INSAID
INTERNATIONAL SCHOOL OF AI AND DATA SCIENCE

Table of Contents

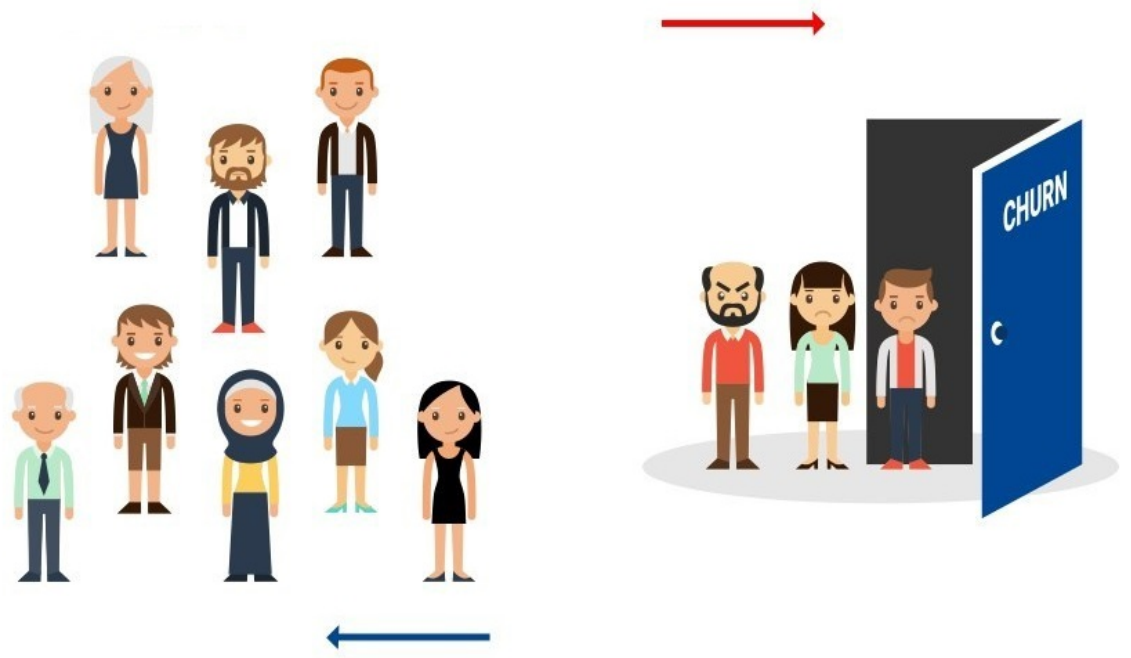
---

1. Introduction

- **Customer churn** (also known as **customer attrition**) refers to when a customer **ceases his or her relationship with a company**.

- It is an **important key business metric** because the **cost** of **retaining** an **existing customer** is far **less expensive than acquiring a new one.**

- [70%](#) of **companies say** it's **cheaper** to **retain** a **customer than acquire one** while the **cost** of **acquiring** a **new customer** can be as much as **seven times more expensive**.
- **Example:** If you start your quarter with 400 customers and end with 380, your churn rate is 5% because you lost 5% of your customers.
- **Businesses** typically **treat** a **customer** as **churned once** a **particular amount** of **time** has **elapsed** since the customer's last interaction.
- **Companies** usually **make** a **distinction** between **voluntary churn** and **involuntary churn**.
  - **Voluntary Churn (Controllable):** It **occurs** due to a **decision** by the **customer** to switch to another company or service provider.
  - **Involuntary Churn (Uncontrollable):** It **occurs** due to **circumstances** such as a **customer's relocation** to a long-term care facility, **death**, or the relocation to a distant location.



Prevention of Customer Churn

- **Lean into your best customers:** Identify pool of customers that are most likely to cancel, and refocus your efforts to keep them on board.
- **Be proactive with communication:** Reach out to your customers before they need you and get the most out of your product or service.
- **Define a roadmap for your new customers:** A new product or service can be overwhelming for a customer. To ease the transition, it's helpful to set up a new customer onboarding process or roadmap to guide new customers through your product or service's features, functionality, and process.
- **Offer incentives:** Give customers a reason to stick around by offering them something special -- a promo, discount, loyalty program, etc.

- **Ask for feedback often:** Getting to the root of the specific issues plaguing your business requires you to take the time to collect feedback early and often.
- **Analyze churn when it happens:** You should be using data before customers churn in order to build strategies to proactively prevent it.First, start with analysis.
  - When are customers most frequently churning?
  - Is it 30, 60, or 90 days after they first start using your product or service?
  - Does churn happen if customers go a specific number of days without using the product or service?
- **Stay competitive:** Market conditions are constantly changing. Businesses focused on what's next -- trends, technology, and product advancements -- position themselves in a good spot in terms of avoiding disruption or "the next big thing."

Churn Scenario in India

- **Absolute Data claims** that in India **6%** is the **average monthly churn rate** for Indian telecom customers.
- According to **The Indian Express**, The telecom **subscriber base** in the country **reached 1,198.89 million** in **April, 2020**.
- But the **growth continued** its **downward trend** in **line** with the **slower pace** of **new customer** additions by **Reliance Jio**.
- **Jio** was **followed by Bharti Airtel** which **added 2.85 million new mobile subscribers, BSNL with 0.81 million, Vodafone 0.75 million and Idea 0.68 million**.
- **Tata Teleservices** was the **biggest loser** of **mobile subscribers** in April. The **net subscriber loss** of the company was **1.46 million**.
- **Reliance Communications lost 1.32 million subscribers, Aircel 0.33 million, Sistema Shyam 0.27 million and MTNL 2,137 subscribers**.
- **State-run BSNL lost 1.86 million customers followed by MTNL which lost 7,888 customers**.

---

2. Problem Statement

- **Companies** have been **experiencing** a **high churn rate** more than ever **due to** the **rapid change** in the **development** of the **technology**.
- **Companies** are **under more pressure** to **generate revenue** from other **areas** or **gain new clients**.
- **For example, 4G technology** has made **great impact** on the **digital life**.

- **Companies** have **either drowned or shaken hands** to survive in such tough competition.
- **For example: Vodafone and IDEA** have **made a pact** with each other **to provide 4G services**.

Scenario:

- **Aster Rhino**, a **USA** based company that **provides telecommunication services** to the **customers**.
- They have been **providing 3G services since 2008** and **started providing 4G services after its launch**.
- Due to **boom in telecomm industry** with **4G technology**, it has **become pain in the neck** fo the **company** to **retain** their **customers**.
- They are **in the middle** to **set more cell sites of 4G network** to **improve** their **4G services**.
- It is **plausible** for **customer** to **choose 4G services over 3G services** due to **benifits of cost, speed, latency etc.**
- Till now they have been **using manual traditional ways** which **now** has **become** a **problem** to **handle** due to **work complication**.
- They have **detailed history** of their **customers** and are **looking** for an **automated solution** to **identify** the **likeliness** of **customer churning** from **using** their **services**.
- In turn, **they decided** to **find** more **optimistic way** and **hired a team of data scientists** to **solve this problem. Consider you are one of them...**

Target Feature	Potential Values
churn	False: Did not churn
	True: Churned

3. Installing and importing libraries

3.1 Installing Libraries

```
!pip install -q datascience                    # Package that is required by panda
!pip install -q pandas-profiling               # Toolbox for Generating Statistics
!pip install -q yellowbrick                    # Toolbox for Measuring Machine Per
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 20.4 MB/s eta 0:00:0
```

3.2 Upgrading Libraries

Note:

- After upgrading, you need to restart the runtime.
- Make sure not to execute the cell above (3.1) and below (3.2) again after restarting the runtime.

```
!pip install -q --upgrade pandas-profiling
!pip install -q --upgrade yellowbrick
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.4/324.4 KB 13.2 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 344.5/344.5 KB 25.9 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 90.5 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.7/102.7 KB 11.3 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 679.5/679.5 KB 37.6 MB/s eta 0:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 92.3 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 KB 29.2 MB/s eta 0:00
  Building wheel for htmlmin (setup.py) ... done
```

3.3 Importing Libraries

```
# For Panel Data Analysis
import pandas as pd
from pandas_profiling import ProfileReport
import pandas.util.testing as tm
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_rows', None)
pd.set_option('mode.chained_assignment', None)

# For Numerical Python
import numpy as np

# For Data Visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from random import randint

# For Feature Selection
import seaborn as sns
from sklearn.feature_selection import SelectFromModel

# For Custom Transformers
from sklearn.base import BaseEstimator, TransformerMixin

# For Imputation
from sklearn.impute import KNNImputer

# For Feature Importances
from yellowbrick.model_selection import FeatureImportances

# For metrics evaluation
from sklearn.metrics import classification_report, plot_confusion_matrix, preci

# For Data Modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# To handle class imbalance problem
from imblearn.over_sampling import SMOTE

# To Disable Warnings
import warnings
warnings.filterwarnings(action = 'ignore')
```

```
<ipython-input-4-81ec32564e14>:3: DeprecationWarning: `import pandas_profilin
  from pandas_profiling import ProfileReport
<ipython-input-4-81ec32564e14>:4: FutureWarning: pandas.util.testing is depre
  import pandas.util.testing as tm
```

4. Data Acquisition and Description

- This dataset is based on the details of customers' account accumulated by Aster Rhino and is accessible **here**.

Records	Features	Dataset Size
3333	21	303 KB

Id	Features	Description
01	**State**	The state of the customer. Contains: [KS, OH, NJ, OK, AL, MA, MO, LA, WV, IN, RI, IA, MT, NY, ID, VT, V AZ, SC, NE, WY, HI, IL, NH, GA, AK, MD, AR, WI, OR, MI, DE, UT, CA, MN, SD, NC, WA, NM, NV, DC, KY, N
02	**AccountLength**	Number of days since the customer started using services. Range: [1, 243]
03	**AreaCode**	The area code of the customer. Contains: [415, 408, 510]
04	**PhoneNumber**	A unique phone number of the customer. Range: [3271058, 4229964]
05	**InternationalPlan?**	Whether the account has an active international plan or not. Contains: [No, Yes]
06	**VoiceMailPlan?**	Whether the account has an active voice mail plan or not. Contains: [No, Yes]
07	**NumEmailMessages**	Total number of voice mail messages consumed. Range: [0, 51]
08	**TotalMorMin**	Total minutes consumed in the morning. Range: [0.0, 350.8]
09	**TotalMorCalls**	Total number of calls consumed in the morning. Range: [0, 165]
10	**TotalMorCharge**	Total charges during morning (in cent). Range: [0.0, 59.64]
11	**TotalEveMin**	Total minutes consumed in the evening. Range: [0.0, 363.7]
12	**TotalEveCalls**	Total number of calls consumed in the evening. Range: [0, 170]
13	**TotalEveCharge**	Total charges during evening (in cent). Range: [0.0, 30.91]
14	**TotalNightMin**	Total number of minutes consumed in the night. Range: [23.2, 395.0]
15	**TotalNightCalls**	Total number of calls consumed in the night. Range: [33, 175]
16	**TotalNightCharge**	Total charges during night (in cent). Range: [1.04, 17.77]
17	**TotalIntMinutes**	Total international minutes consumed if subscribed services. Range: [0.0, 20.0]
18	**TotalIntCalls**	Total international calls consumed. Range: [0, 20]
19	**TotalIntCharge**	Total international charges (in cent). Range: [0.0, 5.4]
20	**CustomerServiceCalls**	Total number of customer service calls consumed by customer. Range: [0, 9]
21	**Churn?**	Whether the customer has churned or not. Contains: [False, True]

```
data = pd.read_csv('https://storage.googleapis.com/telecom-analytics/TeleChurnDat
print('Data Shape:', data.shape)
data.head()
```

```
Data Shape: (3833, 21)
```

	State	AccountLength	AreaCode	PhoneNumber	InternationalPlan?	VoiceMailPl
0	KS	128	415	3824657	No	
1	OH	107	415	3717191	No	
2	NJ	137	415	3581921	No	
3	OH	84	408	3759999	Yes	
4	OK	75	415	3306626	Yes	

4.1 Data Description

- In this section we will get **information about the data** and see some observations.

```
print('Described Column Length:', len(data.describe().columns))
data.describe()
```

```
Described Column Length: 17
```

	AccountLength	AreaCode	PhoneNumber	NumEmailMessages	TotalMorMin	T
count	3833.000000	3833.000000	3.833000e+03	3833.000000	3803.000000	
mean	100.697626	437.447691	3.746407e+06	8.227759	180.078517	
std	39.872358	42.506606	2.758814e+05	13.724437	54.664611	
min	1.000000	408.000000	3.271058e+06	0.000000	0.000000	
25%	73.000000	415.000000	3.506473e+06	0.000000	144.000000	
50%	100.000000	415.000000	3.749107e+06	0.000000	179.900000	
75%	127.000000	510.000000	3.988385e+06	20.000000	216.650000	
max	243.000000	510.000000	4.229964e+06	51.000000	350.800000	

Observations:

- On **average customers perform 8** number of **email messages**.
- **50%** of **customers don't do any email messages** while **75%** of **customers** do **20 number of messages**.
- On **average customers talk 180 minutes** of duration in the **morning**.

