Final Project – Machine Learning

Mika Friedman

Yali Keinan

Introduction

In this project, we addressed a binary classification problem using a dataset containing various features about candidates, with the goal of predicting whether each candidate would be hired. We received two csv files containing a division to train set and test set. We trained a few models on the train data and at the end our chosen best model will be tested on new unseen data - the test set. The project encompassed data exploration, preprocessing, model training, evaluation, and prediction.

Part 1: Exploration

Initially, we explored the train dataset to understand the information it contained. We noted that the dataset **comprised 55,462 rows and 17 columns**, each representing a candidate and their specific features. From the provided data, it appears to be a dataset with **various features and a corresponding label column**.

Missing Values: **Most of the features have missing values.** For data preprocessing and analysis, understanding and dealing with the missing data is crucial. Examples of features with missing values are – "stack_experience", "disability", "B" and others.

We identified two types of features in the data, numerical and categorical. Numerical - "years_of_experience", "prev_salary", "A", "B", "D", "label". Categorical - "age_group", "disability", "worked_in_the_past", "is_dev", "education", "sex", "mental_issues", "C", "country", "stack_experience". We checked the unique values of those features to better understand what information are we dealing with. During preprocessing of the training data, we encoded categorical features into numeric values, transform the country feature into continents. and generated new columns for these encoded values.

We analyzed the employment history of the hired candidates and discovered that approximately **85% had previous work experience**. Additionally, **the average years of experience among the hired candidates is 14 years** (with a standard deviation of 9 years) and their **average previous salary is 67,821** (with a standard deviation of 28,291). About **60% of them are under the age of 35**, roughly **90% are men**, and **most of them are from the USA**.

To explore our data, we computed basic statistics, such as mean, maximum, minimum, and standard deviation. These statistics can help us identify patterns and potential outliers within our data and provide insight into their distribution. Then we visualized the data to observe the distribution of each feature. The **features "A" and "D" displayed a distribution closely resembling a normal distribution** (as shown in appendix 1). **"Years of experience", "B", and "Prev_salary" are left-skewed, with most candidates having lower values.**

We checked for the **presence of duplicates**. Duplicates can introduce bias and lead to misinterpretations. Removing duplicates enhances data quality and improves model accuracy. Our examination confirmed that there are **no duplicates** in the dataset.

We visualized a **correlation matrix** (as shown in appendix 2) to help understand the relationships between different numeric features. We discovered that there are positive correlations between:

B and years_of_experience: A high positive correlation of 0.900905. This suggests that candidates with higher values in the B feature also tend to have more years of experience. **B and "prev_salary": A positive correlation of 0. 391391**. This suggests that candidates with higher values in the B feature also tend to have higher previous salaries.

years_of_experience and prev_salary: A positive correlation of 0.3899, indicates that candidates with more years of experience tend to have higher previous salaries.

"Label" and "D": A positive correlation of 0.406377. This suggests that candidates with higher values in the D feature also tend to get hired.

We also noticed a negative correlation for example between "years_of_experience" and "label". It suggests that as the number of years of experience increases, the likelihood of being hired decreases. This could indicate that the model identifies other factors as more important for the label, or that candidates with more experience may be perceived as overqualified.

Analyzing the correlation matrix, allowed us to make informed decisions about which features to focus on and which ones might be less important. This insight informed our decision to **drop feature "A"**, due to low correlation.

We made some visualizations regarding the categorical features, which helped us understand whether certain values are more commonly associated with candidates who were hired compared to those who were not hired:

We checked the **distribution of education Levels among Hired and Non hired Candidates** (as shown in appendix 3). We can infer from the plots that the **most common education level among hired candidates is a BA/BSc**. However, the plots show that most job applicants also have a BA/BSc. Thus, while a BA/BSc is prevalent among hired candidates, it is also the most common degree among applicants.

We analyzed the **distribution of the top 20 tech stack experiences by hiring status** (as shown in appendix 4) and found that the most common technology among hired candidates is JavaScript. Docker, HTML/CSS, and SQL also have high counts among both hired and non-hired candidates. Additionally, candidates proficient in Node.js and TypeScript are more likely to be hired compared to those who are not. This suggests that these technologies are highly valued and preferred by employers during the hiring process.

We plotted the **distribution of hired and not hired candidates by continent** (as shown in appendix 5) using the library "pycountry-convert". By incorporating pycountry-convert into our data processing, we aimed to see if geographical location has any influence on the hiring outcomes. The plot showed that Europe and North America are the primary sources of talent, followed by Asia. Given the similar proportions of hired and not hired candidates across continents, we understood that the continent of origin may not be a significant factor in the hiring decision.

Part 2: Preprocessing

We decided to drop some more features from the training data: We chose to remove the ID feature because each ID has its own unique value, and it does not provide any useful

information for the prediction process. In addition, we removed columns that we encoded from categorical to numeric.

Handling Missing Values: We adopted different strategies for handling missing values- Median Imputation - For numerical columns, such as "years_of_experience", "prev_salary", "D", and "B", we used **median imputation**. Using the median ensures the imputed values stay within the existing range of data, maintaining the original distribution. For binary categorical columns, such as "worked_in_the_past_encoded", "disability_encoded", "is_dev_encoded" and "mental_issues_encoded", we used the **most frequent value** for imputation. For columns like "education_encoded" and "sex_encoded", we **replaced missing values with specific values** to avoid making strong assumptions. For instance, we filled missing values in "education_encoded" with the category "other" and in "sex_encoded" with "other" to avoid introducing bias. Finally, we ensured that there are no missing values remaining.

After categorizing data and addressing missing values, we **analyzed outliers using box plots** (as shown in appendix 6). Instead of removing outliers, we applied robust normalization techniques to mitigate their impact while preserving valuable information. We applied the following normalization techniques: **Min-Max Scaling and Robust Scaling**. These methods are used to transform features to a common scale, which helps improve the performance and stability of machine learning models. For features such as "years_of_experience", "B" and "prev_salary", we used Robust Scaling to handle their skewed distributions and outliers. For feature “D” Min-Max Scaling was appropriate to transform into a fixed range, making them easier to compare. Later on, we observed a **significant decrease in the AUC scores without normalization**, **so we decided to normalize the data**.

