Hi Folks!

We are **G.E Community**

CHURN PREDICTION ESTIMATOR WITH MACHINE LEARNING APPROACH

Background

Classification is a method for classify an object by determining from the feature indication. This work has an idea to classify the potential of churn from the cellular business.

We're proposed a scheme by machine learning approach with utilization of K-Nearest Neighbor, Logistic Regression, and Random Forest to determine of churn potential.

The main objectives from this work:

- Identify by exploratory data analysis to gain some insight
- Obtain best performance by comparison of model
- Explain by graph visualize according this work

Workflow

1. About Dataset
2. Data Cleaning
3. Data Preprocessing
    - Summary Categorical Data
    - Add new Column (`'Total Calls'`, `'Total Charge'`, `'Many Service Call'`)
    - Correlation
4. Explanatory Data
5. Normalization Data (`MinMaxScalar`, `StandartScalar`)
6. Modelling (`Logistic Regression`, `KNN`, `Random Forest`)

About Dataset

```
Data columns (total 20 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   state                          4250 non-null   object
 1   account_length                 4250 non-null   int64
 2   area_code                      4250 non-null   object
 3   international_plan             4250 non-null   object
 4   voice_mail_plan                4250 non-null   object
 5   number_vmail_messages          4250 non-null   int64
 6   total_day_minutes              4250 non-null   float64
 7   total_day_calls                4250 non-null   int64
 8   total_day_charge               4250 non-null   float64
 9   total_eve_minutes              4250 non-null   float64
 10  total_eve_calls                4250 non-null   int64
 11  total_eve_charge               4250 non-null   float64
 12  total_night_minutes            4250 non-null   float64
 13  total_night_calls              4250 non-null   int64
 14  total_night_charge             4250 non-null   float64
 15  total_intl_minutes             4250 non-null   float64
 16  total_intl_calls               4250 non-null   int64
 17  total_intl_charge              4250 non-null   float64
 18  number_customer_service_calls  4250 non-null   int64
 19  churn                          4250 non-null   object
dtypes: float64(8), int64(7), object(5)
```

```
[ ]  df.shape

     (4250, 20)
```

- Our dataset consist of 4250 rows and 20 columns containing of cellular variabel.
- There are no missing values found in our dataset
- Our dataset consist of 5 categorical data, and 15 numerical data

Data Cleaning (Missing Value, Duplicated, Nunique)

```
state                            0
account_length                   0
area_code                        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
```

```
[176] df.duplicated().sum()

      0

[177] df.nunique()

      state                              51
      account_length                    215
      area_code                           3
      international_plan                  2
      voice_mail_plan                     2
      number_vmail_messages              46
      total_day_minutes                1843
      total_day_calls                   120
      total_day_charge                 1843
      total_eve_minutes                1773
      total_eve_calls                   123
      total_eve_charge                 1572
      total_night_minutes              1757
      total_night_calls                 128
      total_night_charge                992
      total_intl_minutes                168
      total_intl_calls                   21
      total_intl_charge                 168
      number_customer_service_calls      10
      churn                               2
```

Data Preprocessing (Summary Of Categorical Data)

	state	area_code	international_plan	voice_mail_plan	churn
count	4250	4250	4250	4250	4250
unique	51	3	2	2	2
top	WV	area_code_415	no	no	no
freq	139	2108	3854	3138	3652

Data Preprocessing (Summary Of Numerical Data)

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls
count	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000
mean	100.236235	0.093176	0.261647	7.631765	180.259600	99.907294	30.644682	200.173906	100.176471
std	39.698401	0.290714	0.439583	13.439882	54.012373	19.850817	9.182096	50.249518	19.908591
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	73.000000	0.000000	0.000000	0.000000	143.325000	87.000000	24.365000	165.925000	87.000000
50%	100.000000	0.000000	0.000000	0.000000	180.450000	100.000000	30.680000	200.700000	100.000000
75%	127.000000	0.000000	1.000000	16.000000	216.200000	113.000000	36.750000	233.775000	114.000000
max	243.000000	1.000000	1.000000	52.000000	351.500000	165.000000	59.760000	359.300000	170.000000

