Predictive Modeling for Amazon Prime Subscription Plans

May 26, 2024

1 Predictive Modeling for Amazon Prime Subscription Plans: Insights from User Data

1.1 Introduction

This project focuses on analyzing Amazon subscriber data to build predictive models that determine factors influencing subscription plan choices (Annual vs. Monthly). By leveraging machine learning techniques, the goal is to provide insights that can help Amazon optimize its subscription offerings, improve customer retention, and enhance user satisfaction.

1.2 Project Workflow

1. **Data Preprocessing**: Cleaning and preparing the dataset for analysis.
2. **Exploratory Data Analysis (EDA)**: Understanding the data through visualization and summary statistics.
3. **Model Development**: Building and evaluating different machine learning models.
4. **Model Selection**: Choosing the best-performing model based on evaluation metrics.
5. **Conclusion and Recommendations**: Summarizing findings and providing actionable insights.

The dataset includes various features related to Amazon subscribers, such as user demographics, usage patterns, and engagement metrics. Below is a summary of the key features:

Field	Description
User ID	Numeric ID for the user
Name	User's name
Email Address	User's email address
Username	Username
Date of Birth	Date of Birth of the user
Gender	User's gender
Location	User's Country
Membership Start Date	Start date of the Amazon Prime membership
Membership End Date	End date of the membership
Payment Information	Payment method used (Visa, Mastercard, Amex)
Renewal Status	Setting of renewal on subscription (Manual vs Automatic)
Usage Frequency	Frequency of platform usage (Occasional, Regular, Frequent)
Purchase History	Most frequently purchased items
Favorite Genres	Favorite genre of content

Field	Description
Devices Used	Device used to access the platform
Engagement Metrics	Engagement level (Low, Medium, High)
Feedback/Ratings	Average feedback/ratings given by the user
Customer Support Interactions	Number of interactions with customer support
Subscription Plan	Type of subscription plan (Annual vs Monthly)

1.3 Imports

[4]:
```
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime

from sklearn.model_selection import train_test_split, PredefinedSplit,␣
↪GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score,␣
↪recall_score, classification_report, confusion_matrix, f1_score

from xgboost import XGBClassifier, plot_importance

import warnings

# Ignore specific FutureWarning related to seaborn
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2 Data Preprocessing

[6]:
```
# Load the dataset
data = pd.read_csv("C:/Users/luke3/Documents/GitHub/
↪Predictive-Modeling-for-Amazon-Prime-Subscription-Plans-Insights-from-User-Data/
↪data/amazon_prime_users.csv")
```

[7]:
```
# Display the first 5 rows of the data set
data.head()
```

[7]:
```
   User ID               Name                  Email Address  \
0        1      Ronald Murphy      williamholland@example.com
1        2        Scott Allen             scott22@example.org
2        3   Jonathan Parrish            brooke16@example.org
3        4     Megan Williams         elizabeth31@example.net
4        5      Kathryn Brown   pattersonalexandra@example.org

            Username Date of Birth  Gender        Location  \
0      williamholland    1953-06-03    Male  Rebeccachester
1             scott22    1978-07-08    Male   Mcphersonview
2            brooke16    1994-12-06  Female       Youngfort
3         elizabeth31    1964-12-22  Female    Feliciashire
4  pattersonalexandra    1961-06-04    Male    Port Deborah

  Membership Start Date Membership End Date Subscription Plan  \
0            2024-01-15          2025-01-14            Annual
1            2024-01-07          2025-01-06           Monthly
2            2024-04-13          2025-04-13           Monthly
3            2024-01-24          2025-01-23           Monthly
4            2024-02-14          2025-02-13            Annual

  Payment Information Renewal Status Usage Frequency Purchase History  \
0          Mastercard         Manual         Regular      Electronics
1                Visa         Manual         Regular      Electronics
2          Mastercard         Manual         Regular            Books
3                Amex     Auto-renew         Regular      Electronics
4                Visa     Auto-renew        Frequent         Clothing

  Favorite Genres Devices Used Engagement Metrics  Feedback/Ratings  \
0     Documentary     Smart TV             Medium               3.6
1          Horror   Smartphone             Medium               3.8
2          Comedy     Smart TV                Low               3.3
3     Documentary     Smart TV               High               3.3
4           Drama     Smart TV                Low               4.3

   Customer Support Interactions
0                              3
1                              7
2                              8
3                              7
4                              1
```

[8]:
```python
# Check data types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 19 columns):
```

```
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   User ID                      2500 non-null   int64
 1   Name                         2500 non-null   object
 2   Email Address                2500 non-null   object
 3   Username                     2500 non-null   object
 4   Date of Birth                2500 non-null   object
 5   Gender                       2500 non-null   object
 6   Location                     2500 non-null   object
 7   Membership Start Date        2500 non-null   object
 8   Membership End Date          2500 non-null   object
 9   Subscription Plan            2500 non-null   object
 10  Payment Information          2500 non-null   object
 11  Renewal Status               2500 non-null   object
 12  Usage Frequency              2500 non-null   object
 13  Purchase History             2500 non-null   object
 14  Favorite Genres              2500 non-null   object
 15  Devices Used                 2500 non-null   object
 16  Engagement Metrics           2500 non-null   object
 17  Feedback/Ratings             2500 non-null   float64
 18  Customer Support Interactions 2500 non-null   int64
dtypes: float64(1), int64(2), object(16)
memory usage: 371.2+ KB
```

[9]:
```python
# Convert columns to datetime
date_columns = ['Date of Birth', 'Membership Start Date', 'Membership End Date']
data[date_columns] = data[date_columns].apply(pd.to_datetime)
```

[10]:
```python
## Check to see if any email addresses are duplicated
duplicate_email_rows = data[data.duplicated(subset=['Email Address'],␣
↪keep=False)]

duplicate_email_rows.head(10)