We **split the train dataset into training and validation** to ensure reliable model evaluation. This approach allows us to tune the model on one set and evaluate its performance on unseen data, preventing overfitting and ensuring generalizability.
Next, we aimed to enhance the model's accuracy by performing **dimensionality reduction**. We used **Principal Component Analysis** (PCA). High-dimensional data can cause overfitting, but PCA helps simplify the data while preserving key information. Our data’s feature-to-sample ratio indicates that dimensionality is relatively low.
We calculated the **cumulative explained variance plot** (as shown in appendix 7) which showed that **retaining 59 components captures 95% of the variance**, mitigating overfitting and enhancing interpretability. Despite this, **the models performed better without PCA**, as indicated by higher AUC scores. Thus, **we proceeded on the train data without PCA for optimal model performance.**

Part 3: Models

To **determine the optimal hyperparameters** for each model and minimize overfitting, we **implemented grid search and random search along with cross-validation using K-fold**. This comprehensive approach allowed us to explore a wide range of hyperparameters and select the best ones for each model (explanation about the chosen best hyperparameters are shown at appendix 8). After determining the optimal hyperparameters, we commented this part of the project to reduce calculation times, saving the chosen hyperparameters to run the selected models without repeating the search.

Initially, we tested all the models discussed during our course, to identify those with the highest performance. After **evaluating both running times and model fit**, we decided to keep Logistic Regression and Gaussian Naive Bayes (2 out of the 3 simple models), along with Random Forest and AdaBoost (2 out of the 4 advanced models).

Part 4: Model Evaluation

After initially running the models, we sought to improve their performance by refining the feature set. We experimented with the stack_experience feature, beginning by **quantifying the number of technologies** each candidate knows, which yielded an AUC of 0.87. We then categorized the technologies into programming, hardware, and DevOps, **assigning scores** based on the specific technologies each candidate knows. Ultimately, we found that using the **one-hot vector technique significantly enhanced the model's performance**, so we finally chose this type of feature handling. We found this approach reasonable, as most of the features were background data, which didn't significantly affect the model's performance. However, there was a significant variation in the original stack_experience column, with some candidates being proficient in many technologies, others in only a few, and some with no relevant skills at all. **After this change in feature handling, the AUC improved significantly.** It makes sense that candidates proficient in certain technologies are more likely to be hired. To increase the impact of this feature on the model, we decided to split it, emphasizing each technology individually. This shift allowed us to **focus on specific technologies** that might be more in demand or relevant to the roles considered in this project.

We plotted a **graph of ROC Curves with Stratified K-Fold Cross-Validation** (as shown in appendix 9) that allowed us to **visually compare the performance of different models**. By looking at the ROC curves, we can identify which model performs better. Using stratified K-fold cross-validation ensures that each fold maintains a similar class distribution, offering a more reliable evaluation of the model's performance. The average ROC curve and AUC scores across all folds demonstrate the model's stability and robustness, as they are consistently similar. **The mean AUC results that we demonstrated are: Logistic Regression: 0.92, Gaussian NB: 0.76, AdaBoost: 0.98, Random Forest: 0.97.**

To **compare model performance on the training set and unseen validation set**, we calculated the AUC for each. Here is a summary - **Logistic Regression**: AUC scores are very close for both sets, indicating good generalization. **Gaussian Naïve bayes**: Similar AUC scores for training and validation, showing a good fit. **AdaBoost**: AUC scores are very close for both sets, suggesting strong generalization. **Random Forest**: The validation AUC is slightly lower than the training AUC, indicating minor overfitting but still good generalization.

We used grid search, cross-validation, and probability-based evaluation to find the best model. With the highest AUC score, the **AdaBoost classifier is identified as the best performer**, showing high validation AUC and effective generalization.

After we chose AdaBoost as our chosen model, we wanted to **evaluate the contribution of each feature for the model's performance** (as shown in appendix 10), by creating a plot of the top 50 feature importances. From the plot we can learn that the **most valuable features are "D", "C#", "Microsoft SQL Server", "Java", and "JavaScript"**. As we assumed (and explained in the beginning of part 4), the most valuable features are the types of technologies the

candidate knows. **This indicates that these features are indeed helpful in determining whether a candidate will be hired.**

We analyzed the **confusion matrix** (as shown in appendix 11) for the **validation set** to get insights into the models' generalization capabilities and performance across different classes. Interpretation of the confusion matrix:

The confusion matrix presented 4673 TP, 5409 TN, 460 FP and 551 FN. These values helped in calculating the following metrics: **Accuracy**: The model correctly predicts 90.88% of the instances, indicating that the overall correctness of the model is quite high. **Precision**: 92.16% of the positive predictions made by the model are accurate, meaning the model is good at avoiding false positives. **Recall**: 90.75% of actual positive cases are correctly identified by the model, reflecting a strong ability to detect positive instances. **Specificity**: This is the ratio of correctly predicted negative instances to all the instances in the actual negative class. About 91.03% of the actual negatives were correctly identified by the model. **The model shows strong overall performance**, with high accuracy and well-balanced precision and recall. It effectively predicts both positive and negative cases, minimizing errors and demonstrating reliability across various metrics.

Part 5: Prediction

To get prediction on the data, we want to train the model on the **full training data**, including both train and validation. It ensures the model learns from all available data, leading to improved performance and robustness.

