Configuration Manual

MSc Research Project
AI for Business

Ismail Inayat
Student ID: 24144428

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name: ……Ismail Inayat……

Student ID: ……24144428……

Programme: ……MSc AI for Business…… **Year:** ……2025……

Module: …… MSc (Research) Practicum Part 2 ……

Supervisor: …… Anderson Simiscuka……

Submission Due Date: ……11 December 2025……

Project Title: …… Analysis of Employee Turnover factors using AI models……

Word Count: …2039…… **Page Count**……16……

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: ……Ismail Inayat……

Date: ……27 Jan 2026……

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

Analysis of Employee Turnover factors using AI models

Ismail Inayat
X24144428

1. Introduction

The tools, software, and techniques used to complete the investigation will be discussed in the sections that follow in this setup manual. The design describes the resources and the environment needed to construct the models using machine learning that are utilized in this study, alongside how these models have been trained and tested. This work performs exploratory analysis, preprocessing, models training, assessment and turnover factors interpretation. There are two machine learning models such as K-nearest neighbors and decision tree. Similarly, two models of deep learning are trained such as multilayer perceptron's and Tabnet. This work used the HR Analytics dataset, which is publicly available. Four models are tested such as Ensemble decision tree, tuned KNN, deep scaled MLP and TabNet. The ensemble decision tree obtained 85.05% accuracy and tuned KNN achieved 86.39% above than all existing KNN models. Subsequently, deep scaled MLP behave sustainable and achieved 96.68% indicating deep learning strength for extracting representations without manual feature extraction. The TabNet integrated attention mechanisms with feature selection using deep learning architecture. It achieved 99.57% accuracy highest among all available TabNet algorithms. The highest performance explores attention based architecture suited for organized tabular data. Overall results confirmed deep learning outperformed machine learning algorithms with IBM HR dataset. Whereas, machine learning models are less explainable and simple. The deep learning models with improved insight features and considered suitable with HR decision making.

2. System Configuration

The hardware details and software details applied in this experiments are as listed below:

2.1. Hardware Configuration

• Operating system: Windows 10

• Processor: Core i7 10th generation

• System Type: 64-bit operating system

• Hard Disk: 256GB SSD

• Installed physical memory RAM: 8GB

2.2. Software Configurations

All experiments conducted on Jupyter Notebook using Kaggle. Kaggle is an online platform for Python and Jupyter Notebook. Kaggle provides 12Gb of RAM and GPU availability. Kaggle is easy to use and flexible for coding purposes. Kaggle has already contains mostly packages. The required packages for this work is showing below.

```
# Install required packages
!pip install pytorch-tabnet --quiet
!pip install shap --quiet
```

Figure 1: Necessary packages

2.3. Dataset

The dataset is selected such as IBM HR Analytics dataset, which is publicly available. This dataset is commonly used in various current studies (Fallucchi *et al.*, 2020; Qutub *et al.*, 2021; Raza *et al.*, 2022; and Arqawi *et al.*, 2022). This dataset is considered ideal and balanced for employee turnover prediction. The dataset is comprehensive. This research uses a dataset provided Kaggle.com, with 35 columns (Maloku, 2024). The research presented here focuses on major characteristics that determine employee attrition statistics.

3. Configuration and Execution

3.1. Importing Libraries

The first step included to select and import necessary python libraries. The dataset is in CSV format therefore it is required a library that can manage tabular data input / output manipulation as well as data frame operations. A library is required for arrays operations and vector operations. Similarly, for figures and line plotting purpose library is required. The dataset analysis suggested that

statistical plots are also required for this study. For the purpose of dataset splitting into train and test sets library is imported. Dataset having various categorical features therefore a library is required for label encoding. A preprocessing unit is also required for machine and deep learning models. For evaluation roc curve, confusion matric and precision recall curves are selected, and imported relevant libraries. For classification results such as precision, recall and F1-score are required. The selected models for machine learning are KNN and decision tree, and for deep learning are Multi-layer perceptron (MLP) and Tabnet are selected. The libraries for said models are also imported.

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

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc,
    precision_recall_curve, classification_report, accuracy_score
)
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import permutation_importance

# TabNet + PyTorch
import torch
from pytorch_tabnet.tab_model import TabNetClassifier


import shap
```

Figure 2: Required libraries

To prepare dataset for training preprocessing is mandatory. Since most of attributes are categorical and have to convert into numbers. Label encoding applied for categorical columns into numbers. In next step data is splitted into training and testing sets. Training set is 80% and testing set is 20%. Next step is to standardization of features or scaling of features. This helped models for better performance. Since models perform better while features are scaled on similar standardization.

3.2. Data preprocessing

To prepare dataset for training preprocessing is mandatory. Since most of attributes are categorical and have to convert into numbers. Label encoding applied for categorical columns into numbers.

In next step data is splitted into training and testing sets. Training set is 80% and testing set is 20%. Next step is to standardization of features or scaling of features. This helped models for better performance. Since models perform better while features are scaled on similar standardization.