- **25%** of **customers talk 144 minutes** of duration in the **morning** while **50% and 75%** of **customers talk 179 minutes and 216 minutes** in the **morning**.
- On **average customers** like to **perform 100 calls**.
- **25%** of **customers** like to **perform 87 number of calls** while **50% and 75%** of **customers** like to **perform 101 and 114 number of calls**.
- On **average** it take **30 cents** for **morning services**.
- **25%** of **customers** have been **charged** with **24 cents** while **50% and 75%** of **customers** have been **charged** with **30 cents and 36 cents** in the **morning**.
- **Similarly users can understand the information for rest of the features.**

4.2 Data Information

- In this section we will see the **information about the types of features**.

```
data.info(verbose = True, memory_usage = 'deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3833 entries, 0 to 3832
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   State                3833 non-null   object
 1   AccountLength        3833 non-null   int64
 2   AreaCode             3833 non-null   int64
 3   PhoneNumber          3833 non-null   int64
 4   InternationalPlan?   3833 non-null   object
 5   VoiceMailPlan?       3833 non-null   object
 6   NumEmailMessages     3833 non-null   int64
 7   TotalMorMin          3803 non-null   float64
 8   TotalMorCalls        3833 non-null   int64
 9   TotalMorCharge       3833 non-null   float64
 10  TotalEveMin          3822 non-null   float64
 11  TotalEveCalls        3833 non-null   int64
 12  TotalEveCharge       3833 non-null   float64
 13  TotalNightMin        3815 non-null   float64
 14  TotalNightCalls      3833 non-null   int64
 15  TotalNightCharge     3833 non-null   float64
 16  TotalIntMinutes      3833 non-null   float64
 17  TotalIntCalls        3833 non-null   int64
 18  TotalIntCharge       3833 non-null   float64
 19  CustomerServiceCalls 3833 non-null   int64
 20  Churn?               3833 non-null   bool
dtypes: bool(1), float64(8), int64(9), object(3)
memory usage: 1.1 MB
```

Observations:

- We can **observe missing values** in the data.

- **Apart** from **missing information** all the **features** have **correct** type.
- Let's **explore further**.

4.3 Pre Profiling Report

- For quick analysis pandas profiling is very handy.
- Generates profile reports from a pandas DataFrame.
- For each column statistics are presented in an interactive HTML report.

```
`#profile = ProfileReport(df = data)
#profile.to_file(output_file = 'Pre Profiling Report.html')
#print('Accomplished!')
```

Summarize dataset: 100% 287/287 [01:08<00:00, 5.21it/s, Completed]

Generate report structure: 100% 1/1 [00:10<00:00, 10.00s/it]

Render HTML: 100% 1/1 [00:10<00:00, 10.55s/it]

```
#from google.colab import files                  # Use only if you are using Goo
#files.download('Pre Profiling Report.html')      # Use only if you are using Goo
```

Observations:

- **Report show** that there are **total 21 features** out of which **16 are numerical, 3 are boolea and 2 are categorical.**
- Around **59 (0.1%) cells** contains **missing information**.
- Data contains **500 (13%) duplicate rows**.
- **TotalMorCharge** is **highly correlated** to **TotalMorMin**.
- **TotalEveCharge** is **highly correlated** to **TotalEveMin**.
- **TotalNightCharge** is **highly correlated** to **TotalNightMin**.
- **TotalIntCharge** is **highly correlated** to **TotalIntMin**.

---

5. Data Pre-Processing

5.1 Identification & Handling of Missing Data

- In this section we will **analyze** and **identify missing information** such as **null data** and **zero data**.

Before Handling Missing Information

```
missing_frame = pd.DataFrame(index = data.columns.values)
missing_frame['Null Frequency'] = data.isnull().sum().values
nullpercent = data.isnull().sum().values/data.shape[0]
missing_frame['Missing Null %age'] = np.round(nullpercent, decimals = 4) * 100
missing_frame.transpose()
```

	State	AccountLength	AreaCode	PhoneNumber	InternationalPlan?	Voi
Null Frequency	0.0	0.0	0.0	0.0	0.0	
Missing Null %age	0.0	0.0	0.0	0.0	0.0	

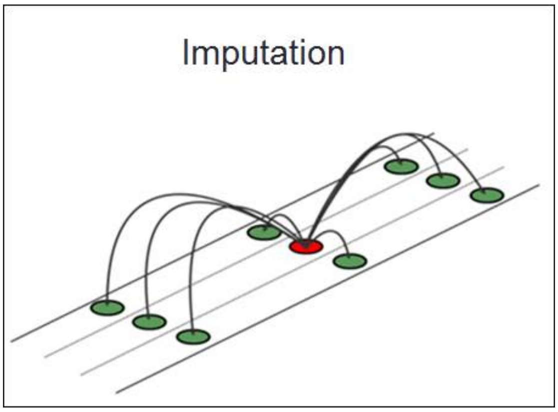
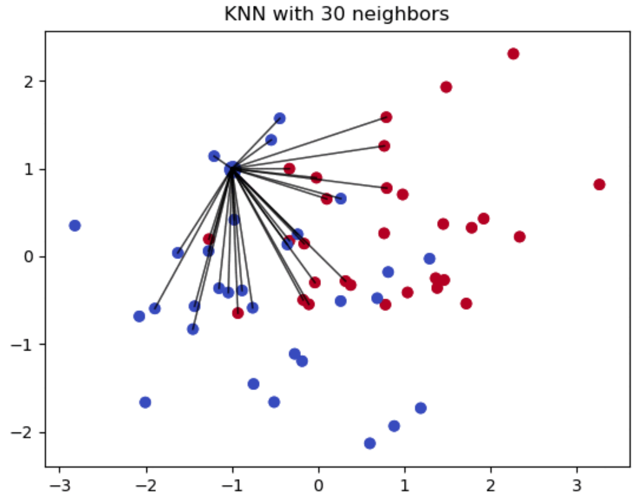
Observation:

- Feature:
  - Problem → Solution {Reason}
- **TotalMorMin:**
  - Null Data → KNN Imputation {Null proportion is small.}
- **TotalEveMin:**
  - Null Data → KNN Imputation {Null proportion is small.}
- **TotalNightMin:**
  - Null Data → KNN Imputation {Null proportion is small.}

KNN Imputer:

- This is an **effective approach** where data **imputation** is **done** using **prediction** by a model.
- A **model** is **created** for **each feature** that has **missing values**, taking as input values of all other input features.

- A **new sample** is **imputed** by **finding** the **samples** in the training set **“closest”** to it and **averages** these **nearby points** to **fill in the value**.

Sample Example	Complete Working
	

Performing Operations

```
knn_imputer = KNNImputer()
raw_num = knn_imputer.fit_transform(data[['TotalMorMin', 'TotalEveMin', 'TotalNig
raw_frame = pd.DataFrame(data = raw_num, columns = ['TotalMorMin', 'TotalEveMin',

data['TotalMorMin'] = raw_frame['TotalMorMin']
data['TotalEveMin'] = raw_frame['TotalEveMin']
data['TotalNightMin'] = raw_frame['TotalNightMin']
```

5.2 Identification & Handling of Redundant Data

- In this section **we will identify redundant rows and columns** in our data if present.
- For **handling duplicate features** we have **created a custom function**.
- This **custom function** will help us to **identify duplicacy in features** with **different name but similar values**:

```
def duplicate_cols(dataframe):
  ls1 = []
  ls2 = []

  columns = dataframe.columns.values
  for i in range(0, len(columns)):
    for j in range(i+1, len(columns)):
      if (np.where(dataframe[columns[i]] == dataframe[columns[j]], True, False).a
        ls1.append(columns[i])
        ls2.append(columns[j])

  if ((len(ls1) == 0) & (len(ls2) == 0)):
    return None
  else:
    duplicate_frame = pd.DataFrame()
    duplicate_frame['Feature 1'] = ls1
    duplicate_frame['Feature 2'] = ls2
    return duplicate_frame


print('Contains Redundant Records?:', data.duplicated().any())
print('Duplicate Count:', data.duplicated().sum())
print('-----------------------------------------------------------------------')
print('Contains Redundant Features?:', duplicate_cols(data))
```

```
Contains Redundant Records?: True
Duplicate Count: 500
-----------------------------------------------------------------------
Contains Redundant Features?: None
```

Performing Operations

```
before_shape = data.shape
print('Data Shape [Before]:', before_shape)

data.drop_duplicates(inplace = True)

after_shape = data.shape
print('Data Shape [After]:', after_shape)

drop_nums = before_shape[0] - after_shape[0]
drop_percent = np.round(drop_nums / before_shape[0], decimals = 3) * 100

print('Drop Ratio:', drop_percent, '%')
```

```
Data Shape [Before]: (3833, 21)
Data Shape [After]: (3333, 21)
Drop Ratio: 13.0 %
```

After Handling Duplicate Data

```
print('Contains Redundant Records?:', data.duplicated().any())
print('Duplicate Count:', data.duplicated().sum())
print('----------------------------------------------------------------------')
print('Contains Redundant Features?:', duplicate_cols(data))
```

```
Contains Redundant Records?: False
Duplicate Count: 0
----------------------------------------------------------------------
Contains Redundant Features?: None
```

5.3 Post Data Profiling

```
profile = ProfileReport(df = data)
profile.to_file(output_file = 'Post Profiling Report.html')
print('Accomplished!')
```

Summarize dataset: 100%	35/35 [00:45<00:00, 1.30s/it, Completed]
Generate report structure: 100%	1/1 [00:07<00:00, 7.39s/it]
Render HTML: 100%	1/1 [00:08<00:00, 8.05s/it]
Export report to file: 100%	1/1 [00:00<00:00, 12.71it/s]

```
Accomplished!
```

```
from google.colab import files                    # Use only if you are using Goog
files.download('Pre Profiling Report.html')       # Use only if you are using Goog
```

Observation:

- Report shows that there's **no major change apart** from the **removal** of **missing information** and **redundant data**.

