1. Abstract

The project aims to develop a model capable of predicting expected compensation ranges for any given role in data science or related fields. In recent years, mismatches between employee and employer compensation expectations have been a key barrier to hiring and talent retention, reinforcing the need to develop tools and models designed to address the issue (Nadarajan & Ong, 2024). The dataset used for the training of the linear regression model is sourced from Kaggle, containing 5736 rows of employee data and 11 features capturing various employee attributes such as salary, experience level, education level and location. These are generally known to be key determinants in deciding the total compensation packages of new hires.

Based on the dataset, 2 base models were developed - Linear Regression and Regression Trees for performance comparison. Regression-specific models were chosen due to the desired output of a single value guiding salary ranges. Subsequently, additional complexities were introduced to refine these base models, such as the usage of an elastic net to regularize the Linear Regression, and the usage of gradient-boosting algorithms such as LightGBM and XGBoost for Regression Trees.

Based on the reported R^2 and RMSE Values, it was concluded that the LightGBM model results in the best performance out of all models. The results of the model when applied to the test data highlights the importance of certain variables, but did not provide accurate predictions for exact salaries given a specific profile.

The analysis has reiterated the importance of job experience, location and job responsibilities in determination of compensation, but more research and analysis will have to be conducted to further refine developed models for more conclusive results.

2. Introduction

I am a member of a hypothetical human resource consulting firm specializing in tech-related hiring. To help the many small businesses globally who are struggling in their search for data science-related talent, the team aims to provide relevant and data-backed advice to help maximize their chances of finding the right hire. As mismatches in compensation expectations are often barriers to success (Criddle, 2023), we aim to solve this by providing recommended benchmarks for compensation, adjusted by seniority, location of work and job-specific requirements. This is done by the construction of a predictive model to determine the ideal compensation ranges for a given employee profile.

The predictive model developed will be able to predict a benchmark compensation value based on data relating to the job posting and requirements, which will help the team in guiding firms on compensation expectations for their desired candidate profile.

The dataset for this machine learning project is sourced from Kaggle, containing details of 5736 employed personnel. In addition to the currently drawn salary for each employee, the dataset further captures various employee attributes, such as experience levels, education level, exact job titles and work and residence locations.

The team constructed four models employing linear regression, regression trees, LightGBM, and XGBoost, respectively. We evaluated these models via R^2 and RMSE metrics on the test set to determine the model with the best performance. LightGBM demonstrated the highest R^2 and lowest RMSE value across all the trained and optimized models, and was hence

chosen as the candidate for implementation. To gain further insight into the underlying workings of LightGBM, we implemented SHAP (SHapley Additive exPlanations) values as part of the results analysis, allowing us to glean insights into categorical data with the most predictive power. By leveraging SHAP values, we can offer clients a more comprehensive understanding of how different factors influence the predicted salary, thereby enhancing the transparency and interpretability of our recommendations.

3. Data

The dataset is sourced from Kaggle, containing relatively up-to-date data with the last update coming in February 2024. The raw, as-is dataset contains 5736 rows of employee data from currently-employed personnel, with 11 distinct features. These features contain a combination of categorical and numerical data.

Categorical data includes experience levels, employment type, job title, salary currency, employee residence, company location and company size. Numerical data includes work year (year the salary was paid), salary in home currency, salary in USD and the job's remote work ratio.

The data provided was very clean in nature with no missing values and no observed data quality issues. Hence, the team was confident that the dataset did not need any major manipulation prior to pre-processing.

The first pre-processing step was to map the 'company location' feature into their respective continents. This step was undertaken after a trial run using the raw data file as-is. The trial run ran into issues after one-hot-encoding each company location as a unique feature, resulting in more than a hundred unique features containing binary data, with very few data points relevant for each company location. With only 5736 data points in total, the dataset was too thin to support the excessive number of features, and mapping each location to its respective continent was a solution to limit the number of features created. This mapping was conducted manually to account for the non-standardised nature of the 'company location' field, which often does not use the full or official names of the respective countries. Using this mapping, the dataset was classified into 5 separate continents, which became significantly more manageable during the one-hot-encoding process.

