Experiment Notebook

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

A. Project

Student Name

Shashikanth Senthil Kumar

Student Id

25218722

Experiment Id

3

B. Experiment Description

experiment_hypothesis

* The Random Forest Classifier will effectively identify key drivers of customer churn, and ViewingHoursPerWeek, and AverageViewingDuration will have the highest impact on predicting churn.
This ensemble method(RandomForest Classifier) will outperform simpler models due to its ability to capture complex feature interactions and reduce overfitting.

experiment_expectations

* We expect the Random Forest model to achieve high recall, capturing a large portion of customers likely to churn, though with some trade-off in precision. The feature importance analysis will provide actionable insights into key churn drivers.
* This experiment should offer a clearer understanding of churn patterns, supporting more targeted retention strategies and better resource allocation.
* Based on the model's findings, we anticipate developing precise, data-driven interventions that can minimize churn and improve customer retention.

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 RandomForest 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 RanndomForest 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 RandomForest 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 Random Forest Classifier was selected for its ability to handle complex datasets and identify key churn drivers. Hyperparameter tuning optimized for recall to effectively capture 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: 5,
min_samples_split: 2,
min_samples_leaf: 1,
class_weight: balanced.

Performance Summary:

* Training Precision: 0.3260, Recall: 0.7097, F1-Score: 0.4468
* Validation Precision: 0.3037, Recall: 0.6564, F1-Score: 0.4152
* Test Precision: 0.3040, Recall: 0.6629, F1-Score: 0.4168

Insights:

* The model achieves high recall, effectively identifying a significant portion of customers likely to churn, though the lower precision indicates some non-churners are also misclassified.
* Feature importance analysis highlights AccountAge, ViewingHoursPerWeek, AverageViewingDuration, and ContentDownloadsPerMonth as major factors driving churn, while features like MonthlyCharges and SupportTicketsPerMonth have minimal impact.

G.1 Import Algorithm

> Rationale:
The Random Forest Classifier is an effective ensemble model for binary classification, combining multiple decision trees to improve accuracy and reduce overfitting. It handles both numerical and categorical data with minimal preprocessing and is particularly adept at managing imbalanced datasets. Its ability to identify feature importance offers valuable insights for churn prediction, helping to target retention strategies effectively.

```
# Import the RandomForestClassifier  model from the sklearn library
from sklearn.ensemble import RandomForestClassifier

# 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 parameter grid for tuning the Random Forest Classifier aims to enhance performance and mitigate overfitting.
* n_estimators: Controls the number of trees; more trees generally improve accuracy by reducing variance.
* max_depth: Limits tree depth to prevent overfitting while capturing essential patterns.
* min_samples_split: Sets the minimum samples needed to split a node, avoiding insignificant splits.
* min_samples_leaf: Ensures a minimum number of samples in leaf nodes, reducing noise sensitivity.
* class_weight: Adjusts for class imbalances, enhancing sensitivity to the minority class, crucial for churn prediction.
This grid explores configurations to balance model complexity and predictive accuracy effectively.

```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [ 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': ['balanced']
}
```

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['min_sample

# 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, min_samples_split, min_samples_leaf, class_weight in param_combinations:
    # Create the model with the current set of hyperparameters
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samp

    # 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, 'min_samples_split': min_samples
                       'min_samples_leaf': min_samples_leaf, 'class_weight': class_weight}

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