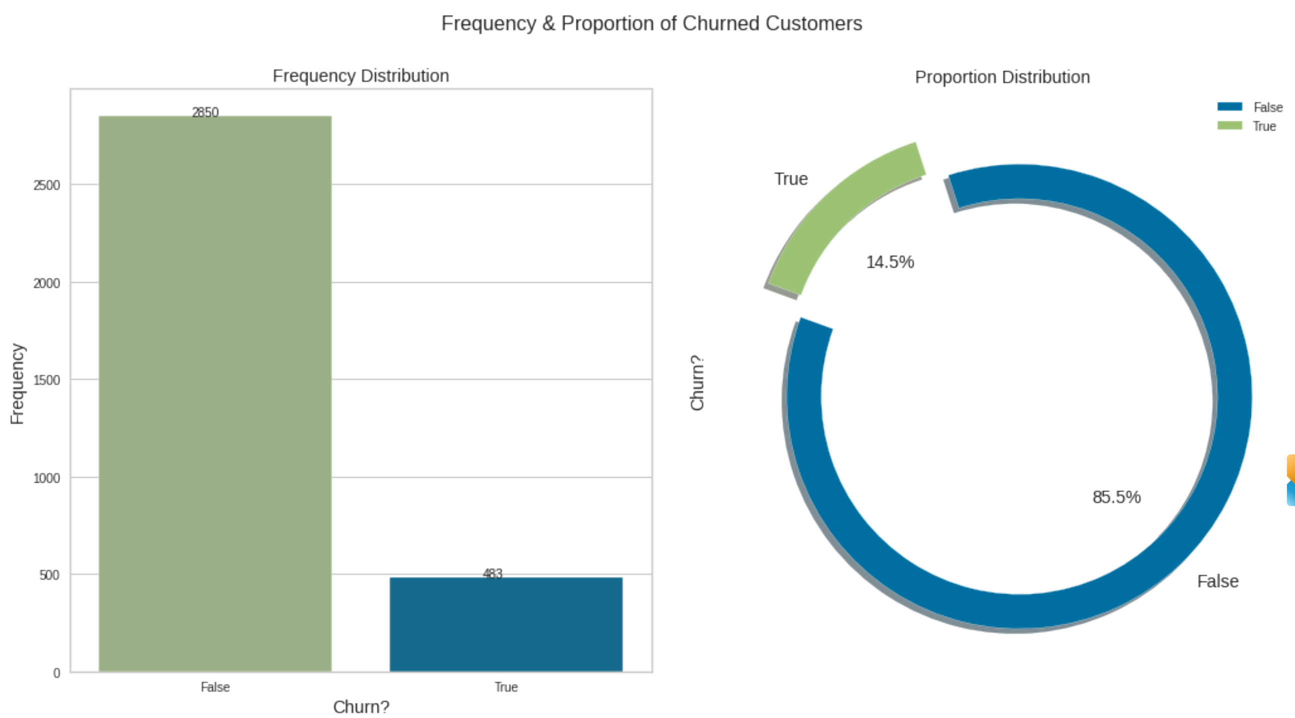
---

6. Exploratory Data Analysis

Question 1: What is the frequency and proportion of customer churn?

```
fig = plt.figure(figsize = [15, 8])
plt.subplot(1, 2, 1)
ax = sns.countplot(x = 'Churn?', data = data, palette = ['#9FBA81', '#0272A2'])
for p in ax.patches:
  percentage = '{}'.format(p.get_height())
  x = p.get_x() + p.get_width() / 2.5
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))
plt.xlabel(xlabel = 'Churn?', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution', size = 14)

plt.subplot(1, 2, 2)
space = np.ones(2)/10
data['Churn?'].value_counts().plot(kind = 'pie', explode = space, fontsize = 14,
                                       shadow = True, startangle = 160, figsize =
plt.ylabel(ylabel = 'Churn?', size = 14)
plt.title(label = 'Proportion Distribution', size = 14)
plt.tight_layout(pad = 3.0)
plt.suptitle(t = 'Frequency & Proportion of Churned Customers', y = 1.02, size =
plt.show()
```



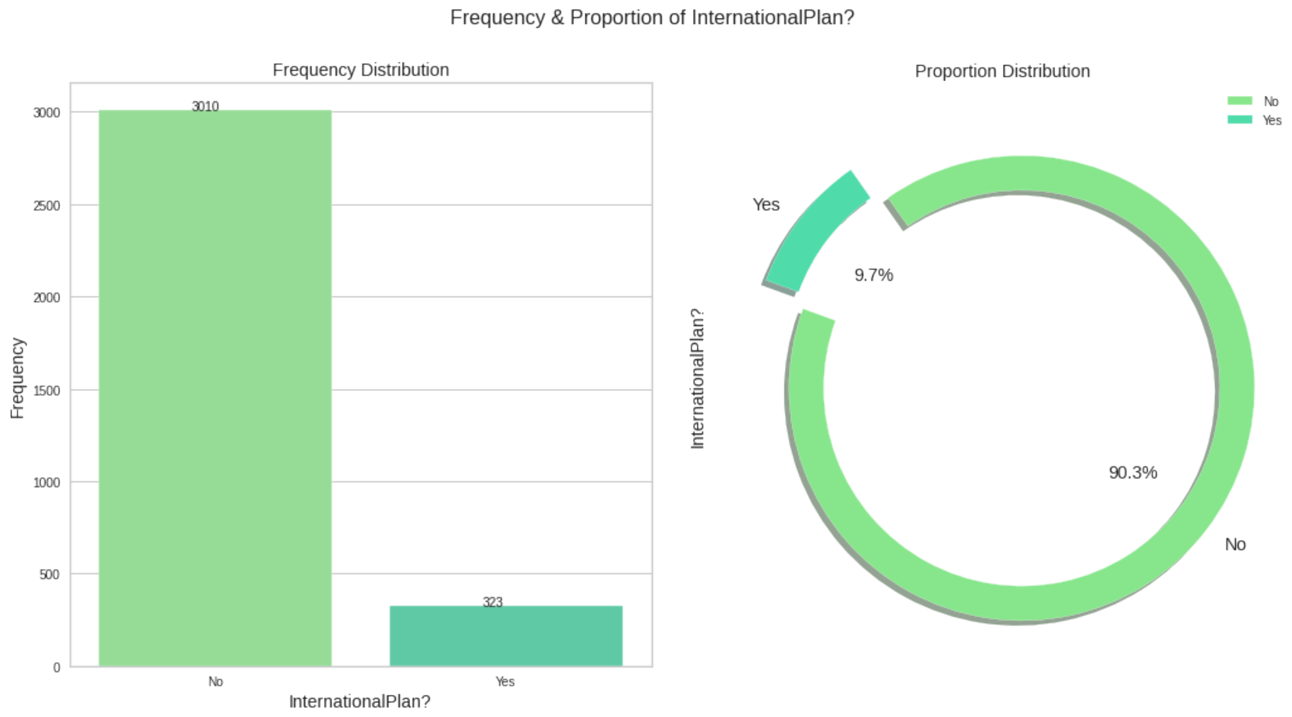
Observations:

- The **churn percentage** is **14.5%**.
- Approximately **483 people churned** out of **3333 people**.
- One thing is clear that the **performance evalution metric** is **not accuracy** because we can **observe class imbalance**.

Question 2: What is the frequency and proportion of InternationalPlan?

```
fig = plt.figure(figsize = [15, 8])
plt.subplot(1, 2, 1)
ax = sns.countplot(x = 'InternationalPlan?', data = data, palette = ['#8AEA8D', '
for p in ax.patches:
  percentage = '{}'.format(p.get_height())
  x = p.get_x() + p.get_width() / 2.5
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))
plt.xlabel(xlabel = 'InternationalPlan?', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution', size = 14)

plt.subplot(1, 2, 2)
space = np.ones(2)/10
data['InternationalPlan?'].value_counts().plot(kind = 'pie', explode = space, fon
                                       shadow = True, startangle = 160, figsize =
plt.ylabel(ylabel = 'InternationalPlan?', size = 14)
plt.title(label = 'Proportion Distribution', size = 14)
plt.tight_layout(pad = 3.0)
plt.suptitle(t = 'Frequency & Proportion of InternationalPlan?', y = 1.02, size =
plt.show()
```



Observation:

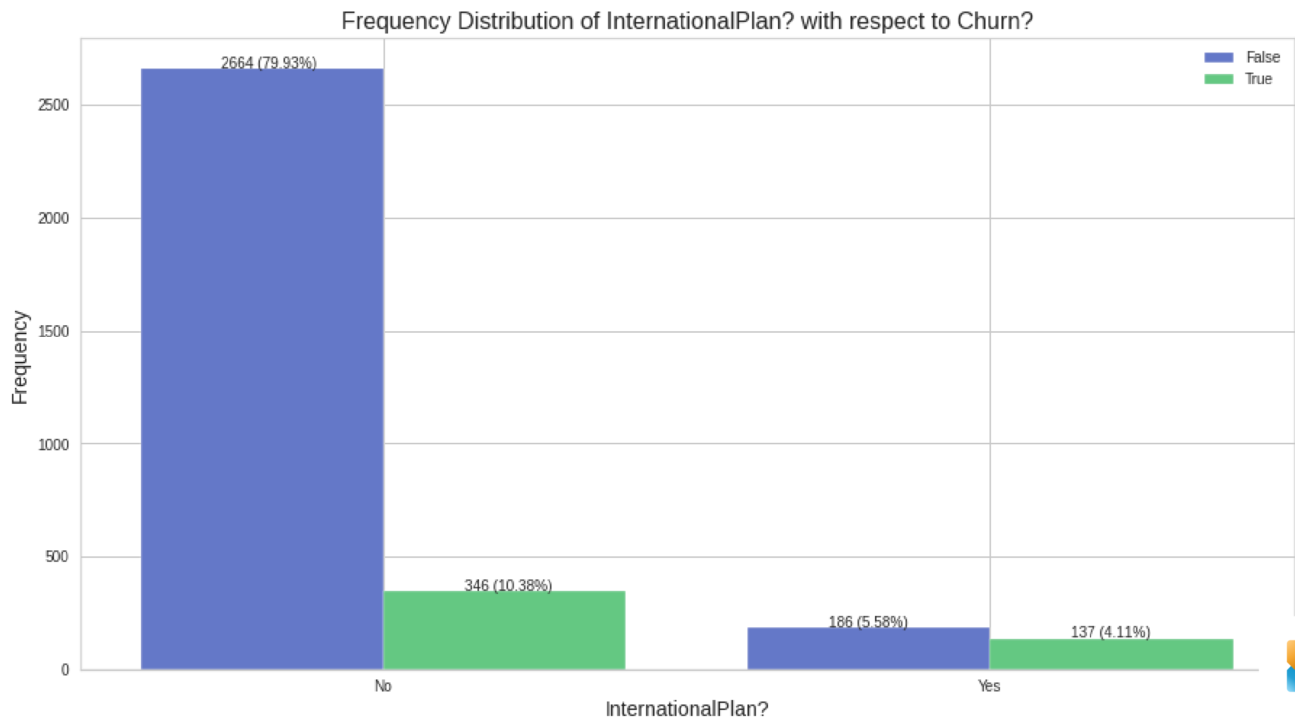
- **3010 (around 90%)** of **customers** are **not using international plan** while only **323 (around 10%)** are **using international plan**.