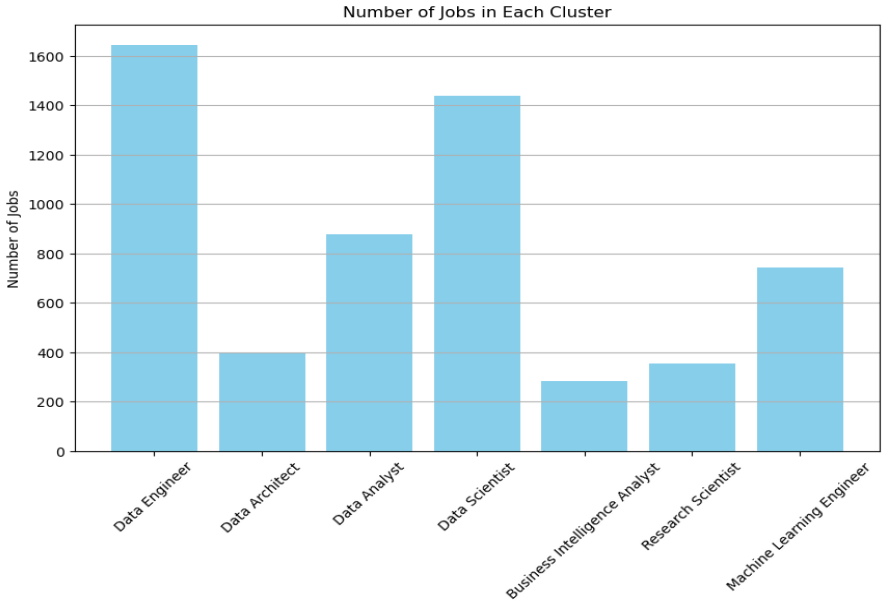


Figure 1: Bar graph of the number of data for each cluster of job titles

The second pre-processing step conducted was to replace the stated job title with cluster names using a KNN algorithm to split the dataset (Appendix 1). Using the KNN algorithm, job titles were grouped into 7 clusters: Data Architect, Business Intelligence, Data Engineer, Data Analyst, Data Scientist, Machine Learning Engineer and Research Scientist (Figure 1). Subsequently, the clusters were checked and individual data entries were reclassified as necessary. Job Title and Cluster number were also dropped as these features were deemed irrelevant after the usage of the KNN algorithm, which contained the necessary features for analysis.

The third pre-processing step called for the exclusion of features deemed irrelevant to the analysis. The team noted that salaries were provided in both the employee's home currency as well as its USD equivalent. To provide a basis for comparison without needing to consider exchange rate volatility, both the salary in the home currency and the stated currency were dropped from the dataset. For the purpose of all subsequent analysis, only the USD equivalent salary was utilized, and the salary prediction in USD was also set as the target variable for the model.

The team further decided to drop the year of the salary payment from the analysis. Initially, the team ran with the hypothesis that salaries could be scaled based on the year to account for inflation trends. However, this was determined not to be the case as our research found that the offered salary was unlikely to be dependent on the year of offer, but much rather as a function of supply and demand of the relevant skill sets sought. Hence, the year of salary payment was also dropped as we concluded that adjusting for inflation was not needed and could potentially introduce noise into the data.

After pre-processing of the dataset, all categorical data was one-hot-encoded as binary variables. These categorical features included Employment Type, Experience Level, Expertise Level, Company Continent, Company Size and Cluster Name. The dataset now contained 22 columns, where 21 of these columns were one-hot-encoded binary fields, while the Salary in USD was an integer data field and set as the target variable.

To ensure the removal of outliers, the team decided to clean the data using the interquartile range (IQR) which removed 88 rows from the dataset. The team's decision stemmed from a data exploration using boxplots of the USD equivalent salary. The boxplot contained numerous values outside of the IQR (Appendix 2), all of which were skewed upwards of the boxplot. These outliers could potentially impact the robustness of the model. In retrospect, this decision improved the R^2 and RMSE values of the team's linear regression model.

Finally, the dataset was split into a 70-30 training/testing split for the training of the model.

4. Methods

The team developed two 'base' models - Linear Regression and Regression Trees for the stated purpose of predicting compensation ranges for a given job opening. Based on these two 'base' models, the team further refined the two approaches via the implementation of advanced techniques and variants to choose the model with the best possible fit.

For Linear Regression, the team performed 5-fold Cross Validation, bootstrapping and Elastic Net regularization to maximally optimize the predictive power of the methodology. For Regression Trees, pruning was conducted to optimize the results, while two advanced gradient-boosting methods XGBoost and LightGBM were also implemented as ensemble methods to maximize the predictive power.

a. Linear Regression

The base Linear Regression used Scikit-Learn's Linear Regression feature, and was trained on the 70-30 split dataset. Subsequently, the model was evaluated on both the R^2 values and RMSE. The linear regression method assumes a linear relationship between the series of one-hot-encoded variables as detailed in part 3, and the dependent variable which is the salary in USD. This approach tries to fit a line based on the available independent variables, and plots a line-of-best-fit from training data to estimate the salary of any new data point with their unique set of independent variables.

In order to account for the risk of overfitting, we implemented both 5-fold Cross Validation and bootstrapping in the evaluation to check for variance in the results.

In a 5-fold Cross Validation, the dataset is shuffled and divided into 5 equal folds. One of these folds is reserved as the test set, while the remaining data is used for training. This process is repeated with each fold in turn being used as the test set. Essentially, 5 models are trained on the data, with each model providing its unique R^2 values and RMSE. This approach demonstrates the impact of potential outlier values on the performance of the model, and such outlier values can potentially be caught by looking into the performance of each iteration.

In bootstrapping, each value in the training set is sampled with replacement, essentially constructing new training datasets based on the already-chosen training dataset. This approach allows us to construct a series of training datasets where the impact of outliers can be flagged out, when bootstraps that amplify the presence of outliers are used in the training.

Comparing the R^2 values and RMSE of the original Linear Regression against the 5-fold Cross Validation and Bootstrapping averages, the base Linear Regression model had similar performance values compared to checking variants. Hence, we are relatively confident that the impact of outliers did not significantly impact the accuracy of the Linear Regression Model.

