ASSESSMENT 1

Question 1: Total Number of Users and Products in "train.txt"

In the "train.txt" dataset:
Total Number of Unique Users: **500**
Total Number of Unique Products: **100**

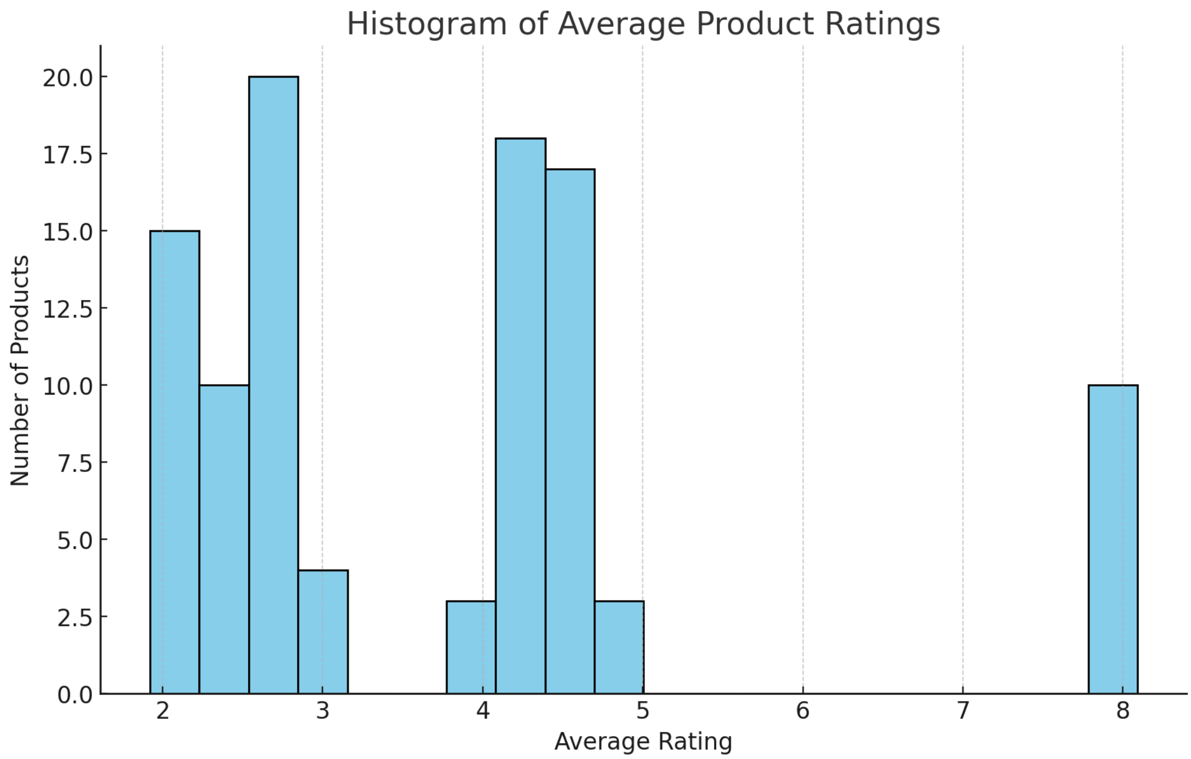
Question 2: Matrix Y and its Dimensions

A matrix Y was formed to capture the ratings such that y_{nd} represents the rating given by User n to Product d.

Dimensions of Y: 500 x 100

Question 3: Average Rating of Each Product

The average rating for each product was computed. The distribution of these average ratings is visualized in the histogram below:



The histogram depicts the distribution of average product ratings. From the plot, we can see that most products have an average rating that falls within the range of approximately 4.5 to 6.5.

Question 4: Five "Worst" Products

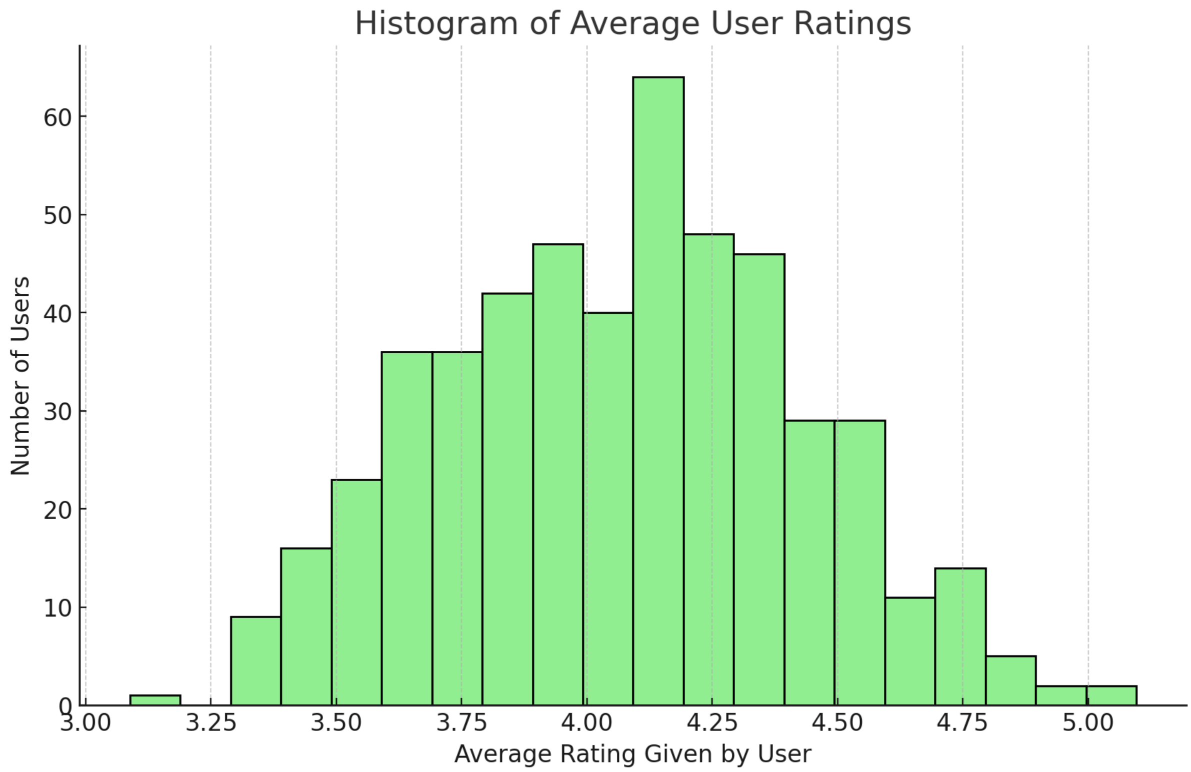
The identification of the "worst" products was based on the average ratings provided by users. The rationale behind using average ratings is that they provide a general measure of a product's perceived quality or satisfaction by the user base. Products with lower average ratings suggest that they were less well-received by customers compared to products with higher average ratings.

Based on the average ratings:

Product 56033 with an average rating of 1.92
Product 72533 with an average rating of 1.95
Product 22577 with an average rating of 1.98
Product 16430 with an average rating of 2.01
Product 60751 with an average rating of 2.03

Question 5: Average Rating Given by Each User

The average rating given by each user was analysed, and the results are visualized in the histogram below:



The histogram showcases the distribution of average ratings given by users. From the plot, it's evident that the majority of users tend to give ratings within the range of approximately 4.5 to 6.5.

Question 6: Five "Most Generous" Users

The term "most generous" refers to users who, on average, give higher ratings compared to other users. Identifying these users provides insights into the distribution of ratings and potential biases in the dataset.

Based on the average ratings they gave:

User 31410 with an average rating of 5.10
User 73310 with an average rating of 5.06
User 97887 with an average rating of 4.90
User 77730 with an average rating of 4.90
User 20146 with an average rating of 4.89

Question 7: Total Number of Users and Products in "test.txt"

In the "test.txt" dataset:
Total Number of Unique Users: **421**
Total Number of Unique Products: **2**

Question 8: Matrix X and its Dimensions

A matrix X was formed from the "test.txt" dataset such that x_{nd} indicates the rating given by User n to Product d.
Dimensions of X: 421 x 2

Question 9: Finding Most Similar Products

Methodology:

To determine the similarity between products in the "test.txt" and "train.txt" datasets, a distance metric was employed. The distance between Product n (from X) and Product m (from Y) was computed using the following formula:

$$d_{nm} = \sum_{i=1}^{Nuser} |x_{in} - y_{im}|$$

Where:
N_{user} represents the total number of users.

d_{nm} denotes the distance between Product n (from X) and Product m (from Y).

