**Build Model**

**## Downsample Majority Class**

**# Check the distribution of the target variable**

**target\_distribution = df\_encoded['default'].value\_counts()**

**target\_distribution**

**# Separate the majority and minority classes**

**df\_majority = df\_encoded[df\_encoded['default'] == 0]**

**df\_minority = df\_encoded[df\_encoded['default'] == 1]**

**# Downsample the majority class**

**df\_majority\_downsampled = resample(df\_majority,**

**replace=False,**

**n\_samples=target\_distribution[1],**

**random\_state=42)**

**# Combine the downsampled majority class with the minority class**

**df\_downsampled = pd.concat([df\_majority\_downsampled, df\_minority])**

**# Shuffle the dataset to mix the data points**

**df\_downsampled = df\_downsampled.sample(frac=1, random\_state=42).reset\_index(drop=True)**

**## Split Features + Target**

**# Splitting the features and target variable**

**X = df\_downsampled.drop('default', axis=1)**

**y = df\_downsampled['default']**

**## Split Data into Test/Training Datasets**

**# Splitting the dataset into training and testing sets**

**X\_train\_downsampled, X\_test, y\_train\_downsampled, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**# Reset Indexes**

**X\_train\_downsampled = X\_train\_downsampled.reset\_index(drop=True)**

**X\_test = X\_test.reset\_index(drop=True)**

**y\_train\_downsampled = y\_train\_downsampled.reset\_index(drop=True)**

**y\_test = y\_test.reset\_index(drop=True)**

**### Training Data Exploration - Post Downsampling**

**# Train model Stats**

**print("Number of Features:", X\_train\_downsampled.shape[1])**

**print("Number Continuous Features:", X\_train\_downsampled.shape[1] - len(cat\_cols))**

**print("Number Categorical Features:", len(cat\_cols))**

**print("Number Train Examples:", X\_train\_downsampled.shape[0])**

**print("Number Positive Train Examples:", (y\_train\_downsampled == 1).sum())**

**print("Number Negative Train Examples:", (y\_train\_downsampled == 0).sum())**

**## Apply RF Hyperparameters**

**# Hyperparameter tuning using GridSearchCV**

**param\_grid = {**

**'n\_estimators': [50, 100, 200],**

**'max\_depth': [None, 10, 20, 30],**

**'min\_samples\_split': [2, 5, 10],**

**'min\_samples\_leaf': [1, 2, 4],**

**'bootstrap': [True, False]**

**}**

**grid\_search = GridSearchCV(estimator=RandomForestClassifier(random\_state=42),**

**param\_grid=param\_grid,**

**cv=3,**

**n\_jobs=-1,**

**verbose=2,**

**scoring='recall\_macro')**

**grid\_search.fit(X\_train\_downsampled, y\_train\_downsampled)**

**# Get the best parameters**

**best\_params = grid\_search.best\_params\_**

**print(best\_params)**

**## Train Model**

**# Set Up Random Forest model**

**# Train the Random Forest classifier with the best parameters**

**rf\_classifier = RandomForestClassifier(\*\*best\_params, random\_state=42)**

**#rf\_classifier = RandomForestClassifier(random\_state=42)**

**#rf\_classifier.fit(X\_train, y\_train)**

**# Assign model**

**model = rf\_classifier**

**#model = lgbm\_model**

**# Train chosen model**

**#model.fit(X\_train, y\_train)**

**# Retrain the Random Forest classifier on the downsampled data**

**model.fit(X\_train\_downsampled, y\_train\_downsampled)**

**# Evaluate Model**

**## Predict on Test Data**

**# Predict on the test data**

**y\_pred = model.predict(X\_test)**

**## Assess Model Peformance**

**### Generate Accuracy Metrics**

**# Evaluate the model**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**classification\_rep = classification\_report(y\_test, y\_pred)**

**roc\_auc = roc\_auc\_score(y\_test, y\_pred)**

**conf\_matrix = confusion\_matrix(y\_test, y\_pred)**

**### Present Table of Metrics**

**# Extract metrics directly from the classification\_report function in a structured format**

**report\_dict = classification\_report(y\_test, y\_pred, output\_dict=True)**

**# Organize the metrics into a dataframe**

**metrics\_df = pd.DataFrame({**

**'Metric': ['Accuracy', 'ROC AUC Score', 'Precision (Class 0)', 'Recall (Class 0)', 'F1-Score (Class 0)',**

**'Precision (Class 1)', 'Recall (Class 1)', 'F1-Score (Class 1)'],**

**'Value': [accuracy, roc\_auc,**

**report\_dict['0']['precision'], report\_dict['0']['recall'], report\_dict['0']['f1-score'],**

**report\_dict['1']['precision'], report\_dict['1']['recall'], report\_dict['1']['f1-score']]**

**})**

**# Display the dataframe in a tabular format**

**display(HTML(metrics\_df.to\_html(index=False, classes="table table-striped table-bordered")))**

**### Generate Confusion Matrix**

**# Plotting the confusion matrix**

**conf\_matrix = confusion\_matrix(y\_test, y\_pred)**

**plt.figure(figsize=(8, 6))**

**sns.heatmap(conf\_matrix, annot=True, fmt='g', cmap='Blues', cbar=False)**

**plt.xlabel('Predicted Label')**

**plt.ylabel('True Label')**

**plt.title('Confusion Matrix')**

**plt.show()**

**# Generate ANCHOR Values**

**#### Suppress Warnings to clean up output**

**import warnings**

**warnings.simplefilter(action='ignore', category=Warning)**

**# 1. Create the Anchor explainer for tabular data**

**# We need to provide the training data for the explainer to learn the data distribution**

**explainer = AnchorTabular(model.predict, X\_train\_downsampled.columns.tolist())**

**# 2. Fit the explainer on the training data**

**# The explainer needs to learn the statistics of the training data to work effectively**

**explainer.fit(X\_train\_downsampled.values)**

**# 3. Select a random sample of 5 instances from the test data**

**random\_sample = X\_test.sample(5, random\_state=42)**

**# 4. For each instance in the random sample, generate an explanation**

**explanations = []**

**for idx, row in random\_sample.iterrows():**

**explanation = explainer.explain(row.values, threshold=0.95) # threshold ensures 95% confidence in explanation**

**explanations.append(explanation.anchor)**

**# 5. Convert explanations to a visually appealing format**

**explanation\_df = pd.DataFrame({'Instance Index': random\_sample.index, 'Anchor Explanation': explanations})**

**# 6. Save the explanations to a CSV file**

**explanation\_filepath = "anchor\_explanations.csv"**

**explanation\_df.to\_csv(explanation\_filepath, index=False)**

**explanation\_df**