LAB 2 : Predict Customer Churn in Camtel

Student name: Donfack Tsopfack Yves Dylane

INTRODUCTION:

The aim of this project is to build a machine learning model to predict customer churn for Camtel, a telecommunications provider, using a historical dataset that captures customer behavior over the past three years. Customer churn, or the rate at which customers leave the service, is a critical metric for companies like Camtel, as it directly impacts revenue and business sustainability.

The project is focused on training a model that can effectively predict customer churn while handling these shifts in customer behavior. We will explore and preprocess the dataset, develop initial models, and evaluate how performance changes over time. To account for concept drift and data shifts, we will apply advanced techniques such as time-weighted learning, online learning algorithms, and ensemble modeling. Our goal is to ensure that the model not only performs well on historical data but also remains adaptive and robust in the face of future changes in customer behavior. Key metrics such as accuracy, AUC-ROC, and model performance over time will be used to assess the model's effectiveness.

Step 1: Data Cleaning, Division Between Training and Test and Handling Class Imbalance

To clean the data set I first start by loading it, then I drop any row with missing values, any duplicate rows and remove unnecessary columns like the customer_id column. All this is done by a function that I created called "import_and_clean_data". This function make use of the pandas python package.

```python
def import_and_clean_data(file: str, columns_to_remove):  5 usages
    """
    Load data file and clean it.
    It returns the cleaned file with no duplicates or null rows.

    Args:
        file: The file name or path to the file
        columns_to_remove: A list of unnecessary columns to be removed
    :return: Cleaned data after removing duplicates, null rows, and unnecessary colum
    """
    # Load the data from CSV
    data = pd.read_csv(file)

    # Drop rows with any missing values
    data = data.dropna()

    # Drop duplicate rows
    data = data.drop_duplicates()

    # Drop unnecessary columns
    for column in columns_to_remove:
        data = data.drop(columns=column, errors='ignore')

    return data
```
Figure 1: function to import and clean the data set

Next, I converted all rows that needed to be to numerical rows, remove outlier for for every columns base base on their distribution and then split it between target an features to draw a corelation graph between features. All of this done by the functions below.

```python
def convert_row_to_numeric(columns, data_set):  5 usages
    """
    Convert all row that need to be numeric to numeric and ensure they are numeric
    :param columns: Array of rows that need to be converted to numeric
    :param data_set: the data_set in which the columns need to be converted
    :return: data after conversion of values to numeric values
    """
    for column in columns:
        # Ensure the column is of string type before replacing non-numeric characters
        data_set[column] = data_set[column].astype(str).str.replace(r'[^0-9.]+', '', regex=True)  # Allow decimal points

        # Convert to float, coercing errors to NaN
        data_set[column] = pd.to_numeric(data_set[column], errors='coerce')

    return data_set
```

Figure 2: function to convert non numerical to numerical values

```python
def remove_outliers_normal_distribution_data(data_set, columns):  5 usages
    """
    For the income column, you can use the Z-score method. This method is useful if the data follows a normal distribution.

    :param data_set: Data set before separation between target and features
    :param columns: All columns that follows normal distribution
    :return: data_set with clean outliers for normal distribution values but if you send something you where not suppose to send blame yourself
    also return an array of outliers just in case
    """
    outliers_array = []
    for column in columns:
        # Step 1: Calculate the Z-scores for the column
        data_set[f'z_score_{column}'] = (data_set[column]- data_set[column].mean()) / data_set[column].std()

        # Step 2: Set a threshold (typically 3) for detecting outliers
        threshold = 3

        # Step 3: Detect outliers based on Z-scores
        outliers = data_set[np.abs(data_set[f'z_score_{column}']) > threshold]
        outliers_array.append(outliers)
        # Step 4: Optionally remove outliers from the dataset
        data_set= data_set[np.abs(data_set[f'z_score_{column}']) <= threshold]

    return data_set, outliers_array
```