Data Preprocessing (Add New Column)

```
[294] total_calls = (
          df["total_day_calls"] + df["total_eve_calls"] + df["total_night_calls"]
          + df["total_intl_calls"]
      )
      df.insert(loc=len(df.columns), column="total_calls", value=total_calls)
      df.head()
```

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	churn	total_calls
16.62	254.4	103	11.45	13.7	3	3.70	1	no	332
10.30	162.6	104	7.32	12.2	5	3.29	0	no	333
5.26	196.9	89	8.86	6.6	7	1.78	2	no	255
12.61	186.9	121	8.41	10.1	3	2.73	3	no	359
29.62	212.6	118	9.57	7.5	7	2.03	3	no	321

```
[291] columns_to_show = ["total_day_minutes", "total_eve_minutes", "total_night_minutes"]

      df.groupby(["churn"])[columns_to_show].describe(percentiles=[])
```

	total_day_minutes						total_eve_minutes						total_night_minutes					
	count	mean	std	min	50%	max	count	mean	std	min	50%	max	count	mean	std	min	50%	max
churn																		
no	3652.0	175.555093	49.549782	0.0	178.25	313.8	3652.0	198.570674	49.897726	0.0	199.2	359.3	3652.0	199.577519	50.521152	0.0	199.3	395.0
yes	598.0	208.990134	69.183493	0.0	220.55	351.5	598.0	209.964883	51.312321	70.9	210.2	349.4	598.0	206.331773	48.959820	47.4	206.1	381.6

Data Preprocessing (Add New Column)

```
[295] columns_to_show = ["total_day_charge", "total_eve_charge", "total_night_charge"]

    df.groupby(["churn"])[columns_to_show].describe(percentiles=[])
```

	total_day_charge						total_eve_charge						total_night_charge					
	count	mean	std	min	50%	max	count	mean	std	min	50%	max	count	mean	std	min	50%	max
churn																		
no	3652.0	29.844948	8.423424	0.0	30.300	53.35	3652.0	16.878743	4.241312	0.00	16.93	30.54	3652.0	8.981131	2.273463	0.00	8.970	17.77
yes	598.0	35.528679	11.761417	0.0	37.495	59.76	598.0	17.847207	4.361545	6.03	17.87	29.70	598.0	9.285033	2.203215	2.13	9.275	17.17

```
total_charge = (
    df["total_day_charge"] + df["total_eve_charge"] + df["total_night_charge"]
    + df["total_intl_charge"]
)
df.insert(loc=len(df.columns), column="total_charge", value=total_charge)
df.head()
```

...	total_night_minutes	total_night_calls	total_night_charge	total_intl_minutes	total_intl_calls	total_intl_charge	number_customer_service_calls	churn	total_calls	total_charge
...	254.4	103	11.45	13.7	3	3.70	1	no	332	59.24
...	162.6	104	7.32	12.2	5	3.29	0	no	333	62.29
...	196.9	89	8.86	6.6	7	1.78	2	no	255	66.80
...	186.9	121	8.41	10.1	3	2.73	3	no	359	52.09
...	212.6	118	9.57	7.5	7	2.03	3	no	321	78.31

Data Preprocessing (Add New Column)