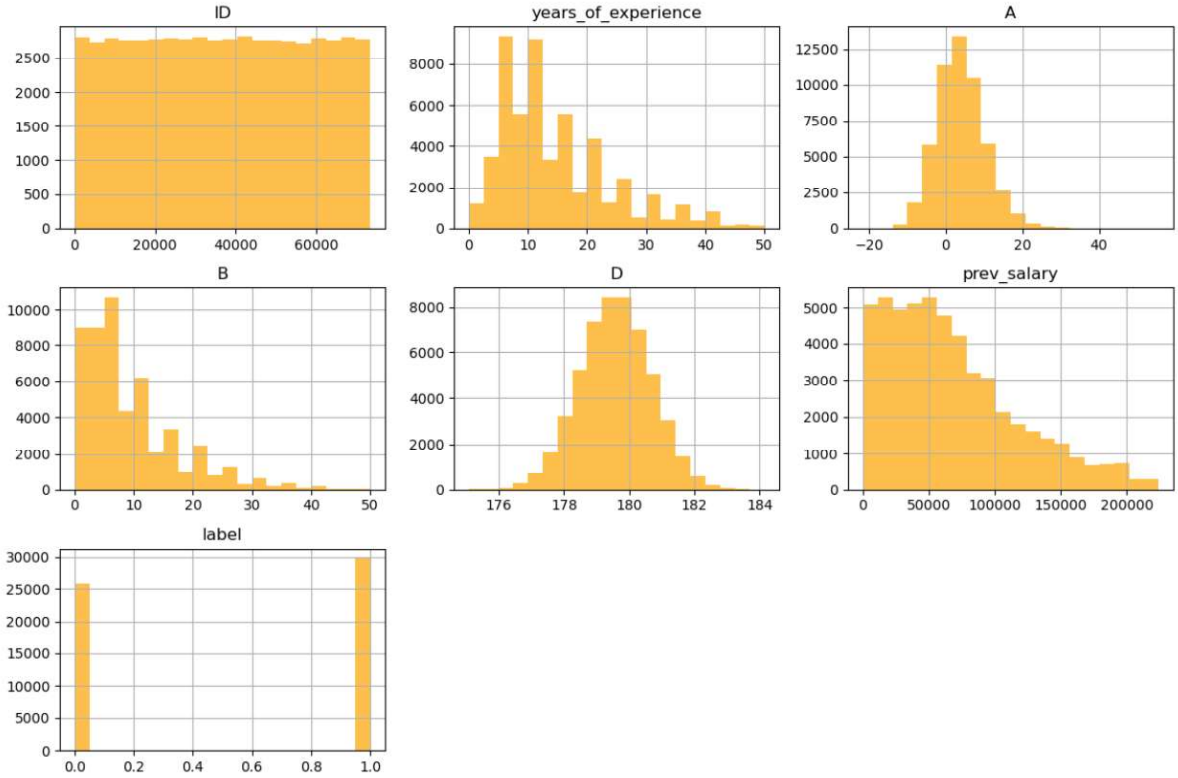
In the final stage of the project, the **Pipeline**, we **executed all the data processing steps** - such as handling features, filling missing values, and dropping columns - on both the training and test datasets. **This ensures that the predictions accurately reflect the assumptions made during the project.** We saved the results in the results CSV file, which now contains the probability of each candidate from the test set being hired.

Part 6: Using Tools Not Covered in the Course

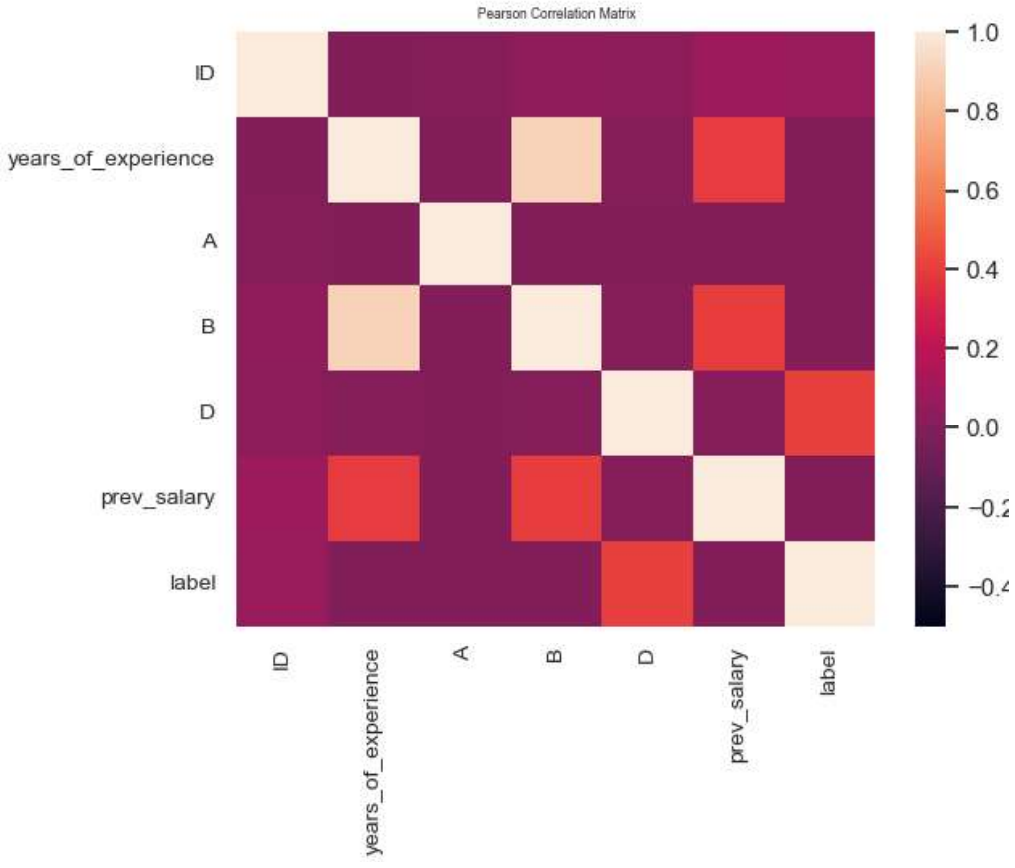
We chose to use the library - **pycountry-convert**. pycountry-convert is a Python library that facilitates the conversion between different geographical naming conventions. It provides functionalities to convert country names to different codes (such as ISO alpha-2 and alpha-3), and also to convert country codes to continent codes and continent names. By incorporating pycountry-convert into our data processing pipeline, we aim to see if geographical location has any influence on the hiring outcomes. refer to the official documentation: https://pypi.org/project/pycountry-convert/

Additionally, we applied a normalization method that have not been covered in the course, **Robust Scaling**. We used this method to handle features with many outliers while preserving the significance of outlier values, as detailed in part 1.

