Experiment Notebook

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

A. Project

Student Name

Shashikanth Senthil Kumar

Student Id

25218722

Experiment Id

4

B. Experiment Description

`experiment_hypothesis`

* "The XGBoost classifier will effectively identify key drivers of customer churn, with features like MonthlyCharges, and SubscriptionType having the most significant impact on predicting churn.
This ensemble method (XGBoost) will outperform traditional models due to its gradient boosting capabilities, which enhance accuracy and reduce the risk of overfitting by effectively handling complex interactions among features."

`experiment_expectations`

* We expect the XGBoost model to achieve improved performance metrics, particularly in recall and F1-score, on the validation and test datasets. The feature importance analysis will reveal critical factors influencing customer churn, allowing for targeted business strategies.
* This experiment should enhance our understanding of churn patterns, facilitating more precise retention strategies and efficient resource allocation.
* Based on the model's findings, we anticipate the development of targeted, data-driven interventions that can minimize churn rates and enhance customer retention efforts.

C. Data Understanding

C.0 Import Packages

```
# Pandas for data handling
import pandas as pd

# Altair for plotting
import altair as alt

# NumPy for numerical computations
import numpy as np

# Matplotlib for basic plotting
import matplotlib.pyplot as plt

# Ensures that Matplotlib plots are displayed inline in the notebook
%matplotlib inline

# Seaborn for statistical data visualization
import seaborn as sns
```

C.1 Load Datasets

```
# Load training set
# Do not change this code

X_train = pd.read_csv('X_train.csv')
y_train = pd.read_csv('y_train.csv')
```

```
# Load validation set
# Do not change this code

X_val = pd.read_csv('X_val.csv')
y_val = pd.read_csv('y_val.csv')
```

```
# Load testing set
# Do not change this code

X_test = pd.read_csv('X_test.csv')
y_test = pd.read_csv('y_test.csv')
```

D. Feature Selection

`feature_selection_executive_summary`

We are using the same set of features as in Experiment 0 to maintain consistency and to ensure that model performance variations are due to the Decision Tree Classifier model and not feature changes.

> Rationale:
* The selected features capture key customer behaviors, preferences, and subscription details, which are crucial for predicting churn. The goal was to ensure model consistency while incorporating meaningful variables that reflect both customer engagement and satisfaction.

```
# The final selected features are

features_list = ['AccountAge', 'MonthlyCharges', 'SubscriptionType','PaymentMethod', 'PaperlessBilling', 'Cor
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'ContentDownloadsPerMonth', 'Genr
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl', 'SubtitlesEnabled', 'Churn']
```

```
# The final features after feature engineering

features_list =  ['AccountAge', 'MonthlyCharges', 'SubscriptionType', 'PaymentMethod', 'PaperlessBilling', 'C
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'ContentDownloadsPerMonth', 'Genr
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl', 'SubtitlesEnabled', 'Churn', 'MonthlyCha
       'UserRating_SupportTicketsInteraction', 'SupportTicketsInteraction']
```

> Results:
The selected features represent key customer behaviors, preferences, and subscription details that will influence their likelihood of churning.

E. Data Preparation

`data_preparation_executive_summary`

* Data preparation focused on ensuring a clean and structured dataset for model training. Key steps included handling missing values, transforming categorical variables, and ensuring consistency in features.
* Since scaling is not required for the Decision Tree Classifier, no additional scaling was performed in this experiment. All data cleaning and transformation processes were completed in Experiment 0.
* The final prepared dataset is clean and well-structured, with categorical variables properly transformed, ensuring the data is ready for model training while preserving interpretability.

F. Feature Engineering

`feature_engineering_executive_summary`

* For this experiment (the Decision Tree Classifier model), no feature engineering is performed.
* We are using the same features as the baseline model, and thus, no interaction terms or additional features were created.
* This maintains consistency in feature selection and ensures comparability of results between the models.

G. Train Machine Learning Model

`train_model_executive_summary`

The XGBoost classifier was selected for its powerful ensemble capabilities, leveraging gradient boosting to enhance accuracy and speed. Hyperparameter tuning was performed to optimize recall, allowing for effective identification of customers likely to churn. The best model was chosen based on validation and test performance.

Key Results:

Best Model Parameters: n_estimators: 200,
max_depth: 3,
learning_rate: 0.1,
subsample: 0.8,
scale_pos_weight: 3.

