Employee Attrition Prediction Model and Recommendations

1. Import the preprocessed dataset:

```
x=pd.read_csv('scaled_x_data.csv')
y=pd.read_csv('y_data.csv')
```

2. Data splitting:

Given the constraints of the dataset's size, an 90/10 split would be a reasonable choice.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

3. Models selection

We are dealing with binary classification tasks, and the options include:

1. Logistic Regression
2. Random Forest
3. Gradient Boosting
4. Support Vector Machines (SVM)
5. Naive Bayes
6. K-Nearest Neighbors (KNN)
7. Decision Trees
8. Neural Networks, including deep learning models.

Model	characteristics	Decision
Random Forest	can work well with mixed data types (numeric and non-numeric). It can handle non-linear relationships in the data. Random Forest is less prone to overfitting compared to individual decision trees.	Selected
XGBoost	powerful ensemble techniques that can handle non-linearity and mixed data types. Robust and can handle small to medium-sized datasets effectively.	Selected
Logistic Regression	assumes linear relationships between features and the log-odds of the target variable. It may not perform well when dealing with complex, non-linear relationships among the features,	Non selected
Decision Trees	can be prone to overfitting when the dataset is small, and features are numerous, potentially leading to poor generalization.	Non selected
Naive Bayes	A simple probabilistic classifier that assumes feature independence. It may not capture the intricate relationships and dependencies among the features, which can be crucial for accurate classification.	Non selected
K-Nearest Neighbors	KNN relies on distances between data points, and it can struggle with high-dimensional datasets.	Non selected
Neural Networks	are not suitable for this case due to the small dataset size, which can lead to overfitting.	Non selected

4. Models training

```
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb


# Create a Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
rf_model.fit(x_train, y_train)


# Create an XGBoost classifier
xgb_model = xgb.XGBClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
xgb_model.fit(x_train, y_train)
```

5. Models evaluation

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
```

```

# Predictions from Random Forest model
rf_predictions = rf_model.predict(x_test)

# Calculate evaluation metrics for Random Forest
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)
rf_roc_auc = roc_auc_score(y_test, rf_predictions)
rf_confusion = confusion_matrix(y_test, rf_predictions)

print("Random Forest Metrics:")
print(f"Accuracy: {rf_accuracy}")
print(f"Precision: {rf_precision}")
print(f"Recall: {rf_recall}")
print(f"F1-Score: {rf_f1}")
print(f"ROC AUC: {rf_roc_auc}")
print(f"Confusion Matrix:\n{rf_confusion}")
```

```
Random Forest Metrics:
Accuracy: 0.8707482993197279
Precision: 0.6
Recall: 0.07692307692307693
F1-Score: 0.13636363636363635
ROC AUC: 0.5345399698340875
Confusion Matrix:
[[253   2]
 [ 36   3]]
```

```python
# Predictions from XGBoost model
xgb_predictions = xgb_model.predict(x_test)

# Calculate evaluation metrics for XGBoost
xgb_accuracy = accuracy_score(y_test, xgb_predictions)
xgb_precision = precision_score(y_test, xgb_predictions)
xgb_recall = recall_score(y_test, xgb_predictions)
xgb_f1 = f1_score(y_test, xgb_predictions)
xgb_roc_auc = roc_auc_score(y_test, xgb_predictions)
xgb_confusion = confusion_matrix(y_test, xgb_predictions)

print("\nXGBoost Metrics:")
print(f"Accuracy: {xgb_accuracy}")
print(f"Precision: {xgb_precision}")
print(f"Recall: {xgb_recall}")
print(f"F1-Score: {xgb_f1}")
print(f"ROC AUC: {xgb_roc_auc}")
print(f"Confusion Matrix:\n{xgb_confusion}")
```

```
XGBoost Metrics:
Accuracy: 0.8707482993197279
Precision: 0.5217391304347826
Recall: 0.3076923076923077
F1-Score: 0.3870967741935484
ROC AUC: 0.6322775263951734
Confusion Matrix:
[[244  11]
 [ 27  12]]
```

both the Random Forest and XGBoost models have a high accuracy, but they exhibit trade-offs between precision and recall. The Random Forest has higher precision but much lower recall, while XGBoost offers a better balance between precision and recall.

6. Hyperparameters tuning

6.1.Selecting the priority metric for hyperparameter tuning

The primary goal of the attrition prediction model is to identify employees at risk of. High sensitivity(recall) ensures that the model can correctly identify as many employees at risk as possible, minimizing false negatives. In this context, false negatives (not identifying employees who are actually at risk) can be costly to the company.
Prioritizing recall helps in addressing this issue effectively.

6.2.Codes and results