# Because this is a sample dataset and the Names are different for each␣
↪duplicated email address or username, I'll ignore this and leave them all in.
```

[10]:
```
     User ID               Name              Email Address      Username  \
6          7  Benjamin Marshall  michaellewis@example.net  michaellewis
33        34  Whitney Underwood        ujones@example.com        ujones
84        85         Keith Baker        ujones@example.com        ujones
107      108       Grant Jensen       tyler29@example.com       tyler29
120      121    Lisa Washington       dbailey@example.net       dbailey
129      130     Jordan Jackson        ubrown@example.org        ubrown
147      148         Troy Smith        john96@example.net        john96
175      176    Alexandra James     twilliams@example.com     twilliams
401      402      Olivia Harper     twilliams@example.com     twilliams
```

443	444	Brent Key	sburke@example.com	sburke

	Date of Birth	Gender	Location	Membership Start Date	\
6	2003-02-09	Male	Carlsonchester	2024-04-08	
33	2002-07-23	Female	New Jenniferport	2024-03-30	
84	1972-07-11	Female	Langton	2024-02-03	
107	1944-07-10	Female	East Mark	2024-04-11	
120	1991-11-03	Male	East Charlotte	2024-03-02	
129	1953-09-28	Female	Lake Amystad	2024-01-05	
147	1980-06-26	Male	West Jennifer	2024-03-27	
175	1986-07-07	Female	Lake Codymouth	2024-02-13	
401	2004-06-22	Female	Port Charles	2024-02-27	
443	1957-04-24	Male	Gonzalezland	2024-01-08	

	Membership End Date	Subscription Plan	Payment Information	Renewal Status	\
6	2025-04-08	Monthly	Amex	Auto-renew	
33	2025-03-30	Annual	Mastercard	Auto-renew	
84	2025-02-02	Annual	Amex	Manual	
107	2025-04-11	Monthly	Amex	Manual	
120	2025-03-02	Annual	Visa	Manual	
129	2025-01-04	Annual	Mastercard	Manual	
147	2025-03-27	Monthly	Visa	Manual	
175	2025-02-12	Monthly	Mastercard	Auto-renew	
401	2025-02-26	Annual	Amex	Auto-renew	
443	2025-01-07	Annual	Amex	Manual	

	Usage Frequency	Purchase History	Favorite Genres	Devices Used	\
6	Frequent	Clothing	Sci-Fi	Tablet	
33	Regular	Electronics	Horror	Smartphone	
84	Occasional	Clothing	Horror	Smartphone	
107	Regular	Clothing	Action	Tablet	
120	Frequent	Books	Drama	Smartphone	
129	Occasional	Electronics	Sci-Fi	Tablet	
147	Regular	Books	Horror	Smart TV	
175	Frequent	Books	Horror	Smart TV	
401	Frequent	Electronics	Documentary	Smartphone	
443	Frequent	Books	Romance	Smart TV	

	Engagement Metrics	Feedback/Ratings	Customer Support Interactions
6	Medium	4.4	10
33	Low	4.5	10
84	Medium	3.0	1
107	Medium	3.2	7
120	High	4.2	2
129	Medium	4.4	3
147	Low	4.3	0
175	High	3.3	8

```
401              High            4.0                              6
443            Medium            3.5                              5
```

[11]:
```
# Remove columns that I won't need
data = data.drop(columns = ['User ID', 'Name', 'Email Address', 'Username'])
```

[12]:
```
# Create an age column based on years since the date of birth
current_date = datetime.now()
current_year = current_date.year
data['Age'] = current_year - data['Date of Birth'].dt.year
data.head()
```

[12]:
```
  Date of Birth  Gender        Location Membership Start Date  \
0    1953-06-03    Male  Rebeccachester            2024-01-15
1    1978-07-08    Male   Mcphersonview            2024-01-07
2    1994-12-06  Female       Youngfort            2024-04-13
3    1964-12-22  Female     Feliciashire            2024-01-24
4    1961-06-04    Male     Port Deborah            2024-02-14

  Membership End Date Subscription Plan Payment Information Renewal Status  \
0          2025-01-14            Annual          Mastercard         Manual
1          2025-01-06           Monthly                Visa         Manual
2          2025-04-13           Monthly          Mastercard         Manual
3          2025-01-23           Monthly                Amex     Auto-renew
4          2025-02-13            Annual                Visa     Auto-renew

  Usage Frequency Purchase History Favorite Genres Devices Used  \
0         Regular      Electronics     Documentary     Smart TV
1         Regular      Electronics          Horror   Smartphone
2         Regular            Books          Comedy     Smart TV
3         Regular      Electronics     Documentary     Smart TV
4        Frequent         Clothing           Drama     Smart TV