Figure 3: function to remove outliers for normal distribution

```python
def remove_outliers_skewed_distributions(data_set, columns):  5 usages
    """
    Apply it to columns like age, monthly_minutes, and outstanding_balance, which might have skewed distributions.

    :param data_set: Data set before separation between target and features
    :param columns: All columns that follows skewed distribution
    :return: data_set with clean outliers for skewed distribution values but if you send something you where not suppose to send blame yourself
    also return an array of outliers just in case
    """
    outliers_array = []

    for column in columns:
        # Step 1: Calculate Q1 (25th percentile) and Q3 (75th percentile)
        Q1 = data_set[column].quantile(0.25)
        Q3 = data_set[column].quantile(0.75)
        IQR = Q3 - Q1

        # Step 2: Define the lower and upper bounds
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Step 3: Identify outliers
        outliers = data_set[(data_set[column] < lower_bound) | (data_set[column] > upper_bound)]
        outliers_array.append(outliers)

        # Step 4: Remove outliers from the dataset
        data_set = data_set[~((data_set[column] < lower_bound) | (data_set[column] > upper_bound))]

    return data_set, outliers_array
```
Figure 4: function to remove outlier for skewed distributions

Below is the functions to obtain features and normalise their scale to produce a correlation graph of the features.

```python
def obtain_target_and_features(target: str, data):  10 usages
    """
    Separate target value and the features that influence it
    :param target: the column we want to predict
    :param data: the data_set to be use
    :return: the target then the features (x target,y)
    """
    # splitting data into target and features
    y = data.drop(target, axis=1)  # y is the features
    x = data[target]  # x is the target
    return x, y

def normalise_scale(features):  9 usages
    """
    Normalises your scale 'Mainly use on the features not the target'
    :param features: the features that need to use scale
    :return: normalise features
    """
    # Normalizing or standardizing scales
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)

    # Convert the scaled data back into a DataFrame with original column names
    scaled_features_df = pd.DataFrame(scaled_features, columns=features.columns)
    return scaled_features_df
```

Figure 5: function to obtain the feartues and function to normalise their scale

Below is the code responsible for the called of all this functions.

