Predicting Customer Churn in the Telecommunications Industry

A Data-Driven Approach to Improve Customer Retention

Project Overview:

This project aims to develop a classification model that will predict customer churn for SyriaTel, a telecommunications company. I have chosen to follow the CRISP-DM method to complete this project. It will involve six stages: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. The project purposes to provide insights into the patterns and factors influencing customer churn, and also develop a predictive model to assist in reducing customer attrition.

Business Understanding:

SyriaTel is the major stakeholder for this project. They are interested in reducing customer churn. By helping them predict customer churn, they can take proactive measures to ensure maximum customer retentions and profit maximization. The project majorly focuses on identifying patterns that facilitate to customer churn and providing recommendations on how to mitigate this.

Data Loading and Exploration.

To be able to understand our data, we have to look into it before further analysis and comment. This step will enable us understand our data and come up with suitable models. It will also help us specifically tailor our analysis to the main purpose of the project.

In [1599]:

In [1599]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
df = pd.read_csv('syria-tel.csv')
df.head()
```

Out[1599]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121

5 rows × 21 columns

In [1600]:

```python
# The information in the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

Categorical columns

In [1601]:

```python
categorical_columns = df.select_dtypes(exclude='number')
categorical_columns.columns
```

Out[1601]:

```
Index(['state', 'phone number', 'international plan', 'voice mail plan',
       'churn'],
      dtype='object')
```

Numerical Columns

In [1602]:

```
numeric_columns = df.select_dtypes(include='number')
numeric_columns.columns
```

Out[1602]:

```
Index(['account length', 'area code', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls'],
      dtype='object')
```

Null values

In [1603]:

```
# check for rows that have null/missing values
missing_values = df[df.isnull().any(axis=1)]
print(f'We have {len(missing_values)} values missing in our dataset')
```

```
We have 0 values missing in our dataset
```

Check for duplicates

In [1604]:

```
#check for duplcates in our dataset
duplicated_values = df[df.duplicated()]
print(f"We have {len(duplicated_values)} duplicate values in our dataset.")
```

```
We have 0 duplicate values in our dataset.
```

Summary Statistics for numeric variable

In [1605]:

```
df.describe()
```

Out[1605]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000

Data Understanding:

We are anlysing the SyriaTel.csv dataset that is available on Kaggle. The dataset has 3333 rows and 21 columns and has no null values or duplicates. Therefore we do not need to impute any missing values or drop any duplicated values in this case. Among the 21 columns five of them are categorical in nature; `'state'`, `'phone number'`, `'international plan'`, `'voice mail plan'`, `'churn'`. `Churn` which is our target variable in the data set is of boolean data type. Thus, we will make it binary later when building our models.

Some of the columns based on domain knowledge are not actually good predictors and thus dropping them before fitiing into our models will be good. For example, the `phone number` variable has nothing to do with customer chruning the company.

Most values in the dataset are numerical in nature. The summary statistics provides a brief overview of the dataset and the range of values observed in each numerical column.

Data Analysis and Visualization

Next, let's analyze the distribution and correlations of our variables to gain a deeper understanding of our data. This analysis will provide valuable insights that can guide us in effectively handling our data. By examining the distributions and correlations, we can uncover patterns and relationships between variables, which will aid in making informed decisions and conducting further analysis.

Dropping the `phone number` variable

Considering that the phone number variable is not expected to be useful for our analysis, we will proceed by dropping it from the dataset.

In [1606]:

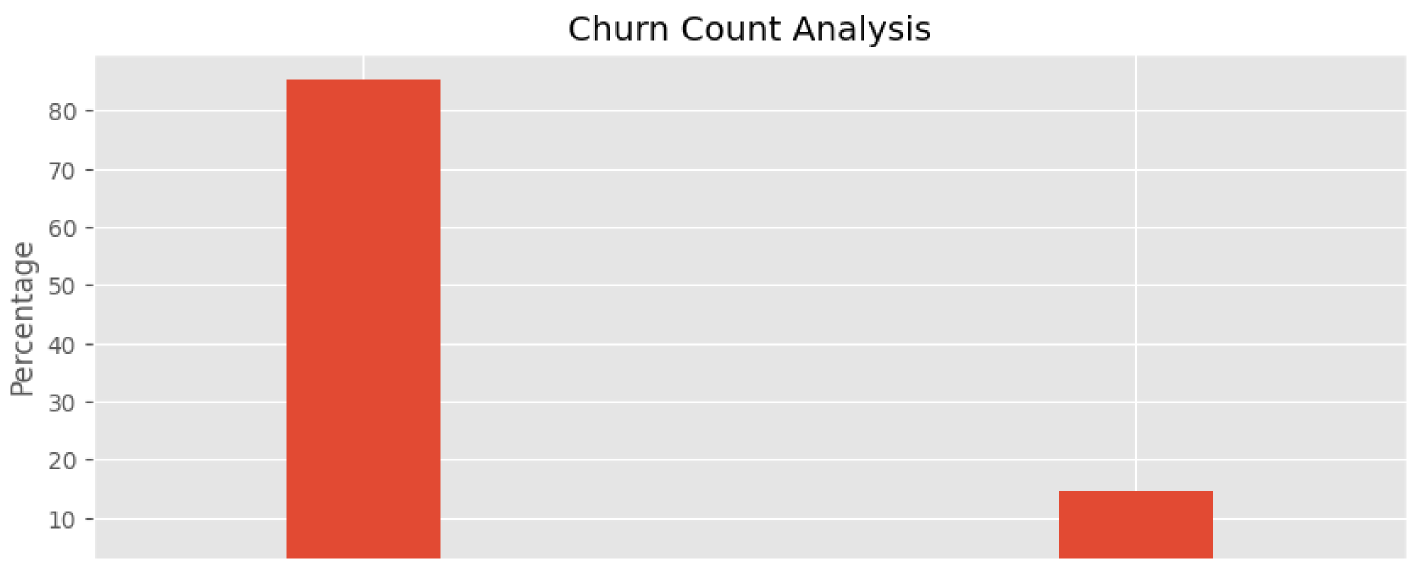
```
df = df.drop(['phone number'], axis=1)
```

Analysis on the churn feature

In [1607]:

```
churn_count = df.churn.value_counts(normalize=True)*100
print(churn_count.round(2))
churn_count.plot(kind='bar', figsize=(10,4), ylabel='Percentage', title='Churn Count Ana
lysis', width=0.2);
```

```
churn
False    85.51
True     14.49
Name: proportion, dtype: float64
```



