Persian Gulf University
Data Mining
Spring Semester 2022-2023

Teacher: Dr. Sahafizade

Ali Behrouzi

Stdn: 4010724002
Email: alibehrouzi97@gmail.com

Final Project
- Customer Churn Prediction

Dataset : Telecom customer

- ## About Dataset

Telecom customer churn prediction

This data set consists of 100 variables and approx 100 thousand records. This data set contains different variables explaining the attributes of telecom industry and various factors considered important while dealing with customers of telecom industry. The target variable here is churn which explains whether the customer will churn or not. We can use this data set to predict the customers who would churn or who wouldn't churn depending on various variables available.

You can get this dataset from here: Kaggle

Variable Description:

1. rev_Mean: Mean monthly revenue (charge amount)
2. mou_Mean: Mean number of monthly minutes of use
3. totmrc_Mean: Mean total monthly recurring charge
4. da_Mean: Mean number of directory assisted calls
5. ovrmou_Mean: Mean overage minutes of use
6. ovrrev_Mean: Mean overage revenue
7. vceovr_Mean: Mean revenue of voice overage
8. datovr_Mean: Mean revenue of data overage
9. roam_Mean: Mean number of roaming calls
10. change_mou: Percentage change in monthly minutes of use vs previous three month average
11. change_rev: Percentage change in monthly revenue vs previous three month average
12. drop_vce_Mean: Mean number of dropped (failed) voice calls
13. drop_dat_Mean: Mean number of dropped (failed) data calls
14. blck_vce_Mean: Mean number of blocked (failed) voice calls
15. blck_dat_Mean: Mean number of blocked (failed) data calls
16. unan_vce_Mean: Mean number of unanswered voice calls
17. unan_dat_Mean: Mean number of unanswered data calls

18. **plcd_vce_Mean: Mean number of attempted voice calls placed**
19. **plcd_dat_Mean: Mean number of attempted data calls placed**
20. **recv_vce_Mean: Mean number of received voice calls**
21. **recv_sms_Mean: N**
22. **comp_vce_Mean: Mean number of completed voice calls**
23. **comp_dat_Mean: Mean number of completed data calls**
24. **custcare_Mean: Mean number of customer care calls**
25. **ccrndmou_Mean: Mean rounded minutes of use of customer care calls**
26. **cc_mou_Mean: Mean unrounded minutes of use of customer care (see CUSTCARE_MEAN) calls**
27. **inonemin_Mean: Mean number of inbound calls less than one minute**
28. **threeway_Mean: Mean number of three way calls**
29. **mou_cvce_Mean: Mean unrounded minutes of use of completed voice calls**
30. **mou_cdat_Mean: Mean unrounded minutes of use of completed data calls**
31. **mou_rvce_Mean: Mean unrounded minutes of use of received voice calls**
32. **owylis_vce_Mean: Mean number of outbound wireless to wireless voice calls**
33. **mouowylisv_Mean: Mean unrounded minutes of use of outbound wireless to wireless voice calls**
34. **iwylis_vce_Mean: N**
35. **mouiwylisv_Mean: Mean unrounded minutes of use of inbound wireless to wireless voice calls**
36. **peak_vce_Mean: Mean number of inbound and outbound peak voice calls**
37. **peak_dat_Mean: Mean number of peak data calls**
38. **mou_peav_Mean: Mean unrounded minutes of use of peak voice calls**
39. **mou_pead_Mean: Mean unrounded minutes of use of peak data calls**
40. **opk_vce_Mean: Mean number of off-peak voice calls**
41. **opk_dat_Mean: Mean number of off-peak data calls**
42. **mou_opkv_Mean: Mean unrounded minutes of use of off-peak voice calls**
43. **mou_opkd_Mean: Mean unrounded minutes of use of off-peak data calls**
44. **drop_blk_Mean: Mean number of dropped or blocked calls**
45. **attempt_Mean: Mean number of attempted calls**
46. **complete_Mean: Mean number of completed calls**
47. **callfwdv_Mean: Mean number of call forwarding calls**
48. **callwait_Mean: Mean number of call waiting calls**
49. **churn: Instance of churn between 31-60 days after observation date**
50. **months: Total number of months in service**

51. **uniqsubs: Number of unique subscribers in the household**
52. **actvsubs: Number of active subscribers in household**
53. **new_cell: New cell phone user**
54. **crclscod: Credit class code**
55. **asl_flag: Account spending limit**
56. **totcalls: Total number of calls over the life of the customer**
57. **totmou: Total minutes of use over the life of the cus**
58. **totrev: Total revenue**
59. **adjrev: Billing adjusted total revenue over the life of the customer**
60. **adjmou: Billing adjusted total minutes of use over the life of the customer**
61. **adjqty: Billing adjusted total number of calls over the life of the customer**
62. **avgrev: Average monthly revenue over the life of the customer**
63. **avgmou: Average monthly minutes of use over the life of the customer**
64. **avgqty: Average monthly number of calls over the life of the customer**
65. **avg3mou: Average monthly minutes of use over the previous three months**
66. **avg3qty: Average monthly number of calls over the previous three months**
67. **avg3rev: Average monthly revenue over the previous three months**
68. **avg6mou: Average monthly minutes of use over the previous six months**
69. **avg6qty: Average monthly number of calls over the previous six months**
70. **avg6rev: Average monthly revenue over the previous six months**
71. **prizm_social_one: Social group letter only**
72. **area: Geogrpahic area**
73. **dualband: Dualband**
74. **refurb_new: Handset: refurbished or new**
75. **hnd_price: Current handset price**
76. **phones: Number of handsets issued**
77. **models: Number of models issued**
78. **hnd_webcap: Handset web capability**
79. **truck: Truck indicator**
80. **rv: RV indicator**
81. **ownrent: Home owner/renter status**
82. **lor: Length of residence**
83. **dwlltype: Dwelling Unit type**

84. **marital: Marital Status**
85. **adults: Number of adults in household**
86. **infobase: InfoBase match**
87. **income: Estimated income**
88. **numbcars: Known number of vehicles**
89. **HHstatin: Premier household status indicator**
90. **dwllsize: Dwelling size**
91. **forgntvl: Foreign travel dummy variable**
92. **ethnic: Ethnicity roll-up code**
93. **kid0_2: Child 0 - 2 years of age in household**
94. **kid3_5: Child 3 - 5 years of age in household**
95. **kid6_10: Child 6 - 10 years of age in household**
96. **kid11_15: Child 11 - 15 years of age in household**
97. **kid16_17: Child 16 - 17 years of age in household**
98. **creditcd: Credit card indicator**
99. **eqpdays: Number of days (age) of current equipment**
100. **Customer_ID: N**

Start Action :

- Load the dataset

```
import pandas as pd
import os

# Load the dataset
dataset_path = os.path.join("Telecom_customer churn.csv")
telecom_df = pd.read_csv(dataset_path)

# Display the first few rows of the dataset
telecom_df.head()
```

- examining the distribution of the 'churn' column

```
# Check the distribution of the 'churn' column
churn_distribution = telecom_df['churn'].value_counts(normalize=True)

# Check for missing values in the dataset
missing_values = telecom_df.isnull().sum()

# Filter out columns with missing values for display
missing_values = missing_values[missing_values > 0]

churn_distribution, missing_values
```

1. Churn Distribution:

- Approximately 50.4% of the users have not churned (labeled as 0).