Feature selection by conducting an Elastic Net regularization was also conducted. By combining penalties of both L1 (Lasso) and L2 (Ridge) regularization methods to achieve a balance between variable selection and shrinkage of coefficients (Giba, n.d.), it diminishes the impact of coefficients with low predictive power to prevent overfitting and reduce variance. We tuned hyperparameters like alpha, L1-ratio and iterations by using GridSearchCV which systematically searches through a grid of hyperparameters to find the optimal combination that yields the best model performance (Appendix 3).

The Elastic Net approach removed 4 of the 22 features initially used to train the regression model, leaving a significant proportion of features in the model. From the calculated R^2 values and RMSE values, the team similarly noted no meaningful difference in performance scores

after regularization compared to both the base model and validation approaches, suggesting that the linear regression methodology had reached its upper limit for performance.

b. Decision Tree (Regression Tree with Pruning)

The base Regression Tree used Scikit-Learn's DecisionTreeRegressor feature and was similarly trained on the 70-30 split of the processed dataset. A Regression tree was chosen due to the nature of the target variable, which called for an integer value prediction based on the series of independent binary variables used.

A Regression Tree is highly readable and interpretable, as decision variables are clearly shown and the usage is similarly displayed. In essence, the model seeks to split the dataset into multiple subsets to decrease the dispersion of target variables in each 'leaf', or each endpoint. Using this approach, all data points are essentially classified based on their features, resulting in a prediction when new data is introduced with its unique collection of features.

The Regression Tree was initially trained without pruning, which resulted in potential overfitting due to the large number of branch nodes. The unpruned tree also had issues with readability and interpretability, as the number of nodes created resulted in difficulty tracing the decisions at each branch.

To reduce potential overfitting and increase the interpretability of the Regression Tree, the tree was pruned using Cost Complexity Pruning to reduce the number of nodes on the tree (Appendix 3). This resulted in a much simpler tree model that could be better generalized and interpreted. The CCP Alpha hyperparameter used to decide the optimal pruning amount was based on a reiterative test to find the parameter that minimized the RMSE recorded on the test set. The team notes that while there was some improvement after pruning, improvements were not significant compared to the base Regression Tree.

Comparing the pruned Regression Tree against the Linear Regression model with Elastic Net Regularization demonstrated that tree models performed slightly better for the goals of the project. Hence, the team implemented more advanced regression trees based on gradient-boosting to further push the performance limits of tree-based models.

c. XGBoost and LightGBM

XGBoost and LightGBM are both considered ensemble models that combine the predictive powers of multiple decision trees, and reiterate on each version of the model to optimize the model's weights. These models are generally considered very powerful tools that improve on the performance of a single decision tree. The team had previously noted the relative outperformance of tree-based models compared to linear regression, and these ensemble models serve to maximally optimize the predictive power of the method.

Via ensemble learning and gradient boosting, errors in each decision tree are progressively eliminated, resulting in a more accurate model. Similarly, these models also handle regularization and pruning automatically to achieve the best prediction accuracy without overfitting. After implementation, the team computed R^2 values and RMSE values for each method, which showed that for this use case, LightGBM produced the highest R^2 value and lowest RMSE value compared to all other models. Hence, this model was selected for usage and further discussion.

The team conducted parameter tuning on both XGBoost and LightGBM by tuning hyperparameters like number of leaves, learning rate, feature fraction, bagging fraction and

bagging frequency using the GridSearchCV as well as using early stopping to prevent overfitting (Appendix 3). The team then performed bootstrapping on the model with the best hyperparameters to evaluate its performance.

While powerful, these methods are often considered 'black boxes' that do not offer significant insight into the decision making process by the algorithm. Hence, for the team's LightGBM implementation, SHAP (SHapley Additive exPlanations) values were also computed as part of the results analysis to analyze the importance of each feature in the prediction of a salary value.

5. Results

Model	R-Square Value	RMSE
Linear Regression with 5-fold CV	0.319	51,558
Linear Regression with Bootstrap	0.316	51,678
Linear Regression with Elastic Net	0.319	51,545
Decision Tree (after pruning)	0.324	51,375
XGBoost with Hyperparameter tuned	0.354	50,218
LightGBM with Hyperparamter tuned	0.386	49,707

Figure 2: Table of the R^2 and RMSE values for each model

Based on the computed R^2 and RMSE values for each model implementation (Figure 2), LightGBM showed the best performance across both metrics. Hence, LightGBM was used as the basis for the team's analysis of the results predicted by the model.

Based on the analysis of the predicted salary against the actual salary of the data point (Figure 3), LightGBM correctly identified the general trend of salaries based on the various features encoded. However, the spread of values remained significant with RMSE of ~49,600. This is further visualized in the residual plot (Figure 4) which shows that errors could not be minimized even with gradient boosting algorithms. The increasing magnitude of residuals at higher predicted salary values indicates potential heteroscedasticity (Boyle, n.d.), where the variance in residuals is inconsistent across the range of predicted values.