Question 3: What is the frequency distribution of InternationalPlan? with respect to Churn?

```
figure = plt.figure(figsize = [15, 8])
ax = sns.countplot(x = 'InternationalPlan?', hue = 'Churn?', data = data, palette

total = data.shape[0]
for p in ax.patches:
  percentage = '{}'.format(p.get_height()) + ' (' +'{:.2f}%'.format(100*p.get_hei
  x = p.get_x() + p.get_width() / 3
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))

plt.xlabel(xlabel = 'InternationalPlan?', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution of InternationalPlan? with respect to C
plt.legend(loc = 'upper right')
plt.grid(b = True)
plt.show()
```



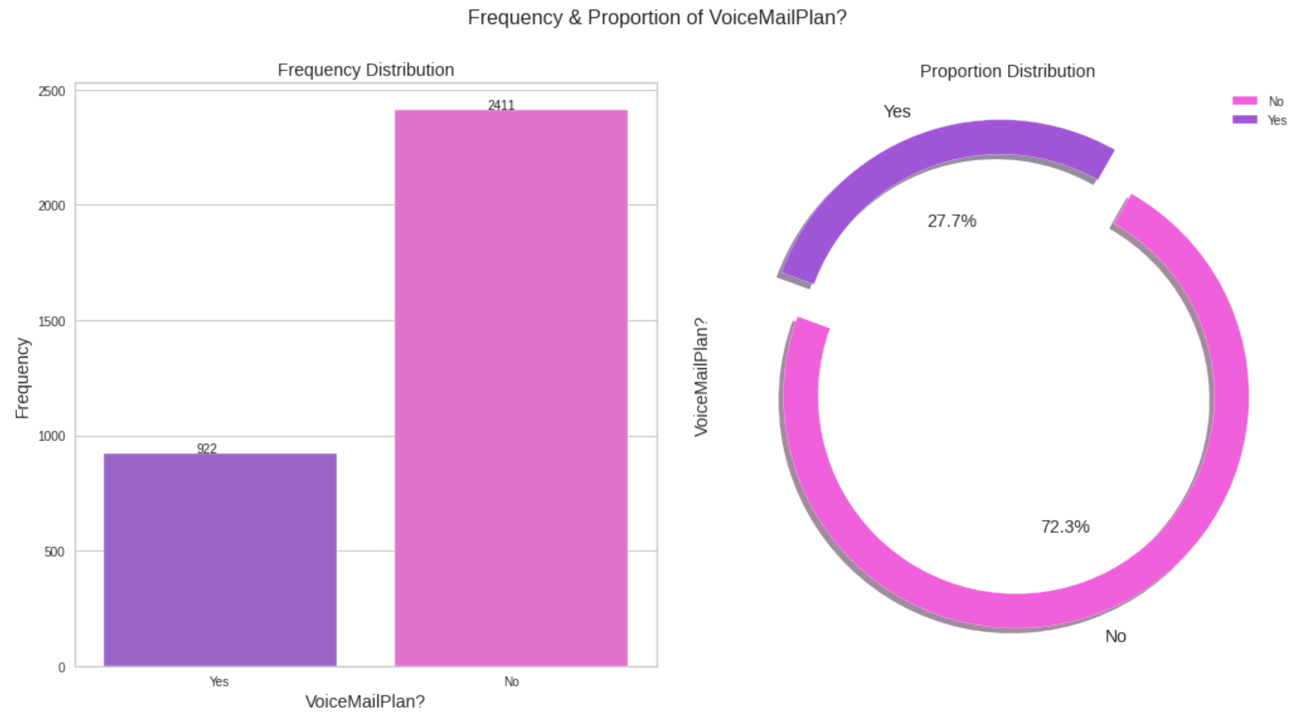
Observation:

- **Customers** who **don't have international plan**:
  - **Un-churned customers** are **approx 7.5X** than **churned customers**.
- **Customers** who **have international plan**:
  - **Un-churned customers** are **approx 1.4% more** than **churned customers**.

Question 4: What is the frequency and proportion of VoiceMailPlan?

```
fig = plt.figure(figsize = [15, 8])
plt.subplot(1, 2, 1)
ax = sns.countplot(x = 'VoiceMailPlan?', data = data, palette = ['#A056DB', '#F46
for p in ax.patches:
  percentage = '{}'.format(p.get_height())
  x = p.get_x() + p.get_width() / 2.5
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))
plt.xlabel(xlabel = 'VoiceMailPlan?', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution', size = 14)

plt.subplot(1, 2, 2)
space = np.ones(2)/10
data['VoiceMailPlan?'].value_counts().plot(kind = 'pie', explode = space, fontsiz
                                       shadow = True, startangle = 160, figsize =
plt.ylabel(ylabel = 'VoiceMailPlan?', size = 14)
plt.title(label = 'Proportion Distribution', size = 14)
plt.tight_layout(pad = 3.0)
plt.suptitle(t = 'Frequency & Proportion of VoiceMailPlan?', y = 1.02, size = 16)
plt.show()
```



Observation:

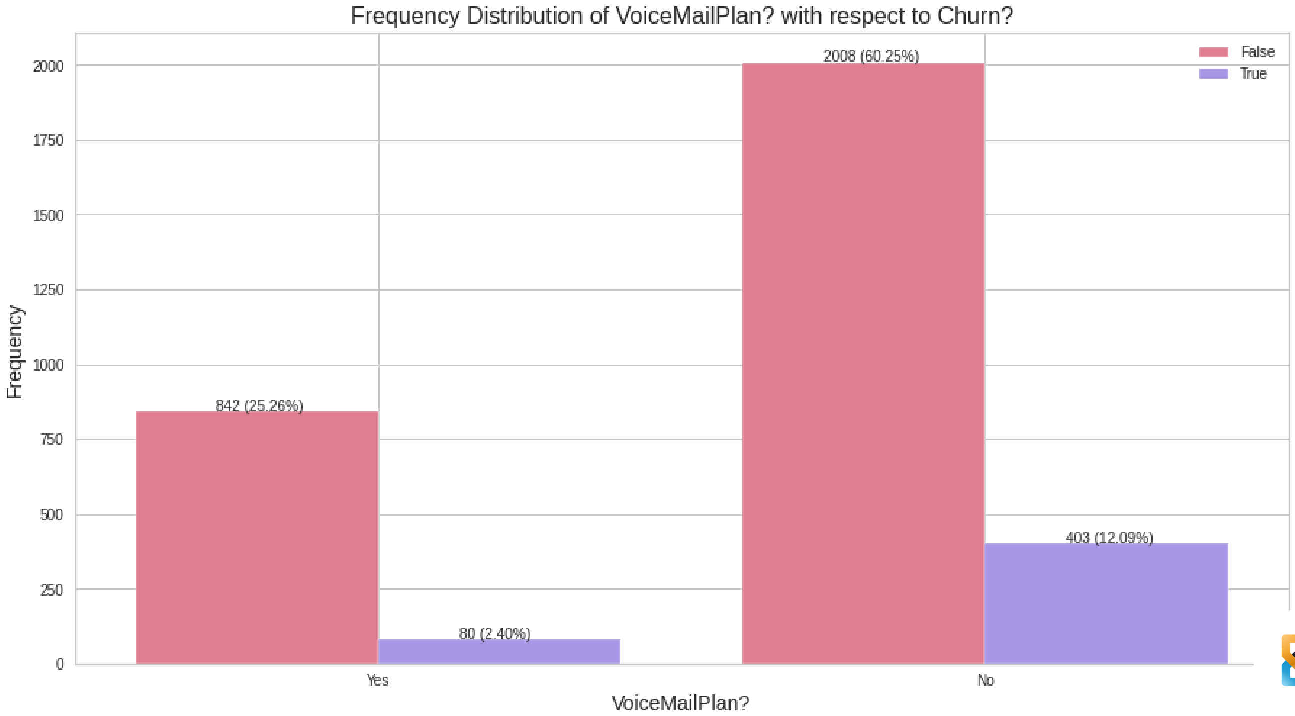
- **2411 (around 72%)** of **customers** are **not using voice mail plan** while only **922 (around 28%)** are **using voice mail plan**.

Question 5: What is the frequency distribution of VoiceMailPlan? with respect to Churn?

```
figure = plt.figure(figsize = [15, 8])
ax = sns.countplot(x = 'VoiceMailPlan?', hue = 'Churn?', data = data, palette = [

total = data.shape[0]
for p in ax.patches:
  percentage = '{}'.format(p.get_height()) + ' (' +'{:.2f}%'.format(100*p.get_hei
  x = p.get_x() + p.get_width() / 3
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))

plt.xlabel(xlabel = 'VoiceMailPlan?', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution of VoiceMailPlan? with respect to Churn
plt.legend(loc = 'upper right')
plt.grid(b = True)
plt.show()
```



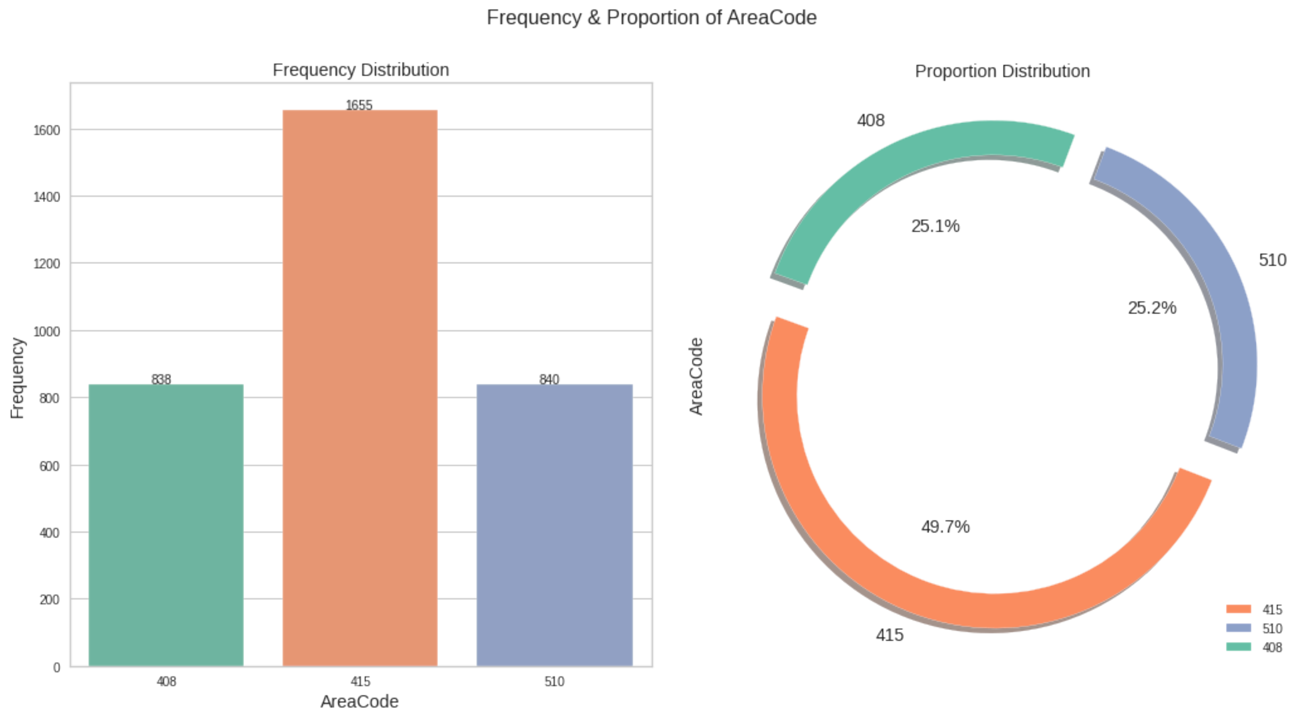
Observation:

- **Customers** who **have voicemail plan**:
  - **Un-churned customers** are **approx 10.5X** than **churned customers**.
- **Customers** who **don't have voicemail plan**:
  - **Un-churned customers** are **approx 5X more** than **churned customers**.