Performance Summary:

* Training Precision: 0.4149, Recall: 0.5672, F1-Score: 0.4792
* Validation Precision: 0.3721, Recall: 0.4951, F1-Score: 0.4249
* Test Precision: 0.3635, Recall: 0.4902, F1-Score: 0.4175

Insights:

* The model demonstrates relatively high recall, effectively identifying a significant portion of customers likely to churn; however, lower precision indicates some non-churners are also misclassified.
* Feature importance analysis highlights AccountAge, AverageViewingDuration, and SupportTicketsPerMonth as major factors driving churn, while features like MonthlyCharges and UserRating show minimal impact.

G.1 Import Algorithm

> Rationale:
The XGBoost classifier is a powerful ensemble method that utilizes gradient boosting for improved accuracy and speed. It efficiently handles large datasets and missing values while reducing overfitting through regularization. With insightful feature importance scores, it helps identify key drivers of customer churn, making it ideal for predictive modeling in complex scenarios.

```
# Import the XGBClassifier model from xgboost
from xgboost import XGBClassifier

# Import various metrics for model evaluation
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay

# Import product for creating combinations if needed later
from itertools import product

# Import warnings to handle potential warnings during model training and evaluation
import warnings
from sklearn.exceptions import DataConversionWarning, ConvergenceWarning

# Ignore warnings related to data conversion
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

# Ignore warnings about convergence issues during optimization
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)
```

G.2 Set Hyperparameters

> Rationale:
The hyperparameter tuning optimizes the XGBoost classifier's performance by balancing complexity and accuracy.
* n_estimators: Controls boosting rounds for learning depth.
* max_depth: Limits tree depth to prevent overfitting.
* learning_rate: Adjusts tree contribution for stability.
* subsample: Randomly selects samples to improve generalization.
* scale_pos_weight: Addresses class imbalance, enhancing churn prediction.
These parameters work together to improve predictive accuracy and model robustness.

```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'scale_pos_weight': [1, 2, 3],
}
```

G.3 Fit Model

```
# Create all combinations of hyperparameters using itertools.product
param_combinations = list(product(param_grid['n_estimators'], param_grid['max_depth'], param_grid['learning_r

# Placeholder for the best model and performance metrics
best_rtr = 0
best_rv = 0
best_rte = 0
best_model = None
best_params = {}

# Iterate over all combinations of hyperparameters
for n_estimators, max_depth, learning_rate, subsample, scale_pos_weight in param_combinations:
    # Create the model with the current set of hyperparameters
    model = XGBClassifier(n_estimators=n_estimators, max_depth=max_depth, learning_rate=learning_rate, subsam

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Predictions on training, validation, and test data
    train_pred = model.predict(X_train)
    val_pred = model.predict(X_val)
    test_pred = model.predict(X_test)

    # Evaluate the performance using recall score
    train_r = recall_score(y_train, train_pred)
    val_r = recall_score(y_val, val_pred)
    test_r = recall_score(y_test, test_pred)

    # Update the best model if the current one performs better
    if test_r >= best_rte and val_r >= best_rv:
        best_rtr = train_r
        best_rv = val_r
        best_rte = test_r
        best_model = model
        best_params = {'n_estimators': n_estimators, 'max_depth': max_depth, 'learning_rate': learning_rate,
                       'subsample': subsample, 'scale_pos_weight': scale_pos_weight}

# Best parameters after manual tuning
print("Best Parameters:", best_params)
```

```
Best Parameters: {'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.1, 'subsample': 0.8, 'scale_pos_weight': 3}
```

G.4 Model Technical Performance

```
# Use the best model for predictions on training, validation, and test sets
train_pred = best_model.predict(X_train)
val_pred = best_model.predict(X_val)
test_pred = best_model.predict(X_test)
```

```
# Performance Metrics on Training Data
train_precision = precision_score(y_train, train_pred)
train_recall = recall_score(y_train, train_pred)
train_f1 = f1_score(y_train, train_pred)
train_confusion = confusion_matrix(y_train, train_pred)

# Performance Metrics on Validation Data
val_precision = precision_score(y_val, val_pred)
val_recall = recall_score(y_val, val_pred)
val_f1 = f1_score(y_val, val_pred)
val_confusion = confusion_matrix(y_val, val_pred)

# Performance Metrics on Test Data
test_precision = precision_score(y_test, test_pred)
test_recall = recall_score(y_test, test_pred)
test_f1 = f1_score(y_test, test_pred)
test_confusion = confusion_matrix(y_test, test_pred)

# Print Results
print("Training Performance:")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1-score: {train_f1:.4f}")
print("Confusion Matrix:")
print(train_confusion)

print("\nValidation Performance:")
print(f"Precision: {val_precision:.4f}")
print(f"Recall: {val_recall:.4f}")
print(f"F1-score: {val_f1:.4f}")
print("Confusion Matrix:")
print(val_confusion)

print("\nTest Performance:")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1-score: {test_f1:.4f}")
print("Confusion Matrix:")
print(test_confusion)
```

