Configuration Manual

MSc Research Project
Master of Science in Data Analytics

Linu Anil
Student ID: X23183852

School of Computing
National College of Ireland

Supervisor: Anh Duong Trinh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Linu Anil ………………………………………………………………………………………………………
Student ID:	X23183853 ……………………………………………………………………………………………………….…
Programme:	Master of Science in Data Analytics ……………………………………………………………… **Year:** 2024 - 2025 …………………………….
Module:	Research Project ……………………………………………………………………………………………….……..
Supervisor:	Anh Doung Trinh ……………………………………………………………………………………………….….
Submission Due Date:	29/12/1014 ……………………………………………………………………………………………….…
Project Title:	Enhancing Efficiency of Employee Attrition Prediction Using Machine Learning and Ensembling Techniques ……………………………………………………………………………………………….…
Word Count:	……………1339………………………… **Page Count**…………………20………………..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Linu Anil ………………………………………………………………………………………………
Date:	28/12/2024 ………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual
Enhancing Efficiency of Employee Attrition Prediction Using Machine Learning and Ensemble Techniques

Linu Anil
X23183853

1. Introduction

In the subsequent parts of this configuration manual the hardware, the software, and the methodologies applied to carry out the research project will be described. Framework provides description of the tools and environment that are required for the development of the machine learning models used in this experimentation as well as training and testing of these models. They encompasses data preprocessing involving missing value handling, categorical variable transformation, feature scaling, outlier removal, data balancing and feature selection, model development and testing and model evaluation. In the manual there are also included few lines of python code used during development and analyses of the research project. The evaluation section outlines the of the results of the findings from the work on: "Enhancing Efficiency of Employee Attrition Prediction Using Machine Learning and Ensemble Techniques". The result of the experimented model shows that stacking model perform with highest accuracy of 99.59%, 100% precision, 98.30% recall and F1 score of 99.14%. The study also highlights how the case of the ensemble learning is useful when handling multiple dimensions of HR data, including handling issues of class imbalanced data and noisy input data. While the scope of generalization is limited by the work's data set, the results show that machine learning can indeed improve workforce analysis.

2. System Configuration

The hardware information and software information used in this research project are as follows:

2.1. Hardware Configuration

- Operating system: macOS Sequoia 15.1.1
- Processor: Apple M1 Chip
- System Type: 64-bit operating system, x64-based processor
- Hard Disk: 256GB SSD
- Installed physical memory RAM: 8GB

2.2. Software Configurations:

Table 1. Software tools specifications

Software/Tools	Version	Details
Anaconda Navigator	2.6.3	GUI for managing Anaconda environments, installing packages, and launching data science tools.
Jupyter Notebook	6.5.4	Interactive web application for creating and sharing documents with live code, equations, and visualizations.
Python	3.11.5	High-level, versatile programming language used for scripting, data analysis, and application development.
NumPy	1.26.4	Library for numerical computing, offering support for large, multi-dimensional arrays and matrices.
Pandas	2.1.4	Data manipulation and analysis library, providing DataFrames for structured data handling.
Scikit-learn	1.4.2	Machine learning library offering tools for classification, regression, clustering, and preprocessing.
Seaborn	0.12.2	Visualization library built on Matplotlib, offering high-level interface for statistical graphics.
Matplotlib	3.7.5	Library for creating static, interactive, and animated visualizations in Python.
XGBoost	2.1.1	Gradient boosting framework optimized for speed and performance, widely used for structured/tabular data.
imbalanced-learn	0.12.4	Library for handling imbalanced datasets through techniques like oversampling and undersampling.
Joblib	1.3.2	Library for efficient serialization of Python objects and lightweight parallel processing.

3. Download and Implementation

This section explain how to run the project on any macOS system.The following are the steps.

1. Download and install Anaconda Navigator software.

   Anaconda Navigator is the front end that enables users to install, launch, update and manage Python/R environments and Data Science tools such as the Jupyter notebooks, Spyder and many more, without even having to write any command scripts.

2. Create a New Environment

   It's a good practice to use separate environments for different projects.

   - Open the **Anaconda Prompt** or your terminal.
   - Create a new environment with a specific Python version:

   ```
   conda create -n myenv python=3.11. 5
   ```

   - Activate the environment:

   ```
   conda activate myenv
   ```

3. Install Jupyter Notebook

If Jupyter Notebook is not already installed in the environment, install it using:

```
conda install -c conda-forge notebook
```

4. Launch Jupyter Notebook
   - While environment is activated, run:

     ```
     jupyter notebook
     ```

   - This will start a local server and open the Jupyter Notebook interface in default web browser.
5. Install Libraries in Environment

   Use the following commands to install the specific versions of tools and libraries:

   ```
   conda install -c conda-forge jupyter
   notebook=6.5.4 numpy=1.26.4 pandas=2.1.4
   matplotlib=3.7.5 seaborn=0.12.2 scikit-learn=1.4.2


   conda install -c conda-forge xgboost=2.1.1
   imbalanced-learn=0.12.4 joblib=1.3.2
   ```

6. Add Kernel to Jupyter

   To ensure environment is listed as a kernel in Jupyter:

   ```
   python -m ipykernel install --user --name=myenv --
   display-name "Python (myenv)”
   ```

3.1. Dataset

The dataset is collected from Kaggle data repository integrates data from four sources to provide a holistic view of workforce dynamics:

1. Employee Feedback (2017-2022):
   - Scales of rating and feedback scores for employee’s performance and satisfaction.
   - Trends in feedback over time.
2. Job Structure:
   - Information about organisational structure, divisions, grades and positions.
   - Provides information concerning the variety of positions in the organization.
3. Office Locations (Canada & US):
   - Location data for 5 Canadian and 3 American offices: city, province/ state, country.
   - Especially emphasizes the geographical distribution in North America.
4. Employee Attrition:
   - Year of leaving observation, reasons and relieving statuses.
   - Examines the information about trends and contributing factors to turnover.

Canonical URL: https://creativecommons.org/publicdomain/zero/1.0/

4. Configuration and Exceution:

4.1. Importing Libraries

For the implementation of the model, the required libraries must be imported for a smooth execution. Below figure shows the imported libraries for the study.

Import Libraries

```
In [1]: import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder, StandardScaler
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score, precision_score, recall_score, f1_score
import joblib
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score, learning_curve, GridSearchCV
```

Figure 1: Importing Required Libraries

4.2. Data Visualization

Each column in the dataset is visualised and compared with the target variable, for categorical columns use bar-chart and for numerical columns use box plot, this make easy analysis of data and it pattern.

```
In [12]: numerical_cols = df.select_dtypes(include=['number']).columns
numerical_cols = numerical_cols.drop('Attrition', errors='ignore')
total_numerical_plots = len(numerical_cols)
rows = (total_numerical_plots // 4) + 1
cols = 4
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(16, 6 * rows))
axes = axes.flatten()
for i, col in enumerate(numerical_cols):
    sns.boxplot(data=df, x='Attrition', y=col, ax=axes[i])
    axes[i].set_title(f"Box plot of {col} by Attrition")
    axes[i].set_xlabel('Attrition')
    axes[i].set_ylabel(col)
for i in range(total_numerical_plots, len(axes)):
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```

