       11
AS        1
      ...
UG        1
US    12926
UZ        3
VN        6
ZA       15
```

There were too many employee residence but again the majority were from USA



And most currencies were also in USD. But even in different currencies would not be a problem as we have a data type showing all currencies in USD

And finally we can check job listings per year:

```
Number of job listings per year:
work_year
2020      75
2021     218
2022    1652
2023    8519
2024    4374
```

We can see that the given job listings are increasing every year. Most probable reason for having less job listings being posted in 2024 with relation to 2023 is most likely due to year 2024 not being over it so not all job listings could not be posted yet.

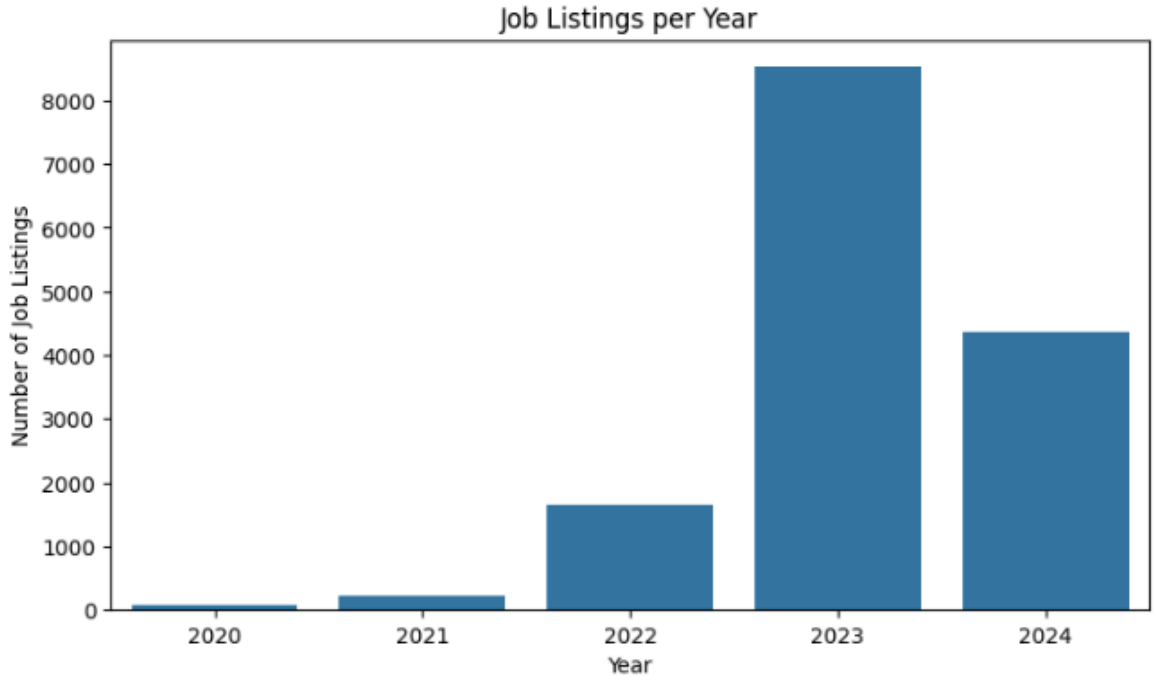
2.3. Data Visualisation and Summary

This kind of data looks a bit complicated to read so we decided to turn all of these into column plots to get a better understanding and it would also be easy to read and understand