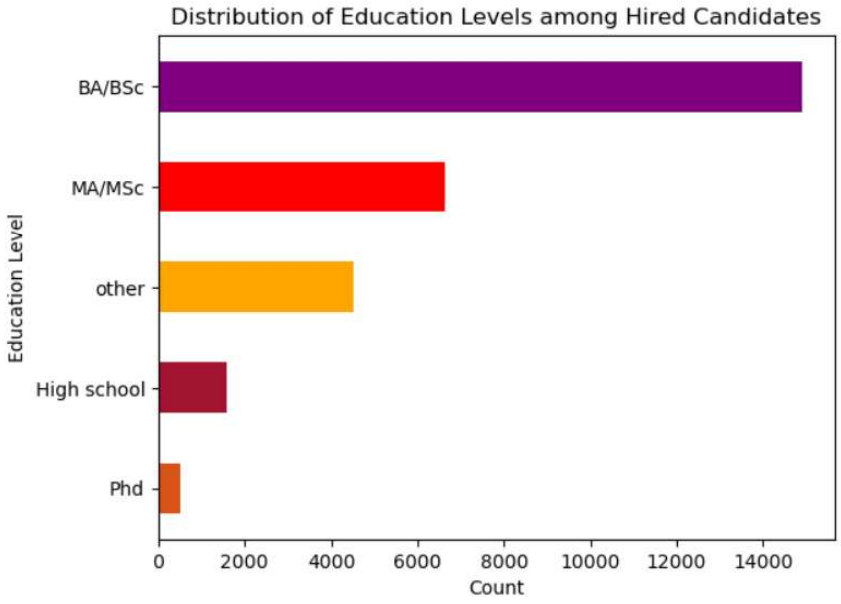
Appendix 1

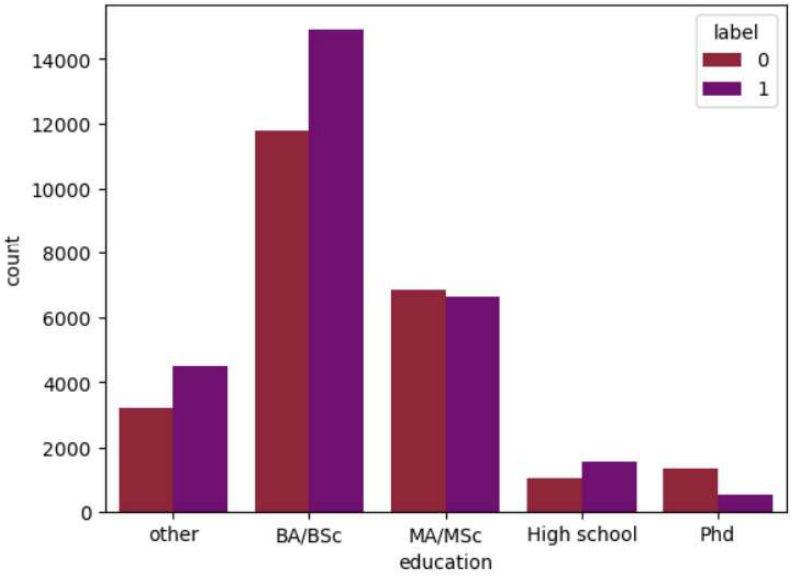


Appendix 2

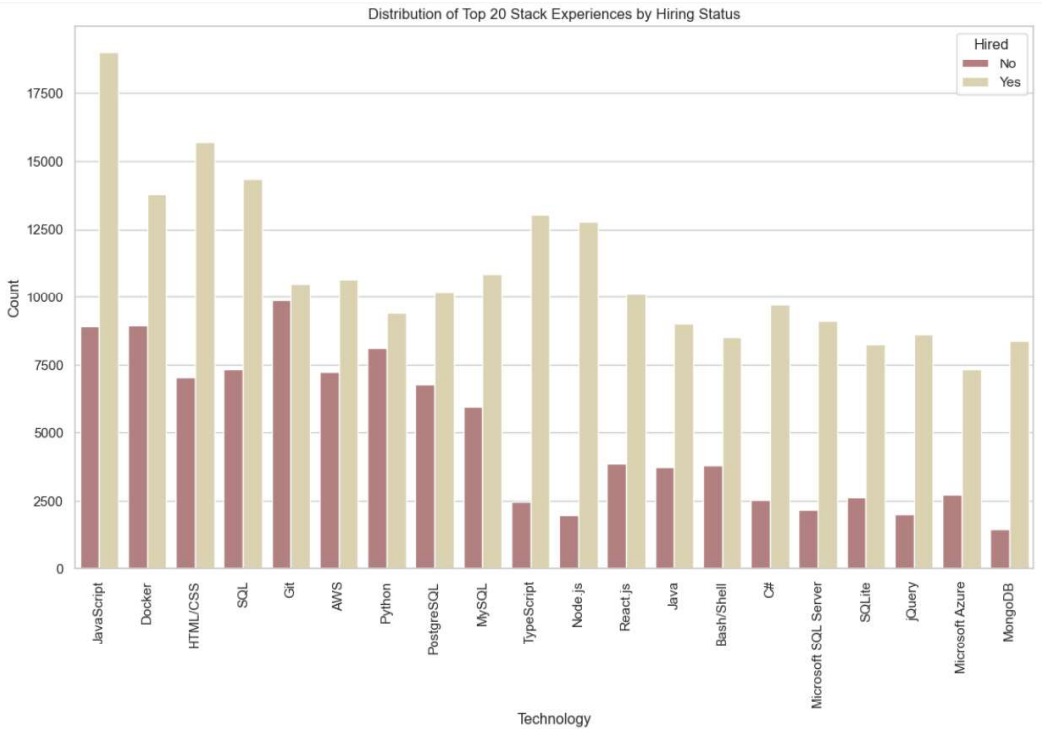


Appendix 3

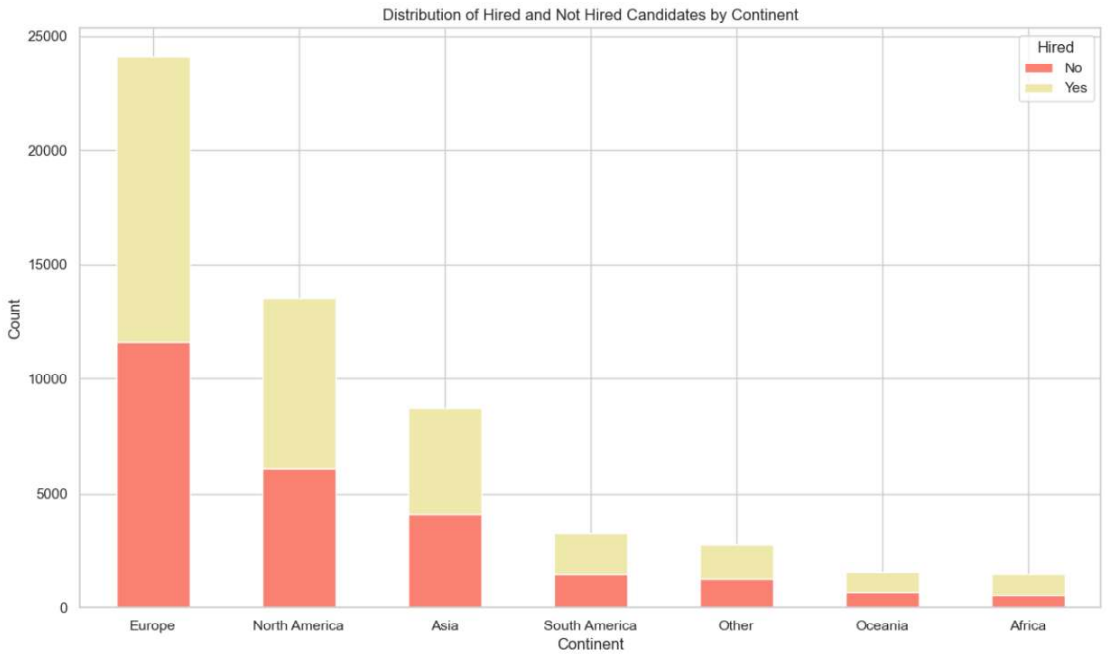

Distribution of Education Levels among Hired Candidates
BA/BSc
MA/MSc
other
High school
Phd
Education Level
0
2000
4000
6000
8000
10000
12000
14000
Count


label
0
1
14000
12000
10000
8000
6000
4000
2000
0
count
other
BA/BSc
MA/MSc
High school
Phd
education