```
In [13]: categorical_cols = df.select_dtypes(include=['object', 'category']).columns
categorical_cols = categorical_cols.drop('Attrition', errors='ignore')
total_categorical_plots = len(categorical_cols)
rows = (total_categorical_plots // 4) + 1
cols = 4
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(16, 6 * rows))
axes = axes.flatten()
for i, col in enumerate(categorical_cols):
    attrition_counts = df.groupby([col, 'Attrition']).size().unstack(fill_value=0)
    x_positions = range(len(attrition_counts))
    bar_width = 0.4
    axes[i].bar(x_positions, attrition_counts['Yes'], width=bar_width,
                label='Yes', color='skyblue')
    axes[i].bar([x + bar_width for x in x_positions], attrition_counts['No'],
                width=bar_width, label='No', color='orange')
    axes[i].set_xticks([x + bar_width / 2 for x in x_positions])
    axes[i].set_xticklabels(attrition_counts.index, rotation=45)
    axes[i].set_title(f"Bar chart of {col} by Attrition")
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')
    axes[i].legend(title='Attrition')
for i in range(total_categorical_plots, len(axes)):
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```

Figure 2: Charts to see the distribution of data against Attrition

4.3. Data Pre-processing

Each In the data preprocessing step the duplicates were analyse and the dataset did not contained any duplicate values and missing values were analysed and 3 columns ('LeavingYear', 'Reason', 'RelievingStatus') contain more than 50% missing values so the columns were dropped. Then checked unique values in each categorical columns and found that 3 columns ('EmployeeCount', 'Over18', 'StandardHours') has only one value so these columns also dropped.

```
In [11]: # Data Preprocessing
         # Drop columns with more than 50% missing values
         df = df.drop(['LeavingYear', 'Reason', 'RelievingStatus',
                       'EmployeeCount', 'Over18', 'StandardHours'], axis=1)
```

Figure 3: Dropping unnecessary columns

4.4. Feature Engineering

1. Use mapping technique to convert categorical columns to numerical with the help of dictionary.

```
In [14]: categorical_cols = ['BusinessTravel', 'Department', 'EducationField',
                             'Gender', 'MaritalStatus', 'OverTime', 'Attrition',
                             'office_code', 'JobLevel_updated']

In [15]: mapping_dict = {
             'BusinessTravel': {'Travel_Rarely': 0, 'Travel_Frequently': 1,
                                'Non-Travel': 2},
             'Department': {'Corporate Functions': 0, 'Marketing': 1,
                            'Delivery': 2,
                            'Product': 3, 'Sales': 4, 'HR': 5},
             'EducationField': {'Doctorate': 0, 'Diploma': 1, 'Masters': 2,
                                'Bachelors': 3},
             'Gender': {'Male': 0, 'Female': 1},
             'MaritalStatus': {'Married': 0, 'Divorced': 1, 'Single': 2},
             'OverTime': {'Yes': 1, 'No': 0},
             'Attrition': {'Yes': 1, 'No': 0},
             'office_code': {'BOS': 0, 'NYC': 1, 'OTT': 2, 'CAL': 3,
                             'PHL': 4,
                             'MKM': 5, 'VAN': 6, 'TOR': 7},
             'JobLevel_updated': {'L7': 6, 'L6': 5, 'L5': 4, 'L3': 3,
                                  'L2': 2, 'L4': 1, 'L1': 0}
         }

In [16]: for col, mapping in mapping_dict.items():
             df[col] = df[col].map(mapping)
```

Figure 4: Covert categorical column to numerical using Mapping technique

2. Using IQR method remove all outliers in the data to ensure data quality.

```
In [17]: numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
         initial_row_count = len(df)
         for col in numerical_cols:
             Q1 = df[col].quantile(0.25)
             Q3 = df[col].quantile(0.75)
             IQR = Q3 - Q1
             lower_bound = Q1 - 1.5 * IQR
             upper_bound = Q3 + 1.5 * IQR
         # Remove rows with outliers (those that are outside 1.5*IQR)
         df_no_outliers = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
         final_row_count = len(df_no_outliers)
         data_removed_percentage = ((initial_row_count - final_row_count) / initial_row_count) * 100
         data_removed_percentage

Out[17]: 0.05959919541086195
```

Figure 5: Remove outliers using IQR method

3. Data split into training and testing for model development and evaluation.

Splitting data into features (X) and target (y)

```
In [18]: X = df.drop('Attrition', axis=1)
         y = df['Attrition']
```

Train-Test Split

```
In [19]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [22]: X_train.shape
Out[22]: (10738, 31)

In [23]: X_test.shape
Out[23]: (2685, 31)
```

Figure 6: Data splitting

4. The X_train and X_test were scaled using StandardScaler to make all the feature to a unique scale which make the training more reliable.

```
In [20]: scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)
```

Figure 7: Scaling the data

5. The X_train_scaled and y_train were balanced using SMOTE to ensure the model train on each class in same level.

```
In [41]: smote = SMOTE(random_state=42)
         X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)
```

Figure 8: Balancing the data using SMOTE

6. The X_train_scaled and X_test_scaled data were under gone through the PCA technique with 95% n_components to reduce the data dimensionality to reduce the execution time and complexity while modelling.

```
In [50]: pca = PCA(n_components=0.95)
         X_train_pca = pca.fit_transform(X_train_scaled)
         X_test_pca = pca.transform(X_test_scaled)
```

Figure 9: PCA for dimensionality reduction

4.5. Experimenting the models

1. Experiment 1: Basic Random Forest and XGBoost models with StandardScaler.

```
In [29]: rf_basic = RandomForestClassifier()
         xgb_basic = xgb.XGBClassifier()

In [30]: rf_basic.fit(X_train_scaled, y_train)

Out[30]: RandomForestClassifier
         RandomForestClassifier()

In [31]: xgb_basic.fit(X_train_scaled, y_train)

Out[31]: XGBClassifier
         XGBClassifier(base_score=None, booster=None, callbacks=None,
                       colsample_bylevel=None, colsample_bynode=None,
                       colsample_bytree=None, device=None, early_stopping_rounds=None,
                       enable_categorical=False, eval_metric=None, feature_types=None,
                       gamma=None, grow_policy=None, importance_type=None,
                       interaction_constraints=None, learning_rate=None, max_bin=None,
                       max_cat_threshold=None, max_cat_to_onehot=None,
                       max_delta_step=None, max_depth=None, max_leaves=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       multi_strategy=None, n_estimators=None, n_jobs=None,
```

Prediction

```
In [32]: rf_pred_train = rf_basic.predict(X_train_scaled)
         rf_pred = rf_basic.predict(X_test_scaled)
         rf_yprob = rf_basic.predict_proba(X_test_scaled)[:, 1]
         xgb_pred_train = xgb_basic.predict(X_train_scaled)
         xgb_pred = xgb_basic.predict(X_test_scaled)
         xgb_yprob = xgb_basic.predict_proba(X_test_scaled)[:, 1]
```