```
fig, axs = plt.subplots(2, 2, figsize=(20, 12), gridspec_kw={'hspace': 0.5, 'wspace': 0.3})
plt.subplot(2, 2, 1)
sns.barplot(x=job_listings_per_year.index, y=job_listings_per_year.values)
plt.title('Job Listings per Year')plt.xlabel('Year')plt.ylabel('Number of Job Listings')
```

Here we are creating a 2x2 grid of subplots using `matplotlib`, setting the figure size and changing the spacing between subplots. Then we select the first subplot and create a bar plot using `seaborn`, passing in the index and values of `job_listings_per_year` as the x and y. The subplot is then given a title, and labels for the x and y axis.

This is how this would look. Only this specific subplot out of the 4 that are displayed:

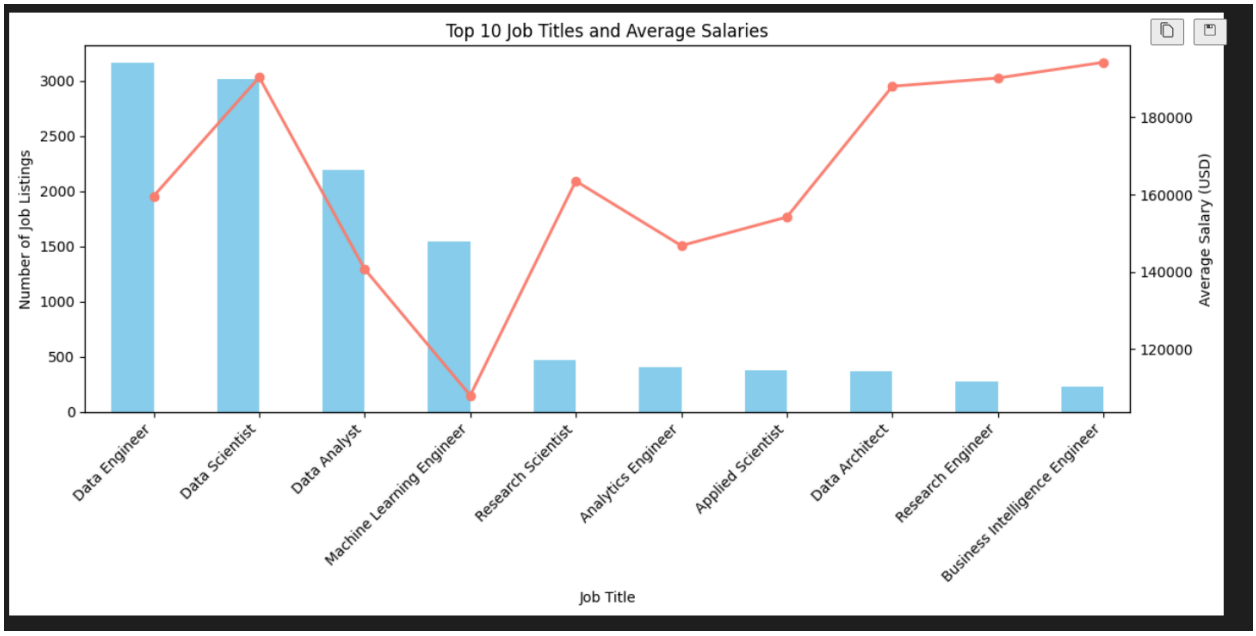


The same process was done for all other 3 dataframes. The only outlier was with Job listings by currency. Since there were too many currencies we decided to use the top 10 most common with `.nlargest(10)`

We have decided to show job listings per employee residence and company location in different plots. The main reason was we wanted to see if the employees were working in the same country as their company location or not. Due to the image being too big I am unable to put it here, but it is given in Jupiter and we can see that employee residence overlaps with company location so this should not be an outlier for us. The code for creating these two subplots was done in the same way as the ones before.

The next problem we wanted to tackle while visualizing the data was that, the name for each job listing more often than less used to be different. So we wanted to see what was the most common job listing. However to make it more interesting, we decided to overlap the top most common job listings with their average price to get a column chart and plot line at the same

time:



From here it is easy to notice that the most common job listings are Data Engineer, Data Scientist, Data Analyst and Machine Learning Engineer. It seems the lowest is Machine Learning Engineer with below 120K USD while the highest is Business Intelligence Engineer. But is it important to note that too few listings can also have a higher impact on salaries.

To finalize the plots we have decided to come up with final 3 different subplots for average salary by year, experience level and company size. This way we can also see correlation with each data type to see if there also might be any outliers like in previous plots.

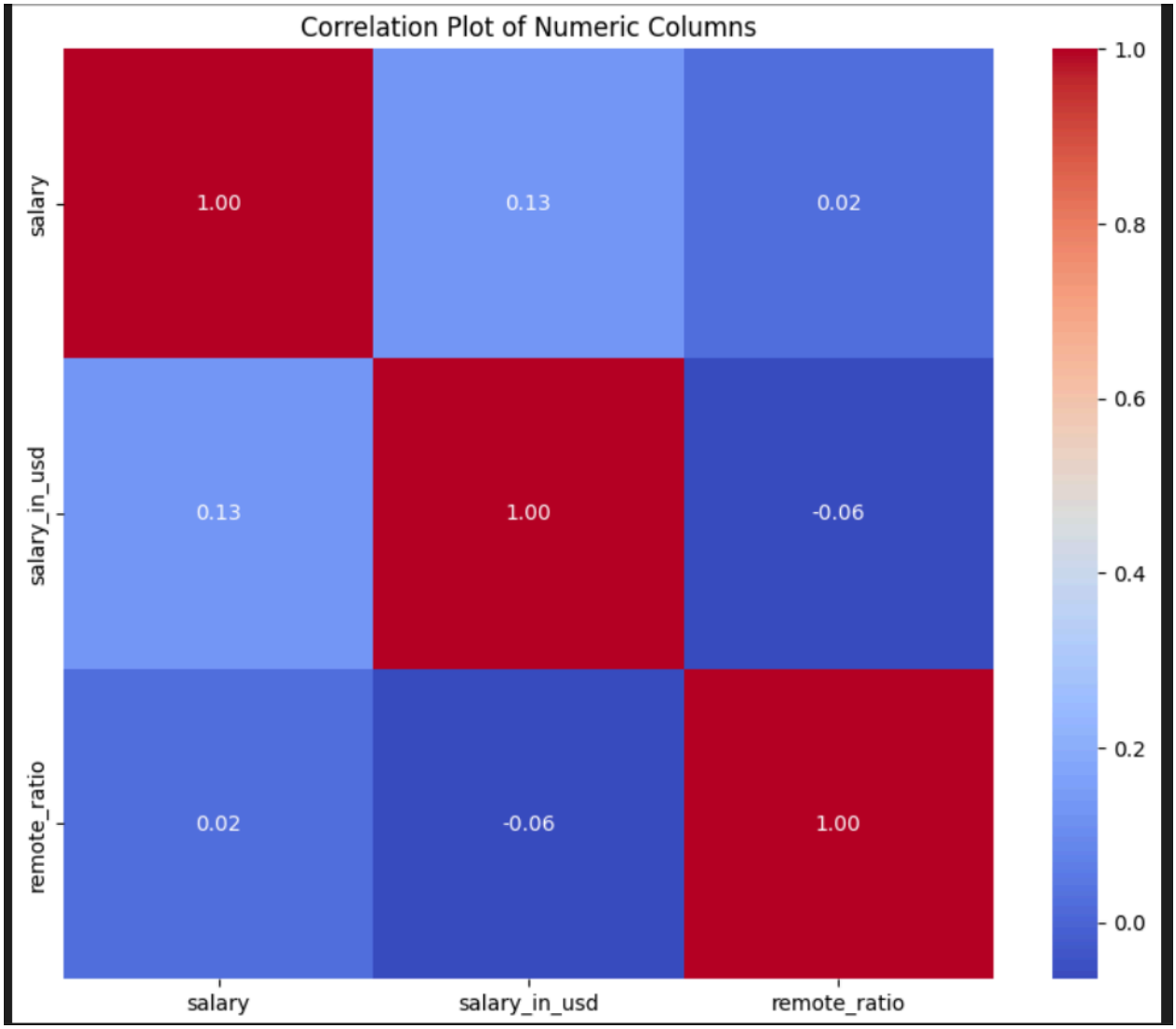
Here we can see the pattern of increased salary year on year and halting around 2024. But since there are too many factors it is safe to assume that the year can not be the only major data influencer here. Same goes for experience level. There were no significant outliers here since we can see the lowest salary is Entry level being less than 100K USD.

Then comes Mid level with just above 120K USD. We then move on to Senior level around 160K USD and the unsurprisingly executive level is paid the most by average salary being above 180K.

2.4 Correlation Plot

Correlation Plot is often useful for both EDA and also for training/testing phase where we can quickly access which data types would be best to utilize, however in this case since we don't have lot of features, and most are not integers, correlation plot is not really useful.

Since we were only able to use salary, salary_in_usd and remote ratio



2.5 EDA - Conclusion

From our analysis, we can conclude that there are several key insights from the dataset. The average salary varies greatly based on job titles, experience levels, and type of employment. Most job listings are for senior-level positions, which likely inflates the average salary. Full-time employment is the dominant type, and most listings are from the USA, with salaries mainly listed in USD. Job listings have increased over the years, though 2024 has fewer due to it not being complete. Finally, the highest salaries are for executive-level positions, while entry-level positions have the lowest.

The next objective is to use the price as the target value for our machine learning model. We aim to predict salaries based on the data we have. This involves selecting relevant features from the dataset and training the model. We are planning to focus on job titles, experience levels, company size, and location. By doing this, we hope to accurately forecast salaries for different roles and conditions.

3 Linear Regression

3.1 Train/Test Split

Before starting to train models we have to decide how to split our dataset. In this case, we decided to use a Train/Test split of 70-30%

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

As a note, we use random state so the randomized set is consistent in all runs, without changing

3.2 Target Value

Since our whole project revolves around getting information and training models based on data science job listings, it would make sense to base the target value for salary. To be more specific, in this case, we chose salary_in_usd. Since there are 2 datasets with salaries and the other “salary” contains all salaries even the ones that are in different currencies. Since this would be a problem both in training and plotting, it is easier and better to use ‘“salary_in_usd”, since this way all salaries have already been converted to dollars.

