

EMPLOYEE ATTRITION PREDICTION PROJECT

Team Name : Model Masters

Team Members

ARUN KARTHIK S 22CSR020

CHANDRU A 22CSR033

ELANKUMARAN R 22CSR053

Introduction

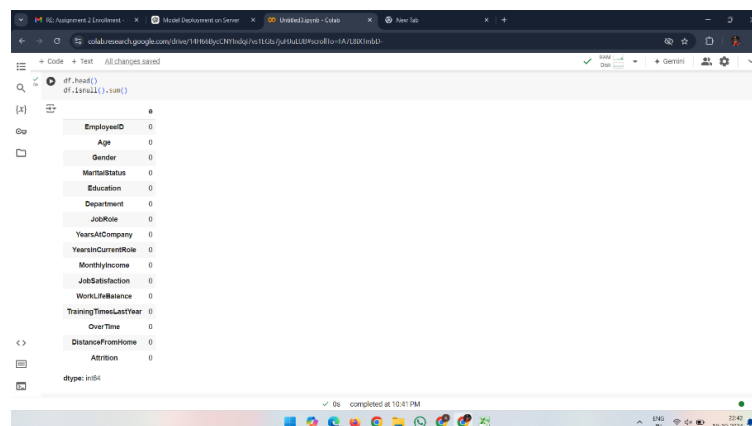
Employee attrition presents challenges for organizations and operational costs. This project utilizes machine learning to predict which employees may be at risk of leaving, allowing the organization to proactively address potential attrition. This predictive insight aims to support retention strategies that improve workforce stability.

Methodology

To predict employee attrition, we followed a structured approach, including data preprocessing, feature selection, model training, and evaluation.

1. Data Collection and Preprocessing

- The dataset provided a balanced view of attrition, with 2512 instances indicating employees left and 2488 instances indicating retention.
- Categorical features were converted to numeric form using label encoding. Additionally, key features were normalized to maintain consistency across model inputs.



```
df.head()
df.info()
```

EmployeeID	0
Age	0
Gender	0
MaritalStatus	0
Education	0
Department	0
JobRole	0
YearsAtCompany	0
YearsInCurrentRole	0
MonthlyIncome	0
JobSatisfaction	0
WorkLifeBalance	0
TrainingTimesLastYear	0
OverTime	0
DistanceFromHome	0
Attrition	0

dtype: int64

Checking missing values

```
numerical_columns = ['Age', 'YearsAtCompany', 'MonthlyIncome', 'DistanceFromHome']

def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] < lower_bound) | (df[column] > upper_bound)]

for col in numerical_columns:
    outliers = detect_outliers_iqr(df, col)
    print(outliers)

# Empty DataFrame
# Columns: [EmployeeID, Age, Gender, MaritalStatus, Education, Department, JobRole, YearsAtCompany, YearsInCurrentRole, MonthlyIncome, JobSatisfaction, WorkLifeBalance, TrainingTimesLastYear]
# Index: []
# Empty DataFrame
# Columns: [EmployeeID, Age, Gender, MaritalStatus, Education, Department, JobRole, YearsAtCompany, YearsInCurrentRole, MonthlyIncome, JobSatisfaction, WorkLifeBalance, TrainingTimesLastYear]
# Index: []
# Empty DataFrame
# Columns: [EmployeeID, Age, Gender, MaritalStatus, Education, Department, JobRole, YearsAtCompany, YearsInCurrentRole, MonthlyIncome, JobSatisfaction, WorkLifeBalance, TrainingTimesLastYear]
# Index: []

[10]: from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()
```

Checking outliers

```
[10]: from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()

# List of categorical columns to be label encoded
categorical_columns = ['Gender', 'MaritalStatus', 'Education', 'Department', 'JobRole', 'Overtime', 'Attrition']

# Apply label encoding
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])

# Display the encoded dataframe
df.head()
```

