EMPLOYMENT PREDICTION USING MACHINE LEARNING

A Project work submitted to

MAHATMA GANDHI UNIVERSITY

In Partial Fulfillment of the Requirements for the Award of the Degree of

MASTER OF SCIENCE IN STATISTICS WITH DATA SCIENCE

Submitted by

SREELAKSHMI EU

Reg.No.230011021118

POST GRADUATE DEPARTMENT OF STATISTICS

SREE SANKARA COLLEGE, KALADY, KERALA

(Reaccredited with A Grade by NAAC

Affiliated to Mahatma Gandhi University, Kottayam)

MARCH 2025

28-03-2025

CERTIFICATE

This is to certify that the dissertation entitled "**Employment Prediction Using Machine Learning"** is a bonafide record of project carried out by **SREELAKSHMI E U**, **PRN:230011021118**, **MSc Statistics with Data Science 2023-25** , under my guidance and supervision in partial fulfilment of the requirements for the completion of **Post Graduate Degree of Science in Statistics with Data Science** at the Department of Statistics, Sree Sankara College, Kalady-683574 , affiliated to Mahatma Gandhi University, Kottayam. It is further certified that the project work has not been previously formed the basis for the award of any other Degree or Diploma or other similar title to any candidate of this or any other university.

Dr. Biju Thomas
Associate Professor
(*Supervising Guide****)***

Head of the Department
Dr. Biju Thomas

Valued by

Name and Signature (with date)
of the External Examiner

College Seal

ICT ACADEMY OF KERALA

Certificate of Internship

Congratulations!

Sreelakshmi E U

Sree Sankara College, Kalady, Ernakulam

Machine Learning & Artificial Intelligence

This certificate is a testimony of your completion of the one-month Internship with the ICT Academy of Kerala in the given project.

Mr. Muraleedharan Manningal
CEO
ICT Academy of Kerala

INTERNSHIP OFFERED BY
ICT ACADEMY OF KERALA

DURATION
1 Month

GRADE *
B

Dates
From: **2025 MAR 03**
To: **2025 APR 03**

Issued On
2025 APR 04

DECLARATION

I, Sreelakshmi E U hereby declare that the dissertation entitled "**Employment Prediction Using Machine Learning"** is a record of original work conducted by me under supervision and guidance of **Dr. Biju Thomas,** Associate Professor, Department of Statistics, Sree Sankara College, Kalady. This project is being submitted in partial fulfilment of the requirements for the degree of **Master of Science in Statistics with Data Science** from Mahatma Gandhi University, Kottayam. I affirm that this project has not been previously submitted as part of any other degree, diploma, associate-ship, fellowship, or similar title, either at this university or any other educational institution Furthermore, I acknowledge that any sources, reference, or materials used in this projects have been duly cited and acknowledged in accordance with academic integrity guidelines.

Place: Kalady **SREELAKSHMI E U**

Date: 28-03-2025

ACKNOWLEDGEMENT

The successful completion of this dissertation owes to the inspiration and constant support that I received from various sources. I avail this opportunity to express my sincere gratitude to all those who helped me directly or indirectly for the completion of work. First and foremost, I am deeply grateful to my guide, **Dr. Biju Thomas**, Associate Professor, Department of Statistics, Sree Sankara College, Kalady for his guidance. I am thankful to my internship mentor **Ms. Maya Mohan**, Instructor at ICT Academy Kerala. Her expertise and support have been instrumental for successfully completing my internship at ICT Academy. I hereby acknowledge my profound gratitude to **Prof.(Dr.)Anilkumar M**, Principal, Sree Sankara College, Kalady for his valuable encouragement in conducting this study. I would also like to thank all the teachers in the department for their supervision, guidance, encouragement, and cooperation throughout this endeavour. Furthermore, I express my heartfelt gratitude to my teachers, parents, friends and all those who have encouraged me in this journey. Their support and belief in my abilities have been instrumental in my success. Finally, I would like to extend my utmost gratitude to the grace of Almighty God for enabling me to successfully complete this training program.

SREELAKSHMI E U

ABSTRACT

The project entitled **"Employment Prediction Using Machine Learning"** is developed in python language and the dataset is stored in Microsoft Excel. This study focuses on predicting whether a candidate will be placed in a job based on their skills, experience, education and other personal and professional attributes. A **Classification** model was developed to predict whether a candidate is likely to be placed or not. Different machine learning algorithms, such as Decision Trees, Random Forests, K-Nearest Neighbours and XGBoost are used to construct predictive models. Various techniques were employed for data preprocessing, and the resulting preprocessed data was utilized for constructing the models. The models performance is assessed using metrics like accuracy and precision, recall, and F1 score. This study aims to identify the most effective machine learning approach for the given classification task. This project displays the practical application of predicting career outcomes and enhancing decision making in recruitment and career development.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

"Employment Prediction" dataset contains a comprehensive collection of information regarding job applicants and their respective employability scores. The dataset has been compiled to assist organizations and recruiters in evaluating the suitability of candidates for various employment opportunities. By utilizing machine learning techniques, this dataset aims to provide valuable insights into the factors influencing employability and enhance the efficiency of the hiring process. Machine learning is a branch of Artificial Intelligence(AI) that focuses on developing algorithms and models capable of learning patterns and making predictions or decisions without being explicitly programmed. It enables computers to automatically learn from data and improve their performance over time. In machine learning, there are several different methods or techniques that can be applied depending on the nature of the problem and the type of the data available. Supervised learning, Unsupervised learning, Reinforcement learning, are some commonly used machine learning methods.

This study is based on an Employment dataset, that is to classify whether or not a candidate will be placed or not. It helps job seekers identify how technologies they have worked with and their level of computer skills can influence their chances of placement.

After collecting the data, the first step is to thoroughly analyse the data and identify the factors. After analysing the data, it needs to undergo different preprocessing techniques, including data cleaning to remove missing values outliers and unnecessary columns. Also data preprocessing includes scaling and encoding of categorical columns of the data. Then split the into training and test sets.

Following data preprocessing, various machine learning techniques can be utilised to develop a model that classifies whether a candidate will be placed or not. Commonly employed algorithms include Decision Tree, K-Nearest Neighbors (KNN), Random Forest and XGBoost Classifiers. Each algorithm has its own strengths and weakness, and we select the most appropriate one based on their requirement and characteristics of the dataset. Evaluation metrics in machine learning are used to assess the performance of a model and determine how well it is achieving its intended objectives. Compare and evaluate the performance of different model using appropriate evaluation metrics and validation techniques, such as Accuracy, Precision, Recall and F1-Score. For predicting whether or not a candidate will be placed, the most accurate model is selected on their performance.

1.2 OBJECTIVES

The major objectives of our project work is focused on the following

- Performing an analysis of the Employment data and the preprocessing the data.
- An evaluation of Decision Tree, K-Nearest Neighbors (KNN), Random Forest and XGBoost classifier using the given set of data.
- Accuracy, Precision, Recall and F1-Score and confusion Matrix computation on above mentioned algorithm.
- Comparative analysis of algorithm in terms of Classification Metrics
- Predict the best accurate model.

CHAPTER 2
LITERATURE REVIEW

The ‘Employment Prediction’ dataset can be used to develop predictive models that assess the employability of new applicants based on their profile attributes, enabling organizations to prioritize candidates efficiently. By analyzing the dataset, recruiters can gain insights into the key attributes and qualifications that contribute most significantly to a candidate's employability.

As per Tom Mitchell (1997), machine learning is defined as the study of algorithms that improve automatically through experience. This theoretical groundwork has enabled the application of various supervised learning models in employment-related tasks. Pratap Dangeti, in Statistics for Machine Learning, emphasizes the importance of statistical foundations and feature engineering in building reliable predictive models—key techniques used in employment prediction to evaluate candidates' attributes such as education, skills, and experience.

Bhardwaj and Pal (2021) developed a predictive hiring model using algorithms like Random Forest and Support Vector Machines (SVM) for the IT sector. Their findings revealed that machine learning can successfully identify suitable candidates and streamline hiring processes. This supports the objectives of the current project, which uses classification models to predict employment outcomes based on various candidate features including education level, computer skills, and prior work experience.

Kapoor et al (2021) and Sundararajan & Wood explored how AI and machine learning technologies are transforming modern HRM practices. These studies highlighted the

effectiveness of predictive analytics in assessing candidate suitability and enhancing recruitment efficiency. Kapoor et al. specifically emphasized the role of AI in automating candidate screening processes, enabling HR departments to focus on more strategic decision-making. Sundararajan & Wood further explored how AI-based systems can predict employee turnover and improve retention strategies, a perspective that aligns with the focus on identifying influential attributes in the employment prediction dataset.

The use of advanced machine learning techniques, including ensemble learning methods, is further supported by the findings of Kapoor et al. (2021). They observed that ensemble methods such as Random Forest and XGBoost demonstrate higher predictive accuracy, especially in datasets with multiple features and potentially imbalanced target variables. This aligns with the present study's findings, where the XGBoost model outperformed other models in terms of accuracy and F1 score, indicating its robustness in employment prediction tasks.

Additionally, Bhardwaj and Pal (2021) highlighted the impact of technical skills and work experience as influential factors in hiring decisions within the IT sector. This insight is corroborated by the current study, where candidates with extensive computer skills and specific technical expertise demonstrated higher placement probabilities. This finding suggests that targeted skill development and training programs could effectively enhance employability.

In conclusion, the literature underscores the significance of integrating machine learning in HRM, particularly in recruitment and employment prediction. By leveraging predictive models to assess candidate profiles, organizations can streamline hiring processes, reduce recruitment biases, and make data-driven hiring decisions. The present study aligns with these trends, contributing to the existing body of knowledge by implementing advanced predictive models and data preprocessing techniques to forecast employment outcomes effectively. Furthermore,

the use of comprehensive datasets and robust algorithms not only enhances model accuracy but also provides actionable insights into the factors that most significantly impact employability.

CHAPTER 3
METHODOLOGY

In this project, the methodology refers to the systematic approach or framework followed to develop, train, evaluate and deploy machine learning models. The initial step in a machine learning project involves gathering the necessary data for training and evaluation purposes. After collecting the data, the subsequent steps involve performing exploratory data analysis, preprocessing the data, selecting suitable machine learning models, and evaluating their performance.

3.1 MACHINE LEARNING

Machine learning is a subfield of artificial intelligence(AI) that focuses on the development of algorithms and models that enable to learn from and make predictions or decisions based on data, without being explicitly programmed for specific tasks.

The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making process based on data inputs. The need for machine

learning is increasing. Machine learning is essential because it can complete tasks that are too difficult for a person to complete alone.

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. Machin learning can be broadly categorized into four main types: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. These categories describe different approaches to training and learning patterns from data. Let's explore this models in details,

3.1.1 Supervised Learning

Supervised learning is the type of machine learning in which machines are trained using well "labelled" training data, and on the basis of tat data, machines predict the output. The labelled data means some input data is already tagged with the correct output. In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

The goal is to learn a mapping between the input features and output labels, enabling the algorithm to make accurate predictions or decision on unseen data. Supervised learning therefore uses patterns to predict label values on additional unlabeled data.

Supervised learning can be further categorized into different types based on the nature of the output labels and the specific task being performed. They are Regression and Classification.

1)Regression

Regression in supervised learning is a type of predictive modelling technique that aims to establish a relationship between a dependent variable and one or more independent variables.

It is used to predict continuous numeric values based on input features. The main goal of regression is to find a mathematical function that best describes the relationship between the independent variables (also known as features, predictors, or input variables) and the dependent variable (also called the target variable or output variable). This function can then be used to make predictions on new, unseen data.

The process of regression involves training a regression model on a labelled dataset, where the input features and their corresponding target values are known. During training, the model learns to capture the patterns and correlations in the data and find the best fitting function that minimizes the prediction errors. Linear regression, polynomial regression, decision tree regression, support vector regression and random forest regression are some common regression algorithms.

2)Classification

The Classification is a supervised learning technique that is used to identify the category of new observation on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observations into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as target/labels or categories. A Classification problem is when the output variable is a category to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.

If the classification problem has only two possible outcomes, then it is called as Binary Classifier. And if a classification problem has more than two outcomes, then it is called as Multi-class Classifier. Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying

attributes and uses it I classifying new data. Classification Algorithms can be further divided into the mainly two category.

*** Linear Models**

* Logistic Regression

* Support Vector Machines

*** Non-linear Models**

* K-Nearest Neighbours

* Kernel SVM

* Naïve Bayes

* Decision Tree Classification

* Random Forest Classification

3.1.2 Unsupervised Learning

In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labelled data, machine learning methods that facilitate unsupervised learning are particularly valuable. The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next.

In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together.

3.1.3 Semi-Supervised Learning

Semi-supervised learning can be considered a hybrid approach that combines elements of both supervised and unsupervised learning. It is a method that uses a small amount of labelled data and to train a model. The goal of semi-supervised learning Is to learn a function that can accurately predict the output variable based on the input variables, similar to supervised learning. However, unlike supervised learning, the algorithm is trained on a dataset that contains both labelled and unlabelled data.

Semi-supervised learning is particularly useful when there is a large amount of unlabeled data available, but it’s too expensive or difficult to label all of it.

3.1.4 Reinforcement Learning

Reinforcement learning is an area of machine learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

3.2 DATASET COLLECTION

In machine learning, a dataset collection refers to the process of gathering and assembling a set of data samples that will be used to train, validate and test machine learning models. These datasets are crucial for developing, testing, and evaluating the performance of various algorithms and models.

The 'Employment Prediction' dataset used for this analysis can be downloaded from kaggle. It contains 73,462 rows and 15 columns**.**

3.3 EXPLORATORY DATA ANALYSIS (EDA)

After the data collecting the next step is exploratory data analysis. Exploratory data analysis (EDA) is an approach that is used to analyze the data and discover trends, patterns or check assumptions such as outliers, missing values in data with the help of statistical summaries and graphical representation. EDA is typically performed at the initial stages of data analysis project to gain a better understanding of the data before applying more advanced statistical or machine learning techniques.

EDA starts by summarizing the key characteristics of the dataset. This includes examining the dimensions of the dataset, checking the data types of each features, understanding the range and distribution of values, and identifying any missing values or inconsistent data. EDA involves computing and analyzing descriptive statistics such as mean, median, standard deviation, minimum, maximum, quartiles and correlations between features. These statistics provide an overview of the central tendencies, variabilities and relationships within the data. Data visualizations I the graphical representations of data in order to visually communicate patterns, trends and insights. Various plots and charts, such as histograms, scatter plots, box plots, heat map, and bar graphs, can be used to visualize the distribution of the data, identify

outliers, explore features interactions, and detect any trends or clusters in the dataset. Also making it more accessible and impactful to a wider audience.

3.4 DATA PREPROCESSING

Data preprocessing refers to the steps and techniques applied to prepare and transform raw data into a format suitable for a machine learning model. It involves cleaning, transforming and organizing the data to ensure its quality, consistency and compatibility with the chosen analysis methods. Data preprocessing is a crucial step in the overall data analysis pipeline and helps in improving the accuracy and reliability of the results. It involves the following steps:

3.4.1 Missing Values

Handling missing values in a dataset is an essential step in data preprocessing and analysis. Missing values can arise due to various reasons such as data collection errors, incomplete records, non response in surveys and data entry errors. Missing values can be imputed using techniques such as mean, median, mode. If missing values are not handled properly, they can lead to biased or inaccurate results.

Mode imputation was used to handle missing values in 'Employment prediction' dataset.

3.4.2 Outliers

Outliers refers to data points that significantly deviate from the majority of the other data points in a dataset. Outliers can arise due to various reasons, such as data entry errors, measurement errors, or genuinely extreme observations. Outliers can have a significant impact on the results of machine learning algorithms, leading to biased or inaccurate predictions. Therefore, identifying and outliers appropriately is crucial.