  Engagement Metrics  Feedback/Ratings  Customer Support Interactions  Age
0             Medium               3.6                              3   71
1             Medium               3.8                              7   46
2                Low               3.3                              8   30
3               High               3.3                              7   60
4                Low               4.3                              1   63
```

[13]:
```
# Check month distribution for memberships started
data['Month'] = data['Membership Start Date'].dt.month
data['Month_Text'] = data['Membership Start Date'].dt.month_name().str.
 ↪slice(stop=3)

# Group by `Month` and `Month_Text`, sum it, and sort. Assign result to new␣
 ↪DataFrame
```

```
data_by_month = data.groupby(['Month', 'Month_Text']).size().
↪reset_index(name='Count').sort_values('Month').head(12)
data_by_month

# April is lower than the rest as the data only goes through 4-13-2024
```

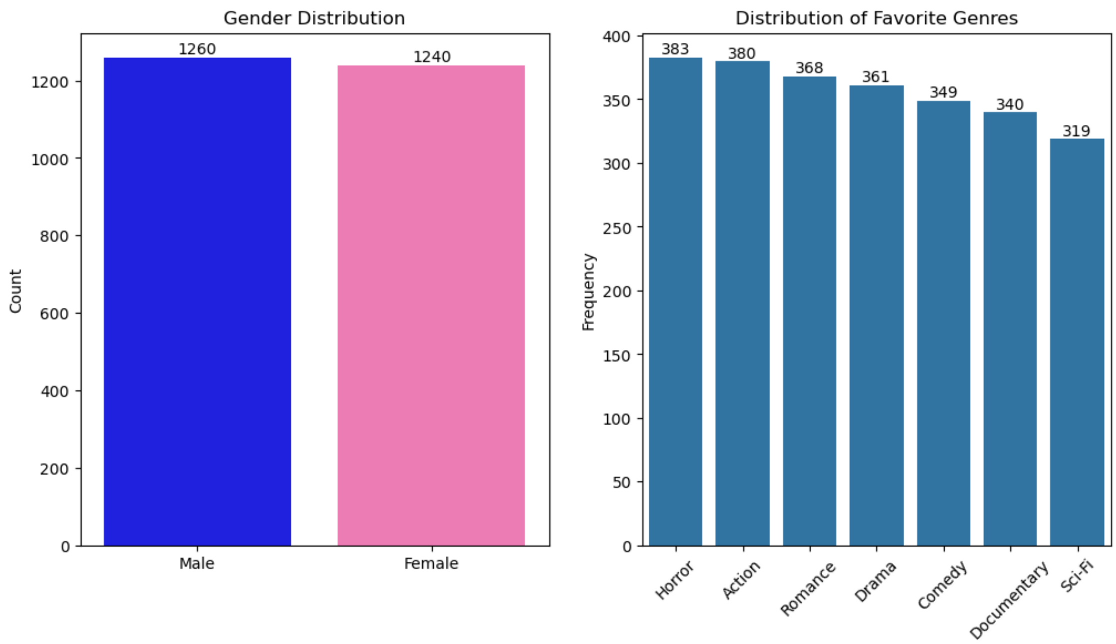
[13]:

	Month	Month_Text	Count
0	1	Jan	773
1	2	Feb	651
2	3	Mar	744
3	4	Apr	332

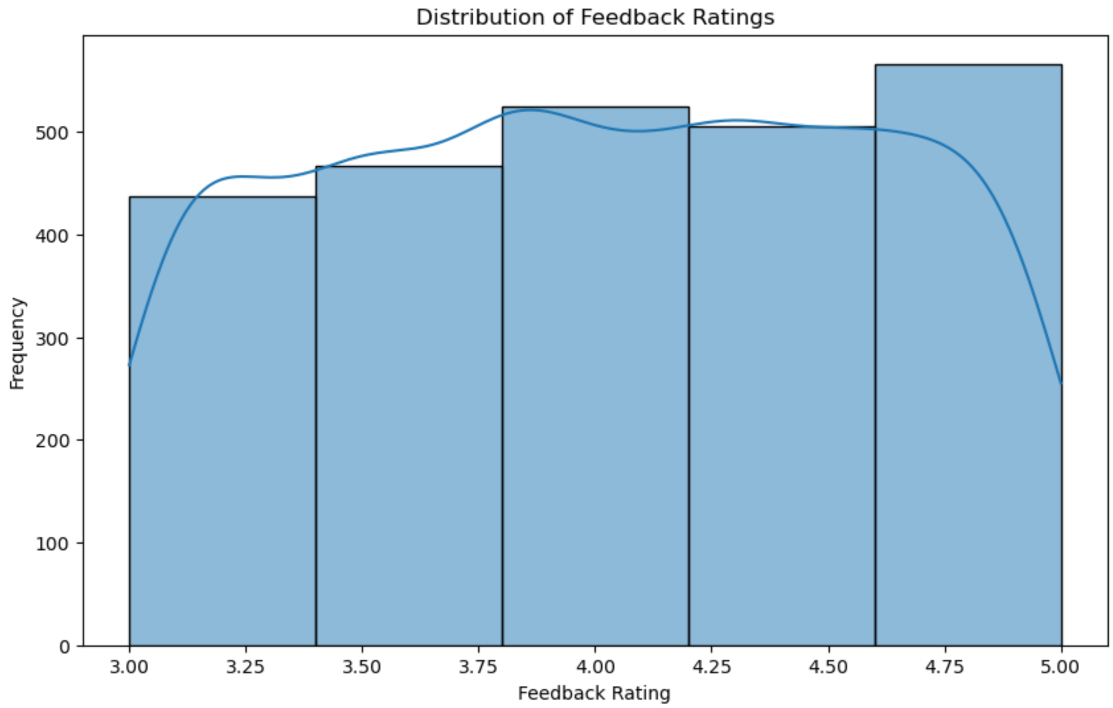
3 Exploratory Data Analysis (EDA)

[15]:

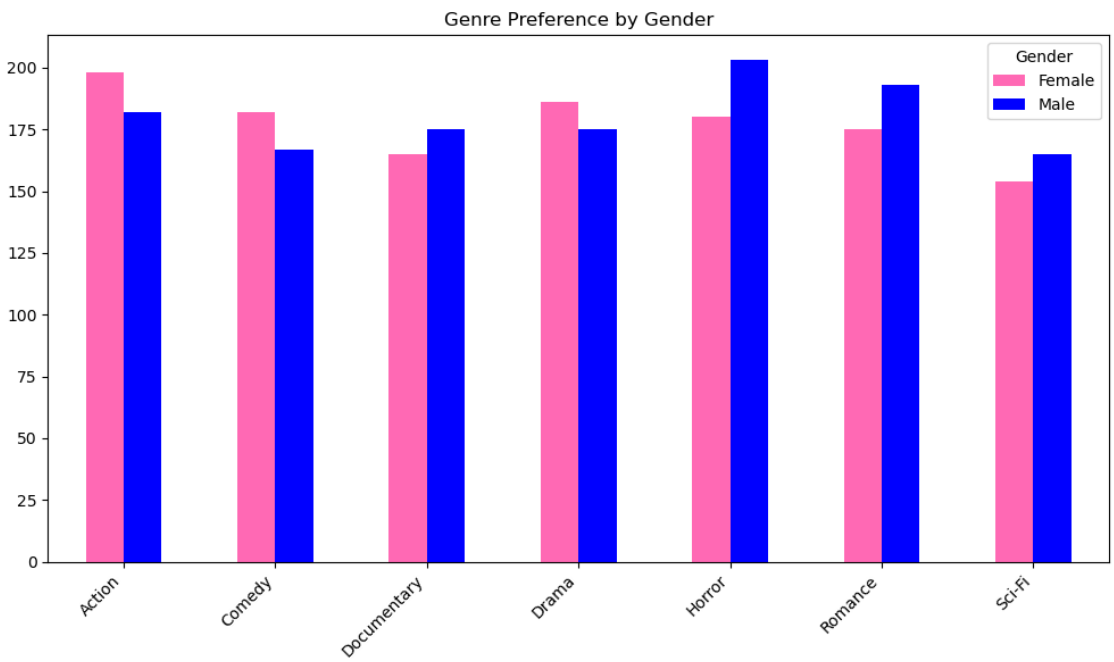
```
# Create subplots for two plots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12,6))