This could have been due to the model having insufficient depth to account for complex nuances in the data, resulting in a model with generally limited predictive power.

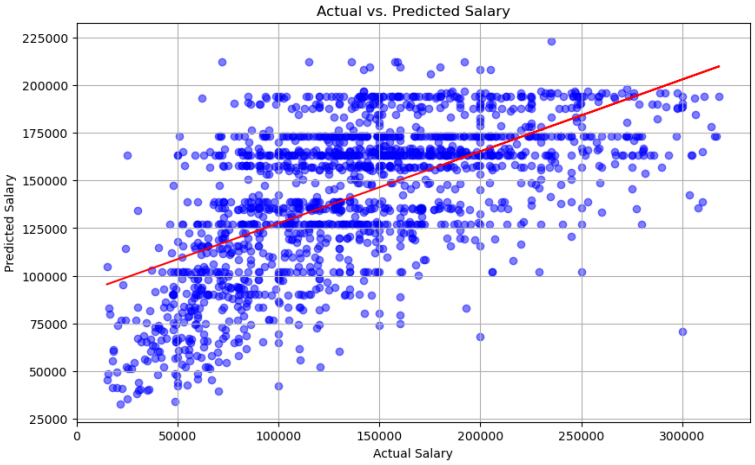


Figure 3: Graph of the actual salary vs predicted salary using the LightGBM model

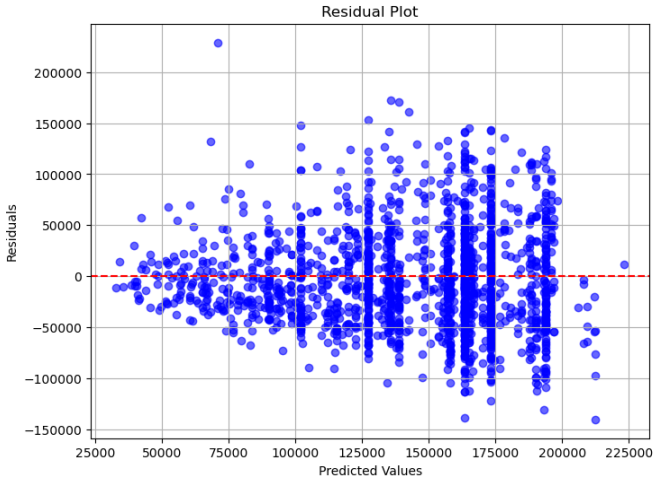


Figure 4: Residual plot of the actual salary vs predicted salary using LightGBM model

In a further SHAP analysis to isolate important predictors for salary ranges (Figure 5), Employee Residence, Job titles and Experience Level were highly impactful. This suggests that a role's compensation was highly dependent on geography, job scope and seniority level. This is in line with the team's understanding of the problem.

Notably, Company Size and Company Continent were not very influential, suggesting that companies, both small and large, are willing to pay similar amounts for a specific talent profile. In addition, the location of the company's continent was not a significant predictor, demonstrating that compensation is largely not tied to a company's base of operations, but much rather the local characteristics of the employee's job market.

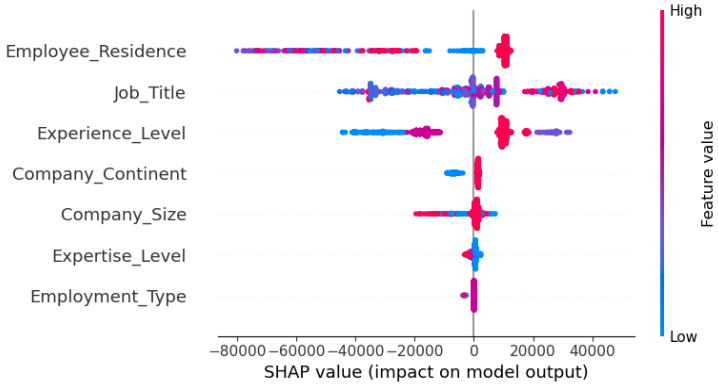


Figure 5: SHAP values beeswarm graph

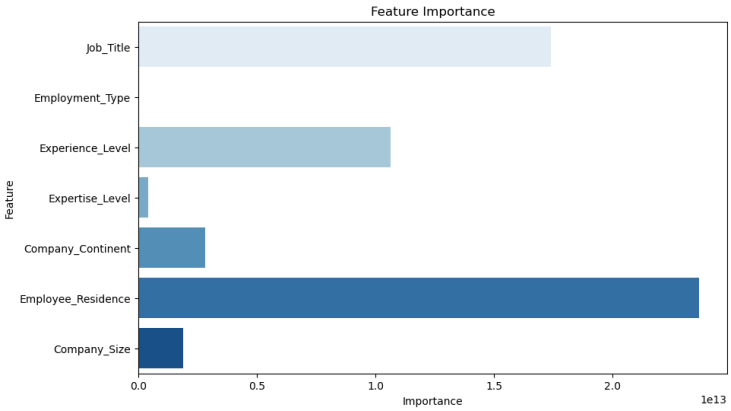


Figure 6: Mean absolute SHAP values for LightGBM model

6. Discussion

The model demonstrates the outsized impact of location, experience levels and job titles in determining salary expectations. While the team does not know of other models in academia or industry that capture un-modelled nuances to arrive at a better prediction score, numerous job sites provide benchmark calculators and salary references tied to specific jobs and specific companies. These benchmark calculators generally use similar methodologies to arrive at a

benchmark salary range, such as the input of Job Title, Years of Experience, Location and Industry Types. The approach of the industry-standard calculations seems to resemble a filtered data table rather than a predictive model, with individual candidates left to determine their own expectations within the given range of results.