```
from sklearn.model_selection import GridSearchCV

# Define a grid of hyperparameters to search
param_grid_rf = {
    'n_estimators': [1, 3, 5, 10, 20, 30],  # Vary the number of trees
    'max_depth': [None, 5, 10, 20],   # Vary the maximum depth of trees
    'min_samples_split': [1, 2, 5, 10],  # Vary the minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4]  # Vary the minimum samples required for a leaf node
}

# Perform grid search with cross-validation
grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5, scoring='recall', n_jobs=-1)

# Fit the grid search to the data
grid_search_rf.fit(x_train, y_train)

# Get the best hyperparameters
best_params_rf = grid_search_rf.best_params_
best_rf_model = grid_search_rf.best_estimator_

# Train the best model on the training data
best_rf_model.fit(x_train, y_train)

# Make predictions and evaluate the best model
rf_predictions_tuned = best_rf_model.predict(x_test)
```

```
# Predictions from tuned Random Forest model
rf_predictions_tuned = best_rf_model.predict(x_test)

# Calculate evaluation metrics for tuned Random Forest
rf_accuracy_tuned = accuracy_score(y_test, rf_predictions_tuned)
rf_precision_tuned = precision_score(y_test, rf_predictions_tuned)
rf_recall_tuned = recall_score(y_test, rf_predictions_tuned)
rf_f1_tuned = f1_score(y_test, rf_predictions_tuned)
rf_roc_auc_tuned = roc_auc_score(y_test, rf_predictions_tuned)
rf_confusion_tuned = confusion_matrix(y_test, rf_predictions_tuned)

# Print or use the metrics as needed
print(f" best_params_rf: {best_params_rf} ")
print("Tuned Random Forest Metrics:")
print(f"Accuracy: {rf_accuracy_tuned}")
print(f"Precision: {rf_precision_tuned}")
print(f"Recall: {rf_recall_tuned}")
print(f"F1-Score: {rf_f1_tuned}")
print(f"ROC AUC: {rf_roc_auc_tuned}")
print(f"Confusion Matrix:\n{rf_confusion_tuned}")
```

```
 best_params_rf: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 3}
Tuned Random Forest Metrics:
Accuracy: 0.826530612244898
Precision: 0.2
Recall: 0.10256410256410256
F1-Score: 0.13559322033898302
ROC AUC: 0.5199095022624435
Confusion Matrix:
[[239  16]
 [ 35   4]]
```

```python
# Define a grid of hyperparameters for XGBoost
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 4, 5,10, 20],
    'learning_rate': [0.01, 0.1, 0.2],
    'min_child_weight': [1, 2, 3]
}


# Perform grid search with cross-validation
grid_search_xgb = GridSearchCV(xgb_model, param_grid_xgb, cv=5, scoring='recall', n_jobs=-1)

# Fit the grid search to the data
grid_search_xgb.fit(x_train, y_train)

# Get the best hyperparameters
best_params_xgb = grid_search_xgb.best_params_
best_xgb_model = grid_search_xgb.best_estimator_

# Train the best model on the training data
best_xgb_model.fit(x_train, y_train)

# Make predictions and evaluate the best model
xgb_predictions_tuned = best_xgb_model.predict(x_test)
```

```python
# Predictions from tuned XGBoost model
xgb_predictions_tuned = best_xgb_model.predict(x_test)

# Calculate evaluation metrics for tuned XGBoost
xgb_accuracy_tuned = accuracy_score(y_test, xgb_predictions_tuned)
xgb_precision_tuned = precision_score(y_test, xgb_predictions_tuned)
xgb_recall_tuned = recall_score(y_test, xgb_predictions_tuned)
xgb_f1_tuned = f1_score(y_test, xgb_predictions_tuned)
xgb_roc_auc_tuned = roc_auc_score(y_test, xgb_predictions_tuned)
xgb_confusion_tuned = confusion_matrix(y_test, xgb_predictions_tuned)

# Print or use the metrics as needed
print(f" best_params_rf: {best_params_xgb} ")
print("Tuned XGBoost Metrics:")
print(f"Accuracy: {xgb_accuracy_tuned}")
print(f"Precision: {xgb_precision_tuned}")
print(f"Recall: {xgb_recall_tuned}")
print(f"F1-Score: {xgb_f1_tuned}")
print(f"ROC AUC: {xgb_roc_auc_tuned}")
print(f"Confusion Matrix:\n{xgb_confusion_tuned}")
```

```
 best_params_rf: {'learning_rate': 0.2, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 200}
Tuned XGBoost Metrics:
Accuracy: 0.891156462585034
Precision: 0.6521739130434783
Recall: 0.38461538461538464
F1-Score: 0.4838709677419355
ROC AUC: 0.6766214177978883
Confusion Matrix:
[[247   8]
 [ 24  15]]
```

7. Thresholds adjust

Considering that hyperparameter tuning did not significantly improve the model's performance, we recommend adjusting the classification thresholds.

7.1.Adjust thresholds on tuned random forest model