# Plot 1: Bar plot of Gender Distribution
gender_counts = data['Gender'].value_counts()
sns.barplot(x=gender_counts.index, y=gender_counts.values, ax=axes[0],␣
↪palette=['blue', 'hotpink'])
axes[0].set_title('Gender Distribution')
axes[0].set_xlabel('')
axes[0].set_ylabel('Count')

# Add counts above the bars
for index, value in enumerate(gender_counts.values):
    axes[0].text(index, value + 0.1, str(value), ha='center', va='bottom')

# Plot 2: Bar plot of Favorite Genres
genre_counts = data['Favorite Genres'].value_counts()
sns.barplot(x=genre_counts.index, y=genre_counts.values, ax=axes[1])
axes[1].set_title('Distribution of Favorite Genres')
axes[1].set_xlabel('')
axes[1].set_ylabel('Frequency')
axes[1].tick_params(axis='x', rotation=45)  # Rotate x-axis labels for better␣
↪readability

# Add counts above the bars
for index, value in enumerate(genre_counts.values):
    axes[1].text(index, value + 0.1, str(value), ha='center', va='bottom')

plt.show()
```



[16]:
```
# Plot: Histogram of Feedback Ratings
plt.figure(figsize=(10,6))

plot = sns.histplot(data=data, x='Feedback/Ratings', bins=5, kde=True)
plot.set_title('Distribution of Feedback Ratings')
plot.set_xlabel('Feedback Rating')
plot.set_ylabel('Frequency')

plt.show()
```



[17]:
```
# Group data by gender and favorite genres, and calculate counts
genre_summary = data.groupby(['Gender', 'Favorite Genres']).size().
    ↪reset_index(name='Count')
#print(genre_summary)


# Pivot the DataFrame to have favorite genres as columns
genre_summary_pivot = genre_summary.pivot(index='Favorite Genres',␣
    ↪columns='Gender', values='Count').fillna(0)