Boxplots are commonly used for visualizing and detecting outliers. Capping is a method used to remove outliers by setting a threshold or limit beyond which extreme values are placed with

a predetermined value. Instead of removing outliers entirely from the dataset, capping allows the retention of the observation but constraints its impact on subsequent analysis. The "iqr_capping" function is used to compute the lower whisker and upper whisker values for a particular variable in the DataFrame. The values above the upper whisker are replaced with the upper whisker value, and the values below the lower whisker are replaced with the lower whisker value. The resulting DataFrame has capped values.

Capping method was used to handle outliers in the 'Employment Prediction' dataset.

3.4.3 Encoding

Machine learning models can only work with numerical values. Encoding is a technique of converting categorical variables into numerical values so that it could be easily fitted to a machine learning model. The common encoding methods are One-hot encoding and label encoding. Frequency encoding and ordinal encoding are other kind of encoding techniques.

Within the Employers prediction dataset categorical variables are encoded using these four (One-hot, label, frequency and ordinal) encoding technique.

Label encoding: Label encoding converts categorical variables into numerical values. It assigns a unique numeric label to each category in the variable. Label encoding is commonly used for variables with inherent ordinal relationships, where the categories have a specific order or rank.

One-hot encoding: One-hot encoding converts categorical variables into numerical values. It creates new columns for each unique category in the original features. For each r corresponding to the present category is marked as 1 and others are marked as 0.

Ordinal encoding: Ordinal encoding converts categorical variables into numerical values based on their rank. This encoding is suitable for features where the categories have order (example: Education level)

Frequency encoding: Frequency encoding converts categorical variables into its frequencies. It is useful when there is correlation between categories and the target variables.

3.4.4 Data Balancing

Handling imbalanced data is an important consideration in machine learning. Over sampling techniques can be used to address the issue of imbalanced datasets, where the number of instances in one class is significantly smaller than the number of instances in another class. This situation can lead to biased models that perform poorly in predicting the minority class. Over sampling aims to mitigate this problem by increasing the representation of the minority class in the training data. The general idea is to create synthetic or duplicate instances of the minority class to balance it with the majority class.

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them.

Since the Employment Prediction dataset is highly balanced there was no need to apply any balancing techniques.

3.4.5 Splitting of Data

Train-test split is a common technique used in machine learning to evaluate the performance of a model. In machine learning, data splitting is typically done to avoid overfitting. The original data is split into two sets, such as **training set** and **testing set**. The training set is the portion of the data used to train the model. It is the dataset on which the model learns the patterns and relationships between the features (input variables) and the target variables (output variables). The testing set, on the other hand, is used to assess the model's performance and evaluate how well it can make predictions on new, unseen data. The testing set should be

independent of the training set and should not be used during the model training process. The train-test split is typically done by specifying a ratio or percentage of the data to allocate to the testing set. Here the data splitted in the ratio 80% for training and 20% for testing.

Let ‘X’ represents the columns except the target column or independent variables used for prediction and ‘Y’ represent the target column or dependent variable.

3.4.6 Scaling

The datasets contain features that are varying in degrees of magnitude, range and units. Therefore in order for machine learning models to interpret these features on the same scale, we need to perform scaling. Scaling is the process of transforming or normalising the features or input variables of a dataset to a specific range or distribution. It is an essential step in preprocessing the data before training a machine learning model. Scaling is performed to ensure that all the features contribute equally to the learning process and to prevent certain features from dominating or having undue influence on the model’s behaviour. Scaling helps to improve the overall performance and generalizations capabilities of machine learning models, leading to more accurate and reliable predictions. Min-Max scaling (Normalization), Standardization (Z-Score scaling) and Robust Scaling are commonly used scaling techniques.

Min-Max Scaling: Min-Max Scaling (Normalization) technique scale the features to a specific range, typically between 0 and 1. It calculates the scaled value of each feature based on the minimum and maximum

values in the dataset. The formula for Min-Max scaling is:

scaled_value=(value-min_value) / (max_value-min_value)

Min=Max scaling is suitable when the distribution of the features is approximately uniform or when preserving the relationships between the data points is important.

3.5 MACHINE LEARNING ALGORITHMS

After the data preprocessing we build suitable models for the data to make predictions. The 'Employment Prediction' dataset belongs to a classification problem. Therefore the classification models used here are Decision tree, Random Forest , KNN and XGBoost.

3.5.1 Decision Tree

Decision Tree is a supervised learning technique that can be used for both classification and regression problems. In a Decision tree, there are two nodes, which are the Decision Node and leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and not contain any further branches.

In a decision tree, for predicting the class of the given dataset, the algorithms starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. To build an optimal decision tree, different criteria can be used for splitting, such as Gini impurity or information gain (entropy).

Below diagram explains the general structure of a decision tree:

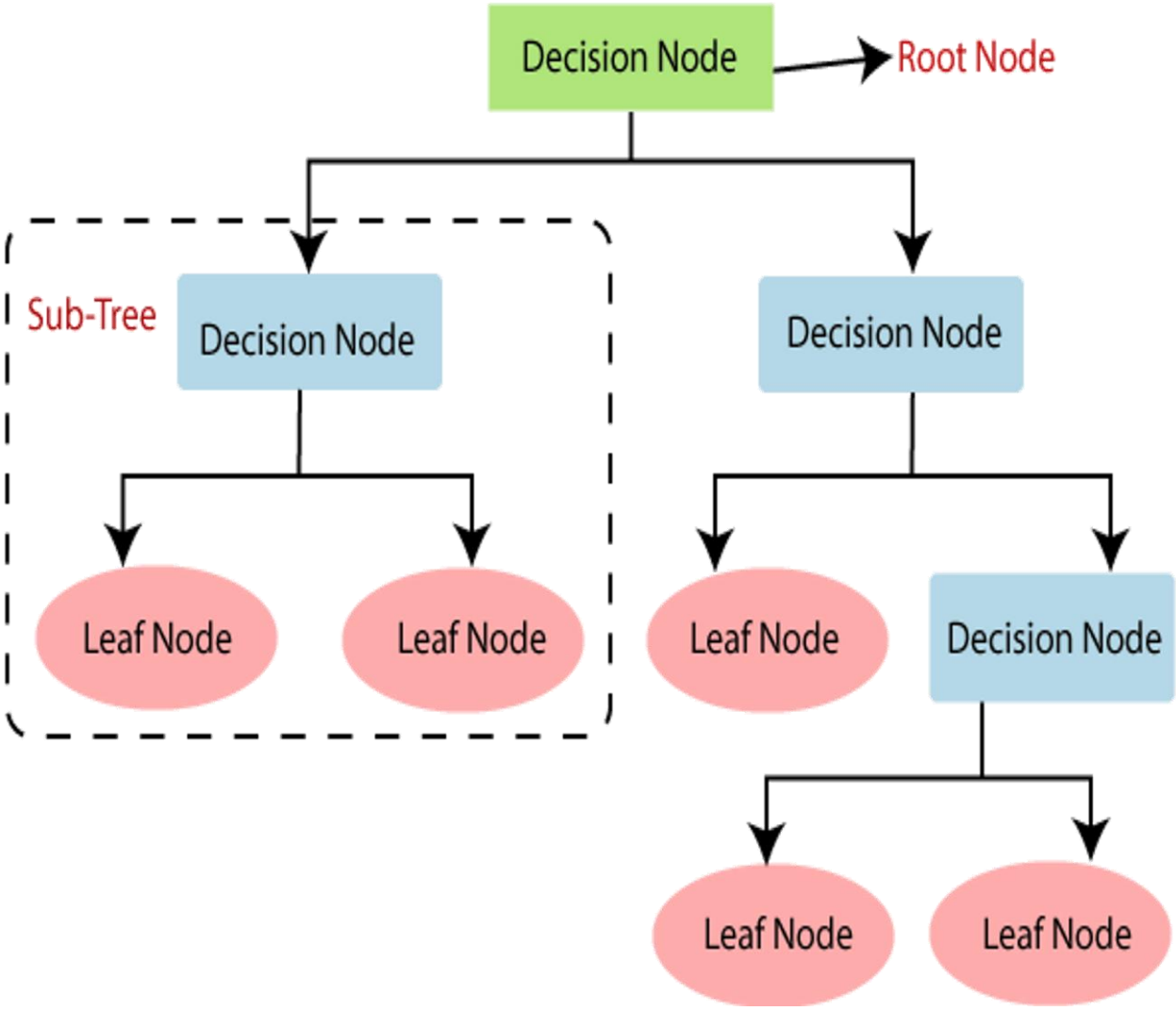


Fig.1 Decision Tree

3.5.2 Random Forest

Random forest (RM) is an ensemble learning method that combines multiple decision trees to make predictions. Random forest is a classifier that combines a number of decision trees on various subsets of the give dataset and takes the average to improve the predictive accuracy of the dataset. Instead of relying on one decision tree, the RF takes the prediction from each tree and based on the majority votes of prediction, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting. The ‘forest’ generated by the random forest algorithm is trained through bagging

or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. Random Forest takes less training as compared to other algorithms and it predicts output with high accuracy, even for the large it runs efficiently.

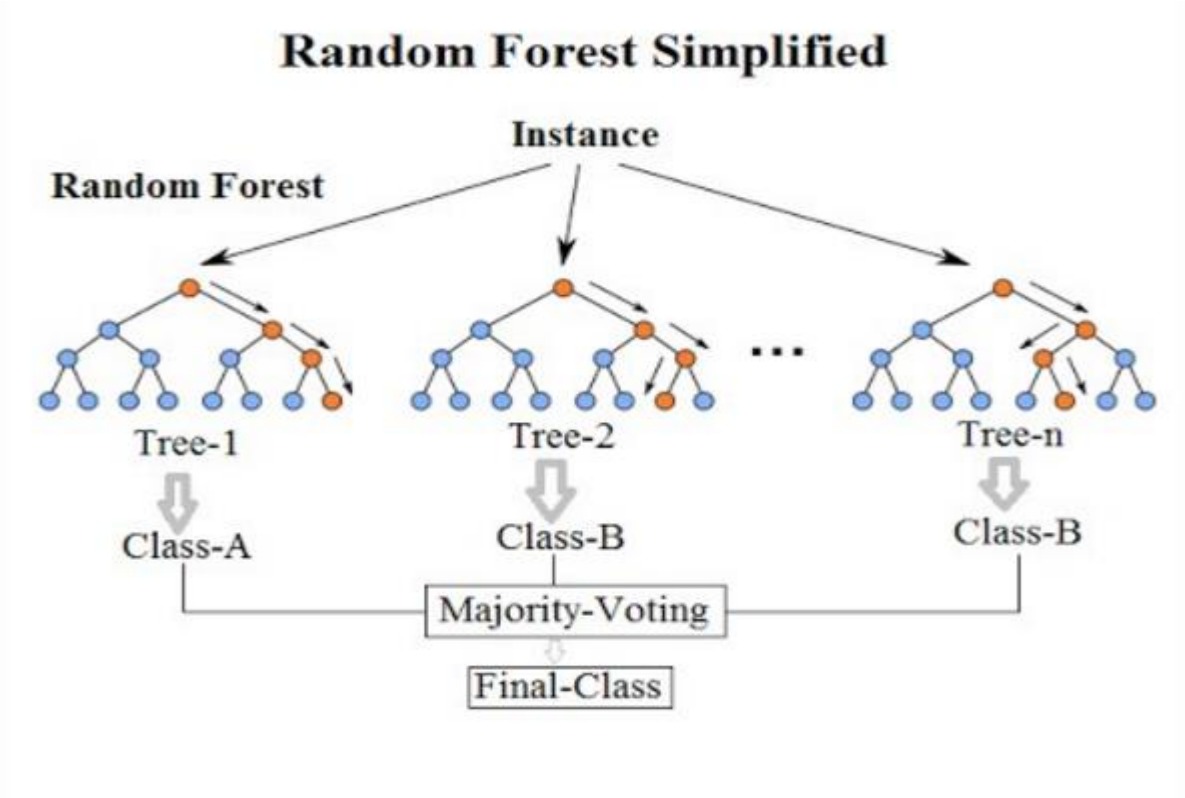


Fig.2 Random Forest

3.5.3 K-Nearest Neighbors (KNN)

The K-Nearest Neighbor algorithm is a pattern recognition model that can be used for classification as well as regression. The K in K-Nearest Neighbor is a positive integer, which is typically small. In either classification or regression, the input will consist of the k closest training examples within a space.

The KNN is a non-parametric supervised machine algorithm. The KNN algorithm is also non-linear. In contrast to simpler models like linear regression, it will work well with data in which the relationship between the independent variable(x) and the dependent variable(y) is not a straight line. It is also called **lazy learner algorithm** because it does not learn from the training

set immediately instead it stores the dataset and the time of classification, it performs an action on the dataset.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1**: Select the number K of the neighbors
- **step-2**: Calculate the Euclidean distance of K number of neighbors
- **step-3**: Take the K nearest neighbours as per the calculated Euclidean distance.
- **step-4**: Among these K neighbours, count the number of the data points in each category.
- **step-5**: Assign the new data points to the category for which the number of the neighbour is maximum.
- **step-6**: Our model is ready

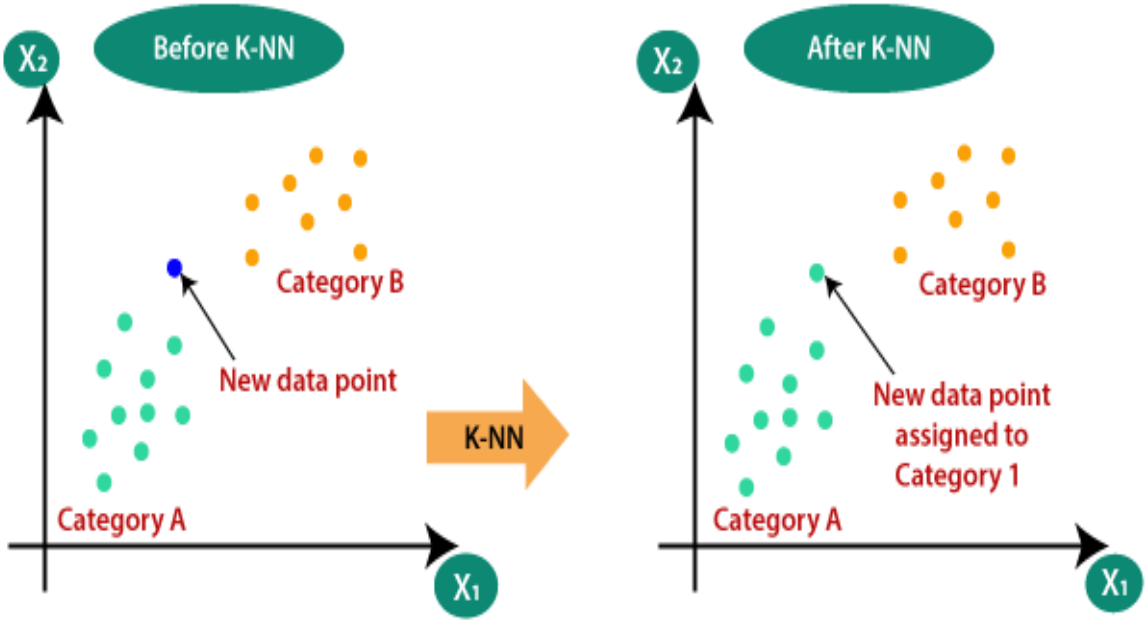


Fig.3

3.5.4 XGBoost

XGBoost is an optimized implementation of gradient boosting and is a type of ensemble learning method. Ensemble learning combines multiple weak models to form a stronger model.

- XGBoost uses decision tree as its base learners combining them sequentially to improve the model's performance. Each new tree is trained to correct the errors made by the previous tree and this process is called boosting.

It has **built-in parallel processing to train models on large datasets quickly**. XGBoost also supports customizations allowing users to adjust model parameters to optimize performance based on the specific problem.

