Overview

This is a project that tries to establish the chances that a customer will abandon our services given the various parameters available in our data. It looks at the information available on the usage of our services by customers and from that information, predicts the likelihood that a customer will abandon our services or not. This is important for purposes of establishing the optimal usage of limited resources on retention initatives aimed at our customers for better tailoring of such strategies to ensure intended output.

In [11]:
```
#Loading data
import pandas as pd
data = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
data.head()
```

Out[11]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	cha
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	3	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	3	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	5	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	7	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	3	

5 rows × 21 columns

Business and Data Understanding

Our data was sourced from a telecommunications company and has 20 features as listed below.

- account length
- account length
- international plan
- voice mail plan
- number of voice mail messages
- total day minutes used
- day calls made
- total day charge
- total evening minutes
- total evening calls
- total evening charge
- total night minutes
- total night calls
- total night charge
- total international minutes used
- total international calls made
- total international charge
- number customer service calls made

Our data did not have any missing values nor did it have any errors during loading.

Splitting the data variables into 'X' and 'Y' variables for purposes of availing input to our model.

In [12]:
```
y = data['churn']
X = data.drop(['state', 'account length','area code','phone number','international plan','voice mail plan','churn'], axis=1)
```

We thereafter split our data into training and testing samples based on Scikit Learn model

In [13]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
2666 667 2666 667
```

Modeling

Preparation for Data Modelling

For our first model, we use the Lot Transformation

In [14]:
```
from sklearn.preprocessing import FunctionTransformer
import numpy as np

log_transformer = FunctionTransformer(np.log, validate=True)

log_columns = ['number vmail messages', 'total day minutes', 'total day calls',
       'total day charge', 'total eve minutes', 'total eve calls',
       'total eve charge', 'total night minutes', 'total night calls',
       'total night charge', 'total intl minutes', 'total intl calls',
       'total intl charge', 'customer service calls']


new_log_columns = ['log_nvm', 'log_tdm', 'log_tdc',
       'log_tdg', 'log_tem', 'log_tec',
       'log_teg', 'log_tnm', 'log_tnc',
       'log_tng', 'logtim', 'log_tic',
       'log_tig', 'log_csc']

X_train_log = pd.DataFrame(log_transformer.fit_transform(X_train[log_columns]),
                           columns=new_log_columns, index=X_train.index)

X_train = pd.concat([X_train.drop(log_columns, axis=1), X_train_log], axis=1)
X_train
```

```
C:\Users\kuta\anaconda3\lib\site-packages\sklearn\preprocessing\_function_transformer.py:205: RuntimeWarning: divide by zero enc
ountered in log
  return func(X, **(kw_args if kw_args else {}))
```

Out[14]:

	log_nvm	log_tdm	log_tdc	log_tdg	log_tem	log_tec	log_teg	log_tnm	log_tnc	log_tng	logtim	log_tic	log_tig	log_csc
817	-inf	4.559126	4.521789	2.787477	5.098035	4.143135	2.632608	5.576706	4.770685	2.475698	1.887070	1.791759	0.576613	0.693147
1373	-inf	4.718499	4.653960	2.946542	5.266311	4.700480	2.800933	5.341856	4.532599	2.240710	1.410987	1.386294	0.104360	1.386294
679	-inf	5.404478	4.356709	3.632574	5.789960	4.709530	3.325036	5.337538	4.644391	2.236445	2.163323	2.197225	0.854415	0.000000
56	-inf	4.843399	4.584967	3.071303	5.192957	4.127134	2.727853	4.947340	4.852030	1.846879	2.079442	0.693147	0.770108	0.000000
1993	-inf	5.376666	4.564348	3.604682	5.584623	4.343805	3.119718	5.365976	4.700480	2.264883	1.504077	1.098612	0.198851	-inf
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1095	-inf	5.614587	4.787492	3.842673	5.291293	4.406719	2.826129	5.080161	4.127134	1.979621	1.791759	1.098612	0.482426	0.000000
1130	-inf	3.558201	4.127134	1.786747	5.197391	4.488636	2.732418	5.527841	4.060443	2.426571	2.541602	0.693147	1.232560	0.000000
1294	-inf	4.472781	4.330733	2.700690	5.568345	4.709530	3.103240	5.218191	4.828314	2.117460	2.219203	1.609438	0.908259	0.000000
860	-inf	5.188503	4.709530	3.416414	5.165928	4.867534	2.700690	5.431974	4.521789	2.331173	2.292535	1.791759	0.982078	0.693147
3174	3.7612	3.397858	4.812184	1.625311	4.860587	4.762174	2.395164	5.786591	4.653960	2.685805	2.151762	1.791759	0.841567	0.693147

2666 rows × 14 columns

Target Audience and Dataset Choice

In [15]:
```
X_test_log = pd.DataFrame(log_transformer.transform(X_test[log_columns]),
                          columns=new_log_columns, index=X_test.index)

