Sample Python Code: Building a Predictive Model for Customer Churn Using Random Forest Algorithm

This document demonstrates the Python code for building a machine learning model to predict customer churn using the Random Forest algorithm. The code is well-commented and structured to highlight important steps such as data preprocessing, model building, evaluation, and optimization.

The Random Forest algorithm is used due to its versatility in classification tasks and ability to handle both regression and classification problems effectively.

Step 1: Loading and Exploring the Dataset

We begin by loading the dataset and exploring its structure. This will help us understand the data before we proceed with any preprocessing.

```python
data = pd.read_csv('customer_churn.csv')
print(data.head())
```

Step 2: Preprocessing the Data

In this step, we handle missing values and encode categorical variables for use in machine learning models.

```python
data = data.dropna()
data = pd.get_dummies(data, drop_first=True)
```

Step 3: Splitting the Data into Training and Testing Sets

We split the dataset into training and testing sets to evaluate our model performance.

```python
X = data[features]
y = data[target]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 4: Building the Random Forest Classifier Model

The Random Forest model is then initialized and trained using the training data.

```python
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Step 5: Evaluating the Model

After training, we evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

```python
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

Step 6: Feature Importance Visualization

We visualize the feature importance using Random Forest's built-in method to better understand which features contribute the most to the predictions.

```python
feature_importances = model.feature_importances_
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.show()
```

Step 7: Model Tuning (Optional but Recommended)

Random Forest performance can be optimized using GridSearchCV, which automates the hyperparameter tuning process.

```python
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42), param_grid=param_grid, cv=3)
grid_search.fit(X_train, y_train)
```

Step 8: Final Evaluation with Tuned Model

Finally, the model is retrained with the best parameters and evaluated again to compare its performance.

```python
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)
y_pred_tuned = best_model.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
print(f"Tuned Accuracy: {accuracy_tuned:.4f}")
```