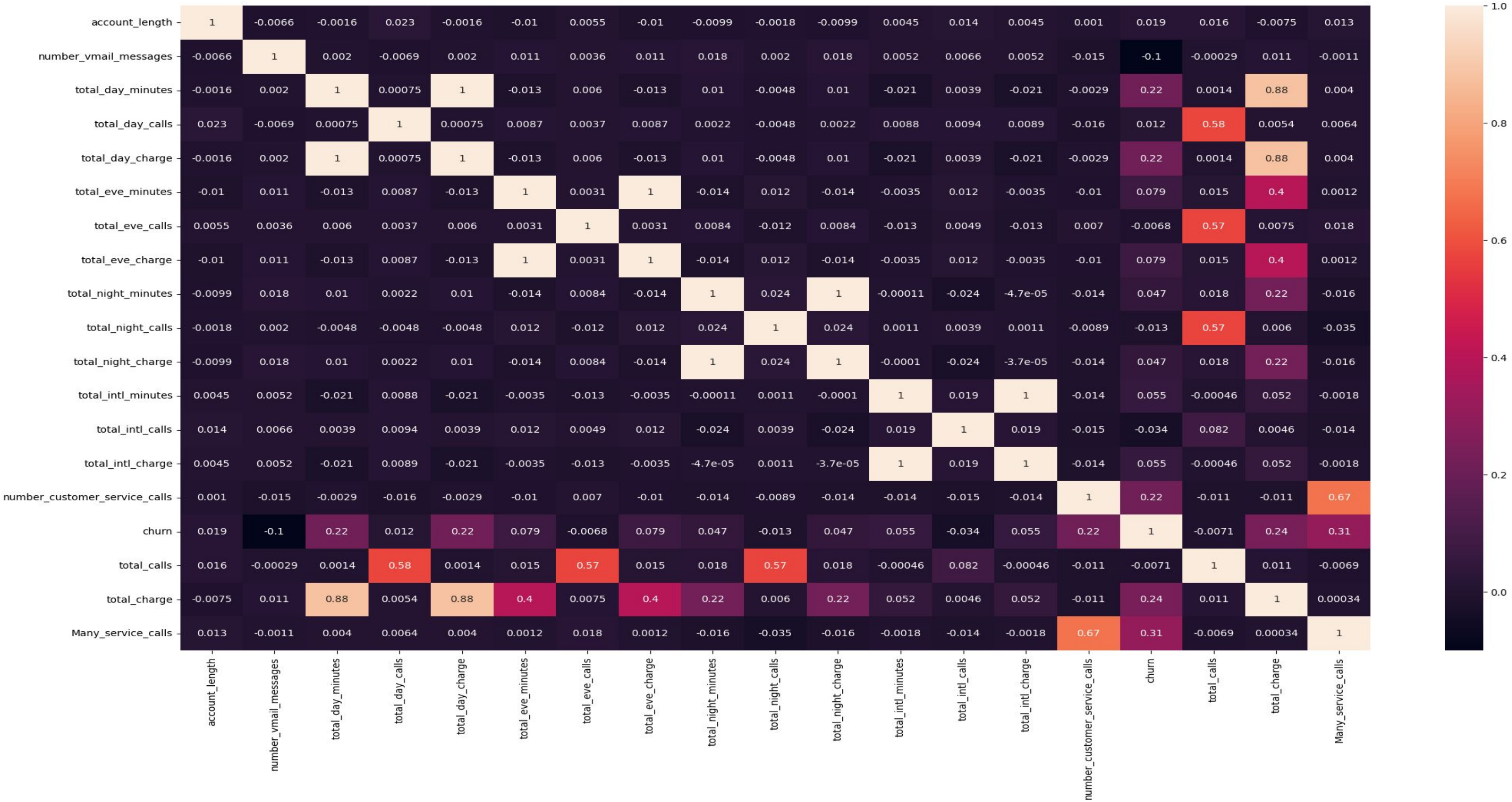
```
[298] pd.crosstab(df["churn"], df["number_customer_service_calls"], margins=True)
```

number_customer_service_calls	0	1	2	3	4	5	6	7	8	9	All
churn											
no	789	1358	845	495	117	32	9	6	1	0	3652
yes	97	166	102	63	92	49	19	7	1	2	598
All	886	1524	947	558	209	81	28	13	2	2	4250