EmployeeID	Age	Gender	MaritalStatus	Education	Department	JobRole	YearsAtCompany	YearsInCurrentRole	MonthlyIncome	JobSatisfaction	WorkLifeBalance	TrainingTimesLastYear	Ov
0	1	54	1	2	3	4	0	17	12	19618.16	2	3	3
1	2	47	1	1	2	0	4	32	7	7858.48	4	2	8
2	3	41	0	1	2	4	2	37	7	10839.85	3	2	2
3	4	28	1	0	1	2	2	3	18	14488.44	5	2	4
4	5	37	0	0	1	2	1	25	12	9591.07	3	1	7

```
[11]: from sklearn.preprocessing import StandardScaler

numerical_columns = ['Age', 'YearsAtCompany', 'YearsInCurrentRole', 'MonthlyIncome', 'JobSatisfaction', 'WorkLifeBalance', 'TrainingTimesLastYear', 'DistanceFromHome']
```

Converting categorical data into values

```
[11]: from sklearn.preprocessing import StandardScaler

numerical_columns = ['Age', 'YearsAtCompany', 'YearsInCurrentRole', 'MonthlyIncome', 'JobSatisfaction', 'WorkLifeBalance', 'TrainingTimesLastYear', 'DistanceFromHome']

scaler = StandardScaler()

df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

df.head()
```

EmployeeID	Age	Gender	MaritalStatus	Education	Department	JobRole	YearsAtCompany	YearsInCurrentRole	MonthlyIncome	JobSatisfaction	WorkLifeBalance	TrainingTimesLastYear	DistanceFromHome
0	1	0.886935	1	2	3	4	0	-0.270644	0.375351	1.645854	-0.721711	-0.022387	-0.52705
1	2	0.319343	1	1	2	0	4	1.046315	-0.534723	-0.577295	0.692022	-0.731171	1.67076
2	3	-0.167164	0	1	2	4	2	1.485301	-0.534723	-0.027906	-0.014844	-0.731171	-0.87667
3	4	-1.140178	1	0	1	2	2	-1.490805	1.467439	0.667770	1.398889	-0.731171	-0.17746
4	5	-0.491502	0	0	1	2	1	0.431734	0.375351	-0.260111	-0.014844	-1.439945	0.87146

```
[12]: import seaborn as sns
import matplotlib.pyplot as plt

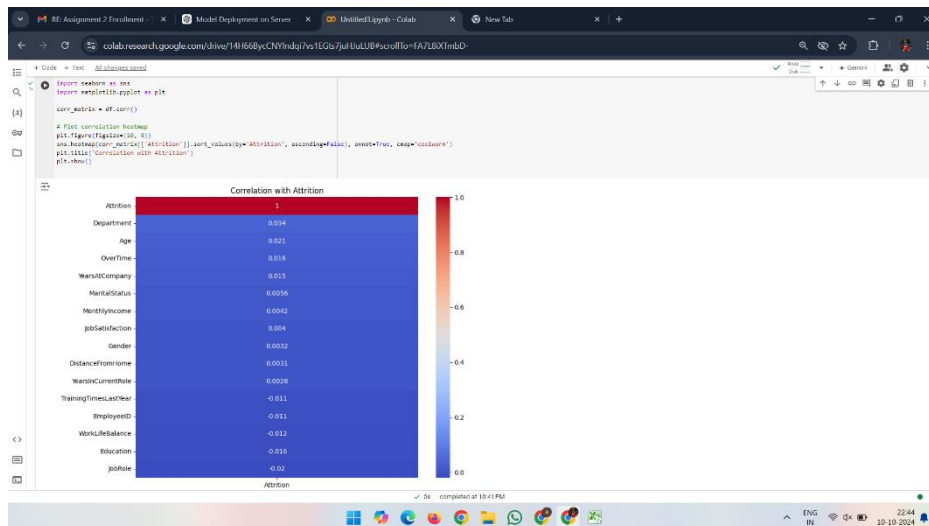
corr_matrix = df.corr()

# Plot correlation heatmap
```