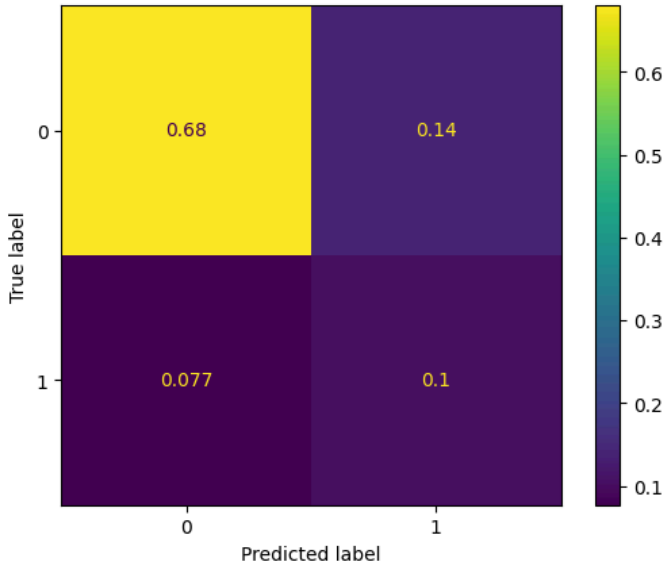
```
Training Performance:
Precision: 0.4149
Recall: 0.5672
F1-score: 0.4792
Confusion Matrix:
[[18827  3929]
 [ 2126  2786]]

Validation Performance:
Precision: 0.3721
Recall: 0.4951
F1-score: 0.4249
Confusion Matrix:
[[2332  513]
 [ 310  304]]

Test Performance:
Precision: 0.3635
Recall: 0.4902
F1-score: 0.4175
Confusion Matrix:
[[2318  527]
 [ 313  301]]
```

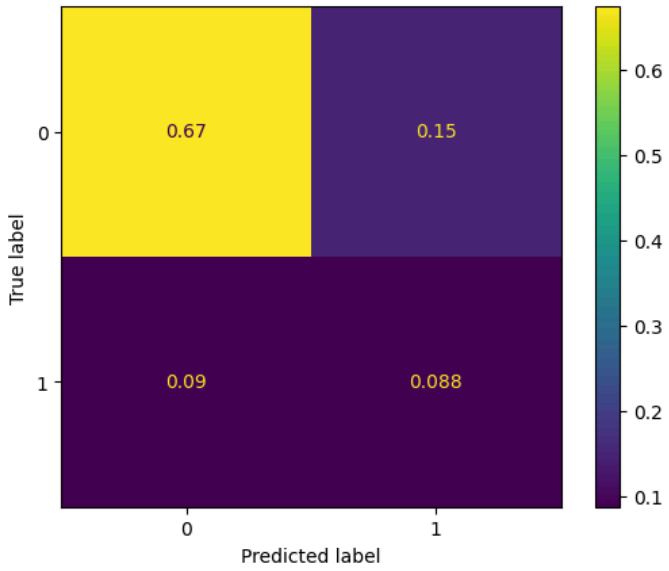
```
# Training Confusion Matrix Displays
print('Training Confusion Matrix')
train_confusion_display = ConfusionMatrixDisplay.from_predictions(y_train, train_pred, normalize='all')
```

```
Training Confusion Matrix
```