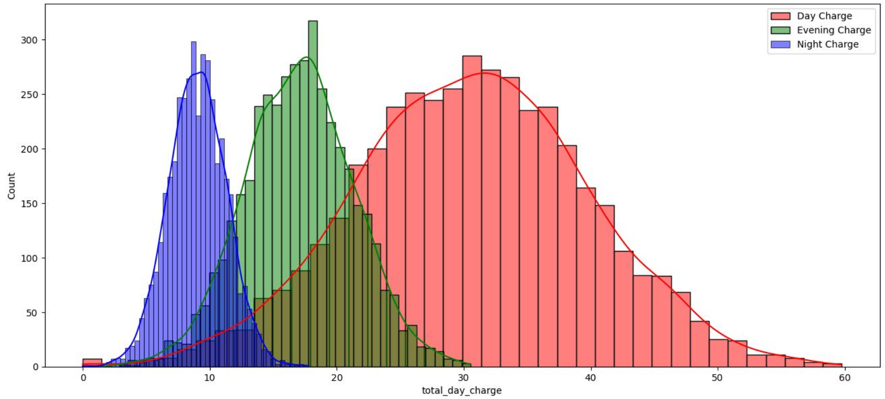
```
[300] df["Many_service_calls"] = (df["number_customer_service_calls"] > 3).astype("int")

      pd.crosstab(df["Many_service_calls"], df["churn"], margins=True)
```

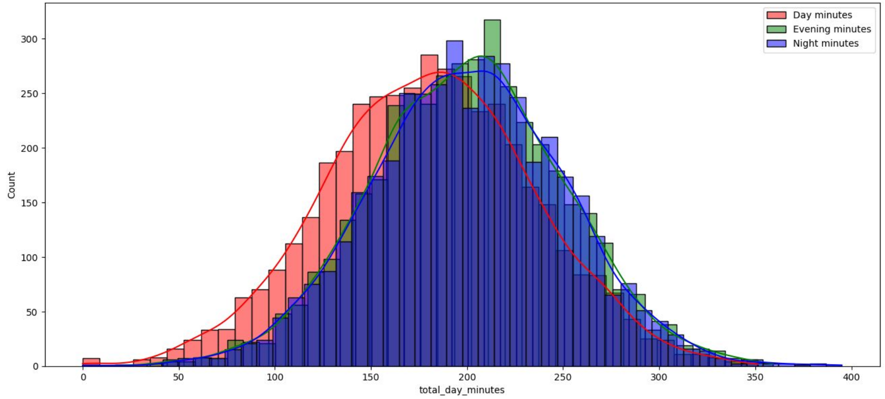
churn	no	yes	All
Many_service_calls			
0	3487	428	3915
1	165	170	335
All	3652	598	4250

Data Preprocessing (Heatmap Correlation)



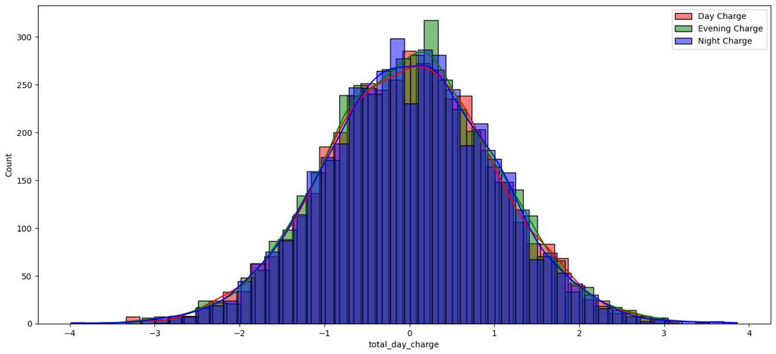
EXPLORATORY DATA ANALYSIS

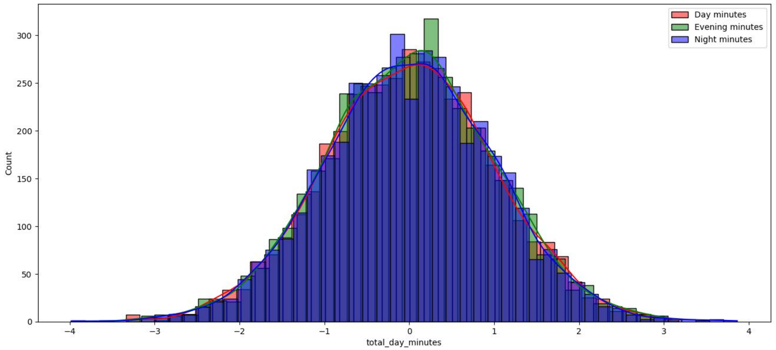




- The figure shown a couple of column doesn’t has a same values of range. As shown from the data centralization, it need some of improvement by did a standardized of dataset