Normalizing the values

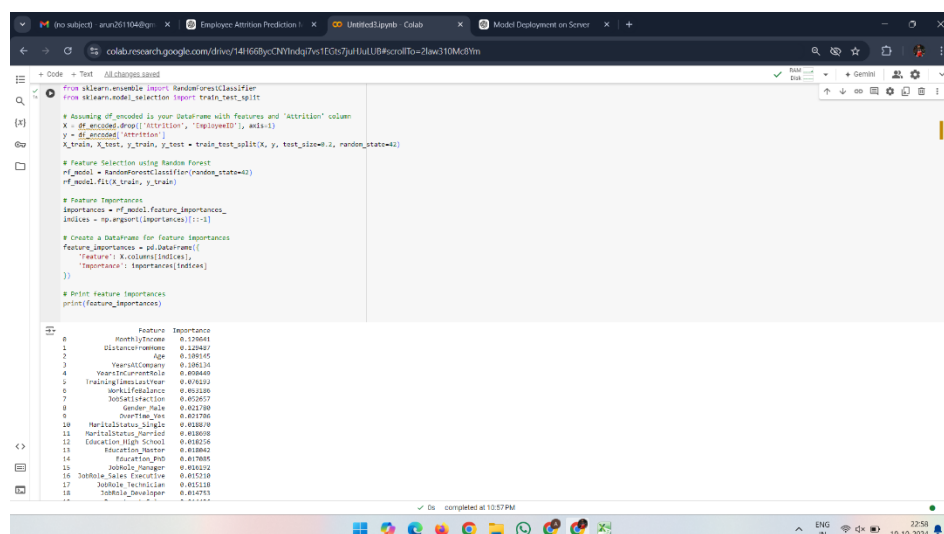
2. Exploratory Data Analysis (EDA)

- Visual and statistical analyses were conducted to uncover patterns in demographics, job roles, and job satisfaction.
- We identified correlations between attrition and specific features like age, monthly income, job satisfaction, and work-life balance.



3. Feature Selection

- We conducted feature importance analysis using Random Forest, identifying key features contributing to attrition prediction, such as 'DistanceFromHome,' 'MonthlyIncome,' and 'YearsAtCompany,' which appeared to have the most influence on employee attrition.

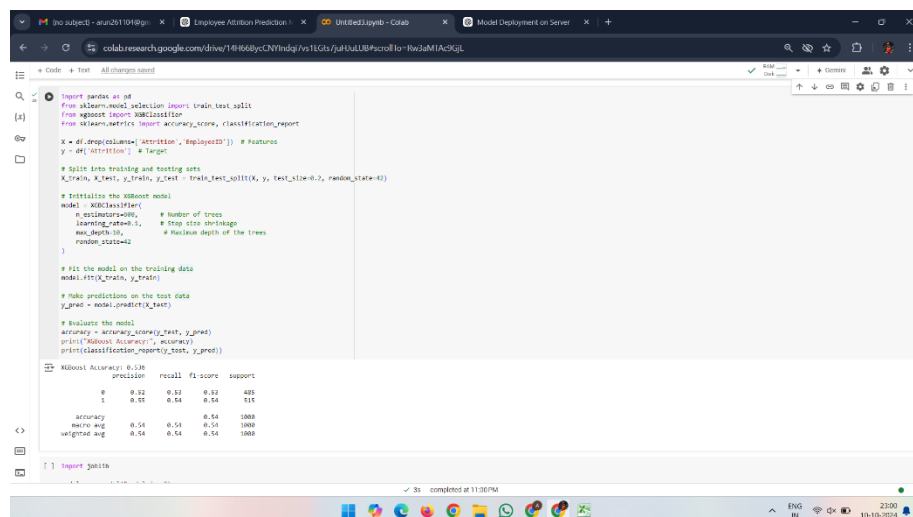


4. Model Selection and Training

- Multiple models were tested, including Logistic Regression, Random Forest, and K-Nearest Neighbors, and were evaluated based on accuracy and other performance metrics.
- XGBClassifier** was chosen as the best-performing model after testing, as it demonstrated the ability to handle complex feature interactions.