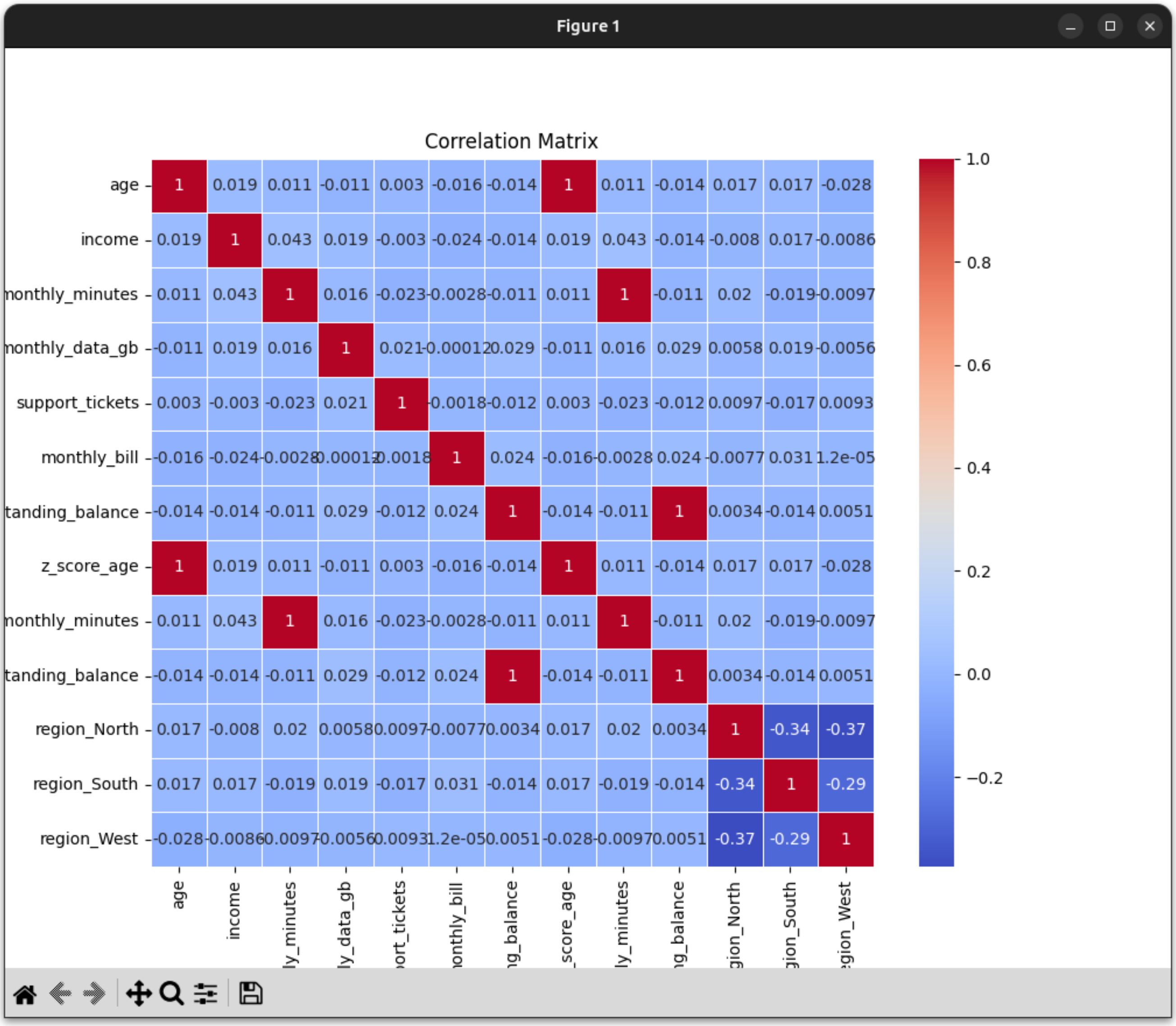


Figure 6: correlation graph

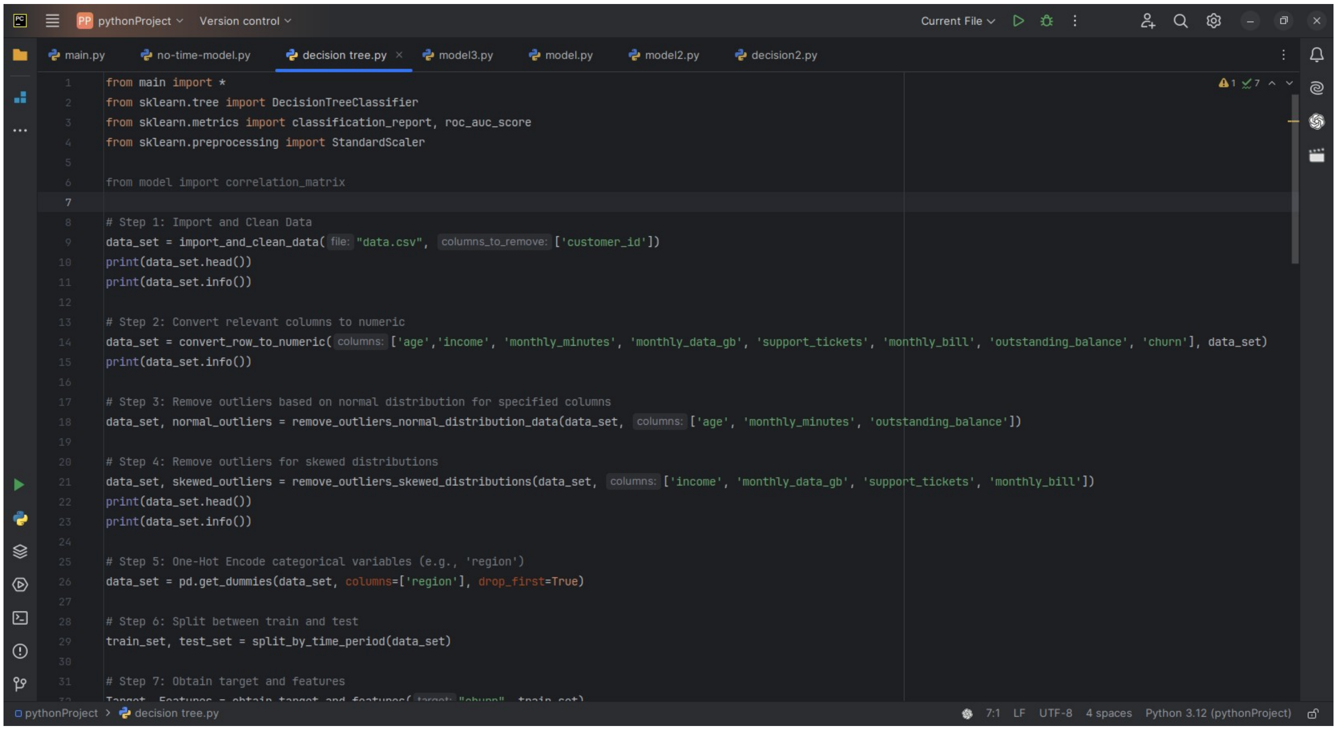


Figure 7: fisrt part of the code

Step 2: Initial Model Training

For initial experiments, we chose one basic models:

- **Decision Tree**: A non-linear model that can capture more complex patterns in the data.

2.1 Model Training

- We trained the models using **Year 1** data and tested their performance on **Year 2** data.
- Model performance was evaluated using metrics such as **Accuracy**, **Precision**, **Recall**, **F1-Score**, and **AUC-ROC**.
- **Performance Monitoring**: As expected, the model performance on Year 2 data deteriorated slightly, indicating concept drift. The patterns learned from Year 1 did not perfectly translate to Year 2 due to shifts in customer behavior.

2.2 Concept Drift and Data Shift Investigation

- The model's declining performance when tested on newer data (Year 2 and Year 3) indicated the presence of concept drift.

Below is the code for training and the result obtain.