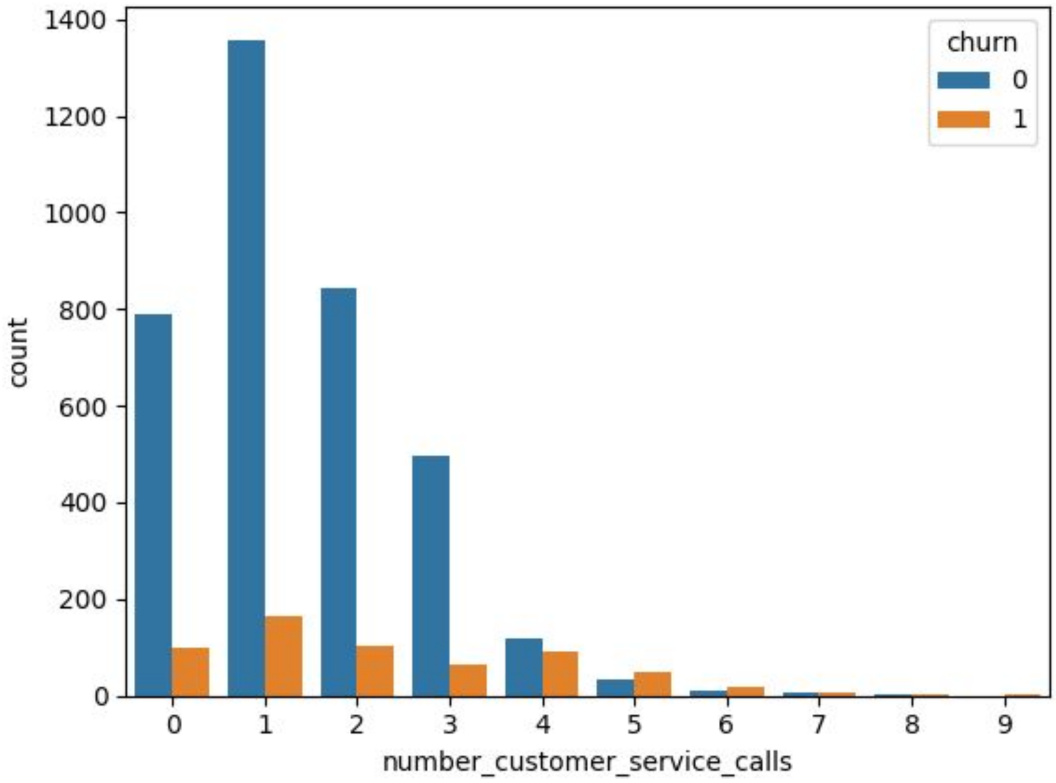
EXPLORATORY DATA ANALYSIS





- The figure shown the impact from normalized and standardized process.

EXPLORATORY DATA ANALYSIS



As shown in figure, we got a decreasing values for 3 number of customer services.