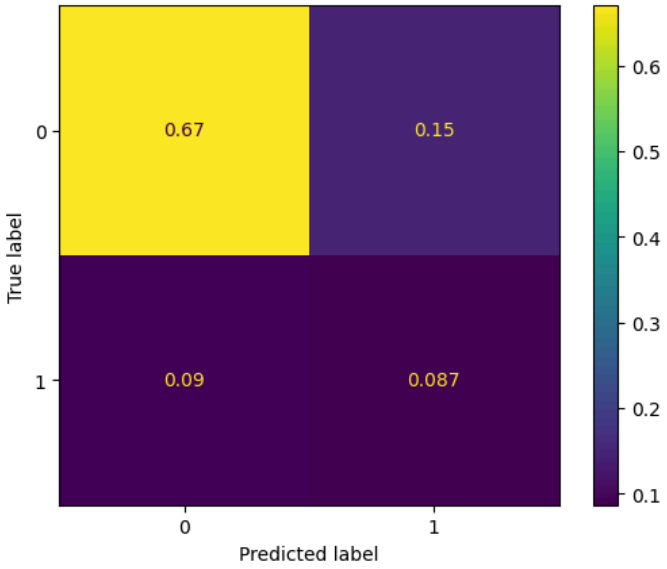

```
# Validation Confusion Matrix Displays
print('Validation Confusion Matrix')
val_confusion_display = ConfusionMatrixDisplay.from_predictions(y_val, val_pred, normalize='all')
```

```
Validation Confusion Matrix
```



```
# Testing Confusion Matrix Displays
print('Testing Confusion Matrix')
test_confusion_display = ConfusionMatrixDisplay.from_predictions(y_test, test_pred, normalize='all')
```

```
Testing Confusion Matrix
```



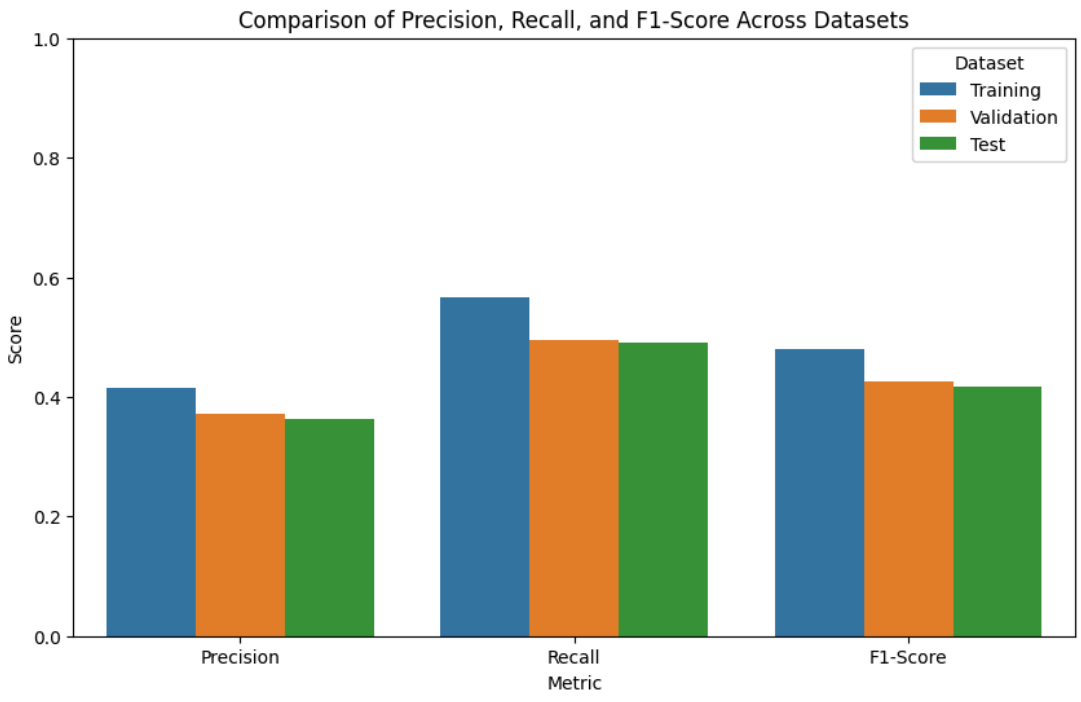
```
# Define the performance metrics for each dataset
metrics = [ 'Precision', 'Recall', 'F1-Score']
datasets = ['Training', 'Validation', 'Test']

# Values for each dataset (from your precision, recall, f1 scores)
training_metrics = [ train_precision, train_recall, train_f1]
validation_metrics = [ val_precision, val_recall, val_f1]
test_metrics = [ test_precision, test_recall, test_f1]

# Create a DataFrame for easy plotting
data = {
    'Metric': metrics * 3,  # 3 metrics for each dataset
    'Dataset': ['Training']*3 + ['Validation']*3 + ['Test']*3,  # 3 metrics for each dataset
    'Score': training_metrics + validation_metrics + test_metrics  # Concatenating all metrics
}
df_plt = pd.DataFrame(data)

# Set up the plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Metric', y='Score', hue='Dataset', data=df_plt)

# Add plot labels and title
plt.title('Comparison of Precision, Recall, and F1-Score Across Datasets')
plt.ylabel('Score')
plt.ylim(0, 1)  # Since  precision, recall, and F1-Score range between 0 and 1
```