Upon further inspection of the dataset, the analysis can be refined with a more robust dataset. For example, experience levels are listed as categorical data, such as 'Senior' and 'Entry' levels. This leaves room for interpretation of what is considered 'Senior' or 'Entry', and may vary by company. Instead, if experience level data was given as integer data representing the number of years on the job, this would bring the methodology in line with industry calculators, and better results may be potentially drawn. Additional data points could also be helpful, especially given the large number of features and categories within each feature - more nuances could potentially be identified given a larger dataset that was not accounted for in our implementation.

Finally, further investigations into other machine learning models might provide more accurate results such as Support Vector Machines. Going back to the years of experience as a potential data point, the relationship between years of experience and salary may resemble an exponential relationship rather than linear, which could be better accounted for by SVM techniques.

7. Conclusion

The analysis has further highlighted and reiterated the importance of certain metrics when determining a role's salary expectations, but did not manage to give a highly accurate prediction of actual salaries offered and accepted by the company and employee.

For the purpose of the investigation, the model has delivered a single integer value for salary expectations when given data on the role's characteristics. This should only serve as a guideline for expectations, as the actual range could vary widely, especially at higher salaries. Companies should exercise discretion and use the predicted value to determine an appropriate range of compensation for a role.

In the wider context of the problem, factors such as location, experience levels and job titles are only indications providing a baseline reference, but candidates have their individual motivations and priorities. In addition to a competitive salary, companies should consider other aspects, such as non-monetary perks or bonus structures to better retain and attract talent above and beyond base salary expectations.

References

1. Boyle, M. (n.d.). *Heteroscedasticity Definition: Simple Meaning and Types Explained*. Investopedia. Retrieved April 19, 2024, from https://www.investopedia.com/terms/h/heteroskedasticity.asp
2. Criddle, C. (2023, April 16). *Tech skills shortfall frustrates start-ups looking to hire*. Financial Times. Retrieved April 19, 2024, from https://www.ft.com/content/20d7183b-b7d9-466d-b95d-aea6258fa143
3. Giba, L. (n.d.). *Elastic Net Regression Explained, Step by Step*. Machine Learning Compass. Retrieved April 19, 2024, from https://machinelearningcompass.com/machine_learning_models/elastic_net_regression/
4. Nadarajan, R., & Ong, J. (2024, March 26). *Lack of required skills, mismatched salary expectations make tech, nursing vacancies hard to fill, say recruiters*. Today

Online. Retrieved April 19, 2024, from https://www.todayonline.com/singapore/lack-required-skills-mismatched-salary-expectations-make-tech-nursing-vacancies-hard-fill-say-recruiters-2391561

Appendix

Appendix 1

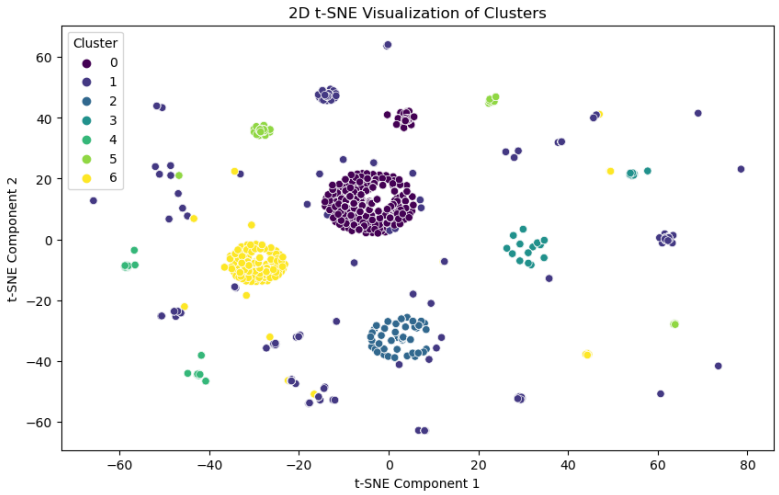


Figure 9: 2D t-SNE Visualization of Clusters

The t-Distributed Stochastic Neighbor Embedding (t-SNE) visualisation is a dimensionality reduction technique used to visualize high-dimensional data in a lower-dimensional space, preserving local structure.

Each point on the plot represents employee data which is coloured according to the cluster that their job title is in. Points which are close together will present a group of job titles that share similar Term Frequency - Inverse Document Frequency (TF-IDF) vectors. In general, points close together will belong to the same cluster. However, since manual processing of some job titles was conducted, there are some outliers across the plot.