```
target = 'salary_in_usd'
```

3.3 Features

For features it becomes a bit tricky to which ones to use for training the model and which ones to opt out from. The below description is from mostly finalized version of what we have decided on. Since some models can capture more complex patterns in contrast to others, so in some

cases it makes more sense to give more features to train, even on the correlation plot they score awfully. We will first start training Linear Regression model, so after a few trial and errors we decided to use 'work_year', 'experience_level', 'employment_type'. However since the data types we are working with are not all integers, we would either have to convert strings into integers/float or use One-Hot encoder. In this case we decided to go with the latter. Using pandas library it is fairly easy to implement categorical features.

```
X = pd.get_dummies(X, columns=['experience_level', 'employment_type'], drop_first=True)
```

Here we use drop_first=True, as it helps in reducing the extra column created when using get.dummies, so it reduces the correlations created among dummy variables.

Note: we did switch to specifically using OneHotEncoder from scikitlearn in models after Linear Regression

4. Models

Firstly, we wanted to use the Linear Regression model, then help the model performance with Ridge and Lasso regularization (L1, L2) to see how this would affect the result. As a note, Linear Regresison assumes a linear relationship between the input features and the target variable.

Ridge adds a penalty equal to the sum of the squared coefficients to the loss function, which shrinks the coefficients and reduces overfitting

Lasso adds a penalty based on the absolute values of the coefficients, which can shrink some to zero, effectively selecting features

Linear Regresion formula is y = a + bx
y is the predicted value of target variable (price)
x is the input feature (work_experience, etc)
a is the value of y when x is zero
b is the coefficient

4.1 Evaluation Metrics

Since we are working with a regression-based problem. We decided to use R2 accuracy and Root Mean Squared Error (RMSE). We will be using these 2 metrics in all sets (Train, Test) to see how the metrics change in each phase.
RMSE Formula –

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

n - number of data points/total number of instance
yi - actual value of the target variable

yi^ - predicted value of target variable

R2 Formula - (1-(SSR/SST)
R2 detects accuracy so here the result we get from dividing **squared differences between actual and predicted value** by **total sum of squares** (how much each actual salary differs from the average salary). And we deduct this from 1. If the value is closer to 1 then the model is performing really well, if it is closer to zero, then it is really bad

4.2 Scaling

Here we use the StandardScaler to ensure all features have the same scale, which helps linear models like Linear Regression perform better by treating each feature equally. However as it can been in the jupiter, we created 2 different codes, one with and another without scaler, yet no change was seen with or without using Standart Scaler.

4.3 Results

After running the Linear Regression Model and its regularizations, we obtained these metrics.

Regression Model -

```
Train RMSE: 61772.17
Train R2: 0.20
Test RMSE: 61307.43
Test R2: 0.20
```

Ridge -

```
Train RMSE (Ridge): 61772.17
Train R2 (Ridge): 0.20
Test RMSE (Ridge): 61307.41
Test R2 (Ridge): 0.20
```

Lasso -

```
Train RMSE (Lasso): 61777.18
Train R2 (Lasso): 0.20
Test RMSE (Lasso): 61292.56
Test R2 (Lasso): 0.20
```

As we can see even after using the regularizations, there were no significant changes and the results stayed fairly same and equally bad in all circumstances. We also increased/decreased alpha value in both regularizaitons but 1 was the one giving best result. This can also be due to having very few “useable” features since using all features creates even worse performance due to there being very little to no correlation between most features and the target value (price).

4.4 GridSearch

We will try hypertuning on Lasso and Ridge (alpha) to see which would be the best to use

```
grid_search = GridSearchCV(lasso_model, param_grid, cv=5, scoring='neg_mean_squared_error')
```

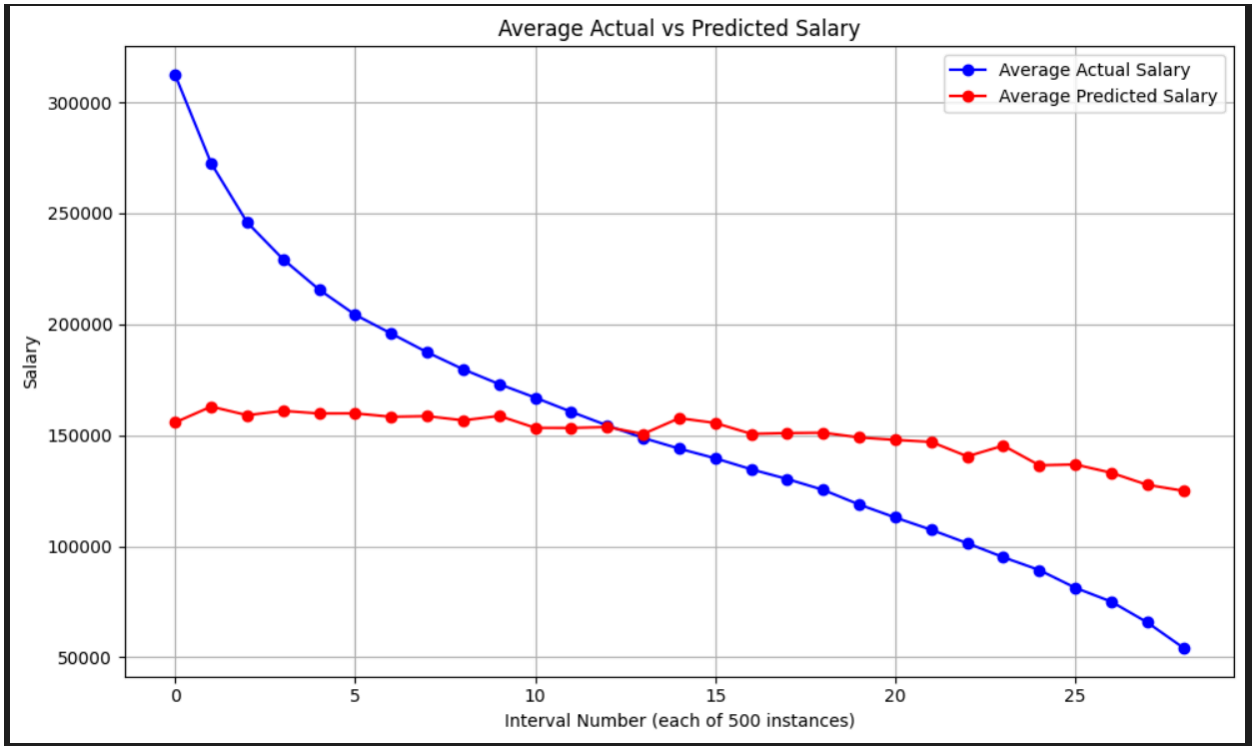
CV here is cross validation and number 5 indicates that we are splitting the data into 5 parts. The model will train on 4 parts and validate on the remaining part, repeating this process 5 times, each time with a different part as the validation set
The GridSearch is done same way for both regularizations

