TeleCo Analytics

Backstory:

In the dynamic landscape of the telecom industry, a formidable challenge emerged – understanding and harnessing customer data for sustainable growth. In 2010, TeleCo Analytics was born in Dallas, Texas, driven by a mission to solve this data dilemma.

The telecom industry was drowning in a sea of customer data, yet struggling to decipher its true significance. TeleCo recognized this as the problem to solve. They embarked on a transformative journey, pioneering data analytics to predict and address customer behavior.

With relentless innovation, they developed a cutting-edge analytics platform. This platform was the answer to the data puzzle, allowing them to predict customer behavior, pinpoint churn factors, and craft precision retention strategies.

Their solution yielded tangible results – reduced churn rates, elevated customer satisfaction, and boosted revenues for telecom partners. Today, TeleCo Analytics continues to be the solution for the telecom industry's data puzzle, predicting customer needs and nurturing enduring connections.

In [1]:
```
# These lines import necessary libraries for data manipulation, visualization, and mac
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Imports specific functionalities from scikit-learn needed for encoding data, splitti
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
# Loads the telecom dataset into a DataFrame called 't'.
t = pd.read_csv("telecomdataset.csv")
```

In [3]:
```
print(t.shape)
t.dtypes
```

```
(3333, 20)
```

Out[3]:

```
state                    object
accountlength             int64
areacode                  int64
internationalplan        object
voicemailplan            object
numbervmailmessages       int64
totaldayminutes         float64
totaldaycalls             int64
totaldaycharge          float64
totaleveningminutes     float64
totaleveningcalls         int64
totaleveningcharge      float64
totalnightminutes       float64
totalnightcalls           int64
totalnightcharge        float64
totalinterminutes       float64
totalintercalls           int64
totalintercharge        float64
customerservicecalls      int64
churn                      bool
dtype: object
```

In [4]:

```python
# Drops any rows with missing values to ensure the quality of the data for analysis.
t= t.dropna()
```

In [5]:

```python
# Initializes a dictionary to hold state abbreviations to unique number mappings.
state_to_number = {}

state_abbreviations = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA

# Maps each state abbreviation to a unique number starting from 1.
for i, state_abbreviation in enumerate(state_abbreviations):
    state_to_number[state_abbreviation] = i + 1  # Adding 1 to start numbering from 1

state_abbreviation_to_convert = "TX"  # Replace with the state abbreviation you want t
if state_abbreviation_to_convert in state_to_number:
    state_number = state_to_number[state_abbreviation_to_convert]
    print(f"{state_abbreviation_to_convert} is assigned the number {state_number}")
else:
    print(f"{state_abbreviation_to_convert} not found in the mapping")
```

```
TX is assigned the number 43
```

In [6]:

```python
state_numbers_to_abbreviations = [(number, abbreviation) for abbreviation, number in s

state_numbers_to_abbreviations.sort()

for state_number, state_abbreviation in state_numbers_to_abbreviations:
    print(f"{state_number}: {state_abbreviation}")
```

```
1: AL
2: AK
3: AZ
4: AR
5: CA
6: CO
7: CT
8: DE
9: FL
10: GA
11: HI
12: ID
13: IL
14: IN
15: IA
16: KS
17: KY
18: LA
19: ME
20: MD
21: MA
22: MI
23: MN
24: MS
25: MO
26: MT
27: NE
28: NV
29: NH
30: NJ
31: NM
32: NY
33: NC
34: ND
35: OH
36: OK
37: OR
38: PA
39: RI
40: SC
41: SD
42: TN
43: TX
44: UT
45: VT
46: VA
47: WA
48: WV
49: WI
50: WY
```

In [7]:

```
t.dtypes
```

Out[7]:
```
state                     object
accountlength              int64
areacode                   int64
internationalplan         object
voicemailplan             object
numbervmailmessages        int64
totaldayminutes          float64
totaldaycalls              int64
totaldaycharge           float64
totaleveningminutes      float64
totaleveningcalls          int64
totaleveningcharge       float64
totalnightminutes        float64
totalnightcalls            int64
totalnightcharge         float64
totalinterminutes        float64
totalintercalls            int64
totalintercharge         float64
customerservicecalls       int64
churn                       bool
dtype: object
```

In [8]:
```python
# Encodes categorical string variables into a numeric format suitable for modeling.
label_encoder = LabelEncoder()
for col in t.columns:
    if t[col].dtype == 'object':
        t[col] = label_encoder.fit_transform(t[col])
```

In [9]:
```python
t.dtypes
```

Out[9]:
```
state                      int32
accountlength              int64
areacode                   int64
internationalplan          int32
voicemailplan              int32
numbervmailmessages        int64
totaldayminutes          float64
totaldaycalls              int64
totaldaycharge           float64
totaleveningminutes      float64
totaleveningcalls          int64
totaleveningcharge       float64
totalnightminutes        float64
totalnightcalls            int64
totalnightcharge         float64
totalinterminutes        float64
totalintercalls            int64
totalintercharge         float64
customerservicecalls       int64
churn                       bool
dtype: object
```