```
Best Parameters: {'n_estimators': 100, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 2, 'class_weight': 'balancec
```

G.4 Model Technical Performance

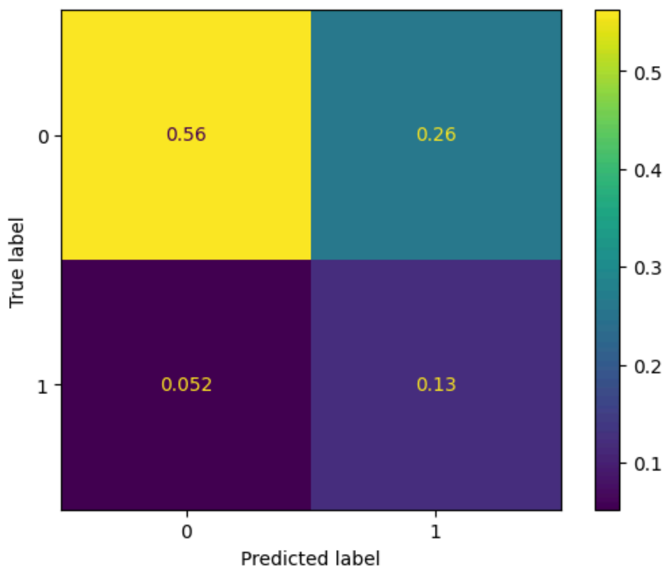
```
# Use the best model for predictions on training, validation, and test sets
train_pred = best_model.predict(X_train)
val_pred = best_model.predict(X_val)
test_pred = best_model.predict(X_test)
```

```python
# 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: 0.3252
Recall: 0.7066
F1-score: 0.4454
Confusion Matrix:
[[15554  7202]
 [ 1441  3471]]

Validation Performance:
Precision: 0.3061
Recall: 0.6645
F1-score: 0.4191
Confusion Matrix:
[[1920  925]
 [ 206  408]]

Test Performance:
Precision: 0.3050
Recall: 0.6612
F1-score: 0.4175
Confusion Matrix:
[[1920  925]
 [ 208  406]]
```

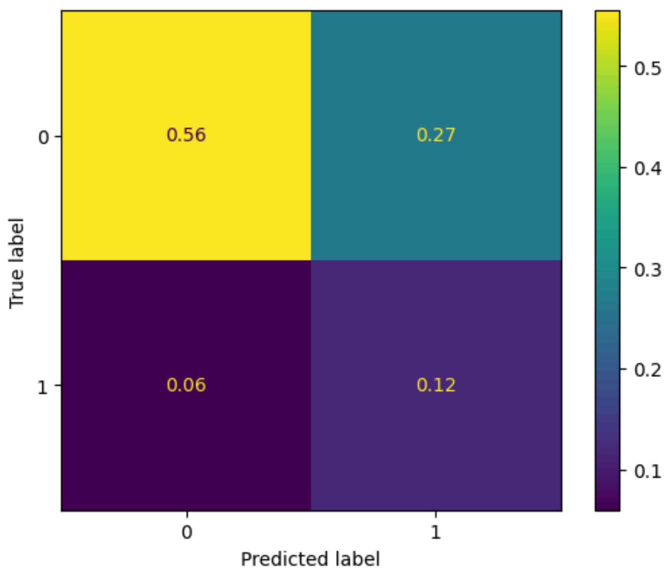
```python
# Training Confusion Matrix Displays
print('Training Confusion Matrix')
train_confusion_display = ConfusionMatrixDisplay.from_predictions(y_train, train_pred, normalize='all')
```

```
Training Confusion Matrix
```



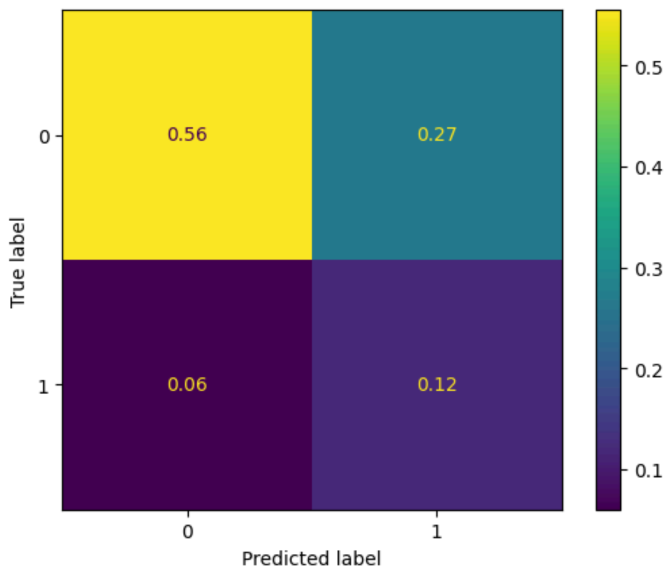
```python
# Validation Confusion Matrix Displays
print('Validation Confusion Matrix')
val_confusion_display = ConfusionMatrixDisplay.from_predictions(y_val, val_pred, normalize='all')
```

