Experiment Notebook

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

A. Project

Student Name	Student Id	Experiment Id
Shashikanth Senthil Kumar	25218722	1

B. Experiment Description

`experiment_hypothesis`

* Customers with higher monthly charges are more likely to churn.

Logistic Regression is expected to perform well due to the binary nature of the target variable.

`experiment_expectations`

* We expect Logistic Regression to perform well since it is a straightforward binary classification algorithm.
* Tuning hyperparameters, such as (C)regularization strength, will help improve its performance.
* We anticipate that features like monthly charges will significantly influence churn prediction.

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 logistic regression 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

* The data preparation process involved standardizing the numerical features in the dataset.
* This step was essential for ensuring that all input features were on the same scale, which is crucial for models like Logistic Regression that are sensitive to feature scaling.
* The StandardScaler was used to transform the features, bringing them to a mean of 0 and a standard deviation of 1.
* By applying this transformation to the training, validation, and test datasets, the model can now treat all features equally during training, leading to improved performance and faster convergence.

> Rationale:
* Standardizing numerical features ensures that all input variables are on the same scale.
* This is particularly important for models like Logistic Regression, which are sensitive to feature scaling.
* By transforming the features to have a mean of 0 and a standard deviation of 1, we ensure that no feature dominates the model due to differences in scale.

```
# Import StandardScaler from sklearn
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply the StandardScaler to the training, validation and test datasets
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

> Results:
* The features in the training, validation, and test datasets were successfully standardized.
* This transformation allows the model to treat all features equally, ensuring better performance and faster convergence during training.

F. Feature Engineering

feature_engineering_executive_summary

* For this experiment (the Logistic Regression 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 logistic regression model was chosen for its simplicity, scalability, and interpretability in predicting customer churn. The model was trained using a comprehensive grid search for hyperparameter tuning, optimizing for recall due to the imbalanced dataset. The best model was selected based on validation and test set performance.

Key Results:

* Best Model Parameters: C: 0.1,Solver: liblinear,Max Iterations: 500,Class Weight: balanced

Performance Summary:

* Training Precision: 0.3167, Recall: 0.6920, F1-Score: 0.4346
* Validation Precision: 0.3128, Recall: 0.6710, F1-Score: 0.4267
* Test Precision: 0.3079, Recall: 0.6906, F1-Score: 0.4259

Insights:

* The model achieves high recall, meaning it identifies most customers who churn. However, precision is lower, resulting in a higher number of false positives. This indicates the model captures many churners but also misclassifies some non-churners.
* Feature importance analysis shows that factors like higher support tickets, monthly charges, and certain content preferences significantly increase churn likelihood. Features like longer subscription duration and higher viewing engagement decrease churn risk.

Business Impact:

The model highlights key drivers of churn, enabling targeted interventions, such as offering discounts to customers with high support needs or

G.1 Import Algorithm

> Rationale:
Logistic Regression is a robust, interpretable algorithm for binary classification problems like churn prediction. It is simple, scalable, and performs well when the data has clear decision boundaries.

```
# Import the Logistic Regression model from the sklearn library
from sklearn.linear_model import LogisticRegression

# 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)
```

> Rationale:
The parameter grid defines key hyperparameters for optimizing Logistic Regression performance:
* C: Controls regularization strength, balancing overfitting and model flexibility.
* solver: Different solvers ('liblinear', 'lbfgs', 'saga') optimize the model based on data size and characteristics.
* max_iter: Varies the maximum number of iterations to ensure model convergence.
* class_weight: since we got a imbalanced dataset we will set class weight to 'balanced'
This grid allows efficient hyperparameter tuning for the best model performance.

```
# Set hyperparameters for Logistic Regression
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs', 'saga'],
    'max_iter': [100, 200, 500],
    'class_weight': ['balanced']
}
```

G.3 Fit Model

```
# Create all combinations of hyperparameters using itertools.product
param_combinations = list(product(param_grid['C'], param_grid['solver'], param_grid['max_iter'], param_grid['

# 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 C, solver, max_iter, class_weight in param_combinations:
    # Create the model with the current set of hyperparameters
    model = LogisticRegression(C=C, solver=solver, max_iter=max_iter, class_weight=class_weight)

    # 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 val_r >= best_rv and test_r >= best_rte:
        best_rtr = train_r
        best_rv = val_r
        best_rte = test_r
        best_model = model
        best_params = {'C': C, 'solver': solver, 'max_iter': max_iter, 'class_weight': class_weight}