# Plot
genre_summary_pivot.plot(kind='bar', figsize=(10, 6), color=['hotpink', 'blue'])
plt.title('Genre Preference by Gender')
plt.xlabel('')
plt.ylabel('')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Gender')
plt.tight_layout()
plt.show()
```



[18]:
```
# Check the options and amounts of usage frequency
data['Usage Frequency'].value_counts()
```

[18]:
```
Usage Frequency
Frequent      851
Regular       827
Occasional    822
Name: count, dtype: int64
```

4 Model Development

[20]:
```
# Create a copy of the dataset to manipulate for the model
data_model = data.copy()


# Drop more features for easier reading and usage, now that the initial EDA is␣
↪complete
data_model = data_model.drop(columns = ['Gender',
                                        'Location',
                                        'Membership Start Date',
                                        'Membership End Date',
                                        'Date of Birth',
                                        'Month_Text',
                                        'Month',
```

```
                                            'Feedback/Ratings',
                                            'Payment Information',
                                            'Age'])
```

[21]:
```
# Define a mapping dictionary for usage freq.
usage_frequency_mapping = {
    'Occasional': 1,
    'Regular': 2,
    'Frequent': 3
}

# Define a mapping dictionary for engagement metrics
engagement_metrics_mapping = {
    'Low': 1,
    'Medium': 2,
    'High': 3
}



# Apply the mapping to the "Usage Frequency" and "Engagement Metrics"
data_model['Usage Frequency'] = data_model['Usage Frequency'].
 ↪map(usage_frequency_mapping)

data_model['Engagement Metrics'] = data_model['Engagement Metrics'].
 ↪map(engagement_metrics_mapping)
```

[22]:
```
# Define the columns to convert to dummy variables
columns_to_convert = ['Renewal Status',
                      'Devices Used',
                      'Purchase History',
                      'Favorite Genres',
                      'Customer Support Interactions']

# Create dummy variables for the specified columns
dummy_variables = pd.get_dummies(data_model[columns_to_convert],␣
 ↪drop_first=True, prefix='', prefix_sep='', dtype=int)

# Concatenate the dummy variables with the original dataset
data_model = pd.concat([data_model, dummy_variables], axis=1)

# Drop the original categorical columns
data_model.drop(columns=columns_to_convert, inplace=True)
```

[23]:
```
# Check back on the data set
data_model.head()
```

[23]:

```
  Subscription Plan  Usage Frequency  Engagement Metrics  Manual  Smartphone  \
0            Annual                2                   2       1           0
1           Monthly                2                   2       1           1
2           Monthly                2                   1       1           0
3           Monthly                2                   3       0           0
4            Annual                3                   1       0           0

   Tablet  Clothing  Electronics  Comedy  Documentary  Drama  Horror  Romance  \
0       0         0            1       0            1      0       0        0
1       0         0            1       0            0      0       1        0
2       0         0            0       1            0      0       0        0
3       0         0            1       0            1      0       0        0
4       0         1            0       0            0      1       0        0

   Sci-Fi
0       0
1       0
2       0
3       0
4       0
```

[24]:

```python
# Split the data into features (X) and target variable (y)
X = data_model.drop(columns=['Subscription Plan'])  # Features
y = data_model['Subscription Plan']  # Target variable

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = .
 ↪8,test_size=0.2, random_state=42)




# Define a list of models
models = [
    LogisticRegression(max_iter=1000, random_state=42),
    DecisionTreeClassifier(random_state=42),
    RandomForestClassifier(random_state=42)
]

# Iterate over each model
for model in models:

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

    # Make predictions on the testing data
    y_pred = model.predict(X_test)
```

```
    # Evaluate the model's performance
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='Annual')
    recall = recall_score(y_test, y_pred, pos_label='Annual')
    f1 = f1_score(y_test, y_pred, pos_label = "Annual")

    # Print the model's performance
    print(f"Model: {model.__class__.__name__}, Accuracy: {accuracy:.5f},␣
↪Precision: {precision:.5f}, Recall: {recall:.5f}, F1: {f1:.5f}")
```