X_test = pd.concat([X_test.drop(log_columns, axis=1), X_test_log], axis=1)
X_test
```

```
C:\Users\kuta\anaconda3\lib\site-packages\sklearn\preprocessing\_function_transformer.py:205: RuntimeWarning: divide by zero enc
ountered in log
  return func(X, **(kw_args if kw_args else {}))
```

Out[15]:

	log_nvm	log_tdm	log_tdc	log_tdg	log_tem	log_tec	log_teg	log_tnm	log_tnc	log_tng	logtim	log_tic	log_tig	log_csc
438	-inf	5.043425	4.532599	3.271468	5.800909	4.663439	3.335770	5.243861	4.812184	2.142416	2.602690	1.098612	1.294727	0.000000
2674	-inf	4.692265	4.762174	2.920470	5.381739	4.820282	2.916689	5.238567	4.948760	2.137710	2.549445	1.791759	1.241269	-inf
1345	-inf	-inf	-inf	-inf	5.072671	4.867534	2.607861	5.118592	4.477337	2.017566	1.916923	0.000000	0.609766	1.386294
1957	-inf	5.360353	4.369448	3.588506	5.318610	4.510860	2.853593	5.051137	4.727388	1.950187	2.322388	0.693147	1.011601	0.000000
2148	-inf	4.969813	4.624973	3.197856	5.414766	4.290459	2.949688	5.428029	4.510860	2.327278	2.302585	1.945910	0.993252	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2577	-inf	5.220896	4.521789	3.449035	5.361292	4.442651	2.896464	5.278625	4.442651	2.177022	2.140066	1.609438	0.832909	0.693147
2763	2.944439	5.047931	4.644391	3.276012	5.222516	4.770685	2.757475	5.261135	4.753590	2.159869	2.104134	0.693147	0.792993	1.098612
3069	3.258097	5.067016	4.510860	3.295096	5.078294	4.844187	2.613007	5.385870	4.477337	2.284421	2.292535	1.098612	0.982078	0.000000
1468	3.295837	4.766438	4.624973	2.994732	5.331752	4.844187	2.866762	5.269918	4.736198	2.169054	1.435085	1.945910	0.122218	1.098612
582	-inf	5.101085	4.691348	3.328985	5.046002	4.499810	2.580974	5.129307	4.762174	2.028148	2.370244	2.079442	1.061257	0.000000

667 rows × 14 columns

One Hot Encoding

We thereafter improve on it by use of One Hot Encoding as below

In [16]:

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)

cat_columns = [col for col in X.columns if X[col].dtype in [object]]
X_train_cat = X_train.loc[:, cat_columns]

X_train_cat.fillna(value='missing', inplace=True)

X_train_ohe = pd.DataFrame(ohe.fit_transform(X_train_cat),
                           columns=cat_columns, index=X_train.index)
```

In [17]:
```
X_test_cat = pd.DataFrame(ohe.transform(X_test[cat_columns]),
                          columns=cat_columns, index=X_test.index)

X_test_cat.fillna(value='missing', inplace=True)
X_test_cat.drop(X_train_cat == '-inf')

X_test_ohe = pd.DataFrame(ohe.fit_transform(X_test_cat),
                          columns=cat_columns, index=X_test.index)
```