Normalize Data (MinMaxScalar and Standard Scalar)

```
    dfset['total_day_calls'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_day_calls'].values.reshape(len(df), 1))
    dfset['total_eve_calls'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_eve_calls'].values.reshape(len(df), 1))
    dfset['total_night_calls'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_night_calls'].values.reshape(len(df), 1))
    dfset['total_intl_calls'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_calls'].values.reshape(len(df), 1))

    dfset['total_day_charge'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_day_charge'].values.reshape(len(df), 1))
    dfset['total_eve_charge'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_eve_charge'].values.reshape(len(df), 1))
    dfset['total_night_charge'] =MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_night_charge'].values.reshape(len(df), 1))
    dfset['total_intl_charge'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_charge'].values.reshape(len(df), 1))

    dfset['total_day_minutes'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_day_minutes'].values.reshape(len(df), 1))
    dfset['total_eve_minutes'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_eve_minutes'].values.reshape(len(df), 1))
    dfset['total_night_minutes'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_night_minutes'].values.reshape(len(df), 1))
    dfset['total_intl_minutes'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_minutes'].values.reshape(len(df), 1))

    dfset['number_vmail_messages'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_minutes'].values.reshape(len(df), 1))
    dfset['number_customer_service_calls'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_minutes'].values.reshape(len(df), 1))
    dfset['account_length'] = MinMaxScaler(feature_range=(0, 1)).fit_transform(dfset['total_intl_minutes'].values.reshape(len(df), 1))
```

```
[316] dfset['total_day_calls'] = StandardScaler().fit_transform(dfset['total_day_calls'].values.reshape(len(df), 1))
    dfset['total_eve_calls'] = StandardScaler().fit_transform(dfset['total_eve_calls'].values.reshape(len(df), 1))
    dfset['total_night_calls'] = StandardScaler().fit_transform(dfset['total_night_calls'].values.reshape(len(df), 1))
    dfset['total_intl_calls'] = StandardScaler().fit_transform(dfset['total_intl_calls'].values.reshape(len(df), 1))

    dfset['total_day_charge'] = StandardScaler().fit_transform(dfset['total_day_charge'].values.reshape(len(df), 1))
    dfset['total_eve_charge'] = StandardScaler().fit_transform(dfset['total_eve_charge'].values.reshape(len(df), 1))
    dfset['total_night_charge'] = StandardScaler().fit_transform(dfset['total_night_charge'].values.reshape(len(df), 1))
    dfset['total_intl_charge'] = StandardScaler().fit_transform(dfset['total_intl_charge'].values.reshape(len(df), 1))
```

The images shown the process of data normalization and standardization. It needed to fit the data in ideal condition for training process such as, has a same range of scale, and normal distribution in all of the variable

PROPOSE METHODS

```
[ ] model_knn =KNeighborsClassifier(n_neighbors=7, metric='minkowski', p=2)
    model_knn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
logreg.fit(x_training,y_training)
```

```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

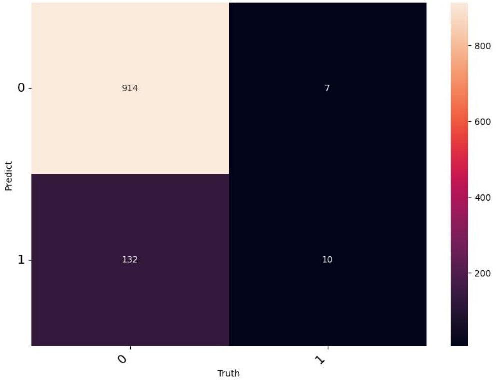
```
[143] from sklearn.ensemble import RandomForestClassifier

      rf = RandomForestClassifier(n_estimators=100, random_state=0)
      rf.fit(x_train, y_train)
```

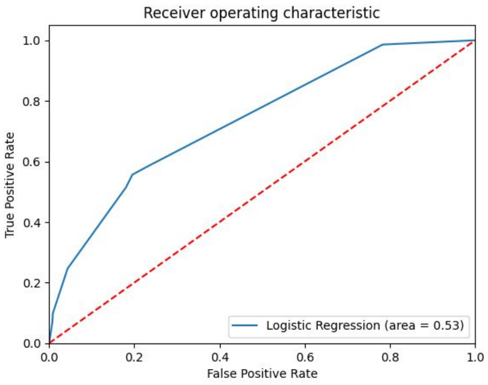
```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

- KNN, Random Forest, and Logistic Regression with Logistic Regression for base model building. All of model will compare to determine the best performance evaluation in supervised learning approach.
- Dataset proportion was set fix on 25% validation set, and 75% of training set.
- Dataset has an indicate as binary classification, proven by the label on churn column (yes/no)