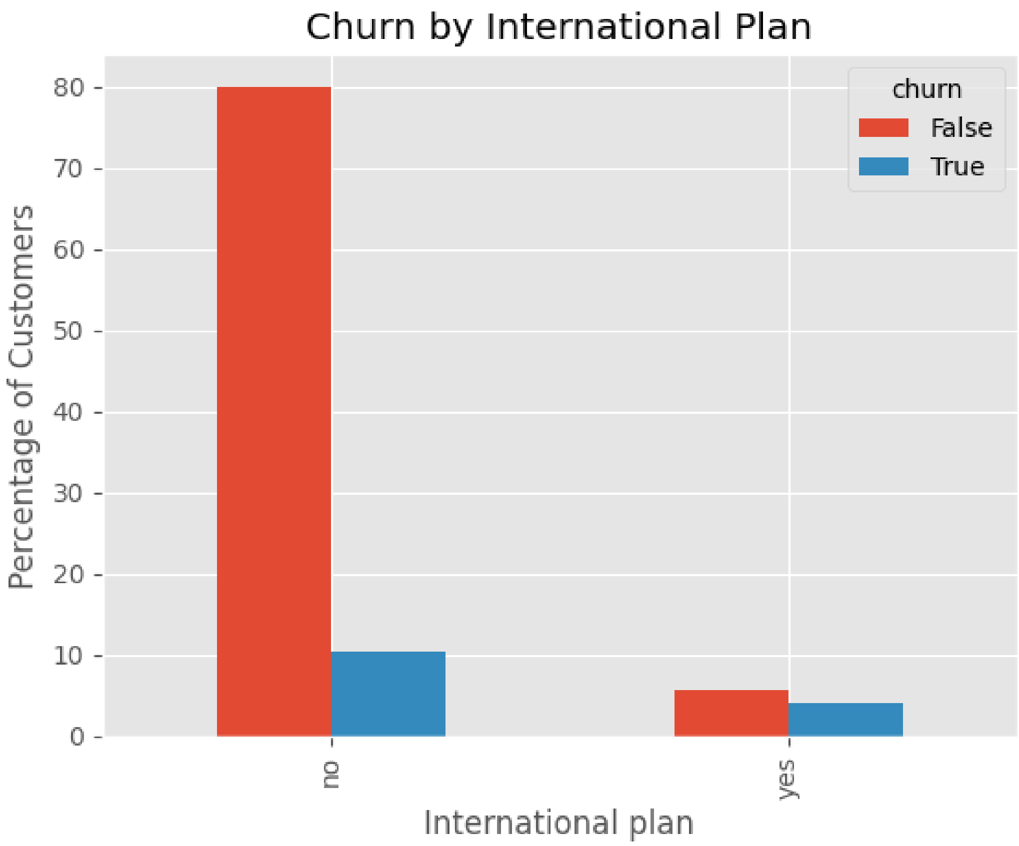
> **The analysis of the churn variable reveals that 85.51% of customers do not churn, while 14.49% of customers churn from the company.**
>
> **This indicates an imbalance in the distribution of the binary classes. To address this issue and prevent the model from making false predictions, we will need to apply class imbalance treatment techniques.**

Churn by International Plan

In [1608]:

```
# Churn distribution per international_plan
international_plan_churn = pd.crosstab(df['international plan'], df['churn'], normalize=
True)*100
print(international_plan_churn.round(2))
international_plan_churn.plot(kind='bar', xlabel='International plan', ylabel='Percentage
of Customers', title='Churn by International Plan')
plt.show()
```

```
churn                False  True
international plan
no                   79.93  10.38
yes                   5.58   4.11
```



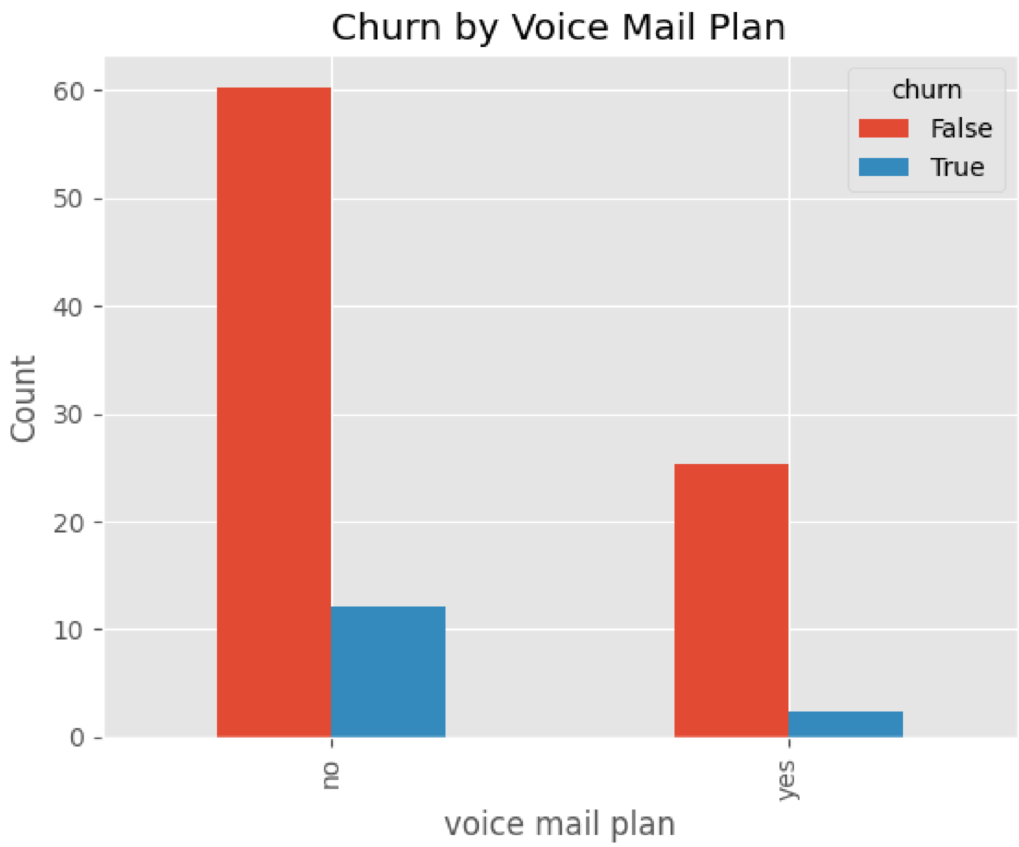
Based on the bargraph above, it is evident that customers without an international plan have a higher percentage in both the 'False' and 'True' categories compared to customers with an international plan. This suggests that having an international plan may be associated with a lower likelihood of churn.

Churn by Voice Mail Plan

In [1609]:

```
# Churn by Voice Mail Plan
vm_plan_churn = pd.crosstab(df['voice mail plan'], df['churn'], normalize=True)*100
print(vm_plan_churn.round(2))
vm_plan_churn.plot(kind='bar', xlabel='voice mail plan', ylabel='Count', title='Churn by Voice Mail Plan')
plt.show()
```

```
churn              False   True
voice mail plan
no                 60.25  12.09
yes                25.26   2.40
```



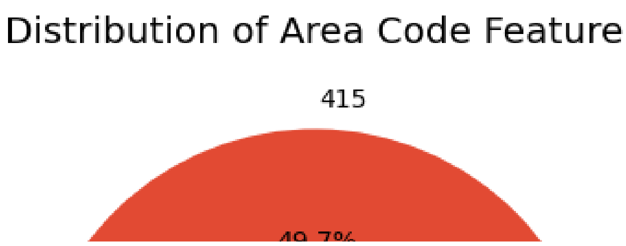
From the graph above, it can be observed that the majority of customers who do not have a voice mail plan are in the 'False' category, while a smaller proportion is in the 'True' category. In addition, customers with a voice mail plan have a higher count in the 'False' category compared to the 'True' category. This may suggest that having a voice mail plan may have some influence on reducing churn,

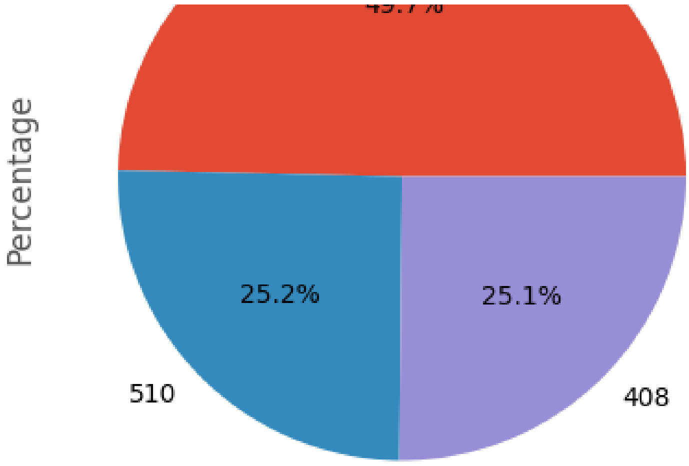
- **Distribution of Area Code Feature**

In [1610]:

```
area_code_counts = df['area code'].value_counts()
print(area_code_counts)
area_code_counts.plot(kind='pie', ylabel="Percentage", title='Distribution of Area Code Feature', figsize=(10,5), autopct='%1.1f%%');
```

```
area code
415    1655
510     840
408     838
Name: count, dtype: int64
```





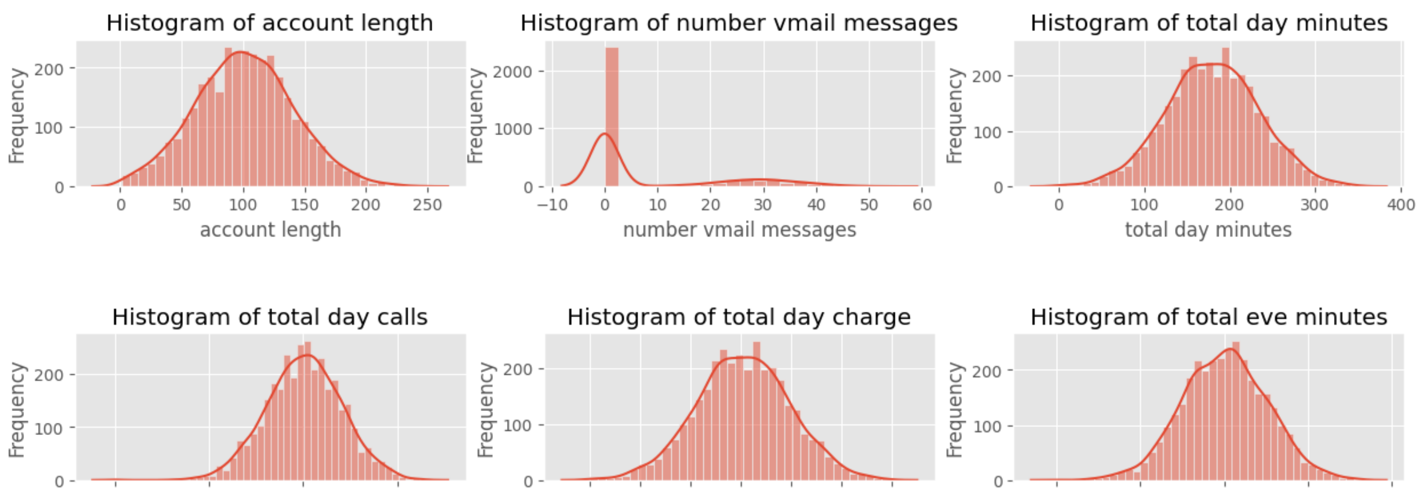
Based on the distribution of area codes, the pie chart shows that 49.7% of customers come from area code 415, which is approximately half of the total number of customers. The remaining customers are evenly distributed between area codes 510 and 408, with each accounting for about 25% of the customer base. This clearly shows that SyriaTel Telecommunications has a huge funbase at area code 415, thus they should capitalize on that area mostly.

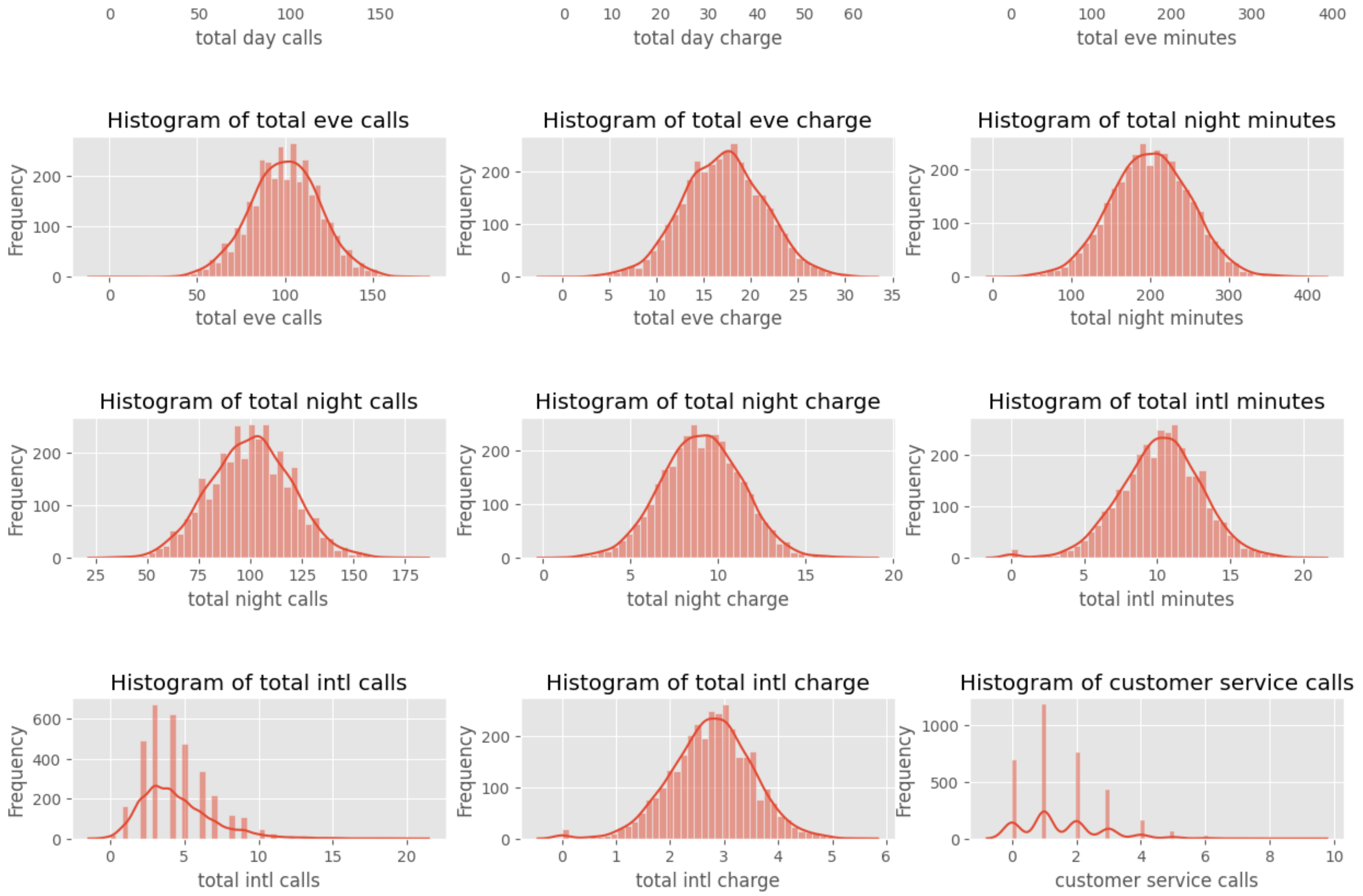
Distribution of numeric variables

The distribution of numeric variables provides insights into the spread and central tendencies of the data. It helps us understand the range, variability, and shape of the variables. Analyzing the distribution can aid in identifying outliers, skewness, or patterns in the data.

In [1611]:

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

fig, ax = plt.subplots(5, 3, figsize=(15, 15))
rows = [0, 1, 2, 3, 4, 5]
columns = [0, 1, 2]

numeric_columns_copy = numeric_columns.drop('area code', axis=1)
for i, column in enumerate(numeric_columns_copy.columns):
    row = i // 3
    col = i % 3

    sns.histplot(numeric_columns_copy[column], ax=ax[row][col],kde=True, kde_kws=dict(cu
t=3))
    ax[row][col].set_title(f"Histogram of {column}")
    ax[row][col].set_xlabel(column)
    ax[row][col].set_ylabel("Frequency")

plt.subplots_adjust(hspace=1)   # Adjust vertical spacing between rows
plt.show()
```





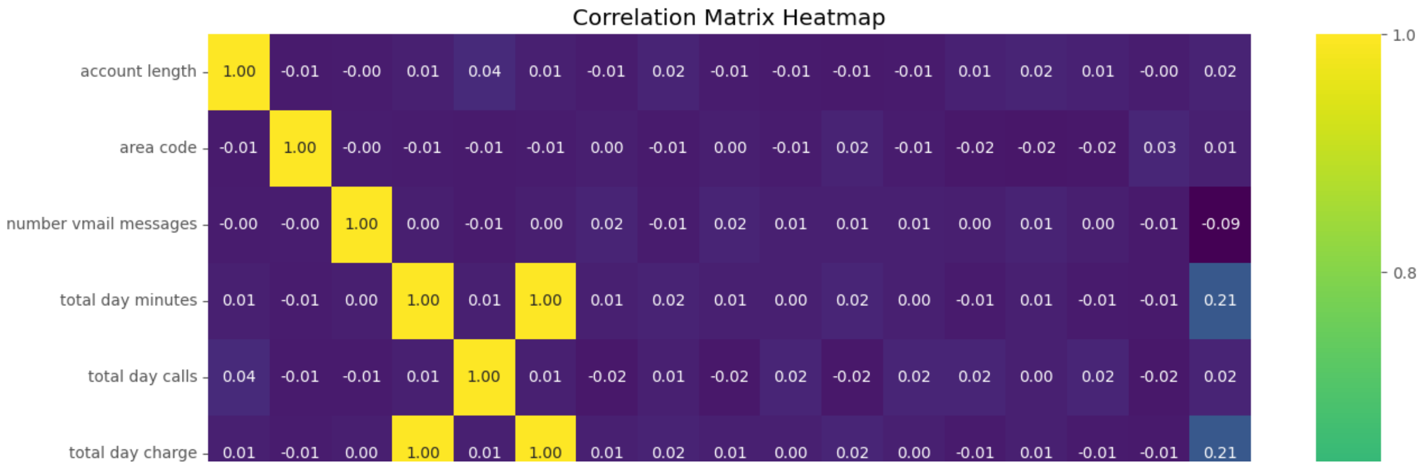
> **The numerical variables exhibit diverse distributions and ranges, indicating variations in customer behavior and call patterns. While some variables follow approximately normal distributions, others display skewed distributions. This suggests that the variables may require different handling approaches based on their distributions for further analysis and modeling.**

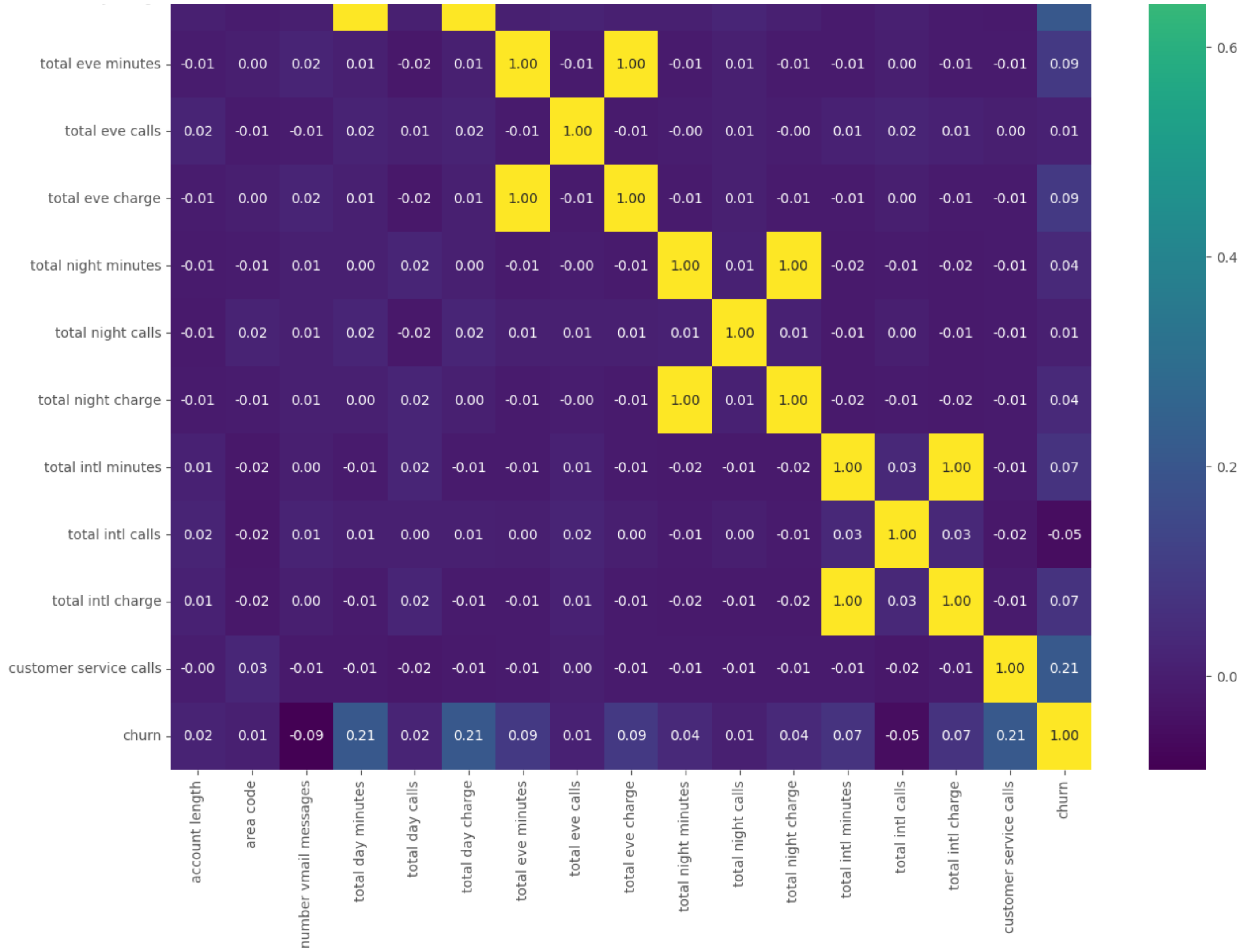
- **Correlation Matrix**

> **The correlation matrix reveals the relationships between variables, indicating how they are associated with each other.**

In [1612]:

```
import seaborn as sns
corr_matrix = df.corr(numeric_only=True)
plt.figure(figsize=(15,15))
sns.heatmap(corr_matrix, annot=True, cmap='viridis', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```





From the above correlation matrix, we can observe that most of the variables are not strongly correlated. However, there ares some variables that exhibit a perfect correlation. This makes sense since some variables are directly correlated. For example, tel charges are directly correlated to the time spent on call.

Data Processing

To prepare our data for modeling, we will:

- **Remove outliers from the dataset**
- **Address multicollinearity by removing highly correlated features**
- **Perform scaling on the numerical variables**
- **Handle class imbalance using the SMOTE method**
- **Proceed with building models for predictions.**

Removing outliers

In [1613]:

```
from scipy import stats

def drop_numerical_outliers(df, z_thresh=3):
    z_scores = stats.zscore(df.select_dtypes(include=np.number))
    mask = (np.abs(z_scores) < z_thresh).all(axis=1)
    df.drop(df.index[~mask], inplace=True)

print("Before dropping numerical outliers, length of the dataframe is: ", len(df))
drop_numerical_outliers(df)
print("After dropping numerical outliers, length of the dataframe is: ", len(df))
```

```
Before dropping numerical outliers, length of the dataframe is:  3333
After dropping numerical outliers, length of the dataframe is:  3169
```

Addressing Multicollinearity

The correlation matrix heatmap previously plotted revealed that, there exists multicollinearity in our variables. Therefore, it is crucial to address multicollinearity to ensure the reliability and accuracy of our models.

In [1614]:

```
print("The original dataframe has {} columns.".format(df.shape[1]))
# Calculate the correlation matrix and take the absolute value
corr_matrix = df.corr(numeric_only=True).abs()

# Create a True/False mask and apply it
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask)

# List column names of highly correlated features (r > 0.90)
to_drop = [c for c in tri_df.columns if any(tri_df[c] >  0.90)]

df = df.drop(to_drop, axis=1) # Drop the features
print("The reduced dataframe has {} columns.".format(df.shape[1]))
```

```
The original dataframe has 20 columns.
The reduced dataframe has 16 columns.
```

Dealing with categorical variables

In [1615]:

```
#Converting churn to binary
df['churn'] = pd.DataFrame(df['churn'].map({False: 0, True: 1}))
```

In [1616]:

```
# Convert 'State' 'International Plan', 'Voice mail Plan' to numeric using one hot encodi
ng technique
cat_cols = ['state', 'international plan', 'voice mail plan']
df = pd.get_dummies(df, columns=cat_cols, dtype=float)
df.head()
```

Out[1616]:

	account length	area code	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	...	state_VA	state_VT	state_WA	state_WI	st
0	128	415	25	110	45.07	99	16.78	91	11.01	3	...	0.0	0.0	0.0	0.0	
1	107	415	26	123	27.47	103	16.62	103	11.45	3	...	0.0	0.0	0.0	0.0	
2	137	415	0	114	41.38	110	10.30	104	7.32	5	...	0.0	0.0	0.0	0.0	
3	84	408	0	71	50.90	88	5.26	89	8.86	7	...	0.0	0.0	0.0	0.0	
4	75	415	0	113	28.34	122	12.61	121	8.41	3	...	0.0	0.0	0.0	0.0	

5 rows × 68 columns

Scaling Numerical Features

In [1617]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_columns = df.select_dtypes(include=[np.number]).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
df.head()
```

Out[1617]:

	account length	area code	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	...	state_VA	state_VT	st
0	0.587963	0.068627	0.510204	0.576271	0.773956	0.487179	0.490082	0.422414	0.643644	0.2	...	0.0	0.0	
1	0.490741	0.068627	0.530612	0.686441	0.450248	0.521368	0.483858	0.525862	0.675974	0.2	...	0.0	0.0	
2	0.629630	0.068627	0.000000	0.610169	0.706088	0.581197	0.238040	0.534483	0.372520	0.4	...	0.0	0.0	
3	0.384259	0.000000	0.000000	0.245763	0.881184	0.393162	0.042007	0.405172	0.485672	0.6	...	0.0	0.0	
4	0.342593	0.068627	0.000000	0.601695	0.466250	0.683761	0.327888	0.681034	0.452608	0.2	...	0.0	0.0	

5 rows × 68 columns

Dataset Splitting

To begin the modeling process, we need to split the dataset into features and the target variable. This will allow us to train our models on the features and make predictions for the target variable.

In [1618]:

```
# Split the dataset into features and target variable
X = df.drop('churn', axis=1)
y = df['churn']
```

In [1619]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Applying SMOTE Technique to Resolve Unbalanced 'churn' Feature

SMOTE (Synthetic Minority Oversampling Technique) is an oversampling technique used to address class imbalance in a dataset. It involves generating synthetic samples for the minority class by interpolating between neighboring instances. This helps to overcome the overfitting issue that can arise from random oversampling.

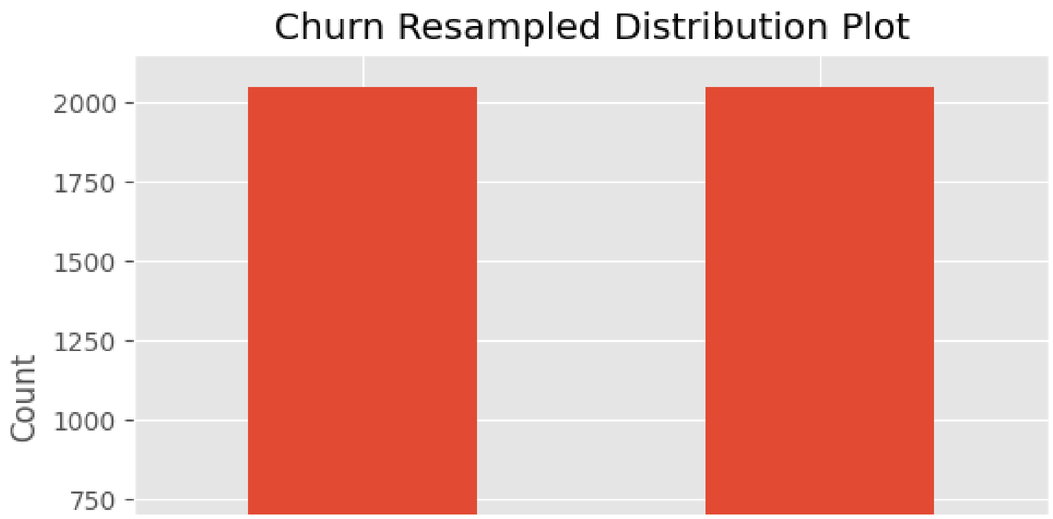
In [1620]:

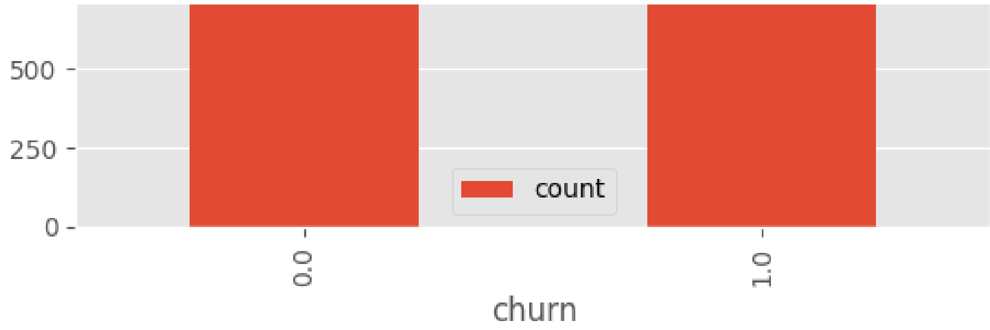
```
from imblearn.over_sampling import SMOTE
X_train_resampled, y_train_resampled = SMOTE(k_neighbors=5, random_state=123).fit_resamp
le(X_train, y_train)
```

Churn Distribution Plot after Resampling

In [1621]:

```
df_y_resampled = pd.DataFrame(y_train_resampled.value_counts())
df_y_resampled.plot(kind='bar', title='Churn Resampled Distribution Plot', ylabel="Count"
);
```





Modeling

In the modeling step, we will train and evaluate different machine learning models on our dataset to make predictions for the target variable. This involves selecting appropriate algorithms, tuning their parameters, and assessing their performance using various evaluation metrics. The goal is to find the model that best captures the patterns and relationships in the data and provides accurate predictions.

Model 1 - Logistic Regression

importing and fitting the model

In [1622]:

```
# Import the necessary classifiers
from sklearn.linear_model import LogisticRegression
# Create a logistic regression model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')

# Fit the model on the resampled training data
logreg.fit(X_train_resampled, y_train_resampled)
```

Out[1622]:

```
▾ LogisticRegression
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinear')
```

Calculating the model Evalution metrics

In [1623]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_
auc_score

# Predict the target variable for the test data
y_pred = logreg.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
logreg_metrics_df = pd.DataFrame({
    "Model Evaluation Metric": ["Accuracy", "Precision", "Recall", "F1 Score", "ROC AUC
Score"],
    "Model Evaluation Score": [accuracy, precision, recall, f1, roc_auc]
})

logreg_metrics_df
```

Out[1623]:

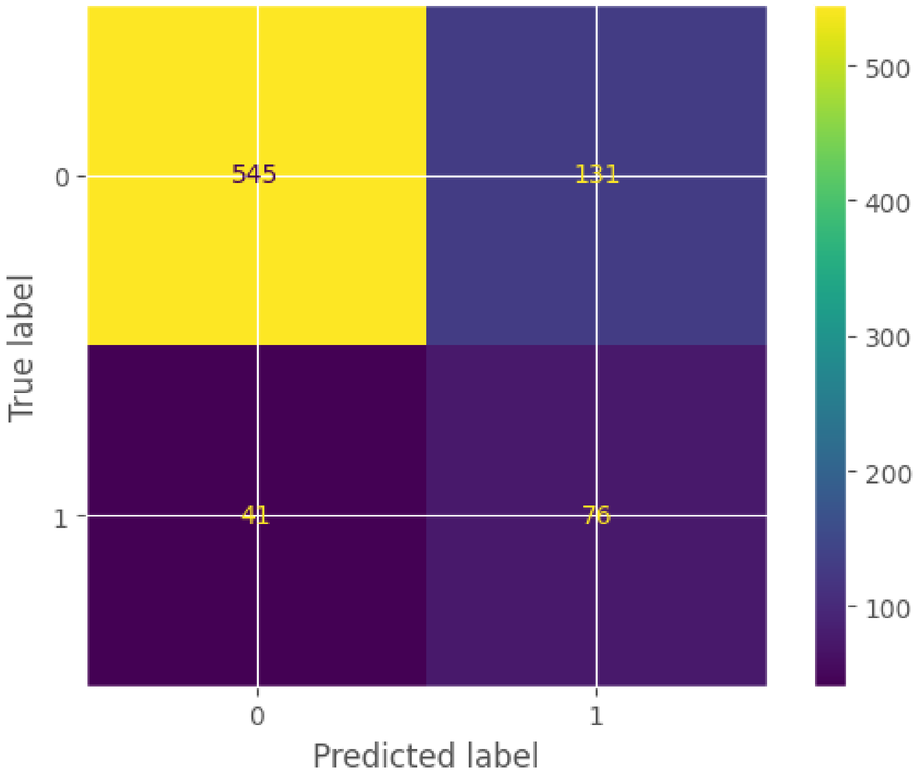
	Model Evaluation Metric	Model Evaluation Score

	Model Evaluation Metric	Model Evaluation Score
0	Accuracy	0.783102
1	Precision	0.367150
2	Recall	0.649573
3	F1 Score	0.469136
4	ROC AUC Score	0.727893

Generating a confusion Matrix

In [1624]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

log_reg_cfm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=log_reg_cfm)
disp.plot();
```



Observations

> **An accuracy of 0.783102 suggests that approximately 78.31% of the predictions made by the model were correct.**
>
> **A precision of 0.367150 indicates that around 36.71% of the predicted positive cases were actually true positive cases.**
>
> **A recall of 0.649573 suggests that the model captured approximately 64.95% of the actual positive cases.**
>
> **F1 Score of 0.469136 indicates the balance between precision and recall in the model's performance. It means that atleast the model will predict 46.91% of the values correctly**
>
> **ROC AUC score of 0.727893 suggests that the model has a moderate level of discrimination power in distinguishing between the two classes.**

Model 2: Random Forest

importing the classifier method and fitting the training set to our model

In [1625]:

```
# import the Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Instantiate the classifier
rf = RandomForestClassifier(n_estimators=100, random_state=123)

# Fit the model to the train set
rf.fit(X_train_resampled, y_train_resampled)
```

Out[1625]:

```
▾        RandomForestClassifier
RandomForestClassifier(random_state=123)
```

Evaluating the models perfomance

In [1626]:

```
# make predictions for the X_test
y_pred = rf.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
rf_metric_df = pd.DataFrame({
    "Model Evaluation Metric": ["Accuracy", "Precision", "Recall", "F1 Score", "ROC AUC
Score"],
    "Model Evaluation Score": [accuracy, precision, recall, f1, roc_auc]
})

rf_metric_df
```

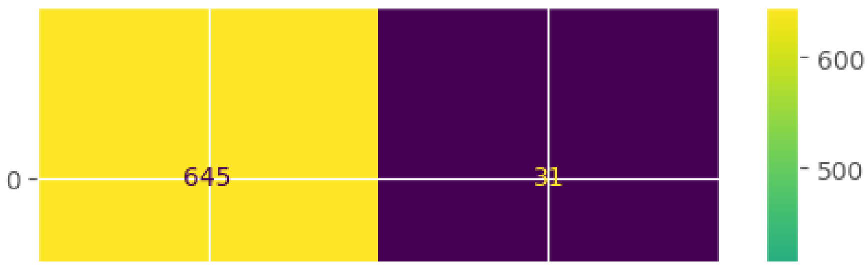
Out[1626]:

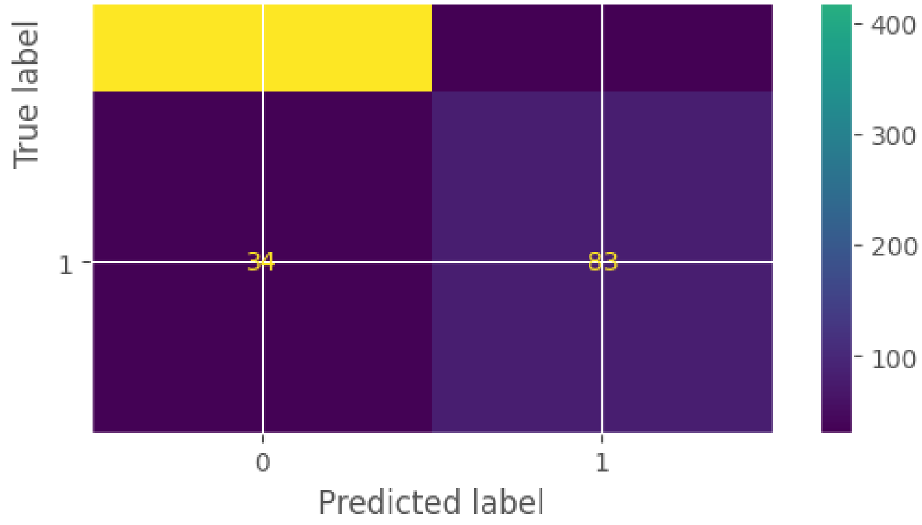
	Model Evaluation Metric	Model Evaluation Score
0	Accuracy	0.918033
1	Precision	0.728070
2	Recall	0.709402
3	F1 Score	0.718615
4	ROC AUC Score	0.831772

Generating a Confusion Matrix for RandomForest Classifier Model

In [1627]:

```
rf_cfm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=rf_cfm)
disp.plot();
```





Observations

An accuracy of 0.918033 suggests that approximately 91.80% of the predictions made by the model were correct.

> **A precision of 0.728070 indicates that around 72.81% of the predicted positive cases were actually true positive cases.**
>
> **A recall of 0.709402 suggests that the model captured approximately 70.94% of the actual positive cases.**
>
> **F1 Score of 0.718615 indicates the balance between precision and recall in the model's performance. It means that atleast the model will predict 71.86% of the values correctly**
>
> **ROC AUC score of 0.831772 suggests that the model has a moderate level of discrimination power in distinguishing between the two classes.**

Model 3: Support Vector Classifier

importing the classifier method and fitting the training set to our model

In [1628]:

```
# mport the SVC classifier model
from sklearn.svm import SVC

# Instatiate it
svc = SVC()

#Fit the training test to the model
svc.fit(X_train_resampled, y_train_resampled)
```

Out[1628]:

```
▾ SVC
SVC()
```

Evaluation the SVC model performance

In [1629]:

```
# make predictions for the X_test
y_pred = svc.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
svc_metric_df = pd.DataFrame({
    "Model Evaluation Metric": ["Accuracy", "Precision", "Recall", "F1 Score", "ROC AUC 
Score"],
    "Model Evaluation Score": [accuracy, precision, recall, f1, roc_auc]
})

svc_metric_df
```

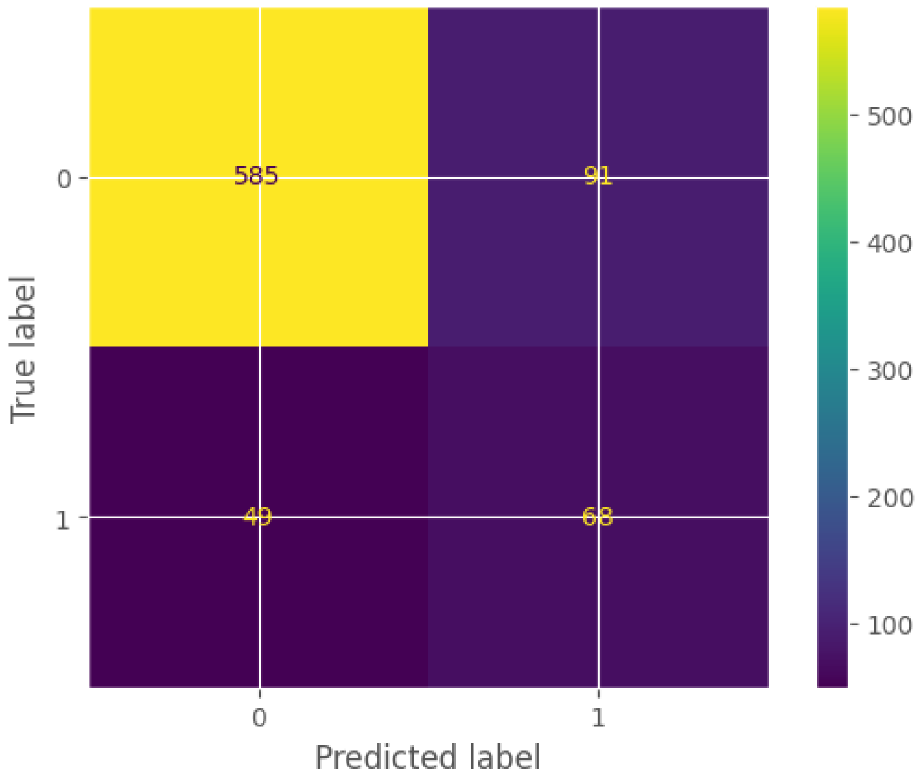
Out[1629]:

	Model Evaluation Metric	Model Evaluation Score
0	Accuracy	0.823455
1	Precision	0.427673
2	Recall	0.581197
3	F1 Score	0.492754
4	ROC AUC Score	0.723291

Generating a Confusion Matrix for SVC model

In [1630]:

```
svc_cfm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=svc_cfm)
disp.plot();
```



Observations

> The SVC model achieved an accuracy of 0.823455, indicating that approximately 82.35% of the predictions made by the model were correct.
>
> The precision score of 0.427673 suggests that around 42.77% of the predicted positive cases were actually true positive cases.

The recall score of 0.581197 indicates that the model captured approximately 58.12% of the actual positive cases.

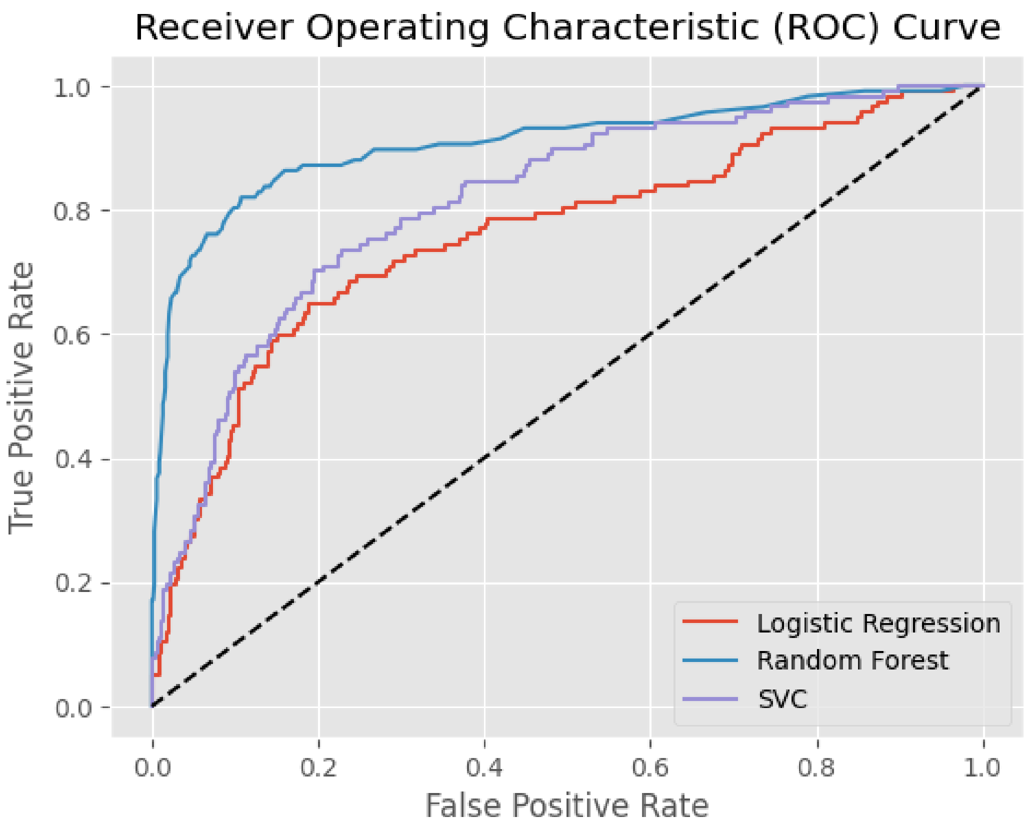
The F1 score of 0.492754 represents the balance between precision and recall, with a higher value indicating a better trade-off between the two.

Lastly, the ROC AUC score of 0.723291 suggests that the model has a moderate level of discrimination power in distinguishing between the two classes.

Plot the ROC Curve for the three models to determine hich one performs best

In [1631]:

```
from sklearn.metrics import roc_curve

# Logistic Regression
logreg_probs = logreg.predict_proba(X_test)[:, 1]
logreg_fpr, logreg_tpr, _ = roc_curve(y_test, logreg_probs)

# Random Forest
rf_probs = rf.predict_proba(X_test)[:, 1]
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)

# SVC
svc_probs = svc.decision_function(X_test)
svc_fpr, svc_tpr, _ = roc_curve(y_test, svc_probs)

# Plotting the ROC curves
plt.plot(logreg_fpr, logreg_tpr, label='Logistic Regression')
plt.plot(rf_fpr, rf_tpr, label='Random Forest')
plt.plot(svc_fpr, svc_tpr, label='SVC')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



Conclusion

Conclusion

Based on the above ROC curves, the Random Forest model demonstrates the best performance among the three models. It exhibits the highest area under the curve (AUC), indicating a better ability to distinguish between the positive and negative classes. The Random Forest model's ROC curve is closer to the top-left corner, which signifies a higher true positive rate and a lower false positive rate. This suggests that the Random Forest model has a better balance between sensitivity and specificity, making it the most effective model for predicting the target variable in this scenario.