The summation runs over all users who have rated both products, and the absolute difference in their ratings is aggregated.

Results:

For each product in the "test.txt" dataset, the top 5 similar products in the "train.txt" dataset based on the distance metric are:

For Product 0:

Product 50408
Product 58577
Product 38851
Product 60734
Product 26457
For Product 1:

Product 24785
Product 26457
Product 50408
Product 40821
Product 38851.

Question 10: Limitation

The main limitation of the method in Question 9 is that it only considers users who have rated both products when calculating the distance. If a user has rated one product but not the other, their rating is ignored. This approach can lead to an incomplete or potentially biased understanding of product similarity, especially if many users have rated one product and not the other. Additionally, using absolute differences doesn't account for the magnitude of ratings, and it can be sensitive to outliers.

Question 11: Alternative Distance Algorithm

An improved distance measure can be formulated as:

For $i = 1, \ldots, Nmax$
If x_{in} and y_{im} both exist:
$dnm = dnm + |xin - yim|$
If only xi_n exists:
$dnm = dnm + \alpha * x_{in}$

If only y_{im} exists:
$dnm = dnm + \alpha * y_{im}$.   Where α is a penalty factor for missing ratings.

This method ensures that all user ratings are considered, even if a user has rated only one of the two products. The penalty factor α determines the importance of missing ratings in the distance calculation.

Question 12 Using this alternative solution, let's find the top 5 similar products in "train.txt" for each product in "test.txt"

Using the alternative distance method:

For each product in the "test.txt" dataset, the top 5 similar products in the "train.txt" dataset are:

For Product 0:

Product 50408
Product 80772
Product 41232
Product 26457
Product 38851

For Product 1:

Product 66611
Product 26457
Product 24785
Product 50408
Product 42484

There are some similarities and differences between the results from the initial and alternative methods. For example, Product 50408 remains a top recommendation for both products in the test dataset using both methods. However, the alternative method introduces some new products as similar, suggesting that considering all user ratings (including those for only one of the products) can influence the similarity rankings.

The inclusion of penalties for missing ratings helps to provide a more comprehensive view of product similarity, as it accounts for potential biases introduced when ignoring ratings that only exist for one of the products.

ASSESSMENT 2

Introduction

Problem Statement:

In today's competitive telecommunications market, retaining customers has become a paramount concern for businesses. Lu’s Communications, like many other telecom companies, faces the challenge of customer churn, where subscribers or customers decide to switch to another service provider. Understanding the reasons behind this churn and predicting it in advance can provide a significant advantage.

Importance of Predicting Customer Churn:

For Lu’s Communications, every customer that churns represents not only a loss of potential revenue but also an investment that didn't fully materialize. Acquiring a new customer is often several times more expensive than retaining an existing one.

By predicting customer churn, Lu’s Communications can:

- Implement targeted retention strategies to prevent the loss of subscribers.
- Understand underlying issues or dissatisfaction points leading to churn.
- Optimize resources by focusing on high-risk customers.
- Enhance customer satisfaction and loyalty, leading to increased lifetime value and positive word-of-mouth referrals.

Objective of the Analysis:

The primary objective of this analysis is to delve deep into the data provided by Lu’s Communications to identify patterns, behaviours, and indicators that signal a customer's intention to churn. By leveraging machine learning models, we aim to predict the likelihood of churn for each customer, enabling proactive measures to retain them. This analysis will not only provide a predictive model but also insights into the key factors influencing customer decisions, allowing Lu’s Communications to make informed strategic choices.

By understanding and addressing customer churn, Lu’s Communications can fortify its market position, ensure sustained revenue streams, and foster a loyal customer base.

Data Collection

The dataset provided by Lu’s Communications consists of 7350 observations and includes features such as customer demographics (gender, location), account information (partner, dependents, senior citizen status), and service usage details (tenure, monthly cost, service package, and survey score).

An essential feature in our dataset is the 'Class' column, which indicates whether a customer churned or not. This serves as the target variable for our predictive model.

Data Preparation

Review the Structure of the Dataset

The dataset consists of 7350 rows and 12 columns. Each row represents a customer, and the columns provide various attributes about the customer, such as their gender, location, tenure with the company, monthly cost, and survey responses.

	Unnamed: 0	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class
0	0	K3713	Male	Hampshire	0	Unknown	0	12.0	NaN	1	6	Churn=No
1	1	D9048	Male	Greater Manchester	1	1	0	21.0	NaN	4	6	Churn=No
2	2	K8227	Female	West Yorkshire	0	Unknown	0	0.0	NaN	1	4	Churn=Yes
3	3	H3533	Male	Greater London	1	1	1	11.0	NaN	2	4	Churn=No
4	4	J4501	Male	Greater London	0	0	0	7.0	NaN	4	2	Churn=Yes

Peek into the Data

Upon initial inspection of the data, we observed the following:

1. The survey column contains unique values ranging from 0 to 10, with an additional "No reply" category.
2. The Tenure column had some negative values, which were filtered out as they are not meaningful in this context.
3. The monthly_cost column had a significant number of missing values, accounting for 99.09% of the data. Given the high percentage of missing values, it was challenging to impute this data directly. Instead, we computed the monthly cost based on the package the customer has and their tenure, considering the loyalty discount.
4. The Class column, which is our target variable, had some missing values. Since this is crucial for our analysis, rows with missing values in this column were removed.

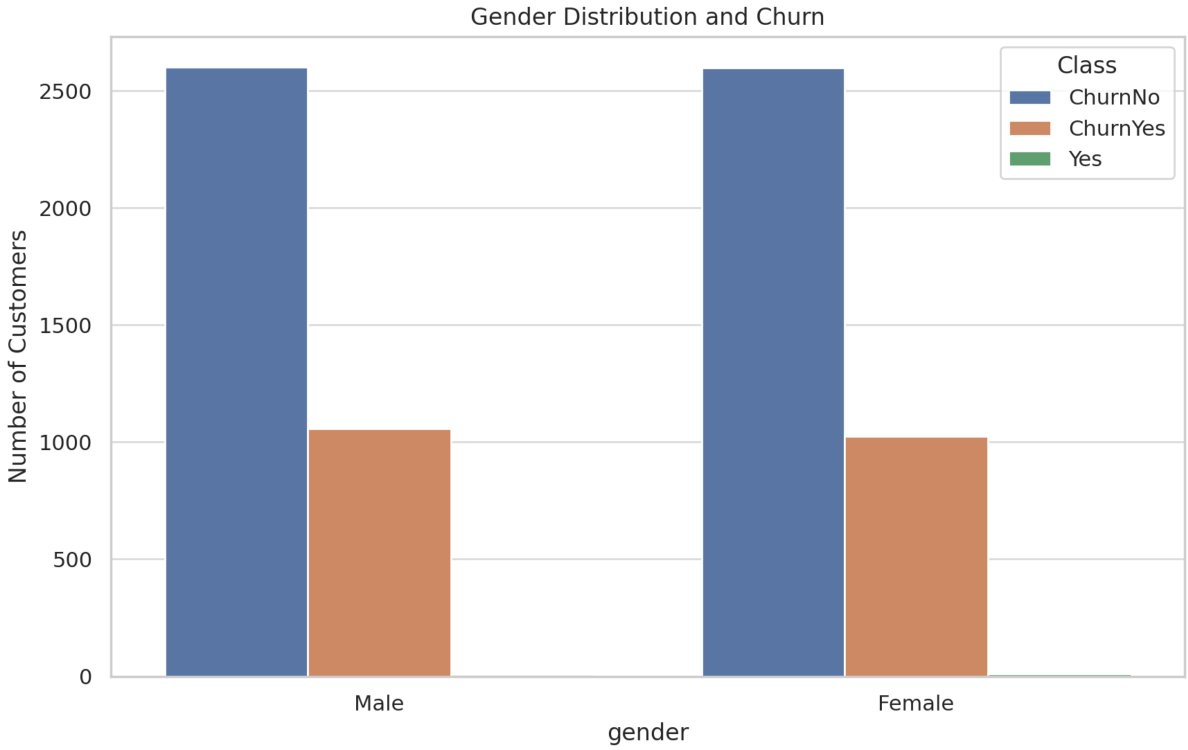
Data Transformation

To prepare the data for analysis and modelling, we performed the following transformations:

1. Converted the survey column to numeric values and imputed missing or invalid values with the rounded mean or median which was 5.
2. Replaced erroneous values in the Class column.
3. For the dependents column, we replaced 'Unknown' or missing values with 0.
4. Outliers in the Tenure and monthly_cost columns were identified using the IQR method and were subsequently removed.