- Approximately 49.6% of the users have churned (labeled as 1).

The classes are fairly balanced, which is good for modeling.

2. Missing Values:

- Several columns have missing values. Some columns, like `ownrent`, `lor`, and `dwlltype`, have a large number of missing values.

- Handling these missing values will be crucial before modeling. We can either impute them with appropriate values or consider dropping some of the columns with a very high percentage of missing values.

- # Exploratory Data Analysis (EDA)

Univariate Analysis

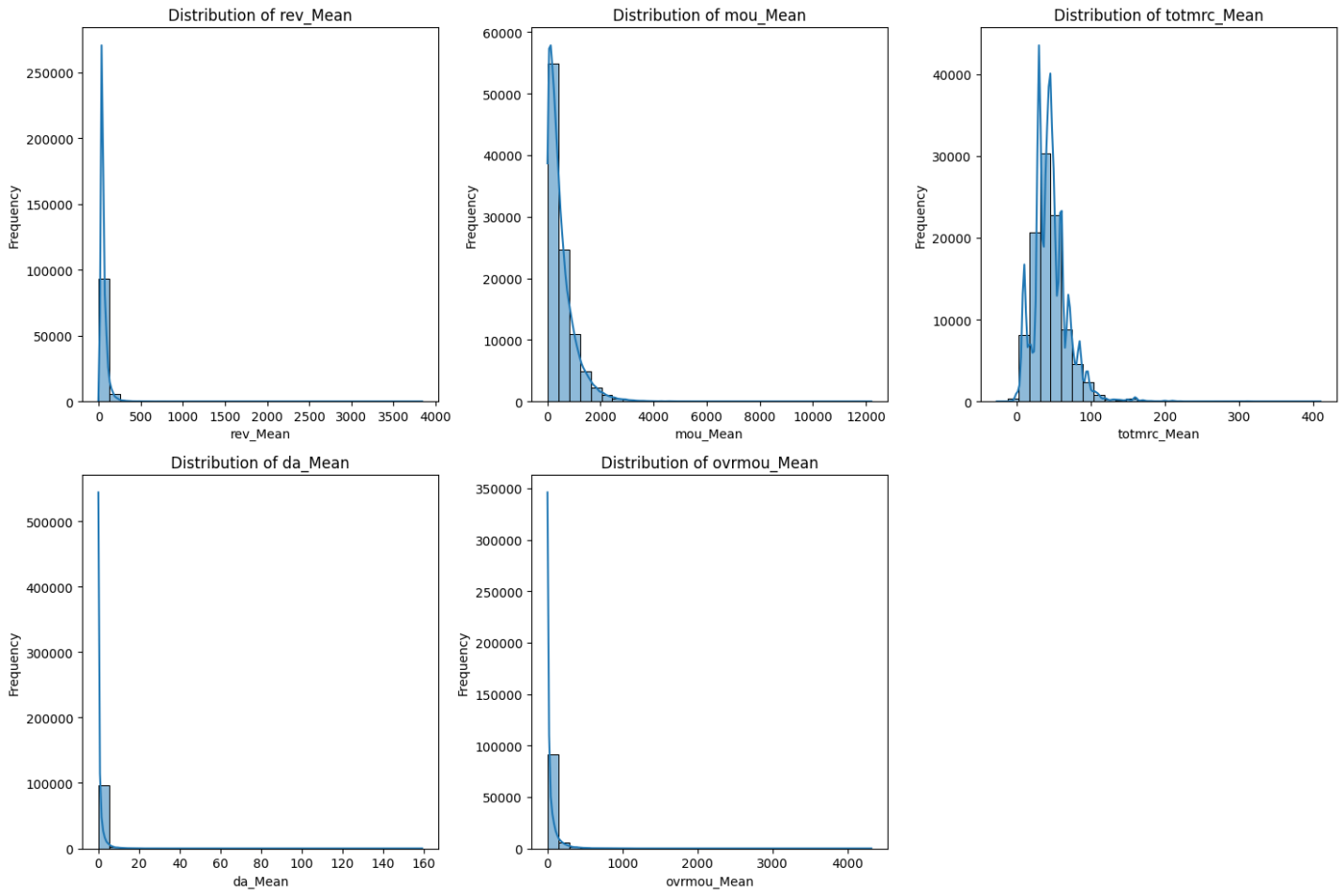
1. distribution of a subset of numerical variables

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Selecting a subset of numerical columns for demonstration
numerical_cols = ['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'da_Mean', 'ovrmou_Mean']

# Plotting histograms for the selected numerical columns
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 3, i)
    sns.histplot(telecom_df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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



Here are the distributions for a subset of the numerical variables:

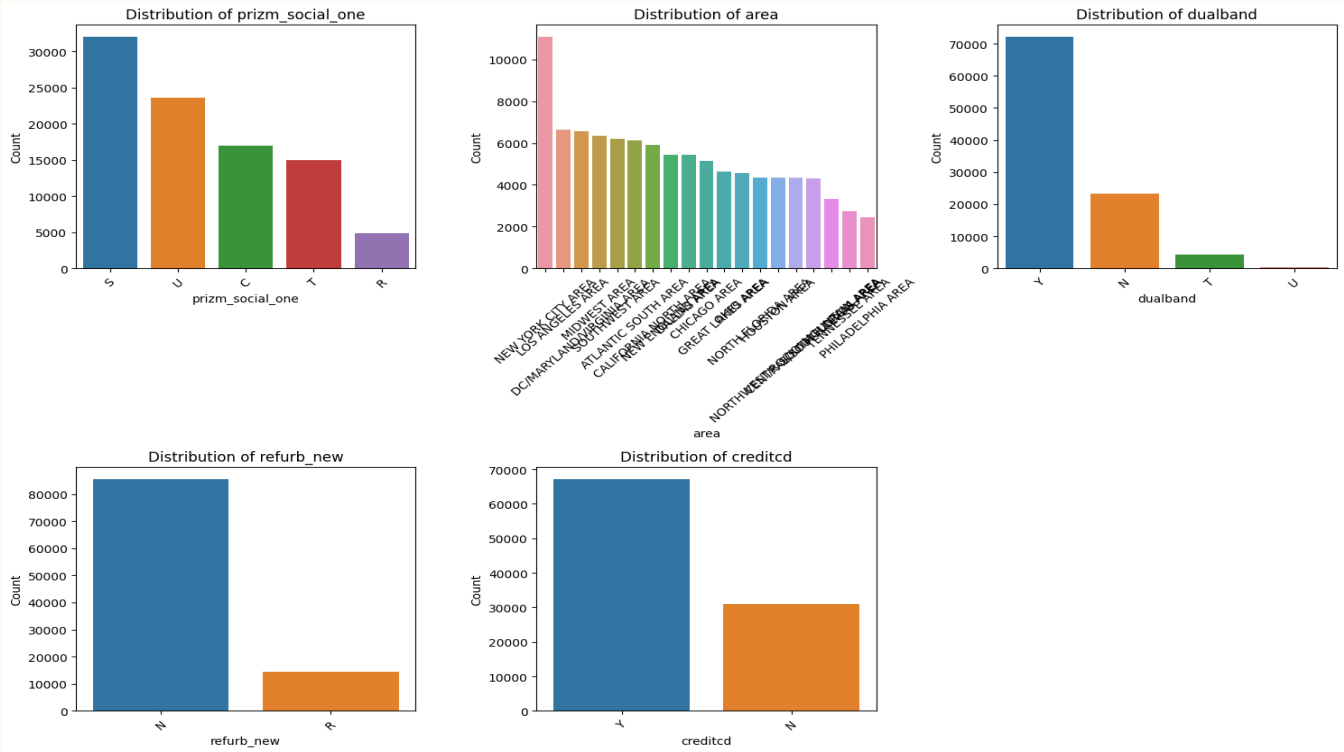
- **rev_Mean:** This seems to be right-skewed with most values clustered towards the lower end.
- **mou_Mean:** This represents the mean number of minutes of use. Similar to rev_Mean, it's also right-skewed.
- **totmrc_Mean:** Right-skewed distribution with a cluster around the lower values.
- **da_Mean:** Most values are concentrated around 0, indicating that many users have no or very low "direct access" usage.
- **ovrmou_Mean:** This variable, representing overage minutes, also has most values clustered near 0, suggesting that many users do not exceed their allocated minutes.