In [12]:
```python
# Creates a new binary column 'churn1' indicating churn status.
t['churn1'] = np.where(t['churn'] == 1, 1, 0)

# Removes the original 'churn' column after encoding it.
t = t.drop(columns=['churn'])
```

In [27]:
```python
state_abbreviation_to_convert = "DC"  # Replace with the state abbreviation you want t
if state_abbreviation_to_convert in state_to_number:
    state_number = state_to_number[state_abbreviation_to_convert]
    print(f"{state_abbreviation_to_convert} is assigned the number {state_number}")
else:
    print(f"{state_abbreviation_to_convert} not found in the mapping")
```

```
DC not found in the mapping
```

In [16]:
```python
print(t.state)
```

```
0       18
1       15
2       34
3       40
4       11
        ..
3328    40
3329     3
3330    49
3331    39
3332    42
Name: state, Length: 3333, dtype: int32
```

In [26]:
```python
t
```

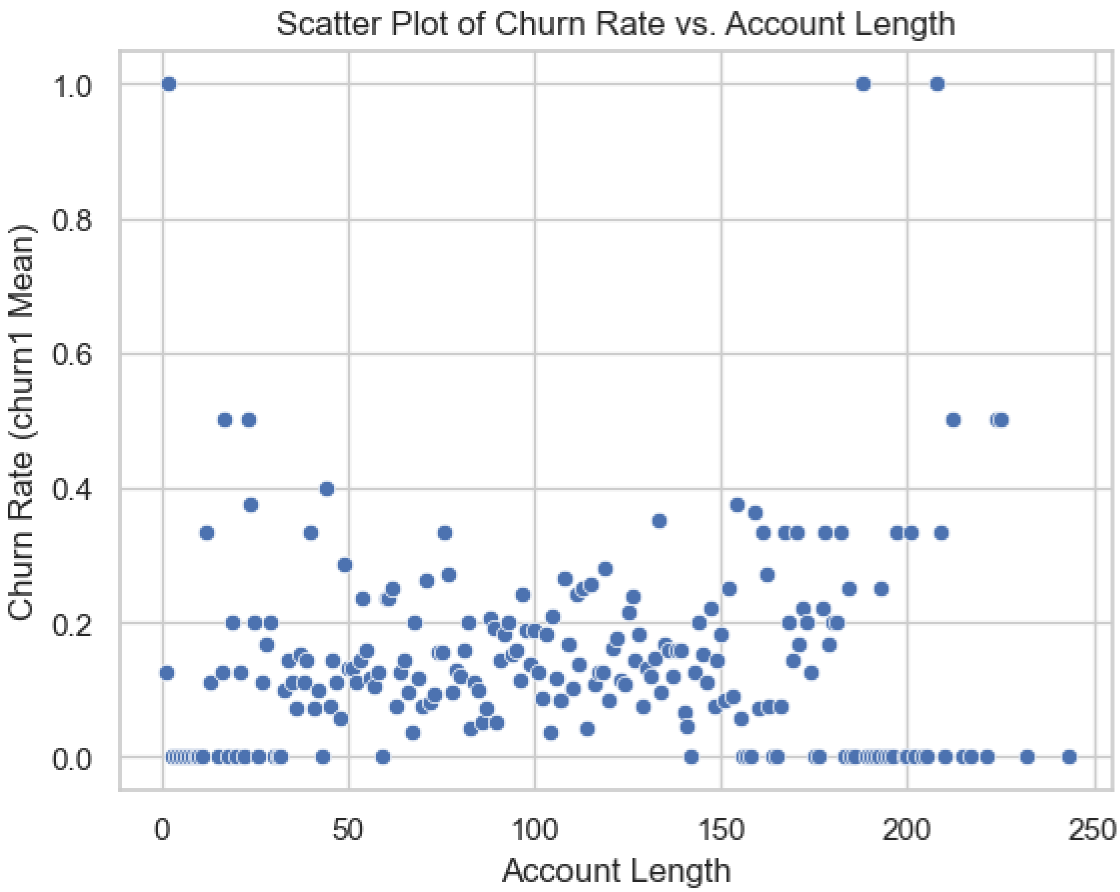
Out[26]:

	state	accountlength	areacode	internationalplan	voicemailplan	numbervmailmessages	totalda
0	18	117	408	0	0	0	
1	15	65	415	0	0	0	
2	34	161	415	0	0	0	
3	40	111	415	0	0	0	
4	11	49	510	0	0	0	
...	...	...	...	...	...	...	
3328	40	79	415	0	0	0	
3329	3	192	415	0	1	36	
3330	49	68	415	0	0	0	
3331	39	28	510	0	0	0	
3332	42	74	415	0	1	25	

