Experiment Notebook

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

A. Project

Student Name	Student Id	Experiment Id
Shashikanth Senthil Kumar	25218722	2

B. Experiment Description

`experiment_hypothesis`

* "The Decision Tree model will identify key drivers of customer churn, and AccountAge, and MonthlyCharges will have the highest impact on predicting churn, based on the structure of the tree.
The tree's performance, measured by recall, precision, and F1-score, will exceed that of previous models (Logistic Regression) due to its ability to capture non-linear relationships between features and churn behavior."

`experiment_expectations`

* We anticipate better recall values due to the Decision Tree's ability to model complex relationships and interactions among features. The model's interpretability will also allow for easier identification of the key drivers of churn.
* This experiment should yield clearer insights into customer behavior and churn patterns, facilitating more informed decision-making regarding retention strategies.
* Based on the model findings, we expect to develop actionable strategies that can be implemented to minimize churn and enhance customer satisfaction effectively.

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
```

```
# 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.

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 Decision Tree Classifier was selected for its versatility, interpretability, and effectiveness in predicting customer churn. The model was trained using a hyperparameter tuning, optimizing for recall. The best model was identified based on validation and test set performance.

Key Results:

Best Model Parameters: Criterion: gini,
Max Depth: 5,
Min Samples Split: 10,
Min Samples Leaf: 4,
Class Weight: balanced.

Performance Summary:

* Training Precision: 0.2861, Recall: 0.7414, F1-Score: 0.4129
* Validation Precision: 0.2734, Recall: 0.7036, F1-Score: 0.3938
* Test Precision: 0.2636, Recall: 0.6873, F1-Score: 0.3810

Insights:

* The model achieves high recall, effectively identifying a significant number of customers who are likely to churn. However, the lower precision indicates a higher number of false positives, meaning that while many churners are captured, some non-churners are also misclassified.
* Feature importance analysis reveals that factors such as AccountAge, AverageViewingDuration, ViewingHoursPerWeek, and ContentDownloadsPerMonth significantly increase the likelihood of churn. Conversely, features like MonthlyCharges and SupportTicketsPerMonth have a minimal impact on churn prediction.

> Rationale:
The Decision Tree Classifier is a versatile and interpretable model for binary classification, effectively handling both numerical and categorical data. It captures non-linear relationships and interactions without extensive preprocessing. Its intuitive decision paths provide actionable insights for churn prediction, making it valuable in identifying key features influencing customer churn.

```
# Import the DecisionTreeClassifier  model from the sklearn library
from sklearn.tree import DecisionTreeClassifier

# 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 hyperparameters for the Decision Tree Classifier are chosen to optimize performance and reduce overfitting:

* Criterion: Selects the impurity measure ('gini' or 'entropy') for better splits.
* Max Depth: Limits tree depth to prevent overfitting while capturing data patterns.
* Min Samples Split: Sets the minimum samples required to split a node, avoiding insignificant splits.
* Min Samples Leaf: Defines the minimum samples for a leaf node, preventing overfitting from sparse leaves.
* Class Weight: Adjusts weights for imbalanced classes, enhancing sensitivity to the minority class (churn).

```
# Set hyperparameters for  DecisionTreeClassifier
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10],
    '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['criterion'], param_grid['max_depth'], param_grid['min_samples_s

# 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 criterion, max_depth, min_samples_split, min_samples_leaf, class_weight in param_combinations:
    # Create the model with the current set of hyperparameters
    model = DecisionTreeClassifier(criterion=criterion, max_depth=max_depth, min_samples_split=min_samples_sp

    # 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 = {'criterion': criterion, 'max_depth': max_depth, 'min_samples_split': min_samples_split
                       'min_samples_leaf': min_samples_leaf, 'class_weight': class_weight}

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

```
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4, 'class_weight': 'balance
```

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)
```

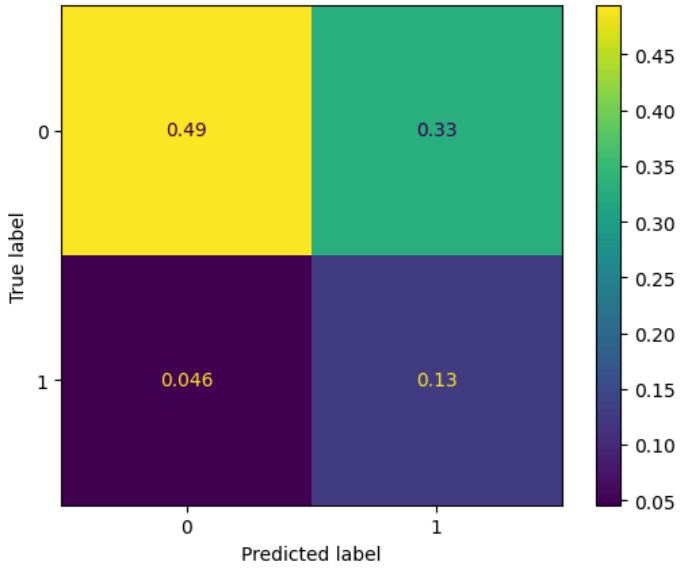
```
Training Performance:
Precision: 0.2861
Recall: 0.7414
F1-score: 0.4129
Confusion Matrix:
[[13668  9088]
 [ 1270  3642]]

Validation Performance:
Precision: 0.2734
Recall: 0.7036
F1-score: 0.3938
Confusion Matrix:
[[1697 1148]
 [ 182  432]]

Test Performance:
Precision: 0.2636
Recall: 0.6873
F1-score: 0.3810
Confusion Matrix:
[[1666 1179]
 [ 192  422]]
```

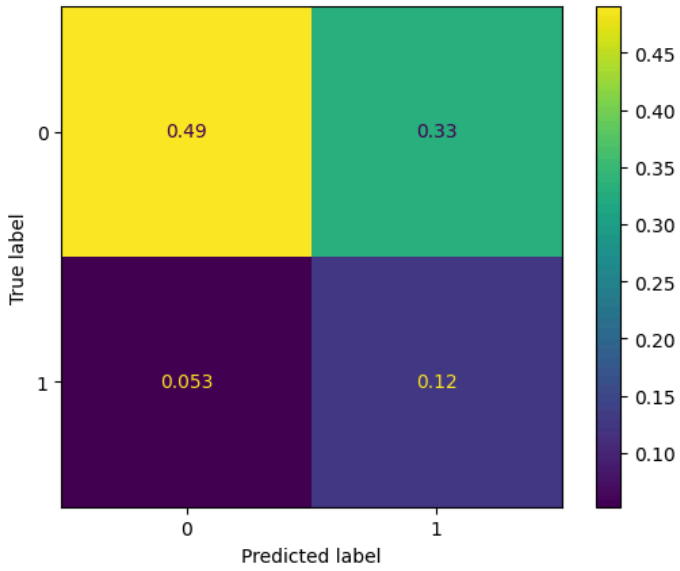
```
# Training Confusion Matrix Displays
print('Training Confusion Matrix')
train_confusion_display = ConfusionMatrixDisplay.from_predictions(y_train, train_pred, normalize='all')
```

```
Training Confusion Matrix
```



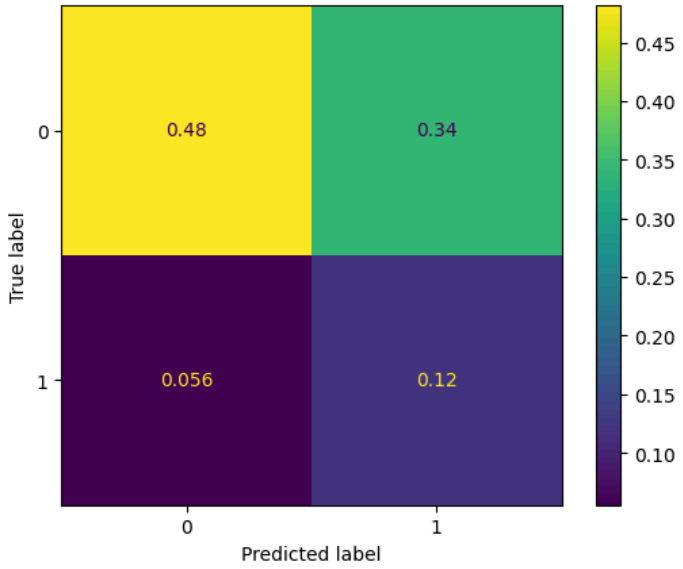

```
# Validation Confusion Matrix Displays
print('Validation Confusion Matrix')
val_confusion_display = ConfusionMatrixDisplay.from_predictions(y_val, val_pred, normalize='all')
```

```
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:
- Criterion: gini
- Max Depth: 5
- Min Samples Split: 10
- Min Samples Leaf: 4
- Class Weight: balanced
*Training Performance:
- Precision: 0.2861
- Recall: 0.7414
- F1-Score: 0.4129
- Confusion Matrix:
```
[[13668,  9088],
 [ 1270,  3642]]
```
*Validation Performance:
- Precision: 0.2734
- Recall: 0.7036
- F1-Score: 0.3938
- Confusion Matrix:
```
[[1697, 1148],
 [ 182,  432]]
```
*Test Performance:
- Precision: 0.2636
- Recall: 0.6873
- F1-Score: 0.3810
- Confusion Matrix:
```
[[1666, 1179],
 [ 192,  422]]
```
* 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 used to train the model
features_list = ['AccountAge', 'MonthlyCharges', 'SubscriptionType', 'PaymentMethod',
       'PaperlessBilling', 'ContentType', 'MultiDeviceAccess',
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration',
       'ContentDownloadsPerMonth', 'GenrePreference', 'UserRating',
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl',
       'SubtitlesEnabled', 'MonthlyChargeTier',
       'UserRating_SupportTicketsInteraction', 'SupportTicketsInteraction']

# Extract feature importances from the Decision Tree 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    0.387731
9                AverageViewingDuration    0.199672
8                   ViewingHoursPerWeek    0.179737
10             ContentDownloadsPerMonth    0.147700
1                        MonthlyCharges    0.051738
18  UserRating_SupportTicketsInteraction    0.023294
13               SupportTicketsPerMonth    0.010128
4                      PaperlessBilling    0.000000
5                           ContentType    0.000000
6                     MultiDeviceAccess    0.000000
7                      DeviceRegistered    0.000000
3                         PaymentMethod    0.000000
2                      SubscriptionType    0.000000
11                      GenrePreference    0.000000
12                           UserRating    0.000000
14                        WatchlistSize    0.000000
15                      ParentalControl    0.000000
16                    SubtitlesEnabled    0.000000
17                    MonthlyChargeTier    0.000000
19            SupportTicketsInteraction    0.000000
```

```
# 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='red')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Feature Importance in Decision Tree 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 :
        less_important_features.append(f"'{feature}' has a minimal impact on predicting Churn with an importa
    else:
        no_importance_features.append(f"'{feature}' has no impact on predicting Churn in this model.")

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

```
Important Features:
'AccountAge' plays a significant role in predicting Churn. Its importance score is 0.3877.
'AverageViewingDuration' plays a significant role in predicting Churn. Its importance score is 0.1997.
'ViewingHoursPerWeek' plays a significant role in predicting Churn. Its importance score is 0.1797.
'ContentDownloadsPerMonth' plays a significant role in predicting Churn. Its importance score is 0.1477.

Less Important Features:
'MonthlyCharges' has a minimal impact on predicting Churn with an importance score of 0.0517.
'UserRating_SupportTicketsInteraction' has a minimal impact on predicting Churn with an importance score of 0.0233.
'SupportTicketsPerMonth' has a minimal impact on predicting Churn with an importance score of 0.0101.

No Importance Features:
'PaperlessBilling' has no impact on predicting Churn in this model.
'ContentType' has no impact on predicting Churn in this model.
'MultiDeviceAccess' has no impact on predicting Churn in this model.
'DeviceRegistered' has no impact on predicting Churn in this model.
'PaymentMethod' has no impact on predicting Churn in this model.
'SubscriptionType' has no impact on predicting Churn in this model.
'GenrePreference' has no impact on predicting Churn in this model.
'UserRating' has no impact on predicting Churn in this model.
'WatchlistSize' has no impact on predicting Churn in this model.
'ParentalControl' has no impact on predicting Churn in this model.
'SubtitlesEnabled' has no impact on predicting Churn in this model.
'MonthlyChargeTier' has no impact on predicting Churn in this model.
'SupportTicketsInteraction' has no impact on predicting Churn in this model.
```

> Results:
* Important Features: Features like AccountAge, AverageViewingDuration, ViewingHoursPerWeek, and ContentDownloadsPerMonth played a significant role in predicting churn.
* Less Important Features: Features such as MonthlyCharges, UserRating_SupportTicketsInteraction, and SupportTicketsPerMonth contributed minimally to the model's predictions.
* No Important Features: Features like PaperlessBilling, ContentType, MultiDeviceAccess, DeviceRegistered, and others had an importance score of 0.0000, indicating no impact on churn prediction.
* Business Impact: The model identifies key drivers of churn, allowing for targeted interventions, such as engaging customers with longer account ages or optimizing content offerings based on viewing behavior. By focusing on high-impact features, the business can implement retention strategies, such as personalized communication and service improvements, potentially saving substantial revenue by reducing churn rates.

H. Experiment Outcomes

Final Outcome of Experiment

Hypothesis Partially Confirmed

> Key Learnings:
* The hypothesis was partially confirmed. Because Decision Tree Model gives AccountAge as most important feature but MonthlyCharges as Less Important feature.
* The Decision Tree identified many churn cases but also produced significant false positives. This suggests that while the model is good at flagging potential churn risks, it needs improvement to reduce misclassification of non-churning customers.
* AccountAge, AverageViewingDuration, and ViewingHoursPerWeek were top churn predictors, while others like PaperlessBilling had no impact.
* The tree's interpretability was valuable but may have oversimplified relationships, needing further fine-tuning.

> Recommendations for Next Experiment:
* Try Random Forest Classifier,it can significantly enhance model performance by combining multiple weak learners. They are powerful in reducing overfitting and improving both precision and recall.
* Conduct more thorough hyperparameter tuning, it can help identify more optimal parameters for classifiers like RandomForest, SVMs, and other ensemble methods.
* Since we have a imbalanced Dataset keep class_weight as balanced to improve the model performance.