Question 6: What is the frequency and proportion of AreaCode?

```
fig = plt.figure(figsize = [15, 8])
plt.subplot(1, 2, 1)
ax = sns.countplot(x = 'AreaCode', data = data, palette = ['#66C2A5', '#FC8D62',
for p in ax.patches:
  percentage = '{}'.format(p.get_height())
  x = p.get_x() + p.get_width() / 2.5
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))
plt.xlabel(xlabel = 'AreaCode', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution', size = 14)

plt.subplot(1, 2, 2)
space = np.ones(3)/10
data['AreaCode'].value_counts().plot(kind = 'pie', explode = space, fontsize = 14
                                       shadow = True, startangle = 160, figsize =
plt.ylabel(ylabel = 'AreaCode', size = 14)
plt.title(label = 'Proportion Distribution', size = 14)
plt.tight_layout(pad = 3.0)
plt.suptitle(t = 'Frequency & Proportion of AreaCode', y = 1.02, size = 16)
plt.show()
```



Observation:

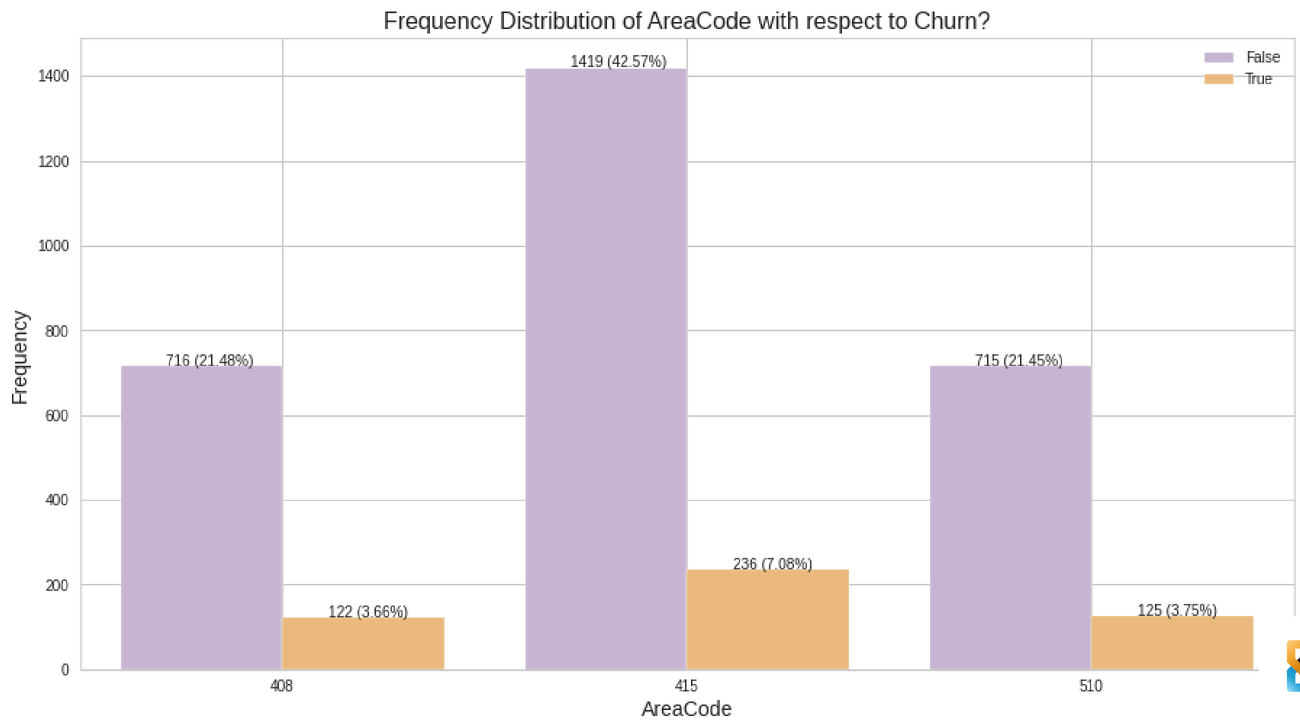
- **838 (25.1%)** of **customers belong** to **area code 408** (**San Jose**).
- **1655 (49.7%)** of **customers belongs** to **area code 415** (**San Francisco Bay area**).
- **840 (25.2%)** of **customers belongs** to **area code 510** (**Oakland**).
- **On observing above** three **points**, we can say that **majority** of **customers belongs** to **San Francisco Bay area**.

Question 7: What is the frequency distribution of AreaCode with respect to Churn?

```
figure = plt.figure(figsize = [15, 8])
ax = sns.countplot(x = 'AreaCode', hue = 'Churn?', data = data, palette = ['#CAB2

total = data.shape[0]
for p in ax.patches:
  percentage = '{}'.format(p.get_height()) + ' (' +'{:.2f}%'.format(100*p.get_hei
  x = p.get_x() + p.get_width() / 3.5
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))

plt.xlabel(xlabel = 'AreaCode', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution of AreaCode with respect to Churn?', si
plt.legend(loc = 'upper right')
plt.grid(b = True)
plt.show()
```



Observation:

- **Customers** who **belongs** to **area code 408**:
  - **Un-churned customers** are **approx 6X** than **churned customers**.
- **Customers** who **belongs** to **area code 415**:
  - **Un-churned customers** are **approx 6X more** than **churned customers**.
- **Customers** who **belongs** to **area code 510**:

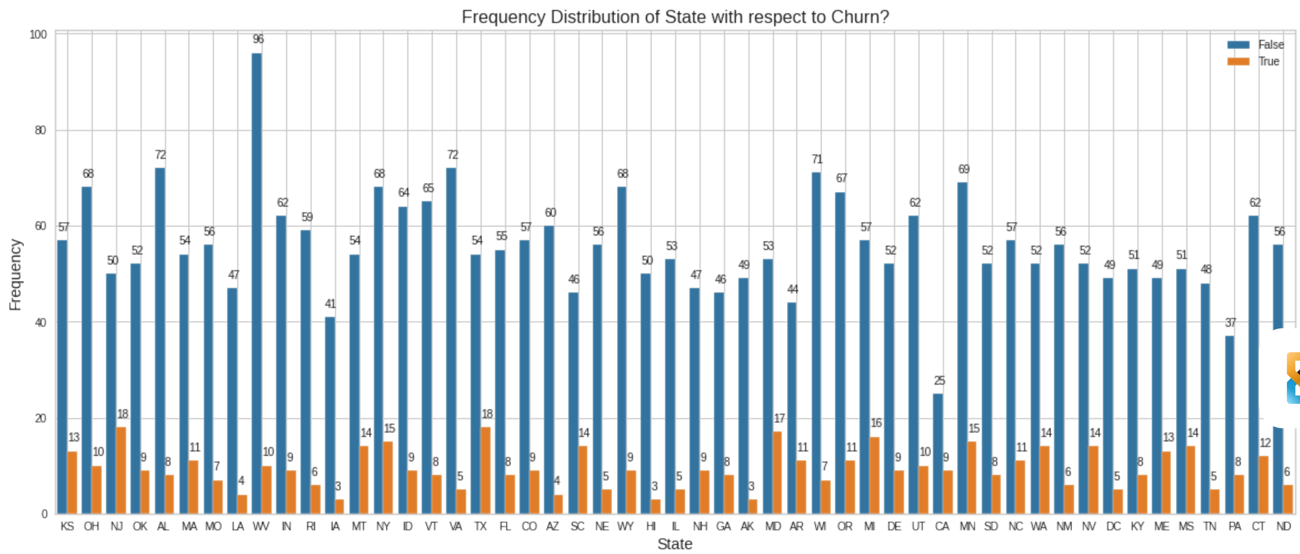
- **Un-churned customers** are **approx 5.5X more** than **churned customers**.

Question 8: What is the frequency distribution of State with respect to Churn?

```
figure = plt.figure(figsize = [20, 8])
ax = sns.countplot(x = 'State', hue = 'Churn?', data = data, palette = ['#1F77B4'

total = data.shape[0]
for p in ax.patches:
  percentage = '{}'.format(p.get_height())
  x = p.get_x() + p.get_width() / 51
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))

plt.xlabel(xlabel = 'State', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution of State with respect to Churn?', size
plt.legend(loc = 'upper right')
plt.grid(b = True)
plt.show()
```



Observation:

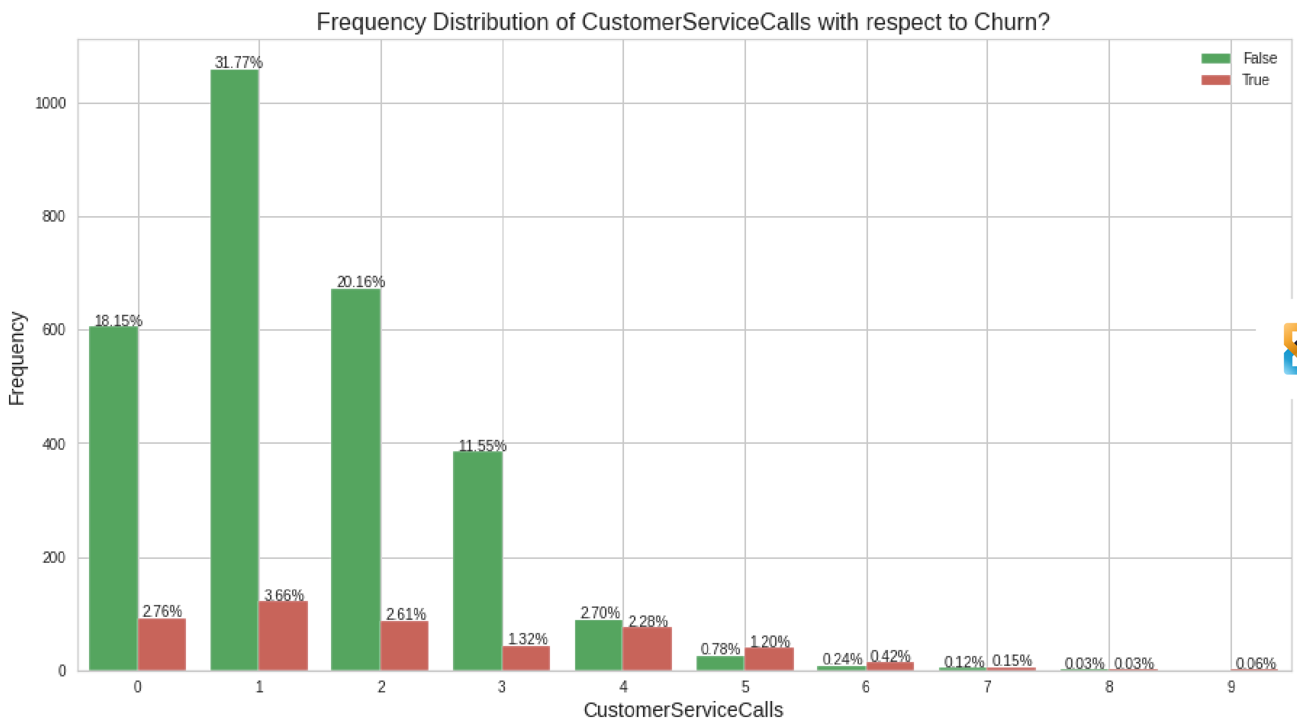
-

Question 9: What is the frequency distribution of CustomerServiceCalls with respect to Churn?

```
figure = plt.figure(figsize = [15, 8])
ax = sns.countplot(x = 'CustomerServiceCalls', hue = 'Churn?', data = data, palet

total = data.shape[0]
for p in ax.patches:
  percentage = '{:.2f}%'.format(100*p.get_height()/total)
  x = p.get_x() + p.get_width() / 10
  y = p.get_y() + p.get_height() + 2
  ax.annotate(percentage, (x, y))

plt.xlabel(xlabel = 'CustomerServiceCalls', size = 14)
plt.ylabel(ylabel = 'Frequency', size = 14)
plt.title(label = 'Frequency Distribution of CustomerServiceCalls with respect to
plt.legend(loc = 'upper right')
plt.grid(b = True)
plt.show()
```



Observation:

- The number of received calls (customer service) has been described in tabular form as below:

No. of Calls	Description
0	**Un-churned customers** are **approx 6.5X more** than **churned customers**.
1	**Un-churned customers** are **approx 9X more** than **churned customers**.
2	**Un-churned customers** are **approx 7.5X more** than **churned customers**.
3	**Un-churned customers** are **approx 9X more** than **churned customers**.
4	**Un-churned customers** are **approx 0.42% more** than **churned customers**.
5	**Un-churned customers** are **approx 1.5X less** than **churned customers**.
6	**Un-churned customers** are **approx 1.75X less** than **churned customers**.
7	**Un-churned customers** are **approx 0.03% less** than **churned customers**.
8	**Un-churned customers** are **equal** to the **churned customers**.
9	**Un-churned customers** are **approx 0.06% less** than **churned customers**.