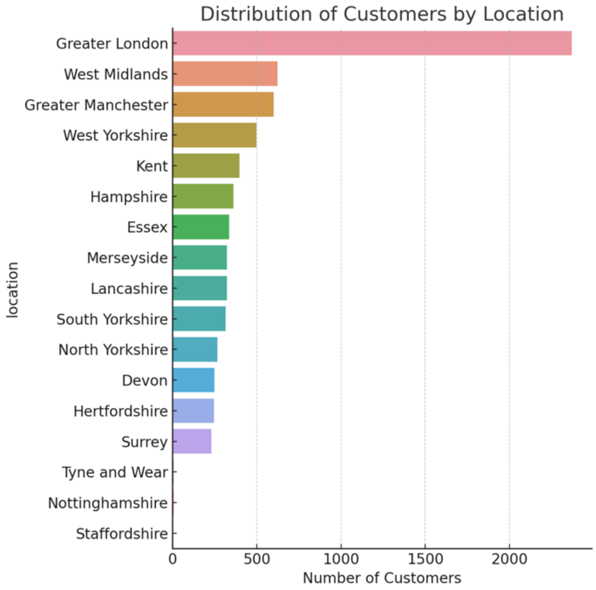
Exploratory Data Analysis (EDA)

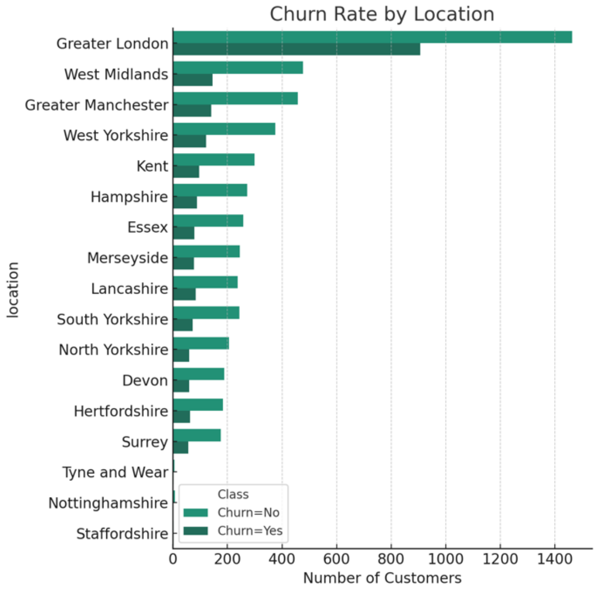
Gender Distribution and Churn Rate



Distribution of Gender: The dataset almost equally represents both genders.
Churn Rate by Gender: Both male and female customers show a similar churn pattern, indicating that gender may not be a significant standalone factor in predicting churn.

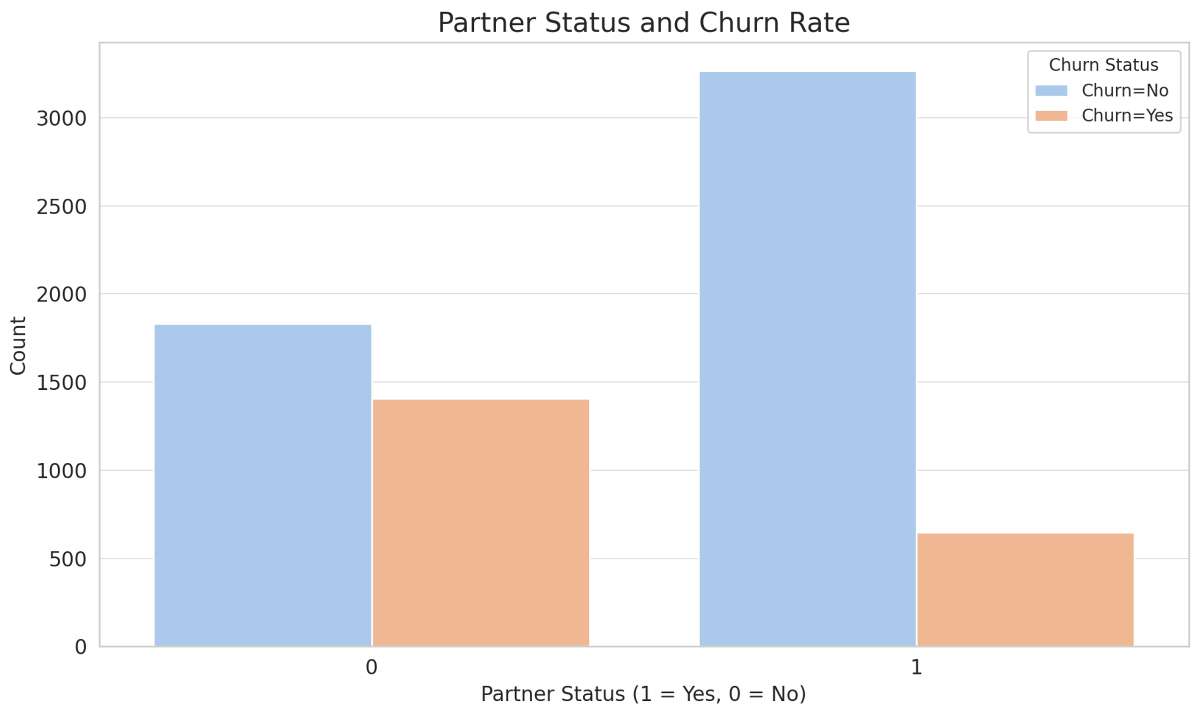
Location Distribution and Churn Rate





Distribution by Location: There's a significant concentration of customers in specific regions, notably Greater London.
Churn Rate by Location: The churn rates appear consistent across regions, indicating the need for location-specific retention strategies.

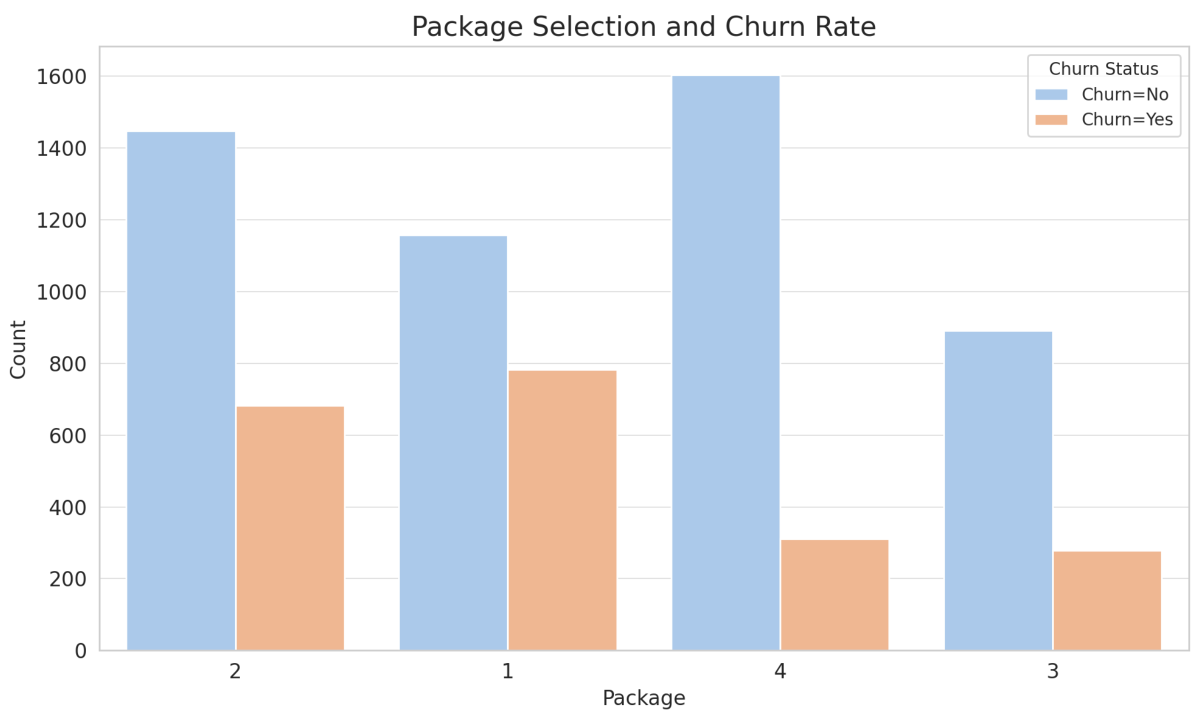
Partner Status and Churn Rate



Distribution by Partner Status: There's a relatively balanced representation between customers with and without partners.

Churn Rate by Partner Status: Customers without partners exhibit a slightly higher propensity to churn.