```
Validation Confusion Matrix
```



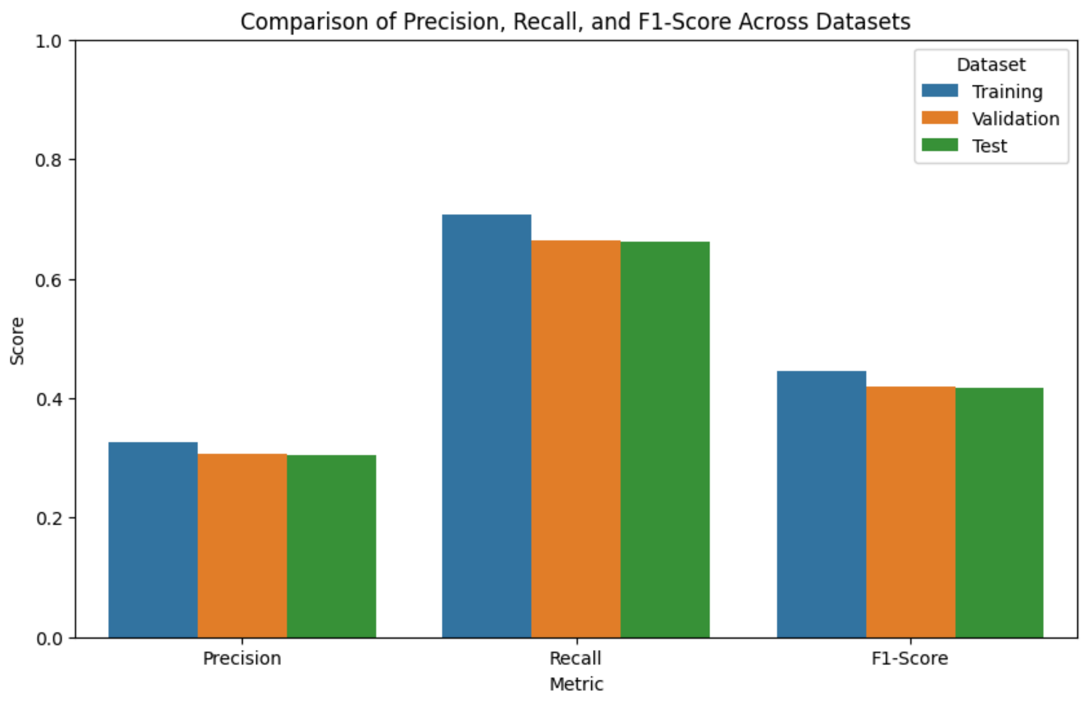
```
# Testing Confusion Matrix Displays
print('Testing Confusion Matrix')
test_confusion_display = ConfusionMatrixDisplay.from_predictions(y_test, test_pred, normalize='all')
```

```
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
```

```
(0.0, 1.0)
```



> Results:
*Best Parameters:
  - n_estimators: 200
  - max_depth: 5
  - min_samples_split: 2
  - min_samples_leaf: 1
  - class_weight: balanced
*Training Performance:
  - Precision: 0.3260
  - Recall: 0.7097
  - F1-score: 0.4468
  - Confusion Matrix:

```
[[15548,  7208],
 [ 1426,  3486]]
```

*Validation Performance:
  - Precision: 0.3037
  - Recall: 0.6564
  - F1-score: 0.4152
  - Confusion Matrix:

```
[[1921,  924],
 [ 211,  403]]
```

*Test Performance:
  - Precision: 0.3040
  - Recall: 0.6629
  - F1-score: 0.4168
  - Confusion Matrix:

```
[[1913,  932],
 [ 207,  407]]
```

* 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 has a high recall, indicating it effectively captures a large number of churn instances, but with lower precision, suggesting it may also have a significant number of false positives.

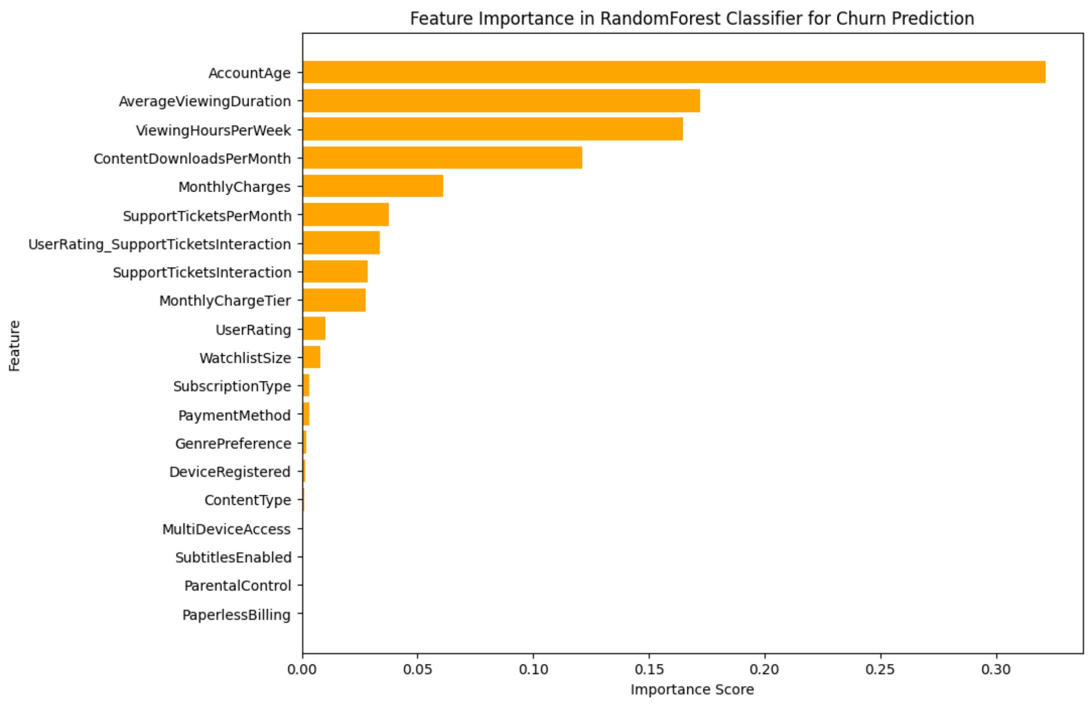
G.5 Business Impact from Current Model Performance

```
# The final features after feature engineering

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

# Extract feature importances from the RandomForest 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    0.321374
9                 AverageViewingDuration    0.172319
8                    ViewingHoursPerWeek    0.164943
10              ContentDownloadsPerMonth    0.121437
1                         MonthlyCharges    0.061327
13                SupportTicketsPerMonth    0.037757
18  UserRating_SupportTicketsInteraction    0.033647
19             SupportTicketsInteraction    0.028508
17                     MonthlyChargeTier    0.027642
12                            UserRating    0.010009
14                         WatchlistSize    0.008183
2                       SubscriptionType    0.003101
3                          PaymentMethod    0.003042
11                       GenrePreference    0.002064
7                       DeviceRegistered    0.001462
5                            ContentType    0.000922
6                      MultiDeviceAccess    0.000772
16                      SubtitlesEnabled    0.000656
15                       ParentalControl    0.000596
4                       PaperlessBilling    0.000239
```