5. Model Evaluation

- We evaluated the models using accuracy, precision, recall, and F1-score metrics. Our final model, XGBClassifier, achieved an average accuracy of **53.6%**, suggesting that the dataset may lack strong patterns needed for precise prediction.



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for training and evaluating an XGBoost model. The code imports necessary libraries, splits the data into training and testing sets, initializes the XGBoost model with specific parameters, fits it to the training data, and evaluates it on the test data. The output of the evaluation is displayed as a table of metrics.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report

X = df.drop(columns=['attrition', 'employeeID']) # Features
y = df['attrition'] # Target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

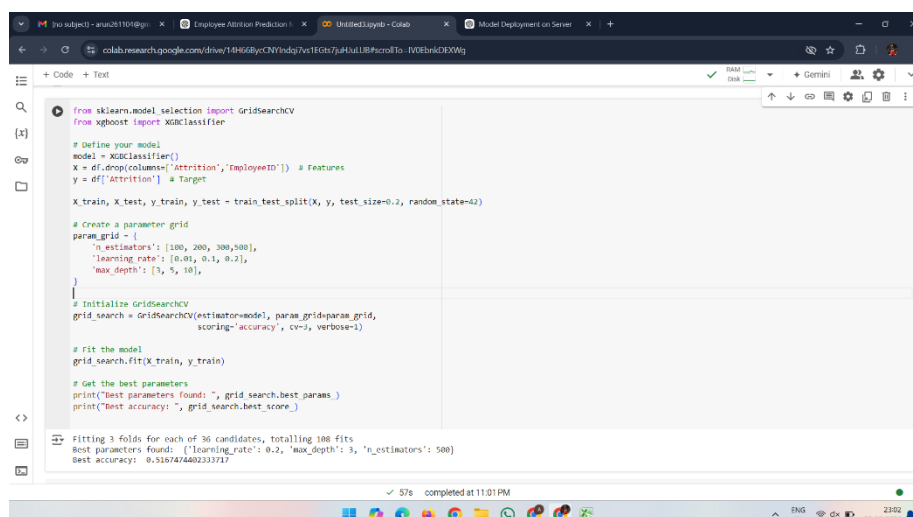
# Initialize the XGBoost model
model = XGBClassifier(
    n_estimators=500, # Number of trees
    learning_rate=0.1, # Step size shrinkage
    max_depth=10, # Maximum depth of the trees
    random_state=42
)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("XGBoost Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.52	0.52	0.52	493
1	0.55	0.54	0.54	111
accuracy	0.54	0.54	0.54	1000
macro avg	0.54	0.54	0.54	1000
weighted avg	0.54	0.54	0.54	1000



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for hyperparameter tuning using GridSearchCV. The code defines an XGBoost model, creates a parameter grid, initializes GridSearchCV, fits it to the training data, and prints the best parameters and accuracy. The output shows the best parameters found and the best accuracy.

```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Define your model
model = XGBClassifier()
X = df.drop(columns=['attrition', 'employeeID']) # Features
y = df['attrition'] # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a parameter grid
param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           scoring='accuracy', cv=5, verbose=1)

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best parameters
print("Best parameters found: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)
```

Fitting 3 folds for each of 36 candidates, totalling 108 fits.
Best parameters found: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 500}
Best accuracy: 0.516747440233717

Findings

- **Key Insights from EDA**

- **Age and Attrition:** Younger employees exhibited higher attrition rates, indicating a potential need for career development opportunities.
- **Income and Attrition:** Employees with lower incomes tended to leave more frequently, suggesting that competitive salaries are a significant factor in retention.
- **Job Satisfaction and Work-Life Balance:** Attrition was higher among employees with lower satisfaction and poor work-life balance, underscoring areas for potential improvement.

- **Model Performance**

Although **XGBClassifier** was the best-performing model, its results were moderate:

- **Overall Accuracy: 0.536 (53.6%)**

- **Precision:**

- Class 0: **0.52** (52%)
- Class 1: **0.55** (55%)

- **Recall:**

- Class 0: **0.53** (53%)
- Class 1: **0.54** (54%)

- **F1-Score:**

- Class 0: **0.53** (53%)
- Class 1: **0.54** (54%)

Despite trying different models and extensive tuning, the model's predictive accuracy remained limited. This low accuracy suggests the dataset may lack strong patterns for reliable classification.

