Final Project Submission - Module 3

- Student name: Maria Kimotho Ruiri
- Student pace: self paced / part time
- Blog post URL: https://maria-kimotho.medium.com/phase-3-project-f0f7

Business Problem Understanding

This analysis aims to:

- Investigate the presence and significance of patterns of the Customer Churn Rate for the SyriaTel telecommunications company.
- Build a classifier predict whether a customer will stop doing business with the company.

Stakeholders can expect benefits from the information in the company's both creators and partners.

Customer Churn

What is it?

Customer churn or customer attrition is the phenomenon where customers of a business no longer purchase or interact with the business for products or services.

Churn rate or customer attrition rate is the critical metric that quantifies membership this phenomenon.

Why is it important?

The customer acquisition between the business running and thriving. Generally, the level of customers is not sudden but the result of a prolonged succession of unsatisfying experiences. In this way, customer churn is a lagging indicator.

Customer churn is a sensitive issue for a business, challenging and extensive investment from several groups of roles. Gaining new customers is a costly more financial consuming endeavour. On the flipping side "conservation is cheap" and "the former impacts the latter" are phrases that are commonly used. Customer churn can be eliminated or prevented the first.

Due to the number of fact, the analysis of the customer churn can consolidate the base business while consuming far less expansion opportunities while avoid crisis and catastrophic consequences on the entire business, even ceasing to exist.

Customer churn is a sensitive issue for a business.

Churn rate = Number of customers lost / Total number of clients at the beginning of the interested time period x 100

Left Unaware different models both parameters and test operations.

EVALUATION

Models fitting will be evaluated by the score

EDA & FEATURE ENGINEERING

```
In [1]: # Import numpy, pandas, and matplotlib
        %matplotlib inline
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import sklearn.preprocessing as preprocessing
        import sklearn.model_selection
        import sklearn.metrics
        import sklearn.ensemble
        import sklearn.tree
        import sklearn.linear_model
```

The dataset imported is in a csv file, SyriaTel Churn Dataset

```
In [2]: # Import the csv file and inspecting dataset
        df = pd.read_csv('data/bigin.csv')
        df.info()
```

```
In [3]: # Displaying the dataset
        df.head()
```

```
In [4]: # Displaying the dataset shape
        print("Number of rows: {:d}\n"\
              "Number of columns: {:d}".format(df.shape[0], df.shape[1]))
```

```
In [5]: # Columns
        df.columns
```

```
In [6]: # Displaying the existing values
        df.describe(include='all')
```

```
In [7]: # Checking for missing values
        df.isnull().sum()
```

```
In [8]: # Checking for duplicates
        df[df.duplicated(keep=False)].shape[0]
```

```
In [9]: # Displaying the existing values
        df.describe(include='all')
```

```
In [10]: # Dropping the phone number column
         df = df.drop('phone number', axis=1)
         df.shape
```

```
In [11]: # Checking the shape
         df.shape
```

```
In [12]: # Checking the first 5 rows
         df.head()
```

```
In [13]: # Checking the last 5 rows
         df.tail()
```

```
In [14]: # Checking the data types
         df.dtypes
```

```
In [15]: # Checking the unique values
         df.nunique()
```

```
In [16]: # Checking the value counts for each column
         for col in df.columns:
             print("{:s}\n{:s}".format(str(df[col].value_counts()), '-'*20))
```

```
In [17]: # Checking the value counts for the churn column
         df['churn'].value_counts()
```

```
In [18]: # Checking the value counts for the churn column in percentage
         df['churn'].value_counts(normalize=True)*100
```

```
In [19]: # Checking the value counts for the area code column
         df['area code'].value_counts()
```

```
In [20]: # Checking the value counts for the area code column in percentage
         df['area code'].value_counts(normalize=True)*100
```

```
In [21]: # Checking the value counts for the area code column in percentage
         df['area code'].value_counts(normalize=True)*100
```

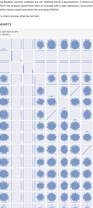
```
In [22]: # Checking the value counts for the area code column in percentage
         df['area code'].value_counts(normalize=True)*100
```

```
In
```

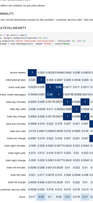
```
sns.histplot(df['total night charge'],palette, hue='churn', multiple='stack')
plt.xlabel('total night charge')
plt.title("Distribution of total night charge by churn")
```

```
sns.histplot(df['international plan'],palette, hue='churn', multiple='stack')
plt.xlabel('international plan')
plt.title("Distribution of international plan by churn")
```