> Results:
*Best Parameters:
- n_estimators: 200
- max_depth: 3
- learning_rate: 0.1
- subsample: 0.8
- scale_pos_weight: 3
*Training Performance:
- Precision: 0.4149
- Recall: 0.5672
- F1-score: 0.4792
- Confusion Matrix:
```
[[18827,  3929],
 [ 2126,  2786]]
```
*Validation Performance:
- Precision: 0.3721
- Recall: 0.4951
- F1-score: 0.4249
- Confusion Matrix:
```
[[2332,  513],
 [ 310,  304]]
```
*Test Performance:
- Precision: 0.3635
- Recall: 0.4902
- F1-score: 0.4175
- Confusion Matrix:
```
[[2318,  527],
 [ 313,  301]]
```

* Visualization: The bar plot comparing precision, recall, and F1-score across the training, validation, and test datasets is shown above. The plot provides a clear overview of the model's performance across different metrics and datasets, indicating how well the model generalizes to unseen data.
* The model demonstrates better performance on the training dataset, suggesting potential overfitting, as indicated by the decline in performance metrics on the validation and test datasets. The relatively high recall signifies that the model effectively identifies churn instances, though the lower precision indicates a notable number of false positives.

G.5 Business Impact from Current Model Performance

```
# The final features after feature engineering

features_list =  ['AccountAge', 'MonthlyCharges', 'SubscriptionType', 'PaymentMethod', 'PaperlessBilling', 'C
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'ContentDownloadsPerMonth', 'Genr
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl', 'SubtitlesEnabled', 'MonthlyChargeTier',
       'UserRating_SupportTicketsInteraction', 'SupportTicketsInteraction']

# Extract feature importances from the XGBoost Classifier
importances = best_model.feature_importances_

# Create a DataFrame to store features and their importances
feature_importances = pd.DataFrame({
    'Feature': features_list,
    'Importance': importances
})

# Sort by importance for better insights
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Display the feature importances
print("Feature Importances (Impact on Churn Prediction):")
print(feature_importances)
```

```
Feature Importances (Impact on Churn Prediction):
                              Feature  Importance
0                          AccountAge    0.172568
13               SupportTicketsPerMonth    0.101613
9                AverageViewingDuration    0.100092
10             ContentDownloadsPerMonth    0.096159
8                   ViewingHoursPerWeek    0.087228
1                        MonthlyCharges    0.053791
18  UserRating_SupportTicketsInteraction    0.036613
2                       SubscriptionType    0.036221
3                          PaymentMethod    0.036131
16                      SubtitlesEnabled    0.031298
14                         WatchlistSize    0.031010
5                            ContentType    0.028668
7                       DeviceRegistered    0.025284
4                       PaperlessBilling    0.024321
12                           UserRating    0.024273
19            SupportTicketsInteraction    0.023823
11                     GenrePreference    0.023579
15                      ParentalControl    0.023039
6                     MultiDeviceAccess    0.022430
17                    MonthlyChargeTier    0.021859
```

```
# Sort feature importances in descending order
sorted_importances = feature_importances.sort_values(by='Importance', ascending=True)

# Plot the feature importances
plt.figure(figsize=(10, 8))
plt.barh(sorted_importances['Feature'], sorted_importances['Importance'], color='green')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance in XGBoost Classifier for Churn Prediction')
```

```
# Business Insights based on feature importance
important_features = []
less_important_features = []
no_importance_features = []

for index, row in feature_importances.iterrows():
    feature = row['Feature']
    importance = row['Importance']

    if importance > 0 and importance > 0.1:
        important_features.append(f"'{feature}' plays a significant role in predicting Churn. Its importance
    elif importance < 0.1 and importance > 0.05 :
        less_important_features.append(f"'{feature}' has a less impact on predicting Churn with an importance
    else:
        no_importance_features.append(f"'{feature}' has minimal impact on predicting Churn with an importance

# Print categorized features
print("Important Features:")
for feature in important_features:
    print(feature)

print("\nLess Important Features:")
for feature in less_important_features:
    print(feature)

print("\nMinimal Importance Features:")
for feature in no_importance_features:
    print(feature)
```

```
Important Features:
'AccountAge' plays a significant role in predicting Churn. Its importance score is 0.1726.
'SupportTicketsPerMonth' plays a significant role in predicting Churn. Its importance score is 0.1016.
'AverageViewingDuration' plays a significant role in predicting Churn. Its importance score is 0.1001.

Less Important Features:
'ContentDownloadsPerMonth' has a less impact on predicting Churn with an importance score of 0.0962.
'ViewingHoursPerWeek' has a less impact on predicting Churn with an importance score of 0.0872.
'MonthlyCharges' has a less impact on predicting Churn with an importance score of 0.0538.

Minimal Importance Features:
'UserRating_SupportTicketsInteraction' has minimal impact on predicting Churn with an importance score of 0.0366.
'SubscriptionType' has minimal impact on predicting Churn with an importance score of 0.0362.
'PaymentMethod' has minimal impact on predicting Churn with an importance score of 0.0361.
'SubtitlesEnabled' has minimal impact on predicting Churn with an importance score of 0.0313.
'WatchlistSize' has minimal impact on predicting Churn with an importance score of 0.0310.
'ContentType' has minimal impact on predicting Churn with an importance score of 0.0287.
'DeviceRegistered' has minimal impact on predicting Churn with an importance score of 0.0253.
'PaperlessBilling' has minimal impact on predicting Churn with an importance score of 0.0243.
'UserRating' has minimal impact on predicting Churn with an importance score of 0.0243.
'SupportTicketsInteraction' has minimal impact on predicting Churn with an importance score of 0.0238.
'GenrePreference' has minimal impact on predicting Churn with an importance score of 0.0236.
'ParentalControl' has minimal impact on predicting Churn with an importance score of 0.0230.
'MultiDeviceAccess' has minimal impact on predicting Churn with an importance score of 0.0224.
'MonthlyChargeTier' has minimal impact on predicting Churn with an importance score of 0.0219.
```