```
param_grid = {
    'alpha': [0.1, 1.0, 10.0, 100.0, 1000.0]
```

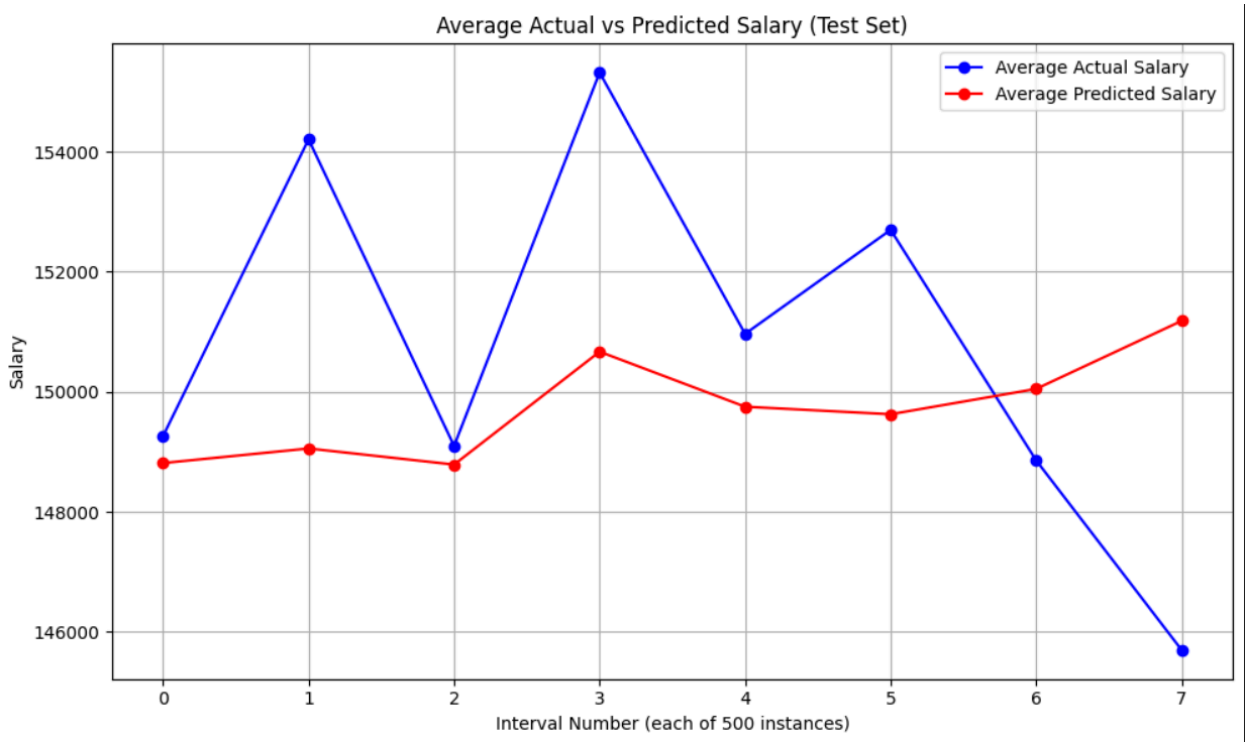
Afterwards for Ridge best alpha was 0.1 and for Lasso 100, however there was no significant impact on the result

4.5 Plotting Results

Plotting the results in the beginning was a bit tricky since there were too many datasets and just showing a scatter plot would be of no use to understand the prediction as better as possible. So instead we wanted to use a plot line. But for this we would have to take the averages. Since we can't take averages of the years (there are only 4) we decided to take averages of every 500 and next 500 instances. Ultimately being able to show the whole dataset by average salary. Here is the result:



From the first glance it is obvious that the model is not being able to capture the needed trend lines. Furthermore, we plotted this way and found out the data placement is a bit weird, as it just keeps going down (salary). Another point is, this is done for the whole dataset, so we wanted to check it for the test set as well.



Here the issue is opposite, since we are only using 30% for Test, we are not able to clearly see or catch the full picture. So we decided to use scatter plot as a visualization for Test set

As expected, the model is unable to capture most of the needed data points

5. Decision Trees

The next models we will be using will be from the Decision Tree category. Ultimately we have decided to test with Random Forest. But we will be testing first with just default model then use it with hyperparameters to see if there would be any noticeable changes.
Decision trees make decisions by breaking down data into smaller subsets based on specific criteria, creating a tree-like structure where each branch represents a decision based on input features.

5.1 Random Forest

Random Forest works by combining multiple decision trees during training, where each tree is built on a random subset of data and features, and then averaging the predictions of these trees to improve accuracy and reduce overfitting. But before implementing the model we again tested with couple more features and found out that, even though some features really badly effect performance in Linear Regression models, they do sometimes do the opposite when it comes to

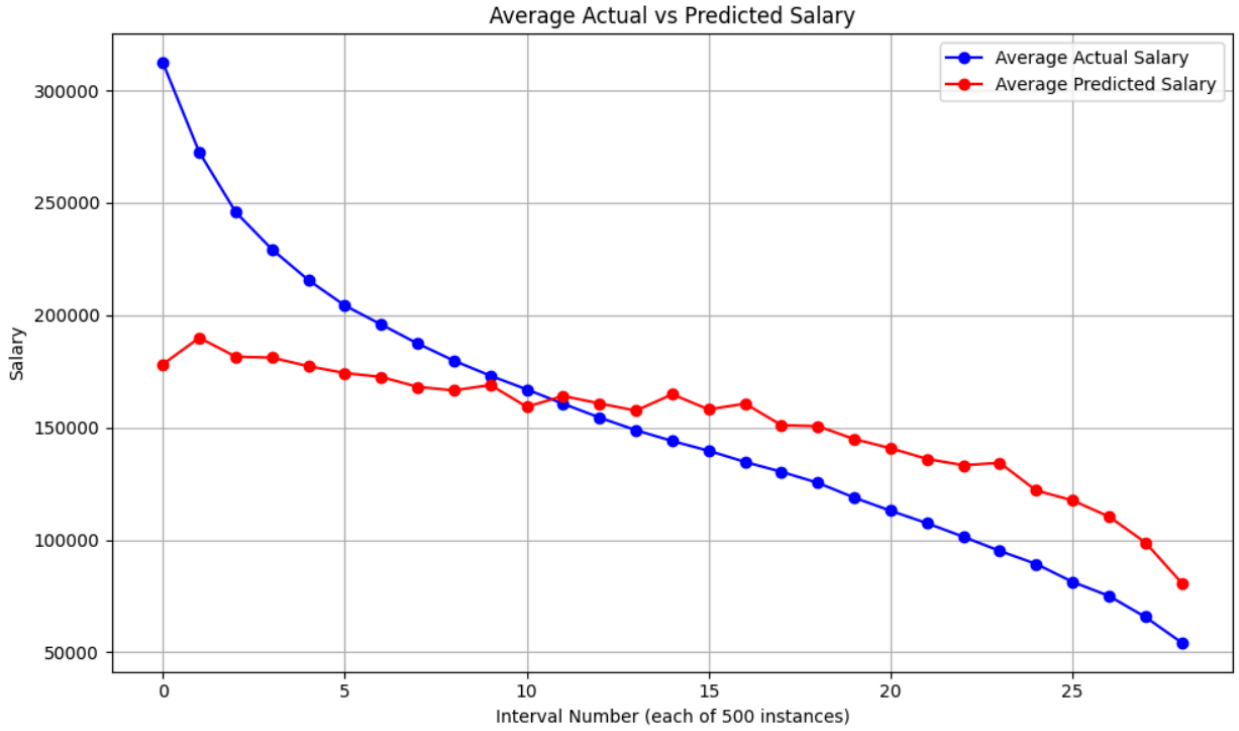
Decision Trees and actually make the performance better. This is most likely due to Decision Trees being able to capture a bit more complex patterns. So for this model we are using:

```
features = ['work_year', 'experience_level', 'employment_type', 'job_title', 'salary_currency', 'employee_residence']
target = 'salary_in_usd'

X = df[features]
y = df[target]

X = pd.get_dummies(X, columns=['experience_level', 'employ (parameter) test_size: Float | None ncy', 'employee_residence'], drop_first=True)
```

For hyperparameters, we are only specifying the number of estimators and putting it to 100. Now the results are slightly better than Linear Regression:

```
Train RMSE: 52772.18
Train R2: 0.42
Test RMSE: 56371.71
Test R2: 0.33
```

We are also able to see slight changes in plottings as well



5.2 RandomForestRegressor

Now we wanted to get a bit more in-depth to random forests by adding parameters

Before adding new hyperparameters, we made slight changes to the code. The first was changing get.dummies from pandas to OneHotEncoder from scikitlearn, and the other was using a pipeline.
In this context is to capture the entire workflow into a single object. Which gives us a more easier handling process. Another reason was that if overfitting is occurring, the pipeline will deal with it since it does the processes sequentially, this prevents data leakage so we do not get over-optimistic results.

```
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(
        n_estimators=100,
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        random_state=42
    ))
])
```

Worth noting that in preprocessor we are only including OneHotEncoder for categorical features, as any other preprocessing was not needed so far.

5.3 Difference between Gridsearch and Manual

We did try GridSearch to hyper tune the hyperparameters

```
param_grid = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(model_pipeline, param_grid, cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
```

CV here is cross validation and number 5 indicates that we are splitting the data into 5 parts. The model will train on 4 parts and validate on the remaining part, repeating this process 5 times, each time with a different part as the validation set
N_jobs setting this to -1 means using all available CPU cores to get the result as fast as possible
Putting verbose to 2 to get more detailed response (as example, if it was 0, it would not tell me what are the best hyperparameters to use)

Neg_mean_squared_error here the negative sign is used because, by default, higher scores are considered better in Scikit-Learn, and we want to minimize MSE.
And this was the result:

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters found:  {'model__max_depth': 30, 'model__min_samples_leaf': 2, 'model__min_samples_split': 10, 'model__n_estimators': 300}
Train RMSE: 54436.27
Train R2: 0.38
Test RMSE: 56020.29
Test R2: 0.33
```

When trying manually:

```
n_estimators=100,
max_depth=None,
min_samples_split=2,
min_samples_leaf=1,
random_state=42
```