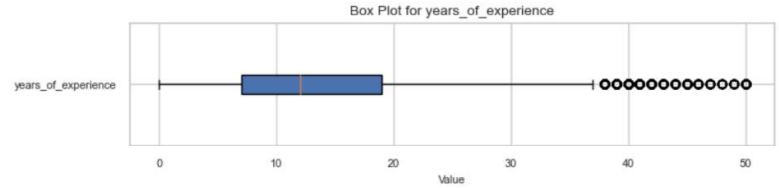
Appendix 4

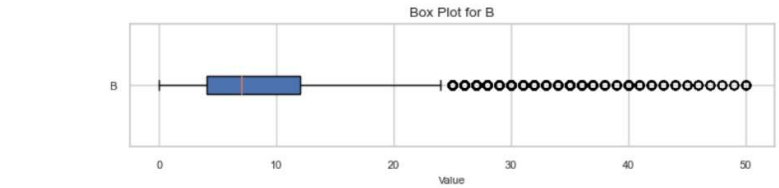


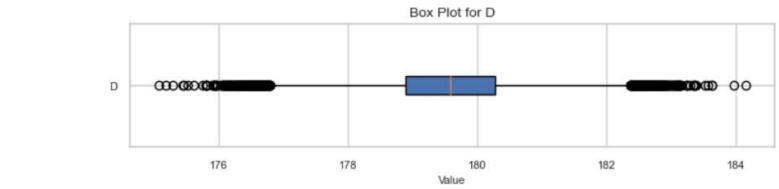
Appendix 5

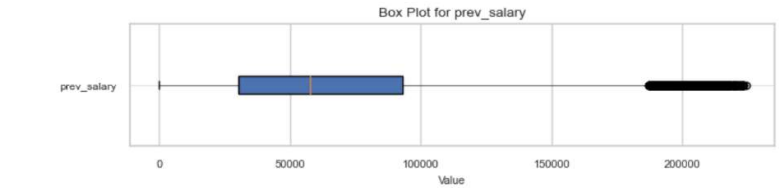


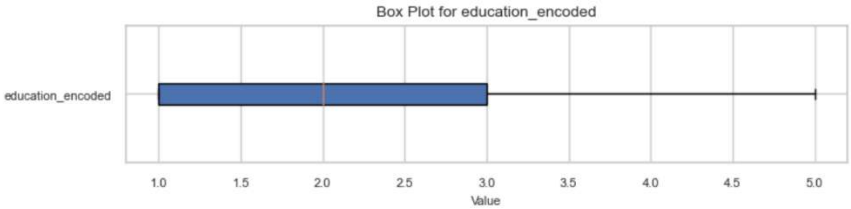
Appendix 6

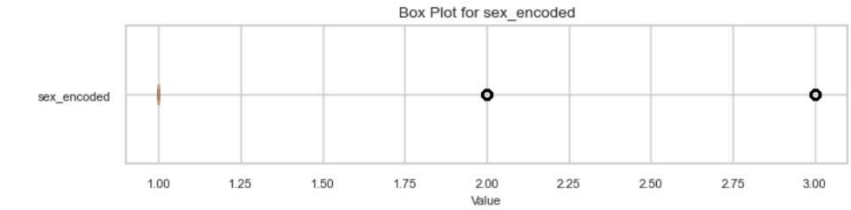

Box Plot for years_of_experience
years_of_experience
0
10
20
30
40
50
Value

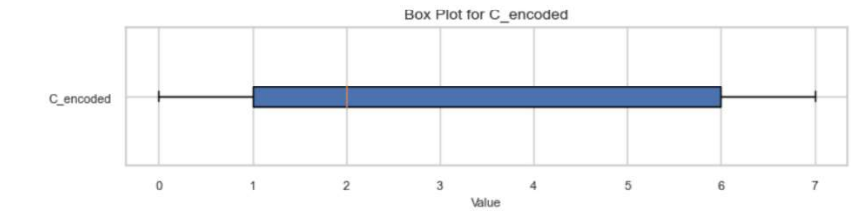

Box Plot for B
B
0
10
20
30
40
50
Value

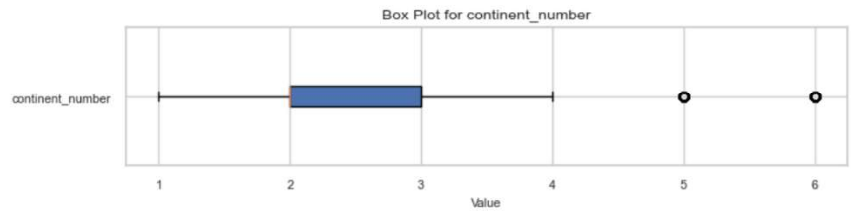

Box Plot for D
D
176
178
180
182
184
Value


Box Plot for prev_salary
prev_salary
0
50000
100000
150000
200000
Value

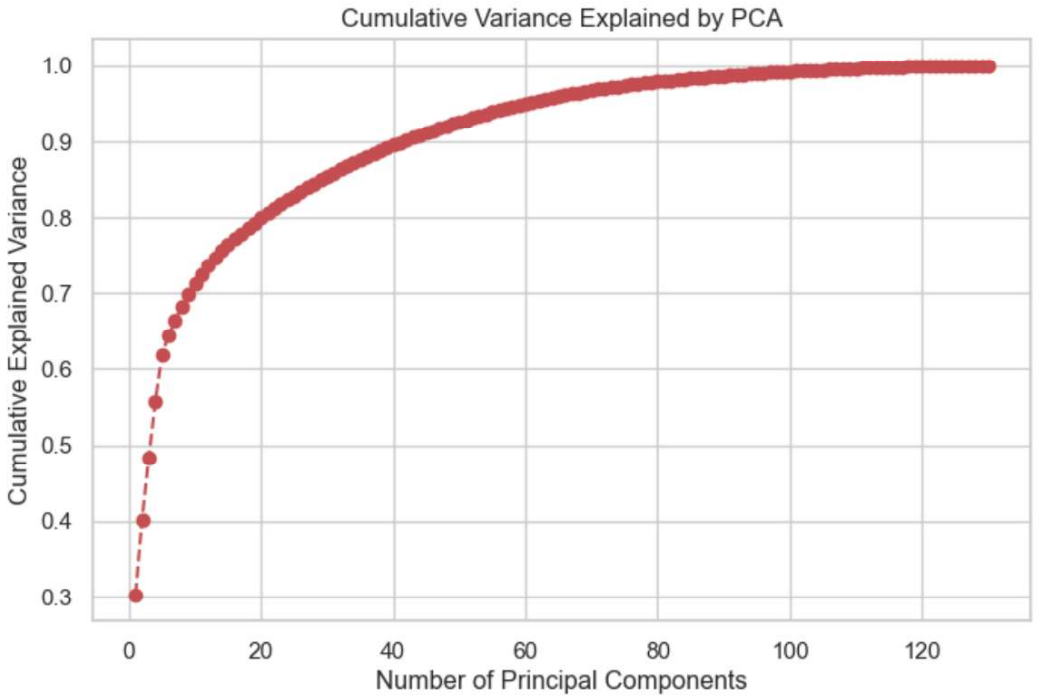

Box Plot for education_encoded
education_encoded
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Value


Box Plot for sex_encoded
sex_encoded
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Value


Box Plot for C_encoded
C_encoded
0
1
2
3
4
5
6
7
Value


Box Plot for continent_number
continent_number
1
2
3
4
5
6
Value

Appendix 7


Cumulative Variance Explained by PCA
Cumulative Explained Variance
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
120
Number of Principal Components

Appendix 8

Logistic Regression Hyperparameters:

C=0.1: Controls the inverse of regularization strength. A lower C value increases bias by penalizing large coefficients. It decreases variance by preventing the model from becoming too complex, thus reducing overfitting.
max_iter=50: Sets the maximum number of iterations. ensures convergence. potentially increasing bias by not reaching the optimal solution.
penalty='l1': Lasso regularization promotes sparse models and feature selection by driving some coefficients to zero. It increases bias but simplifies the model and reduces variance by lowering complexity, making the model less sensitive to training data.
solver='saga': The 'saga' solver is optimized for large datasets and helps reduce variance by efficiently finding a solution without getting stuck in local minima.

Default Hyperparameters that we haven't defined manually:
dual=False**.**
tol=1e-4.
fit_intercept=True.
intercept_scaling=1.
class_weightNone.
multi_class='deprecated'.
verbose, default=0.

Gaussian Naive Bayes Hyperparameters:

'var_smoothing': 0.021544346900318822
Stabilizes variance, reducing sensitivity to fluctuations and improving generalization. Slightly increasing bias as it generalizes the model more.

Default Hyperparameters that we haven't defined manually:
priors=None.

Random Forest Hyperparameters:

max_depth=20: Limits tree depth to 20, reducing bias by capturing detailed patterns but can increase variance due to potential overfitting.
max_features='sqrt': Uses the square root of the total features for splits. This increases bias slightly but reduces variance by adding randomness, making the ensemble more robust.
n_estimators=315: Uses 315 trees. More trees reduce bias and variance by averaging errors, but beyond a certain number, they don't significantly improve performance and increase computational cost.

Default Hyperparameters that we haven't defined manually:
criterion='gini'.
bootstrap=True.
min_samples_split=2.
min_samples_leaf=1.

min_weight_fraction_leaf=0.0.
max_leaf_nodes=None.

AdaBoost Hyperparameters:

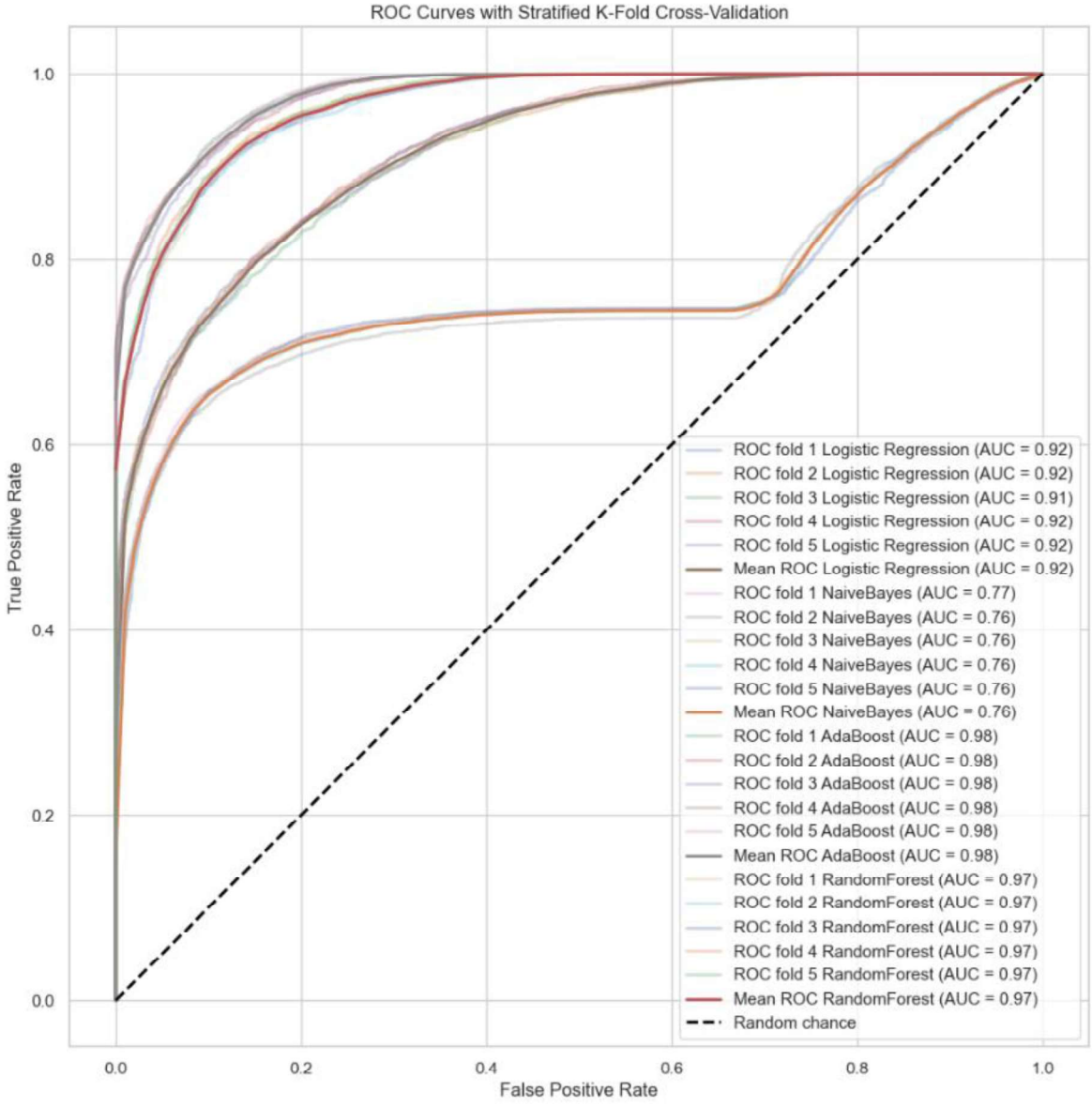
base_estimator__max_depth= 3:
The maximum depth of the base estimator (decision tree).
Shallow trees (lower max_depth) are used to keep the base learners simple, increasing bias. low max_depth reduces variance by ensuring that the base learners do not overfit, and the boosting process focuses on difficult cases iteratively.

learning_rate=0.1:
The contribution of each base learner to the final model. A lower learning rate increases bias because each model's contribution is scaled down, and more iterations are needed to reduce error. It reduces variance by smoothing the model's learning process, preventing it from becoming too sensitive to individual data points.

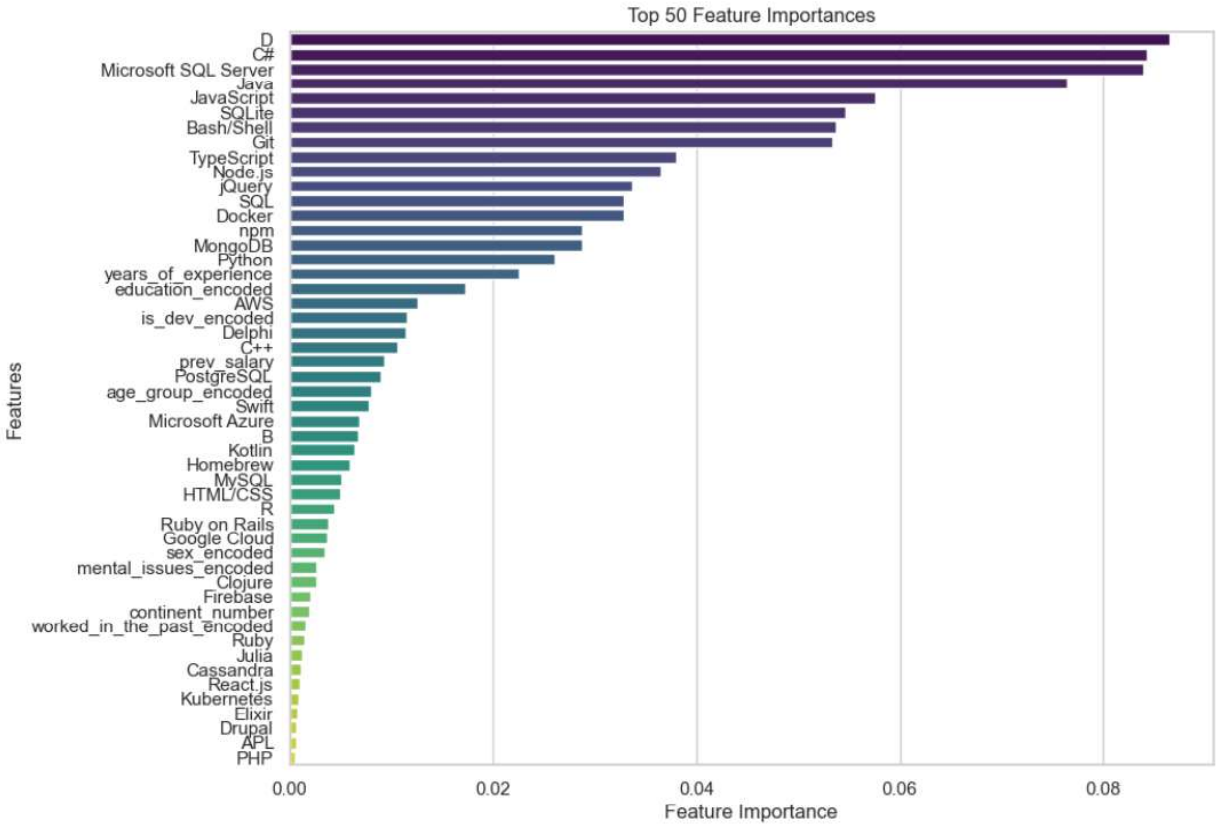
n_estimators=200:
The number of boosting stages. More boosting stages reduce bias by iteratively focusing on the residuals of previous models. Adding more stages reduces variance as the ensemble learns from mistakes iteratively, but beyond a certain point, it may not provide significant improvements and could increase computational cost.

Default Hyperparameters that we haven't defined manually:
algorithm='SAMME.R'

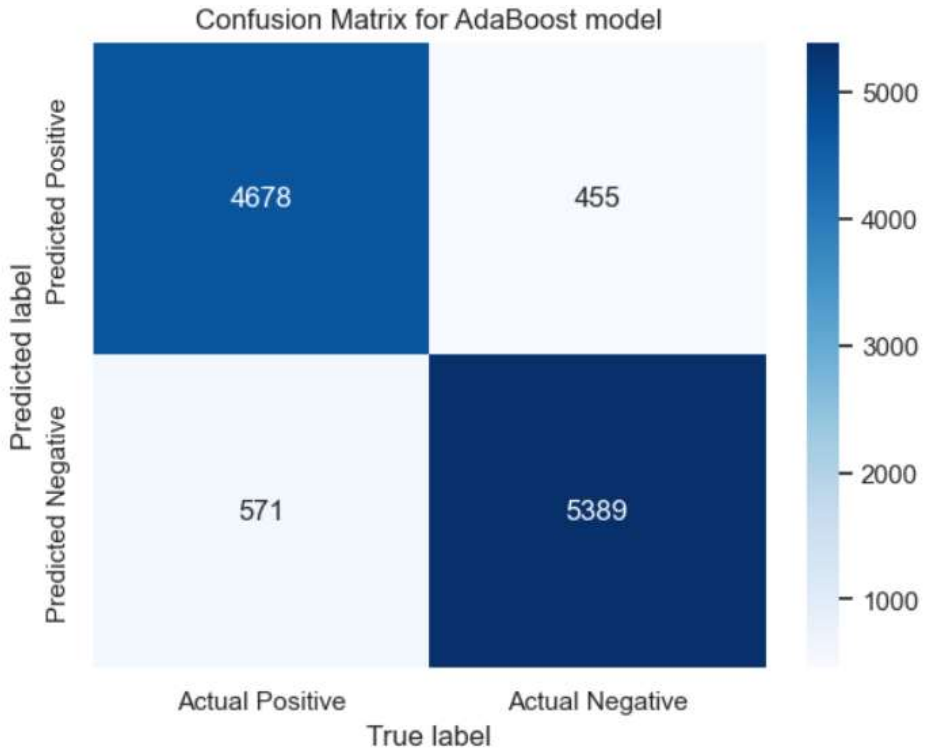
Appendix 9


ROC Curves with Stratified K-Fold Cross-Validation
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 1 Logistic Regression (AUC = 0.92)
ROC fold 2 Logistic Regression (AUC = 0.92)
ROC fold 3 Logistic Regression (AUC = 0.91)
ROC fold 4 Logistic Regression (AUC = 0.92)
ROC fold 5 Logistic Regression (AUC = 0.92)
Mean ROC Logistic Regression (AUC = 0.92)
ROC fold 1 NaiveBayes (AUC = 0.77)
ROC fold 2 NaiveBayes (AUC = 0.76)
ROC fold 3 NaiveBayes (AUC = 0.76)
ROC fold 4 NaiveBayes (AUC = 0.76)
ROC fold 5 NaiveBayes (AUC = 0.76)
Mean ROC NaiveBayes (AUC = 0.76)
ROC fold 1 AdaBoost (AUC = 0.98)
ROC fold 2 AdaBoost (AUC = 0.98)
ROC fold 3 AdaBoost (AUC = 0.98)
ROC fold 4 AdaBoost (AUC = 0.98)
ROC fold 5 AdaBoost (AUC = 0.98)
Mean ROC AdaBoost (AUC = 0.98)
ROC fold 1 RandomForest (AUC = 0.97)
ROC fold 2 RandomForest (AUC = 0.97)
ROC fold 3 RandomForest (AUC = 0.97)
ROC fold 4 RandomForest (AUC = 0.97)
ROC fold 5 RandomForest (AUC = 0.97)
Mean ROC RandomForest (AUC = 0.97)
Random chance

Appendix 10


Top 50 Feature Importances
Features
D
C#
Microsoft SQL Server
Java
JavaScript
SQLite
Bash/Shell
Git
TypeScript
Node.js
jQuery
SQL
Docker
npm
MongoDB
Python
years_of_experience
education_encoded
AWS
is_dev_encoded
Delphi
C++
prev_salary
PostgreSQL
age_group_encoded
Swift
Microsoft Azure
B
Kotlin
Homebrew
MySQL
HTML/CSS
R
Ruby on Rails
Google Cloud
sex_encoded
mental_issues_encoded
Clojure
Firebase
continent_number
worked_in_the_past_encoded
Ruby
Julia
Cassandra
React.js
Kubernetes
Elixir
Drupal
APL
PHP
0.00
0.02
0.04
0.06
0.08
Feature Importance

Appendix 11




```
Accuracy: 0.9075092400613
Precision: 0.9221423682409309
Recall: 0.9041946308724832
Specificity: 0.911357880381843
```

Appendix 12

Contribution of each one of us:

Part 1: Exploration: Mika, Visualization: Yali

Part 2: Together

Part 3: Logistic Regression, Random forest- Mika, Gaussian Naïve Bayes, AdaBoost- Yali

Part 4-6: Together