```
Model: LogisticRegression, Accuracy: 0.52200, Precision: 0.51515, Recall:
0.68273, F1: 0.58722
Model: DecisionTreeClassifier, Accuracy: 0.51800, Precision: 0.51418, Recall:
0.58233, F1: 0.54614
Model: RandomForestClassifier, Accuracy: 0.53800, Precision: 0.53689, Recall:
0.52610, F1: 0.53144
```

[25]:

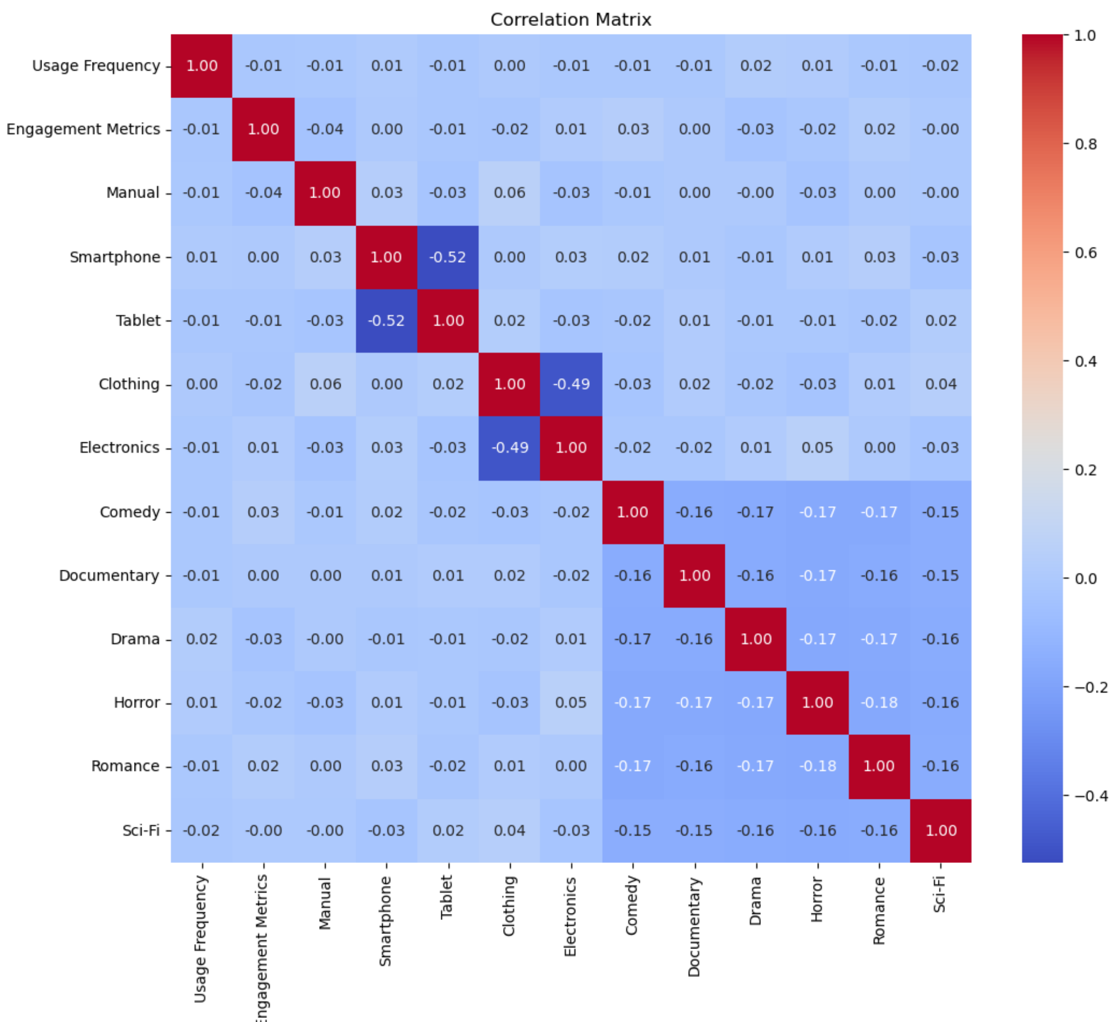
```
# Retest the data to determine if a different split improves accuracy

# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.7,␣
↪test_size=0.3, random_state=42)

models = [
    LogisticRegression(max_iter=1000, random_state=42),
    DecisionTreeClassifier(random_state=42),
    RandomForestClassifier(random_state=42)
]

# Iterate over each model
for model in models:

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='Annual')
    recall = recall_score(y_test, y_pred, pos_label='Annual')
    f1 = f1_score(y_test, y_pred, pos_label = "Annual")

    # Print the model's performance
    print(f"Model: {model.__class__.__name__}, Accuracy: {accuracy:.5f},␣
↪Precision: {precision:.5f}, Recall: {recall:.5f}, F1: {f1:.5f}")
```

```
Model: LogisticRegression, Accuracy: 0.49200, Precision: 0.51382, Recall:
0.56743, F1: 0.53930
Model: DecisionTreeClassifier, Accuracy: 0.54667, Precision: 0.56265, Recall:
0.60560, F1: 0.58333
Model: RandomForestClassifier, Accuracy: 0.52800, Precision: 0.55256, Recall:
```

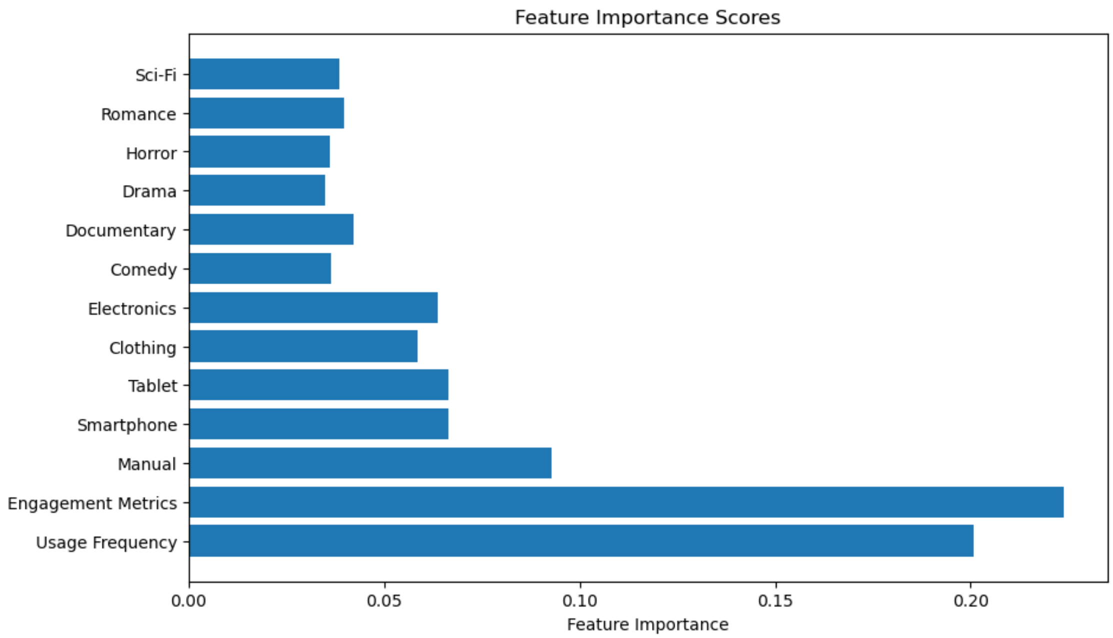
```
0.52163, F1: 0.53665
```

[26]:
```
# The Decision tree has the highest accuracy, precision, and recall.
↪Outperforming the other models and performing better in the 70-30 split
↪compared to the 80-20 split, most likely due to more data learn more
↪effective.
```

[27]:
```
# Calculate correlation matrix
corr_matrix = X.corr()

# Plot heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



[28]:
```
# Get feature importances for random forest model
feature_importances = models[2].feature_importances_

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(X.columns, feature_importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importance Scores')
plt.show()
```



4.0.1 Testing new hyperparameters utilizing GridSearch to see if a better random forest can be created

[30]:
```
# Separate into labels and features
y = data_model['Subscription Plan']
X = data_model.drop('Subscription Plan', axis=1)

# train, val, test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
↪random_state = 42)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size = 0.25,␣
↪random_state = 42)

# Set hyperparamaters
cv_params = {'n_estimators' : [50,100],
             'max_depth' : [10,50],
```

```
                'min_samples_leaf' : [0.5,1],
                'min_samples_split' : [0.001, 0.01],
                'max_features' : ["sqrt"],
                'max_samples' : [.5,.9]}

# Split index
split_index = [0 if x in X_val.index else -1 for x in X_train.index]
custom_split = PredefinedSplit(split_index)

# Instantiate the model
rf = RandomForestClassifier(random_state=42)

# search over parameters
rf_val = GridSearchCV(rf, cv_params, cv=custom_split, refit='f1', n_jobs = -1,␣
 ↪verbose = 1)

# Fit the model
rf_val.fit(X_train, y_train)
```

```
Fitting 1 folds for each of 32 candidates, totalling 32 fits
```

[30]:
```
GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, …, -1, -1])),
             estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [10, 50], 'max_features': ['sqrt'],
                         'max_samples': [0.5, 0.9],
                         'min_samples_leaf': [0.5, 1],
                         'min_samples_split': [0.001, 0.01],
                         'n_estimators': [50, 100]},
             refit='f1', verbose=1)
```

[31]:
```
#obtain optimal paramaters
rf_val.best_params_
```

[31]:
```
{'max_depth': 50,
 'max_features': 'sqrt',
 'max_samples': 0.9,
 'min_samples_leaf': 1,
 'min_samples_split': 0.001,
 'n_estimators': 100}
```

[32]:
```
# Use optimal parameters on GridSearchCV.
rf_opt = RandomForestClassifier(n_estimators = 100, max_depth = 50,
                                min_samples_leaf = 1, min_samples_split = 0.001,
                                max_features="sqrt", max_samples = 0.9,␣
 ↪random_state = 42)
```

[33]:
```
# Fit the optimal model.
```

```
rf_opt.fit(X_train, y_train)
```

[33]:
```
RandomForestClassifier(max_depth=50, max_samples=0.9, min_samples_split=0.001,
                       random_state=42)
```

[34]:
```
# Predict on test set
y_pred = rf_opt.predict(X_test)

# Get scores
rf_pc_test = precision_score(y_test, y_pred, pos_label = "Annual")
rf_rc_test = recall_score(y_test, y_pred, pos_label = "Annual")
rf_ac_test = accuracy_score(y_test, y_pred)
rf_f1_test = f1_score(y_test, y_pred, pos_label = "Annual")

# Print results
table = pd.DataFrame({'Model': ["Decision Tree","Tuned Random Forest"],
                      'F1':  [0.58333, rf_f1_test],
                      'Recall': [0.60560, rf_rc_test],
                      'Precision': [0.56265, rf_pc_test],
                      'Accuracy': [0.54667, rf_ac_test]
                    }
                   )
table
```

[34]:

	Model	F1	Recall	Precision	Accuracy
0	Decision Tree	0.583330	0.605600	0.562650	0.54667
1	Tuned Random Forest	0.547655	0.555215	0.540299	0.52160

4.0.2 We can see that the Decision Tree with 75/25 split has better scores all around

[36]:
```
data_model_binary = pd.get_dummies(data_model, drop_first=True, dtype=int)
data_model_binary.head()
```

[36]:

	Usage Frequency	Engagement Metrics	Manual	Smartphone	Tablet	Clothing	\
0	2	2	1	0	0	0	
1	2	2	1	1	0	0	
2	2	1	1	0	0	0	
3	2	3	0	0	0	0	
4	3	1	0	0	0	1	