Evaluation

Building, Evaluating, and Validating the Model

In [18]:
```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)

train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test

mse_train = np.sum((y_train - y_hat_train)**2)/len(y_train)
mse_test = np.sum((y_test - y_hat_test)**2)/len(y_test)
print('Train Mean Squared Error:', mse_train)
print('Test Mean Squared Error:', mse_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28000\3259245223.py in <module>
      1 from sklearn.linear_model import LinearRegression
      2 linreg = LinearRegression()
----> 3 linreg.fit(X_train, y_train)
      4
      5 y_hat_train = linreg.predict(X_train)

~\anaconda3\lib\site-packages\sklearn\linear_model\_base.py in fit(self, X, y, sample_weight)
    660         accept_sparse = False if self.positive else ["csr", "csc", "coo"]
    661
--> 662         X, y = self._validate_data(
    663             X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True
    664         )

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, c
opy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, orde
r, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite == "allow-nan")
    801
    802     if ensure_min_samples > 0:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114             raise ValueError(
```

```
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None else X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

In [19]:
```
from sklearn.metrics import mean_squared_error

train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error:', test_mse)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28000\1080736140.py in <module>
      1 from sklearn.metrics import mean_squared_error
      2
----> 3 train_mse = mean_squared_error(y_train, y_hat_train)
      4 test_mse = mean_squared_error(y_test, y_hat_test)
      5 print('Train Mean Squared Error:', train_mse)

NameError: name 'y_hat_train' is not defined
```

In [20]:
```
reg = LinearRegression().fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28000\2141724561.py in <module>
----> 1 reg = LinearRegression().fit(X_train, y_train)

~\anaconda3\lib\site-packages\sklearn\linear_model\_base.py in fit(self, X, y, sample_weight)
    660         accept_sparse = False if self.positive else ["csr", "csc", "coo"]
    661
--> 662         X, y = self._validate_data(
    663             X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True
    664         )

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, c
opy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, orde
r, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite == "allow-nan")
    801
    802     if ensure_min_samples > 0:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None else X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

In [22]:
```
poly = PolynomialFeatures(2)

poly_2 = PolynomialFeatures(4)

reg_poly = LinearRegression().fit(poly.fit_transform(X_train), y_train)

reg_poly_2 = LinearRegression().fit(poly_2.fit_transform(X_train), y_train)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28000\4214741341.py in <module>
----> 1 poly = PolynomialFeatures(2)
      2
      3 poly_2 = PolynomialFeatures(4)
      4
      5 reg_poly = LinearRegression().fit(poly.fit_transform(X_train), y_train)

NameError: name 'PolynomialFeatures' is not defined
```

In [23]:
```
print(f"""
Simple Linear Regression
Train MSE: {mean_squared_error(y_train, reg.predict(X_train))}
Test MSE:  {mean_squared_error(y_test, reg.predict(X_test))}

6th Degree Polynomial
Train MSE: {mean_squared_error(y_train, reg_poly.predict(poly.transform(X_train)))}
Test MSE:  {mean_squared_error(y_test, reg_poly.predict(poly.transform(X_test)))}

2nd Degree Polynomial
Train MSE: {mean_squared_error(y_train, reg_poly_2.predict(poly_2.transform(X_train)))}
Test MSE:  {mean_squared_error(y_test, reg_poly_2.predict(poly_2.transform(X_test)))}
""")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28000\2095095322.py in <module>
      1 print(f"""
      2 Simple Linear Regression
----> 3 Train MSE: {mean_squared_error(y_train, reg.predict(X_train))}
      4 Test MSE:  {mean_squared_error(y_test, reg.predict(X_test))}
      5

NameError: name 'reg' is not defined
```

Our linear regression is as above and based on the results, we utilize onboard K-Fold Cross Validation model.