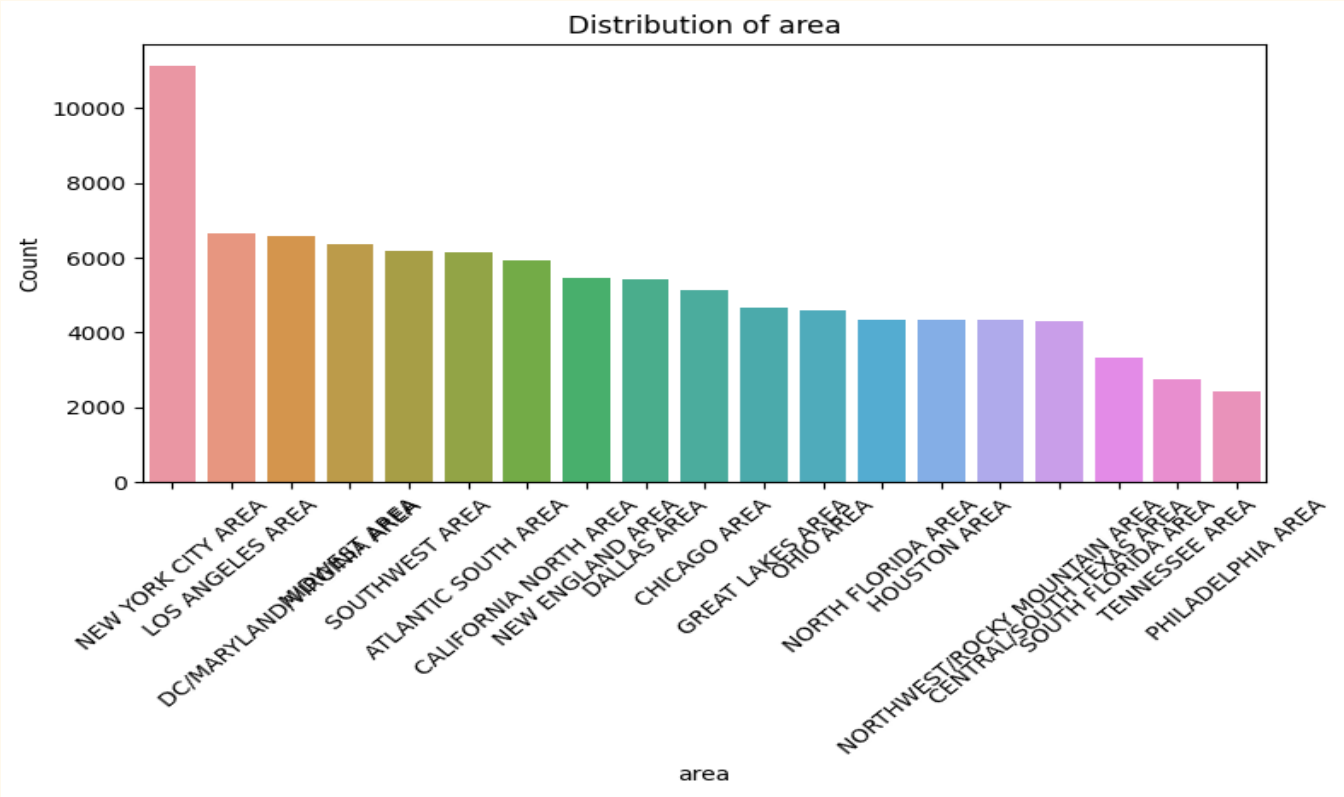
2. visualize the distribution of a few categorical variables

```
# Selecting a subset of categorical columns for demonstration
categorical_cols = ['prizm_social_one', 'area', 'dualband', 'refurb_new', 'creditcd']

# Plotting count plots for the selected categorical columns
plt.figure(figsize=(15, 10))
for i, col in enumerate(categorical_cols, 1):
    plt.subplot(2, 3, i)
    sns.countplot(data=telecom_df, x=col, order=telecom_df[col].value_counts().index)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)

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





Here are the distributions for a subset of the categorical variables:

- **prizm_social_one:** This variable has several categories, with 'S' and 'U' being the most frequent.
- **area:** The dataset contains data from multiple areas, with some areas having more observations than others.
- **dualband:** Most users have the 'Y' category, indicating they might be using dual-band devices.
- **refurb_new:** A vast majority of users have 'N', suggesting they might be using new devices rather than refurbished ones.
- **creditcd:** Most users seem to have the 'Y' category, possibly indicating they have a credit card.

Bivariate Analysis

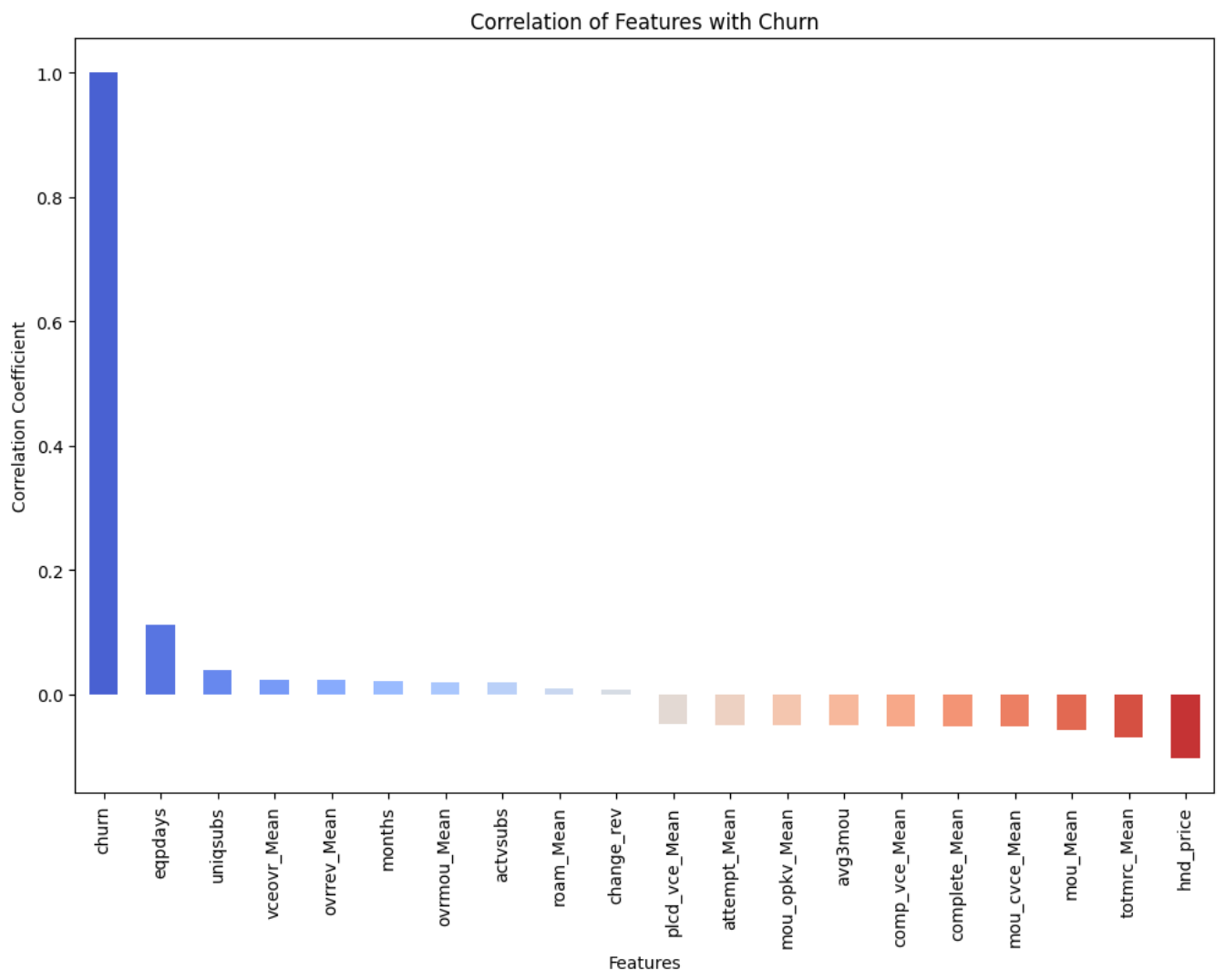
1. Correlation Matrix:

- We'll visualize the correlation of numerical features with the target variable 'churn'.