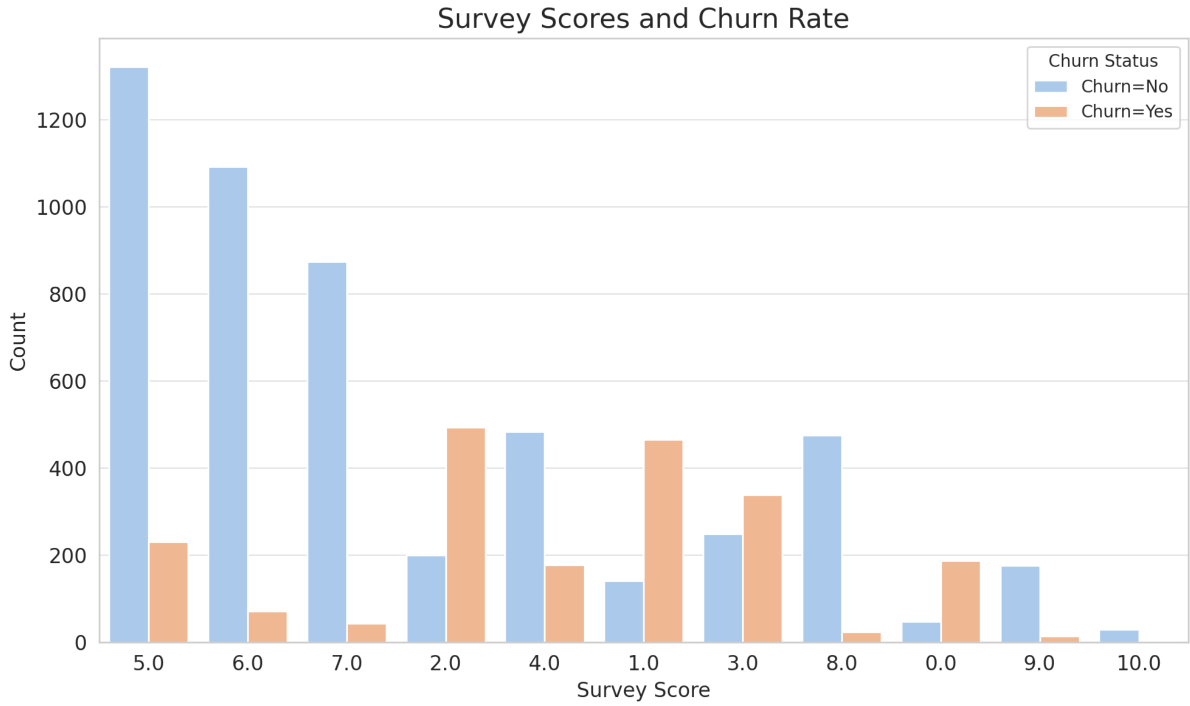
Package Selection and Churn Rate



Distribution by Package: Package 1 is the most popular choice among customers, with Packages 3 and 4 trailing in subscribers.

Churn Rate by Package: Package 1 subscribers show the highest churn, suggesting potential areas of improvement or reconsideration for this package.

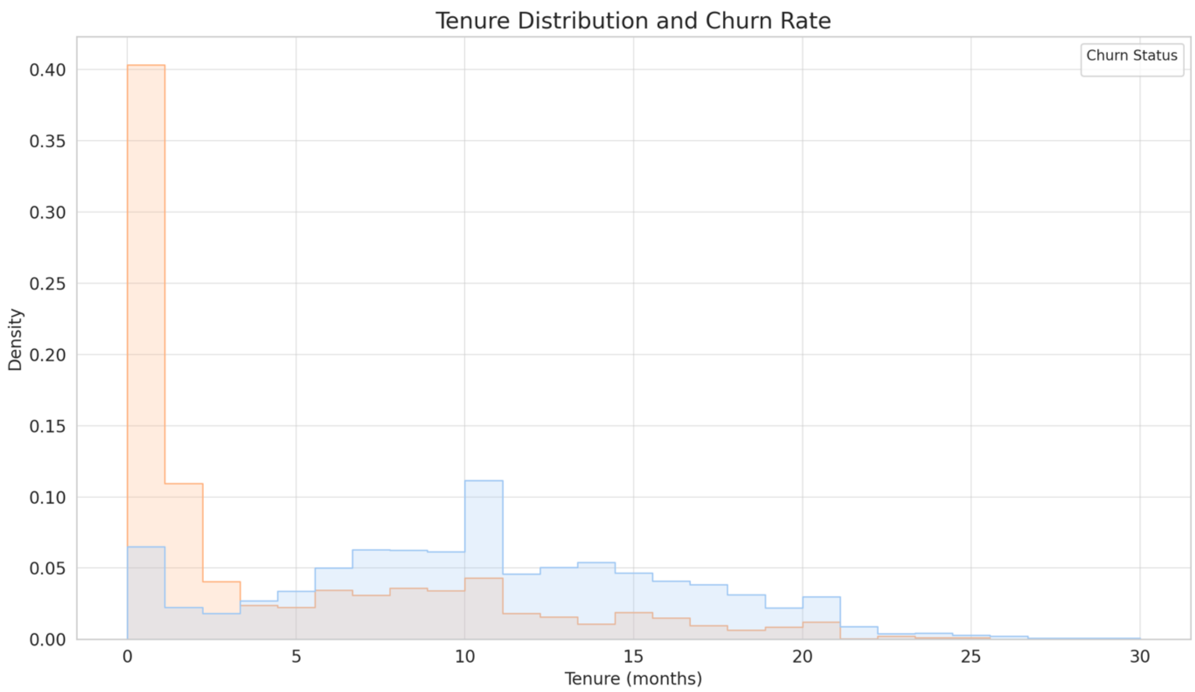
Survey Scores and Churn Rate



Distribution by Survey Scores: Most customers seem satisfied with the services, giving scores of 7 and above.

Churn Rate by Survey Scores: There's a direct correlation between lower survey scores and higher churn rates, emphasizing the importance of customer satisfaction in retention.

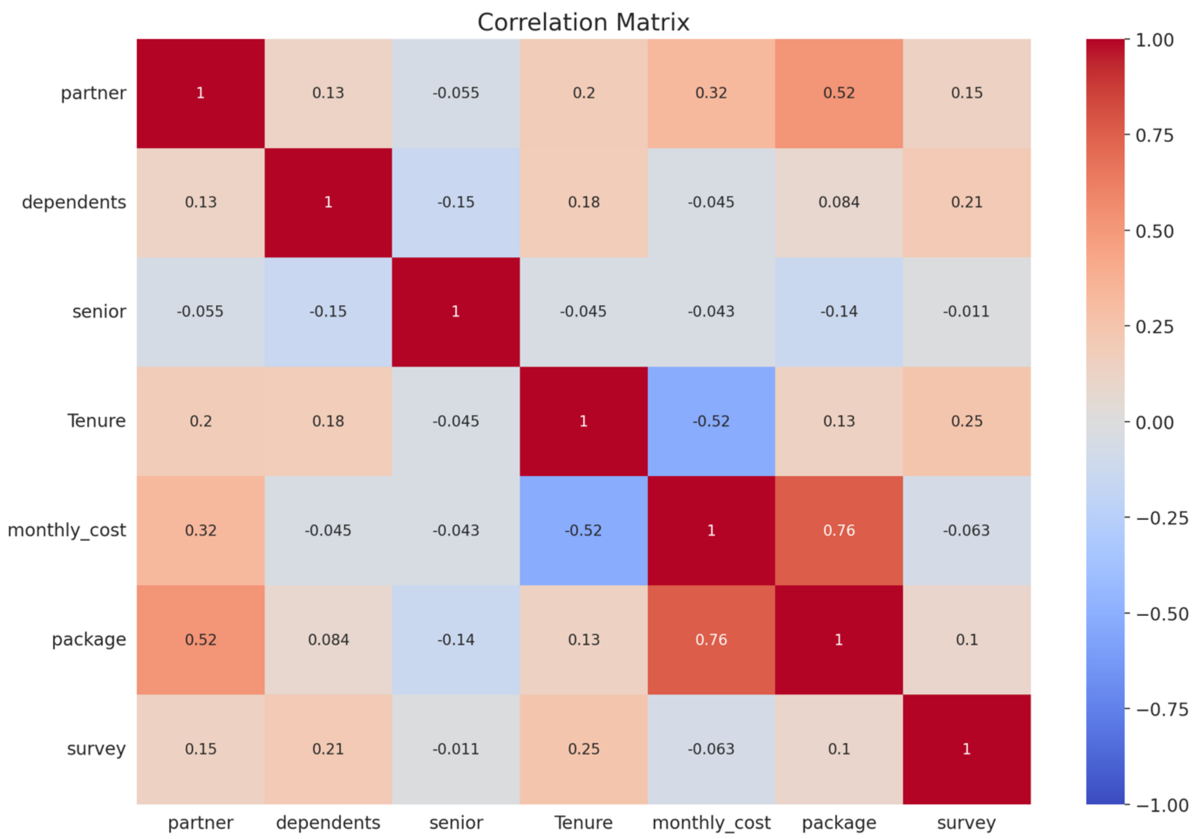
Tenure Distribution and Churn Rate



Distribution by Tenure: A significant portion of customers have been with the company for less than 10 years.