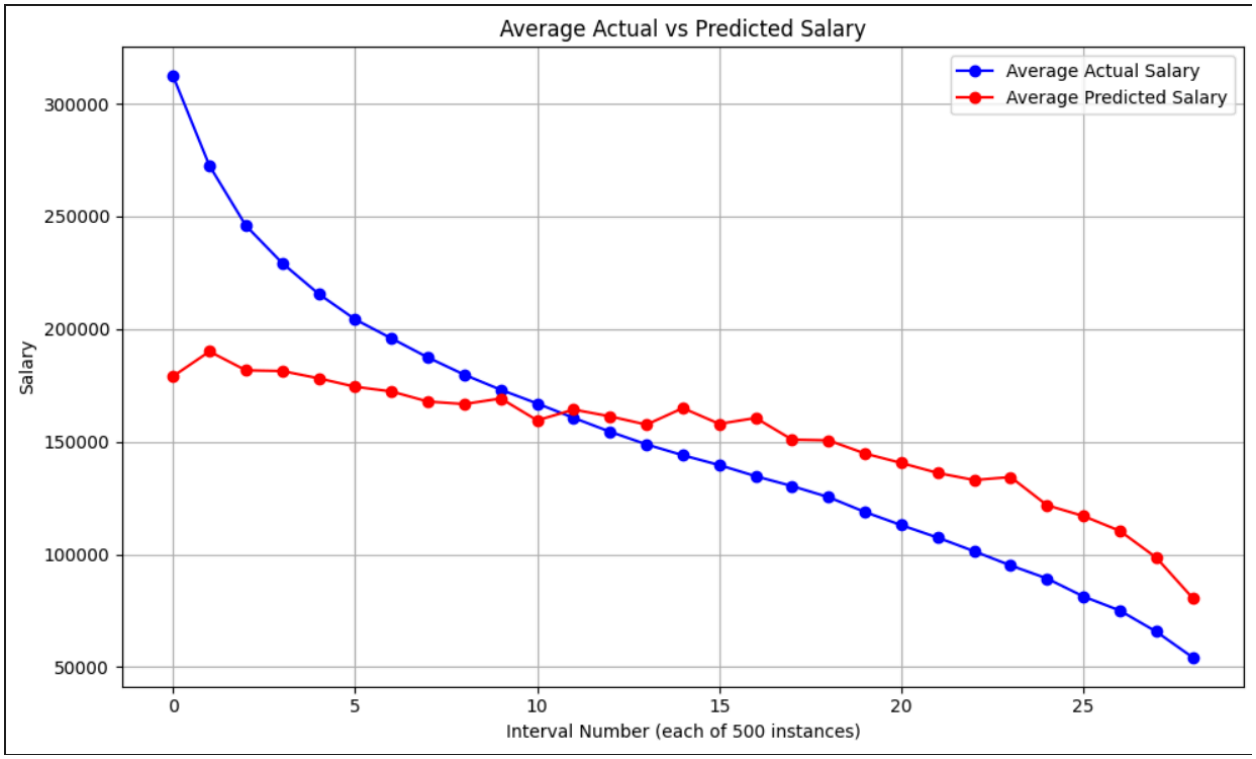
Here we manually tried with different hyperparameters to try to get the best ones.
N_estimator is the number of trees in the forest (number of decision trees that make up the ensemble)
Max depth is the maximum depth of each tree, so here using none means it will grow till the leaves contain less than min_samples_split
Min_samples_split means the minimum number of samples/branches required to split the internal node so the depth will stop when there are only 2 left
Min_samples_leaf is minimum number of samples a leaf note must contain, too high can lead to underfitting

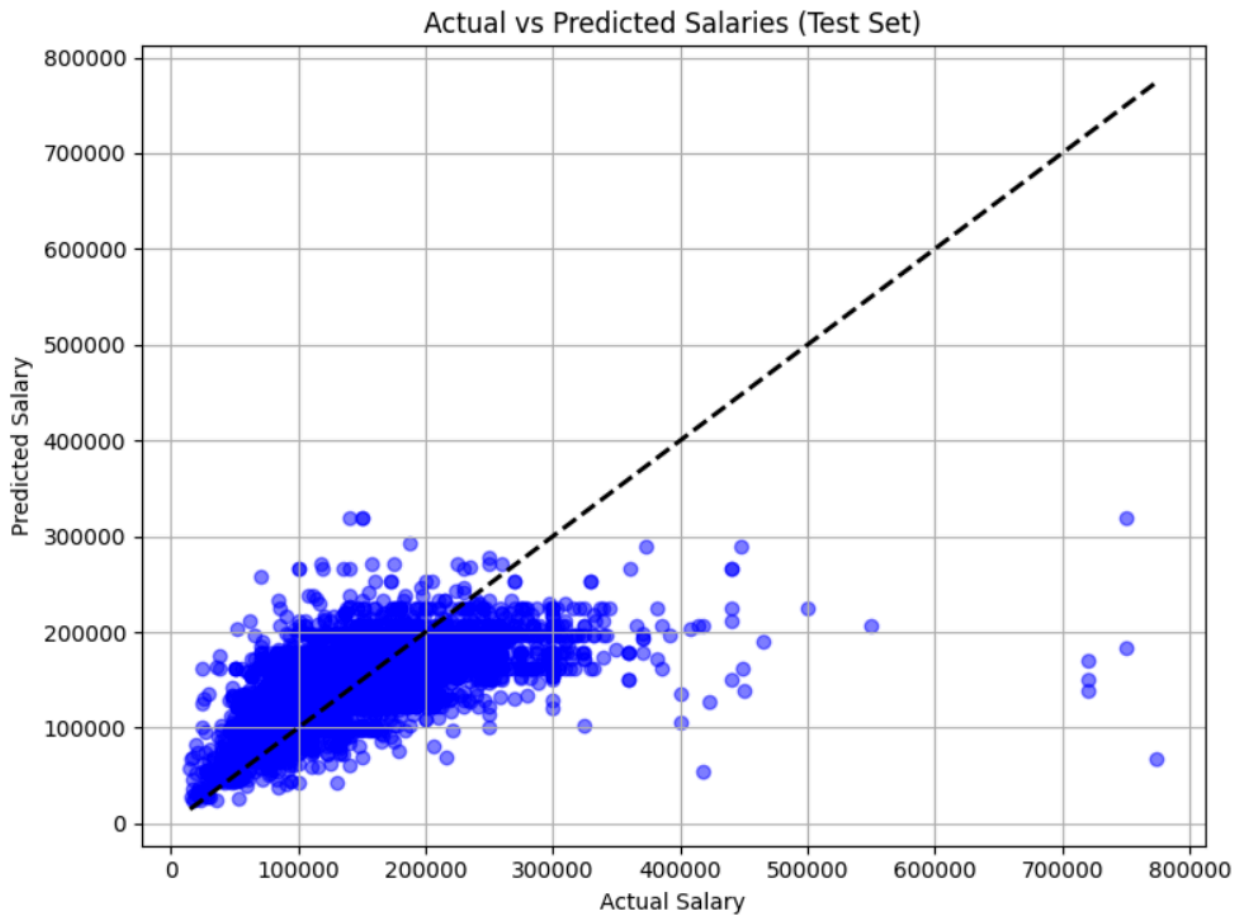
And the results were:

```
Train RMSE: 52772.18
Train R2: 0.42
Test RMSE: 56371.71
Test R2: 0.33
```

The big difference between the manually trying hyperparameters and gridsearch was that Our Train set was the highest with 42% while Gridsearch was able to get a lower test set error. However this is just due to our own method just overfitting.

