UNIVERSITY OF WATERLOO

Spark's MLLib - Salary Prediction Based on Job Postings

Group 16

Dahiya, Bijender

Mak, Colton

Perez Martinez, Arely

Shankar, Arvind

Varkey, George

Batara, Sarah

Objective

The objective of this report is to leverage Spark's Machine Learning capabilities for predicting salaries based on US job postings. This entails acquiring proficiency in utilizing Spark's library for pipeline preprocessing and implementing Machine Learning models. Given Spark's widespread adoption as an industry-standard tool for handling large datasets, acquiring expertise in its application is crucial for effective data analysis.

There is nothing more important to workers than the pay they receive for their hard work and dedication. However, as we know, many factors can determine the level of salaries. For example, we could expect that larger companies with higher revenues would offer higher salaries for the same job in a smaller company. Location and cost of living should also determine the level of pay that we receive. On the other hand, it is expected that the type of job performed or the skill sets required will determine the level of pay.

Data Preparation

Data was obtained from Kaggle competition.
(https://www.kaggle.com/competitions/salary-prediction-for-job-postings/data)

The dataset had 33,000 data points with the following features:

- ID -ID of the job posting
- Job - Job name
- Jobs_Group - Jobs grouped in a few categories
- Profile - Lead/Senior/Junior or none
- Remote - Remote/Hybrid or none
- Company - Company name
- Location - the language the book is written in
- City - City
- State - State
- Frequency_Salary - year/month/week/day
- Mean_Salary - mean salary (USD)
- Skills - Skills, titles, certification, etc required
- Sector - Sector Company
- Sector_Group - Sector company grouped
- Revenue - Revenue size of the company
- Employee - Number of employee size of the company
- Company_Score - Score given by the users
- Reviews - Number of reviews of the Company_Score
- Director - Name of the Company Director
- Director_Score - Score of the company Director
- URL - Company website

The majority of these features are categorical, thus there is significant preprocessing that had to be done to the dataset before it could be fed into a Machine Learning model.

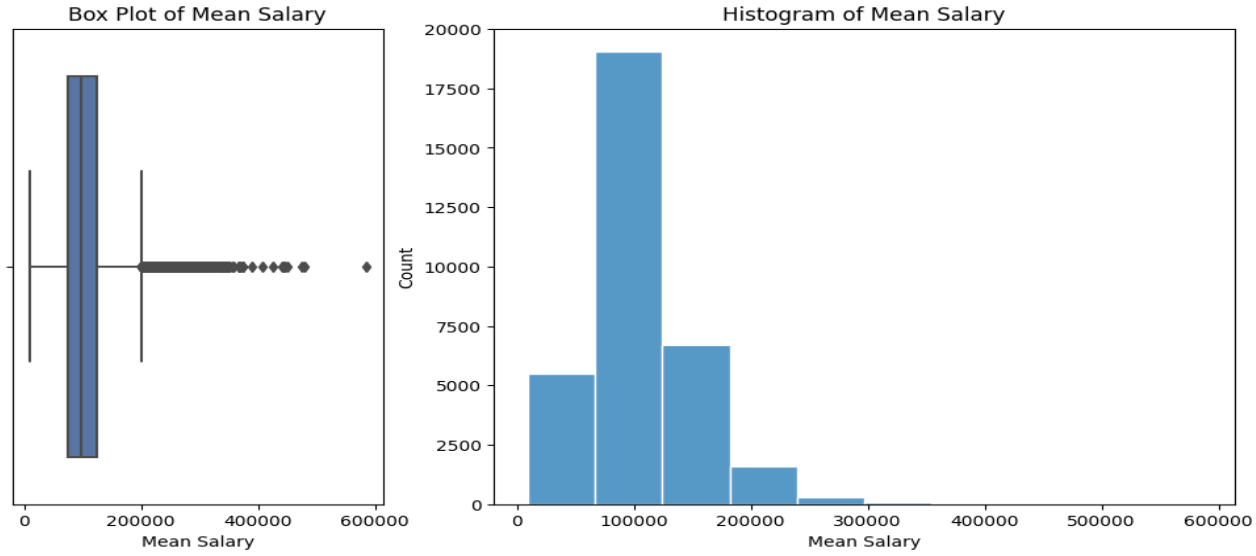
Preprocessing steps executed:

1. Tackling null values
    - A number of the features such as Location, City, State, Sector Group, Remote/Hybrid role had null values for large percentage of its data
    - In each of the columns, they made up more than 50% of the dataset, thus removing them would not be an option as it could potentially skew the result.
    - Different custom functions had to be created to deal with these null values.
    - For example, for a null State, the function created would check the value of Location if it is listed as Remote or generic location such as "United States".The function would assign a new value for Remote : "RM" or "OT" for others so that we do not affect the other datasets if we were to assign a particular state arbitrarily.
2. Handling Free-Text Skills column
    - This is an interesting column as while it is free text, there are characteristics of the column that determine which vectorizer to convert the free text to numeric values
    - Anything can be defined as a skill, for example, "Chinese" was such a skill that was listed once in the dataset.
    - There were a total of 99 distinct skills that were listed. If we were to transform it with One Hot Encoder, it would have created a high-dimensionality dataset that may be computationally too expensive to complete.
    - We do not expect to see stop words such as "the" and "and"
    - Each row could have unlimited skills defined
    - As a result, we picked HashingVectorizer to convert the Skills text. It uses a hashing function to convert words into numerical indices, making the process much faster.
    - However, it comes at the cost of not being able to retrieve the original words from indices.
3. Handling of short Free-Text column
    - There are commonly used words in the Job title such as Business Analyst, Financial Analyst, or Controller
    - We do not expect to see stop words such as "the" and "and"
    - The word count is typically low (less than 10)
    - As a result, we picked StringIndexer to convert Job Title column which is a straightforward way to convert it
4. String Indexer
    - There are 7 features with categorical values, namely Frequency Salary (Yearly, Daily, Weekly etc), and Sector Group (Healthcare, Information Technology), that we needed to transform.
    - The String Indexer is an alternative way of preprocessing to OneHotEncoding. With the number of features and the multiple values within each feature, usage of OneHotEncoding could introduce a high-dimensionality problem.

5. New features
    - We created a new feature that groups US States, based on commonly used Pay Zones (Zone A, Zone B, Zone C and Zone RM (Remote))

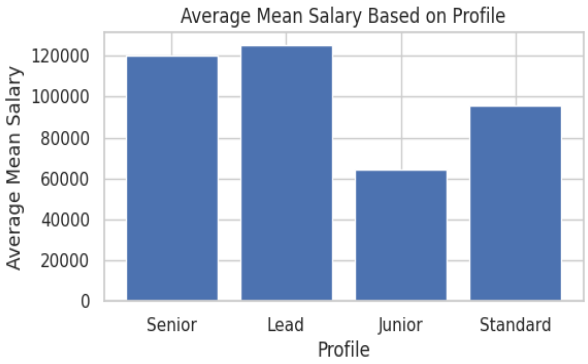
Analysis

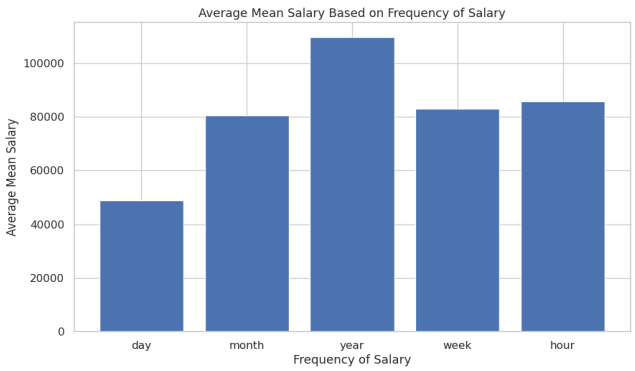
We first observed the distribution of our target variable: Mean Salary. With some outliers greater than $300K, we have decided not to remove them as these cases could represent a portion of the total population as these high salaries also correspond to executive roles such as CIO positions. Also, there are only 78 rows of data having Mean Salary greater than $300,000
Total average of Mean Salary: $104,938. Min: $9,200. Max: $585,000



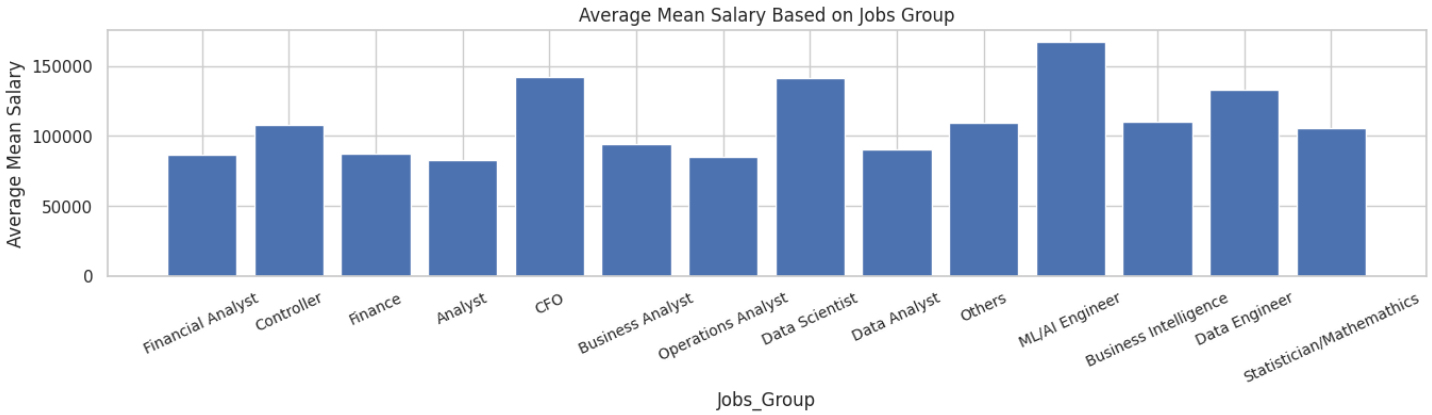
Most of our features are categorical, so we observed their distributions vs our target variable (Mean_ Salary).

From the graphs below (Profile and Frequency of Salary vs Average. Mean Salary), we can see that leaders and senior roles, as well as employees receiving only a yearly compensation, have higher salaries vs junior profiles or employees receiving a daily compensation.

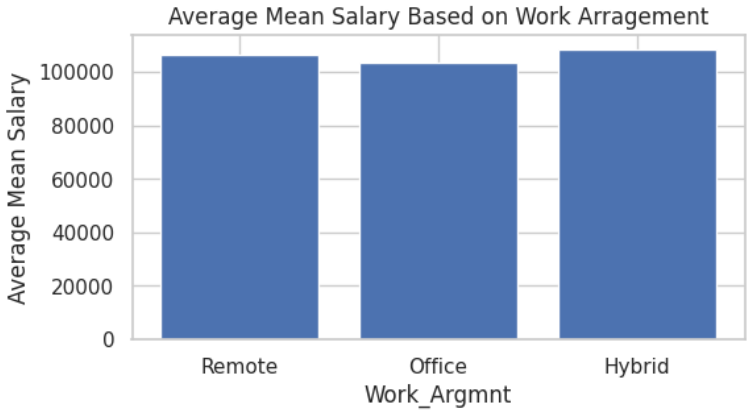




We also observed from our Jobs Group feature (see graph below) that CFOs, Data Scientists and ML/AI Engineers show the highest salaries, contrary to Analysts roles with the lowest average Mean Salary.

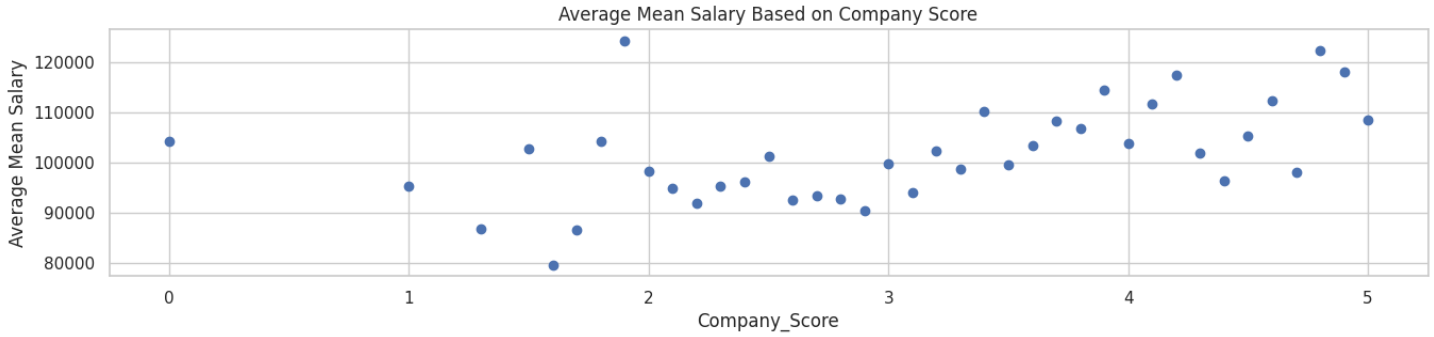


There are other features, like Work Arrangement (Remote, Office or Hybrid) where there is not a clear difference between the average salaries.



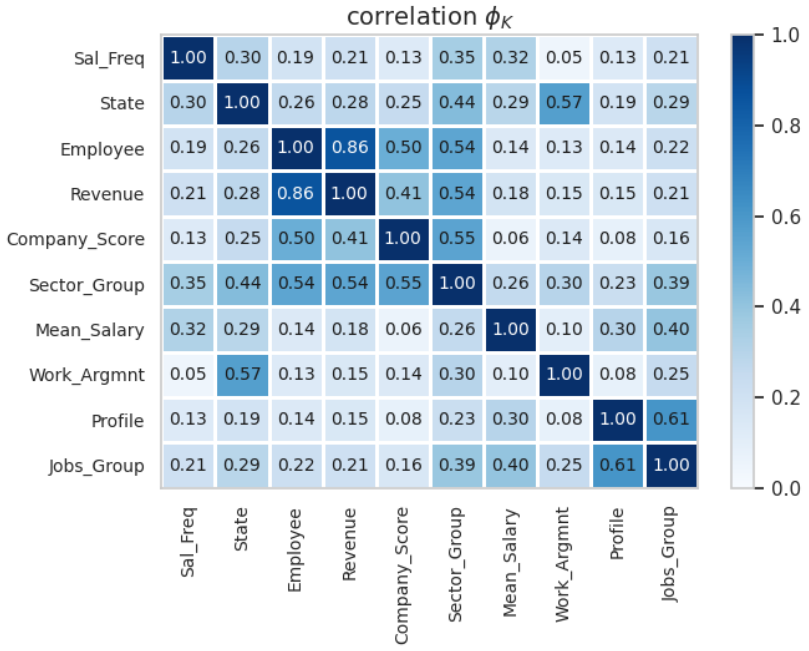
We also have features like Company Revenue and Number of Employees in the company, where there is only a slight difference in average salary for the largest companies with the highest number of employees and the smallest ones versus companies in mid range sizes.

Similarly, It was found that Company Score is a relevant feature that correlates positively at 0.33 with our target variable, meaning that the higher the company score, the higher the salary will be.



Feature Selection

Logically, columns such as ID, number of reviews, the name of directors or director score, as well as URL should not affect the salary, hence they were dropped.



After the data preparation and our data analysis, and further supported by the Phink correlation matrix we decided to exclude the Pay_Zone column due to multicollinearity with the State column. The State column was retained as it had a better correlation with our target variable, Mean Salary.

From our analysis, we decided to include the following features to build our different models:

Jobs group, profile, work arrangement, State, Skills, Sector Group, Company Score, Revenue, Employee, and Salary Frequency

Machine Learning

Pipeline Pre-processing and One-Hot Encoding

Due to the prevalence of categorical variables in the dataset, it was crucial to convert these features into a numerical vector before employing the machine learning model. This conversion was accomplished through the utilization of PySpark's StringIndexer, Imputer and VectorAssembler functions.

By employing the StingIndexer function, we initially transformed the indexed column corresponding to the categorical features. Subsequently, the VectorAssembler function was applied to convert the indexed columns into an independent feature vector.Lastly, we utilized the StandardScaler to scale the feature vector before feeding it into a ML algorithm.

scaled_Ind_Feat	Mean_Salary
(107,[2,3,4,35,79...	115000.0
(107,[0,3,32,35,3...	185000.0
(107,[1,3,35,102,...	84500.0
(107,[0,1,2,3,35,...	111625.0
(107,[0,2,4,80,10...	102690.4
(107,[1,2,35,52,7...	124650.0
(107,[0,2,3,4,35,...	92000.0
(107,[0,1,2,3,4,7...	115000.0
(107,[3,47,52,80,...	106870.0
(107,[0,2,3,35,45...	78500.5
(107,[0,2,79,103]...	89127.5
(107,[0,3,4,38,43...	80748.40000000001
(107,[3,35,80,96,...	68878.5
(107,[0,3,35,83,1...	135900.0
(107,[0,1,3,33,35...	93050.0
(107,[0,35,52,73,...	80850.0
(107,[0,1,3,4,80]...	36800.0
(107,[0,3,4,45,10...	88172.8
(107,[0,3,35,43,1...	120000.0
(107,[3,35,45],[0...	119072.5

Train - Test Split

The entire dataset was then split into Training and Test datasets (80% for Train and 20% for Test) using the randomSplit function.

Running ML Model and Performing Model Comparison

We chose to perform our analysis on three different model types: linear regression, random forest, and decision tree and obtained the following results:

Performance Metric	**Linear Regression**	**Random Forest**	**Decision Tree**
MSE:	1342954359.00	1044664656.24	1317326501.34
MAE:	26942.13	23501.76	26560.14
RMSE Squared	36646.34	32321.27	36294.99
R Squared	0.3057	0.4599	0.3190
Explained Variance	568239047.40	659048344.29	607456885.22

After the initial model training, we performed hyperparameter tunings for potential model improvements and obtained the following results:

Performance Metric	**Linear Regression**	**Random Forest**	**Decision Tree**
MSE:	1410063797.75	955611683.97	1267277566.23
MAE:	27726.72	22062.17	5467.79
RMSE Squared	37550.82	30912.97	35598.84
R Squared	0.2846	0.5152	0.3570
Explained Variance	564518487.31	835600608.46	945523007.50

Both the random forest and decision tree models had improved performance after hyperparameter tunings, shown by the decreased MSE, MAE and RMSE values and the increased R Squared and explained variance metrics. In contrast, the linear regression model performed worse after hyperparameter tuning.

From the obtained performance metrics of each model, we can reach the following conclusions:

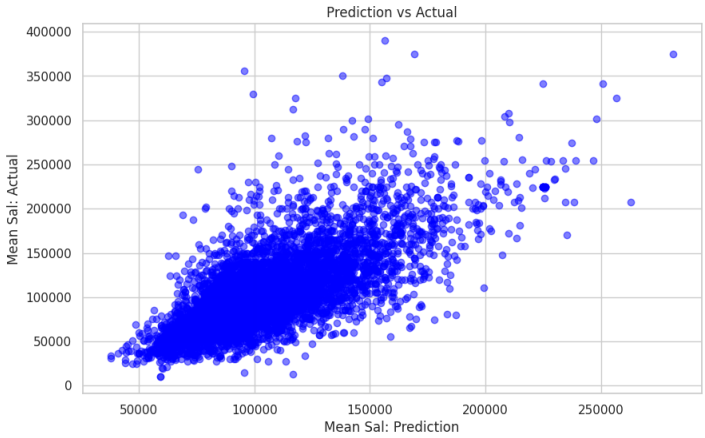
Linear Regression Metrics: The Linear Regression model has the highest MSE and RMSE, indicating a less accurate fit to the data compared to the other models.

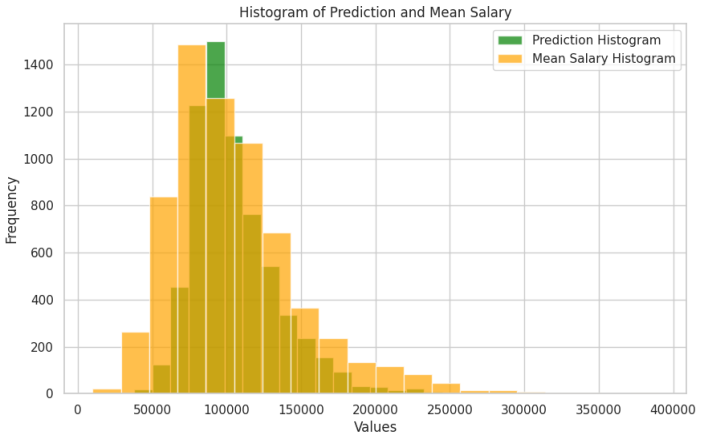
RandomForest Metrics: The RandomForest model performs better, as indicated by lower MSE, MAE, and RMSE values, and a higher R Squared.

DecisionTree Metrics: The DecisionTree model has moderate performance, with R Squared indicating the proportion of variance in the predictable dependent variable.

In conclusion, the **RandomForest** model seems to be the most effective based on the provided metrics

We also plotted the following graphs to visualize the differences between the RandomForest model's predicted mean salaries and the actual mean salaries from the dataset.





Conclusions

In conclusion, we were able to train a RandomForest model on a Kaggle dataset to accurately predict the salary of US workers using a variety of features ranging from the frequency of pay to the employee's physical location.

Although the dataset included a wide range of features, not all of the listed features are effective predictors of an employee's salary. Some features such as the URL of a company's website or the name of the director have an extremely low impact as they logically have little to no influence on salary.

Exploratory data analysis through simple bar plots also showed the distribution of average mean salaries for various categorical features. For example, employees with a senior or lead title have much higher salaries than those with a junior title. Similarly, employees with a yearly salary frequency have higher salaries than those with a daily salary frequency. Thus we can infer that an employee's title and salary frequency are better predictors of an employee's salary.

We chose to perform our analysis with the selected features on three different models: Linear Regression, RandomForest, and DecisionTree. After training each model along with performing hyperparameter tuning, we have determined that the RandomForest model is the most effective in predicting employee salaries based on its performance metrics. It has the lowest MSE, MSAE and RMSE squared values, indicating a lower amount of error. In addition, its R-squared value of 0.5152 shows that the model is a good fit with some predictive power.

While our resulting RandomForest model stands as a good predictor of worker salaries, it is important to acknowledge the limitations of its current scope and the potential for future improvements. Although its performance metrics are better than the Linear Regression and DecisionTree models that we tested, numerous other methods such as Gradient Boosting or Neural Networks have the potential to have better results. Future improvements that are outside the scope of this project can explore more advanced machine learning models along with diverse datasets that include data from countries other than the United States. This can also include real-time data, which could ensure the model's continued effectiveness and relevance in the long-term.

Python vs Pyspark

While working on this project we had the opportunity to explore and work with Pyspark, a computing framework built on top of Apache Spark that has been designed to process large volumes of data. In this particular case, we worked with a relatively small dataset so, either Python or Pyspark are great tools to work with. However, it represented a higher level of complexity and required a learning curve, because it has a different and less intuitive syntax than Python, which also required a deeper understanding of Spark's architecture and concepts.