Logistic Regression | Result

```
[167] from sklearn.linear_model import LogisticRegression
      logreg = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                                  intercept_scaling=1, l1_ratio=None, max_iter=100,n_jobs=None, penalty='l2',
                                  random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                                  warm_start=False)
      logreg.fit(x_training,y_training)

                      LogisticRegression
      LogisticRegression(random_state=0, solver='liblinear')

[161] logreg.score(x_training,y_training)

      0.8575462817696894

      y_pred = logreg.predict(x_testing)
      logreg_acc = round(metrics.accuracy_score(y_testing,y_pred)*100,2)
      print("Accuracy",logreg_acc,"%")

      Accuracy 86.92 %
```





- According the result of Logistic Regression, we got 86.92% for accuracy measurement.
- Shown in confusion matrix the dataset potential to be imbalance.

K-Nearest Neighbors | Result

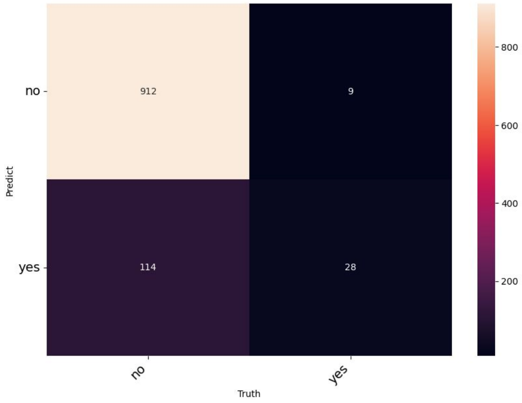
```
from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] model_knn =KNeighborsClassifier(n_neighbors=7, metric='minkowski', p=2)
    model_knn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
[ ] pred_knn =model_knn.predict(x_test)
    knn_acc = round(metrics.accuracy_score(y_test,pred_knn)*100,2)
    print("Accuracy",knn_acc,"%")
```

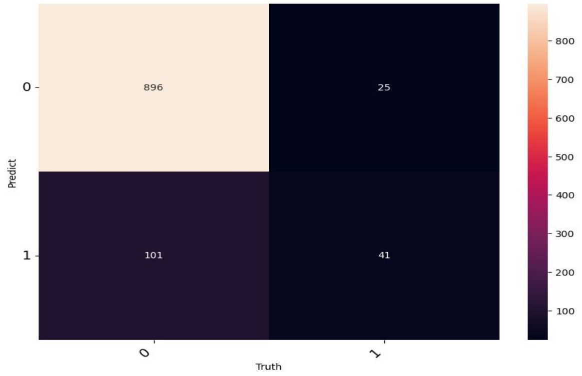
```
Accuracy 88.43 %
```



- According the result obtained from KNN, we got 88.43% of accuracy measurement.
- Shown on the confusion matrix, the performance shown imbalance dataset which mean the label of no shown the more large of number than yes label.

Random Forest | Result

```
[143] from sklearn.ensemble import RandomForestClassifier

      rf = RandomForestClassifier(n_estimators=100, random_state=0)
      rf.fit(x_train, y_train)

        RandomForestClassifier
      RandomForestClassifier(random_state=0)

      rf.score(x_train,y_train)

      1.0

[144] pred_rf =rf.predict(x_test)
      rf_acc = round(metrics.accuracy_score(y_test,pred_rf)*100,2)
      print("Accuracy",rf_acc,"%")

      Accuracy 88.15 %
```



- According the result obtained from Random Forest, we got 88.15% of accuracy measurement.
- Shown on the confusion matrix, the performance shown imbalance dataset which mean the label of no shown the more large of number than yes label.

CONCLUSION

Methods	Accuracy (%)
Logistic Regression	86.92
KNN	88.43
Random Forest	88.15

As shown in table, KNN has a good evaluation of accuracy measurement. Based on this result, KNN has an ability to use for churn prediction.

Other method still has a potential to use as a classification method, especially in supervised learning.