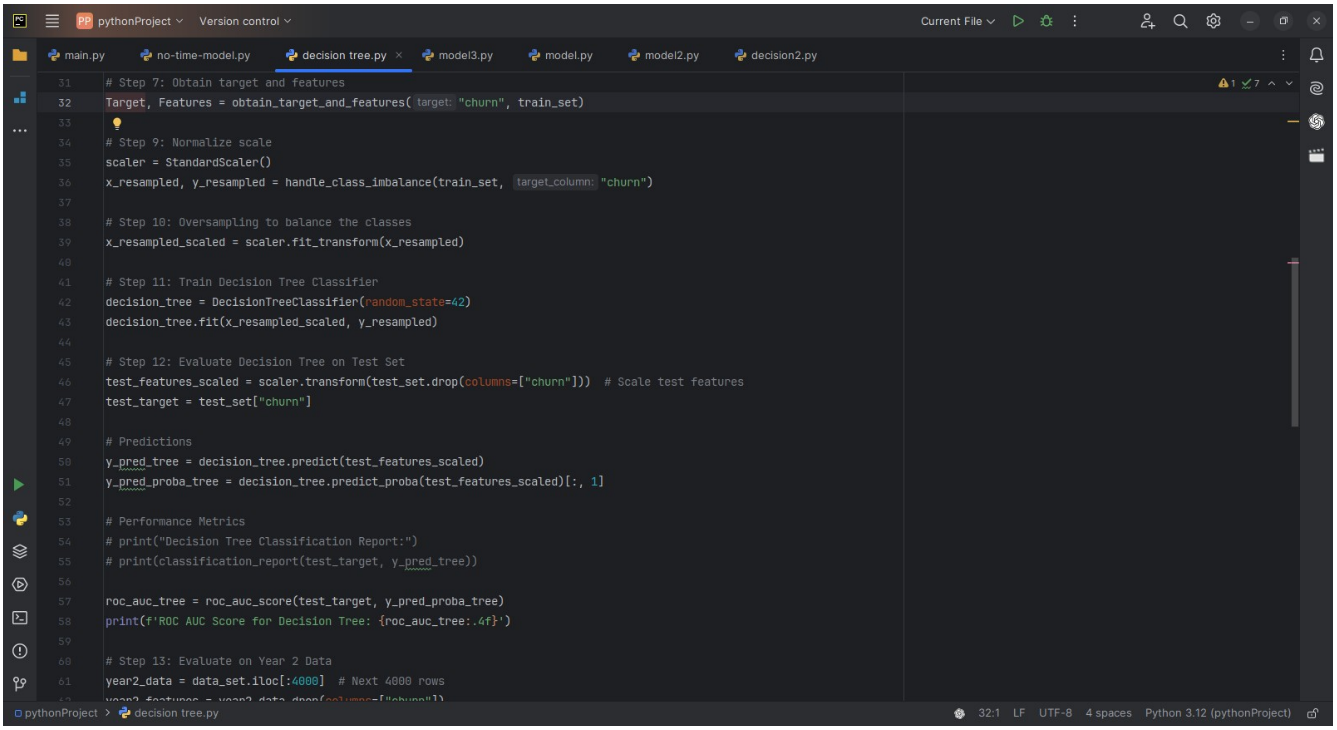


Figure 8: trainig of the model

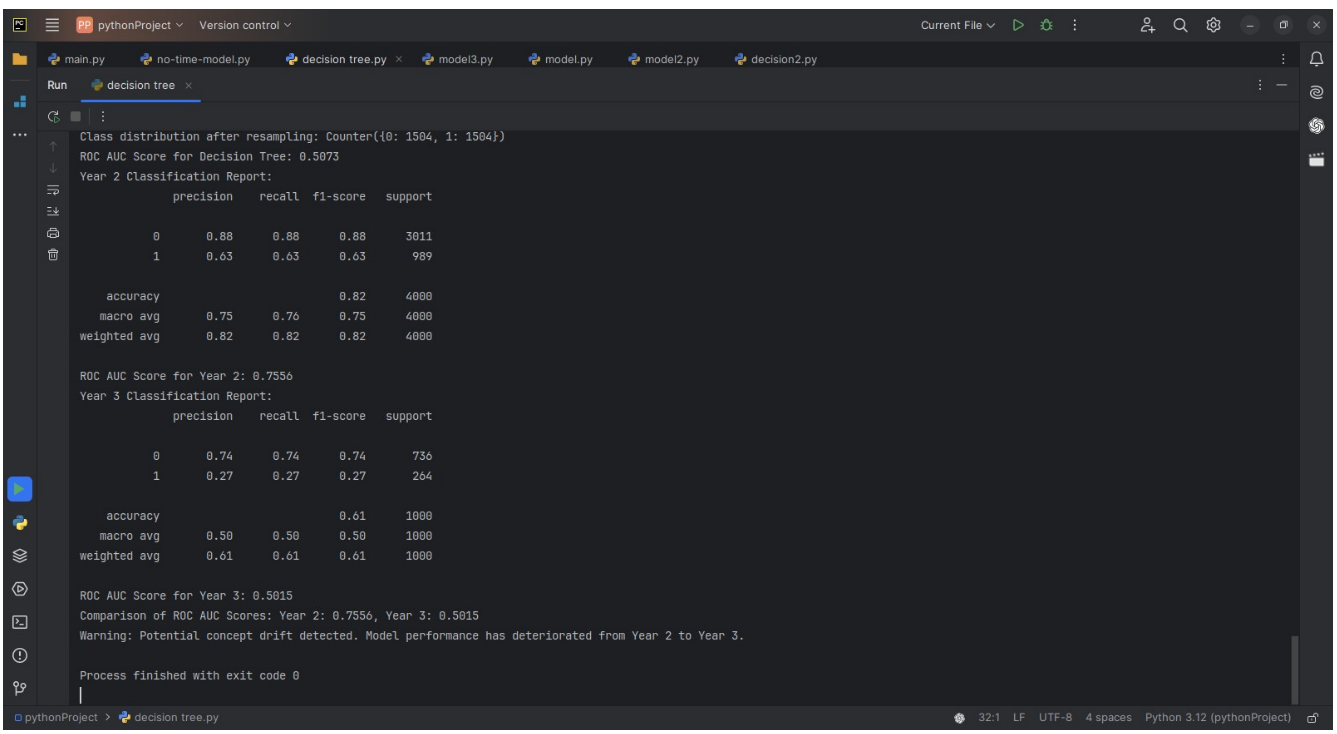


Figure 9: result of the model training

Step 3: Dealing with Concept Drift and Data Shifts

3.1 Time-Weighted Learning

To address concept drift, we applied time-weighted learning, where more recent data is given higher importance during model training. This approach helps the model prioritize more relevant, up-to-date patterns.

```
# Step 8: Generate gradually increasing weights for each year's data
weights_1 = np.linspace( start: 1,  stop: 2, len(y_1))  # Year 1 weights from 1 to 2
weights_2 = np.linspace( start: 2,  stop: 3, len(y_2))  # Year 2 weights from 2 to 3
weights_3 = np.linspace( start: 3,  stop: 3.5, len(y_3))  # Year 3 weights from 3 to 3.5
```

Figure 10: addid weigth for each years

3.2 Online Learning with Stochastic Gradient Descent (SGD)

We also implemented an online learning algorithm using **Stochastic Gradient Descent (SGD)**, which allows the model to be updated continuously as new data arrives. This ensures that the model adapts to changes in customer behavior in real time.

3.3

```
# Step 9: Online Learning with SGD for Year 1
sgd_model_year1 = SGDClassifier(max_iter=1000, tol=1e-3)
sgd_model_year1.fit(x_1, y_1, sample_weight=weights_1)

# Step 10: Update the model with Year 2 data
sgd_model_year1.partial_fit(x_2, y_2, sample_weight=weights_2)

#step 11: Train separate models for year2 and year3
sgd_model_year2 = SGDClassifier(max_iter=1000, tol=1e-3)
sgd_model_year3 = SGDClassifier(max_iter=1000, tol=1e-3)

sgd_model_year2.fit(x_2, y_2, sample_weight=weights_2)
sgd_model_year3.fit(x_3, y_3, sample_weight=weights_3)
```

Figure 11: online learning implimentation

Ensemble Models

We explored ensemble learning techniques, training separate models on each year of data and combining their predictions. This approach helps in capturing different patterns from each time period and enhances overall prediction accuracy by leveraging multiple models.