Churn Rate by Tenure: Longer-tenured customers tend to stay, indicating increased loyalty or satisfaction over time.

Correlation Matrix:



The moderate positive correlation between partner and dependents suggests that customers with partners are more likely to have dependents. Businesses can consider this correlation when targeting specific customer segments.

The slight negative correlation between Tenure and monthly_cost implies loyalty discounts or packages that reward long-term customers with reduced rates.

Higher monthly costs could be a factor contributing to customer churn. This might be due to customers perceiving less value for the amount they're paying or facing affordability issues.

Data Pre-processing

Dealing with Outliers

Using the Interquartile Range (IQR) method, outliers were identified and treated in the monthly_cost feature.

Missing Value Imputation

Missing values in the monthly_cost column were addressed by calculating the expected cost based on customer packages and tenure. The missing values in the Class column were filled using the mode.

Data Scaling

To ensure consistent performance across models, especially those sensitive to feature scales, data was standardized using the StandardScaler.
Before building our models, it was crucial to ensure our data was standardized and clean. This phase involved consistent data scaling and addressing outliers.

The dataset has been successfully one-hot encoded and split into training and test sets. The training set contains 5735 samples, and the test set contains 1434 samples. Both sets have 24 features.

Model Development and Evaluation

Four machine learning models were trained and evaluated:

We built and tested multiple models to predict customer churn. The models, along with their accuracy scores, are as follows:

Model	Accuracy (%)
Logistic Regression	88.08
Decision Tree Classifier	88.01
Random Forest Classifier	91.42
Gradient Boosting Classifier	91.77

Logistic Regression

Accuracy: 88.08%

Overview:
Logistic Regression, being a simple and interpretable model, offered a good baseline accuracy.

Decision Tree Classifier

Accuracy: 88.01%

Overview: While this model provides a visual representation of decision-making processes, it tends to overfit, which might impact its reliability on new, unseen data.

Random Forest Classifier
Accuracy: 91.42%
Overview: An ensemble method, the Random Forest Classifier constructs multiple decision trees and amalgamates their predictions for a more robust outcome.

Gradient Boosting Classifier
Accuracy: 91.77%
Overview: Another ensemble method, the Gradient Boosting Classifier, builds trees in a sequential manner. Each tree corrects errors from its predecessors, leading to enhanced accuracy.

Conclusion:

Through this comprehensive analysis, we aimed to understand and predict customer churn for Lu's Communications. Here's a summary of our findings and insights:

Key Observations:

1. Customers without partners showed a slightly higher propensity to churn.
2. Package 1, despite being the most popular, had the highest churn rate, suggesting potential areas of improvement.
3. Lower survey scores directly correlated with higher churn rates, emphasizing the importance of customer satisfaction.
4. Longer-tenured customers showed more loyalty, indicating the company's success in retaining long-term subscribers.

Correlation Insights:

1. A moderate positive correlation was observed between having a partner and dependents.

2. A slight negative correlation between tenure and monthly cost suggests the presence of loyalty discounts.

Model Performance:

1. Four machine learning models were trained and evaluated.
2. The Gradient Boosting Classifier emerged as the best model with an accuracy of 91.77%, closely followed by the Random Forest Classifier at 91.42%.
3. Logistic Regression and Decision Tree Classifier, despite their simplicity, provided competitive accuracy scores of 88.08% and 88.01% respectively.

Recommendations:

- Lu's Communications should focus on improving Package 1 offerings or communication, given its high churn rate.

- Emphasis should be placed on enhancing customer satisfaction, as indicated by the direct correlation between survey scores and churn rates.
- Targeted retention strategies can be developed for customers without partners and those in specific regions with higher churn rates.

Final Thoughts:

- Predicting customer churn requires more than just creating an accurate model; it also involves understanding its causes and patterns. With these insights in hand, Lu's Communications can make informed decisions, maximize resources efficiently, and strengthen customer loyalty.
- Lu's Communications now employs the Gradient Boosting Classifier as its best model to anticipate and proactively address customer churn, guaranteeing ongoing growth with loyal customer relations.
- Understanding and managing customer churn is an ongoing effort at Lu's Communications, but the insights and tools gained through this analysis make us better equipped to deal with the competitive telecoms market.

Appendix (ASSESSMENT 1)

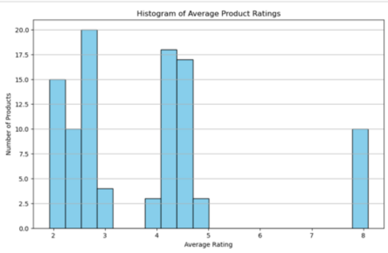
In [1]:
```
import pandas as pd

# Read the train.txt file into a dataframe
train_df = pd.read_csv('train.txt', delimiter=' , ', engine='python'

# Clean the columns to get just the numerical values
train_df['User'] = train_df['User'].str.replace('(', '', regex=False
train_df['Product'] = train_df['Product'].str.replace('(', '', regex
train_df['Rating'] = train_df['Rating'].str.replace('(', '', regex=F

# Calculate the number of unique users and products
num_users = train_df['User'].nunique()
num_products = train_df['Product'].nunique()

num_users, num_products
```

Out[1]: (500, 100)

In [2]:
```
# Create a pivot table for the ratings, where rows are users, colum
Y = train_df.pivot(index='User', columns='Product', values='Rating'

# Get the shape/dimensions of Y
dimensions_Y = Y.shape
dimensions_Y
```

Out[2]: (500, 100)

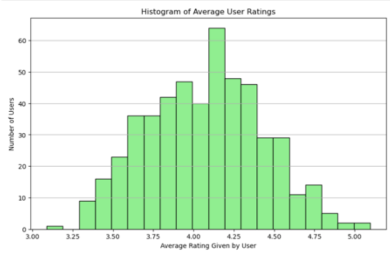
In [3]:
```
import matplotlib.pyplot as plt

# Calculate the average rating for each product
average_product_rating = Y.mean()

# Plot the histogram
plt.figure(figsize=(10,6))
plt.hist(average_product_rating, bins=20, color='skyblue', edgecolo
plt.title('Histogram of Average Product Ratings')
plt.xlabel('Average Rating')
plt.ylabel('Number of Products')
plt.grid(axis='y')
plt.show()
```



In [4]:
```
# Identify the 5 products with the lowest average ratings
worst_products = average_product_rating.nsmallest(5)
worst_products
```

Out[4]:
```
Product
56033    1.920530
72533    1.954839
22577    1.976898
16430    2.013468
60751    2.034247
dtype: float64
```


In [5]:
```
# Calculate the average rating given by each user
average_user_rating = Y.mean(axis=1)

# Plot the histogram
plt.figure(figsize=(10,6))
plt.hist(average_user_rating, bins=20, color='lightgreen', edgecolo
plt.title('Histogram of Average User Ratings')
plt.xlabel('Average Rating Given by User')
plt.ylabel('Number of Users')
plt.grid(axis='y')
plt.show()
```



In [6]:
```
# Identify the 5 users with the highest average ratings (most gener
most_generous_users = average_user_rating.nlargest(5)
most_generous_users
```

Out[6]:
```
User
31410    5.096154
73310    5.061538
97887    4.904762
77730    4.896552
20146    4.890625
dtype: float64
```

Page 4 of 6

In [7]:
```
 Load the test data
est_df = pd.read_csv('test.txt', sep=',', header=None)
est_df.columns = ['User', 'Product', 'Rating']

 Remove parentheses and unnecessary text
est_df['User'] = test_df['User'].str.replace('(', '', regex=False).s
est_df['Product'] = test_df['Product'].str.replace(')', '', regex=Fa
est_df['Rating'] = test_df['Rating'].str.replace(')', '', regex=Fals

 Convert data types for User and Rating to integer, and Product to f
est_df['User'] = test_df['User'].astype(int)
est_df['Product'] = test_df['Product'].astype(int)
est_df['Rating'] = test_df['Rating'].astype(float)

 Count the total number of unique users and products
um_users_test = test_df['User'].nunique()
um_products_test = test_df['Product'].nunique()

rint(f"Number of unique users in test data: {num_users_test}")
rint(f"Number of unique products in test data: {num_products_test}")
```

```
Number of unique users in test data: 421
Number of unique products in test data: 2
```