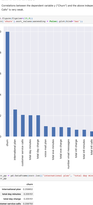
```
sns.histplot(df['total intl charge'],palette, hue='churn', multiple='stack')
plt.xlabel('total intl charge')
plt.title("Distribution of total intl charge by churn")
```




```
sns.histplot(df['voice mail plan'],palette, hue='churn', multiple='stack')
plt.xlabel('voice mail plan')
plt.title("Distribution of voice mail plan by churn")
```




```
sns.histplot(df['customer service calls'],palette, hue='churn', multiple='stack')
plt.xlabel('customer service calls')
plt.title("Distribution of customer service calls by churn")
```




```
df['Churn'].value_counts()
df['Churn'].value_counts(normalize=True)
sns.boxplot(data=df,x='Churn',y='Total day charge')
plt.title('Total day charge by Churn')
```

```
Churn
False    2850
True     483
Name: count, dtype: int64
Churn
False    0.855856
True     0.144144
Name: Churn, dtype: float64
```

FEATURES SELECTION

Target:

- Dependent Variable
- Churn

Predictors:

- Categorical Predictors
  - International plan
  - Voice mail plan
  - Customer Service calls
- Numerical Predictors
  - All remaining ones

ASSUMPTIONS FOR LINEAR REGRESSION

Being Boolean, survey variables are not violating the 0-5 assumption. It makes sense check the assumptions when independent variables are continuous. Another possibility to check assumptions is to transform the continuous variables into ordinal variables, assuming the ordinal ones behave like a normal variable. Let's see if the models in between these two ways differ.

Let's check anyway what we can here:

LINEARITY

```
sns.pairplot(df)
plt.show()
```

No bad linear relationships detected from the scatterplot matrix

NO AUTOCORRELATION

Condition not violated - no pairs plots above

NORMALITY

No non-normalisation except for the variable "customer service calls". Bad Histogram plots above

MULTICOLLINEARITY

```
corr = df.corr()
sns.heatmap(corr,cmap='Blues')
plt.title('Churn Features Correlations')
plt.show()
```

Highest scores are related to:

- International plan (0.26)
- Voice mail plan (0.21)
- Churn (1.00)
- customer service calls (0.21)

Correlations between the dependent variable y ("Churn") and the above independent variables ("International Plan", "Third Day Minutes", "Third Day Charge", "Customer Services Calls") are very weak.

```
pd.Series(corr['churn'].sort_values(ascending=False),name='corr')
```

```
corr_df = pd.DataFrame(corr.loc['international plan', 'total day minutes', 'total day charge', 'customer service calls'],'churn')
corr_df.sort_values(ascending=False)
```

	churn
international plan	0.259852
total day minutes	0.205151
total day charge	0.205151
customer service calls	0.208750

FEATURES SELECTION AND MODELING

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
```

```
# Separating the data into X (predictors) and y (target) variables as series.
X = df[['international plan', 'total day charge', 'customer service calls']]
y = df['churn']
print(X.shape, y.shape)
X = X.astype(int).values.reshape(-1, 3)
print(X.shape, y.shape)
```

```
(3333, 3) (3333,)
(3333, 3) (3333,)
```

```
# Multiple linear regression X train/test Dataframes
X = df[['international plan', 'total day charge', 'customer service calls']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```
# Feature selection and model scores.
k = 3
fs = SelectKBest(score_func=f_classif, k=k)
fs.fit(X_train, y_train)
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
```

```
# Logistic regression model.
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_fs, y_train)
yhat = lr.predict(X_test_fs)
print("Coefficients: ", lr.coef_)
print("Intercept: ", lr.intercept_)
```

```
Coefficients:  [[0.2600 0.2100 0.2100]]
Intercept:  [-2.0000]
```

```
# Accuracy score for logistic regression.
print('Accuracy Score: ', accuracy_score(y_test, yhat))
print('Confusion Matrix:\n', confusion_matrix(y_test, yhat))
print('Classification Report:\n', classification_report(y_test, yhat))
```

```
Accuracy Score:  0.8571
Confusion Matrix:
 [[1425   0]
 [ 238   0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1425
           1       0.00      0.00      0.00       238

    accuracy                           0.86      1663
   macro avg       0.43      0.50      0.46      1663
weighted avg       0.73      0.86      0.79      1663
```

```
# Binarized predictions for converting and roc curve.
y_test_binary = (y_test.astype(int).values.reshape(-1, 1) == 1).astype(int)
y_pred_binary = (yhat.astype(int).reshape(-1, 1) == 1).astype(int)

y_test_binary = (y_test.astype(int).ravel() == 1).astype(int)
y_pred_binary = (yhat.astype(int).ravel() == 1).astype(int)
fpr, tpr, thresholds = roc_curve(y_test_binary, y_pred_binary)
print("Binarized shapes:", y_test_binary.shape, y_pred_binary.shape)
```