The process can be broken down as follows:

1. **Start with a base learner**: The first model decision tree is trained on the data. In regression tasks this base model simply predict the average of the target variable.
2. **Calculate the errors**: After training the first tree the errors between the predicted and actual values are calculated.
3. **Train the next tree**: The next tree is trained on the errors of the previous tree. This step attempts to correct the errors made by the first tree.
4. **Repeat the process**: This process continues with each new tree trying to correct the errors of the previous trees until a stopping criterion is met.
5. **Combine the predictions**: The final prediction is the sum of the predictions from all the trees.

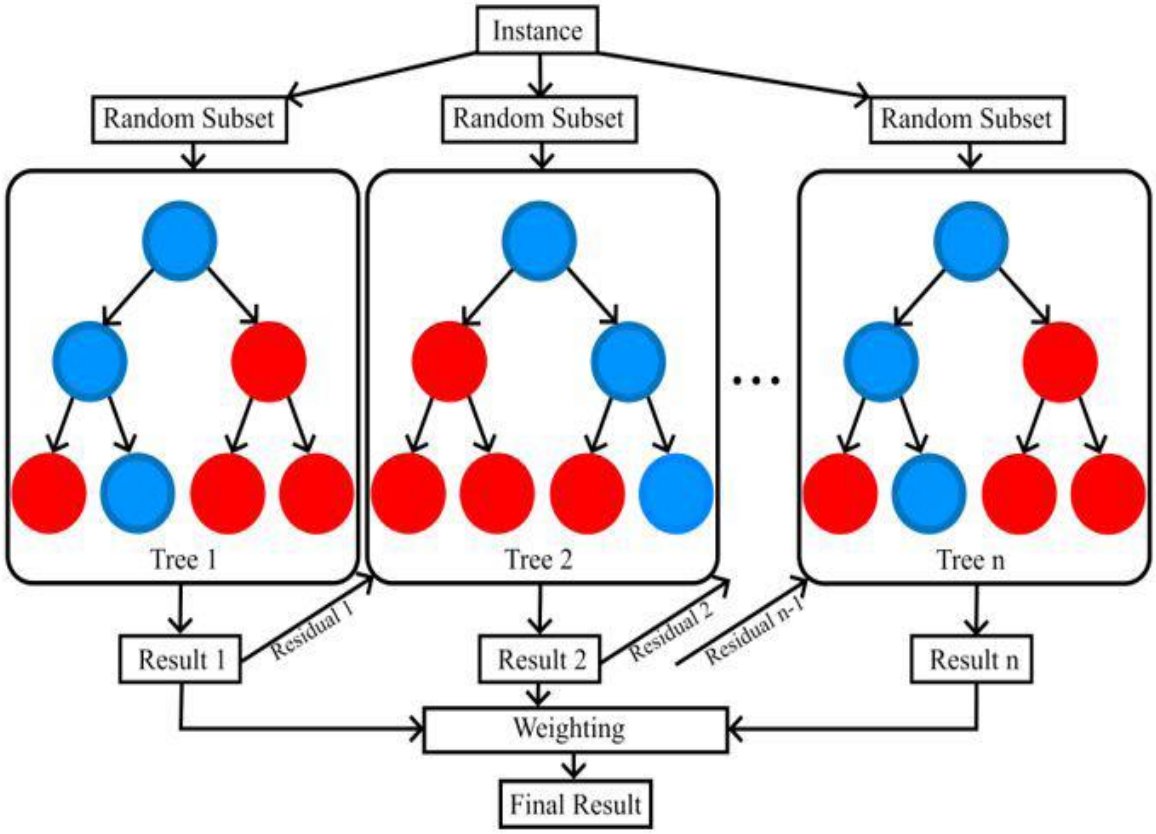


Fig.4 XGBoost

3.6 EVALUATION METRICS

Evaluation metrics in machine learning are quantitative measures used to asses and quantify the performance of machine learning models. These metrics provide insights into how well a model is performing on a given task, such as classification or regression. Here are some commonly used evaluation metrics used in classification:

Accuracy

It calculates the proportion of correctly classified instances over the total number of instances in the dataset.

it can be formulated as:

Accuracy= Number of correct predictions/Total number of predictions

Precision:

The precision determines the proportion of positive predictions that was actually correct. It can be calculated as the True Positive or predictions that actually true to the total positive predictions (True Positive and False Positive).

Recall or Sensitivity:

It represents the ratio of true positive to the sum of true positives and false negatives, measuring the proportion of positive instances correctly identified.

F1 Score:

The F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.

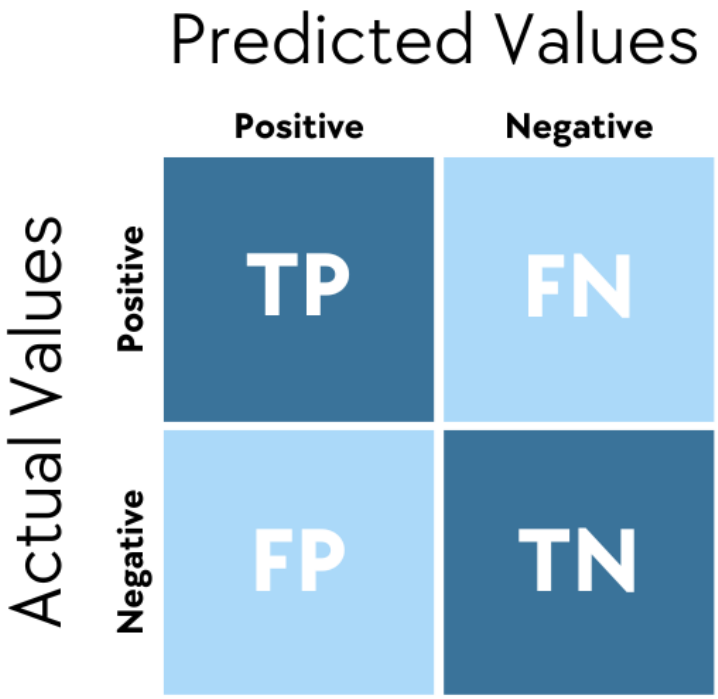
The formula for calculating the F1 is given below:

F1 score = 2* (precision * recall)/ (precision + recall)

Confusion Matrix:

A confusion matrix is a tabular representation that provides a comprehensive view of the performance of a classification model. It summarizes the predictions made by the model against the actual true label of the data. Each row of the matrix corresponds to the true labels, while each column corresponds to the predicted labels.

The values in the confusion matrix represent the counts or frequencies of each type of classification results. A confusion matrix provides an easy summary of the predictive results in a classification problem.



True Positive (TP): The model correctly predicted positive instances.

True Negative (TN): The model correctly predicted negative instances.

False Positive (FP): The model incorrectly predicted positive instances (Type 1 error)

False Negative (FN): The model incorrectly predicted negative instances (Type II error)

From the confusion matrix,

Accuracy=(TP+TN)/(TP+TN+FP+FN)

Precision=TP/(TP+FP)

Recall (True Positive Rate) =TP/(TP+FN)

Specificity (True Negative Rate) =TN/(TN+FP)

3.7 LIBRARIES

To work on this project using machine learning, we need various libraries

- Scikit-learn: This is a widely used machine learning library in Python that provides various algorithms for classification and also offers utilities for data preprocessing, features selection and model evaluation.
- Pandas: Pandas is a python library used for working with data sets. It has functions for analysing, cleaning, exploring and manipulating data.
- Numpy: It is a Python library used for working with arrays.
- Matplot and seaborn: These libraries provide functionalities for data visualization in python.

3.8 DATASET DESCRIPTION

The dataset used for this analysis can be downloaded from kaggle. The dataset predicts whether a candidate will be placed on their skills, experience, and other details. The model helps job seekers to understand which skills and qualifications improve their chances of employment. The dataset contains **73,462** rows and **15** columns. Each columns provides information about various features of data such as, Education Level, Computer Skills, Coding Experience, Main Branch etc. the dataset contains eight categorical and seven numerical columns. There are also outliers in the data.

Feature description

Age: Age of the candidate (<35 and >35)

EdLevel: The highest level of education attained by the candidate (e.g: Bachelor's, Master's, PhD)

YearsCode: Total years of coding experience of the candidate

YearsCodePro: Total years of professional coding experience of the candidate.

Unnamed: represents the index of the candidates.

Country: The country where as the respondent resides.

MainBranch: The main area candidates work in (developer or not-developer)

HaveWorkedWith: The technologies, platforms or databases that a candidate has previously used.

ComputerSkills: Technical abilities of the candidates.

PreviousSalary: Salary of the candidate earned in their last job before seeking new employment.

MentalHealth: Whether a candidate is experiencing stress or other mental health challenges.

Gender: Gender of the candidates.

Accessibility: Whether a candidate can easily access work opportunities (Technology, transport, or other necessary facilities)

Employment: Employment status of the candidates (example: currently working or unemployed)

Employed: Target variable whether the candidate placed or not.

CHAPTER 4
RESULTS

4.1 EXPLORATORY DATA ANALYSIS

Descriptive statistics of Numerical Columns: It gives the mean, median, standard deviation, minimum value, 25th quartile, 50th quartile, 75th quartile and maximum value of the numerical columns in the data.

	Unnamed: 0	Employment	Years Code	YearCode Pro	Previous Salary	Computer Skills	Employed	
count	73462.000000	73462.000000	73462.000000	73462.000000	73462.000000	73462.000000	73462.000000	
mean	36730.500000	0.883096	14.218902	9.098377	67750.260611	13.428221	0.536223	
std	21206.797075	0.321308	9.405172	7.960201	49488.142118	7.057835	0.498690	
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	
25%	18365.250000	1.000000	7.000000	3.000000	28839.000000	8.000000	0.000000	
50%	36730.500000	1.000000	12.000000	7.000000	57588.000000	13.000000	1.000000	
75%	55095.750000	1.000000	20.000000	12.000000	95979.000000	17.000000	1.000000	
max	73461.000000	1.000000	50.000000	50.000000	224000.000000	107.000000	1.000000	

Count plot:

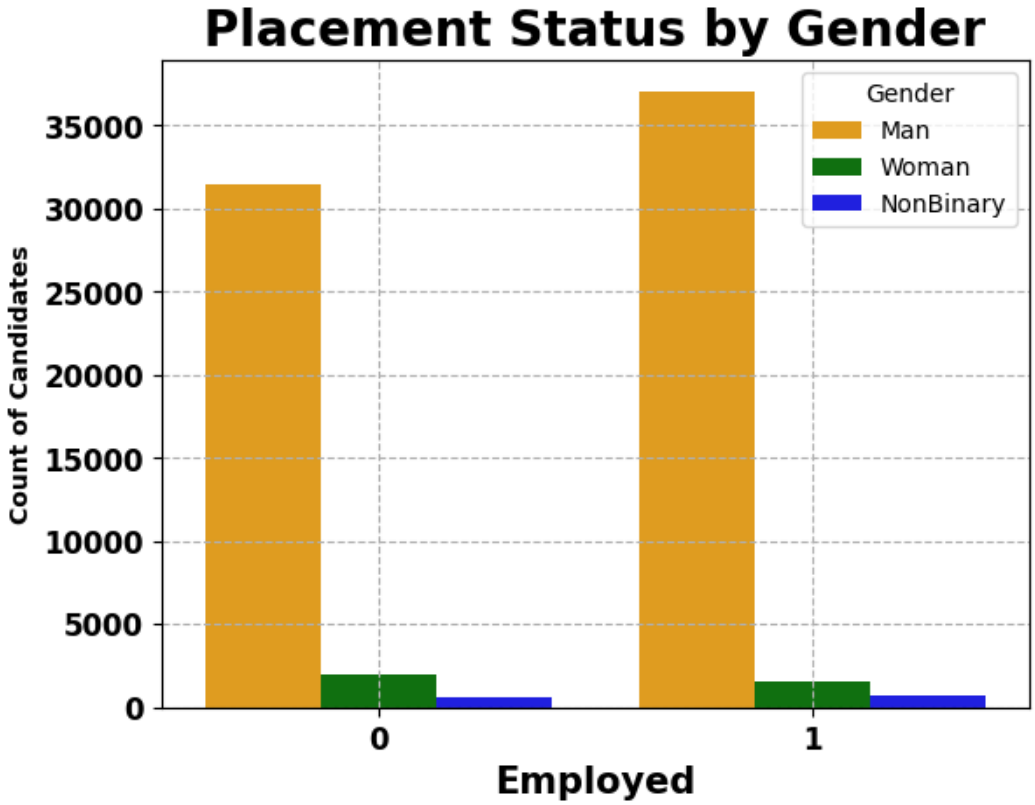
A count plot is a graphical representation commonly used in machine learning and data analysis to visualize the frequency or count of categorical variable in a dataset. A count plot consists of a bar chart where each bar represents a unique category and height of the bar corresponds to the count or frequency of the category in the dataset. The x-axis displays the category while y-axis displays the count or frequency values.

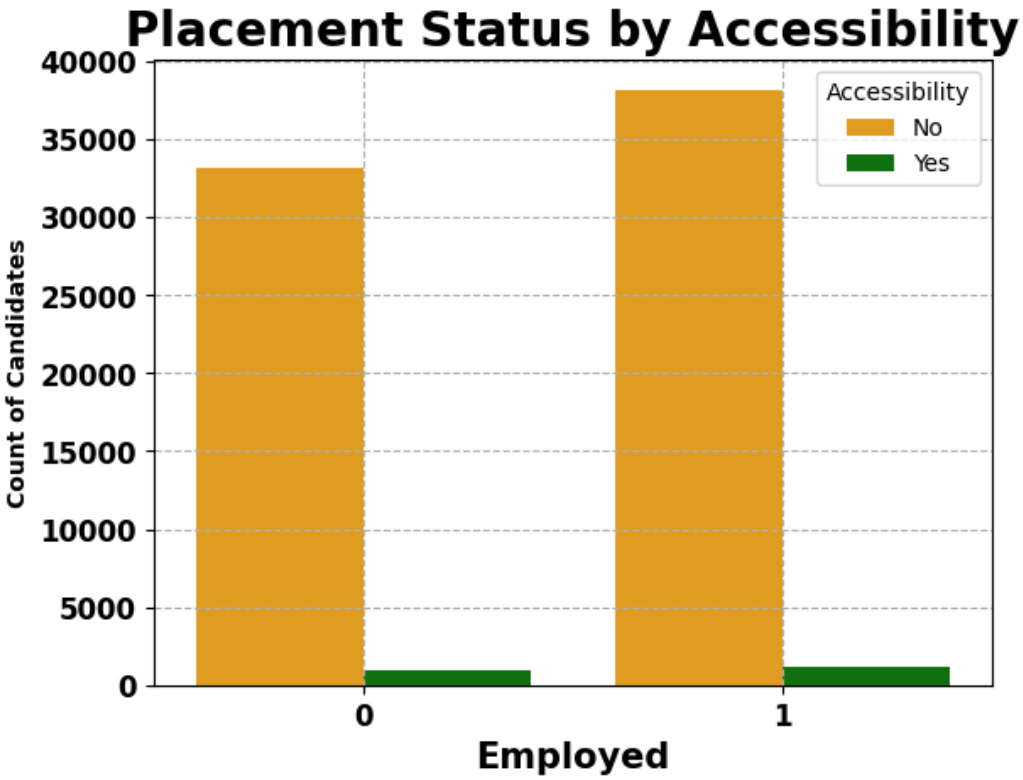
Fig.5 Count plot of Target column

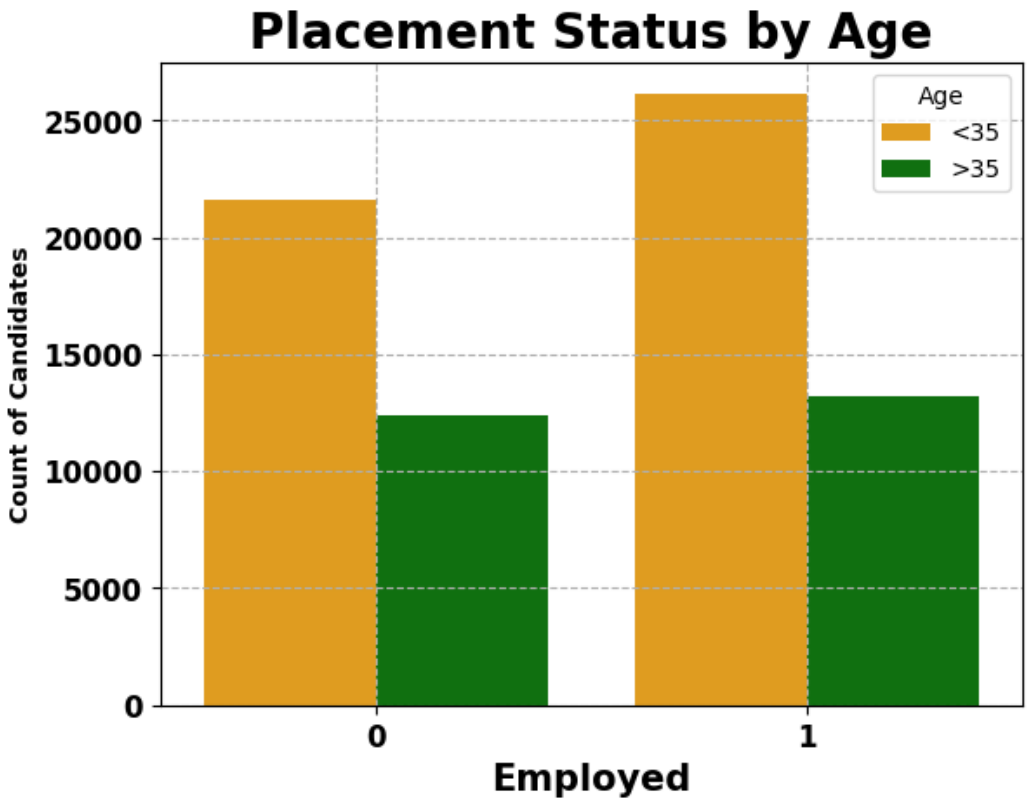
In the classification problem, the target is the 'Employed' column. Where **0** indicates candidate is **not placed** and **1** indicates candidate is **placed**. And is necessary to check whether the given data is balanced or imbalanced.

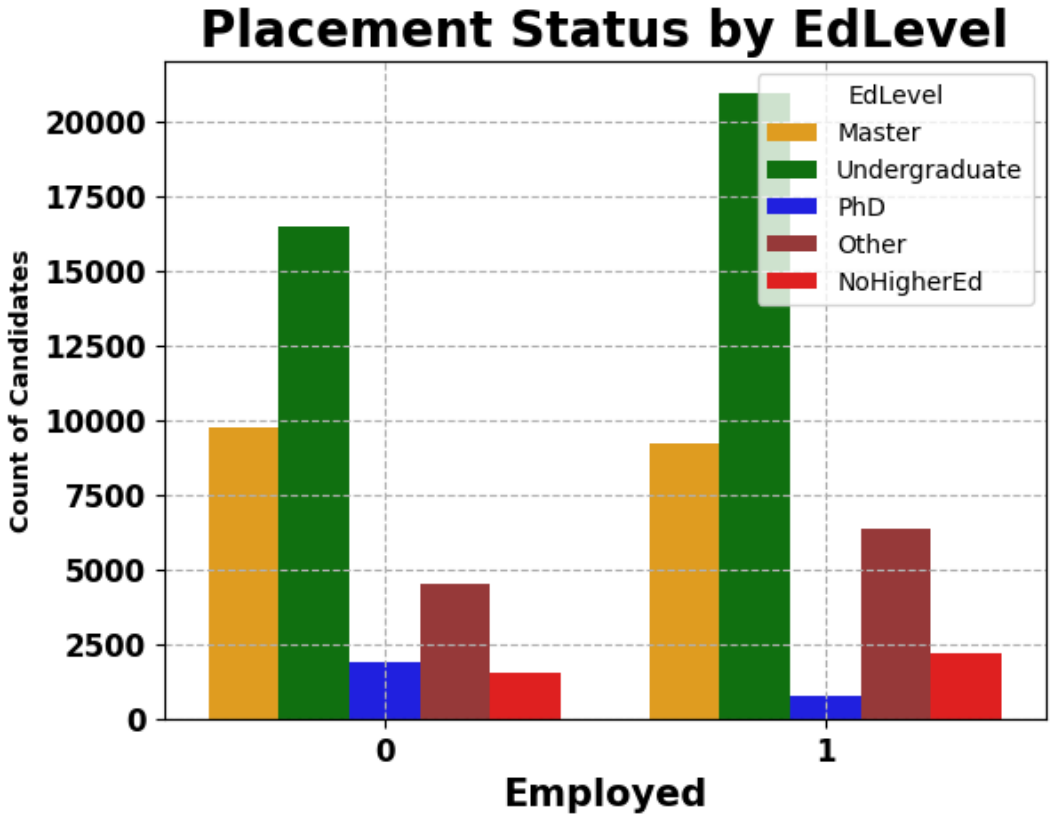
Fig.5 indicates that the target column 'Employed' highly balanced, nearly equal number of placed and not placed candidates.

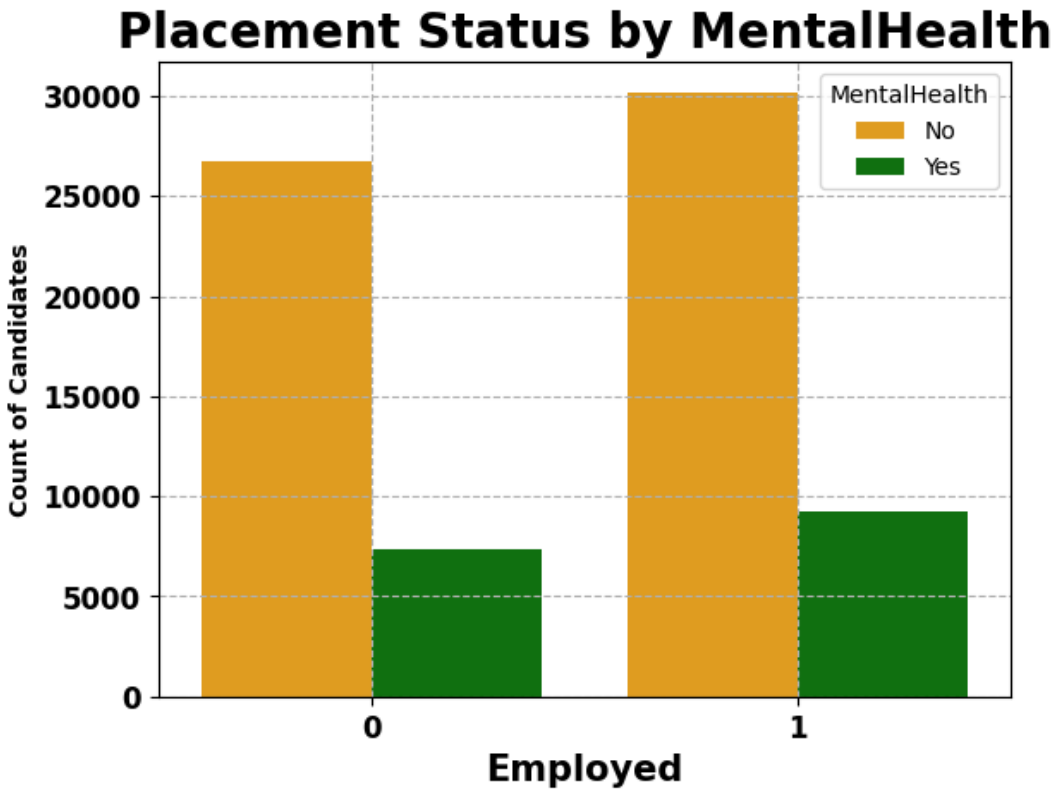
Analysis of Candidate Features for Placement Status

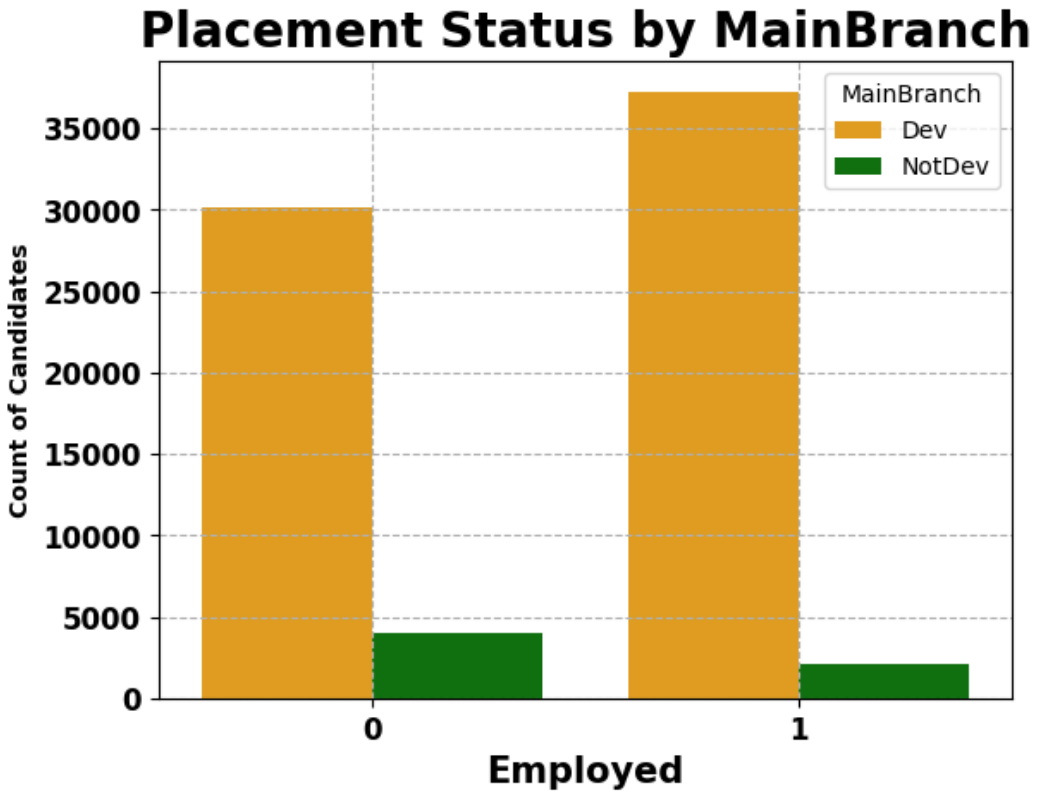


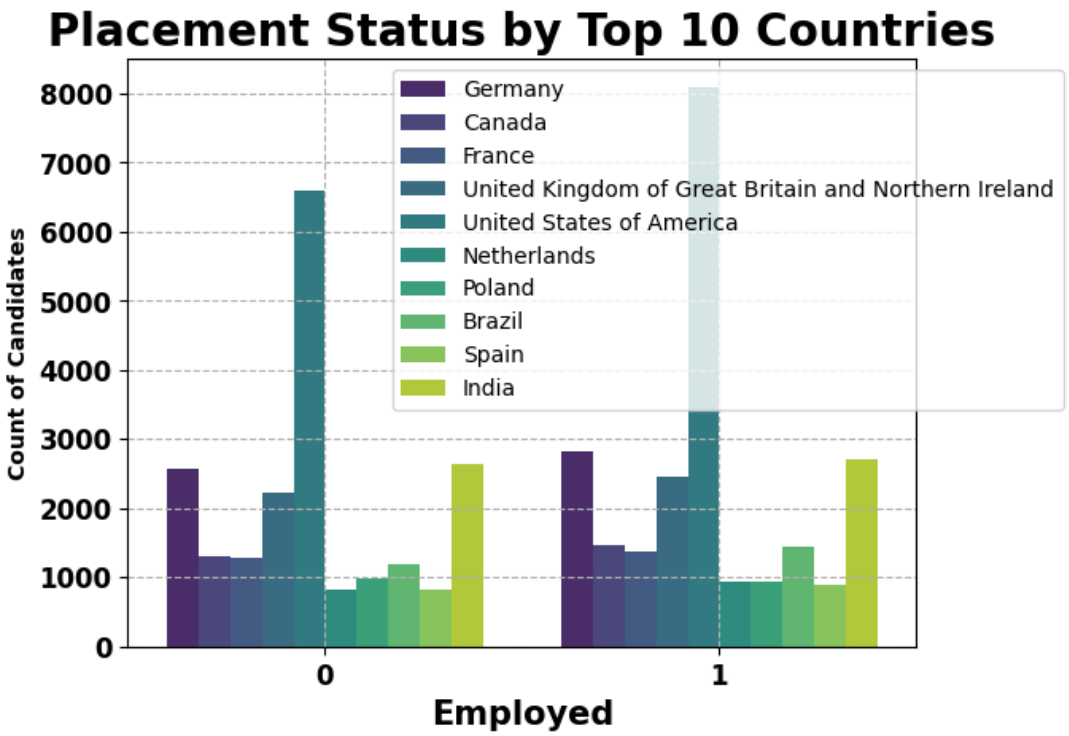



Placement Status by Age
Age
<35
>35
Count of Candidates
25000
20000
15000
10000
5000
0
0
1
Employed


Placement Status by EdLevel
EdLevel
Master
Undergraduate
PhD
Other
NoHigherEd
Count of Candidates
20000
17500
15000
12500
10000
7500
5000
2500
0
0
1
Employed


Placement Status by MentalHealth
MentalHealth
No
Yes
Count of Candidates
30000
25000
20000
15000
10000
5000
0
0
1
Employed


Placement Status by MainBranch
MainBranch
Dev
NotDev
Count of Candidates
35000
30000
25000
20000
15000
10000
5000
0
0
1
Employed


Placement Status by Top 10 Countries
Germany
Canada
France
United Kingdom of Great Britain and Northern Ireland
United States of America
Netherlands
Poland
Brazil
Spain
India
Count of Candidates
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1
Employed

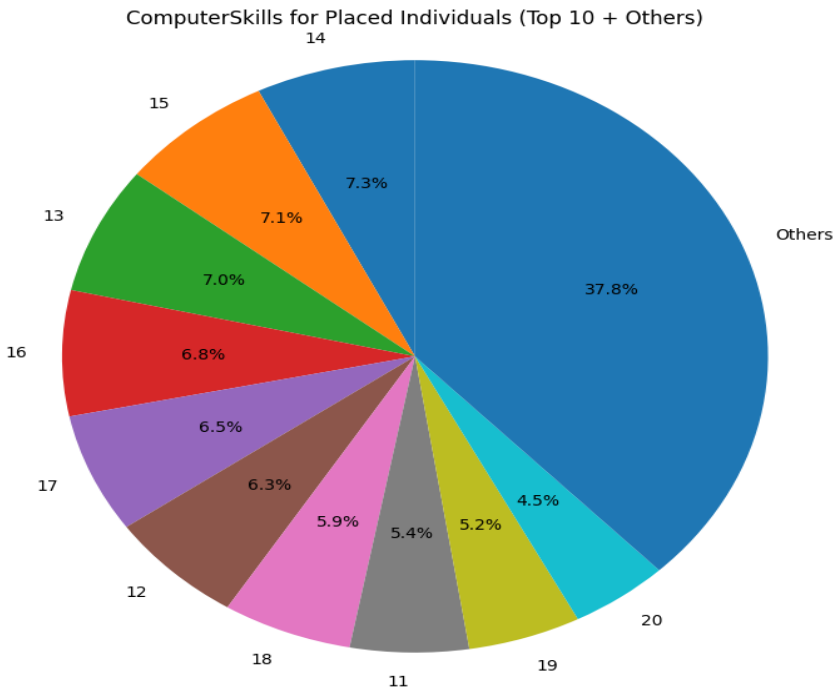

Placement Status
Employment
0
1
Count of Candidates
35000
30000
25000
20000
15000
10000
5000
0
0
1
Employed

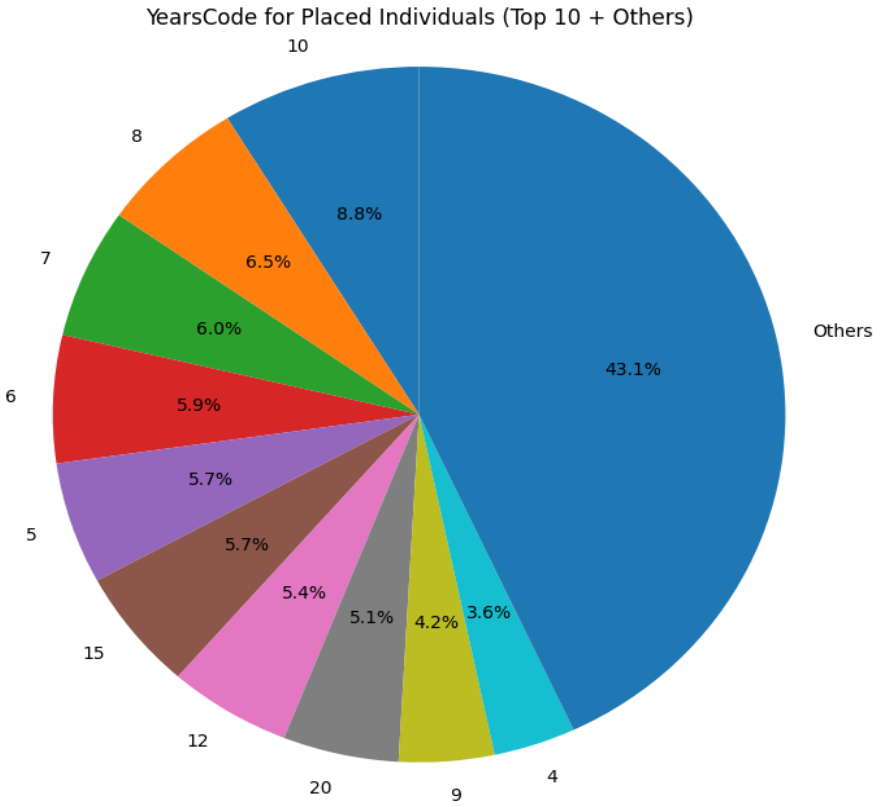
Fig.6

Fig.6 visually explores the relationship between some features and the 'Employed' column of the candidates. Each count plot displays the distribution of each categorical column's values based on whether a candidate placed or not.

Observations from Fig.6:

- Majority of the candidates are Man
- Majority of the candidates are not accessible, as indicated by the Accessibility feature.
- Most candidates are under the age of 35.
- Most candidates who are placed have an educational level of Under-graduate.
- Most candidates have the role of Developers.
- Most of the candidates in the dataset belongs to USA.
- Majority candidates have previous employment experience.
- **Pie Chart:**





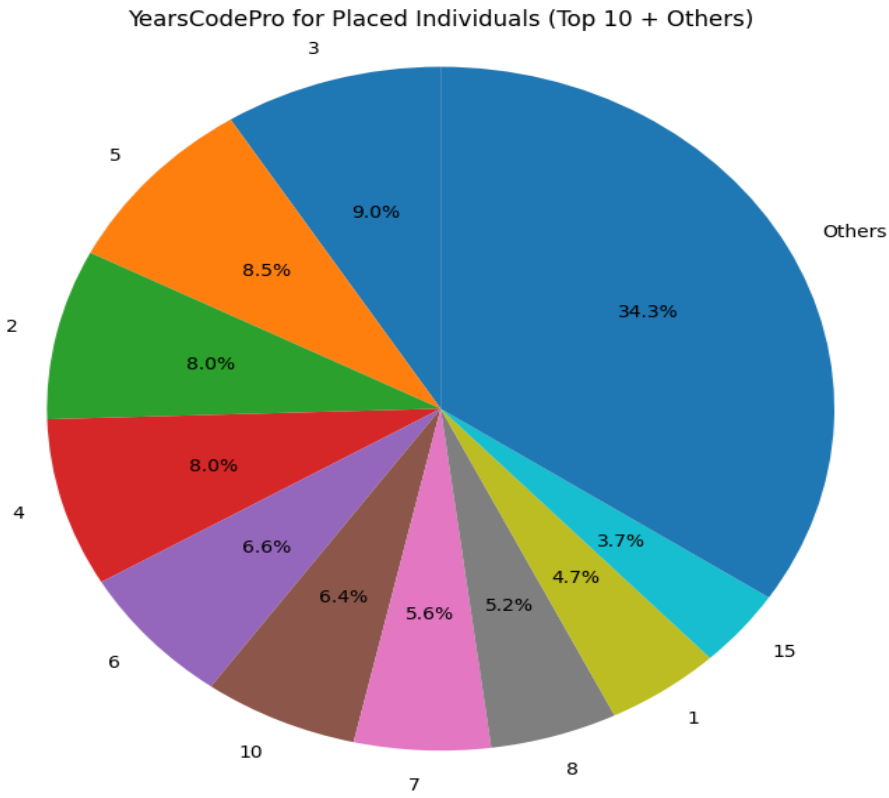


Fig.8

The above figure represents the pie chart of ComputerSkills, YearsCode and YearsCodePro.

Observations from Fig.8:

- Candidates with a ComputerSkills level of 14 have a higher placement rate compared to others.
- Candidates with 10 YearsCode have a higher placement rate compared to others.
- Candidates with 3 YearsodePro have a higher placement rate compared to others.

4.2 DATA PREPROCESSING

The column ‘Unnamed: 0’ has been excluded from the dataset due to its lack of relevance.

Handling Missing Values:

The ‘HaveWorkedWith’ feature contains **63** missing values. In this feature, initially, all technologies were organized into broader categories such as Programming Languages, Databases, Web Frameworks, Cloud Platforms and DevOps Tools. After arranging, the most common technologies (mode) of the feature was identified, and the 63 missing values were replaced with most common technologies.

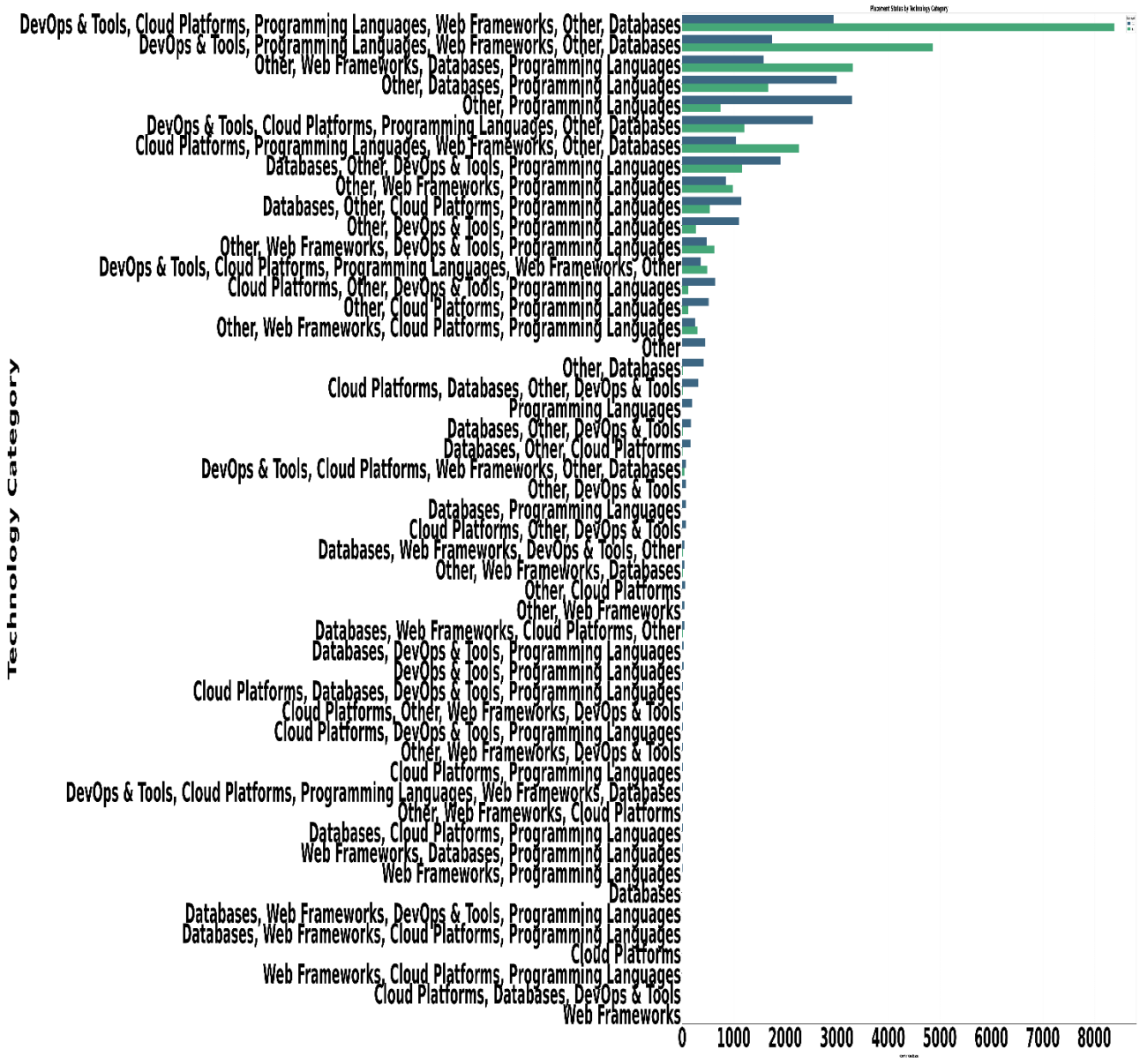


Fig.9

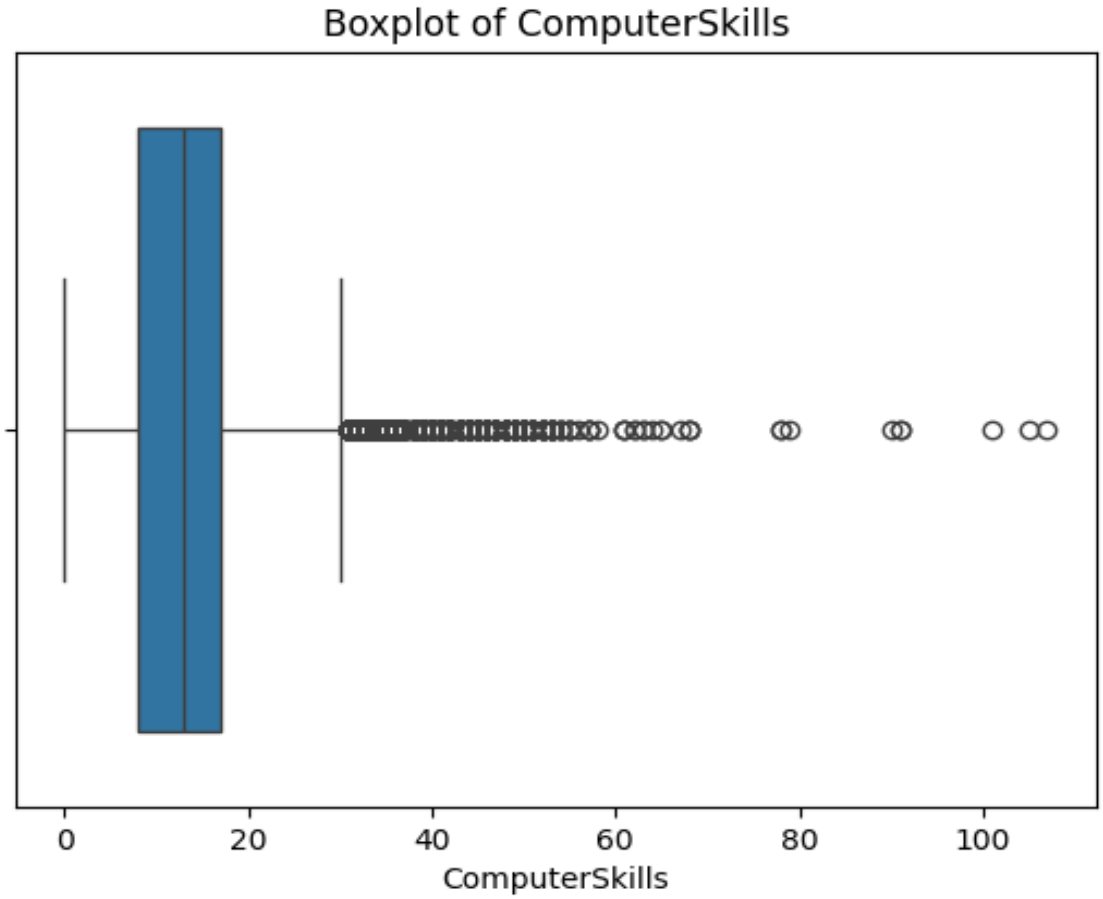
from Fig.9 Most candidates who are placed have experience working with technologies from all the major categories in the feature 'HaveWorkedWith'.

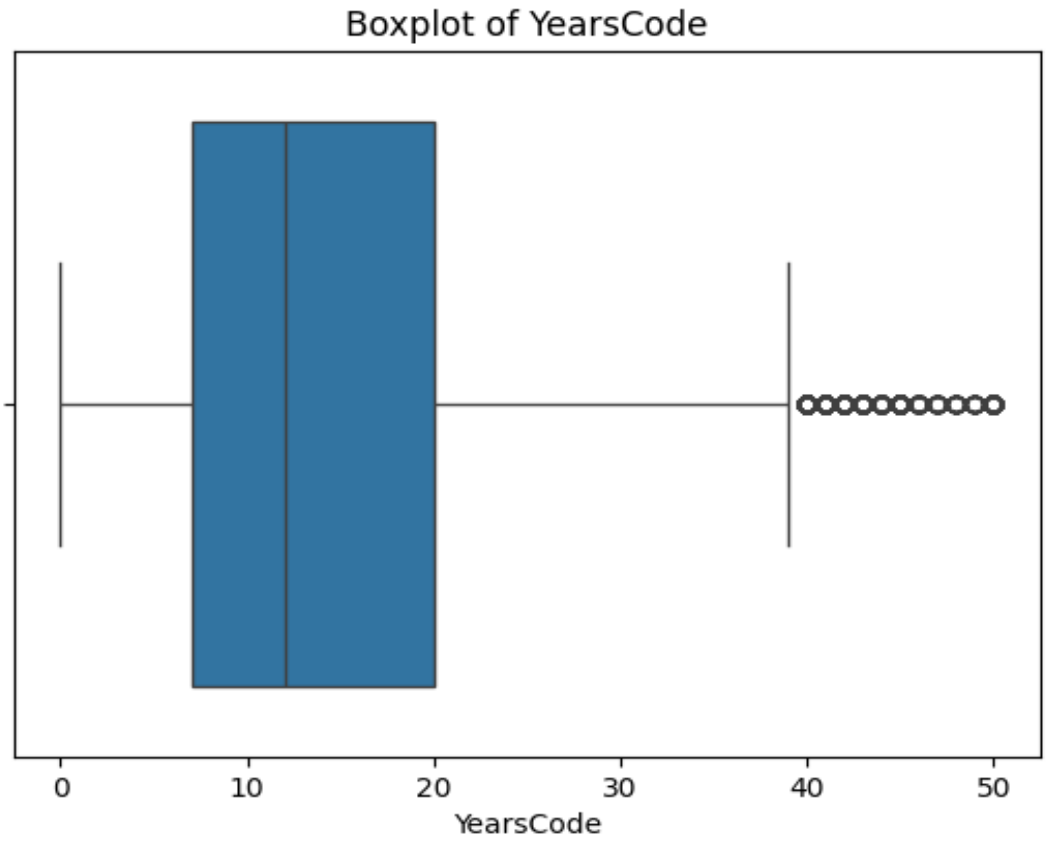
Handling Outliers:

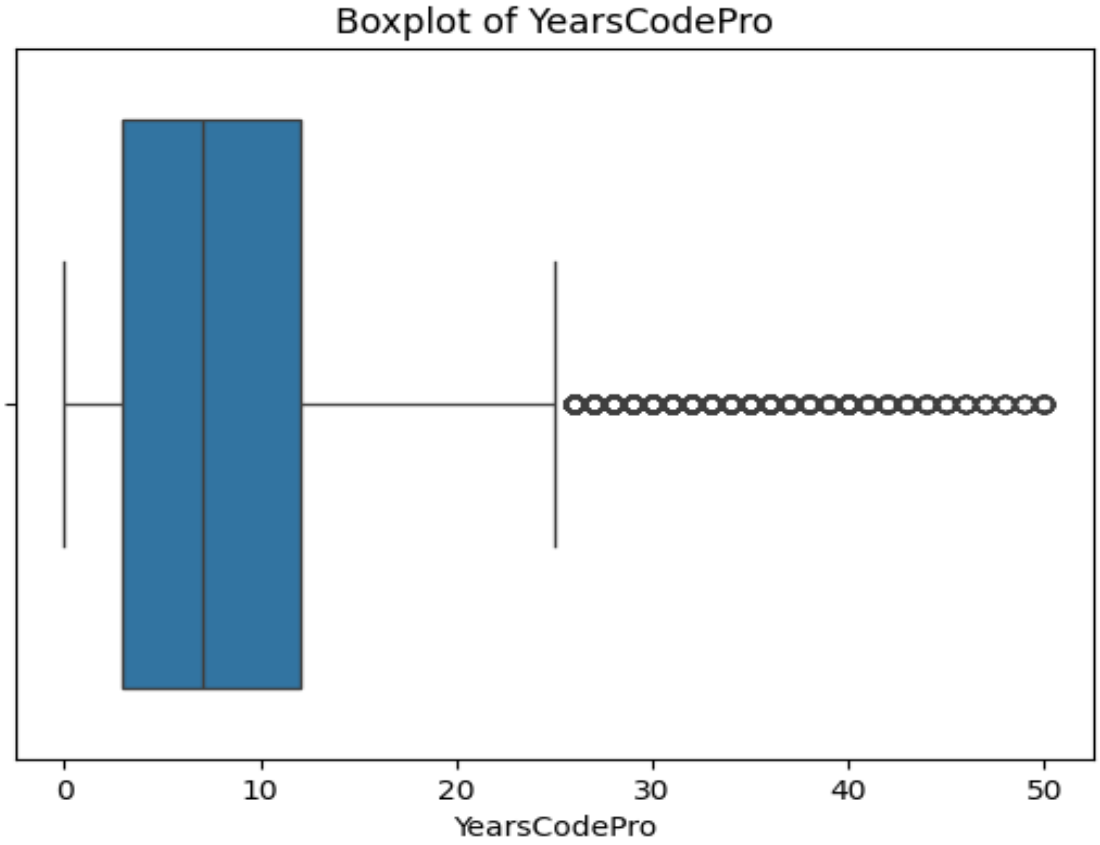
Box Plot:

Box Plot provides a quick visual summary of the variability of the values in the dataset. The box in a box plot represents the interquartile range (IQR), which measures the spread of the

middle 50% of the data. The bottom edge of the box corresponds to the first quartile (Q1), which is the 25th percentile, while the top edge represents the third quartile (Q3), the 75th percentile. The length of the box represents the range between Q1 and Q3. Inside the box, a horizontal line or marker represents the median or the second quartile(Q2). The whiskers extend from the box and represent the variability of the data outside the interquartile range. Points beyond the whiskers are considered potential outliers and are represented as individual data points.




Boxplot of YearsCode
0
10
20
30
40
50
YearsCode


Boxplot of YearsCodePro
0
10
20
30
40
50
YearsCodePro

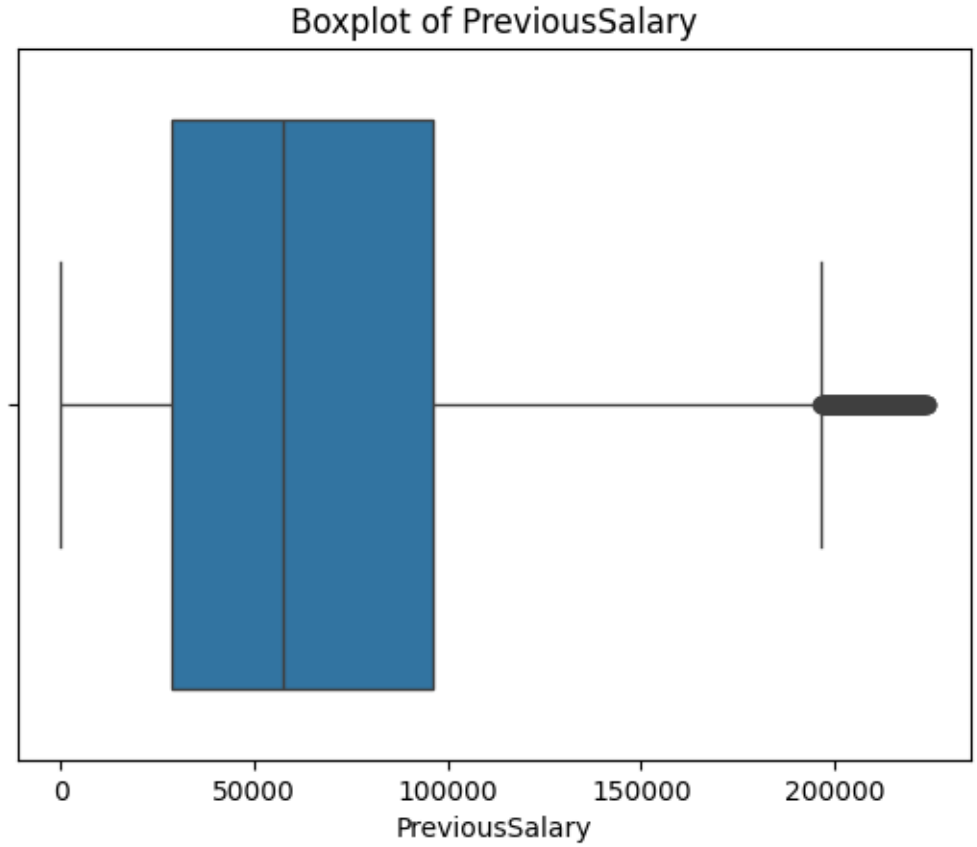


Fig.10

From the figure it is clear that the boxplot contains outliers. Capping method was used to handle outliers in the ComputerSkills, YearsCode, YearsCodePro and PreviousSalary features.

Encoding:

Label Encoding: ‘Gender’, ‘Accessibility’, ’MainBranch’, ’Age’, ‘MentalHealth’.

One-Hot Encoding: ‘HaveWorkedWith’

Ordinal Encoding: ‘EdLevel’

Frequency Encoding: ‘Country’

Heat Map:

The heat map is a graphical representation of the data where the values of a matrix are depicted using a color scale. It uses a grid of cells, each representing a data point, and assigns colors to those cells based on their values. Heat maps are commonly used to explore the correlation between the variables. Each cell in the matrix is filled with color representing the strength and direction of the correlation between the corresponding pair of variables.

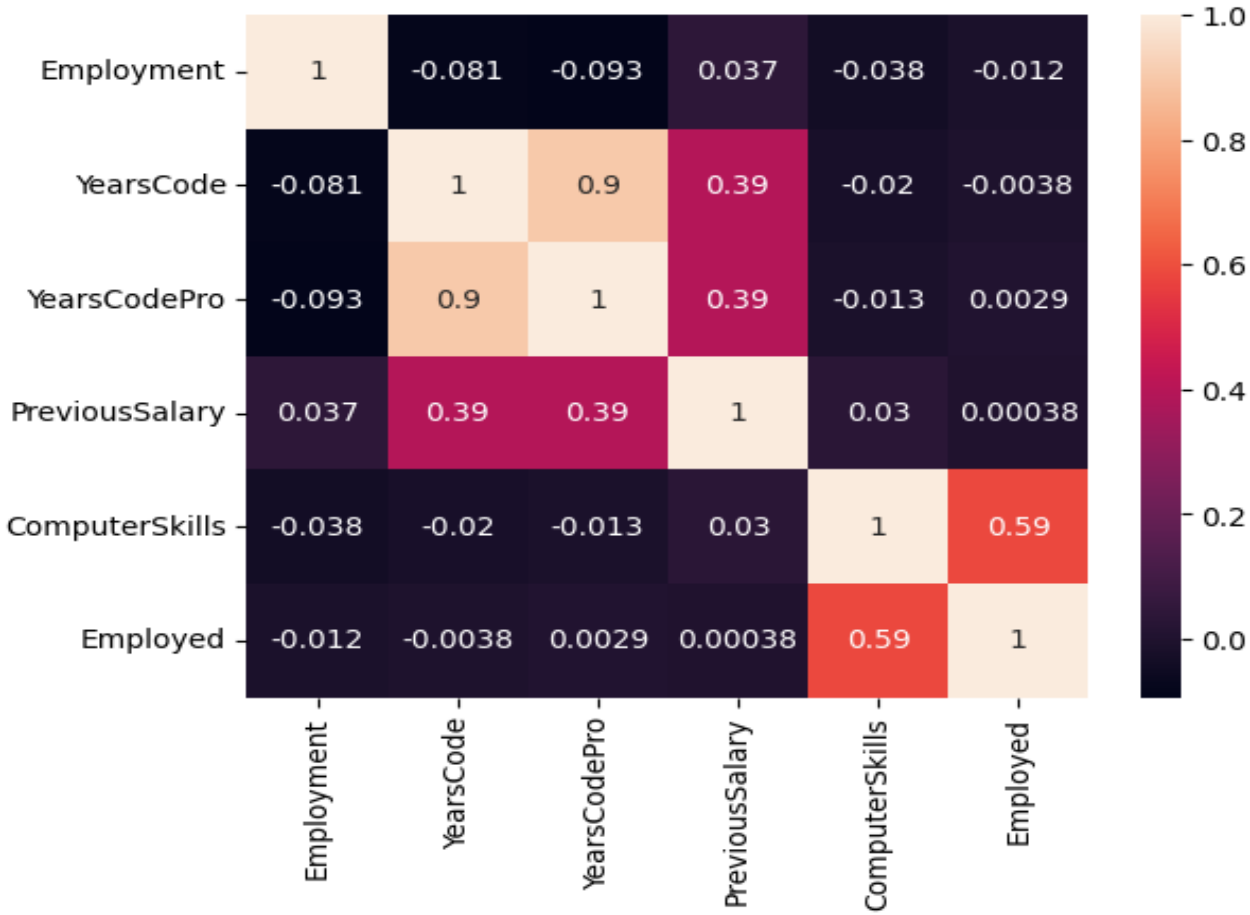


Fig.11

From figure 11 we would easily understand that the highest correlation is that of between **Employed** and **ComputerSkills.**

Splitting of Data:

Here, the data split in the ratio 80% for training and 20% for testing.

Let **'x'** represent the columns except the target column or independent variables used for prediction and **'y'** represent the target column or dependent variable.

4.3 EVALUATION OF THE MODELS

1) Decision Tree

Accuracy= 0.8026271013407745

Precision= 0.8060382008626001

Recall= 0.8314478200076268

F1_score= 0.8185458640971093

An accuracy of **0.8026271013407745** indicates that approximately 80.26% of predictions made by the decision tree model were correct. A precision of **0.8060382008626001** suggests that around 80.6% of the instances predicted as positive by the decision tree model were indeed positive. A recall of **0.8314478200076268** indicates that the model correctly identified approximately 83.14% of the positive instances. With an F1 score of approximately **0.818545**, the decision tree model demonstrates a relatively balanced performance between precision and recall in the **Employment** dataset.

Confusion Matrix

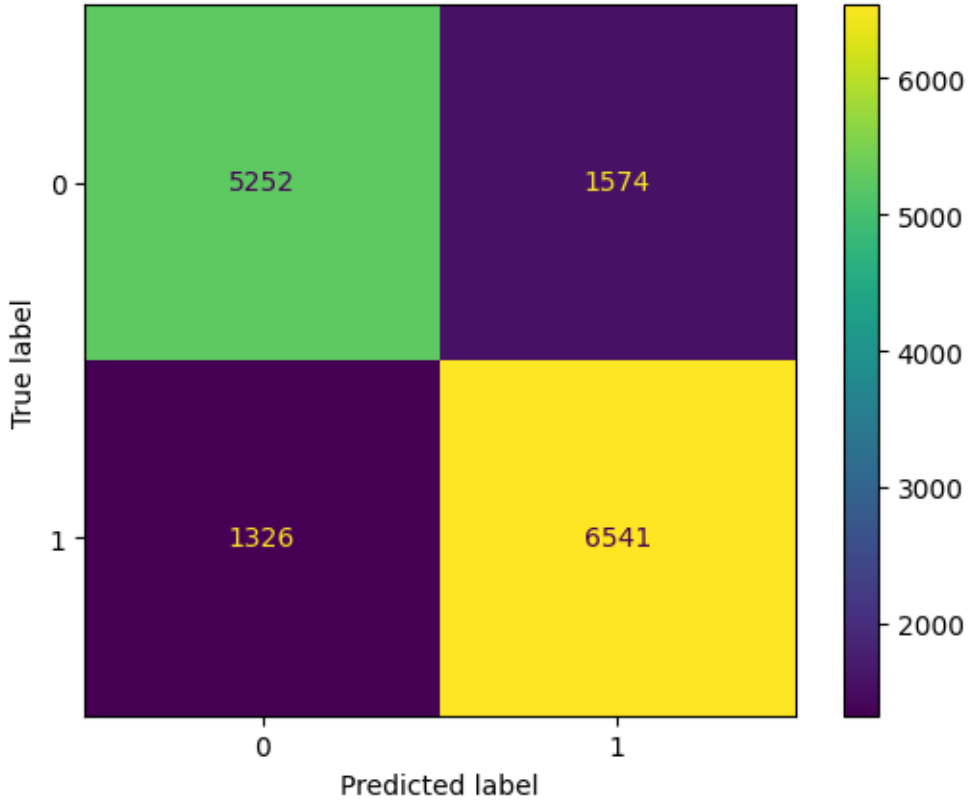


Fig.12

2) KNN

Accuracy= 0.7100660178316205

Precision= 0.8160154196600666

Recall= 0.591966442099911

F1_score= 0.6861647266833653

The KNN model correctly predicted the outcome in the dataset with an overall accuracy of approximately **71%**. The model achieved a precision of approximately **81.6%.** This means that roughly 81.6% of the instances predicted as positive by the model were indeed positive. Also the model has a recall rate of approximately **59.19%**. This suggests that the model correctly identified about **59.19%** of positive instances in the dataset. With an F1 score of approximately **0.6861**, suggests that the KNN model has achieved a relatively balanced performance between precision and recall in the dataset.

Confusion Matrix

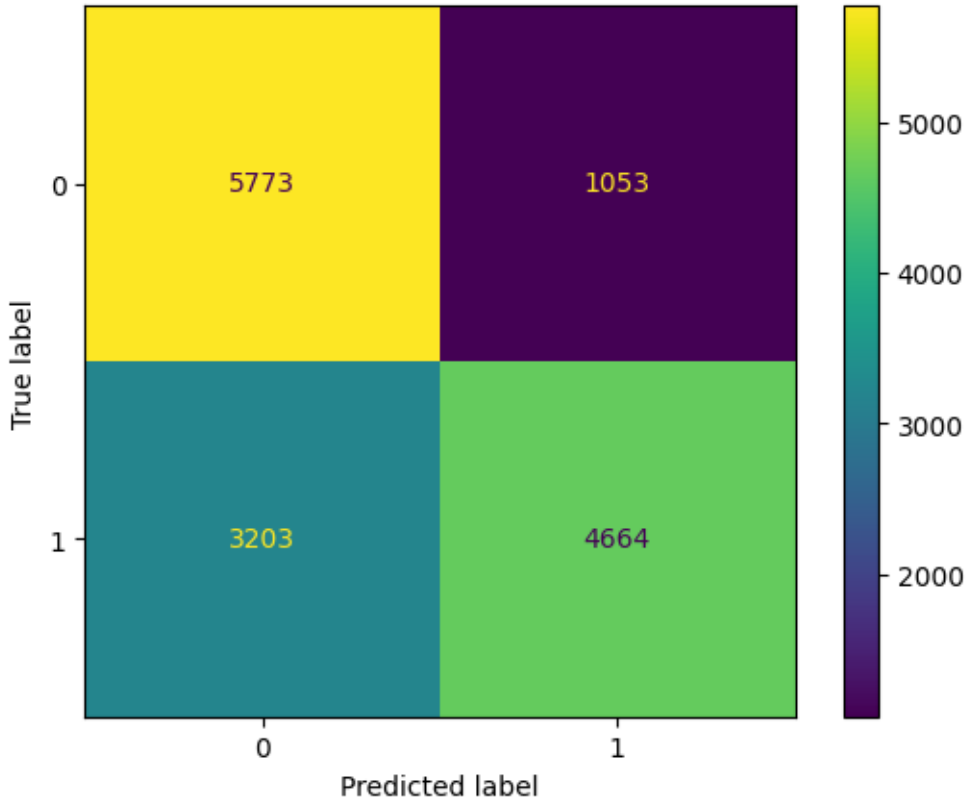


Fig.13

3) Random Forest Classifier

Accuracy= 0.80807187095896

Precision= 0.8059900569904208

Recall= 0.8449218253463836

F1_score= 0.8249968971081048

The Random Forest model correctly predicted the outcome in the dataset with an overall accuracy of approximately **80.80%**. The model achieved a precision of approximately **80.59%.** This means that roughly 80.59% of the instances predicted as positive by the model were indeed positive. Also the model has a recall rate of approximately **84.49%**. This suggests that the model correctly identified about **84.49%** of positive instances in the dataset. With an F1 score of approximately **0.82499**, suggests that the Random Forest model has achieved a relatively balanced performance between precision and recall in the dataset.

Confusion Matrix

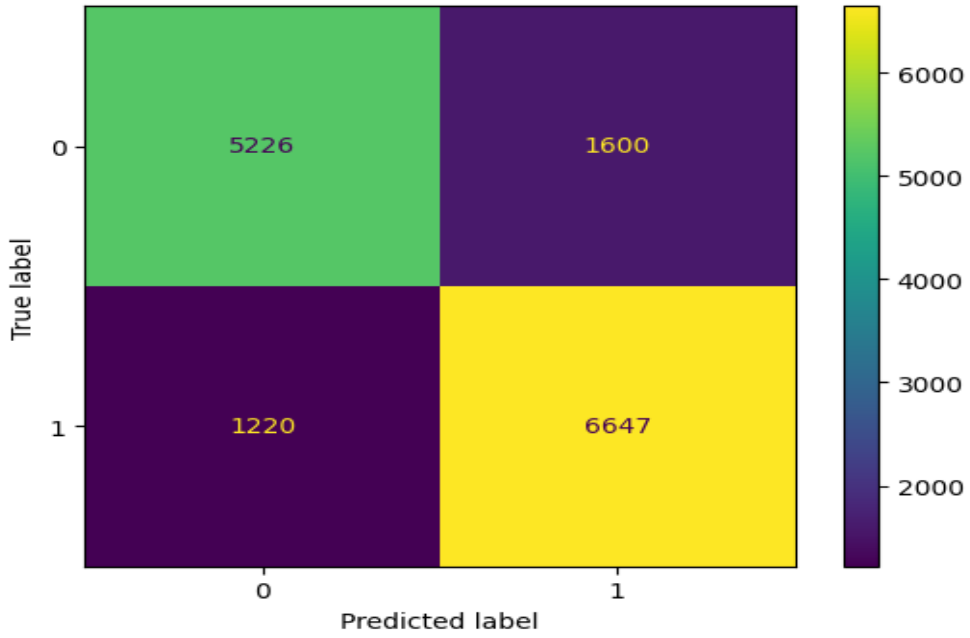


Fig.14

4) XGBoost

Accuracy= 0.8103178384264615

Precision= 0.807804168686379

Recall= 0.8473369772467269

F1_score= 0.8270984552391587

The XGBoost model correctly predicted the outcome in the dataset with an overall accuracy of approximately **81.03%**. The model achieved a precision of approximately **80.78%.** This means that roughly 80.78% of the instances predicted as positive by the model were indeed positive. Also the model has a recall rate of approximately **84.73%**. This suggests that the model correctly identified about **84.73%** of positive instances in the dataset. With an F1 score of approximately **0.82709**, suggests that the XGBoost model has achieved a relatively balanced performance between precision and recall in the dataset.

Confusion Matrix

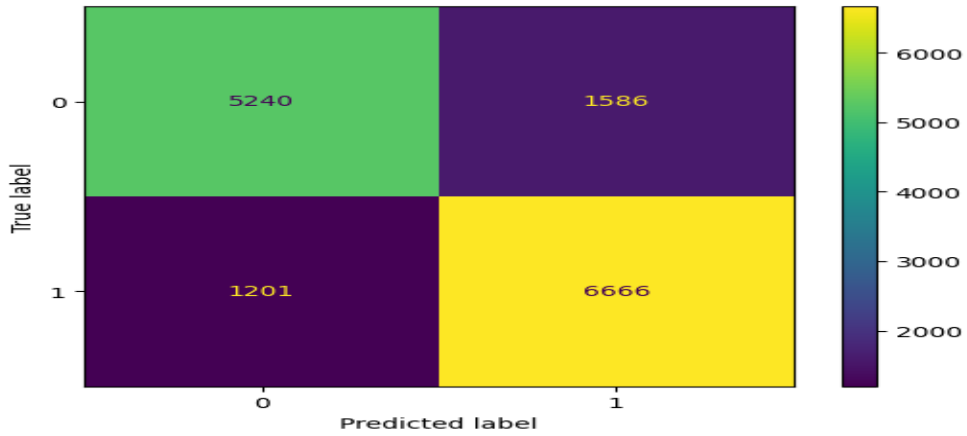


Fig.15

4.5 COMPARISON

To compare the models and choose the best one, we look at different evaluation metrics such as accuracy, precision, recall and F1 score. These metrics provide insights into different aspects of the model's performance.

Algorithm	Accuracy	Precision	Recall	F1 Score
DecisionTreeClassifier	80.2627	80.6038	83.1448	81.8546
KNeighborsClassifier	71.0338	81.5812	59.2856	68.669
RandomForestClassifier	80.8072	80.599	84.4922	82.4997
XGBClassifier	81.0318	80.7804	84.7337	82.7098

The table presented above depict the performance of different algorithms. Upon analysis, it is evident that the XGBoost algorithm exhibits the highest accuracy (81.03%) compared to other models.

4.6 FEATURE IMPORTANCE

The below helps quickly identify which features have the greatest impact on the model's prediction. Features with higher importance scores contribute more to the model's overall decision-making process.

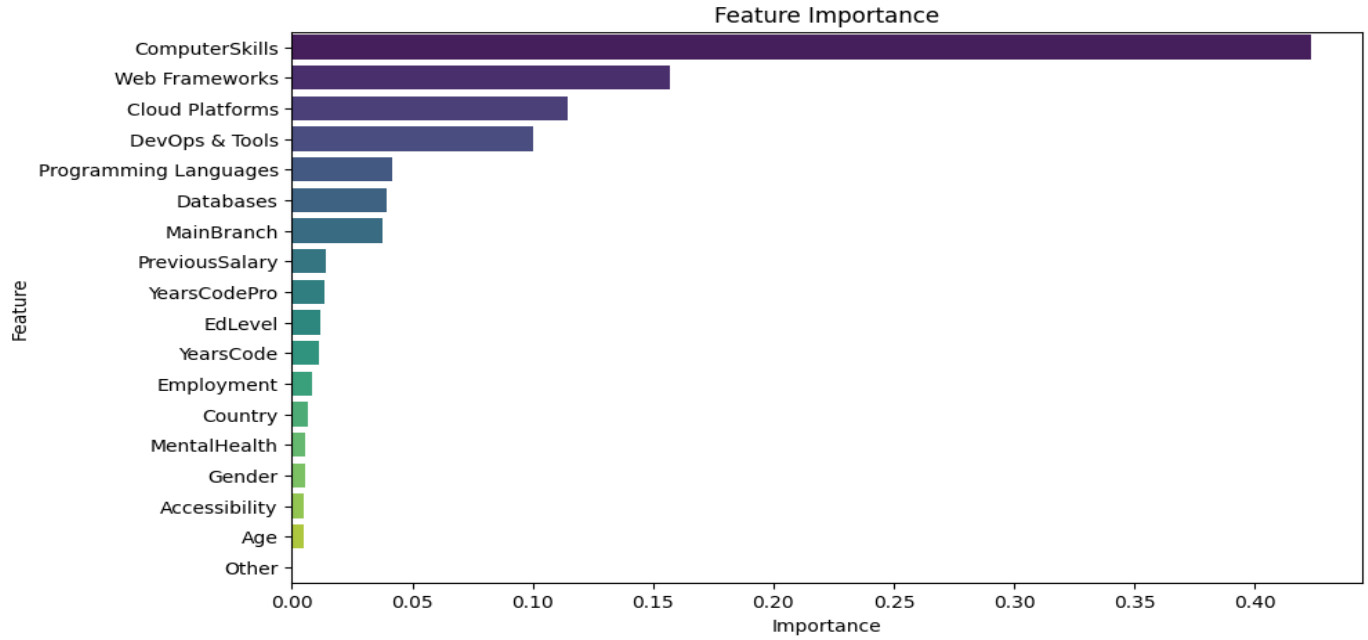


Fig.16

'ComputerSkills' is the most important feature that influences the predictions of the XGBoost Classifier.

CHAPTER 5
CONCLUSION

In conclusion, after analyzing the Employment dataset and evaluating different machine learning models, the XGBoost Classifier emerged as the most accurate model for predicting whether a candidate will be placed or not. The accuracy score of 81.03% indicates that the model successfully identifies the target variable in approximately 81.03% of the cases in the testing dataset. This high accuracy demonstrates the model's effectiveness in distinguishing between placed and not placed candidates based on relevant features such as education level, computer skills, prior work experience, and specific technologies the candidates have worked with.

Additionally, the XGBoost Classifier achieved a higher F1 score compared to other models, signifying a balanced trade-off between precision and recall. This suggests that the XGBoost model is not only capable of correctly identifying placed candidates but also effectively minimizing false negatives, which is particularly crucial in employment prediction where accurate classification can significantly impact recruitment strategies and decision-making processes.

In contrast, both the Decision Tree and Random Forest models achieved an accuracy of 80%, indicating that they are also viable alternatives for employment prediction. However, their slightly lower F1 scores compared to XGBoost suggest that these models may have a higher tendency to misclassify candidates in certain cases, particularly when dealing with imbalanced data or overlapping feature sets. Despite this, the Decision Tree model offers interpretability, allowing HR professionals to understand the key factors influencing employment outcomes more clearly.

Furthermore, data preprocessing played a pivotal role in model performance. Handling missing values using mode imputation, capping outliers to reduce their impact, and encoding categorical variables through techniques such as frequency encoding and ordinal encoding contributed to a more robust and reliable dataset. By effectively addressing data quality issues, the models were able to generate more accurate predictions and maintain consistency across different testing scenarios.

Beyond model performance, the findings of this study emphasize the significance of certain features in employment prediction. The analysis revealed that candidates with advanced computer skills, extensive coding experience, and exposure to multiple programming languages tend to have higher placement rates. Additionally, accessibility, education level, and previous salary emerged as influential factors, indicating that these attributes can significantly affect a candidate's employability.

Implementing such predictive models in real-world HR systems can provide organizations with valuable insights into candidate suitability, streamline the recruitment process, and reduce hiring costs. By identifying key skills and attributes that contribute to successful placement, HR departments can make data-driven decisions to target the right candidates, prioritize skill development, and align hiring strategies with organizational goals.

In conclusion, the project underscores the importance of evaluating multiple machine learning models to identify the most effective algorithm for employment prediction. While the XGBoost model demonstrated superior accuracy and balanced predictive capabilities, the Decision Tree and Random Forest models remain valuable due to their interpretability and generalization capabilities. Ultimately, this study highlights the potential of machine learning as a transformative tool in HR analytics, aiding organizations in making more informed and data-driven hiring decisions.

The potential applications of the developed model extend beyond candidate selection, as the insights derived from the analysis can be leveraged for training and development programs. For instance, by identifying skill gaps among candidates who were not successfully placed, organizations can design targeted training modules to enhance specific competencies. Additionally, integrating predictive analytics into HR systems can facilitate long-term workforce planning by anticipating emerging skill requirements and aligning recruitment strategies with future organizational needs.

Furthermore, the findings of this study underscore the importance of data-driven decision-making in HR processes. The ability to systematically assess candidate profiles based on objective criteria not only improves the quality of hiring but also minimizes bias in the selection process. As machine learning models continue to evolve, they offer the potential to enhance HR practices by providing deeper insights into employee performance, retention, and career progression, ultimately contributing to a more efficient and data-centric approach to human resource management.

REFERENCE

1. Bhardwaj A, & Pal S (2021),A Predictive Model for Employee Hiring in IT Sector Using Machine Learning Algorithms, *journal of Advanced Research in Computer Science and Software Engineering, 11(4), 22-28.*
2. Dangeti, P (2017),*Statistics for Machine Learning*, Packt Publishing.
3. Kapoor, K Dwivedi, Y. K., Piercy, N. C, & Williams, M. D. (2021), *Artificial Intelligence and Human Resource Management. Springer.*
4. Mitchell, T. M. (1997), *Machine Learning McGraw-Hill.*
5. https://www.geeksforeseeks.org/regression-classification-supervised-machine-learning
6. http://www.kaggle.com/datasets/ayushtankha/70k-job-applicants-data-human-resource

APPENDIX

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns


data=pd.read_csv('/content/stackoverflow_full.csv')


data.shape


data.head()


data.tail()


data.info()


data.describe()


data.isna().sum()


data.shape


#To find hidden missing values
data.isin(["?", "None", "NA", "", "null"]).sum()


# to find the percentage of missing values in HaveWorkedWith
data.isna().sum()/len(data)*100


data[data['HaveWorkedWith'].isna()].index


data['Age'].unique()


sns.countplot(x='Employed',data=data,palette=['red','green'])
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status',weight='bold',size=20)
plt.xticks(size=12,weight='bold',color='black')
plt.xlabel('Employed',size=15,weight='bold')
plt.ylabel('Count of Candidates',weight='bold')
plt.yticks(size=12,weight='bold',color='black')
plt.show()


sns.countplot(x='Employed', hue='Gender', data=data, palette=['orange', 'green','blue'])  # Specify 'hue' for Gender
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by Gender', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


sns.countplot(x='Employed', hue='Accessibility', data=data, palette=['orange', 'green'])
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by Accessibility', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


sns.countplot(x='Employed', hue='Age', data=data, palette=['orange','green'])  # Using a different palette for Age
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by Age', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
```

```
plt.yticks(size=12, weight='bold', color='black')
plt.show()


sns.countplot(x='Employed', hue='EdLevel', data=data, palette=['orange', 'green','blue','brown','red'])  # Using a different palette for
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by EdLevel', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


sns.countplot(x='Employed', hue='MentalHealth', data=data, palette=['orange','green'])
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by MentalHealth', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


sns.countplot(x='Employed', hue='MainBranch', data=data, palette=['orange','green'])
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by MainBranch', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


# Get the top 10 most frequent countries
top_countries = data['Country'].value_counts().index[:10]

# Filter the data to include only the top countries
filtered_data = data[data['Country'].isin(top_countries)]

sns.countplot(x='Employed', hue='Country', data=filtered_data, palette='viridis')
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status by Top 10 Countries', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1))


data['ComputerSkills'].nunique()


# Create the countplot
sns.countplot(x='Employed',hue='Employment',data=data, palette=['orange', 'green'])  # Specify 'x' for Employed
plt.grid(visible=True, linestyle='--')
plt.title('Placement Status', weight='bold', size=20)
plt.xticks(size=12, weight='bold', color='black')
plt.xlabel('Employed', size=15, weight='bold')
plt.ylabel('Count of Candidates', weight='bold')
plt.yticks(size=12, weight='bold', color='black')
plt.show()


# Filter data for employed individuals (Employed == 1)
employed_data = data[data['Employed'] == 1]

# Get top 10 ComputerSkills and their counts for employed individuals
top_10_skills_employed = employed_data['ComputerSkills'].value_counts().head(10)

# Create a new DataFrame with top 10 and "Others" for employed individuals
skills_counts_employed = pd.DataFrame({'ComputerSkills': top_10_skills_employed.index, 'Count': top_10_skills_employed.values})
other_count_employed = employed_data['ComputerSkills'].value_counts().sum() - top_10_skills_employed.sum()
skills_counts_employed = pd.concat([skills_counts_employed, pd.DataFrame({'ComputerSkills': ['Others'], 'Count': [other_count_employed]

# Create pie chart for employed individuals
plt.figure(figsize=(8, 8))
plt.pie(skills_counts_employed['Count'], labels=skills_counts_employed['ComputerSkills'], autopct='%1.1f%%', startangle=90)
plt.title('ComputerSkills for Placed Individuals (Top 10 + Others)')
plt.axis('equal')
plt.show()


# Filter data for employed individuals (Employed == 1)
employed_data = data[data['Employed'] == 1]
```

```
# Get top 10 YearsCode and their counts for employed individuals
top_10_yearscode_employed = employed_data['YearsCode'].value_counts().head(10)

# Create a new DataFrame with top 10 and "Others" for employed individuals
yearscode_counts_employed = pd.DataFrame({'YearsCode': top_10_yearscode_employed.index, 'Count': top_10_yearscode_employed.values})
other_count_employed = employed_data['YearsCode'].value_counts().sum() - top_10_yearscode_employed.sum()
yearscode_counts_employed = pd.concat([yearscode_counts_employed, pd.DataFrame({'YearsCode': ['Others'], 'Count': [other_count_employed]

# Create pie chart for employed individuals
plt.figure(figsize=(8, 8))
plt.pie(yearscode_counts_employed['Count'], labels=yearscode_counts_employed['YearsCode'], autopct='%1.1f%%', startangle=90)
plt.title('YearsCode for Placed Individuals (Top 10 + Others)')
plt.axis('equal')
plt.show()


# Filter data for employed individuals (Employed == 1)
employed_data = data[data['Employed'] == 1]

# Get top 10 YearsCodePro and their counts for employed individuals
top_10_yearscodepro_employed = employed_data['YearsCodePro'].value_counts().head(10)

# Create a new DataFrame with top 10 and "Others" for employed individuals
yearscodepro_counts_employed = pd.DataFrame({'YearsCodePro': top_10_yearscodepro_employed.index, 'Count': top_10_yearscodepro_employed.v
other_count_employed = employed_data['YearsCodePro'].value_counts().sum() - top_10_yearscodepro_employed.sum()
yearscodepro_counts_employed = pd.concat([yearscodepro_counts_employed, pd.DataFrame({'YearsCodePro': ['Others'], 'Count': [other_count_

# Create pie chart for employed individuals
plt.figure(figsize=(8, 8))
plt.pie(yearscodepro_counts_employed['Count'], labels=yearscodepro_counts_employed['YearsCodePro'], autopct='%1.1f%%', startangle=90)
plt.title('YearsCodePro for Placed Individuals (Top 10 + Others)')
plt.axis('equal')
plt.show()


#to extract all unique technologies
all_technologies = []
for tech_list in data['HaveWorkedWith'].astype(str).tolist():# This converts each value in the 'HaveWorkedWith' column to a string,this
    all_technologies.extend(tech_list.split(";"))#This part splits the tech_list string using the semicolon (;)
unique_technologies = set(all_technologies)
print(unique_technologies)
```

Data Preprocessing

Handling Missing Values

```
# Define technology categories
technology_categories = {
    "Programming Languages": {"Python", "Java", "C++", "JavaScript", "C#", "Go", "Rust", "Swift", "Kotlin"},
    "Databases": {"MySQL", "PostgreSQL", "MongoDB", "SQLite", "Oracle", "SQL Server"},
    "Web Frameworks": {"Django", "Flask", "React", "Angular", "Vue.js", "Node.js", "Spring Boot"},
    "Cloud Platforms": {"AWS", "Azure", "Google Cloud"},
    "DevOps & Tools": {"Docker", "Kubernetes", "Terraform", "Jenkins"}
}


# Function to categorize technologies, marking unknown ones as 'Other'
def categorize_technologies(tech_list):
    categories = set()
    unknown_technologies = set()  # Track unknown technologies

    for tech in tech_list:
        found = False
        for category, tech_set in technology_categories.items():
            if tech in tech_set:
                categories.add(category)
                found = True
                break  # Stop searching once a category is found
        if not found:
            unknown_technologies.add(tech)  # Collect unknown technologies

    # If unknown technologies exist, mark as 'Other'
    if unknown_technologies:
        categories.add("Other")

    return ', '.join(categories) if categories else "Other"

# Apply the function to categorize each row
if 'HaveWorkedWith' in data.columns:
```

```
    data['HaveWorkedWith'] = data['HaveWorkedWith'].dropna().apply(lambda x: categorize_technologies(x.split(';')))

    # Find the most common category
    most_common_category = data['HaveWorkedWith'].mode()[0]
    # Fill missing values with the most common category
    data['HaveWorkedWith'].fillna(most_common_category, inplace=True)
```

```
# Set the figure size
plt.figure(figsize=(50,50))  # Adjust width and height as needed

# Create the count plot
sns.countplot(y='HaveWorkedWith', hue='Employed', data=data, order=data['HaveWorkedWith'].value_counts().index, palette='viridis')
plt.grid(visible=True, linestyle='--', axis='x')  # Grid on x-axis
plt.title('Placement Status by Technology Category', weight='bold', size=20)
plt.yticks(size=75, weight='bold', color='black')  # y-axis ticks
plt.ylabel('Technology Category', size=75, weight='bold')  # y-axis label
plt.xlabel('Count of Candidates', weight='bold')  # x-axis label
plt.xticks(size=75, weight='bold', color='black')  # x-axis ticks
plt.legend(title='Employed')  # Add legend for 'Employed'
plt.show()
```

```
data.isna().sum()
```

```
data=data.drop(['Unnamed: 0'],axis=1)
```

```
data['Employed'].unique()
```

```
data['Gender'].unique()
```

```
data['EdLevel'].unique()
```

```
sns.boxplot(data)
```

```
plt.hist(data['YearsCode'])
plt.show()
```

```
plt.hist(data['YearsCodePro'])
plt.show()
```

```
plt.hist(data['PreviousSalary'])
plt.show()
```

```
sns.heatmap(data.corr(numeric_only=True),annot=True)
plt.show()
```

```
plt.hist(data['ComputerSkills'])
plt.show()
```

```
def identify_outliers_boxplot(data, column):
  sns.boxplot(x=data[column])
  plt.title(f'Boxplot of {column}')
  plt.show()
```

```
identify_outliers_boxplot(data, 'ComputerSkills')
```

```
identify_outliers_boxplot(data, 'YearsCode')
```

```
identify_outliers_boxplot(data, 'YearsCodePro')
```

```
identify_outliers_boxplot(data,'PreviousSalary')
```

Handling Outliers

```
# Apply capping
from scipy.stats.mstats import winsorize
# Apply winsorize to each column individually
```

```
for column in ['ComputerSkills', 'YearsCode', 'YearsCodePro','PreviousSalary']:  # Corrected 'YeatsCodePro' to 'YearsCodePro'
    data[column] = winsorize(data[column], limits=[0, 0.05])  # Using limits for clarity


sns.boxplot(data)


data['Accessibility'].unique()


data['MainBranch'].unique()
```

Encoding

```
#Label encoding
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
# Apply label encoding to the specified columns
for column in ['Gender', 'Accessibility', 'MainBranch', 'Age', 'MentalHealth']:
    data[column] = le.fit_transform(data[column])


data['EdLevel'].unique()


# ordinalencoding
from sklearn.preprocessing import OrdinalEncoder
edlevels=['NoHigherEd','Undergraduate','Master','PhD','Other']
oe=OrdinalEncoder(categories=[edlevels],dtype=int)
D=oe.fit_transform(data[['EdLevel']])
data['EdLevel']=D


# Apply one-hot encoding
data_encoded = data['HaveWorkedWith'].str.get_dummies(sep=', ')

# Merge with original dataset and drop 'HaveWorkedWith'
data = pd.concat([data, data_encoded], axis=1).drop(columns=['HaveWorkedWith'])


data.head()


data['Country'].nunique()


#This code is used to calculate the total number of employed individuals for each country
data.groupby('Country')['Employed'].sum()


data['Country'].value_counts()


# Frequency Encoding
fe=data.groupby('Country').size()/len(data)#calculates the frequency of each unique value in the 'Country' column
data['Country']=data['Country'].map(fe)#remap places the original 'Country' values with their corresponding frequencies.
```

Scaling

```
# Scaling (mini max bz right skewed)
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
data[['YearsCode','YearsCodePro','ComputerSkills','PreviousSalary']]=scaler.fit_transform(data[['YearsCode','YearsCodePro','ComputerSki


#to check balance
data['Employed'].value_counts()/len(data)*100


majority_class = data['Employed'].value_counts().max()
minority_class = data['Employed'].value_counts().min()
imbalance_ratio = majority_class / minority_class
print(f"Imbalance Ratio: {imbalance_ratio}")#1.0 - 1.5  Balanced


x=data.drop(['Employed'],axis=1)
y=data['Employed']


from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt=DecisionTreeClassifier(criterion = "entropy",max_depth=10,min_samples_split=20)


model1=dt.fit(x_train,y_train)


y_pred_dec=model1.predict(x_test)


from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('Accuracy=', accuracy_score(y_test, y_pred_dec))
print('Precision=', precision_score(y_test, y_pred_dec))
print('Recall=', recall_score(y_test, y_pred_dec))
print('F1_score=', f1_score(y_test, y_pred_dec))


acc_dec=accuracy_score(y_test, y_pred_dec)
pre_dec=precision_score(y_test, y_pred_dec)
rec_dec=recall_score(y_test, y_pred_dec)
f1_dec=f1_score(y_test, y_pred_dec)


from sklearn.metrics import ConfusionMatrixDisplay
disp=ConfusionMatrixDisplay(confusion_matrix(y_test,y_pred_dec))
disp.plot()
plt.show()
```

KNN

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
neighbors=np.arange(15,20)
metric_k=[]
for k in neighbors:
  classifier=KNeighborsClassifier(n_neighbors=k)
  classifier.fit(x_train,y_train)
  y_pred=classifier.predict(x_test)
  accuracy=accuracy_score(y_test,y_pred)
  metric_k.append(accuracy)


classifier=KNeighborsClassifier(n_neighbors=2,metric='minkowski',p=2)
classifier.fit(x_train,y_train)
y_pred_knn=classifier.predict(x_test)


from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('Accuracy=', accuracy_score(y_test, y_pred_knn))
print('Precision=', precision_score(y_test, y_pred_knn))
print('Recall=', recall_score(y_test, y_pred_knn))
print('F1_score=', f1_score(y_test, y_pred_knn))


acc_knn=accuracy_score(y_test, y_pred_knn)
pre_knn=precision_score(y_test, y_pred_knn)
rec_knn=recall_score(y_test, y_pred_knn)
f1_knn=f1_score(y_test, y_pred_knn)


from sklearn.metrics import ConfusionMatrixDisplay
disp=ConfusionMatrixDisplay(confusion_matrix(y_test,y_pred_knn))
disp.plot()
plt.show()
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='entropy',random_state=42,n_estimators=200,min_samples_split=5, max_depth=10)
model=rf.fit(x_train,y_train)
y_pred_ran=rf.predict(x_test)



from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('Accuracy=', accuracy_score(y_test, y_pred_ran))
print('Precision=', precision_score(y_test, y_pred_ran))
print('Recall=', recall_score(y_test, y_pred_ran))
print('F1_score=', f1_score(y_test, y_pred_ran))
```

```
acc_ran=accuracy_score(y_test, y_pred_ran)
pre_ran=precision_score(y_test, y_pred_ran)
rec_ran=recall_score(y_test, y_pred_ran)
f1_ran=f1_score(y_test, y_pred_ran)


from sklearn.metrics import ConfusionMatrixDisplay
disp=ConfusionMatrixDisplay(confusion_matrix(y_test,y_pred_ran))
disp.plot()
plt.show()
```

XGBoost

```
from xgboost import  XGBClassifier
from sklearn.metrics import  accuracy_score


model3 = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, reg_alpha=0.1, random_state=42)


model3.fit(x_train, y_train)
y_pred_xgb = model3.predict(x_test)


from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('Accuracy=', accuracy_score(y_test, y_pred_xgb))
print('Precision=', precision_score(y_test, y_pred_xgb))
print('Recall=', recall_score(y_test, y_pred_xgb))
print('F1_score=', f1_score(y_test, y_pred_xgb))


acc_xgb=accuracy_score(y_test, y_pred_xgb)
pre_xgb=precision_score(y_test, y_pred_xgb)
rec_xgb=recall_score(y_test, y_pred_xgb)
f1_xgb=f1_score(y_test, y_pred_xgb)


from sklearn.metrics import ConfusionMatrixDisplay
disp=ConfusionMatrixDisplay(confusion_matrix(y_test,y_pred_xgb))
disp.plot()
plt.show()
```

Comparing

```
alg = ['DecisionTreeClassifier', 'KNeighborsClassifier', 'RandomForestClassifier', 'XGBClassifier'] # Removed 'ArtificialNeuralNetwork'
accuracy_scores=[acc_dec,acc_knn,acc_ran,acc_xgb]
precision_scores=[pre_dec,pre_knn,pre_ran,pre_xgb]
recall_scores=[rec_dec,rec_knn,rec_ran,rec_xgb]
f1_scores=[f1_dec,f1_knn,f1_ran,f1_xgb]
# create a dictionary to store the metrics
metrics = {
    'Algorithm': alg,
    'Accuracy': accuracy_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1 Score': f1_scores
}
# Create a DataFrame
df = pd.DataFrame(metrics)
# Multiply the scores by 100 to represent as percentage in the final output
df[['Accuracy', 'Precision', 'Recall', 'F1 Score']] = df[['Accuracy', 'Precision', 'Recall', 'F1 Score']] * 100
# print DataFrame
from tabulate import tabulate
table=tabulate(df, headers='keys', showindex=False)
print(table)


# Feature importance
featre_rank=pd.DataFrame({'Feature':x.columns,'Importance':model3.feature_importances_})
featre_rank2=featre_rank.sort_values('Importance',ascending=False)
plt.figure(figsize=(10,6))
sns.barplot(x='Importance',y='Feature',orient='h',data=featre_rank2,palette='viridis')
plt.title('Feature Importance')
plt.show()
```