Regardless, this is the visualization/plotting we get after finishing the model:





Since there weren't big differences in evaluation metrics between Random Forest and the same model with more hyperparameters (and different encoding systems and a pipeline), the line and scatter plots are also similar due to this.

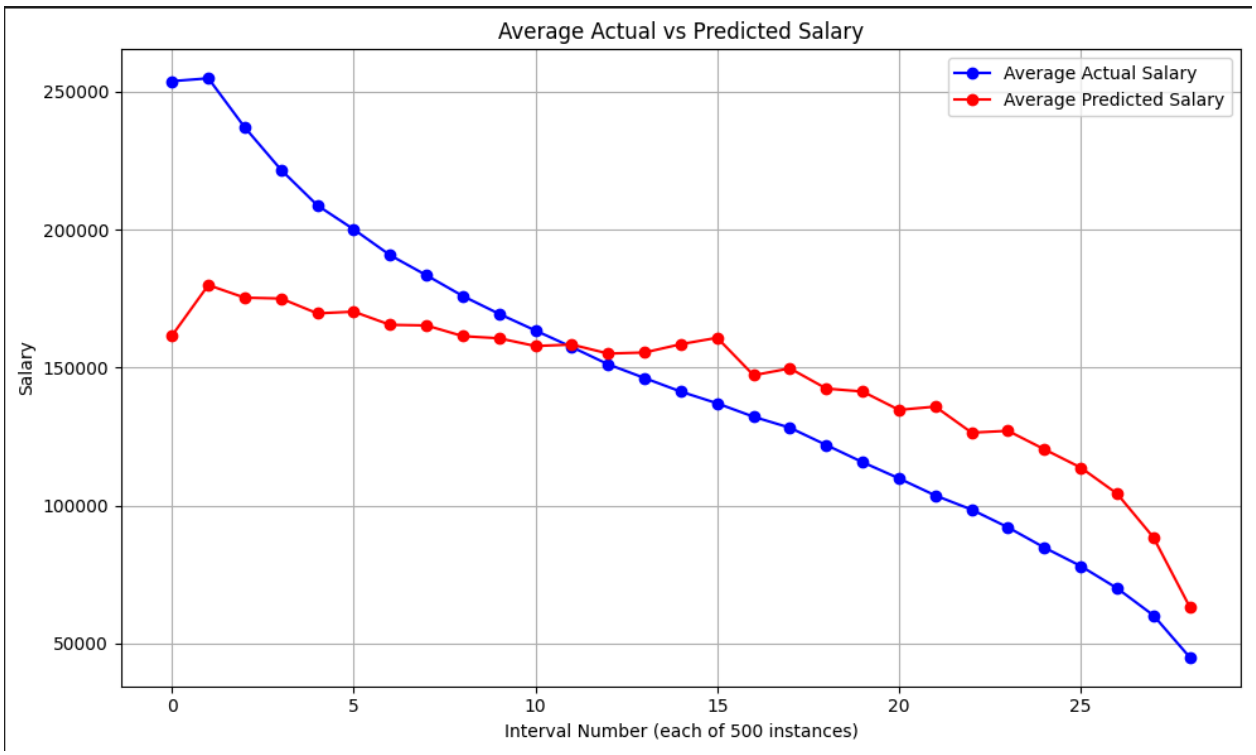
5.4 Removing Outliers

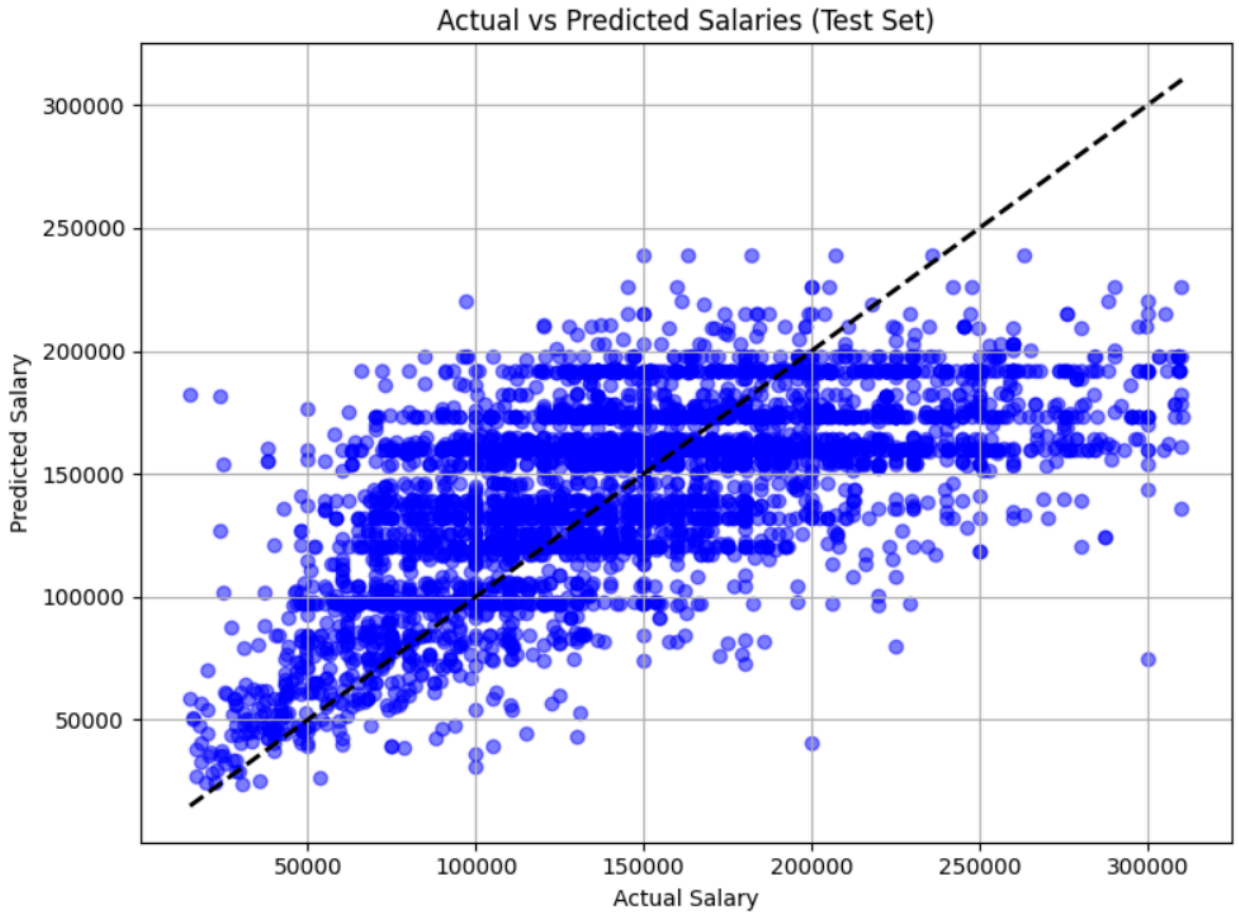
We want to remove extreme outliers so it does not damage our model and increases the result. So we are using the IQR method which is the range between the 25th and 75th percentiles of the data. Any data points lying more than 1.5 times the IQR below the 25th percentile or above the 75th percentile are considered outliers and are excluded from the dataset.

```
Q1 = df[target].quantile(0.25)
Q3 = df[target].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df[target] >= lower_bound) & (df[target] <= upper_bound)]
```