```python
# Calculate the correlation of all numerical columns with 'churn'
correlation_with_churn = telecom_df.corr()['churn'].sort_values(ascending=False)

# Select top and bottom correlated features for visualization
top_correlated = correlation_with_churn.head(10)
bottom_correlated = correlation_with_churn.tail(10)
selected_correlation = pd.concat([top_correlated, bottom_correlated])

# Plotting
plt.figure(figsize=(12, 8))
selected_correlation.plot(kind='bar', color=sns.color_palette('coolwarm', len(selected_correlation)))
plt.title('Correlation of Features with Churn')
plt.ylabel('Correlation Coefficient')
plt.xlabel('Features')
plt.show()
```



- **Positive Correlation:** Features that have a positive correlation indicate that as the feature value increases, the likelihood of churn also increases.
- **Negative Correlation:** Features that have a negative correlation indicate that as the feature value increases, the likelihood of churn decreases.

From the visualization:

- Features like **eqpdays** (equipment days) have a strong positive correlation with churn, suggesting that users with older equipment might be more likely to churn.
- On the other hand, features like **mou_Mean** (mean minutes of usage) and **rev_Mean** (mean revenue) show a negative correlation, indicating that users with higher usage or revenue are less likely to churn.

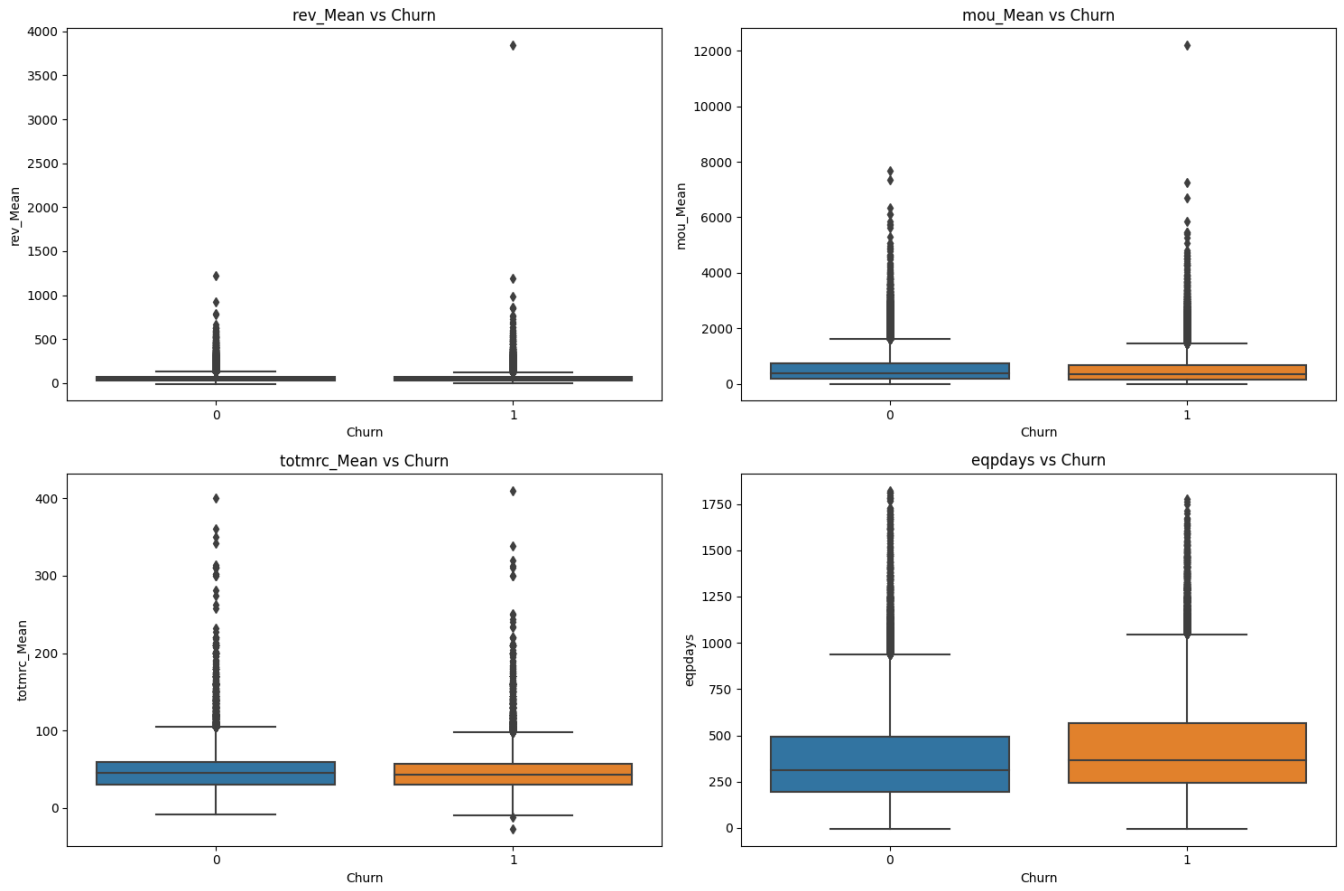
2. Box Plots

- We'll choose a few numerical features and visualize their distribution with respect to the 'churn' column

```
# Selecting a subset of numerical columns for demonstration
selected_numerical_cols = ['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'eqpdays']

# Plotting box plots for the selected numerical columns against 'churn'
plt.figure(figsize=(15, 10))
for i, col in enumerate(selected_numerical_cols, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(data=telecom_df, x='churn', y=col)
    plt.title(f'{col} vs Churn')
    plt.xlabel('Churn')
    plt.ylabel(col)

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



- **rev_Mean: The median revenue for non-churn users is slightly higher than for churn users. There are also several outliers present, especially for churn users.**

- **mou_Mean:** Non-churn users tend to have a higher median of minutes of use compared to churn users. Again, there are outliers indicating users with very high usage.
- **totmrc_Mean:** Non-churn users have a slightly higher median of total monthly recurring charge than churn users.
- **eqpdays:** The median equipment age is higher for churn users, which aligns with the positive correlation we observed earlier. Users with older equipment are more likely to churn.