In [8]:
```
# Check unique values in the 'Product' column of the test dataset
unique_product_values_test = test_df['Product'].unique()
unique_product_values_test
```

Out[8]: `array([0, 1])`

In [9]:
```
# Assign numerical identifiers to the product labels
test_df['Product'] = test_df['Product'].replace({'Product test0': 9

# Calculate the number of unique users and products in test data
num_users_test = test_df['User'].nunique()
num_products_test = test_df['Product'].nunique()

num_users_test, num_products_test
```

Out[9]: `(421, 2)`

In [10]:
```
# Create a pivot table for the ratings in the test dataset
X = test_df.pivot(index='User', columns='Product', values='Rating')

# Get the shape/dimensions of X
dimensions_X = X.shape
dimensions_X
```

Out[10]: `(421, 2)`

In [11]:
```
import numpy as np

def compute_distance(product_from_X, product_from_Y):
    """Compute the distance between two products using the given fo
    common_users = product_from_X.dropna().index.intersection(produ
    return np.abs(product_from_X.loc[common_users] - product_from_Y

# Calculate the distance for each product in X to all products in Y
similarities = {}
for product_X in X.columns:
    distances = {}
    for product_Y in Y.columns:
        distances[product_Y] = compute_distance(X[product_X], Y[pro
    # Sort products in Y by their distance to the current product i
    top_5_similar = sorted(distances, key=distances.get)[:5]
    similarities[product_X] = top_5_similar

similarities
```

Out[11]:
```
{0: [50408, 58577, 38851, 60734, 26457],
 1: [24785, 26457, 50408, 40821, 38851]}
```

Page 6 of 6

In [14]:
```
def compute_alternative_distance(product_from_X, product_from_Y, al
    """Compute the distance between two products using the alternat

    # Calculate the absolute differences for common ratings
    common_users = product_from_X.dropna().index.intersection(produ
    common_distance = np.abs(product_from_X.loc[common_users] - pro

    # Add penalties for ratings that only exist in one of the produ
    x_only_users = product_from_X.dropna().index.difference(product
    x_penalty = (product_from_X.loc[x_only_users] * alpha).sum()

    y_only_users = product_from_Y.dropna().index.difference(product
    y_penalty = (product_from_Y.loc[y_only_users] * alpha).sum()

    return common_distance + x_penalty + y_penalty

# Calculate the distance for each product in X to all products in Y
alternative_similarities = {}
for product_X in X.columns:
    distances = {}
    for product_Y in Y.columns:
        distances[product_Y] = compute_alternative_distance(X[produ
    # Sort products in Y by their distance to the current product i
    top_5_similar = sorted(distances, key=distances.get)[:5]
    alternative_similarities[product_X] = top_5_similar

alternative_similarities
```

Out[14]:
```
{0: [50408, 80772, 41232, 26457, 38851],
 1: [66611, 26457, 24785, 50408, 42484]}
```

In [ ]:

Appendix (ASSESSMENT 2)

In [1]:
```
import pandas as pd

# Load the dataset
data = pd.read_csv('Group 2.csv')

# Display the first few rows of the dataset
data
```

Out[1]:

	Unnamed: 0	customer_id	gender	location	partner	dependents	senior	Tenure
0	0	K3713	Male	Hampshire	0	Unknown	0	12.0
1	1	D9048	Male	Greater Manchester	1	1	0	21.0
2	2	K8227	Female	West Yorkshire	0	Unknown	0	0.0
3	3	H3533	Male	Greater London	1	1	1	11.0
4	4	J4501	Male	Greater London	0	0	0	7.0
...	...	...	...	...	...	...	...	...
7345	7345	H7244	Female	Greater London	0	0	0	1.0
7346	7346	F1052	Male	Hampshire	0	0	0	2.0
7347	7347	I5775	Male	Greater London	1	1	0	16.0
7348	7348	E9984	Female	Greater London	0	1	0	17.0
7349	7349	F2835	Male	West Yorkshire	0	1	0	13.0

7350 rows × 12 columns

In [2]:
```
# Drop the unnecessary column
data.drop(columns=['Unnamed: 0'], inplace=True)

# Check for missing values
missing_values = data.isnull().sum()

# Basic statistics for numeric columns
numeric_stats = data.describe()

# Distribution of the target variable
target_distribution = data['Class'].value_counts()

missing_values, numeric_stats, target_distribution
```


Out[2]:
```
(customer_id        0
 gender             0
 location           0
 partner            0
 dependents         0
 senior             0
 Tenure             0
 monthly_cost    7283
 package            0
 survey             0
 Class             49
 dtype: int64,
            partner       senior       Tenure      package
 count  7350.000000  7350.000000  7350.000000  7350.000000
 mean      0.549252     0.172925     8.665947     2.425850
 std       0.497602     0.378208     6.404877     1.151303
 min       0.000000     0.000000    -4.690416     1.000000
 25%       0.000000     0.000000     3.000000     1.000000
 50%       1.000000     0.000000     8.000000     2.000000
 75%       1.000000     0.000000    14.000000     4.000000
 max       1.000000     1.000000    30.000000     4.000000,
 Churn=No     5203
 Churn=Yes    2081
 Y$e$s$$        17
 Name: Class, dtype: int64)
```

Handle the negative values in the Tenure column. Investigate and address the anomalies in the Class column. Decide how to handle missing values in the monthly_cost column. Investigate the dependents column to handle the "Unknown" entries.

In [3]:
```
data_summary = data.describe(include='all')
data_info = data.info()

data_summary
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7350 entries, 0 to 7349
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   customer_id   7350 non-null   object
 1   gender        7350 non-null   object
 2   location      7350 non-null   object
 3   partner       7350 non-null   int64
 4   dependents    7350 non-null   object
 5   senior        7350 non-null   int64
 6   Tenure        7350 non-null   float64
 7   monthly_cost  67 non-null     object
 8   package       7350 non-null   int64
 9   survey        7350 non-null   object
 10  Class         7301 non-null   object
dtypes: float64(1), int64(3), object(7)
memory usage: 631.8+ KB
```


Out[3]:

	customer_id	gender	location	partner	dependents	senior	Te
count	7350	7350	7350	7350.000000	7350	7350.000000	7350.00
unique	6752	2	17	NaN	3	NaN	
top	K4807	Male	Greater London	NaN	1	NaN	
freq	4	3694	2436	NaN	3454	NaN	
mean	NaN	NaN	NaN	0.549252	NaN	0.172925	8.665
std	NaN	NaN	NaN	0.497602	NaN	0.378208	6.404
min	NaN	NaN	NaN	0.000000	NaN	0.000000	-4.69
25%	NaN	NaN	NaN	0.000000	NaN	0.000000	3.000
50%	NaN	NaN	NaN	1.000000	NaN	0.000000	8.000
75%	NaN	NaN	NaN	1.000000	NaN	0.000000	14.000
max	NaN	NaN	NaN	1.000000	NaN	1.000000	30.000

In [4]:
```
# Unique values in the 'survey' column
unique_survey_values = data['survey'].unique()
unique_survey_values
```

Out[4]:
```
array(['6', '4', '2', '3', '8', '1', '5', '7', 'No reply', '9', '0', '10'
],
      dtype=object)
```

In [5]:
```
# Count of 'No reply' in the 'survey' column
no_reply_count = data[data['survey'] == 'No reply'].shape[0]
no_reply_count
```

Out[5]: 552

In [6]:
```
# Filter out rows where 'Tenure' has negative values
data = data[data['Tenure'] >= 0]

# Verify if there are any negative values left in the 'Tenure' column
remaining_negative_tenure = data[data['Tenure'] < 0].shape[0]

remaining_negative_tenure
```

Out[6]: 0

In [7]:
```
# Calculate the number of missing values in each column
missing_values = data.isnull().sum()
missing_values_percentage = (missing_values / len(data)) * 100

# Combine the missing values count and percentage into a DataFrame
missing_df = pd.DataFrame({'Missing Values': missing_values, 'Percenta
missing_df.sort_values(by='Percentage (%)', ascending=False)
```

Out[7]:

	Missing Values	Percentage (%)
monthly_cost	7152	99.085619
Class	49	0.678858
customer_id	0	0.000000
gender	0	0.000000
location	0	0.000000
partner	0	0.000000
dependents	0	0.000000
senior	0	0.000000
Tenure	0	0.000000
package	0	0.000000
survey	0	0.000000