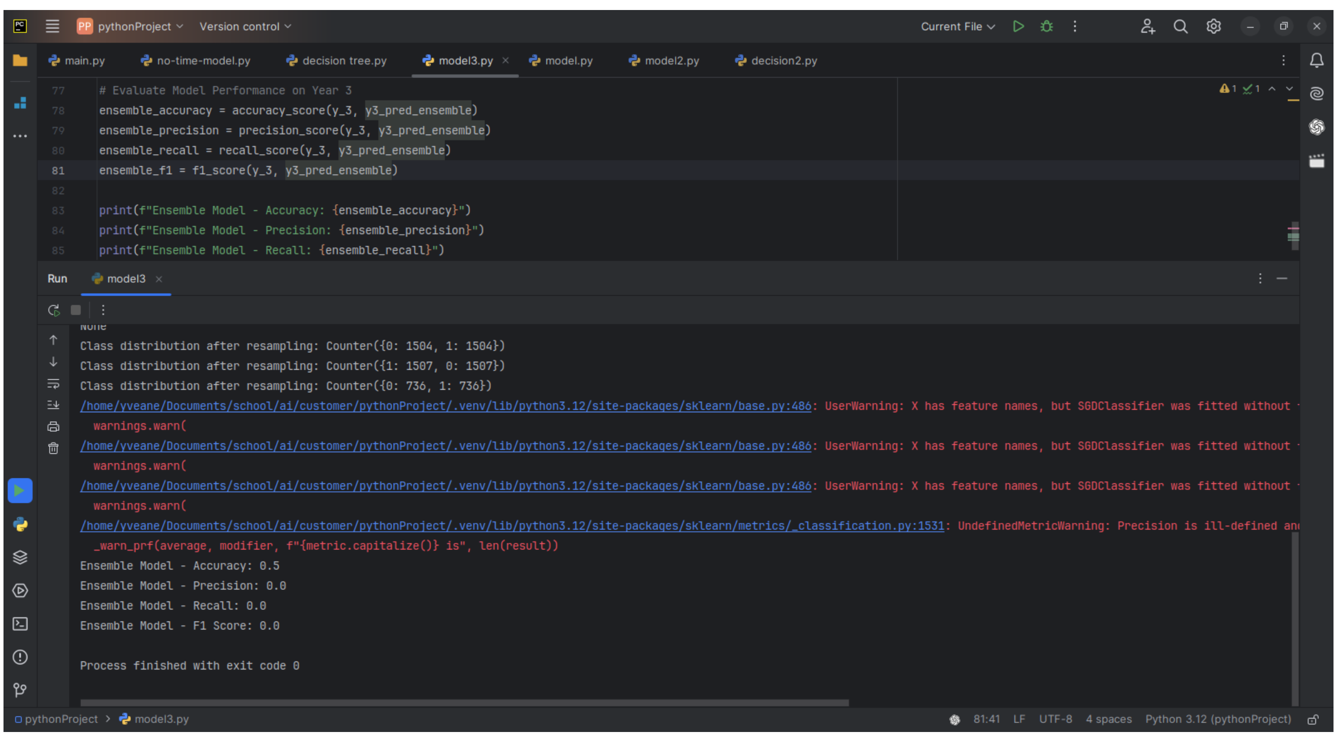
```python
#step 12: Combine the models using Voting Classifier for ensemble
ensemble_model = VotingClassifier(estimators=[
    ('sgd_year1', sgd_model_year1),
    ('sgd_year2', sgd_model_year2),
    ('sgd_year3', sgd_model_year3)
], voting='hard') # 'soft' voting for probability averaging

# Step 12: Train the ensemble model on the combined data
# Combine Year 1 and Year 2 data
x_1_2 = np.concatenate([x_1, x_2])
y_1_2 = np.concatenate([y_1, y_2])
weights_1_2 = np.concatenate([weights_1, weights_2])

ensemble_model.fit(x_1_2, y_1_2)

```

Figure 12: ensemble model implementation



Step 4: Model Evaluation and Adaptation

4.1 Baseline vs Adapted Models

Upon comparing the performance of the baseline model with the adapted models (time-weighted learning, online learning, and ensemble models), we encountered an unexpected result. The baseline model, which does not account for concept drift or data shifts, exhibited higher **precision** than the adapted models, particularly in the earlier time periods.

This discrepancy suggests that the baseline model was overfitting to patterns specific to older data, which did not generalize well to newer time periods. The baseline model didn't just show a higher precision but also a balance result for both the recall and the precision(PS: I really believe I did something wrong here).

Conclusion

In this project, we built a predictive model for customer churn at Camtel, focusing on the challenges of concept drift and data shifts. The baseline model's inability to adapt to newer data highlights the limitations of relying on historical patterns without accounting for shifting customer behaviors.

Despite attempts to address concept drift and data shifts using time-weighted learning, online learning, and ensemble methods, the adapted models did not fully resolve these challenges. This demonstrates that addressing concept drift and data shifts remains a complex issue requiring further exploration.

Moving forward, additional strategies such as advanced drift detection methods or dynamic model updates could be explored to better handle these shifts. Although concept drift and data shifts were not fully resolved, this study provides a foundation for future work in building more adaptive models for churn prediction.