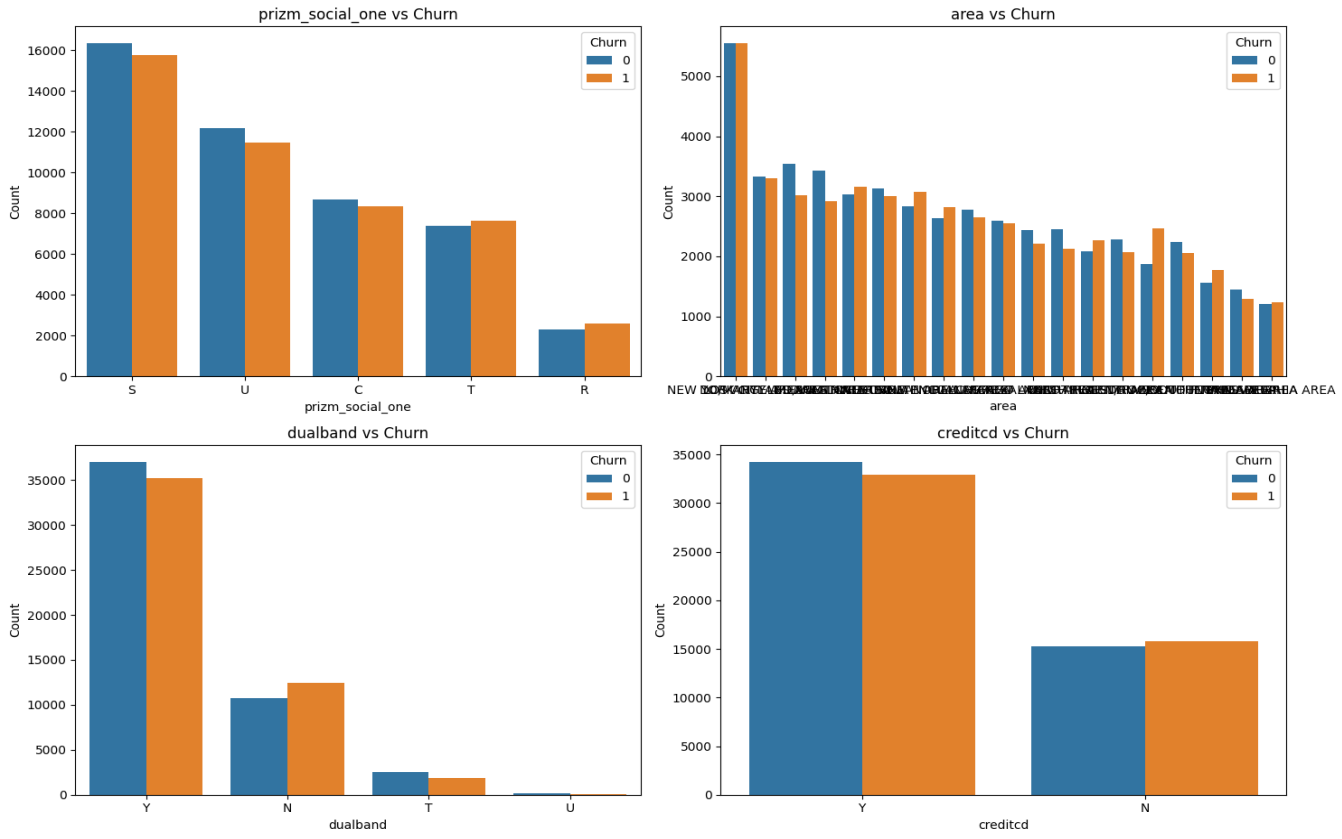
3. Categorical vs 'churn':