In [8]:

```
# Compute the 'monthly_cost' for missing values
# Define package costs
data = data.copy()
package_costs = {1: 26, 2: 34, 3: 40, 4: 45}
# Function to compute monthly cost with corrections
def compute_monthly_cost(row):
    if pd.isnull(row['monthly_cost']):
        cost = package_costs[row['package']]
        # Applying loyalty discount
        discount = min(0.02 * row['Tenure'], 0.5) # Maximum discount is 5
        cost = cost * (1 - discount)
        return cost
    else:
        # If 'monthly_cost' is a string, remove unnecessary symbols and c
        if isinstance(row['monthly_cost'], str):
            return float(row['monthly_cost'].replace('$', '').replace(','
        return row['monthly_cost']

# Apply the function to the 'monthly_cost' column
data['monthly_cost'] = data.apply(compute_monthly_cost, axis=1)

# Check the first few rows of the dataset again
data.head()
```


Out[8]:

	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_cost
0	K3713	Male	Hampshire	0	Unknown	0	12.0	19.76
1	D9048	Male	Greater Manchester	1	1	0	21.0	26.10
2	K8227	Female	West Yorkshire	0	Unknown	0	0.0	26.00
3	H3533	Male	Greater London	1	1	1	11.0	26.52
4	J4501	Male	Greater London	0	0	0	7.0	38.70

In [9]:

```
# Identify columns with missing values and their count
missing_values = data.isnull().sum()
missing_columns = missing_values[missing_values > 0]

missing_columns
```

Out[9]:

```
Class    49
dtype: int64
```

In [10]:

```
data
```

Out[10]:

	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_c
0	K3713	Male	Hampshire	0	Unknown	0	12.0	19
1	D9048	Male	Greater Manchester	1	1	0	21.0	26
2	K8227	Female	West Yorkshire	0	Unknown	0	0.0	26
3	H3533	Male	Greater London	1	1	1	11.0	26
4	J4501	Male	Greater London	0	0	0	7.0	38
...	...	...	...	...	...	...	...	
7345	H7244	Female	Greater London	0	0	0	1.0	25
7346	F1052	Male	Hampshire	0	0	0	2.0	24
7347	I5775	Male	Greater London	1	1	0	16.0	30
7348	E9984	Female	Greater London	0	1	0	17.0	1
7349	F2835	Male	West Yorkshire	0	1	0	13.0	

7218 rows × 11 columns

In [11]:

```
# Convert the column to numeric, setting errors='coerce' to turn invalid
data['survey'] = pd.to_numeric(data['survey'], errors='coerce')

# Now compute the rounded mean (since invalid values are NaN, they won't
rounded_mean_survey = round(data['survey'].mean())

# Fill NaN values with the rounded mean
data['survey'].fillna(rounded_mean_survey, inplace=True)

# Replace 'Y$e$s$$' with 'Churn=Yes' in 'Class'
data['Class'].replace('Y$e$s$$', 'Churn=Yes', inplace=True)

# For 'dependents', replace 'Unknown' or missing values with 0
data['dependents'].replace('Unknown', 0, inplace=True)
data['dependents'].fillna(0, inplace=True)

# Check if the operations were successful
print(data[['survey', 'dependents']].isnull().sum())
```

```
survey        0
dependents    0
dtype: int64
```

In [12]:

```
rounded_mean_survey
```

Out[12]:

```
5
```

In [13]:

```
# Check for missing values
data.isnull().sum()
```

Out[13]:

```
customer_id      0
gender           0
location         0
partner          0
dependents       0
senior           0
Tenure           0
monthly_cost     0
package          0
survey           0
Class           49
dtype: int64
```

In [14]:

```
# Remove rows where 'Class' column is null or missing
data = data.dropna(subset=['Class'])

data.isnull().sum()
```


Out[14]:

```
customer_id     0
gender          0
location        0
partner         0
dependents      0
senior          0
Tenure          0
monthly_cost    0
package         0
survey          0
Class           0
dtype: int64
```

In [15]:

```
data['Class'].value_counts()
```

Out[15]:

```
Churn=No     5098
Churn=Yes    2071
Name: Class, dtype: int64
```

In [16]:

```
data
```

Out[16]:

	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_c
0	K3713	Male	Hampshire	0	0	0	12.0	19
1	D9048	Male	Greater Manchester	1	1	0	21.0	26
2	K8227	Female	West Yorkshire	0	0	0	0.0	26
3	H3533	Male	Greater London	1	1	1	11.0	26
4	J4501	Male	Greater London	0	0	0	7.0	38
...	...	...	...	...	...	...	...	
7345	H7244	Female	Greater London	0	0	0	1.0	25
7346	F1052	Male	Hampshire	0	0	0	2.0	24
7347	I5775	Male	Greater London	1	1	0	16.0	30
7348	E9984	Female	Greater London	0	1	0	17.0	1
7349	F2835	Male	West Yorkshire	0	1	0	13.0	25

7169 rows × 11 columns

In [11]:
```python
# Convert the column to numeric, setting errors='coerce' to turn invalid
data['survey'] = pd.to_numeric(data['survey'], errors='coerce')

# Now compute the rounded mean (since invalid values are NaN, they won't
rounded_mean_survey = round(data['survey'].mean())

# Fill NaN values with the rounded mean
data['survey'].fillna(rounded_mean_survey, inplace=True)

# Replace 'Y$e$s$$' with 'Churn=Yes' in 'Class'
data['Class'].replace('Y$e$s$$', 'Churn=Yes', inplace=True)

# For 'dependents', replace 'Unknown' or missing values with 0
data['dependents'].replace('Unknown', 0, inplace=True)
data['dependents'].fillna(0, inplace=True)

# Check if the operations were successful
print(data[['survey', 'dependents']].isnull().sum())
```

```
survey        0
dependents    0
dtype: int64
```

In [12]:
```python
rounded_mean_survey
```

Out[12]: 5

In [13]:
```python
# Check for missing values
data.isnull().sum()
```

Out[13]:
```
customer_id     0
gender          0
location        0
partner         0
dependents      0
senior          0
Tenure          0
monthly_cost    0
package         0
survey          0
Class          49
dtype: int64
```

In [14]:
```python
# Remove rows where 'Class' column is null or missing
data = data.dropna(subset=['Class'])

data.isnull().sum()
```


Out[14]:
```
customer_id     0
gender          0
location        0
partner         0
dependents      0
senior          0
Tenure          0
monthly_cost    0
package         0
survey          0
Class           0
dtype: int64
```

In [15]:
```python
data['Class'].value_counts()
```

Out[15]:
```
Churn=No     5098
Churn=Yes    2071
Name: Class, dtype: int64
```

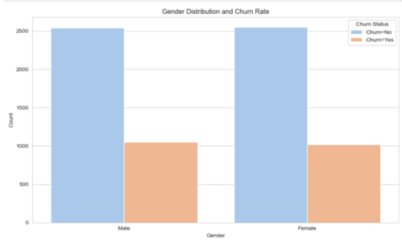
In [16]:
```python
data
```

Out[16]:

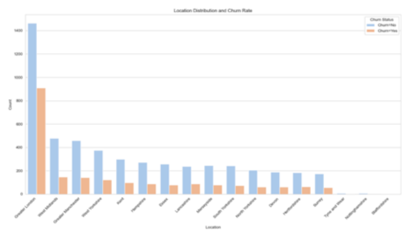
	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_c
0	K3713	Male	Hampshire	0	0	0	12.0	19
1	D9048	Male	Greater Manchester	1	1	0	21.0	26
2	K8227	Female	West Yorkshire	0	0	0	0.0	26
3	H3533	Male	Greater London	1	1	1	11.0	26
4	J4501	Male	Greater London	0	0	0	7.0	38
...	...	...	...	...	...	...	...	
7345	H7244	Female	Greater London	0	0	0	1.0	25
7346	F1052	Male	Hampshire	0	0	0	2.0	24
7347	I5775	Male	Greater London	1	1	0	16.0	30
7348	E9984	Female	Greater London	0	1	0	17.0	1
7349	F2835	Male	West Yorkshire	0	1	0	13.0	25

7169 rows × 11 columns

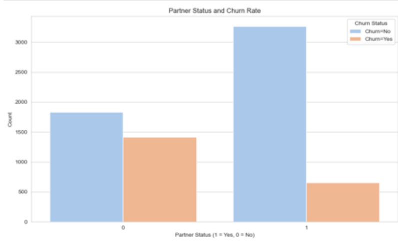
In [17]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style and color palette for the plots
sns.set_style("whitegrid")
sns.set_palette("pastel")

