**Introduction & Dataset Acquisition:**

our work on building a machine learning algorithm to predict employee attrition and departure timeframe.

We started by collecting relevant data from IBM Human Resources records. This dataset contains various attributes of employees, such as age, job satisfaction, salary, and performance ratings, as well as a lot of other information.

Our goal is to assist decision-making in the HR department by providing insights into when to initiate the hiring process and strategies to retain valuable employees.

**Data Preprocessing & Exploration**

After acquiring the dataset, we performed several preprocessing steps, including handling missing values, encoding categorical variables, and scaling numerical features. Additionally, we mapped the targeted variables, to binary values 0,1 to facilitate model training.

Before diving into our model, let's take a look at the features we selected for our analysis.

. We conducted visualizations using Tableau to explore various employee attributes and their correlations with attrition and departure timeframe.

. All features selected to be applied to the algorithm had a high impact on our target value. And we confirmed this by testing multiple features collections until finding the best accuracy.

. These features were chosen based on their potential impact on employee attrition, as identified through our exploratory analysis.

**Using python, we created a code to take any random 100 rows from our original dataset and drop the attrition column so we can use this dataset for testing and applying our algorithm and so we can check the accuracy of the outcome.**

**Departure Timeframe Prediction Initial Approach**

We started by building a neural network model for predicting the timeframe until employee departure.

This model utilized features such as age, job satisfaction, and total working years to forecast the duration an employee stays in the company.

**Departure Timeframe Prediction Revised Approach**

Due to the complexity and variability of the data, we adopted an ensemble learning technique.

This involved training multiple neural network models with dropout regularization and averaging their predictions for improved accuracy and robustness.

**Key Changes**

We summarized the changes made in transitioning from a neural network to a random forest model:

. Data balancing with RandomOverSampler.

. Feature engineering by changing and adding relevant features.

. Changed Optimizer: Replaced the fixed learning rate with the Adam optimizer and set the learning rate to 0.001.

. Refactored Code into a Class: Created a custom class KerasRegressorWrapper to wrap the Keras model. This class implements the BaseEstimator and RegressorMixin from scikit-learn, allowing it to be used in scikit-learn pipelines and grid searches.

. Hyperparameter optimization using grid search: Utilized scikit-learn's GridSearchCV to search for the best combination of batch size and epochs for the neural network.

. Evaluation Metrics: Evaluated the best model using mean squared error, mean absolute error, root mean squared error, and R-squared.

(The goal was to improve the model's performance metrics, such as accuracy, precision, recall, or F1-score, by finding the optimal values for hyperparameters like learning rate, regularization strength, number of hidden layers, and so on)

Data Reading and Preprocessing:

* We read the dataset.
* mapped to numeric labels (0 for 'No', 1 for 'Yes').
* select relevant features for the timeframe prediction model.

Data Splitting:

* dataset is split into training and testing sets using the (train\_test\_split) function from scikit-learn.
* The split is done with a test size of 20% and a random state for reproducibility.

Data Scaling:

* Standard scaling is applied to the training and testing features using StandardScaler.

Neural Network Model Definition:

* We define a neural network model architecture using Karas' Sequential API.
* The model consists of multiple dense layers with ReLU activation functions, followed by dropout layers to prevent overfitting.
* The output layer has a single neuron with a linear activation function, suitable for regression tasks.

Model Training:

* Multiple neural network models are trained to capture different initializations and reduce variance.
* Each model is trained for 150 epochs with a batch size of 64.
* The mean squared error loss function is used for training, along with mean absolute error, mean squared error, and accuracy as metrics.
* Training is performed on the scaled training data, with a validation split of 20% to monitor model performance during training.

Model Evaluation and Storage:

* The trained models are stored in a list for further analysis and ensemble predictions.
* Located the best evaluated model and saved for future use.

**We implemented both approaches using Python and relevant libraries such as pandas, scikit-learn, and TensorFlow.**

**Conclusion**

In conclusion, our efforts aimed to provide HR departments with predictive analytics tools to mitigate employee attrition and optimize workforce management.

By leveraging machine learning algorithms, we can empower organizations to make data-driven decisions and foster a positive work environment.

**Questions and Discussion**

Thank you for your attention. We're now open to any questions or discussions regarding our presentation and work on employee attrition prediction.