Figure 10: Basic Random Forest and XGBoost models with StandardScaler

2. Experiment 2: Random Forest and XGBoost models after removing outliers.

```
In [38]: X_outliers_removed = df_no_outliers.drop('Attrition', axis=1)
         y_outliers_removed = df_no_outliers['Attrition']

In [39]: X_train_out, X_test_out, y_train_out, y_test_out = train_test_split(X_outliers_

In [40]: X_train_scaled_out = scaler.fit_transform(X_train_out)
         X_test_scaled_out = scaler.transform(X_test_out)

In [41]: rf_outliers = RandomForestClassifier()
         xgb_outliers = xgb.XGBClassifier()

In [42]: rf_outliers.fit(X_train_scaled_out, y_train_out)

Out[42]: RandomForestClassifier
         RandomForestClassifier()

In [43]: xgb_outliers.fit(X_train_scaled_out, y_train_out)

Out[43]: XGBClassifier
         XGBClassifier(base_score=None, booster=None, callbacks=None,
                       colsample_bylevel=None, colsample_bynode=None,
                       colsample_bytree=None, device=None, early_stopping_rounds=None,
                       enable_categorical=False, eval_metric=None, feature_types=None,
                       gamma=None, grow_policy=None, importance_type=None,
                       interaction_constraints=None, learning_rate=None, max_bin=None,
                       max_cat_threshold=None, max_cat_to_onehot=None,
                       max_delta_step=None, max_depth=None, max_leaves=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       multi_strategy=None, n_estimators=None, n_jobs=None,

In [44]: rf_pred_out_train = rf_outliers.predict(X_train_scaled_out)
         rf_pred_out = rf_outliers.predict(X_test_scaled_out)
         rf_yprob_out = rf_outliers.predict_proba(X_test_scaled_out)[:, 1]

         xgb_pred_out_train = xgb_outliers.predict(X_train_scaled_out)
         xgb_pred_out = xgb_outliers.predict(X_test_scaled_out)
         xgb_yprob_out = xgb_outliers.predict_proba(X_test_scaled_out)[:, 1]
```

Figure 11: Random Forest and XGBoost models after removing outliers.

3. Experiment 3: Random Forest and XGBoost models with SMOTE

```
In [49]: smote = SMOTE(random_state=42)
         X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

In [50]: rf_smote = RandomForestClassifier()
         xgb_smote = xgb.XGBClassifier()

In [51]: rf_smote.fit(X_train_smote, y_train_smote)

Out[51]: RandomForestClassifier
         RandomForestClassifier()

In [52]: xgb_smote.fit(X_train_smote, y_train_smote)

Out[52]: XGBClassifier
         XGBClassifier(base_score=None, booster=None, callbacks=None,
                       colsample_bylevel=None, colsample_bynode=None,
                       colsample_bytree=None, device=None, early_stopping_rounds=None,
                       enable_categorical=False, eval_metric=None, feature_types=None,
                       gamma=None, grow_policy=None, importance_type=None,
                       interaction_constraints=None, learning_rate=None, max_bin=None,
                       max_cat_threshold=None, max_cat_to_onehot=None,
                       max_delta_step=None, max_depth=None, max_leaves=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       multi_strategy=None, n_estimators=None, n_jobs=None,

In [53]: rf_pred_smote_train = rf_smote.predict(X_train_smote)
         rf_pred_smote = rf_smote.predict(X_test_scaled)
         rf_yprob_smote = rf_smote.predict_proba(X_test_scaled)[:, 1]

         xgb_pred_smote_train = xgb_smote.predict(X_train_smote)
         xgb_pred_smote = xgb_smote.predict(X_test_scaled)
         xgb_yprob_smote = xgb_smote.predict_proba(X_test_scaled)[:, 1]
```

Figure 12: Random Forest and XGBoost models with SMOTE

4. Experiment 4: Random Forest and XGBoost models with PCA

```
In [58]: pca = PCA(n_components=0.95)
         X_train_pca = pca.fit_transform(X_train_scaled)
         X_test_pca = pca.transform(X_test_scaled)

In [59]: rf_pca = RandomForestClassifier()
         xgb_pca = xgb.XGBClassifier()

In [60]: rf_pca.fit(X_train_pca, y_train)

Out[60]: RandomForestClassifier
         RandomForestClassifier()

In [61]: xgb_pca.fit(X_train_pca, y_train)

Out[61]: XGBClassifier
         XGBClassifier(base_score=None, booster=None, callbacks=None,
                       colsample_bylevel=None, colsample_bynode=None,
                       colsample_bytree=None, device=None, early_stopping_rounds=None,
                       enable_categorical=False, eval_metric=None, feature_types=None,
                       gamma=None, grow_policy=None, importance_type=None,
                       interaction_constraints=None, learning_rate=None, max_bin=None,
                       max_cat_threshold=None, max_cat_to_onehot=None,
                       max_delta_step=None, max_depth=None, max_leaves=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       multi_strategy=None, n_estimators=None, n_jobs=None,

In [62]: rf_pred_pca = rf_pca.predict(X_test_pca)
         rf_pred_pca_train = rf_pca.predict(X_train_pca)
         rf_yprob_pca = rf_pca.predict_proba(X_test_pca)[:, 1]

         xgb_pred_pca = xgb_pca.predict(X_test_pca)
         xgb_pred_pca_train = xgb_pca.predict(X_train_pca)
         xgb_yprob_pca = xgb_pca.predict_proba(X_test_pca)[:, 1]
```

Figure 13: Random Forest and XGBoost models with PCA

5. Experiment 5: Hyper parameter Tuning for Random Forest and XGBoost models

```
In [67]: # Random Forest Hyperparameter Tuning
         rf_param_grid = {
             'n_estimators': [100, 150],
             'max_depth': [None, 10, 20],
             'min_samples_split': [2, 5],
             'min_samples_leaf': [1, 2]
         }

In [68]: rf_grid_search = GridSearchCV(estimator=RandomForestClassifier(),
                                       param_grid=rf_param_grid, cv=5)

In [69]: rf_grid_search.fit(X_train_scaled, y_train)

Out[69]:  ▸ GridSearchCV
          ▸ estimator: RandomForestClassifier
              ▸ RandomForestClassifier

In [70]: rf_best_model = rf_grid_search.best_estimator_

In [71]: print("Best Parameters:", rf_grid_search.best_params_)

         Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split':
         mators': 150}

In [72]: rf_pred_tuned = rf_best_model.predict(X_test_scaled)
         rf_yprob_tuned = rf_best_model.predict_proba(X_test_scaled)[:, 1]

         rf_pred_tuned_train = rf_best_model.predict(X_train_scaled)
```

Figure 14: Hyper parameter Tuning for Random Forest

```
In [75]: # XGBoost Hyperparameter Tuning
         xgb_param_grid = {
             'n_estimators': [100, 150],
             'learning_rate': [0.01, 0.1],
             'max_depth': [3, 5],
             'subsample': [0.8, 1.0]
         }

In [76]: xgb_grid_search = GridSearchCV(estimator=xgb.XGBClassifier(),
                                        param_grid=xgb_param_grid, cv=5)

In [77]: xgb_grid_search.fit(X_train_scaled, y_train)

Out[77]:  ▸ GridSearchCV
          ▸ estimator: XGBClassifier
              ▸ XGBClassifier

In [78]: print("Best Parameters:", xgb_grid_search.best_params_)

         Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150, ':
         1.0}

In [79]: xgb_best_model = xgb_grid_search.best_estimator_

In [80]: xgb_pred_tuned = xgb_best_model.predict(X_test_scaled)
         xgb_yprob_tuned = xgb_best_model.predict_proba(X_test_scaled)[:, 1]

         xgb_pred_tuned_train = xgb_best_model.predict(X_train_scaled)
```