```
Binarized shapes: (1663,) (1663,)
```

```
roc_auc = roc_auc_score(y_test_binary, y_pred_binary)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.5000
```

```
# Coefficients and intercept.
coefficients = lr.coef_[0]
intercept = lr.intercept_[0]
feature_names = ['international plan', 'total day charge', 'customer service calls']
coef_df = pd.DataFrame({'Feature': feature_names, 'Coef': coefficients})
print("Coefficients:\n", coef_df)
```

```
Coefficients:
                 Feature  Coef
0  international plan  0.26
1      total day charge  0.21
2  customer service calls  0.21
```

```
fig_w = lr.predict_proba(X_test_fs)[:,1]
fig_w[:5]
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1])
```

```
fig = px.pie(coef_df, values='Coef', names='Feature')
fig.show()
```



```
fig = px.bar(coef_df, x='Feature', y='Coef')
fig.show()
```

```
# Cross-validation.
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(max_iter=1000))
])

for k in range(1, 4):
    fs = SelectKBest(score_func=f_classif, k=k)
    X_train_fs = fs.fit_transform(X_train, y_train)

    lr = pipeline.fit(X_train, y_train)

    cv_scores = cross_val_score(
        pipeline,
        X_train_fs,
        y_train,
        cv=5,
        scoring='accuracy'
    )
    print(f"k={k}: Cross-validation Coefficients={cv_scores.mean():.4f}")
```

```
k=1: Cross-validation Coefficients=0.8571
k=2: Cross-validation Coefficients=0.8571
k=3: Cross-validation Coefficients=0.8571
```



```
# Cross-validation.
from sklearn.model_selection import cross_val_score

# Pipeline with StandardScaler.
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(max_iter=1000))
])

# Cross-validation.
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring='accuracy')
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.8571 0.8571 0.8571 0.8571 0.8571]
Mean CV accuracy: 0.8571
```

```
# Feature selection.
from sklearn.feature_selection import SelectKBest, f_classif

k = 3
fs = SelectKBest(score_func=f_classif, k=k)
X_train_fs = fs.fit_transform(X_train, y_train)
X_test_fs = fs.transform(X_test)

print("Selected features:", fs.get_support())
```

```
Selected features: [True True True]
```

```
# Logistic regression.
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_fs, y_train)
yhat = lr.predict(X_test_fs)

print("Coefficients: ", lr.coef_)
print("Intercept: ", lr.intercept_)
```

```
Coefficients:  [[0.2600 0.2100 0.2100]]
Intercept:  [-2.0000]
```

```
# Accuracy score.
print('Accuracy Score: ', accuracy_score(y_test, yhat))
```

```
Accuracy Score:  0.8571
```

```
# Confusion matrix.
print('Confusion Matrix:\n', confusion_matrix(y_test, yhat))
```

```
Confusion Matrix:
 [[1425   0]
 [ 238   0]]
```

```
# Classification report.
print('Classification Report:\n', classification_report(y_test, yhat))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1425
           1       0.00      0.00      0.00       238

    accuracy                           0.86      1663
   macro avg       0.43      0.50      0.46      1663
weighted avg       0.73      0.86      0.79      1663
```

```
# Binarized predictions for converting and roc curve.
y_test_binary = (y_test.astype(int).values.reshape(-1, 1) == 1).astype(int)
y_pred_binary = (yhat.astype(int).reshape(-1, 1) == 1).astype(int)

y_test_binary = (y_test.astype(int).ravel() == 1).astype(int)
y_pred_binary = (yhat.astype(int).ravel() == 1).astype(int)
fpr, tpr, thresholds = roc_curve(y_test_binary, y_pred_binary)
print("Binarized shapes:", y_test_binary.shape, y_pred_binary.shape)
```

```
Binarized shapes: (1663,) (1663,)
```

```
roc_auc = roc_auc_score(y_test_binary, y_pred_binary)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.5000
```

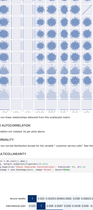
```
# Coefficients and intercept.
coefficients = lr.coef_[0]
intercept = lr.intercept_[0]
feature_names = ['international plan', 'total day charge', 'customer service calls']
coef_df = pd.DataFrame({'Feature': feature_names, 'Coef': coefficients})
print("Coefficients:\n", coef_df)
```