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

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

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

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

```
# 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:
  - C: 0.1
  - Solver: liblinear
  - Class Weight: balanced
  - Max Iterations: 500
* Training Performance:
  - Precision: 0.3167
  - Recall: 0.6920
  - F1-Score: 0.4346
  - Confusion Matrix:
    ```
    [[1542, 7332],
     [1513, 3399]]
    ```
* Validation Performance:
  - Precision: 0.3128
  - Recall: 0.6710
  - F1-Score: 0.4267
  - Confusion Matrix:
    ```
    [[1940,  905],
     [202,  412]]
    ```
* Test Performance:
  - Precision: 0.3079
  - Recall: 0.6906
  - F1-Score: 0.4259
  - Confusion Matrix:
    ```
    [[1892, 953],
     [190, 424]]
    ```
* 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 high recall but relatively low precision, indicating that while it successfully captures most instances of churn, it also has a high rate of false positives.

```
# The final features after feature engineering used for model training

features_list =  ['AccountAge', 'MonthlyCharges', 'SubscriptionType', 'PaymentMethod', 'PaperlessBilling', 'C
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'ContentDownloadsPerMonth', 'Genr
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl', 'SubtitlesEnabled', 'MonthlyChargeTier',
       'UserRating_SupportTicketsInteraction', 'SupportTicketsInteraction']
# Extract feature coefficients
coefficients = pd.DataFrame({
    'Feature': features_list,
    'Coefficient': best_model.coef_.flatten()
})

# Sort by absolute value of coefficient for better insights
coefficients['Absolute_Coefficient'] = coefficients['Coefficient'].abs()
coefficients = coefficients.sort_values(by='Absolute_Coefficient', ascending=False)

# Display the coefficients
print("Feature Coefficients (Impact on Churn Prediction):")
print(coefficients)
```

```
# Plot the coefficients for a better understanding
plt.figure(figsize=(10, 8))
plt.barh(coefficients['Feature'], coefficients['Coefficient'], color='blue')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Feature Importance in Churn Prediction')
```

```
# Business Insights based on the feature coefficients
for index, row in coefficients.iterrows():
    feature = row['Feature']
    coef = row['Coefficient']

    if coef > 0:
        print(f"Positive Impact: '{feature}' increases the predicted likelihood of Churn. Higher values for '
    else:
        print(f"Negative Impact: '{feature}' decreases the predicted likelihood of Churn. Higher values for '
```

> Results:
* Positive Impacts: Features like SupportTicketsPerMonth, MonthlyCharges, UserRating, PaymentMethod, PaperlessBilling, MonthlyChargeTier, GenrePreference, and WatchlistSize increase churn risk, indicating that customers with higher support needs and charges are more likely to leave.
* Negative Impacts: Features such as SubscriptionType, SubtitlesEnabled, ViewingHoursPerWeek, MultiDeviceAccess, ContentType, ContentDownloadsPerMonth, AccountAge, DeviceRegistered, AverageViewingDuration, UserRating_SupportTicketsInteraction, and ParentalControl decrease churn risk, highlighting the importance of engagement and long-term subscriptions.
* Business Use Case: Use the model's insights to target at-risk customers (high support tickets and charges) with a 50% discount for retention. Additionally, promoting content to increase viewing hours can help reduce churn.

Final Outcome of Experiment

Hypothesis Confirmed

> Key Learnings:
* Churn Risk and Charges: MonthlyCharges has a positive impact on the churn, this indicates that higher the monthlycharges the higher possiblity to churn. So. Customers paying higher monthly charges are more likely to churn, like mentioned in the hypothesis. This suggests that pricing may need adjustment to prevent customer loss.
* Key Drivers: High support ticket volume, user ratings, and payment methods were significant churn predictors, emphasizing the importance of customer service in retention.
* Model Performance: Logistic Regression was effective for churn prediction but needs further feature engineering to further improve performance.

> Recommendations for Next Experiment:

* Move from Logistic Regression to Decision Trees to capture non-linear relationships between features and churn. This shift can provide a more flexible modeling approach that may uncover complex interactions not addressed by the linearity assumption of Logistic Regression.
* As Decision Trees can handle non-linear relationships and interactions naturally, retain features that showed strong predictive power in previous analyses, while still considering the removal of low-importance features like SupportTicketsInteraction.
* Experiment with various hyperparameters (e.g., max depth, min samples split) to optimize the model's performance and avoid overfitting while ensuring a good balance between bias and variance.