```python
# 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='orange')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance in RandomForest Classifier for Churn Prediction')
```

```
Text(0.5, 1.0, 'Feature Importance in RandomForest 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
    else :
        less_important_features.append(f"'{feature}' has a minimal impact on predicting Churn with an importa


# 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)
```

```
Important Features:
'AccountAge' plays a significant role in predicting Churn. Its importance score is 0.3214.
'AverageViewingDuration' plays a significant role in predicting Churn. Its importance score is 0.1723.
'ViewingHoursPerWeek' plays a significant role in predicting Churn. Its importance score is 0.1649.
'ContentDownloadsPerMonth' plays a significant role in predicting Churn. Its importance score is 0.1214.

Less Important Features:
'MonthlyCharges' has a minimal impact on predicting Churn with an importance score of 0.0613.
'SupportTicketsPerMonth' has a minimal impact on predicting Churn with an importance score of 0.0378.
'UserRating_SupportTicketsInteraction' has a minimal impact on predicting Churn with an importance score of 0.0336.
'SupportTicketsInteraction' has a minimal impact on predicting Churn with an importance score of 0.0285.
'MonthlyChargeTier' has a minimal impact on predicting Churn with an importance score of 0.0276.
'UserRating' has a minimal impact on predicting Churn with an importance score of 0.0100.
'WatchlistSize' has a minimal impact on predicting Churn with an importance score of 0.0082.
'SubscriptionType' has a minimal impact on predicting Churn with an importance score of 0.0031.
'PaymentMethod' has a minimal impact on predicting Churn with an importance score of 0.0030.
'GenrePreference' has a minimal impact on predicting Churn with an importance score of 0.0021.
'DeviceRegistered' has a minimal impact on predicting Churn with an importance score of 0.0015.
'ContentType' has a minimal impact on predicting Churn with an importance score of 0.0009.
'MultiDeviceAccess' has a minimal impact on predicting Churn with an importance score of 0.0008.
'SubtitlesEnabled' has a minimal impact on predicting Churn with an importance score of 0.0007.
'ParentalControl' has a minimal impact on predicting Churn with an importance score of 0.0006.
'PaperlessBilling' has a minimal impact on predicting Churn with an importance score of 0.0002.
```

> Results:
* Important Features: Features like AccountAge, AverageViewingDuration, ViewingHoursPerWeek, and ContentDownloadsPerMonth played a significant role in predicting churn.
* Less Important Features: Features like MonthlyCharges, UserRating_SupportTicketsInteraction, SupportTicketsInteraction, WatchlistSize, and UserRating all have lower importance scores, showing less direct impact on churn.
* Minimal Important Features:Features like SubscriptionType, PaymentMethod, ParentalControl, SubtitlesEnabled and others have very low importance scores (less than 0.005), indicating that they contribute minimal 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 Confirmed

> Key Learnings:
* AccountAge, AverageViewingDuration, and ViewingHoursPerWeek were the most significant features, driving churn predictions.
* Features like MonthlyCharges and UserRating had little effect on predicting churn.
* The Random Forest model successfully captured a large portion of churn cases, but precision was lower, indicating some non-churners were misclassified.
* The Random Forest model outperformed simpler models, but there is room to improve precision.

> Recommendations for Next Experiment:
* Since the current model excels in recall but falls short in precision, consider using ensemble methods like XGBoost, which can fine-tune the balance between recall and precision.
* Experiment with Tuning parameters like n_estimators, max_depth, learning_rate, and subsample to control the complexity of the model.
* Adjust the scale_pos_weight parameter to give more weight to the churn class.It will help you to reduce tthe class imbalance.
* Post-training, analyze feature importance from the XGBoost model to further refine feature selection and focus on the most impactful predictors of churn.