Source Code

```
import numpy as np
import seaborn as sns
import matplotlib as plyplt
```

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

# For modeling
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score,
accuracy_score

# ML models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

# For clustering
from sklearn.cluster import KMeans

# For imbalance handling
from imblearn.over_sampling import SMOTE

# Load data
df = pd.read_csv('/Users/gres1/Downloads/HR_comma_sep.csv')
```

```
df.head
df.tail
```

```
print (df)
```

```
from IPython.display import display

display(df)

display(df.head())

display(df.tail())
```

```
import pandas as pd

# Load the data
df = pd.read_csv('/Users/gres1/Downloads/HR_comma_sep.csv')

# Check for missing values
print("Missing values per column:")
print(df.isnull().sum())

print("\nDataFrame info:")
print(df.info())

print("\nDescriptive statistics:")
print(df.describe())
```

```
mport seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['number'])

# Correlation heatmap on numeric features
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Matrix (Numeric Columns Only)')
plt.show()

# Distribution of the target variable
sns.countplot(x='left', data=df)
plt.title('Employee Turnover Count')
plt.show()

# Boxplots of key numeric features grouped by turnover status
features = ['satisfaction_level', 'last_evaluation', 'number_project',
            'average_montly_hours', 'time_spend_company', 'promotion_last_5years']
```

```
for feat in features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='left', y=feat, data=df)
    plt.title(f'{feat} by Employee Turnover')
    plt.show()
```

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Features and target
X = df.drop('left', axis=1)
y = df['left']

# For simplicity, encode categorical variables first (e.g., 'salary' and 'sales')
X = pd.get_dummies(X, drop_first=True)

# Split into train and test sets before applying SMOTE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Apply SMOTE to training data only
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check the balance after SMOTE
print("Original training target distribution:")
print(y_train.value_counts())

print("\nAfter SMOTE training target distribution:")
print(y_train_smote.value_counts())
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import make_scorer, accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score

import numpy as np
import pandas as pd

# Define models
models = {
```

```
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(probability=True),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

# Use StratifiedKFold for balanced splits
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Custom scoring function dictionary
scoring = {
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),
    'roc_auc': make_scorer(roc_auc_score)
}

results = {}

for name, model in models.items():
    print(f"Training and evaluating {name}...")
    scores = {}
    for score_name in scoring.keys():
        cv_scores = cross_val_score(model, X_train_smote, y_train_smote, cv=skf,
scoring=scoring[score_name])
        scores[score_name] = np.mean(cv_scores)
    results[name] = scores

# Convert results to DataFrame for easier comparison
results_df = pd.DataFrame(results).T
print("\nCross-Validation Performance Metrics:")
print(results_df)




from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score,
RocCurveDisplay

# Train on SMOTE-balanced training set
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_smote, y_train_smote)

# Predict on the original test set
y_pred = rf_model.predict(X_test)
y_proba = rf_model.predict_proba(X_test)[:, 1]
```

```
# Evaluation metrics
print("Classification Report on Test Set:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("ROC AUC Score:", roc_auc_score(y_test, y_proba))

# ROC Curve
RocCurveDisplay.from_estimator(rf_model, X_test, y_test)
plt.title("Random Forest ROC Curve on Test Set")
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

# Filter only employees who left
df_left = df[df['left'] == 1][['satisfaction_level', 'last_evaluation']]

# KMeans Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df_left['cluster'] = kmeans.fit_predict(df_left)

# Plotting the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(
    data=df_left,
    x='satisfaction_level',
    y='last_evaluation',
    hue='cluster',
    palette='Set2',
    s=100,
    alpha=0.8
)
plt.title('Clusters of Employees Who Left')
plt.xlabel('Satisfaction Level')
plt.ylabel('Last Evaluation')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```