	Electronics	Comedy	Documentary	Drama	Horror	Romance	Sci-Fi	\
0	1	0	1	0	0	0	0	
1	1	0	0	0	1	0	0	
2	0	1	0	0	0	0	0	
3	1	0	1	0	0	0	0	
4	0	0	0	1	0	0	0	

```
   Subscription Plan_Monthly
0                          0
1                          1
2                          1
3                          1
4                          0
```

[37]:
```
# XGBoost

# Separate into labels and features
y = data_model_binary['Subscription Plan_Monthly']
X = data_model_binary.drop('Subscription Plan_Monthly', axis=1)

# train, val, test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
 ↪random_state = 42)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size = 0.25,␣
 ↪random_state = 42)

# Set hyperparamaters
cv_params = {'learning_rate': [0.1],
             'max_depth': [8],
             'min_child_weight': [2],
             'n_estimators': [500]
             }
xgb = XGBClassifier(objective='binary:logistic', random_state=42)
```

[38]:
```
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),

}

xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')
```

[39]:
```
%%time
# fit the GridSearch model to training data

xgb_cv.fit(X_train, y_train)
```

```
CPU times: total: 15.6 s
Wall time: 1.13 s
```

[39]:
```
GridSearchCV(cv=4,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
```

```
                                       colsample_bynode=None,
                                       colsample_bytree=None, device=None,
                                       early_stopping_rounds=None,
                                       enable_categorical=False, eval_metric=None,
                                       feature_types=None, gamma=None,
                                       grow_policy=None, importance_type=None,
                                       interaction_constraints=None,
                                       learning_rate=None,…
             param_grid={'learning_rate': [0.1], 'max_depth': [8],
                         'min_child_weight': [2], 'n_estimators': [500]},
             refit='f1',
             scoring={'accuracy': make_scorer(accuracy_score,
response_method='predict'),
                      'f1': make_scorer(f1_score, response_method='predict'),
                      'precision': make_scorer(precision_score,
response_method='predict'),
                      'recall': make_scorer(recall_score,
response_method='predict')})
```

[40]:
```python
# Examine best score and paramaters
print(xgb_cv.best_score_)

print(xgb_cv.best_params_)
```

```
0.4903714701922771
{'learning_rate': 0.1, 'max_depth': 8, 'min_child_weight': 2, 'n_estimators':
500}
```

[41]:
```python
# Predict on test set
y_pred = xgb_cv.predict(X_test)

# Get scores
xgb_pc_test = precision_score(y_test, y_pred)
xgb_rc_test = recall_score(y_test, y_pred)
xgb_ac_test = accuracy_score(y_test, y_pred)
xgb_f1_test = f1_score(y_test, y_pred)

# Print results
table = pd.DataFrame({'Model': ["Decision Tree","Tuned Random Forest",␣
 ↪"XGBoost"],
                        'F1':  [0.58333, rf_f1_test, xgb_pc_test],
                        'Recall': [0.60560, rf_rc_test, xgb_rc_test],
                        'Precision': [0.56265, rf_pc_test, xgb_ac_test],
                        'Accuracy': [0.54667, rf_ac_test, xgb_f1_test]
                      }
                    )
table
```

[41]:

	Model	F1	Recall	Precision	Accuracy
0	Decision Tree	0.583330	0.605600	0.562650	0.546670
1	Tuned Random Forest	0.547655	0.555215	0.540299	0.521600
2	XGBoost	0.483871	0.501672	0.505600	0.492611

5 Model Selection

5.1 Results and Evaluation

Tree-based Machine Learning and Gradient Boosting

After conducting feature engineering, the decision tree model achieved f1-score of 58.3%, recall of 60.6%, precision of 56.3%, and accuracy of 54.7%, on the test set.

5.2 Conclusion, Recommendations, Next Steps

The models and the feature importances extracted from the models confirm that employees at the company are overworked.

To retain customers, the following recommendations could be presented to the stakeholders:

- Personalized Subscription Plans: Utilize the insights from the predictive models to offer personalized subscription plans tailored to individual user preferences, usage frequency, and engagement metrics. This can enhance user satisfaction and retention by providing plans that better suit their needs.
- Promotional Campaigns: Target promotional campaigns towards users who are more likely to churn based on the predictive models. Offer incentives or discounts to encourage them to renew their subscriptions or upgrade to higher-tier plans.
- Content Recommendations: Leverage the favorite genres and purchase history of users to provide targeted content recommendations. This can increase user engagement and satisfaction by delivering content that aligns with their interests.
- Customer Support Enhancement: Identify users who have had frequent interactions with customer support and proactively address their concerns or issues. Improving the customer support experience can lead to higher retention rates and positive feedback.
- Optimized Renewal Strategies: Implement strategies to optimize subscription renewal processes, such as providing seamless renewal options, offering incentives for automatic renewal, or sending timely reminders before subscription expiration dates.
- Continuous Monitoring and Feedback: Regularly monitor user feedback and engagement metrics to identify evolving preferences and trends. Incorporate user feedback into product development and subscription offerings to ensure continued relevance and satisfaction.
- Further Model Refinement: Continuously refine and update the predictive models based on new data and feedback. This can improve the accuracy and effectiveness of the models over time, leading to better decision-making and outcomes.

- More data: This model could be significantly improved with more datapoints.