3333 rows × 20 columns

In [17]:
```python
# Plots the churn rate against account length to visualize any patterns.

sns.set(style="whitegrid")

data = t.groupby(["accountlength"], as_index=False)["churn1"].mean()


sns.scatterplot(data=data, x="accountlength", y="churn1", marker="o")
```

```
plt.xlabel("Account Length")
plt.ylabel("Churn Rate (churn1 Mean)")
plt.title("Scatter Plot of Churn Rate vs. Account Length")

plt.show()
```



In [18]:
```
# Prepares the feature matrix (X) and target vector (y), then splits them into trainin

X = t.drop(columns=['churn1'])
y = t['churn1']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [19]:
```
# Prepares the feature matrix (X) and target vector (y), then splits them into trainin

clf = RandomForestClassifier(random_state=3125)
clf.fit(X_train, y_train)
```

Out[19]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=3125)
```

In [20]:
```
# Uses the trained classifier to make predictions on the test set.
y_pred = clf.predict(X_test)

# Prints the accuracy of the model on the test data.
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_pred)}")
```

```
Accuracy: 0.9535232383808095
Confusion Matrix:
[[579   3]
 [ 28  57]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       582
           1       0.95      0.67      0.79        85

    accuracy                           0.95       667
   macro avg       0.95      0.83      0.88       667
weighted avg       0.95      0.95      0.95       667
```

In [21]:
```python
pd.DataFrame(confusion_matrix(y_test, y_pred),
             columns=["Predicted negative", 'Predicted positive'],
             index=['Actual negative', 'Actual positive'])
```

Out[21]:

	Predicted negative	Predicted positive
Actual negative	579	3
Actual positive	28	57

In [22]:
```python
# Retrieves the feature importances from the trained Random Forest model.
feature_importances = clf.feature_importances_
feature_names = X.columns  # Assuming column names in 'X' match 'newdata'
for name, importance in sorted(zip(feature_names, feature_importances), key=lambda x:
    print(f"Feature: {name}, Importance: {importance}")
```

```
Feature: totaldayminutes, Importance: 0.14275874523798063
Feature: totaldaycharge, Importance: 0.1275998431608136
Feature: customerservicecalls, Importance: 0.11880570765767055
Feature: internationalplan, Importance: 0.07716178992782396
Feature: totaleveningcharge, Importance: 0.06646466204504638
Feature: totaleveningminutes, Importance: 0.06411765609961638
Feature: totalintercalls, Importance: 0.05043259163202092
Feature: totalinterminutes, Importance: 0.04044416167128564
Feature: totalintercharge, Importance: 0.0398620534520861
Feature: totalnightcharge, Importance: 0.03622609074798718
Feature: totalnightminutes, Importance: 0.035343561686735386
Feature: accountlength, Importance: 0.030777547732424308
Feature: totaldaycalls, Importance: 0.030043845560503785
Feature: numbervmailmessages, Importance: 0.02984471726447462
Feature: totalnightcalls, Importance: 0.02847440625154134
Feature: totaleveningcalls, Importance: 0.028350866655604833
Feature: state, Importance: 0.026315816824269974
Feature: voicemailplan, Importance: 0.018983256187404252
Feature: areacode, Importance: 0.007992680204710098
```

In [23]:
```python
# Prepares a new dataset 'newdata' for making churn predictions.
newdata = pd.DataFrame({
    "state" : [18, 14],
    "accountlength" : [117, 65],
    "areacode" : [408, 415],
    "internationalplan" : [0, 0],
    "voicemailplan" : [0, 0],
    "numbervmailmessages" : [0,0],
    "totaldayminutes" : [184.5, 129.1],
```

```
    "totaldaycalls"  : [97, 37],
    "totaldaycharge" : [31.37, 21.95],
    "totaleveningminutes" : [351.6, 228.5],
    "totaleveningcalls" : [80, 83],
    "totaleveningcharge" : [29.89, 19.42],
    "totalnightminutes" : [215.8, 208.8],
    "totalnightcalls" : [90, 111],
    "totalnightcharge" : [9.71, 9.4],
    "totalinterminutes" : [8.7, 12.7],
    "totalintercalls" : [4, 6],
    "totalintercharge" : [2.35, 3.43],
    "customerservicecalls" : [1, 4]
})
```

In [25]:

```
# Adds a column to 'newdata' with predictions of customer churn using the trained mode
newdata = newdata[feature_names]  # Ensure the feature order matches the trained model

newdata["predictcustomerchurn"] = clf.predict(newdata)

# Prints the new data with the predicted churn.
newdata
```

Out[25]:

	state	accountlength	areacode	internationalplan	voicemailplan	numbervmailmessages	totaldaym
0	18	117	408	0	0	0	
1	14	65	415	0	0	0	