```
#  Preprocessing
# Encode categorical columns
le = LabelEncoder()
for col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col].astype(str))

# Features
X = df.drop('Attrition', axis=1)
y = df['Attrition'].astype(int)   # ensure 0/1 ints, not bools

# Train-test split (stratified)
X_train_df, X_test_df, y_train_series, y_test_series = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale numeric features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train_df)
X_test = scaler.transform(X_test_df)

# Convert to numpy dtypes
X_train = np.array(X_train, dtype=np.float32)
X_test = np.array(X_test, dtype=np.float32)
y_train = np.array(y_train_series, dtype=np.int64)
y_test = np.array(y_test_series, dtype=np.int64)

print("Shapes / dtypes:")
print("X_train:", X_train.shape, X_train.dtype)
print("y_train unique:", np.unique(y_train), y_train.dtype)
```

Figure 3: Code for preprocessing

3.3. Exploratory Data Analysis

In next step exploratory data analysis is performed for visualization of data provide better understanding before model training. Bar chart applied to show employee left or stay. Total left and stayed employees are counted. This process is performed for observance of balanced data or imbalanced data. A data frame is made at this level. It contained numeric values or features. Next heat map drew to differentiate numeric features. Red are showing strong while blue is showing weak correlations.

```
# Exploratory plots
plt.figure(figsize=(6,4))
pd.Series(y).value_counts().plot(kind='bar')
plt.title('Attrition distribution (0=Stay, 1=Leave)')
plt.show()

num_df = pd.DataFrame(X, columns=df.drop('Attrition',
axis=1).columns).select_dtypes(include=[np.number])
if num_df.shape[1] >= 2:
    plt.figure(figsize=(10,6))
    sns.heatmap(num_df.corr(), cmap='coolwarm', center=0)
    plt.title('Correlation heatmap')
    plt.show()
```

Figure 4: Exploratory analysis

3.4. Helper function

A helper function is designed to evaluate machine and deep learning models. It represented visual performance for accuracy, classification, ROC curve, confusion matrix and precision recall curve. An evaluation function eval is created. Its parameters included model name, testing true labels, models predicted labels and predicted probabilities for ROC and precision recall curves. Similarly, models accuracies and classification reports for each model. The report included each model accuracy, F1-score, precision, recall and support.

```
# Helper functions: evaluation & plotting
def eval_and_plot(name, y_true, y_pred, y_proba=None):
    print(f"--- {name} ---")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print(classification_report(y_true, y_pred, digits=4))

    # Confusion matrix
    ConfusionMatrixDisplay(confusion_matrix(y_true, y_pred)).plot(cmap='Blues')
    plt.title(f'{name} - Confusion Matrix')
    plt.show()

    # ROC & PR
    if y_proba is not None:
        fpr, tpr, _ = roc_curve(y_true, y_proba)
        roc_auc = auc(fpr, tpr)
        plt.figure()
        plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.3f}')
        plt.plot([0,1],[0,1],'--', color='gray')
        plt.title(f'{name} - ROC Curve')
        plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate')
        plt.legend(); plt.show()

        precision, recall, _ = precision_recall_curve(y_true, y_proba)
        plt.figure()
        plt.plot(recall, precision)
        plt.title(f'{name} - Precision-Recall')
        plt.xlabel('Recall'); plt.ylabel('Precision'); plt.show()
```

Figure 5: Helper function

3.5. Ensemble Decision tree

Decision tree is defined and configured for training. Decision tree splitted data in branches according to features. It works as decision process. It is simple and explainable. Casual decision tree is not properly fitted with dataset and producing low performance. Therefore, an ensemble tree classifier is designed using Random Forest, Extra trees, Gradient Boosting, Ada boosting and XGBoost . It is named as pickle-safe model. This algorithm is designed to achieve highest accuracy specifically for employee turnover prediction.

The classifiers are combined using voting. The results are evaluated and confusion matrix, ROC curve and precision recall curves are plotted. Dataset is splitted into 80% and 20% ratios for training and testing purpose respectively. Stratify is used to keep ratio 1 for left and 0 for stay. Reproducible splitted is ensured with random state configuration. A 400 decision trees were built with random subsets from features. Class imbalance is handled with class weight function. An extra classifier is developed for extra randomness for splitting tree.

Adaboost classifier is applied for boosting weak learners towards strong model. XGBoost is powerful boosting model having speed along with accuracy optimization. For better generalization randomness added using colsample and subsample. Multithreading is disable for ensuring for saving model safely. Each algorithm predicted probability for classes. Soft voting combined the average probability of model with steady and robust performance instead single decision tree. All algorithms and parameters serialized saved avoiding issue of parallel processing. All classifiers are combined and trained. Predicted labels and predicted probability are calculated. Accuracy, ROC-AUC calculated. At the end confusion matrix and ROC curve is plotted.

```
from sklearn.ensemble import (
    RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier,
AdaBoostClassifier, VotingClassifier
)
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix,
roc_curve, ConfusionMatrixDisplay

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

#  Define tree models (all picklable)
rf = RandomForestClassifier(n_estimators=400, max_depth=None,
class_weight='balanced', random_state=42, n_jobs=1)
et = ExtraTreesClassifier(n_estimators=400, max_depth=None, random_state=42,
n_jobs=1)
gb = GradientBoostingClassifier(n_estimators=400, learning_rate=0.05,
max_depth=5, random_state=42)
ada = AdaBoostClassifier(n_estimators=300, learning_rate=0.05, random_state=42)
xgb = XGBClassifier(n_estimators=700, learning_rate=0.05, max_depth=8,
subsample=0.9, colsample_bytree=0.9,
                    random_state=42, n_jobs=1, use_label_encoder=False,
eval_metric='logloss')
lgb = LGBMClassifier(n_estimators=700, learning_rate=0.05, num_leaves=50,
subsample=0.9, colsample_bytree=0.9,
                     class_weight='balanced', random_state=42, n_jobs=1)

#  Voting Ensemble (soft voting, picklable)
ensemble = VotingClassifier(
    estimators=[('rf', rf), ('et', et), ('gb', gb), ('ada', ada), ('xgb', xgb),
('lgb', lgb)],
    voting='soft',
    n_jobs=1  # important: disable parallelization to avoid pickle errors
)

# Train ensemble
ensemble.fit(X_train, y_train)

#  Predict
y_pred = ensemble.predict(X_test)
y_proba = ensemble.predict_proba(X_test)[:, 1]
```

```
# Evaluate
acc = accuracy_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_proba)
print(f"?  Pickle-Safe Ensemble Accuracy: {acc*100:.2f}%")
print(f"?  ROC-AUC: {roc:.3f}")

#  Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm).plot(cmap='Greens', values_format='d')
plt.title("Confusion Matrix – Pickle-Safe Tree Ensemble")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label=f"AUC = {roc:.3f}")
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve – Pickle-Safe Tree Ensemble")
plt.legend(); plt.grid(True); plt.show()
```

Figure 6: Code for ensemble decision tree

3.6. Tuned KNN

KNN is defined and configured for training. KNN is good for five nearest neighbors for better prediction. KNN requires scaled data and feasible for small data. In this case traditional KNN showed limited compatibility with HR dataset. Therefore, KNN needs improvements. The optimized and fine-tuned KNN is designed and implemented for this case. This algorithm is scaled, tuned and bagging to achieve high accuracy. A pipeline of scalar and KNN has designed. Scalar standardized numeric values and KKN utilized classifier. The data automatically scaled whenever training and testing performed. A grid of hyperparameters have been set. A grid search set up for testing uniform nearest neighbors. The closer neighbor considered more influence. Two types of distance Euclidean and Manhattan are applied. A cosine metric is set to compare similarity among directions instead distances only.

A grid based searching is designed to achieve best setting for KNN. This grid search trained several versions for pipeline utilizing various combination of hyperparameters. A reliable evaluation applied using cross validation by 34 fold. A best model is retrieved for KNN as fine-tuned and optimized. A bagging classifier is used to wraps best KNN using ensemble having 249 various slightly models. The training sample of 90% seen by KNN. Each KNN utilized 90% features. All these steps reduced overfitting as well as generalization improvement. Next KNN training is performed with bagged ensemble with training set for predicted classes and positive classes probabilities. Accuracy for training data evaluated to check overfitting problems. Last but not least evaluations and plots such as confusion matrix, precision recall and ROC curve produced.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

param_grid = {
    'knn__n_neighbors': [3, 5, 7, 9, 11, 13, 15,17],
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2],
    'knn__metric': ['cosine'],
}

grid = GridSearchCV(
    pipe,
    param_grid,
    scoring='accuracy',
    cv=34,
    n_jobs=1,   # ?  avoid pickling errors
    verbose=1
)
grid.fit(X_train, y_train)
print("Best KNN Params:", grid.best_params_)

best_knn = grid.best_estimator_
bag_knn = BaggingClassifier(
    estimator=best_knn,
    n_estimators=249,
    max_samples=0.9,
    max_features=0.9,
    random_state=42,
    n_jobs=1   # ?  avoid pickling
)

bag_knn.fit(X_train, y_train)
y_pred_knn = bag_knn.predict(X_test)
y_proba_knn = bag_knn.predict_proba(X_test)[:, 1]

eval_and_plot("Optimized KNN (Safe + Tuned)", y_test, y_pred_knn, y_proba_knn)
```

```python
train_acc = accuracy_score(y_train, bag_knn.predict(X_train))
print(f"Training Accuracy: {train_acc:.4f}")
```

Figure 7: Code for tuned KNN

3.7. Deep Scaled Multilayer perceptron

Multilayer perceptron (MLP) is a neural network based model. Next MLP is defined and configured for training. It is suggested for complex patterns. Traditional MLP has various limitations while achieving highest accuracy with HR dataset. Therefore, an enhanced deep MLP is designed. Numerical values are normalized with scalar function for input features. MLP performs well when data is normalized. Similarly, scaling confirms equally contribution of all features. Deep MLP is configured with 2 hidden layers having 2048 and 1024 neurons to design deep network. Efficient deep learning activation function ReLU is applied. For adaptive learning adam optimization algorithm included.

Overfitting is prevented through L2 regularization with $1e^{-3}$. Learning rate set constant for fixed training. For stable and slow learning initial learning is set to $3e^{-1}$. A large number of epochs for training are adapted. Early stoppage of training applied while validation stops improving. It is set to 20 as consecutive improvements are not achieved. Reproducibility is ensured. Mini batches sample of 12 is adjusted. Training data of 10% is reserved for early collapse. Neural network is trained on scaled data. Internal biases when minimized classification loss. Prediction and probability is collected. Evaluation is performed using accuracies, confusion matrix, ROC curve and precision recall curve.

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

# Scale input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

mlp = MLPClassifier(
    hidden_layer_sizes=(2048),
    activation='relu',
    solver='adam',
    alpha=1e-3,
    learning_rate='constant',
    learning_rate_init=3e-1,
    max_iter=1200,
    early_stopping=True,
    n_iter_no_change=20,
    random_state=42,
    verbose=False,
    batch_size=12,
    validation_fraction=0.1
)

mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
y_proba_mlp = mlp.predict_proba(X_test)[:,1]
eval_and_plot("Deep MLP (Scaled)", y_test, y_pred_mlp, y_proba_mlp)
train_acc = accuracy_score(y_train, mlp.predict(X_train))
print(f"Final Accuracy: {train_acc:.4f}")
```

Figure 8: Code for deep scaled MLP

3.8. Tabnet

Tabnet is considered as model for tabular data. Tabnet deep learning model is defined and configured. It is better for structured data. For Tabnet GPU is required for faster training. Model's hyper parameters are adjusted as 32 no of decisions and 32 are attention features to controlling mode's capacity. Decision steps are adjusted to 5. Features reusability is adjusted to 0.1. Sparsity is $1e^{-3}$ to avoid overfitting. The adaptive learning is achieved using adam. Learning rate is adaptive and set to 0.2. for feature selection procedure sparsemax is choose, it chooses only relevant features. Learning rate schedule is set to reduce it for every epoch. LR is reduced with multiplying gamma after step epochs.

Reproducibility is ensured by random seed. After adjustment of hyperparameters model training is started. Training performance as well as validation performance is evaluated after every epoch. Training process stopped if improvement is not increase after 50 epochs. A small and stable batch size are chosen. Final batch is kept even it is smaller. Overfitting and under fitting is observed. Prediction and probabilities are observed. Probability for employee leaving class is extracted. Model is evaluated by calculating accuracy and plotting confusion matrix, precision recall curve and ROC curve.

```
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Device:", DEVICE)

tabnet = TabNetClassifier(
    n_d=32, n_a=32, n_steps=5, gamma=0.1,
    lambda_sparse=1e-3,
    optimizer_fn=torch.optim.Adam,
    optimizer_params=dict(lr=2e-1),
    mask_type='sparsemax',
    scheduler_params={"step_size":30, "gamma":0.2},
    scheduler_fn=torch.optim.lr_scheduler.StepLR,
    verbose=1,
    seed=42
)


tabnet.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    eval_name=['train','valid'],
    eval_metric=['accuracy'],
    max_epochs=150,
    patience=40,
    batch_size=32,
    drop_last=False
)
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
plt.plot(tabnet.history['train_accuracy'], label='Train Accuracy')
plt.plot(tabnet.history['valid_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('TabNet Training vs Validation Accuracy')
plt.show()
y_pred_tab = tabnet.predict(X_test)
y_proba_tab = tabnet.predict_proba(X_test)[:,1]
eval_and_plot("TabNet", y_test, y_pred_tab, y_proba_tab)


last_train_acc = tabnet.history['train_accuracy'][-1]


print(f" Final Accuracy: {last_train_acc:.4f}")
```

Figure 9: Code for proposed TabNet

3.9. Predicting top 10 turnover features

After implemented four proposed models employee turnover prediction is made. Top ten turnover features are predicted for each model. First feature importance for TabNet is collected and bar chart is plotted. Next top ten turnover features for Ensemble decision tree are selected. Turnover predicted features are collecting from voting classifier as all tree algorithms are ensemble in voting classifier. The predicted features are collected and bar chart is plotted. The predicted features for KNN are collected through permutation importance for estimation since KKN do not have feature importance as built-in. The predicted features are collected and bar chart is plotted. Finally, top ten predicted features of deep ensemble MLP are collected. A SHAP is used for kernel explainer. A mean SHAP is computed for features. The predicted features are collected and bar chart is plotted.

```
import shap
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

feat_names = df.drop('Attrition', axis=1).columns

# TabNet
try:
    tabnet_importances = tabnet.feature_importances_
    idx = np.argsort(tabnet_importances)[::-1][:10]
    top_tab = [(feat_names[i], tabnet_importances[i]) for i in idx]
    print("Top 10 TabNet Features:")
    for f, v in top_tab:
        print(f"{f:<25}: {v:.4f}")
    plt.figure(figsize=(8,4))
    sns.barplot(x=[v for _,v in top_tab], y=[f for f,_ in top_tab])
    plt.title("TabNet - Top 10 Features"); plt.show()
except Exception as e:
    print("TabNet feature importance not available:", e)


    #  internal trained model
fitted_rf = ensemble.named_estimators_['rf']    # 'rf' is the name you gave in
VotingClassifier

# Get top 10 features
importances = fitted_rf.feature_importances_
feat_names = X.columns
idx = np.argsort(importances)[::-1][:10]
top_rf = [(feat_names[i], importances[i]) for i in idx]

print("Top 10 ENSEMBLE DECISION TREE Features (from ensemble):")
for f, v in top_rf:
    print(f"{f:<25}: {v:.4f}")

plt.figure(figsize=(8,4))
sns.barplot(x=[v for _,v in top_rf], y=[f for f,_ in top_rf])
plt.title("ENSEMBLE DECISION TREE - Top 10 Features (from Voting Ensemble)")
plt.show()


# KNN
try:
    # use permutation importance
    from sklearn.inspection import permutation_importance
    result = permutation_importance(bag_knn, X_test, y_test, n_repeats=5,
random_state=42)
    idx = np.argsort(result.importances_mean)[::-1][:10]
    top_knn = [(feat_names[i], result.importances_mean[i]) for i in idx]
    print("\n Top 10 KNN Features (via Permutation Importance):")
    for f, v in top_knn:
        print(f"{f:<25}: {v:.4f}")
    plt.figure(figsize=(8,4))
    sns.barplot(x=[v for _,v in top_knn], y=[f for f,_ in top_knn])
    plt.title("KNN - Top 10 Features"); plt.show()
except Exception as e:
    print(" KNN feature importance not available:", e)


# MLP
try:
    explainer = shap.KernelExplainer(mlp.predict_proba, X_train[:100])
    shap_values = explainer.shap_values(X_test[:100])
    shap_mean = np.mean(np.abs(shap_values[1]), axis=0)
    idx = np.argsort(shap_mean)[::-1][:10]
    top_mlp = [(feat_names[i], shap_mean[i]) for i in idx]
    print("\nTop 10 MLP Features (via SHAP):")
    for f, v in top_mlp:
        print(f"{f:<25}: {v:.4f}")
    plt.figure(figsize=(8,4))
    sns.barplot(x=[v for _,v in top_mlp], y=[f for f,_ in top_mlp])
    plt.title("MLP - Top 10 Features"); plt.show()
except Exception as e:
    print(" MLP SHAP error:", e)
```

Figure 10: Predicting top 10 turnover features

References

"IBM HR Analytics Dataset," Kaggle. [Online]. Available: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset

Kaggle Note Book. [Online]. Available: https://www.kaggle.com/code/ismailinayat12/analysis-of-employee-turover-using-ai-models/edit