Fitting and Evaluating the model with K-Fold Cross-Validation

We hereby try and improve the results by using the K-Fold model to better test whether our model can give us better results

In [24]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(linreg, X, y)
```

Out[24]:
```
array([0.08425802, 0.08974608, 0.08309254, 0.11223599, 0.13328332])
```

In [25]:
```
cross_val_score(linreg, X, y, cv=10)
```

Out[25]:
```
array([0.05382462, 0.10802493, 0.11576401, 0.07857363, 0.03844762,
       0.10228752, 0.12342448, 0.09215267, 0.113743  , 0.1567804 ])
```

In [26]:
```
cross_val_score(linreg, X, y, scoring="neg_mean_squared_error")
```

Out[26]:
```
array([-0.1098794 , -0.0908435 , -0.12067452, -0.11606324, -0.11775698])
```

In [27]:
```
from sklearn.metrics import make_scorer

cross_val_score(linreg, X, y, scoring=make_scorer(mean_squared_error))
```

Out[27]:
```
array([0.1098794 , 0.0908435 , 0.12067452, 0.11606324, 0.11775698])
```

In [28]:
```
from sklearn.model_selection import cross_validate

cross_validate(linreg, X, y)["test_score"]
```

Out[28]:
```
array([0.08425802, 0.08974608, 0.08309254, 0.11223599, 0.13328332])
```

In [29]:
```
cross_validate(linreg, X, y)
```

Out[29]:
```
{'fit_time': array([0.00603104, 0.0042491 , 0.01037645, 0.01151133, 0.01099968]),
 'score_time': array([0.0040009 , 0.00421858, 0.00399923, 0.00400066, 0.0034349 ]),
 'test_score': array([0.08425802, 0.08974608, 0.08309254, 0.11223599, 0.13328332])}
```

In [30]:
```
cross_validate(linreg, X, y, cv=10)
```

Out[30]:
```
{'fit_time': array([0.00744247, 0.00708771, 0.0055213 , 0.00502491, 0.0049181 ,
        0.0049932 , 0.00499797, 0.00600028, 0.00499892, 0.00399995]),
 'score_time': array([0.00242305, 0.00399566, 0.00307393, 0.00293207, 0.00308204,
        0.00299954, 0.00392723, 0.00300479, 0.00300002, 0.00399756]),
 'test_score': array([0.05382462, 0.10802493, 0.11576401, 0.07857363, 0.03844762,
        0.10228752, 0.12342448, 0.09215267, 0.113743  , 0.1567804 ])}
```

In [31]:
```
cross_validate(linreg, X, y, scoring=["r2", "neg_mean_squared_error"])
```

Out[31]:
```
{'fit_time': array([0.00651908, 0.00504994, 0.00536418, 0.00548911, 0.00434732]),
 'score_time': array([0.00446677, 0.00303125, 0.00409245, 0.00344205, 0.00328374]),
 'test_r2': array([0.08425802, 0.08974608, 0.08309254, 0.11223599, 0.13328332]),
 'test_neg_mean_squared_error': array([-0.1098794 , -0.0908435 , -0.12067452, -0.11606324, -0.11775698])}
```

In [32]:
```
cross_validate(linreg, X, y, return_train_score=True)
```

Out[32]:
```
{'fit_time': array([0.00618577, 0.00699449, 0.00613594, 0.00499439, 0.00499988]),
 'score_time': array([0.00303984, 0.00300002, 0.0030148 , 0.0020628 , 0.00399971]),
 'test_score': array([0.08425802, 0.08974608, 0.08309254, 0.11223599, 0.13328332]),
 'train_score': array([0.12004413, 0.11494275, 0.12043026, 0.11234461, 0.10577745])}
```

Conclusion

Our data is not able to give us intended results based on our initial model and our improved model. This does not however reflect the inadequacy of the library. Instead, it only indicates that we need to try other models to stand a better chance of giving us a better prediction of a customer dropping our services arising from their behavious as demonstrated by available data of activities such customers potray.