# Plotting Gender Distribution and Churn Rate
plt.figure(figsize=(10, 6))
sns.countplot(x='gender', hue='Class', data=data)
plt.title("Gender Distribution and Churn Rate")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.legend(title="Churn Status")

plt.tight_layout()
plt.show()
```



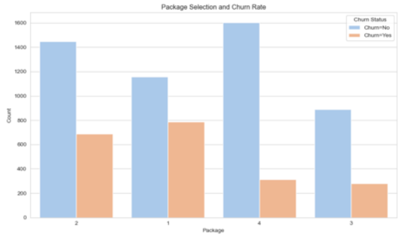
In [18]:
```python
# Plotting Location Distribution and Churn Rate
plt.figure(figsize=(14, 8))
sns.countplot(x='location', hue='Class', data=data, order=data['location'
plt.title("Location Distribution and Churn Rate")
plt.xlabel("Location")
plt.ylabel("Count")
plt.legend(title="Churn Status")
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```

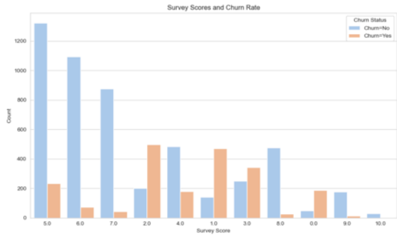
In [19]:
```python
# Plotting Partner Status and Churn Rate
plt.figure(figsize=(10, 6))
sns.countplot(x='partner', hue='Class', data=data)
plt.title("Partner Status and Churn Rate")
plt.xlabel("Partner Status (1 = Yes, 0 = No)")
plt.ylabel("Count")
plt.legend(title="Churn Status")

plt.tight_layout()
plt.show()
```



In [20]:
```
# Plotting Package Selection and Churn Rate
plt.figure(figsize=(10, 6))
sns.countplot(x='package', hue='Class', data=data, order=data['package'].
plt.title("Package Selection and Churn Rate")
plt.xlabel("Package")
plt.ylabel("Count")
plt.legend(title="Churn Status")

plt.tight_layout()
plt.show()
```

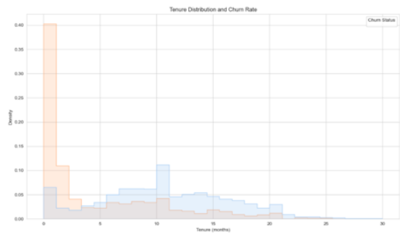


In [21]:
```
# Plotting Survey Scores and Churn Rate
plt.figure(figsize=(10, 6))
sns.countplot(x='survey', hue='Class', data=data, order=data['survey'].va
plt.title("Survey Scores and Churn Rate")
plt.xlabel("Survey Score")
plt.ylabel("Count")
plt.legend(title="Churn Status")

plt.tight_layout()
plt.show()
```

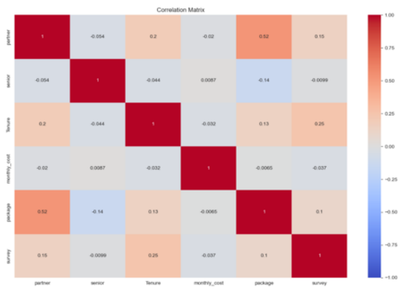



In [22]:
```
# Plotting Tenure Distribution and Churn Rate
plt.figure(figsize=(12, 7))
sns.histplot(data=data, x='Tenure', hue='Class', element='step', stat='de
plt.title("Tenure Distribution and Churn Rate")
plt.xlabel("Tenure (months)")
plt.ylabel("Density")
plt.legend(title="Churn Status")

plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose l
abel start with an underscore are ignored when legend() is called with no
argument.
```

In [23]:
```
# Generating the correlation matrix
correlation_matrix = data.corr()

# Plotting the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vma
plt.title("Correlation Matrix")

plt.tight_layout()
plt.show()
```

In [24]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [25]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Take a copy to avoid the SettingWithCopyWarning
data = data.copy()

# Drop customer_id column
data.drop('customer_id', axis=1, inplace=True)

# Label encode gender and Class columns
label_encoder = LabelEncoder()
data['gender'] = label_encoder.fit_transform(data['gender'])
data['Class'] = label_encoder.fit_transform(data['Class'])

# One-hot encode location column
data = pd.get_dummies(data, columns=['location'], drop_first=True)

# Split data into features (X) and target variable (y)
X = data.drop('Class', axis=1)
y = data['Class']

# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

X_train.head()
```

Out[25]:

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	loc
4605	0	1	1	0	11.0	31.20	3	9.0	
3997	0	1	1	0	17.0	26.40	3	5.0	
4764	0	1	0	0	22.0	25.20	4	3.0	
6349	0	1	0	1	1.0	33.32	2	6.0	
5775	0	1	1	0	3.0	24.44	1	3.0	

5 rows × 24 columns

In [26]:
```
X_train
```


Out[26]:

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	loc
4605	0	1	1	0	11.0	31.20	3	9.0	
3997	0	1	1	0	17.0	26.40	3	5.0	
4764	0	1	0	0	22.0	25.20	4	3.0	
6349	0	1	0	1	1.0	33.32	2	6.0	
5775	0	1	1	0	3.0	24.44	1	3.0	
...	...	...	...	...	...	...	...	...	...
3858	0	1	1	1	1.0	33.32	2	6.0	
5317	1	1	0	0	0.0	34.00	2	0.0	
5353	0	0	0	0	12.0	19.76	1	2.0	
5520	0	0	0	0	1.0	25.48	1	6.0	
883	1	1	0	0	18.0	28.80	4	6.0	

5735 rows × 24 columns

In [27]:
```
X_test
```

Out[27]:

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	loc
7083	1	1	0	0	6.0	22.88	1	0.0	
466	0	1	0	0	0.0	40.00	3	1.0	
7076	1	1	1	0	8.0	37.80	4	5.0	
1693	1	0	0	0	0.0	26.00	1	1.0	
2963	1	0	1	0	2.0	24.96	1	0.0	
...	...	...	...	...	...	...	...	...	...
3219	1	1	0	0	3.0	42.30	4	6.0	
5077	1	1	0	1	7.0	22.36	1	4.0	
285	1	1	1	1	0.0	34.00	2	1.0	
2140	0	1	1	0	12.0	34.20	4	6.0	
1751	1	0	0	0	1.0	25.48	1	3.0	

1434 rows × 24 columns

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize the models
logistic_reg = LogisticRegression(max_iter=1000, random_state=42)

# Train the Logistic Regression model
logistic_reg.fit(X_train, y_train)

# Predict on the test set
logistic_reg_preds = logistic_reg.predict(X_test)

# Calculate accuracy for Logistic Regression
logistic_reg_accuracy = accuracy_score(y_test, logistic_reg_preds)

logistic_reg_accuracy
```

Out[28]: 0.8800557880055788

In [29]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Train the Decision Tree Classifier
decision_tree.fit(X_train, y_train)

# Predict on the test set
decision_tree_preds = decision_tree.predict(X_test)

# Calculate accuracy for Decision Tree Classifier
decision_tree_accuracy = accuracy_score(y_test, decision_tree_preds)

decision_tree_accuracy
```

Out[29]: 0.8800557880055788

In [30]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest Classifier
random_forest = RandomForestClassifier(random_state=42)

# Train the Random Forest Classifier
random_forest.fit(X_train, y_train)

# Predict on the test set
random_forest_preds = random_forest.predict(X_test)

# Calculate accuracy for Random Forest Classifier
random_forest_accuracy = accuracy_score(y_test, random_forest_preds)

random_forest_accuracy
```

Out[30]: 0.9142259414225942

In [31]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Initialize the Gradient Boosting Classifier
gradient_boosting = GradientBoostingClassifier(random_state=42)

# Train the Gradient Boosting Classifier
gradient_boosting.fit(X_train, y_train)

# Predict on the test set
gradient_boosting_preds = gradient_boosting.predict(X_test)

# Calculate accuracy for Gradient Boosting Classifier
gradient_boosting_accuracy = accuracy_score(y_test, gradient_boosting_pre

gradient_boosting_accuracy
```

Out[31]: 0.9177126917712691

In [32]:
```
# Using the provided results to create the table
results_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree Classifier', 'Random
    'Accuracy (%)': [88.08, 88.01, 91.42, 91.77]
})

results_df
```

Out[32]:

	Model	Accuracy (%)
0	Logistic Regression	88.08
1	Decision Tree Classifier	88.01
2	Random Forest Classifier	91.42
3	Gradient Boosting Classifier	91.77

In [ ]:

In [ ]: