

Load the datafile

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

import warnings
warnings.filterwarnings('ignore')
```

Import Packages

```
# IMPORT PACKAGES
import pandas as pd
import matplotlib.pyplot as plt
```

Read the file

[+ Code](#) [+ Text](#)

```
# Read the data file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/customer_data_edited.csv')
```

```
data.head()
```

	recordID	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_
0	1	HI	101	510		no	no	0	70.9
1	2	MT	137	510		no	no	0	223.6
2	3	OH	103	408		no	yes	29	294.7
3	4	NM	99	415		no	no	0	216.8
4	5	SC	108	415		no	no	0	197.4

5 rows × 22 columns

Drop the irrelevant columns

```
# Drop the specified columns from the dataset
data = data.drop(['recordID', 'state', 'area_code', 'customer_id'], axis=1)

# Display the first 5 rows of the updated dataset
data.head()
```

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
0	101		no	no	0	70.9	123
1	137		no	no	0	223.6	86
2	103		no	yes	29	294.7	95
3	99		no	no	0	216.8	123
4	108		no	no	0	197.4	78

Visualizations

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

# Visualizations
fig, axes = plt.subplots(3, 2, figsize=(14, 18))

# Histograms for call details
sns.histplot(data['total_day_minutes'], bins=30, kde=True, ax=axes[0, 0])
axes[0, 0].set_title('Distribution of Total Day Minutes')

sns.histplot(data['total_eve_minutes'], bins=30, kde=True, ax=axes[0, 1])
axes[0, 1].set_title('Distribution of Total Evening Minutes')

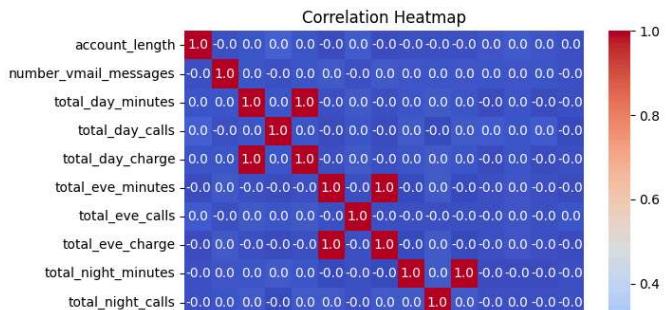
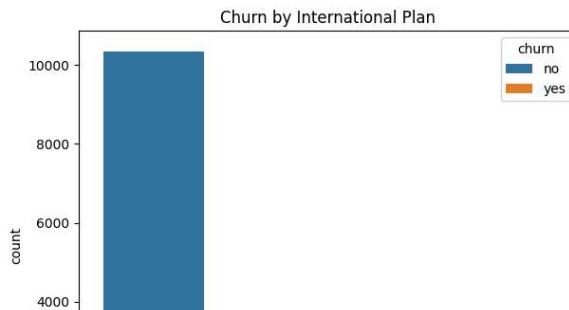
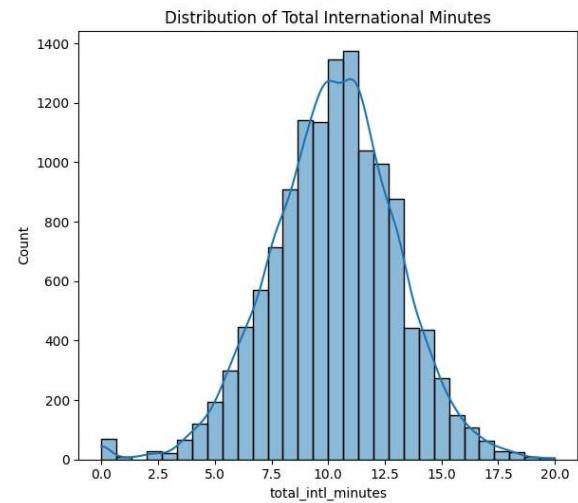
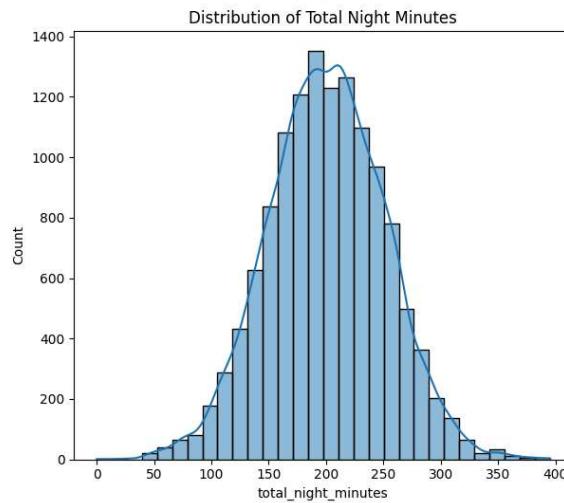
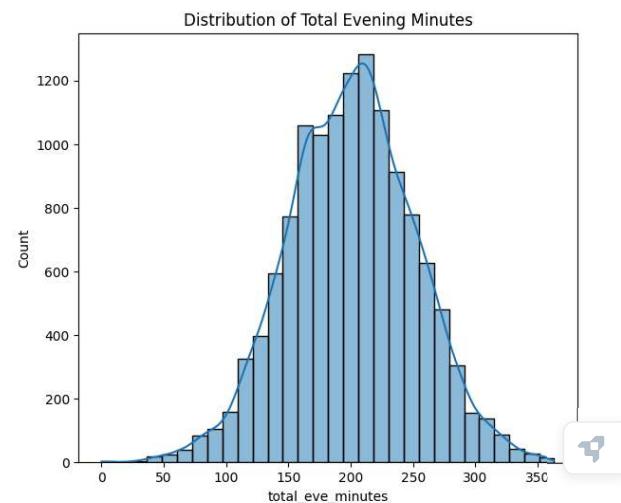
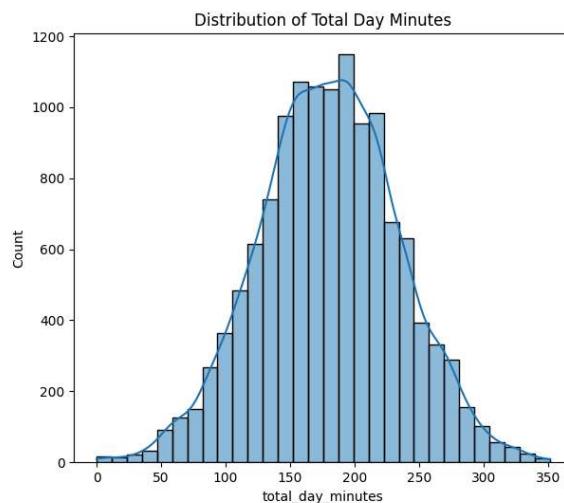
sns.histplot(data['total_night_minutes'], bins=30, kde=True, ax=axes[1, 0])
axes[1, 0].set_title('Distribution of Total Night Minutes')

sns.histplot(data['total_intl_minutes'], bins=30, kde=True, ax=axes[1, 1])
axes[1, 1].set_title('Distribution of Total International Minutes')

# Bar chart for churn rates by international plan
sns.countplot(x='international_plan', hue='churn', data=data, ax=axes[2, 0])
axes[2, 0].set_title('Churn by International Plan')

# Correlation heatmap
corr = data.select_dtypes(include=[np.number]).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".1f", ax=axes[2, 1])
axes[2, 1].set_title('Correlation Heatmap')

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



Missing Values

```
# Step 1: Handling Missing Values
# Checking for any missing values in the dataset
missing_data = data.isnull().sum()
missing_data
```

account_length	0
international_plan	0
voice_mail_plan	0
number_vmail_messages	0
total_day_minutes	0
total_day_calls	0
total_day_charge	0
total_eve_minutes	0
total_eve_calls	0
total_eve_charge	0
total_night_minutes	0
total_night_calls	0
total_night_charge	0
total_intl_minutes	0
total_intl_calls	0
total_intl_charge	0
number_customer_service_calls	0
churn	0

dtype: int64

Data Type Conversions

```
# Step 2: Data Type Conversions
# Correcting data types: converting 'yes'/'no' to binary format for 'international_plan' and 'voice_mail_plan'
data['international_plan'] = data['international_plan'].map({'yes': 1, 'no': 0})
data['voice_mail_plan'] = data['voice_mail_plan'].map({'yes': 1, 'no': 0})
# Output the number of missing values per column and the head of the dataframe to see the changes
data.head()
```

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
0	101	0	0	0	70.9	123	12.05
1	137	0	0	0	223.6	86	38.01
2	103	0	1	29	294.7	95	50.10
3	99	0	0	0	216.8	123	
4	108	0	0	0	197.4	78	33.56

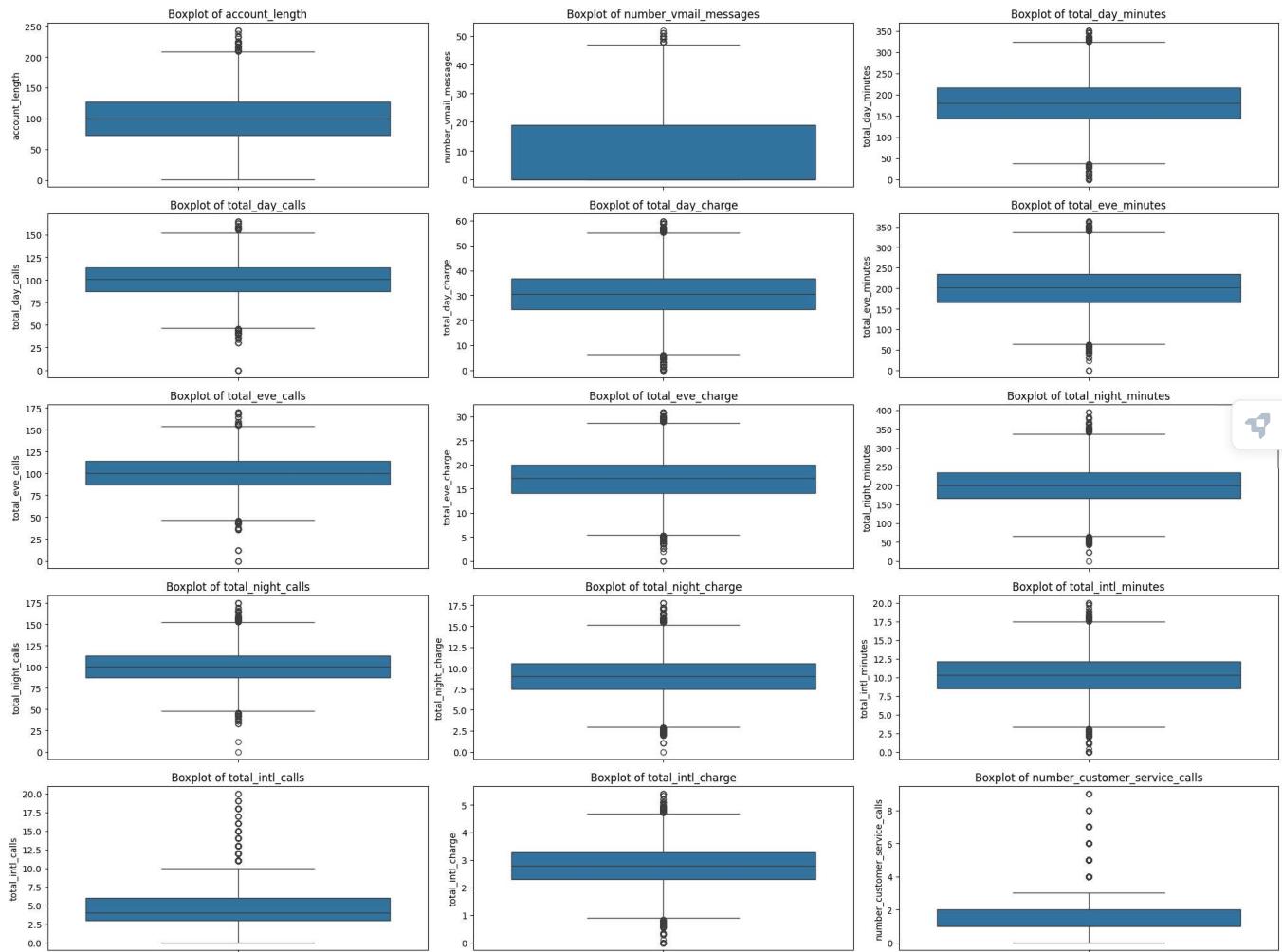
Checking for outliers using Box plot

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

# Identifying outliers using boxplots for key numerical columns
numerical_columns = ['account_length', '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', 'number_customer_service_calls']

# Plotting boxplots for the numerical variables to identify outliers visually
plt.figure(figsize=(20, 15))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(y=data[col])
    plt.title(f'Boxplot of {col}')

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



Handling outliers - using capping and flooring method

```

import numpy as np

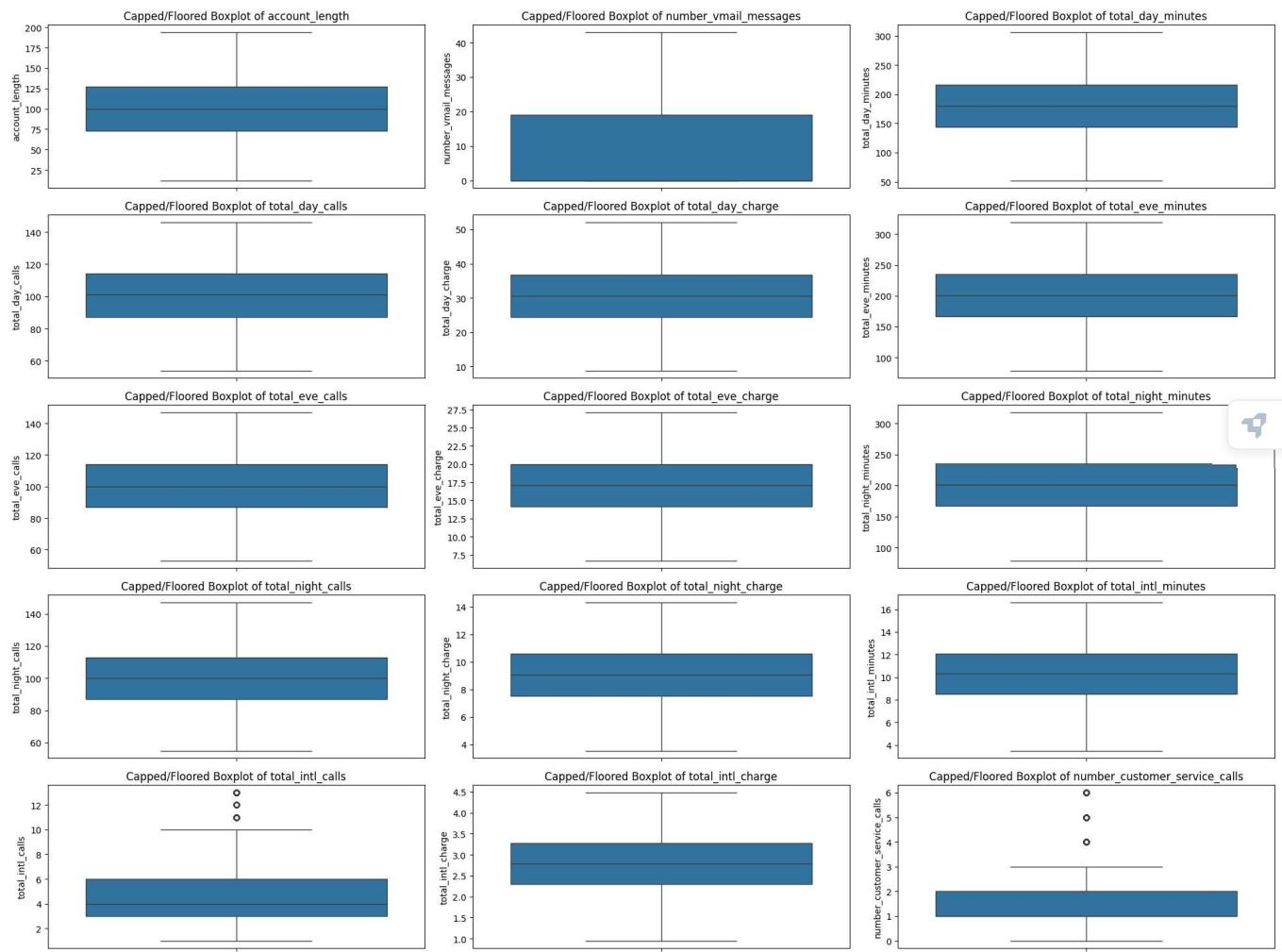
# Reapplying capping and flooring for each numerical column at the 1st and 99th percentiles
for column in numerical_columns:
    # Calculate the 1st and 99th percentiles
    lower_bound = data[column].quantile(0.01)
    upper_bound = data[column].quantile(0.99)

    # Cap and floor the outliers
    data[column] = np.where(data[column] > upper_bound, upper_bound, data[column])
    data[column] = np.where(data[column] < lower_bound, lower_bound, data[column])

# Checking if the capping and flooring has been applied by replotting the boxplots
plt.figure(figsize=(20, 15))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(y=data[col])
    plt.title(f'Capped/Floored Boxplot of {col}')

plt.tight_layout()
plt.show()

```



Normalization

```

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Creating scaler objects
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()

# Selecting numerical columns for scaling (excluding categorical one-hot encoded columns)
numerical_columns = ['international_plan', 'voice_mail_plan']

# Applying Standardization
data_standardized = data.copy()
data_standardized[numerical_columns] = standard_scaler.fit_transform(data[numerical_columns])

# Applying Min-Max Scaling
data_minmax_scaled = data.copy()
data_minmax_scaled[numerical_columns] = minmax_scaler.fit_transform(data[numerical_columns])

# Displaying the first few rows of the standardized and min-max scaled data
data_standardized_head = data_standardized.head()
data_minmax_scaled_head = data_minmax_scaled.head()

data_standardized_head, data_minmax_scaled_head

```

	account_length	international_plan	voice_mail_plan	number_vmail_messages
0	101.0	-0.326366	-0.612851	0.0
1	137.0	-0.326366	-0.612851	0.0

```

2      103.0      -0.326366      1.631717      29.0
3      99.0      -0.326366     -0.612851      0.0
4     108.0      -0.326366     -0.612851      0.0

total_day_minutes  total_day_calls  total_day_charge  total_eve_minutes \
0            70.9        123.0       12.05        211.9
1           223.6         86.0       38.01        244.8
2           294.7         95.0       50.10        237.3
3           216.8        123.0       36.86        126.4
4           197.4         78.0       33.56        124.0

total_eve_calls  total_eve_charge  total_night_minutes  total_night_calls \
0            73.0        18.01       236.0        73.0
1           139.0        20.81       94.2        81.0
2           105.0        20.17      300.3        127.0
3           88.0         10.74      220.6        82.0
4           101.0        10.54      204.5        107.0

total_night_charge  total_intl_minutes  total_intl_calls \
0            10.62        10.6        3.0
1             4.24         9.5        7.0
2            13.51        13.7        6.0
3             9.93        15.7        2.0
4             9.20         7.7        4.0

total_intl_charge  number_customer_service_calls  churn
0            2.86          3.0    no
1            2.57          0.0    no
2            3.70          1.0    no
3            4.24          1.0    no
4            2.08          2.0    no
account_length  international_plan  voice_mail_plan  number_vmail_messages \
0            101.0          0.0        0.0        0.0
1            137.0          0.0        0.0        0.0
2            103.0          0.0        1.0        29.0
3            99.0          0.0        0.0        0.0
4            108.0          0.0        0.0        0.0

total_day_minutes  total_day_calls  total_day_charge  total_eve_minutes \
0            70.9        123.0       12.05        211.9
1           223.6         86.0       38.01        244.8
2           294.7         95.0       50.10        237.3
3           216.8        123.0       36.86        126.4
4           197.4         78.0       33.56        124.0

total_eve_calls  total_eve_charge  total_night_minutes  total_night_calls \
0            73.0        18.01       236.0        73.0
1           139.0        20.81       94.2        81.0
2           105.0        20.17      300.3        127.0
3           88.0         10.74      220.6        82.0
4           101.0        10.54      204.5        107.0

total_night_charge  total_intl_minutes  total_intl_calls \
0            10.62        10.6        3.0
1             4.24         9.5        7.0

```

Checking the data balance in churn column

```

import matplotlib.pyplot as plt
import seaborn as sns

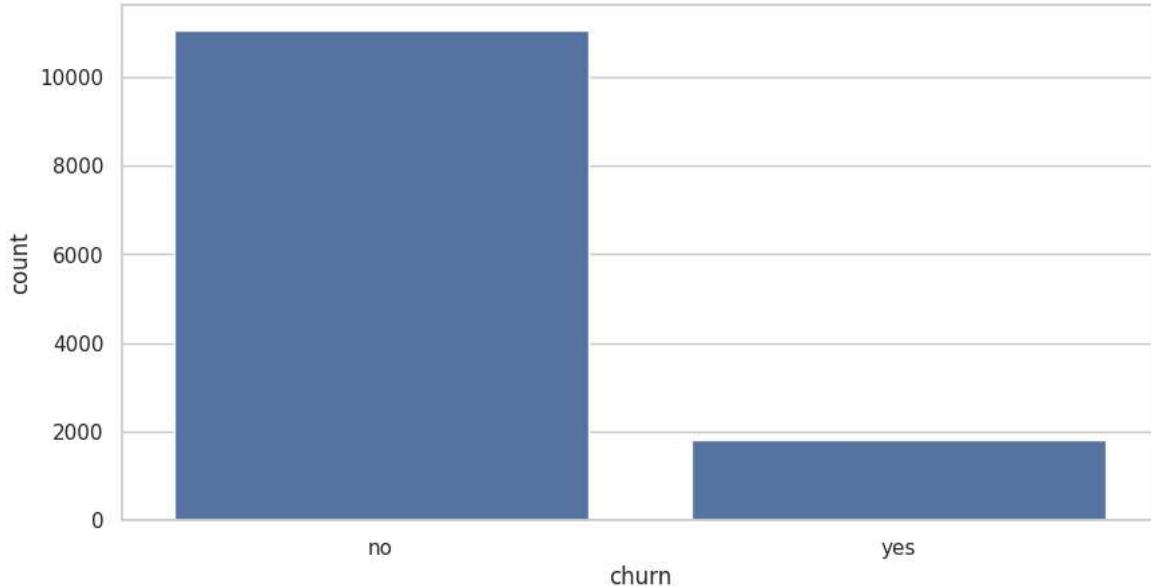
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Distribution of customer churn
plt.figure(figsize=(10, 5))
sns.countplot(x='churn', data=data)
plt.title('Distribution of Customer Churn')

plt.show()

```

Distribution of Customer Churn



Handling Data Imbalance using SMOTE METHOD

```

from imblearn.over_sampling import SMOTE

# Setting up the feature matrix and target vector
X = data.drop('churn', axis=1)
y = data['churn']

# Encoding categorical variables using one-hot encoding
X = pd.get_dummies(X)

# Initializing SMOTE
smote = SMOTE()

# Resampling the dataset
X_resampled, y_resampled = smote.fit_resample(X, y)

# Checking the new class distribution
new_churn_distribution = pd.Series(y_resampled).value_counts(normalize=True)
new_churn_distribution

```

```

churn
no      0.5
yes     0.5
Name: proportion, dtype: float64

```

Visualizations

```

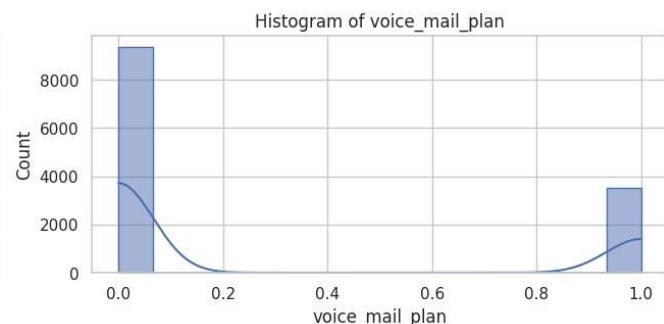
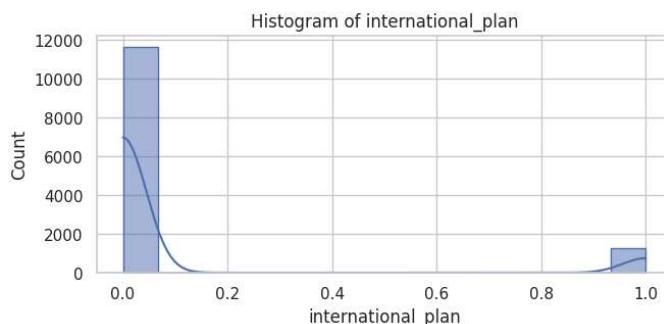
# Generating histograms for numerical variables
plt.figure(figsize=(20, 15))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(5, 3, i)
    sns.histplot(data[col], kde=True, element='step')
    plt.title(f'Histogram of {col}')

plt.tight_layout()
plt.show()

# Summary statistics for numerical variables
summary_statistics = data[numerical_columns].describe()

# Displaying summary statistics
summary_statistics

```



	international_plan	voice_mail_plan
count	12892.000000	12892.000000
mean	0.096261	0.273038
std	0.294961	0.445537
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	1.000000
max	1.000000	1.000000

Logistic regression model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Prepare the data
X = data.drop('churn', axis=1)
y = data['churn'].apply(lambda x: 1 if x == 'yes' else 0) # Convert target to binary

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the Logistic Regression model
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train_scaled, y_train)

# Predicting the test set results
y_pred = logistic_model.predict(X_test_scaled)

# Evaluating the model
accuracy_logistic = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(accuracy_logistic)
print(conf_matrix, report)
```

		precision	recall	f1-score	support
0	0.88	0.97	0.92	2221	
1	0.49	0.18	0.26	358	
accuracy			0.86	2579	
macro avg	0.69	0.57	0.59	2579	
weighted avg	0.83	0.86	0.83	2579	

Hyper parameter tuning for logistic regression model

```

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assuming you've already prepared your feature matrix (X) and target vector (y)
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the model and parameters grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'] # 'saga' supports both l1 and l2 penalty
}

# Setting up the GridSearchCV
grid_search = GridSearchCV(estimator=LogisticRegression(random_state=42),
                           param_grid=param_grid,
                           cv=3, # 3-fold cross-validation
                           scoring='accuracy', # You can choose other metrics
                           verbose=1)

# Fit grid search
grid_search.fit(X_train_scaled, y_train)

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Evaluate the model with best parameters on the test set
best_lr = grid_search.best_estimator_
y_pred = best_lr.predict(X_test_scaled)
print("Accuracy on Test Set:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

```

Fitting 3 folds for each of 20 candidates, totalling 60 fits
 Best Parameters: {'C': 0.001, 'penalty': 'l2', 'solver': 'liblinear'}
 Best Score: 0.8678365486087732

Accuracy on Test Set: 0.8646762310973245

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.99	0.93	2221
1	0.56	0.12	0.19	358
accuracy			0.86	2579
macro avg	0.72	0.55	0.56	2579
weighted avg	0.83	0.86	0.82	2579

Ensamble method using Stacking classifier

```

from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define base models
base_models = [
    ('lr', LogisticRegression(random_state=42)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('svc', SVC(probability=True, random_state=42))
]

# Define a meta-model
meta_model = LogisticRegression(random_state=42)

# Create the stacking ensemble
stacked_model = StackingClassifier(estimators=base_models, final_estimator=meta_model, cv=5)

# Train the stacked model
stacked_model.fit(X_train_scaled, y_train)

# Predict the test set results
y_pred = stacked_model.predict(X_test_scaled)

# Evaluate the model
accuracy_ensemble = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_ensemble)
print("Classification Report:\n", classification_report(y_test, y_pred))

```

```

Accuracy: 0.9965102753005041
Classification Report:
      precision    recall   f1-score   support
          0       1.00     1.00     1.00     2221
          1       0.99     0.98     0.99      358

      accuracy                           1.00     2579
     macro avg       0.99     0.99     0.99     2579
weighted avg       1.00     1.00     1.00     2579

```

Random forest model

```

from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
random_forest_model = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, random_state=42)

# Train the model on the scaled training data
random_forest_model.fit(X_train_scaled, y_train)

# Predicting the test set results
y_pred_rf = random_forest_model.predict(X_test_scaled)

# Evaluating the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print(accuracy_rf)
print(conf_matrix_rf)
print(report_rf)

```

```

0.9965102753005041
[[2221  0]
 [ 9 349]]
      precision    recall   f1-score   support
          0       1.00     1.00     1.00     2221
          1       1.00     0.97     0.99      358

```

accuracy		1.00	2579
macro avg	1.00	0.99	2579
weighted avg	1.00	1.00	2579

Hyper parameter tuning for random forest model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Setting up the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_leaf': [1, 2, 4]
}

# Setting up the GridSearchCV
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=5, # 5-fold cross-validation
                           scoring='accuracy', # Metric of choice
                           verbose=1,
                           n_jobs=-1) # Use all available cores

# Running Grid Search
grid_search.fit(X_train_scaled, y_train)

# Best parameters and best accuracy
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 300}
Best Score: 0.9897214936108133
```

Gradient Boosting classifier

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assume X and y are your feature matrix and target vector, respectively
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Gradient Boosting Classifier
gbm_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gbm_model.fit(X_train_scaled, y_train)

# Predict the test set results
y_pred_gbm = gbm_model.predict(X_test_scaled)

# Evaluate the model
accuracy_gbm = accuracy_score(y_test, y_pred_gbm)
conf_matrix_gbm = confusion_matrix(y_test, y_pred_gbm)
report_gbm = classification_report(y_test, y_pred_gbm)
```

```
Accuracy: 0.9701434664598682
Confusion Matrix:
[[2215    6]
 [ 71  287]]
Classification Report:
precision    recall    f1-score   support
      0       0.97      1.00      0.98     2221
      1       0.98      0.80      0.88      358
      accuracy           0.97      2579
      macro avg       0.97      0.90      0.93      2579
      weighted avg     0.97      0.97      0.97      2579
```

Hyper parameter tuning for Gradient Boosting Classifier

```

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Prepare your features (X) and target (y)
# X_train, X_test, y_train, y_test = ...

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the model and parameters grid
param_grid = {
    'n_estimators': [100, 150],
    'learning_rate': [0.1, 0.05],
    'max_depth': [3, 4],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2]
}

gbm = GradientBoostingClassifier(random_state=42)
grid_search = GridSearchCV(estimator=gbm, param_grid=param_grid, cv=3, scoring='accuracy', verbose=1)

# Fit grid search
grid_search.fit(X_train_scaled, y_train)

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Evaluate the model with best parameters on the test set
best_gbm = grid_search.best_estimator_
y_pred = best_gbm.predict(X_test_scaled)
print("Accuracy on Test Set:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

```

```

Fitting 3 folds for each of 32 candidates, totalling 96 fits
Best Parameters: {'learning_rate': 0.1, 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
Best Score: 0.9747891194666128
Accuracy on Test Set: 0.983714618069019
Classification Report:
      precision    recall  f1-score   support

          0       0.98     1.00     0.99    2221
          1       1.00     0.88     0.94     358

   accuracy                           0.98    2579
    macro avg       0.99     0.94     0.96    2579
weighted avg       0.98     0.98     0.98    2579

```

Support Vector Classifier Model

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assume you've prepared your feature matrix (X) and target vector (y)
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Support Vector Classifier
svm_model = SVC(C=1.0, kernel='rbf', gamma='scale', random_state=42)

# Train the SVM on the scaled training data
svm_model.fit(X_train_scaled, y_train)

# Predict the test set results
y_pred_svm = svm_model.predict(X_test_scaled)
```

Hyper parameter tuning for Support Vector Classifier

```
# Accuracy SVM - Accuracy accuracy/test - y_pred_svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the model and parameters grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto']
}

# Setting up the GridSearchCV
grid_search = GridSearchCV(estimator=SVC(random_state=42), param_grid=param_grid, cv=3, scoring='accuracy', verbose=1)

# Fit grid search
grid_search.fit(X_train_scaled, y_train)

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Evaluate the model with best parameters on the test set
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test_scaled)
```