```
Coefficients:
                 Feature  Coef
0  international plan  0.26
1      total day charge  0.21
2  customer service calls  0.21
```

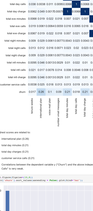
```
fig_w = lr.predict_proba(X_test_fs)[:,1]
fig_w[:5]
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1])
```

```
fig = px.pie(coef_df, values='Coef', names='Feature')
fig.show()
```




```
fig = px.bar(coef_df, x='Feature', y='Coef')
fig.show()
```




```
# Cross-validation.
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(max_iter=1000))
])

for k in range(1, 4):
    fs = SelectKBest(score_func=f_classif, k=k)
    X_train_fs = fs.fit_transform(X_train, y_train)

    lr = pipeline.fit(X_train, y_train)

    cv_scores = cross_val_score(
        pipeline,
        X_train_fs,
        y_train,
        cv=5,
        scoring='accuracy'
    )
    print(f"k={k}: Cross-validation Coefficients={cv_scores.mean():.4f}")
```

```
k=1: Cross-validation Coefficients=0.8571
k=2: Cross-validation Coefficients=0.8571
k=3: Cross-validation Coefficients=0.8571
```



```
# Cross-validation.
from sklearn.model_selection import cross_val_score

# Pipeline with StandardScaler.
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(max_iter=1000))
])

# Cross-validation.
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring='accuracy')
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.8571 0.8571 0.8571 0.8571 0.8571]
Mean CV accuracy: 0.8571
```

```
# Feature selection.
from sklearn.feature_selection import SelectKBest, f_classif

k = 3
fs = SelectKBest(score_func=f_classif, k=k)
X_train_fs = fs.fit_transform(X_train, y_train)
X_test_fs = fs.transform(X_test)

print("Selected features:", fs.get_support())
```

```
Selected features: [True True True]
```

```
# Logistic regression.
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_fs, y_train)
yhat = lr.predict(X_test_fs)

print("Coefficients: ", lr.coef_)
print("Intercept: ", lr.intercept_)
```

```
Coefficients:  [[0.2600 0.2100 0.2100]]
Intercept:  [-2.0000]
```

```
# Accuracy score.
print('Accuracy Score: ', accuracy_score(y_test, yhat))
```

```
Accuracy Score:  0.8571
```

```
# Confusion matrix.
print('Confusion Matrix:\n', confusion_matrix(y_test, yhat))
```

```
Confusion Matrix:
 [[1425   0]
 [ 238   0]]
```

```
# Classification report.
print('Classification Report:\n', classification_report(y_test, yhat))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1425
           1       0.00      0.00      0.00       238

    accuracy                           0.86      1663
   macro avg       0.43      0.50      0.46      1663
weighted avg       0.73      0.86      0.79      1663
```

```
# Binarized predictions for converting and roc curve.
y_test_binary = (y_test.astype(int).values.reshape(-1, 1) == 1).astype(int)
y_pred_binary = (yhat.astype(int).reshape(-1, 1) == 1).astype(int)

y_test_binary = (y_test.astype(int).ravel() == 1).astype(int)
y_pred_binary = (yhat.astype(int).ravel() == 1).astype(int)
fpr, tpr, thresholds = roc_curve(y_test_binary, y_pred_binary)
print("Binarized shapes:", y_test_binary.shape, y_pred_binary.shape)
```

```
Binarized shapes: (1663,) (1663,)
```

```
roc_auc = roc_auc_score(y_test_binary, y_pred_binary)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.5000
```

```
# Coefficients and intercept.
coefficients = lr.coef_[0]
intercept = lr.intercept_[0]
feature_names = ['international plan', 'total day charge', 'customer service calls']
coef_df = pd.DataFrame({'Feature': feature_names, 'Coef': coefficients})
print("Coefficients:\n", coef_df)
```

```
Coefficients:
                 Feature  Coef
0  international plan  0.26
1      total day charge  0.21
2  customer service calls  0.21
```

```
fig_w = lr.predict_proba(X_test_fs)[:,1]
fig_w[:5]
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1])
```

```
fig = px.pie(coef_df, values='Coef', names='Feature')
fig.show()
```

```
fig = px.bar(coef_df, x='Feature', y='Coef')
fig.show()
```

```
In[ ]: from sklearn.metrics import classification_report

In[ ]: y_train_1 = y_train
       y_test_1 = y_test
       y_train_1.shape
       y_test_1.shape
       y_train_1[y_train_1==1].shape
       y_test_1[y_test_1==1].shape

In[ ]: X_train_1[X_train_1[X_train_1[
```