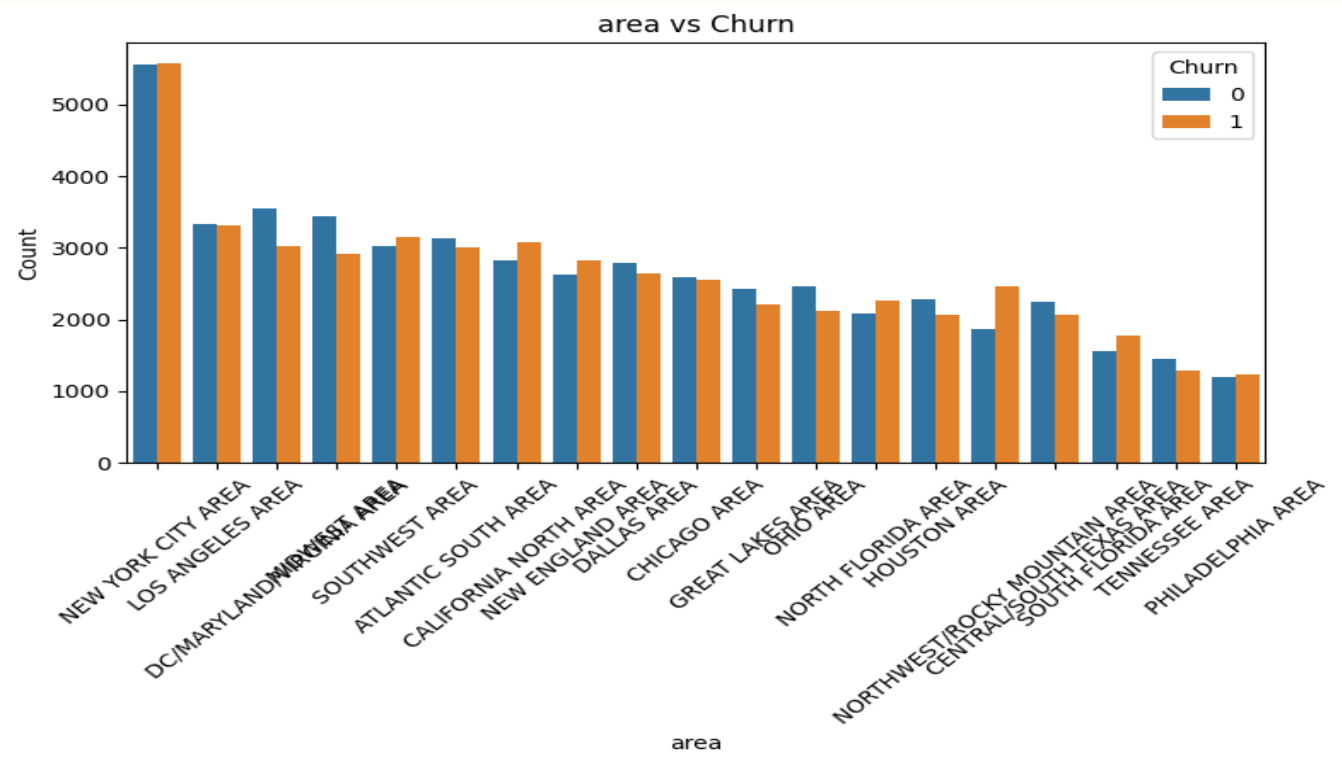
- We'll select a few categorical features and visualize how different categories might be affecting the churn rate using stacked bar plots or count plots.

```
# Selecting a subset of categorical columns for demonstration
selected_categorical_cols = ['prizm_social_one', 'area', 'dualband', 'creditcd']

# Plotting count plots for the selected categorical columns against 'churn'
plt.figure(figsize=(15, 10))
for i, col in enumerate(selected_categorical_cols, 1):
    plt.subplot(2, 2, i)
    sns.countplot(data=telecom_df, x=col, hue='churn', order=telecom_df[col].value_counts().index)
    plt.title(f'{col} vs Churn')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.legend(title='Churn', loc='upper right')

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


prizm_social_one vs Churn
Churn
0
1
Count
prizm_social_one
S
U
C
T
R
area vs Churn
area
dualband vs Churn
dualband
Y
N
T
U
creditcd vs Churn
creditcd


area vs Churn
Churn
0
1
Count
area
NEW YORK CITY AREA
LOS ANGELES AREA
SOUTHWEST AREA
ATLANTIC SOUTH AREA
CALIFORNIA NORTH AREA
NEW ENGLAND AREA
DALLAS AREA
CHICAGO AREA
GREAT LAKES AREA
OHIO AREA
NORTH FLORIDA AREA
HOUSTON AREA
NORTHWEST/ROCKY MOUNTAIN AREA
CENTRAL/SOUTH TEXAS AREA
SOUTH FLORIDA AREA
TENNESSEE AREA
PHILADELPHIA AREA

- **prizm_social_one:** Different categories within this feature show varying churn rates. For instance, category 'S' seems to have a higher churn rate compared to others.
- **area:** The churn rate varies across different areas, with some areas showing a higher proportion of churned users than others.
- **dualband:** The churn rate is slightly higher for users in the 'Y' category compared to the 'N' category.
- **creditcd:** Users in the 'Y' category (possibly indicating they have a credit card) seem to have a slightly lower churn rate compared to the 'N' category.

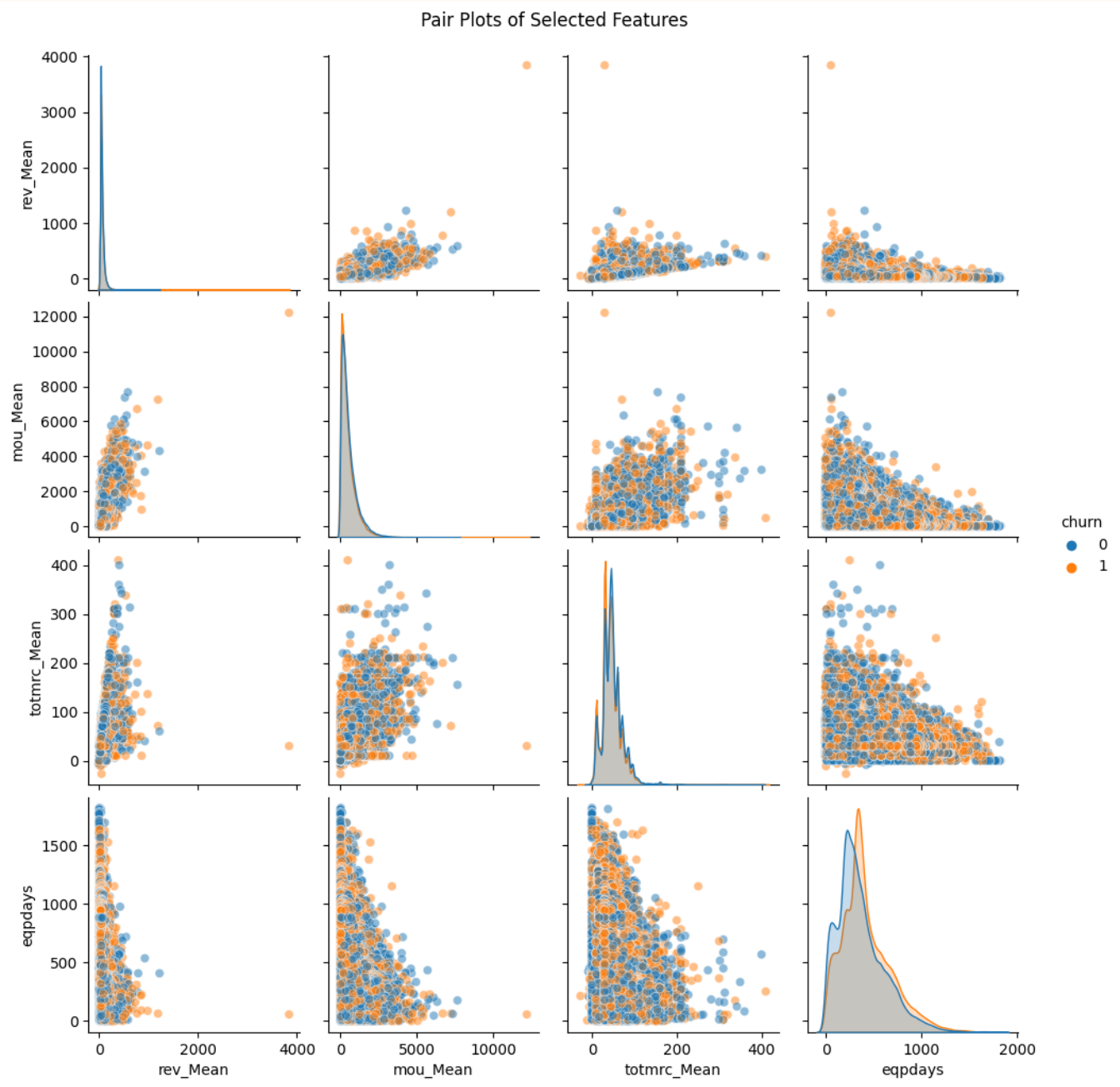
Multivariate Analysis

1. Pair Plots:

- Create scatter plots for a few selected features to visualize the relationships between them and see if any clusters or patterns emerge.

```
# Selecting a smaller subset of numerical columns for the pair plot (for visualization clarity)
selected_for_pairplot = ['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'eqpdays', 'churn']

# Creating a pair plot for the selected columns
sns.pairplot(telecom_df[selected_for_pairplot], hue='churn', plot_kws={'alpha':0.5})
plt.suptitle('Pair Plots of Selected Features', y=1.02)
plt.show()
```



The pair plots showcase the relationships between selected numerical features and how they are distributed based on the churn status:

- **The diagonal plots represent the distribution of individual features, separated by churn status.**

- The off-diagonal scatter plots show the relationships between two features, colored by churn status.

From these plots, you can observe:

- Patterns of concentration where churned and non-churned users seem to cluster.
- Relationships or trends between some features.

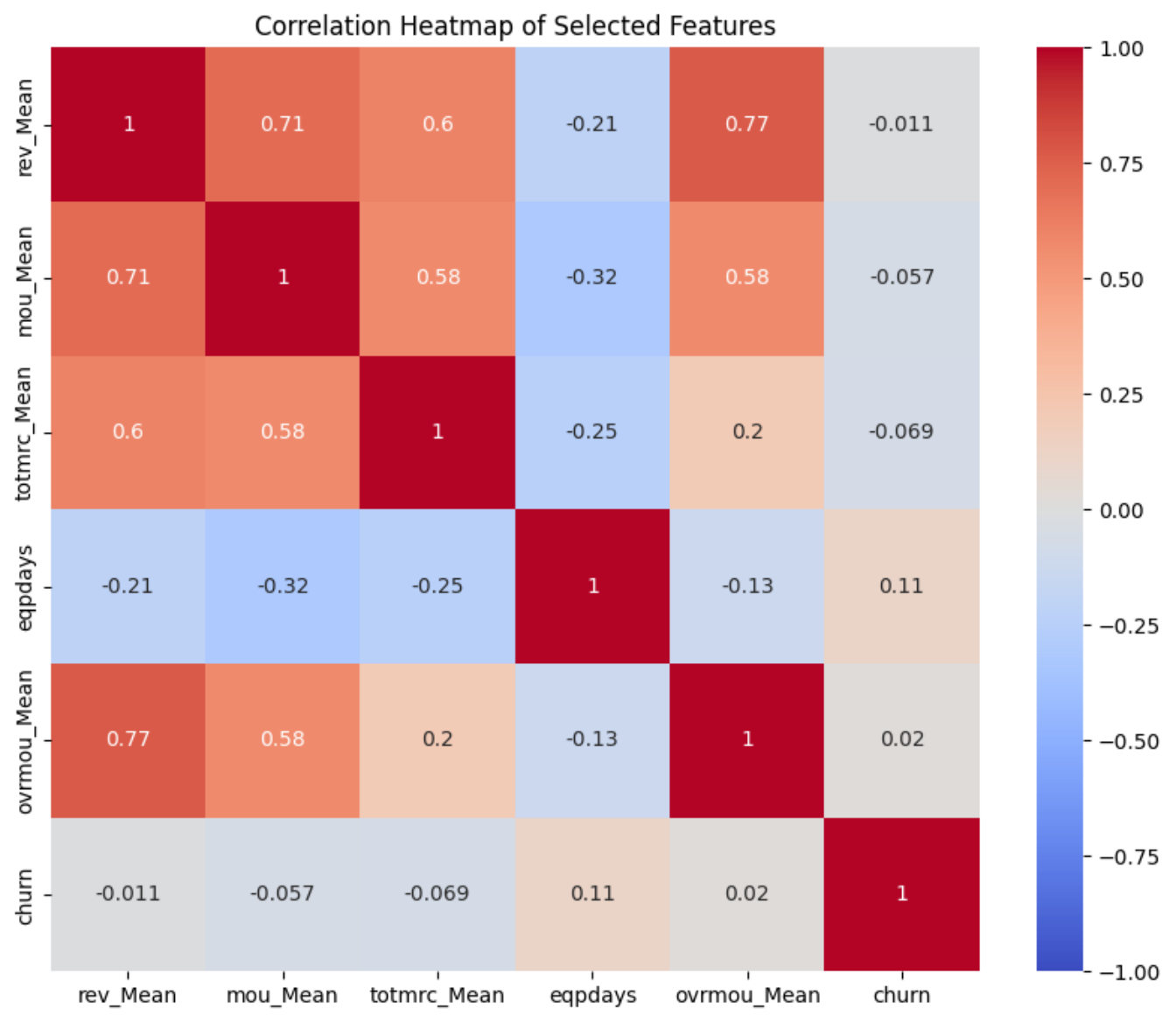
2. Heatmaps:

- This can be especially useful if we want to observe the behavior of groups of features against the churn rate or to view the correlation between multiple features.

```
# Selecting a subset of numerical columns for the heatmap
heatmap_cols = ['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'eqpdays', 'ovrmou_Mean', 'churn']

# Calculating the correlation matrix for the selected columns
correlation_matrix = telecom_df[heatmap_cols].corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Selected Features')
plt.show()
```



The heatmap displays the correlation between the selected numerical features:

- **The color intensity and the annotated values represent the strength and direction of the correlation.**
- **Values close to 1 or -1 represent strong positive or negative correlations, respectively, while values close to 0 represent weak or no correlation.**

From the heatmap:

- **rev_Mean, mou_Mean,** and **totmrc_Mean** have strong positive correlations with each other, which makes intuitive sense as they are all related to the user's usage and revenue.
- **eqpdays** has a positive correlation with **churn**, reaffirming our earlier observation that users with older equipment are more likely to churn.

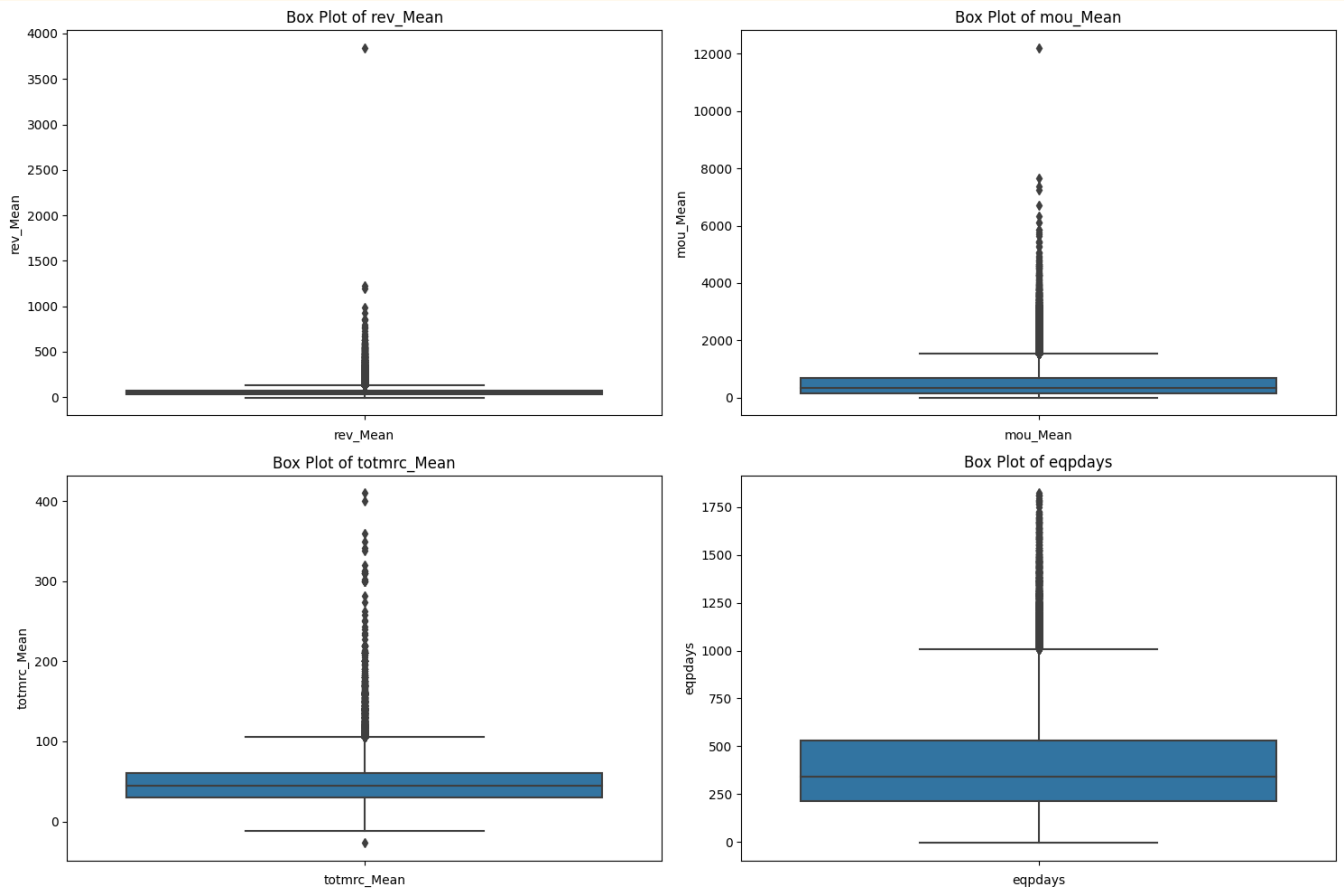
Outlier Detection

1. Visual Method