Appendix 2

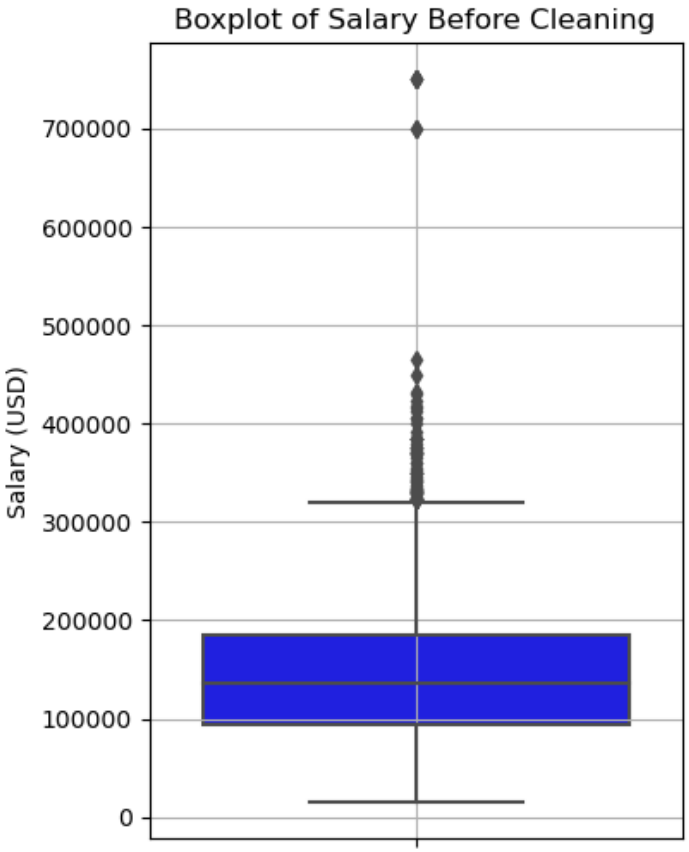


Figure 7: Boxplot of Salary in USD before cleaning

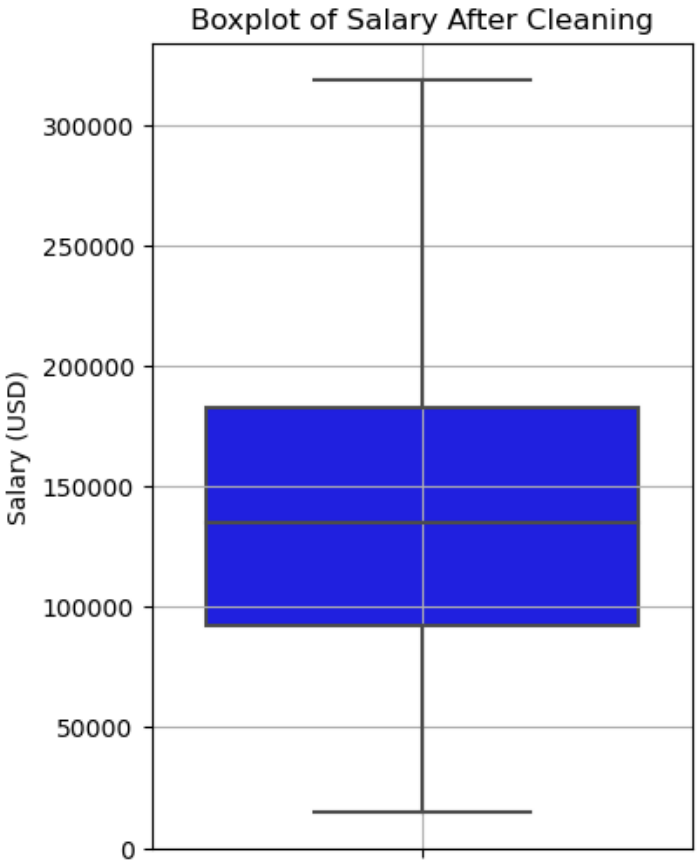


Figure 8: Boxplot of Salary in USD after removing outliers using IQR method

Appendix 3 (Codes)