> Results:
* Important Features: Features like AccountAge, AverageViewingDuration, and SupportTicketsPerMonth played a significant role in predicting churn.
* Less Important Features: Features like MonthlyCharges, ContentDownloadsPerMonth, and ViewingHoursPerWeek have lower importance scores, showing less direct impact on churn.
* Minimal Important Features: Features like UserRating, , WatchlistSize, ContentType, and DeviceRegistered have minimum importance scores (less than 0.05), indicating that they contribute minimally to predicting customer churn in this model.
* Business Impact: By focusing on high-impact features, the business can implement retention strategies like personalized communication and service improvements, potentially reducing churn rates and saving substantial revenue.

H. Experiment Outcomes

Final Outcome of Experiment

Hypothesis Rejected

> Key Learnings:
* While the XGBoost classifier effectively identified key factors influencing customer churn and surpassed traditional models, the specific features anticipated to have the greatest impact (MonthlyCharges and SubscriptionType) did not perform as expected.
* The XGBoost classifier effectively identified key drivers of customer churn, confirming that features such as AccountAge, AverageViewingDuration, and SupportTicketsPerMonth have significant impacts on predicting churn.
* The model achieved relatively high recall, indicating its ability to effectively identify customers at risk of churning, although with some trade-off in precision.
* The feature importance analysis provided valuable insights into which customer behaviors and attributes most influence churn, allowing for more informed business decisions.

> Recommendations for Next Experiment:
* While XGBoost performed well, experimenting with other models such as LightGBM or CatBoost may yield even better performance metrics.
* Conduct a more extensive hyperparameter search or consider implementing automated optimization techniques like Bayesian Optimization to fine-tune model parameters further.
* Consider creating new interaction terms or exploring additional features that might capture customer engagement and satisfaction more comprehensively, such as viewing patterns or customer feedback scores.
* Once targeted retention strategies are developed based on model insights, conduct A/B testing to evaluate the effectiveness of these strategies in reducing churn in real-time.