Note: These are few question, from here if you would like to explore further, you are most welcome.

---

7. Post Data Processing & Feature Selection

- In this part we will **perform encoding over categorical features** and **feed it** to the **Random Forest** because machines can't understand human language.
- **Random Forest** will then **identify important features** for our model **using threshold** over the information gain over reduction in impurity.
- And **finally** we will **split** our **data** for the **model development**.

7.1 Encoding Categorical Features

- Before encoding the features we must identify the cardinality of the features.
- Then decide which type of encoding we should perform (Target, Dummy etc.).

```
cat_features = []
label_len = []

# Identify Categorical Features
for i in data.columns:
  if (data[i].dtype == object):
    cat_features.append(i)

# Identify Labels Length per Feature
for i in cat_features:
  label_len.append(len(data[i].unique()))

print('Total Categorical Features:', len(cat_features))

# Categorical Feature Frame Representation
cat_frame = pd.DataFrame(data = {'Length': label_len}, index = cat_features)
cat_frame.transpose()
```

```
Total Categorical Features: 3
```

	State	InternationalPlan?	VoiceMailPlan?
Length	51	2	2

Observation:

- We can **observe** that **State** has **high cardinality labels**. We will **perform K Fold Target Encoding** over this feature.
- For **rest** of the **features** we will **perform dummy encoding**.
- For **KFold Target Encoding** we have **created a class** as follows:

K Fold Target Encoding:

- **Target encoding** is **one of the most powerful techniques** in feature engineering.
- It has been **widely applied** and **developed** in **different forms**.
- The **limitation** of **Target Encoding** is the **overfitting** of the **data**.
- The **goal** of **K Fold Target Encoding** is to **reduce** the **overfitting** by **adding regularization** to the mean encoding.
- Let's take an example as follows:

Example | **Calculation**

#	Feature	Target	Encoded Feature
1	A	1	0.6
2	B	0	0.3
3	B	0	0.3
4	B	1	0.3
5	B	1	0.3
6	A	1	0.6
7	B	0	0.3
8	A	0	0.6
9	A	0	0.6
10	B	0	0.3
11	A	1	0.6
12	A	0	0.6
13	B	1	0.3
14	A	0	0.6
15	A	1	0.6
16	B	0	0.3
17	B	0	0.3
18	B	0	0.3
19	A	1	0.6
20	A	1	0.6

Here the count of A = 10, B = 10. The Mean(A): 6/10 = 0.6, Mean(B): 3/10 = 0.3

- In the above diagram, we can observer overfitting over label A. To handle this overfitting we will perform 5 fold target encoding.
- Each fold's categories are encoded based on the mean of the rest of the fold's categories.
- For first fold, mean values are estimated based on the rest of the folds i.e. Fold 2, Fold 3, Fold 4 & Fold 5.

Fold 1 View | **Calculation**

#	Folds	Feature	Target	K Fold Encoded Feature
1	Fold 1	A	1	0.556
2		B	0	0.285
3		B	0	0.285
4		B	1	0.285
5	Fold 2	B	1	
6		A	1	
7		B	0	
8		A	0	
9	Fold 3	A	0	
10		B	0	
11		A	1	
12		A	0	
13	Fold 4	B	1	
14		A	0	
15		A	1	
16		B	0	
17	Fold 5	B	0	
18		B	0	
19		A	1	
20		A	1	

The Mean(A): 5/9 = 0.556, Mean(B): 2/7 = 0.2

- For second fold, mean values are estimated based on the rest of the folds i.e. Fold 1, Fold 3, Fold 4 & Fold 5.

Fold 2 View	Calculation

#	Folds	Feature	Target	K Fold Encoded Feature
1	Fold 1	A	1	0.556
2		B	0	0.285
3		B	0	0.285
4		B	1	0.285
5	**Fold 2**	**B**	**1**	**0.25**
6		**A**	**1**	**0.625**
7		**B**	**0**	**0.25**
8		**A**	**0**	**0.625**
9	Fold 3	A	0	
10		B	0	
11		A	1	
12		A	0	
13	Fold 4	B	1	
14		A	0	
15		A	1	
16		B	0	
17	Fold 5	B	0	
18		B	0	
19		A	1	
20		A	1	

The Mean(A): 5/8 = 0.625, Mean(B): 2/8 = 0.250

- For third fold, mean values are estimated based on the rest of the folds i.e. Fold 1, Fold 2, Fold 4 & Fold 5.

Fold 3 View	Calculation

#	Folds	Feature	Target	K Fold Encoded Feature
1	Fold 1	A	1	0.556
2		B	0	0.285
3		B	0	0.285
4		B	1	0.285
5	Fold 2	B	1	0.25
6		A	1	0.625
7		B	0	0.25
8		A	0	0.625
9	**Fold 3**	**A**	**0**	**0.714**
10		**B**	**0**	**0.333**
11		**A**	**1**	**0.714**
12		**A**	**0**	**0.714**
13	Fold 4	B	1	
14		A	0	
15		A	1	
16		B	0	
17	Fold 5	B	0	
18		B	0	
19		A	1	
20		A	1	

The Mean(A): 5/7 = 0.714, Mean(B): 3/9 = 0.333

- For fourth fold, mean values are estimated based on the rest of the folds i.e. Fold 1, Fold 2, Fold 3 & Fold 5.

Fold 4 View **Calculation**

#	Folds	Feature	Target	K Fold Encoded Feature
1	Fold 1	A	1	0.556
2		B	0	0.285
3		B	0	0.285
4		B	1	0.285
5	Fold 2	B	1	0.25
6		A	1	0.625
7		B	0	0.25
8		A	0	0.625
9	Fold 3	A	0	0.714
10		B	0	0.333
11		A	1	0.714
12		A	0	0.714
13	**Fold 4**	**B**	**1**	**0.25**
14		**A**	**0**	**0.625**
15		**A**	**1**	**0.625**
16		**B**	**0**	**0.25**
17	Fold 5	B	0	
18		B	0	
19		A	1	
20		A	1	

The Mean(A): 5/8 = 0.625, Mean(B): 2/8 = 0.250

- For fifth fold, mean values are estimated based on the rest of the folds i.e. Fold 1, Fold 2, Fold 3 & Fold 4.

Fold 5 View **Calculation**

#	Folds	Feature	Target	K Fold Encoded Feature
1	Fold 1	A	1	0.556
2		B	0	0.285
3		B	0	0.285
4		B	1	0.285
5	Fold 2	B	1	0.25
6		A	1	0.625
7		B	0	0.25
8		A	0	0.625
9	Fold 3	A	0	0.714
10		B	0	0.333
11		A	1	0.714
12		A	0	0.714
13	Fold 4	B	1	0.25
14		A	0	0.625
15		A	1	0.625
16		B	0	0.25
17	Fold 5	B	0	0.375
18		B	0	0.375
19		A	1	0.5
20		A	1	0.5

The Mean(A): 4/8 = 0.500, Mean(B): 3/8 = 0.375

```
class KFoldTargetEncoder(BaseEstimator, TransformerMixin):
  def __init__(self ,colnames , targetName, n_fold = 5, verbosity = True, discard
    self.colnames = colnames
    self.targetName = targetName
    self.n_fold = n_fold
    self.verbosity = verbosity
    self.discardOriginal_col = discardOriginal_col

  def fit(self, X, y = None):
    return self

  def transform(self,X):
    assert(type(self.targetName) == str)
    assert(type(self.colnames) == str)
    assert(self.colnames in X.columns)
    assert(self.targetName in X.columns)
    mean_of_target = X[self.targetName].mean()
    kf = KFold(n_splits = self.n_fold, shuffle = False, random_state = 42)
    col_mean_name = self.colnames + '_' + 'Kfold_Target_Enc'
    X[col_mean_name] = np.nan
    for tr_ind, val_ind in kf.split(X):
      X_tr, X_val = X.iloc[tr_ind], X.iloc[val_ind]
      X.loc[X.index[val_ind], col_mean_name] = X_val[self.colnames].map(X_tr.grou
      X[col_mean_name].fillna(mean_of_target, inplace = True)
    if self.verbosity:
      encoded_feature = X[col_mean_name].values
      print('Correlation between the new feature, {} and, {} is {}.'.format(col_m
    if self.discardOriginal_col:
      X = X.drop(self.colnames, axis=1)
    return X
```

Performing Operations:

```
# Dummy Encoding -> ServiceProvider, DownloadOrUpload, Technology
data = pd.get_dummies(data = data, columns = ['InternationalPlan?', 'VoiceMailPla

####################################################################################

# Performing Target Encoding -> ServiceArea
kfold_te = KFoldTargetEncoder(colnames = 'State', targetName = 'Churn?', discardO
data = kfold_te.fit_transform(X = data)
```

```
Correlation between the new feature, State_Kfold_Target_Enc and, Churn? is 0.(
```

7.2 Feature Selection using Random Forest

```
X = data.drop('Churn?', axis = 1)
y = data['Churn?']
```

```
# Have some patience, may take some time :)
selector = SelectFromModel(RandomForestClassifier(n_estimators = 100, random_stat
selector.fit(X, y)

# Extracting list of important features
selected_feat = X.columns[(selector.get_support())].tolist()

print('Total Features Selected are', len(selected_feat))

# Estimated by taking mean(default) of feature importance
print('Threshold set by Model:', np.round(selector.threshold_, decimals = 2))
print('Features:', selected_feat)
```

```
Total Features Selected are 6
Threshold set by Model: 0.05
Features: ['TotalMorMin', 'TotalMorCharge', 'TotalEveMin', 'TotalEveCharge',
```

Observation:

- The important features marked by Random Forest are:

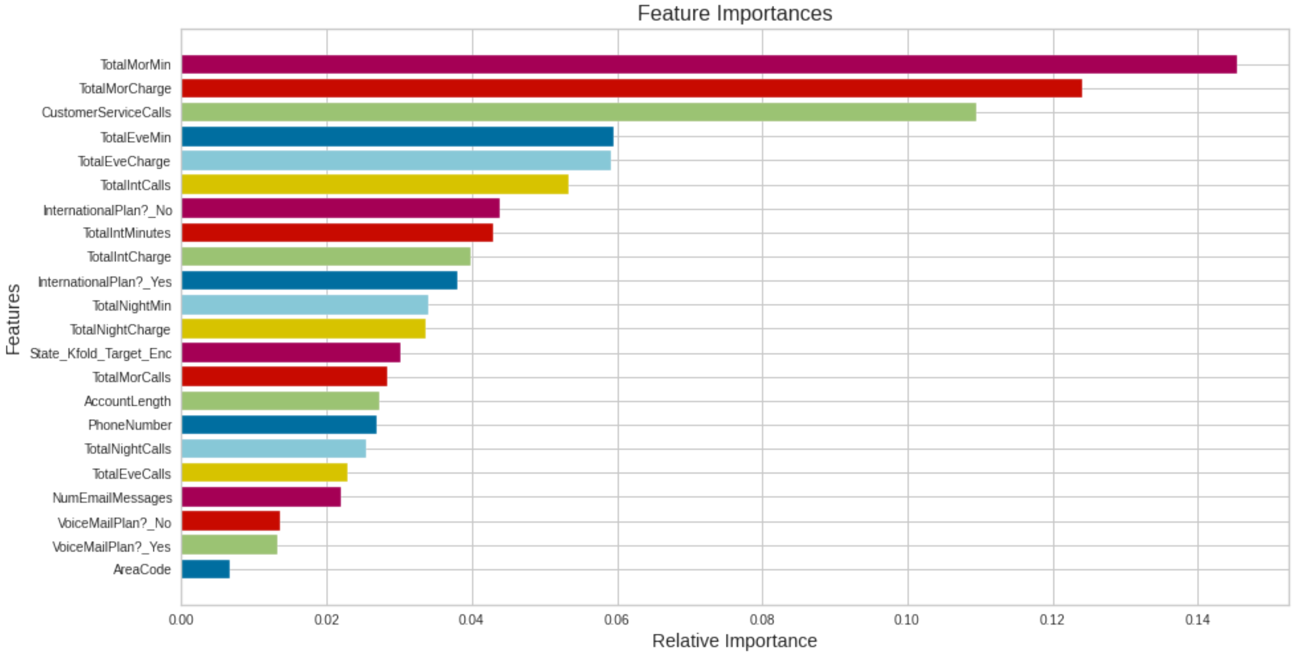
TotalMorMin	TotalMorCharge	TotalEveMin	TotalEveCharge	TotalIntCalls	CustomerServiceCalls