Code to show Hypertuning of Lightgbm

In [34]:
```
from sklearn.model_selection import train_test_split
import lightgbm as lgb
from sklearn.metrics import mean_squared_error, r2_score

# Split your data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_cleaned, y_train_cleaned, test_size=0.2, random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'num_leaves': [15, 31, 50],
    'learning_rate': [0.01, 0.05, 0.1],
    'feature_fraction': [0.6, 0.8, 0.9],
    'bagging_fraction': [0.6, 0.8, 0.9],
    'bagging_freq': [3, 5, 7]
}
# Create the grid search object
grid_search = GridSearchCV(estimator=lgb.LGBMRegressor(objective='regression', metric='mse', verbose=-1),
                           param_grid=param_grid,
                           cv=5,
                           scoring='neg_mean_squared_error',
                           verbose=2,
                           n_jobs=-1)

# Perform grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Train the model with the best hyperparameters and early stopping
best_model = lgb.LGBMRegressor(objective='regression',
                               metric='mse',
                               num_boost_round=1000,
                               verbose=-1,
                               **best_params)

# Create the LightGBM dataset
train_data = lgb.Dataset(X_train, label=y_train)
val_data = lgb.Dataset(X_val, label=y_val)

# Add early stopping by passing a callback to the train() method
best_model = lgb.train(params=best_params,
                       train_set=train_data,
                       num_boost_round=1000,
                       valid_sets=[train_data, val_data],
                       valid_names=['train', 'val'],
                      callbacks=[
        lgb.early_stopping(stopping_rounds=3),
    ])

# Evaluate the best model
y_pred_best = best_model.predict(X_test_cleaned, num_iteration=best_model.best_iteration)
mse_best = mean_squared_error(y_test_cleaned, y_pred_best)
rmse_best = np.sqrt(mse_best)
r2_best = r2_score(y_test_cleaned, y_pred_best)
```

```
print("\nMetrics for Best Model:")
print("Mean Squared Error:", mse_best)
print("Root Mean Squared Error:", rmse_best)
print("R-squared:", r2_best)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best Hyperparameters: {'bagging_fraction': 0.9, 'bagging_freq': 3, 'feature_fraction': 0.6, 'learning_rate': 0.1, 'num_leaves':
15}
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000210 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 146
[LightGBM] [Info] Number of data points in the train set: 3162, number of used features: 6
[LightGBM] [Info] Start training from score 140740.076534
Training until validation scores don't improve for 2 rounds
Early stopping, best iteration is:
[60]    train's l2: 2.23936e+09 val's l2: 2.38362e+09

Metrics for Best Model:
Mean Squared Error: 2482403086.881797
Root Mean Squared Error: 49823.72012286715
R-squared: 0.3637428547165034
```

In [36]:
```
import numpy as np

# Number of bootstrap iterations
n_iterations = 100

# Initialize an empty list to store RMSE values
rmse_scores = []

# Perform bootstrap resampling
for _ in range(n_iterations):
    # Generate bootstrap sample indices
    indices = np.random.choice(len(X_test_cleaned), len(X_test_cleaned), replace=True)

    # Extract bootstrap sample
    X_bootstrap = X_test_cleaned.iloc[indices]  # Use iloc to index DataFrame by integer position
    y_bootstrap = y_test_cleaned.iloc[indices]

    # Make predictions on the bootstrap sample
    y_pred_bootstrap = best_model.predict(X_bootstrap, num_iteration=best_model.best_iteration)

    # Calculate RMSE for the bootstrap sample
    rmse_bootstrap = np.sqrt(mean_squared_error(y_bootstrap, y_pred_bootstrap))

    # Append RMSE to the list
    rmse_scores.append(rmse_bootstrap)

# Calculate the mean and standard deviation of RMSE scores
mean_rmse = np.mean(rmse_scores)
std_rmse = np.std(rmse_scores)

print("Mean RMSE (Bootstrap):", mean_rmse)
print("Standard Deviation RMSE (Bootstrap):", std_rmse)
```

```
Mean RMSE (Bootstrap): 49707.04388744717
Standard Deviation RMSE (Bootstrap): 876.0312661574835
```

Code for the hypertuning of Elastic Net Grid

In [51]:
```
# Initialize ElasticNet model
elastic_net = ElasticNet(random_state=42)

# Define hyperparameters to tune
params = {
    'alpha': [0.1, 0.5, 1, 2, 5, 10], #this is lambda is slides formula 6ModelSelection
    # Currently, l1_ratio <= 0.01 is not reliable
    'l1_ratio': np.arange(0.05, 1.05, 0.05), #this is the alpha in the slides notes
    'max_iter': [10000] #e.g. in the case of gradient descent this no. is how many times the ship moves left and right
}

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=elastic_net, param_grid=params, cv=5, scoring='neg_mean_squared_error',
                           verbose=3, #can be 2 or 3 or 4 (look up website each verbose returns diff things)
                          #for 3 for each training it tells you training and test error
                          return_train_score=True)

# Fit the model on the training data
grid_search.fit(X_train_scaled, y_train)
```