```
# Selecting a subset of numerical columns for box plots
selected_for_outliers = ['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'eqpdays']

# Plotting box plots for outlier detection
plt.figure(figsize=(15, 10))
for i, col in enumerate(selected_for_outliers, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(y=telecom_df[col])
    plt.title(f'Box Plot of {col}')
    plt.xlabel(col)

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



- **rev_Mean:** There are several data points above the upper whisker, indicating potential outliers.
- **mou_Mean:** Like rev_Mean, there are data points above the upper whisker which can be considered as outliers.
- **totmrc_Mean:** Similarly, potential outliers can be observed above the upper whisker.
- **eqpdays:** Most data points are clustered below, but there are a few above the upper whisker, suggesting potential outliers.

2. **Statistical Method:**

- **IQR (Interquartile Range): Data points outside (1.5 \times \text{IQR}) from the Q1 and Q3 quartiles can be considered outliers.**

```python
# Function to calculate the lower and upper bounds for outliers based on IQR
def calculate_iqr_bounds(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return lower_bound, upper_bound

# Calculate outliers for the selected columns
outliers_data = {}
for col in selected_for_outliers:
    lower_bound, upper_bound = calculate_iqr_bounds(telecom_df[col])
    outliers = telecom_df[(telecom_df[col] < lower_bound) | (telecom_df[col] > upper_bound)]
    outliers_data[col] = {
        "Lower Bound": lower_bound,
        "Upper Bound": upper_bound,
        "Outliers Count": len(outliers)
    }

outliers_data_df = pd.DataFrame(outliers_data).T
outliers_data_df
```

	Lower Bound	Upper Bound	Outliers Count
rev_Mean	-22.975	126.985	5983.0
mou_Mean	-677.625	1531.375	5154.0
totmrc_Mean	-14.985	104.975	1800.0
eqpdays	-265.000	1007.000	2585.0

- **rev_Mean: Outliers are values below -22.975 and above 126.985. There are 5,983 potential outliers.**

- **mou_Mean:** Outliers are values below -677.625 and above 1,531.375. There are 5,154 potential outliers.
- **totmrc_Mean:** Outliers are values below -14.985 and above 104.975. There are 1,800 potential outliers.
- **eqpdays:** Outliers are values below -265 and above 1,007. There are 2,585 potential outliers.

Data Preprocessing

Handling Missing Values

```
# Re-checking columns with missing values
missing_values_percentage = (telecom_df.isnull().sum() / len(telecom_df)) * 100
missing_values_percentage = missing_values_percentage[missing_values_percentage > 0].sort_values(ascending=False)

missing_values_percentage
```

- Columns like **numbcars**, dwllsize, **HHstatin**, and a few others have a significant percentage of missing values, ranging from around 25% to almost 50%.
- Several columns have a moderate percentage of missing values, and some have very few missing values.

for columns with high percentage of missing value, drop them and for rest of them do imputation

```
# Dropping columns with more than 25% missing values
threshold = 25
columns_to_drop = missing_values_percentage[missing_values_percentage > threshold].index
telecom_df.drop(columns=columns_to_drop, inplace=True)

# Impute numerical columns with median and categorical columns with mode
for col in telecom_df.columns:
    if telecom_df[col].dtype == 'object':
        # Categorical columns
        mode_value = telecom_df[col].mode()[0]
        telecom_df[col].fillna(mode_value, inplace=True)
    else:
        # Numerical columns
        median_value = telecom_df[col].median()
        telecom_df[col].fillna(median_value, inplace=True)

# Check if there are any more missing values
remaining_missing = telecom_df.isnull().sum().sum()
remaining_missing
```

The missing values have been successfully handled. There are no more missing values in the dataset.

Encoding Categorical Variables:

- **Convert categorical variables into a format suitable for modeling.**

```
# Identifying categorical columns and their number of unique values
categorical_cols_info = telecom_df.select_dtypes(include='object').nunique().sort_values()

categorical_cols_info
```

```
creditcd              2
kid11_15              2
asl_flag              2
kid6_10               2
kid3_5                2
kid0_2                2
refurb_new            2
infobase              2
kid16_17              2
new_cell              3
hnd_webcap            3
dualband              4
prizm_social_one      5
marital               5
ethnic               17
area                 19
crclscod             54
dtype: int64
```

One-Hot Encoding :

```
# Performing one-hot encoding for categorical columns
telecom_df_encoded = pd.get_dummies(telecom_df, drop_first=True)

# Checking the shape of the dataset after one-hot encoding
telecom_df_encoded.shape
```

```
(100000, 187)
```

Feature Scaling:

- Standardize or normalize numerical features to bring them onto a similar scale.

```
from sklearn.preprocessing import StandardScaler

# Extracting the target variable
y = telecom_df_encoded['churn']
X = telecom_df_encoded.drop('churn', axis=1)

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Checking the first few rows after scaling
X_scaled.head()
```

Now, all the numerical features have a mean of 0 and a standard deviation of 1.

Splitting the Data:

- Divide the dataset into training and testing sets to evaluate model performance.

```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.20, random_state=42)

# Checking the shapes of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((80000, 186), (20000, 186), (80000,), (20000,))
```

- Training data: 80,000 samples
- Testing data: 20,000 samples

Building The Model :

Import libraries:

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

Create a formal Dense model with 3 layers and compile it with Adam optimizer binary_crossentropy loss function:

```
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))  # Binary classification -> 1 output neuron with sigmoid activation
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

Last 3 epochs results :

```
Epoch 18/20
2000/2000 [==============================] - 5s 3ms/step - loss: 0.6531 - accuracy: 0.6101 - val_loss: 0.6634 - val_accuracy: 0.6022
Epoch 19/20
2000/2000 [==============================] - 8s 4ms/step - loss: 0.6523 - accuracy: 0.6100 - val_loss: 0.6652 - val_accuracy: 0.6024
Epoch 20/20
2000/2000 [==============================] - 5s 3ms/step - loss: 0.6523 - accuracy: 0.6070 - val_loss: 0.6628 - val_accuracy: 0.6046
```

Evaluate the Model :

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
625/625 [==============================] - 1s 2ms/step - loss: 0.6621 - accuracy: 0.6079
Test Accuracy: 60.79%
```

Plot Training History:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))

# Plotting accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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