Potential Feature Estimation: Below features are plotted against their relative importance (in %age), of each feature.

```
# Have some patience, may take some time :)
figure = plt.figure(figsize = [15, 8])

# If you don't want relative importance, use relative = False in below method
viz = FeatureImportances(selector.estimator, relative = False)
viz.fit(X, y)

plt.xlabel('Relative Importance', size = 14)
plt.ylabel('Features', size = 14)
plt.title(label = 'Feature Importances', size = 16)
plt.show()
```



7.3 Data Preparation

- Now we will **split** our **data** in **training** and **testing** part for further development.

```
X = data[selected_feat]
y = data['Churn?']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random

print('Training Data Shape:', X_train.shape, y_train.shape)
print('Testing Data Shape:', X_test.shape, y_test.shape)
```

```
Training Data Shape: (2999, 6) (2999,)
Testing Data Shape: (334, 6) (334,)
```

Observation:

- Now that we have split out our data, we are **good to go with model development**.

---

8. Model Development & Evaluation

- In this section we will **develop different models using only important feature** and **tune** our **model if required**.
- Then we will **compare the results** obtained from them and **make our observation**.
- For **evaluation purpose** we will **focus** on **recall value** for **both** the **classes**.
- **Remember** that **we want generalize results** i.e. same results or error on testing data as that of training data.
- We will **observer whether** the **SMOTE** is **required** or not **because** we want to **focus** on **recall** values of **both** the **classes**.

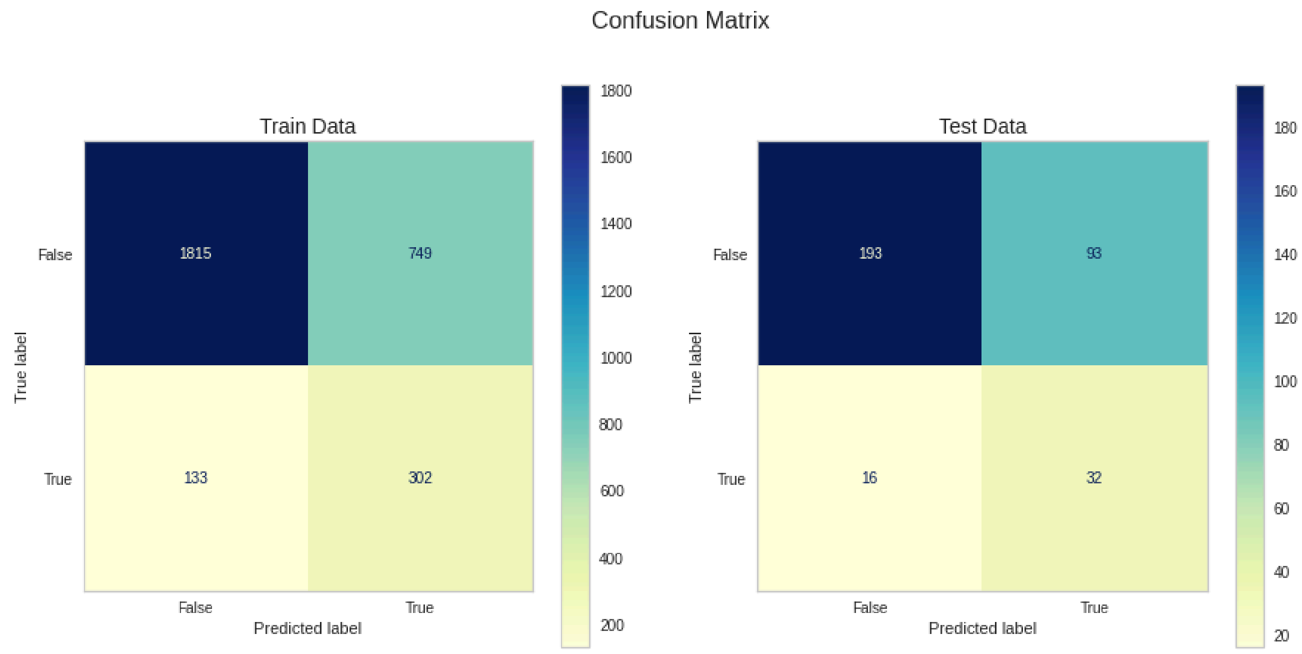
8.1 Baseline Models

8.1.1 Logistic Regression

```
log  = LogisticRegression(random_state = 42, class_weight = 'balanced')
log.fit(X_train, y_train)

y_train_pred_count = log.predict(X_train)
y_test_pred_count = log.predict(X_test)

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = False, figsize=(15,
plot_confusion_matrix(estimator = log, X = X_train, y_true = y_train, values_fo
plot_confusion_matrix(estimator = log, X = X_test, y_true = y_test, values_form
ax1.set_title(label = 'Train Data', size = 14)
ax2.set_title(label = 'Test Data', size = 14)
ax1.grid(b = False)
ax2.grid(b = False)
plt.suptitle(t = 'Confusion Matrix', size = 16)
plt.show()
```



Observation:

- **Train Data:**
    - Model predicted **1815 instances correctly** for **negative class** while **302 instances** were predicted **correctly** for **positive class**.
    - Model **identified 133 instances negative but in actual** they **were positive**.
    - Model **identified 749 instances positive but in actual** they **were negative**.
- **Test Data:**
    - Model predicted **193 instances correctly** for **negative class** while **32 instances** were predicted **correctly** for **positive class**.
    - Model **identified 16 instance negative but in actual** they **were positive**.
    - Model **identified 93 instances positive but in actual** they **were negative**.

```
logistic_report_train = classification_report(y_train, y_train_pred_count)
logistic_report_test = classification_report(y_test, y_test_pred_count)
print('                    Training Report           ')
print(logistic_report_train)
print('                    Testing Report            ')
print(logistic_report_test)
```

```
                    Training Report
              precision    recall  f1-score   support

       False       0.93      0.71      0.80      2564
        True       0.29      0.69      0.41       435
```

```
    accuracy                           0.71      2999
   macro avg       0.61      0.70      0.61      2999
weighted avg       0.84      0.71      0.75      2999

                  Testing Report
              precision    recall  f1-score   support

       False       0.92      0.67      0.78       286
        True       0.26      0.67      0.37        48

    accuracy                           0.67       334
   macro avg       0.59      0.67      0.57       334
weighted avg       0.83      0.67      0.72       334
```

Observation:

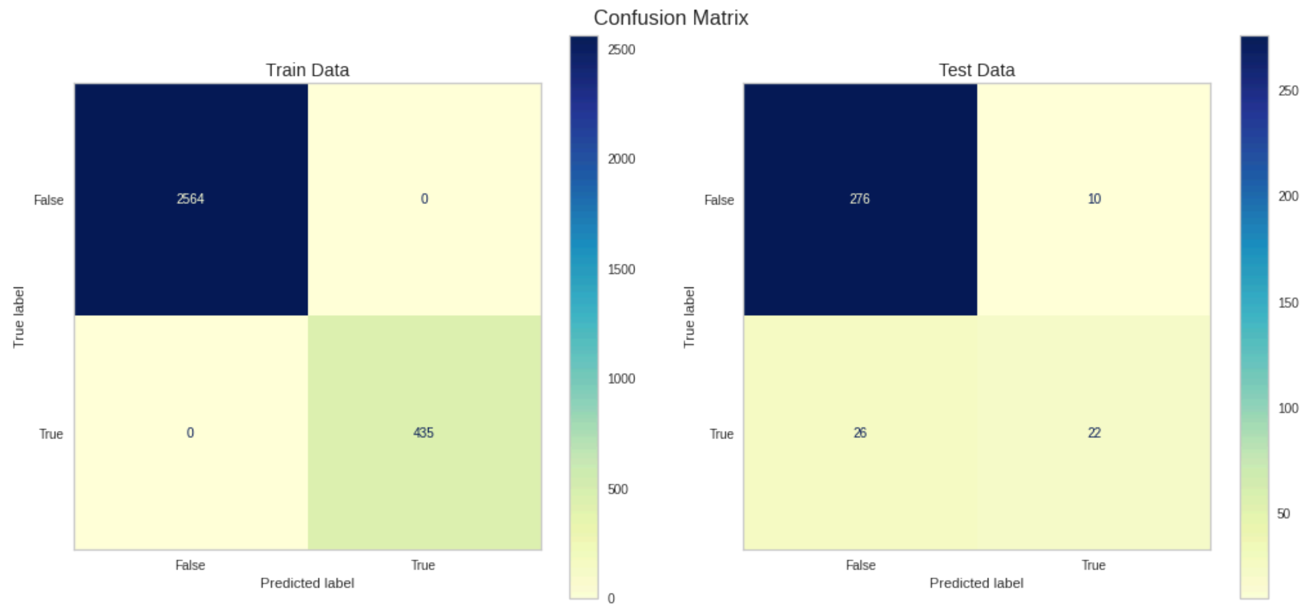
- We can **observe** that the **recall values** of **both** the **classes** are **trying** to **generalize** well.
- But we **need better results** and in order to that we **will try more complex models** in upcoming part.

8.1.2 Random Forest Classifier

```
rfc = RandomForestClassifier(n_jobs = -1, class_weight = 'balanced', random_state
rfc.fit(X_train, y_train)

y_train_pred_count = rfc.predict(X_train)
y_test_pred_count = rfc.predict(X_test)

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = False, figsize=(15
plot_confusion_matrix(estimator = rfc, X = X_train, y_true = y_train, values_fo
plot_confusion_matrix(estimator = rfc, X = X_test, y_true = y_test, values_forma
ax1.set_title(label = 'Train Data', size = 14)
ax2.set_title(label = 'Test Data', size = 14)
ax1.grid(b = False)
ax2.grid(b = False)
plt.suptitle(t = 'Confusion Matrix', size = 16)
plt.tight_layout(pad = 3.0)
plt.show()
```



Observation:

- **Train Data:**
  - Model predicted **2564 instances correctly** for **negative class** while **435 instances** were predicted **correctly** for **positive class**.
  - Model **identified 0 instances negative but in actual** they **were positive**.
  - Model **identified 0 instances positive but in actual** they **were negative**.
- **Test Data:**
  - Model predicted **276 instances correctly** for **negative class** while **22 instances** were predicted **correctly** for **positive class**.
  - Model **identified 26 instance negative but in actual** they **were positive**.
  - Model **identified 10 instances positive but in actual** they **were negative**.

```
rfc_report_train = classification_report(y_train, y_train_pred_count)
rfc_report_test = classification_report(y_test, y_test_pred_count)
print('                    Training Report          ')
print(rfc_report_train)
print('                    Testing Report           ')
print(rfc_report_test)
```

```
                    Training Report
              precision    recall  f1-score   support
```

```
       False       1.00      1.00      1.00      2564
        True       1.00      1.00      1.00       435

    accuracy                           1.00      2999
   macro avg       1.00      1.00      1.00      2999
weighted avg       1.00      1.00      1.00      2999

                   Testing Report
              precision    recall  f1-score   support

       False       0.91      0.97      0.94       286
        True       0.69      0.46      0.55        48

    accuracy                           0.89       334
   macro avg       0.80      0.71      0.74       334
weighted avg       0.88      0.89      0.88       334
```