Recommendations

Based on the findings, the following actions are recommended to address areas with higher attrition risks:

- **Career Development for Younger Employees:** Offer clear growth paths, learning opportunities, and mentorship to encourage retention.
- **Enhanced Salary Reviews:** Regularly review and adjust compensation, particularly for lower-paying roles, to stay competitive in the market.
- **Improved Job Satisfaction and Work-Life Balance:** Implement flexible working options and wellness programs, especially for employees in roles or departments reporting lower satisfaction levels.

OUTPUT SCREENSHOTS

localhost:5000 says
Prediction: employee will leave the company

Age: 39

Marital Status: Married

Education Level: Master

Department: R&D

Job Role: Analyst

Years at Company: 9

Years in Current Role: 7

Monthly Income: 85786.78

Job Satisfaction (1-5): 3

Work Life Balance (1-5): 3

Training Times Last Year: 1

OverTime: Yes

Distance from Home (km): 10

Submit

localhost:5000 says
Prediction: employee will not leave the company

Age: 48

Marital Status: Single

Education Level: PhD

Department: HR

Job Role: Manager

Years at Company: 50

Years in Current Role: 8

Monthly Income: 6150.75

Job Satisfaction (1-5): 5

Work Life Balance (1-5): 5

Training Times Last Year: 80

OverTime: No

Distance from Home (km): 18.99

Submit

Conclusion

This project yielded insights into factors that influence employee attrition, yet the model's limited performance indicates that the dataset may lack strong, identifiable patterns for precise prediction. For future work, adding new features or integrating external data sources, such as employee feedback or industry trends, may improve accuracy and support more effective retention strategies.

SAMPLE CODE

#SERVER CODE TO INTEGRATE FRONTEND FORM WITH BACKEND AND PREDICT USING MODEL

```
from flask import Flask, request, jsonify, render_template
import pandas as pd
import pickle
```

```
# Load the trained model
```

```
with open('model.pkl', 'rb') as model_file:
```

```
    model = pickle.load(model_file)
```

```
app = Flask(__name__)
```

```
@app.route('/')
def home():
```

```
    return render_template('form.html')
```

```
@app.route('/predict', methods=['POST'])
```

```
def predict():
```

```
    try:
```

```
        data = request.json['input'][0]
```

```
        print(data)
```

```
    # Prepare the input for the model
```

```
    input_data = {
```

```

'Age': int(data['Age']),
'Gender': int(data['Gender']),
'MaritalStatus': int(data['MaritalStatus']),
'Education': int(data['Education']),
'Department': int(data['Department']),
'JobRole': int(data['JobRole']),
'YearsAtCompany': int(data['YearsAtCompany']),
'YearsInCurrentRole': int(data['YearsInCurrentRole']),
'MonthlyIncome': float(data['MonthlyIncome']),
'JobSatisfaction': int(data['JobSatisfaction']),
'WorkLifeBalance': int(data['WorkLifeBalance']),
'TrainingTimesLastYear': int(data['TrainingTimesLastYear']),
'OverTime': int(data['OverTime']),
'DistanceFromHome': float(data['DistanceFromHome']) # Change to float
}

```

```

# Convert input data to DataFrame

```

```

input_df = pd.DataFrame([input_data])

```

```

# Make prediction

```

```

prediction = model.predict(input_df)

```

```

if(prediction[0]==1):

```

```

    res='employee will leave the company'

```

```

else:

```

```

    res='employee will not leave the company'

```

```

# Convert prediction to standard Python type

```

```

response = {

```

```

    'predictions': res # Convert to int

```

```

}

```



```
return jsonify(response)
```

```
except Exception as e:
```

```
    print(f"Error: {e}") # Log the error message
```

```
    return jsonify({'error': str(e)}), 500 # Return the error as a JSON response
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True, host='localhost', port=5000)
```