Figure 15: Hyper parameter Tuning for XGBoost models

6. Experiment 6: Create stacked models using the best models

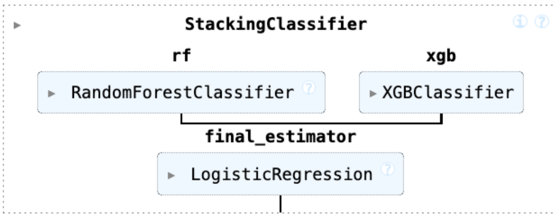
```
In [84]: base_learners = [
             ('rf', rf_best_model),
             ('xgb', xgb_best_model)
         ]

In [85]: meta_model = LogisticRegression()

In [86]: stacked_model = StackingClassifier(estimators=base_learners,
                                            final_estimator=meta_model)

In [87]: # Train the stacked model
         stacked_model.fit(X_train_scaled, y_train)

Out[87]:
```




```
In [88]: stacked_pred_train = stacked_model.predict(X_train_scaled)
         stacked_pred_test = stacked_model.predict(X_test_scaled)
         stacked_yprob = stacked_model.predict_proba(X_test_scaled)[:, 1]
```

Figure 16: Stacked models using the best models

4.6. Evaluation

1. Experiment 1: Evaluation of Basic Random Forest and XGBoost Model:

```
In [34]: print("Basic Random Forest Model Evaluation:")
         print(confusion_matrix(y_test, rf_pred))
         print(classification_report(y_test, rf_pred))
         rf_acc = accuracy_score(y_test, rf_pred)
         precision_rf = precision_score(y_test, rf_pred)
         recall_rf = recall_score(y_test, rf_pred)
         f1_rf = f1_score(y_test, rf_pred)
         print(f"Accuracy: {rf_acc}")
         print(f"Precision: {precision_rf}")
         print(f"Recall: {recall_rf}")
         print(f"F1 Score: {f1_rf}")

         Basic Random Forest Model Evaluation:
         [[2038    0]
          [  36  611]]
                       precision    recall  f1-score   support

                    0       0.98      1.00      0.99      2038
                    1       1.00      0.94      0.97       647

             accuracy                           0.99      2685
            macro avg       0.99      0.97      0.98      2685
         weighted avg       0.99      0.99      0.99      2685

         Accuracy: 0.9865921787709497
         Precision: 1.0
         Recall: 0.9443585780525502
         F1 Score: 0.9713831478537361
```

Figure 17: Basic Random Forest Model Evaluation

```
In [37]: print("Basic XGBoost Model Evaluation:")
print(confusion_matrix(y_test, xgb_pred))
print(classification_report(y_test, xgb_pred))
xgb_acc = accuracy_score(y_test, xgb_pred)
precision_xgb = precision_score(y_test, xgb_pred)
recall_xgb = recall_score(y_test, xgb_pred)
f1_xgb = f1_score(y_test, xgb_pred)
print(f"Accuracy: {xgb_acc}")
print(f"Precision: {precision_xgb}")
print(f"Recall: {recall_xgb}")
print(f"F1 Score: {f1_xgb}")

fpr_xgb, tpr_xgb, _ = roc_curve(y_test, xgb_yprob)  # Make sure to use pr
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)
print(f"ROC AUC: {roc_auc_xgb}")
```

```
Basic XGBoost Model Evaluation:
[[2037    1]
 [  13  634]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2038
           1       1.00      0.98      0.99       647

    accuracy                           0.99      2685
   macro avg       1.00      0.99      0.99      2685
weighted avg       0.99      0.99      0.99      2685

Accuracy: 0.9947858472998138
Precision: 0.9984251968503937
Recall: 0.9799072642967542
F1 Score: 0.9890795631825273
ROC AUC: 0.9979917881730884
```

Figure 18: Basic XGBoost Model Evaluation

2. Experiment 2: Evaluation of Random Forest and XGBoost models after removing outliers

```
In [46]: print("Random Forest Model after Outliers Removal Evaluation:")
print(confusion_matrix(y_test_out, rf_pred_out))
print(classification_report(y_test_out, rf_pred_out))
rf_out_acc = accuracy_score(y_test_out, rf_pred_out)
precision_rf_out = precision_score(y_test_out, rf_pred_out)
recall_rf_out = recall_score(y_test_out, rf_pred_out)
f1_rf_out = f1_score(y_test_out, rf_pred_out)
print(f"Accuracy: {rf_out_acc}")
print(f"Precision: {precision_rf_out}")
print(f"Recall: {recall_rf_out}")
print(f"F1 Score: {f1_rf_out}")

fpr_rf_out, tpr_rf_out, _ = roc_curve(y_test_out, rf_yprob_out)  # Predic
roc_auc_rf_out = auc(fpr_rf_out, tpr_rf_out)
print(f"ROC AUC: {roc_auc_rf_out}")
```

```
Random Forest Model after Outliers Removal Evaluation:
[[2052    4]
 [  26  601]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2056
           1       0.99      0.96      0.98       627

    accuracy                           0.99      2683
   macro avg       0.99      0.98      0.98      2683
weighted avg       0.99      0.99      0.99      2683

Accuracy: 0.9888184867685427
Precision: 0.9933884297520661
Recall: 0.9585326953748007
F1 Score: 0.9756493506493507
ROC AUC: 0.9991373906999548
```

Figure 19: Evaluation of Random Forest model after removing outliers

```
In [48]: print("XGBoost Model after Outliers Removal Evaluation:")
         print(confusion_matrix(y_test_out, xgb_pred_out))
         print(classification_report(y_test_out, xgb_pred_out))
         xgb_out_acc = accuracy_score(y_test_out, xgb_pred_out)
         precision_xgb_out = precision_score(y_test_out, xgb_pred_out)
         recall_xgb_out = recall_score(y_test_out, xgb_pred_out)
         f1_xgb_out = f1_score(y_test_out, xgb_pred_out)
         print(f"Accuracy: {xgb_out_acc}")
         print(f"Precision: {precision_xgb_out}")
         print(f"Recall: {recall_xgb_out}")
         print(f"F1 Score: {f1_xgb_out}")

         fpr_xgb_out, tpr_xgb_out, _ = roc_curve(y_test_out, xgb_yprob_
         roc_auc_xgb_out = auc(fpr_xgb_out, tpr_xgb_out)
         print(f"ROC AUC: {roc_auc_xgb_out}")

         XGBoost Model after Outliers Removal Evaluation:
         [[2055    1]
          [  14  613]]
                       precision    recall  f1-score   support

                    0       0.99      1.00      1.00      2056
                    1       1.00      0.98      0.99       627

             accuracy                           0.99      2683
            macro avg       1.00      0.99      0.99      2683
         weighted avg       0.99      0.99      0.99      2683

         Accuracy: 0.9944092433842714
         Precision: 0.998371335504886
         Recall: 0.9776714513556619
         F1 Score: 0.9879129734085415
         ROC AUC: 0.9988790733466137
```

Figure 20: Evaluation of XGBoost model after removing outliers

3. Experiment 3: Evaluation of Random Forest and XGBoost models with SMOTE

```
In [55]: print("Random Forest Model with SMOTE Evaluation:")
         print(confusion_matrix(y_test, rf_pred_smote))
         print(classification_report(y_test, rf_pred_smote))
         rf_smote_acc = accuracy_score(y_test, rf_pred_smote)
         precision_rf_smote = precision_score(y_test, rf_pred_smote)
         recall_rf_smote = recall_score(y_test, rf_pred_smote)
         f1_rf_smote = f1_score(y_test, rf_pred_smote)
         print(f"Accuracy: {rf_smote_acc}")
         print(f"Precision: {precision_rf_smote}")
         print(f"Recall: {recall_rf_smote}")
         print(f"F1 Score: {f1_rf_smote}")

         fpr_rf_smote, tpr_rf_smote, _ = roc_curve(y_test, rf_yprob_smote)
         roc_auc_rf_smote = auc(fpr_rf_smote, tpr_rf_smote)
         print(f"ROC AUC: {roc_auc_rf_smote}")

         Random Forest Model with SMOTE Evaluation:
         [[2038    0]
          [  30  617]]
                       precision    recall  f1-score   support

                    0       0.99      1.00      0.99      2038
                    1       1.00      0.95      0.98       647

             accuracy                           0.99      2685
            macro avg       0.99      0.98      0.98      2685
         weighted avg       0.99      0.99      0.99      2685

         Accuracy: 0.9888268156424581
         Precision: 1.0
         Recall: 0.9536321483771252
         F1 Score: 0.9762658227848101
         ROC AUC: 0.9982549488618868
```

Figure 21: Evaluation of Random Forest Model with SMOTE

```
In [57]: print("XGBoost Model with SMOTE Evaluation:")
print(confusion_matrix(y_test, xgb_pred_smote))
print(classification_report(y_test, xgb_pred_smote))
xgb_smote_acc = accuracy_score(y_test, xgb_pred_smote)
precision_xgb_smote = precision_score(y_test, xgb_pred_smote)
recall_xgb_smote = recall_score(y_test, xgb_pred_smote)
f1_xgb_smote = f1_score(y_test, xgb_pred_smote)
print(f"Accuracy: {xgb_smote_acc}")
print(f"Precision: {precision_xgb_smote}")
print(f"Recall: {recall_xgb_smote}")
print(f"F1 Score: {f1_xgb_smote}")

fpr_xgb_smote, tpr_xgb_smote, _ = roc_curve(y_test, xgb_yprob_smote)
roc_auc_xgb_smote = auc(fpr_xgb_smote, tpr_xgb_smote)
print(f"ROC AUC: {roc_auc_xgb_smote}")
```

```
XGBoost Model with SMOTE Evaluation:
[[2037    1]
 [  12  635]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2038
           1       1.00      0.98      0.99       647

    accuracy                           1.00      2685
   macro avg       1.00      0.99      0.99      2685
weighted avg       1.00      1.00      1.00      2685

Accuracy: 0.9951582867783985
Precision: 0.9984276729559748
Recall: 0.98145285935085
F1 Score: 0.9898674980514419
ROC AUC: 0.9974904936045127
```

Figure 22: Evaluation of XGBoost Model with SMOTE

4. Experiment 4: Evaluation of Random Forest and XGBoost models with PCA

```
In [64]: print("Random Forest Model with PCA Evaluation:")
print(confusion_matrix(y_test, rf_pred_pca))
print(classification_report(y_test, rf_pred_pca))
rf_pca_acc = accuracy_score(y_test, rf_pred_pca)
precision_rf_pca = precision_score(y_test, rf_pred_pca)
recall_rf_pca = recall_score(y_test, rf_pred_pca)
f1_rf_pca = f1_score(y_test, rf_pred_pca)
print(f"Accuracy: {rf_pca_acc}")
print(f"Precision: {precision_rf_pca}")
print(f"Recall: {recall_rf_pca}")
print(f"F1 Score: {f1_rf_pca}")

fpr_rf_pca, tpr_rf_pca, _ = roc_curve(y_test, rf_yprob_pca)  # Predict
roc_auc_rf_pca = auc(fpr_rf_pca, tpr_rf_pca)
print(f"ROC AUC: {roc_auc_rf_pca}")
```

```
Random Forest Model with PCA Evaluation:
[[2005   33]
 [ 151  496]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.96      2038
           1       0.94      0.77      0.84       647

    accuracy                           0.93      2685
   macro avg       0.93      0.88      0.90      2685
weighted avg       0.93      0.93      0.93      2685

Accuracy: 0.9314711359404096
Precision: 0.9376181474480151
Recall: 0.7666151468315301
F1 Score: 0.8435374149659864
ROC AUC: 0.966089052970379
```

Figure 23: Evaluation Random Forest models with PCA

```
In [66]: print("XGBoost Model with PCA Evaluation:")
         print(confusion_matrix(y_test, xgb_pred_pca))
         print(classification_report(y_test, xgb_pred_pca))
         xgb_pca_acc = accuracy_score(y_test, xgb_pred_pca)
         precision_xgb_pca = precision_score(y_test, xgb_pred_pca)
         recall_xgb_pca = recall_score(y_test, xgb_pred_pca)
         f1_xgb_pca = f1_score(y_test, xgb_pred_pca)
         print(f"Accuracy: {xgb_pca_acc}")
         print(f"Precision: {precision_xgb_pca}")
         print(f"Recall: {recall_xgb_pca}")
         print(f"F1 Score: {f1_xgb_pca}")

         fpr_xgb_pca, tpr_xgb_pca, _ = roc_curve(y_test, xgb_yprob_pca)  # Pred
         roc_auc_xgb_pca = auc(fpr_xgb_pca, tpr_xgb_pca)
         print(f"ROC AUC: {roc_auc_xgb_pca}")

XGBoost Model with PCA Evaluation:
[[2012   26]
 [ 107  540]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      2038
           1       0.95      0.83      0.89       647

    accuracy                           0.95      2685
   macro avg       0.95      0.91      0.93      2685
weighted avg       0.95      0.95      0.95      2685

Accuracy: 0.9504655493482309
Precision: 0.9540636042402827
Recall: 0.8346213292117465
F1 Score: 0.8903544929925804
ROC AUC: 0.9822719185551796
```

Figure 24: Evaluation XGBoost models with PCA

5. Experiment 5: Evaluation of Hyper parameter Tuning for Random Forest and XGBoost models

```
In [71]: print("Best Parameters:", rf_grid_search.best_params_)

Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_esti
mators': 150}
```

Figure: Best parameters for Random Forest model

```
In [74]: print("Tuned Random Forest Model Evaluation:")
         print(confusion_matrix(y_test, rf_pred_tuned))
         print(classification_report(y_test, rf_pred_tuned))
         rf_tuned_acc = accuracy_score(y_test, rf_pred_tuned)
         precision_rf_tuned = precision_score(y_test, rf_pred_tuned)
         recall_rf_tuned = recall_score(y_test, rf_pred_tuned)
         f1_rf_tuned = f1_score(y_test, rf_pred_tuned)
         print(f"Accuracy: {rf_tuned_acc}")
         print(f"Precision: {precision_rf_tuned}")
         print(f"Recall: {recall_rf_tuned}")
         print(f"F1 Score: {f1_rf_tuned}")

         fpr_rf_tuned, tpr_rf_tuned, _ = roc_curve(y_test, rf_yprob_tuned)
         roc_auc_rf_tuned = auc(fpr_rf_tuned, tpr_rf_tuned)
         print(f"ROC AUC: {roc_auc_rf_tuned}")

Tuned Random Forest Model Evaluation:
[[2037    1]
 [  41  606]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      2038
           1       1.00      0.94      0.97       647

    accuracy                           0.98      2685
   macro avg       0.99      0.97      0.98      2685
weighted avg       0.98      0.98      0.98      2685

Accuracy: 0.9843575418994414
Precision: 0.9983525535420099
Recall: 0.9366306027820711
F1 Score: 0.9665071770334929
ROC AUC: 0.9980266740280876
```

Figure 25: Evaluation of Random Forest tuned model

```
In [78]: print("Best Parameters:", xgb_grid_search.best_params_)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 150, 'subsample':
1.0}
```

Figure: Best parameters for XGBoost model

```
In [82]: print("Tuned XGBoost Model Evaluation:")
print(confusion_matrix(y_test, xgb_pred_tuned))
print(classification_report(y_test, xgb_pred_tuned))
xgb_tuned_acc = accuracy_score(y_test, xgb_pred_tuned)
precision_xgb_tuned = precision_score(y_test, xgb_pred_tuned)
recall_xgb_tuned = recall_score(y_test, xgb_pred_tuned)
f1_xgb_tuned = f1_score(y_test, xgb_pred_tuned)
print(f"Accuracy: {xgb_tuned_acc}")
print(f"Precision: {precision_xgb_tuned}")
print(f"Recall: {recall_xgb_tuned}")
print(f"F1 Score: {f1_xgb_tuned}")
```

```
Tuned XGBoost Model Evaluation:
[[2038    0]
 [  14  633]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2038
           1       1.00      0.98      0.99       647

    accuracy                           0.99      2685
   macro avg       1.00      0.99      0.99      2685
weighted avg       0.99      0.99      0.99      2685

Accuracy: 0.9947858472998138
Precision: 1.0
Recall: 0.9783616692426584
F1 Score: 0.9890625
```

Figure 26: Evaluation of XGBoost tuned model

6. Experiment 6: Evaluation of Stacked models using the best experimented models

```
In [90]: # Stacked Model Evaluation
print("Stacked Model Evaluation:")
print(confusion_matrix(y_test, stacked_pred_test))
print(classification_report(y_test, stacked_pred_test))
stacked_acc = accuracy_score(y_test, stacked_pred_test)
stacked_precision = precision_score(y_test, stacked_pred_test)
stacked_recall = recall_score(y_test, stacked_pred_test)
stacked_f1 = f1_score(y_test, stacked_pred_test)
print(f"Accuracy: {stacked_acc}")
print(f"Precision: {stacked_precision}")
print(f"Recall: {stacked_recall}")
print(f"F1 Score: {stacked_f1}")

# ROC AUC for Stacked Model
fpr_stacked, tpr_stacked, _ = roc_curve(y_test, stacked_yprob)
roc_auc_stacked = auc(fpr_stacked, tpr_stacked)
print(f"ROC AUC: {roc_auc_stacked}")
```

```
Stacked Model Evaluation:
[[2038    0]
 [  11  636]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2038
           1       1.00      0.98      0.99       647

    accuracy                           1.00      2685
   macro avg       1.00      0.99      0.99      2685
weighted avg       1.00      1.00      1.00      2685

Accuracy: 0.9959031657355679
Precision: 1.0
Recall: 0.9829984544049459
F1 Score: 0.9914263445050663
ROC AUC: 0.9990778000069772
```

Figure 27: Evaluation of Stacked models using the best experimented models

4.7. Comparing Results

1. Making the results of each experiment to a data frame.

```
In [94]: # Lists of metrics for each model
models = ['Random Forest', 'XGBoost', 'Random Forest (Outliers Removed)',
          'XGBoost (Outliers Removed)',
          'Random Forest (SMOTE)', 'XGBoost (SMOTE)', 'Random Forest (PCA)',
          'XGBoost (PCA)',
          'Random Forest (Tuned)', 'XGBoost (Tuned)', 'Stacked Model']

accuracy = [rf_acc, xgb_acc, rf_out_acc, xgb_out_acc, rf_smote_acc, xgb_smote_acc,
            rf_pca_acc, xgb_pca_acc, rf_tuned_acc, xgb_tuned_acc, stacked_acc]

precision = [precision_rf, precision_xgb, precision_rf_out, precision_xgb_out,
             precision_rf_smote,
             precision_xgb_smote, precision_rf_pca, precision_xgb_pca,
             precision_rf_tuned, precision_xgb_tuned,
             stacked_precision]

recall = [recall_rf, recall_xgb, recall_rf_out, recall_xgb_out,
          recall_rf_smote, recall_xgb_smote,
          recall_rf_pca, recall_xgb_pca, recall_rf_tuned,
          recall_xgb_tuned, stacked_recall]

f1_score = [f1_rf, f1_xgb, f1_rf_out, f1_xgb_out, f1_rf_smote, f1_xgb_smote,
            f1_rf_pca, f1_xgb_pca, f1_rf_tuned, f1_xgb_tuned, stacked_f1]

# Create the DataFrame
df_metrics = pd.DataFrame({
    'Model': models,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1_score
})

# Display the DataFrame
df_metrics
```

Figure 28: Evaluation Result to data frame

2. Visualizing the results of each experiment barchart.

```
In [95]: fig, ax = plt.subplots(figsize=(14, 8))
bar_width = 0.2  # Width of the bars
x = np.arange(len(df_metrics['Model']))  # The x locations for the models
bar1 = ax.bar(x - 1.5*bar_width, df_metrics['Accuracy'], bar_width,
              label='Accuracy', color='blue')
bar2 = ax.bar(x - 0.5*bar_width, df_metrics['Precision'], bar_width,
              label='Precision', color='green')
bar3 = ax.bar(x + 0.5*bar_width, df_metrics['Recall'], bar_width,
              label='Recall', color='orange')
bar4 = ax.bar(x + 1.5*bar_width, df_metrics['F1 Score'], bar_width,
              label='F1 Score', color='purple')
ax.plot(x, df_metrics['Accuracy'], marker='o', color='blue', linestyle='-',
        linewidth=2, markersize=6, label='Accuracy Line')
ax.plot(x, df_metrics['Precision'], marker='o', color='green', linestyle='-',
        linewidth=2, markersize=6, label='Precision Line')
ax.plot(x, df_metrics['Recall'], marker='o', color='orange', linestyle='-',
        linewidth=2, markersize=6, label='Recall Line')
ax.plot(x, df_metrics['F1 Score'], marker='o', color='purple', linestyle='-',
        linewidth=2, markersize=6, label='F1 Score Line')
ax.set_xlabel('Model')
ax.set_ylabel('Score')
ax.set_title('Model Performance Comparison with Bars and Lines')
ax.set_xticks(x)
ax.set_xticklabels(df_metrics['Model'], rotation=45, ha='right')
ax.set_ylim(0.6, 1.05)  # Adjusted the upper limit to 1.05 for some space on top
def add_values(bars):
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, height + 0.02, f'{height:.2f}',
                ha='center', va='bottom', fontsize=10)
add_values(bar1)
add_values(bar2)
add_values(bar3)
add_values(bar4)
ax.legend(title='Metrics')
plt.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9, hspace=0.5)  # Adjust 'h
plt.tight_layout()
plt.show()
```

Figure 29: Compared visualization of experimented models

3. Visualizing the ROC curve of every model for easy comparison

In [102]:
```
plt.figure(figsize=(10, 8))

plt.plot(fpr_rf, tpr_rf, color='blue', lw=1,
         label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_xgb, tpr_xgb, color='green', lw=1,
         label=f'XGBoost (AUC = {roc_auc_xgb:.2f})')
plt.plot(fpr_rf_out, tpr_rf_out, color='red', lw=1,
         label=f'RF (Outliers Removed) (AUC = {roc_auc_rf_out:.2f})')
plt.plot(fpr_rf_smote, tpr_rf_smote, color='purple', lw=1,
         label=f'RF (SMOTE) (AUC = {roc_auc_rf_smote:.2f})')
plt.plot(fpr_xgb_smote, tpr_xgb_smote, color='brown', lw=1,
         label=f'XGB (SMOTE) (AUC = {roc_auc_xgb_smote:.2f})')
plt.plot(fpr_rf_pca, tpr_rf_pca, color='pink', lw=1,
         label=f'RF (PCA) (AUC = {roc_auc_rf_pca:.2f})')
plt.plot(fpr_xgb_pca, tpr_xgb_pca, color='cyan', lw=1,
         label=f'XGB (PCA) (AUC = {roc_auc_xgb_pca:.2f})')
plt.plot(fpr_rf_tuned, tpr_rf_tuned, color='violet', lw=1,
         label=f'RF (Tuned) (AUC = {roc_auc_rf_tuned:.2f})')
plt.plot(fpr_xgb_tuned, tpr_xgb_tuned, color='magenta', lw=1,
         label=f'XGB (Tuned) (AUC = {roc_auc_xgb_tuned:.2f})')

# Plot the diagonal line (Random classifier)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)

# Set axis limits to zoom into the top-left corner
plt.xlim([0.0, 0.2])
plt.ylim([0.8, 1.0])

# Labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve Comparison')

# Legend
plt.legend(loc='lower right')

# Show the plot
plt.grid(True)
plt.show()
```

Figure 30: ROC curve of all models

4. Find the best model and perform Cross Validation and make a chart for check over fitting or under fitting in the best model.

In [91]:
```
# Cross-Validation
print("\nPerforming Cross-Validation...")
cv_scores = cross_val_score(stacked_model, X_train_scaled, y_train, cv=5,
                            scoring='accuracy')
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean()}")
```

```
Performing Cross-Validation...
Cross-Validation Scores: [0.99534451 0.99534451 0.99487896 0.99534234 0.99394504]
Mean CV Accuracy: 0.9949710695882436
```

In [92]:
```
# Learning Curve
train_sizes, train_scores, test_scores = learning_curve(stacked_model, X_train_scaled,
                                                        y_train, cv=5, scoring='accuracy',
                                                        n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
```

In [93]:
```
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, 'o-', label='Training Score')
plt.plot(train_sizes, test_mean, 'o-', label='Validation Score')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2)
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2)
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve')
plt.legend(loc='best')
plt.show()
```

Figure 31: Cross Validation of best model

4.8. Saving the Best model and preprocessing

The model and preprocessing were saved using joblib library.

```
In [98]: joblib.dump(stacked_model, 'stacked_model.pkl')

Out[98]: ['stacked_model.pkl']

In [99]: joblib.dump(scaler, 'scaler.pkl')

Out[99]: ['scaler.pkl']
```

Figure 32: Saving The best model and preprocessing

4.9. Model Deployment

The best model was deployed using streamlit.

```python
import streamlit as st
import numpy as np
import joblib

# Load the pre-trained models
scaler = joblib.load("scaler.pkl")
model = joblib.load("stacked_model.pkl")

# Mapping inputs to numerical values
business_travel_map = {'Travel_Rarely': 0, 'Travel_Frequently': 1, 'Non-Travel': 2}
department_map = {'Corporate Functions': 0, 'Marketing': 1, 'Delivery': 2, 'Product': 3, 'Sales': 4, 'HR': 5}
education_field_map = {'Doctorate': 0, 'Diploma': 1, 'Masters': 2, 'Bachelors': 3}
gender_map = {'Male': 0, 'Female': 1}
marital_status_map = {'Married': 0, 'Divorced': 1, 'Single': 2}
over_time_map = {'Yes': 1, 'No': 0}
office_code_map = {'BOS': 0, 'NYC': 1, 'OTT': 2, 'CAL': 3, 'PHL': 4, 'MKM': 5, 'VAN': 6, 'TOR': 7}
job_level_updated_map = {'L7': 7, 'L6': 6, 'L5': 5, 'L3': 3, 'L2': 2, 'L4': 4, 'L1': 1}

# Function to map categorical inputs to numerical values
def map_categorical_input(input_value, input_mapping):
    return input_mapping.get(input_value, None)

# Function to prepare input data
def prepare_input_data(inputs):
    # Convert inputs to a list of numerical values using maps for categorical features
    input_data = np.array([
        inputs['Joining_Year'],
        inputs['age'],
        inputs['daily_rate'],
        inputs['distance_from_home'],
        map_categorical_input(inputs['education_field'], education_field_map),
        map_categorical_input(inputs['gender'], gender_map),
        map_categorical_input(inputs['marital_status'], marital_status_map),
        map_categorical_input(inputs['over_time'], over_time_map),
        map_categorical_input(inputs['business_travel'], business_travel_map),
        map_categorical_input(inputs['department'], department_map),
        inputs['employee_number'],
        inputs['environment_satisfaction'],
        inputs['HourlyRate'],
        inputs['job_involvement'],
        inputs['job_satisfaction'],
        inputs['monthly_income'],
        inputs['monthly_rate'],
        inputs['num_companies_worked'],
        inputs['percent_salary_hike'],
        inputs['performance_rating'],
        inputs['relationship_satisfaction'],
        inputs['stock_option_level'],
        inputs['total_working_years'],
        inputs['training_times_lastYear'],
        inputs['work_life_balance'],
        inputs['years_at_company'],
        inputs['years_in_current_role'],
        inputs['years_since_last_promotion'],
        inputs['years_with_curr_manager'],
        map_categorical_input(inputs['office_code'], office_code_map),
        map_categorical_input(inputs['job_level_updated'], job_level_updated_map)
    ]).reshape(1, -1)

    return input_data

# Function to scale and apply PCA transformation to input data
def transform_input_data(input_data):
    input_data_scaled = scaler.transform(input_data)
    return input_data_scaled

# Function to predict attrition
def predict_attrition(input_data_scaled):
    prediction = model.predict(input_data_scaled)
    return prediction[0]
```

Figure 33: Streamlit app prediction

```python
# Streamlit app layout
st.title("Employee Attrition Prediction")

# Collect input data from the user
inputs = {
    'Joining_Year': st.number_input('JoiningYear', min_value=2005, max_value=2021, value=2015),
    'age': st.slider('Age', min_value=18, max_value=60, value=30),
    'business_travel': st.selectbox('BusinessTravel', ['Travel_Rarely', 'Travel_Frequently', 'Non-Travel']),
    'daily_rate': st.number_input('DailyRate', min_value=102, max_value=1499, value=800),
    'department': st.selectbox('Department',
                               ['Corporate Functions', 'Marketing', 'Delivery', 'Product', 'Sales', 'HR']),
    'distance_from_home': st.number_input('DistanceFromHome', min_value=1, max_value=29, value=15),
    'education_field': st.selectbox('Education Field', ['Doctorate', 'Diploma', 'Masters', 'Bachelors']),
    'employee_number': st.number_input('EmployeeNumber', min_value=1, max_value=2068, value=1500),
    'environment_satisfaction': st.number_input('EnvironmentSatisfaction', min_value=1, max_value=4, value=2),
    'gender': st.selectbox('Gender', ['Male', 'Female']),
    'HourlyRate': st.number_input('HourlyRate', min_value=30, max_value=100, value=60),
    'job_involvement': st.number_input('JobInvolvement', min_value=1, max_value=4, value=3),
    'job_satisfaction': st.number_input('JobSatisfaction', min_value=1, max_value=4, value=3),
    'marital_status': st.selectbox('Marital Status', ['Married', 'Divorced', 'Single']),
    'monthly_income': st.number_input('MonthlyIncome', min_value=1009, max_value=19999, value=15000),
    'monthly_rate': st.number_input('MonthlyRate', min_value=2094, max_value=26999, value=20000),
    'num_companies_worked': st.number_input('NumCompaniesWorked', min_value=0, max_value=9, value=5),
    'over_time': st.selectbox('Overtime', ['Yes', 'No']),
    'percent_salary_hike': st.number_input('PercentSalaryHike', min_value=11, max_value=25, value=15),
    'performance_rating': st.number_input('PerformanceRating', min_value=1, max_value=5, value=3),
    'relationship_satisfaction': st.number_input('RelationshipSatisfaction', min_value=1, max_value=4, value=3
    'stock_option_level': st.number_input('StockOptionLevel', min_value=0, max_value=3, value=2),
    'total_working_years': st.number_input('TotalWorkingYears', min_value=0, max_value=40, value=20),
    'training_times_lastYear': st.number_input('TrainingTimesLastYear', min_value=0, max_value=6, value=4),
    'work_life_balance': st.number_input('WorkLifeBalance', min_value=1, max_value=4, value=3),
    'years_at_company': st.number_input('YearsAtCompany', min_value=1, max_value=15, value=3),
    'years_in_current_role': st.number_input('YearsInCurrentRole', min_value=1, max_value=7, value=5),
    'years_since_last_promotion': st.number_input('YearsSinceLastPromotion', min_value=1, max_value=6, value=5
    'years_with_curr_manager': st.number_input('YearsWithCurrManager', min_value=1, max_value=6, value=4),
    'office_code': st.selectbox('office_code', ['BOS', 'NYC', 'OTT', 'CAL', 'PHL', 'MKM', 'VAN', 'TOR']),
    'job_level_updated': st.selectbox('JobLevel_updated', ['L7', 'L6', 'L5', 'L3', 'L2', 'L4', 'L1'])
}

# Add Predict button
if st.button('Predict Attrition'):
    try:
        # Prepare the input data
        input_data = prepare_input_data(inputs)

        # Scale and apply PCA transformation
        input_data_pca = transform_input_data(input_data)

        # Make the prediction
        prediction = predict_attrition(input_data_pca)

        # Display the prediction result
        if prediction == 0:
            st.write("Prediction: Employee will attrite (Attrition = 1).")
        else:
            st.write("Prediction: Employee will not attrite (Attrition = 0).")
    except Exception as e:
        st.error(f"Error in prediction: {e}")
```

Figure 34: Streamlit app layout

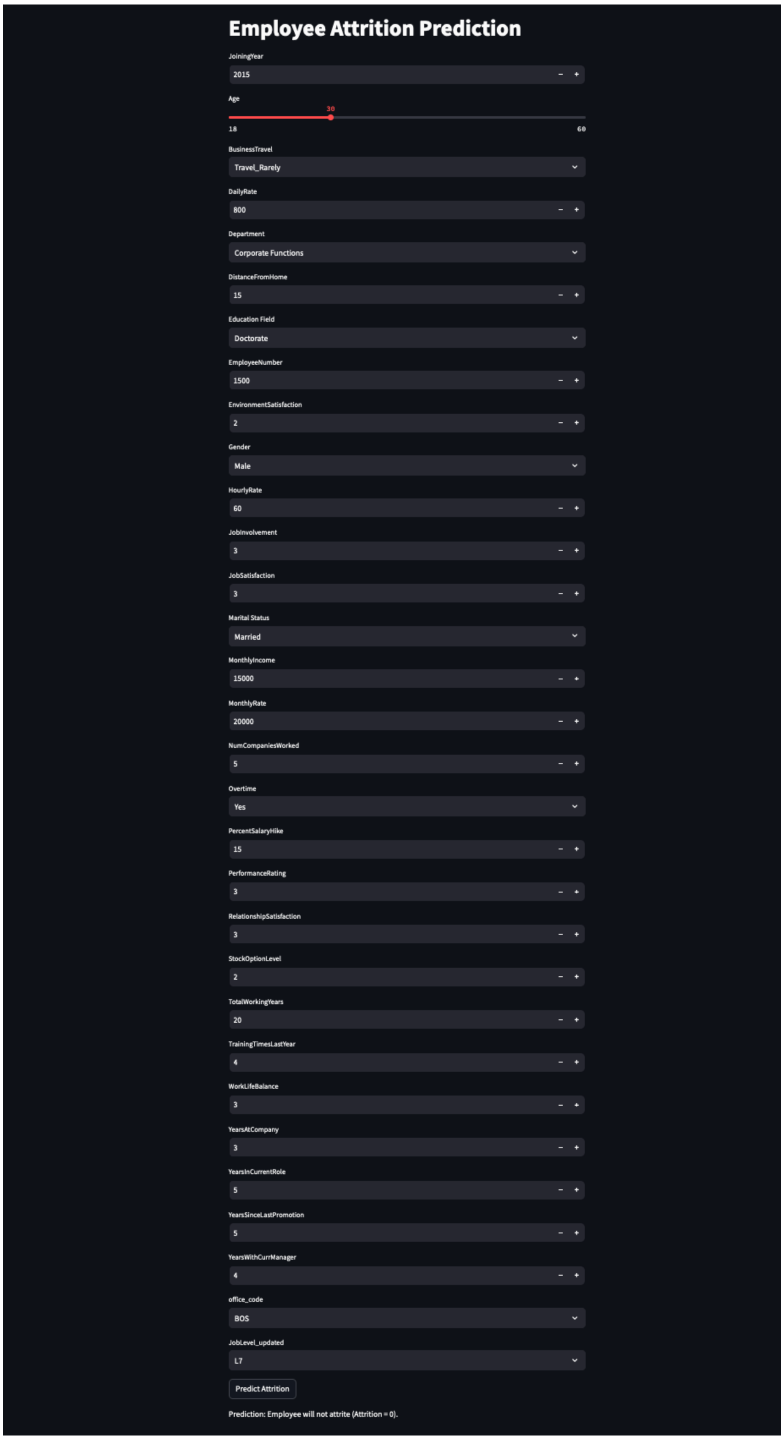


Figure 34: Streamlit app front-end

References

[1] “Anaconda Navigator Distribution,” *Anaconda Distribution*, May 06, 2022. https://www.anaconda.com/products/individual