Observation:

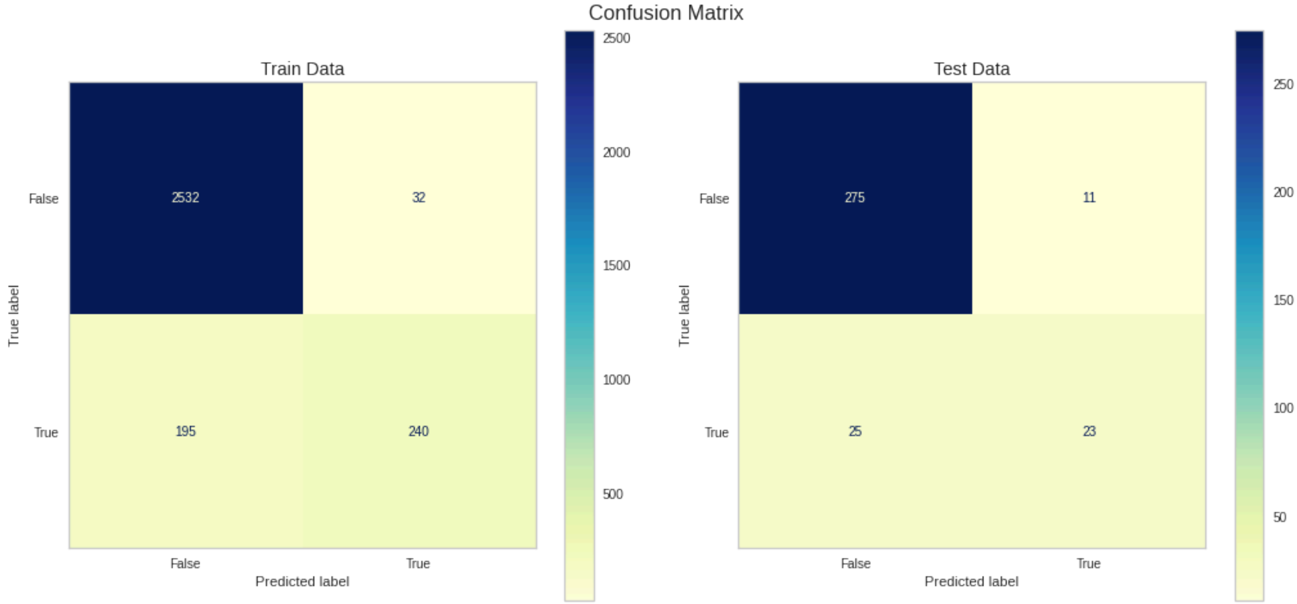
- Here we can **observe** some **good results** over the **splitted data sets (Train & Test)**.
- But **recall scores** on **test set** are **not good** for **positive class(True)**.
- This is **due to** the **class imbalance**, but **before oversampling** our data we will **try one more model**.

8.1.3 Extreme Gradient Boosting Classifier

```
xgb = XGBClassifier(random_state = 42, n_jobs = -1)
xgb.fit(X_train, y_train)

y_train_pred_count = xgb.predict(X_train)
y_test_pred_count = xgb.predict(X_test)

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = False, figsize=(15,
plot_confusion_matrix(estimator = xgb, X = X_train, y_true = y_train, values_form
plot_confusion_matrix(estimator = xgb, X = X_test, y_true = y_test, values_format
ax1.set_title(label = 'Train Data', size = 14)
ax2.set_title(label = 'Test Data', size = 14)
ax1.grid(b = False)
ax2.grid(b = False)
plt.suptitle(t = 'Confusion Matrix', size = 16)
plt.tight_layout(pad = 3.0)
plt.show()
```



Observation:

- **Train Data:**
    - Model predicted **2532 instances correctly** for **negative class** while **240 instances** were predicted **correctly** for **positive class**.
    - Model **identified 195 instances negative but in actual** they **were positive**.
    - Model **identified 32 instances positive but in actual** they **were negative**.
- **Test Data:**
    - Model predicted **275 instances correctly** for **negative class** while **23 instances** were predicted **correctly** for **positive class**.
    - Model **identified 25 instance negative but in actual** they **were positive**.
    - Model **identified 11 instances positive but in actual** they **were negative**.

```
xgb_report_train = classification_report(y_train, y_train_pred_count)
xgb_report_test = classification_report(y_test, y_test_pred_count)
print('                    Training Report          ')
print(xgb_report_train)
print('                    Testing Report           ')
print(xgb_report_test)
```

```
                    Training Report
              precision    recall  f1-score   support
```

```
       False       0.93      0.99      0.96      2564
        True       0.88      0.55      0.68       435

    accuracy                           0.92      2999
   macro avg       0.91      0.77      0.82      2999
weighted avg       0.92      0.92      0.92      2999

                  Testing Report
              precision    recall  f1-score   support

       False       0.92      0.96      0.94       286
        True       0.68      0.48      0.56        48

    accuracy                           0.89       334
   macro avg       0.80      0.72      0.75       334
weighted avg       0.88      0.89      0.88       334
```

Observation:

- The **recall scores** for **train data** has **dropped significantly** with respect to **previous models**.
- Regarding **test data** there's only **1% drop** in **recall** for **negative class** while for **positive class** it has **increased** to **2%** only.

8.2 Oversampling Models: Using Essential Features

SMOTE Technique & its Implementation

- **SMOTE refers to Synthetic Minority Oversampling Technique**.
- It **aims** to **balance class distribution** by **randomly increasing minority class** examples **by replicating** them.
- It **synthesises new minority instances between existing** minority **instances**.
- **It generates the virtual training records by linear interpolation for the minority class.**
- These **synthetic training records are generated by randomly selecting** one or more of the k-nearest neighbors for each **example in the minority class**.
- **After** the **oversampling process**, the **data is reconstructed** and several classification models can be applied for the processed data.

Before Implimenting SMOTE

```
print('Training Data Shape:', X_train.shape, y_train.shape)
print('Testing Data Shape:', X_test.shape, y_test.shape)
```

```
Training Data Shape: (2999, 6) (2999,)
Testing Data Shape: (334, 6) (334,)
```

Performing SMOTE Operation

```
# Have some patience, may take some time

sm = SMOTE(random_state = 42, ratio = 1)
X1, y1 = sm.fit_sample(X, y)

X_new = pd.DataFrame(data = X1, columns = X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_new, y1, test_size = 0.2, r


print('Training Data Shape:', X_train.shape, y_train.shape)
print('Testing Data Shape:', X_test.shape, y_test.shape)
```

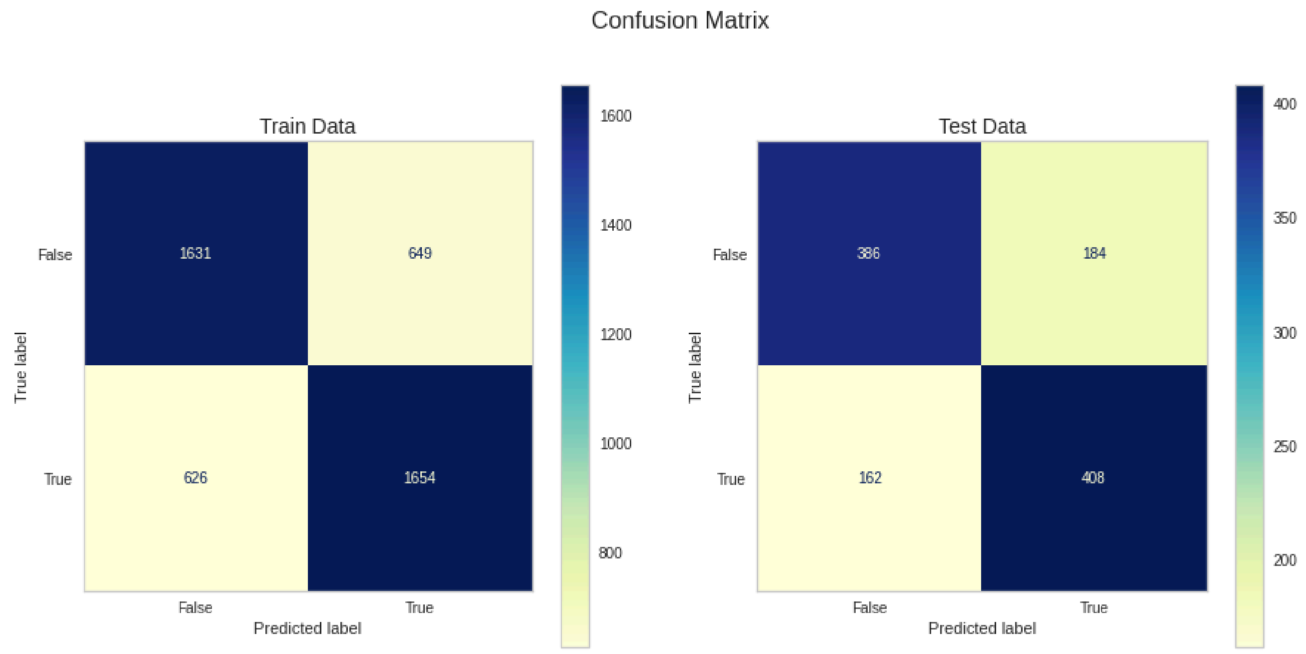
```
Training Data Shape: (4560, 6) (4560,)
Testing Data Shape: (1140, 6) (1140,)
```

8.2.1 Logistic Regression

```
log  = LogisticRegression(random_state = 42, class_weight = 'balanced')
log.fit(X_train, y_train)

y_train_pred_count = log.predict(X_train)
y_test_pred_count = log.predict(X_test)

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = False, figsize=(15
plot_confusion_matrix(estimator = log, X = X_train, y_true = y_train, values_fo
plot_confusion_matrix(estimator = log, X = X_test, y_true = y_test, values_form
ax1.set_title(label = 'Train Data', size = 14)
ax2.set_title(label = 'Test Data', size = 14)
ax1.grid(b = False)
ax2.grid(b = False)
plt.suptitle(t = 'Confusion Matrix', size = 16)
plt.show()
```



Observation:

- **Train Data:**
  - Model predicted **1631 instances correctly** for **negative class** while **1654 instances** were predicted **correctly** for **positive class**.
  - Model **identified 626 instances negative but in actual** they **were positive**.
  - Model **identified 649 instances positive but in actual** they **were negative**.
- **Test Data:**
  - Model predicted **386 instances correctly** for **negative class** while **408 instances** were predicted **correctly** for **positive class**.
  - Model **identified 162 instance negative but in actual** they **were positive**.
  - Model **identified 184 instances positive but in actual** they **were negative**.

```
logistic_report_train = classification_report(y_train, y_train_pred_count)
logistic_report_test = classification_report(y_test, y_test_pred_count)
print('                    Training Report          ')
print(logistic_report_train)
print('                    Testing Report           ')
print(logistic_report_test)
```

```
                              Training Report
                        precision    recall  f1-score   support

               False       0.72      0.72      0.72      2280
                True       0.72      0.73      0.72      2280
```

```
    accuracy                           0.72      4560
   macro avg       0.72      0.72      0.72      4560
weighted avg       0.72      0.72      0.72      4560

                  Testing Report
              precision    recall  f1-score   support

       False       0.70      0.68      0.69       570
        True       0.69      0.72      0.70       570

    accuracy                           0.70      1140
   macro avg       0.70      0.70      0.70      1140
weighted avg       0.70      0.70      0.70      1140
```

Observation:

- We can **observe better results** over **than** the **logisitic baseline model**.
- **Earlier recall scores** were **low** but **now** they have **improved a little bit**.
- **Let's try more complex models.**