And there were slight improvements with the results which also affected the plots as well

```
Train RMSE: 44926.35
Train R2: 0.43
Test RMSE: 46264.28
Test R2: 0.39
```





In Both plots we can see the effect ourselves, since the extremes were excluded, the plots especially scatter plots adjusted better to know results.

5.5 XGBoost

This model works by sequentially building new models that predict errors of the previous models, aiming to minimize the overall prediction error. It combines the predictions from these weak learners to create a strong predictive model. In theory, XGBoost should be able to give better performance and detect higher complexity.

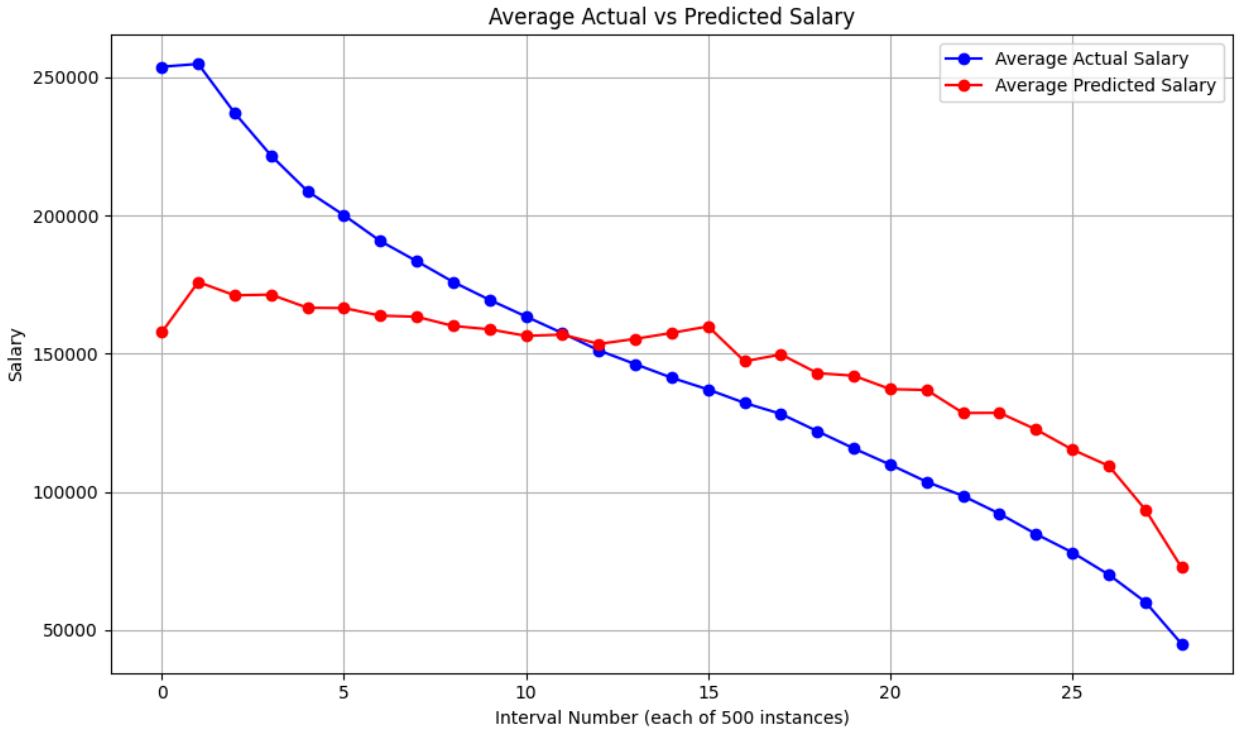
Since both of these models are in the same category, we did not change anything regarding features
With hyperparameters the only new one we are using is
Learning_rate which is how fast should the model learn, higher learner rate will give output faster however also potentially lose on good results. While a low learning rate would do the opposite.
In this model we manually tried with different hyperparameters and decided in the end to use 1000 estimators, max depth as none, learning rate at 0.01.

5.6 Results

The biggest change was that we reduced overfitting with XGboost, relative to RandomForest.
While in RandomForest R2 score for train was 0.45, test set at 0.38.
With XGBoost however R2 score is around 0.42 for train and 0.4 for test set

```
Train RMSE: 45292.78
Train R2: 0.42
Test RMSE: 46218.45
Test R2: 0.40
```

However there weren't any major differences in line and scatter plots



6. Conclusion

Based on our work and exploration of data science job salaries, we found that salaries vary significantly based on different data types, and some datasets badly affect some models while helping others. Using machine learning models like Linear Regression, Ridge, Lasso, Random Forest, and XGBoost, we aimed to predict salaries with different results and success. In all the models the success was still relatively low, this can be due to only being able to work with at max 7 features at once which is too low, also since there aren't much numerical features while our target value is numerical which does impact the result as it crushes the correlation between different features. Ultimately, XGBoost showed the best performance, reducing overfitting and offering improved predictive results relative to other models.