```python
from sklearn.metrics import classification_report
# Function to adjust threshold and calculate sensitivity
def adjust_threshold_and_evaluate(model, x, y, threshold):
    y_prob = model.predict_proba(x)[:, 1]  # Get probability scores for the positive class
    y_pred = (y_prob > threshold).astype(int)  # Adjust threshold for class prediction

    sensitivity = recall_score(y, y_pred)
    return sensitivity, y_pred

# Define a range of threshold values to test
threshold_values = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
sensitivity_scores = []
y_preds = []
# Iterate through different threshold values
for threshold in threshold_values:
    sensitivity, y_pred = adjust_threshold_and_evaluate(best_rf_model, x_test, y_test, threshold)
    sensitivity_scores.append(sensitivity)
    y_preds.append(y_pred)

# Find the threshold that maximizes sensitivity
best_threshold = threshold_values[sensitivity_scores.index(max(sensitivity_scores))]
print(f"Best Threshold for Max Sensitivity: {best_threshold}")

# Calculate sensitivity for the best threshold
best_sensitivity = max(sensitivity_scores)
print(f"Max Sensitivity: {best_sensitivity}")

# Evaluate other metrics for the model with the best threshold
best_y_pred = y_preds[sensitivity_scores.index(max(sensitivity_scores))]
print("Classification Report for Model with Max Sensitivity:")
print(classification_report(y_test, best_y_pred))
```

```
Best Threshold for Max Sensitivity: 0.1
Max Sensitivity: 0.5641025641025641
Classification Report for Model with Max Sensitivity:
              precision    recall  f1-score   support

           0       0.90      0.61      0.73       255
           1       0.18      0.56      0.27        39

    accuracy                           0.60       294
   macro avg       0.54      0.59      0.50       294
weighted avg       0.81      0.60      0.67       294
```

7.2.Adjust thresholds on tuned XGBoost model

```
Best Threshold for Max Sensitivity: 0.1
Max Sensitivity: 0.6666666666666666
Classification Report for Model with Max Sensitivity:
              precision    recall  f1-score   support

           0       0.94      0.79      0.86       255
           1       0.33      0.67      0.44        39

    accuracy                           0.78       294
   macro avg       0.63      0.73      0.65       294
weighted avg       0.86      0.78      0.80       294
```

7.3.Adjust thresholds on non-tuned random forest model

```
Best Threshold for Max Sensitivity: 0.1
Max Sensitivity: 0.8717948717948718
Classification Report for Model with Max Sensitivity:
              precision    recall  f1-score   support

           0       0.96      0.47      0.63       255
           1       0.20      0.87      0.33        39

    accuracy                           0.52       294
   macro avg       0.58      0.67      0.48       294
weighted avg       0.86      0.52      0.59       294
```

7.4.Adjust thresholds on non-tuned XGBoost model

```
Best Threshold for Max Sensitivity: 0.1
Max Sensitivity: 0.5128205128205128
Classification Report for Model with Max Sensitivity:
              precision    recall  f1-score   support

           0       0.92      0.86      0.89       255
           1       0.36      0.51      0.43        39

    accuracy                           0.82       294
   macro avg       0.64      0.69      0.66       294
weighted avg       0.85      0.82      0.83       294
```

7.5.Comparative results table!

case	Model	Accuracy	Precision	Recall	F1
Without hyperparameters tuning	RF	0.87	0.6	0.076	0.13
	XGB	0.87	0.52	0.38	0.38
After hyperparameters tuning	RF	0.82	0.2	0.1	0.13
	XGB	0.89	0.65	0.38	0.48
After adjusting thresholds for tuned models	RF	0.5	0.18	0.79	0.3
	XGB	0.78	0.33	0.66	0.44
After adjusting thresholds for non-tuned models	RF	0.52	0.2	0.87	0.33
	XGB	0.82	0.36	0.51	0.43

Based on the comparative table adjusting the thresholds on 0.1, using non-tuned RF model gave the best results. However, it's important to note that these results still fall short of the desired performance criteria, with an accuracy of 0.52 and a precision of 0.2.

8. Recommendations and actionable insights :

To enhance model performance, several strategies can be implemented. Data augmentation methods should be considered to increase dataset diversity and size. Class imbalance, if present, can be addressed through techniques like oversampling or class weighting. Additionally, the establishment of a continuous data gathering process ensures that the models remain up-to-date and aligned with evolving domain trends.

Beyond the technical aspects, fostering better communication between the Human Resource Manager and employees is essential. This proactive approach can help address issues such as overtime, job satisfaction, commute distance, monthly income, and business travel. Open channels of communication allow for the timely identification of concerns, enabling the HR team to implement effective solutions and foster a more productive and satisfied workforce.