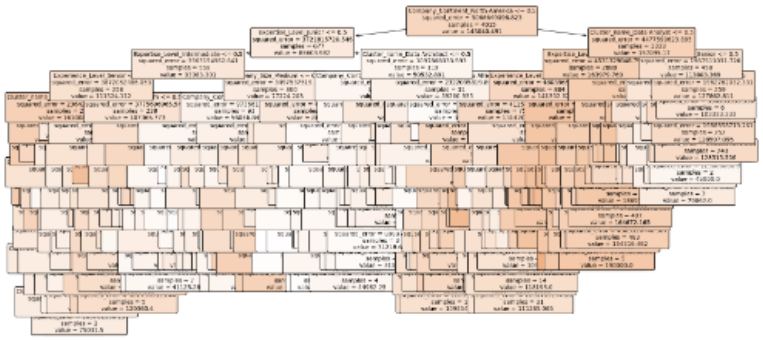
```
[CV 3/5] END alpha=0.1, l1_ratio=0.1, max_iter=10000;, score=(train=-3558009335.673, test=-3707773287.221) total time=   0.0
s
[CV 4/5] END alpha=0.1, l1_ratio=0.1, max_iter=10000;, score=(train=-3498322361.418, test=-3938903187.565) total time=   0.0
s
[CV 5/5] END alpha=0.1, l1_ratio=0.1, max_iter=10000;, score=(train=-3554007301.762, test=-3702505649.993) total time=   0.0
s
[CV 1/5] END alpha=0.1, l1_ratio=0.15000000000000002, max_iter=10000;, score=(train=-3605379368.855, test=-3530573070.621) t
otal time=   0.0s
[CV 2/5] END alpha=0.1, l1_ratio=0.15000000000000002, max_iter=10000;, score=(train=-3686769690.399, test=-3185881444.249) t
otal time=   0.0s
[CV 3/5] END alpha=0.1, l1_ratio=0.15000000000000002, max_iter=10000;, score=(train=-3557380376.159, test=-3707066141.498) t
otal time=   0.0s
[CV 4/5] END alpha=0.1, l1_ratio=0.15000000000000002, max_iter=10000;, score=(train=-3497709102.260, test=-3938338324.251) t
otal time=   0.0s
[CV 5/5] END alpha=0.1, l1_ratio=0.15000000000000002, max_iter=10000;, score=(train=-3553429883.320, test=-3702515219.013) t
otal time=   0.0s
[CV 1/5] END alpha=0.1, l1_ratio=0.2, max_iter=10000;, score=(train=-3604778793.808, test=-3530372742.368) total time=   0.0
s
[CV 2/5] END alpha=0.1, l1_ratio=0.2, max_iter=10000;, score=(train=-3686159503.164, test=-3185218474.648) total time=   0.0
```

Code for pruning of decision tree

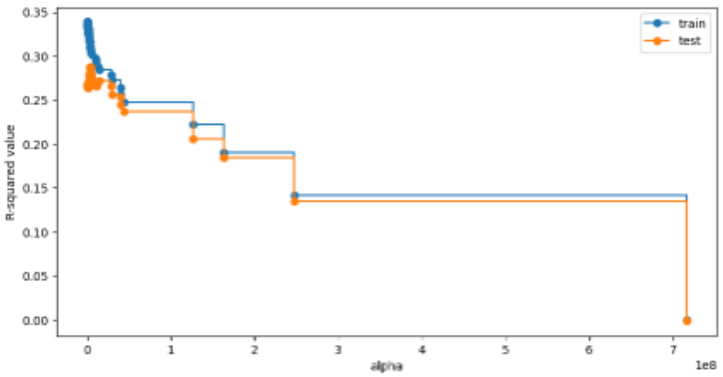
In [35]:
```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, mean_squared_error
import numpy as np
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

In [36]:
```
# Create a Decision Tree Classifier
clf = DecisionTreeRegressor(random_state=42)

# Fit the classifier on the entire training data
clf.fit(X_train, y_train)

# Evaluate the classification tree on the test data
y_pred = clf.predict(X_test)

# Calculate evaluation metric (e.g., Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
r_squared = clf.score(X_test, y_test)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r_squared)

path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities



plt.figure(figsize=(20,10))  # Set the figure size
plot_tree(clf, feature_names=X_train.columns.tolist(), filled=True, rounded=True, fontsize=10)
plt.show()
```

```
Mean Squared Error: 3523531530.3464875
Root Mean Squared Error: 59359.34248156715
R-squared: 0.2669716472649719
```



In [37]:
```
# Create a Decision Tree Regressor
clf = DecisionTreeRegressor(random_state=42)

# Fit the classifier on the entire training data
clf.fit(X_train, y_train)

# Cost-complexity pruning
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Create an array of decision trees
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeRegressor(random_state=42, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# Calculate the validation scores
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

# Plotting validation curves
plt.figure(figsize=(10, 5))
plt.plot(ccp_alphas, train_scores, marker='o', label="train", drawstyle="steps-post")
plt.plot(ccp_alphas, test_scores, marker='o', label="test", drawstyle="steps-post")
plt.xlabel("alpha")
plt.ylabel("R-squared value")
plt.legend()
plt.show()
```



In [38]:
```
# Find the best alpha value that maximizes R-squared on the test set
best_alpha = ccp_alphas[np.argmax(test_scores)]

# Create a decision tree regressor with the best alpha value
pruned_tree = DecisionTreeRegressor(random_state=42, ccp_alpha=best_alpha)

# Fit the pruned tree on the entire training data
pruned_tree.fit(X_train, y_train)

# Predict on the test set
y_pred = pruned_tree.predict(X_test)

# Calculate R-squared value
r_squared = pruned_tree.score(X_test, y_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Print R-squared and RMSE
print("R-squared:", r_squared)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Best alpha:", best_alpha)

# Visualize the pruned decision tree
plt.figure(figsize=(20, 10))
plot_tree(pruned_tree, feature_names=X_train.columns.tolist(), filled=True, rounded=True, fontsize=10)
plt.show()
```

```
R-squared: 0.2875692128401023
Mean Squared Error (MSE): 3424522847.4196134
Root Mean Squared Error (RMSE): 58519.42282199657
Best alpha: 4169753.6486175494
```