In [17]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from scipy.stats import uniform
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
```

executed in 6ms, finished 16:30:43 2023-04-24

1 Areas of improvement

> - Provide a better way to record the results. With IPython the focus is on the small. The fix will be to add functions to make easier to read and write eh dataframes that show the results
> - Clean up the graphs. Instead of using just bar charts used stacked bar charts.
> - Write the test routine and create a separate file with a Columns and a pipeline. I did not include since I started looking at the times in a granualr way

In [18]:
```python
# Removed the warnings to make the presentation cleaner
import warnings
warnings.filterwarnings('ignore')
```

executed in 3ms, finished 16:30:43 2023-04-24

2 Recording times

The purpoe of this section is to reocrd the accuracy of the preprpocesing and parameters to understand which algorithm should be used and the parameters.

There are two types of test

> - The first test will test the basic workflow including normalizaiton and running the algorithm.

> - The secondt est will the parameters of the algorihm and use the best results from the algorithm.

In [19]:

```
# A list of the machine learning algorithms
algorithm_list = [ "Description", "SVC", "GAUSSIAN", "K Nearest Neighbor", "Decision Tree"

# The name of the file for normalziation
file="data.csv"

# A class that allows to store the parameter need for the
class Test_Data:

    def __init__(self, description, balanced, normalization, reverse_normalization, \
                one_hot_encoding, algorithm, parameter_grid):
        self.description = description
        self.balanced = balanced
        self.normalization = normalization
        self.reverse_normalization = reverse_normalization
        self.one_hot_encoding = one_hot_encoding
        self.algorithm = algorithm
        self.parameter_grid = parameter_grid

    def get_description(self):
        return self.description

    def get_balanced(self):
        return self.balanced

    def get_do_normalization(self):
        return self.normalization

    def get_reverse_normalization(self):
        return self.reverse_normalization

    def get_one_hot_encoding(self):
        return self.one_hot_encoding

    def get_algorithm(self):
        return self.algorithm

    def get_parameter_grid(self):
        return self.parameter_grid

#
# Different permutations to get the accuracy of different preprocessing methods.
#

test1 = Test_Data("Initial Test with Integer encoding ", \
                   balanced=False, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=False, algorithm=None, pa

test2 = Test_Data("Initial Test with One Hot Encoding", \
                   balanced=False, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=True, algorithm=None, par

test3 = Test_Data("Balanced the Data with Integer encoding", \
                   balanced=True, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=False, algorithm=None, pa

test4 = Test_Data("Balanced the Data with One Hot Encoding", \
                   balanced=True, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=True, algorithm=None, par

test5 = Test_Data("Balanced, Normalization introduced with Integer Encoding", \
                   balanced=True, normalization=True, \
                   reverse_normalization=False, one_hot_encoding=False, algorithm=None, pa

test6 = Test_Data("Balanced, Normalization introduced with One Hot Encoding", \
                   balanced=True, normalization=True, \
                   reverse_normalization=False, one_hot_encoding=True, algorithm=None, par

test7 = Test_Data("Balanced Normalization Reversed with Integer Encoding", \
                   balanced=True, normalization=False, \
```

```
                   reverse_normalization=True, one_hot_encoding=False, algorithm=None, par

test8 = Test_Data("Balanced Normalization Reversed with One Hot Encoding", \
                   balanced=True, normalization=False, \
                   reverse_normalization=True, one_hot_encoding=True, algorithm=None, para

test9 = Test_Data("No Balanced Normalization Introduced With Integer Encoding", \
                  balanced=False, normalization=True, \
                  reverse_normalization=False, one_hot_encoding=False, algorithm=None,para

test10 = Test_Data("No Balanced Normalization Introduced With One Hot Encoding", \
                  balanced=False, normalization=True, \
                  reverse_normalization=False, one_hot_encoding=True, algorithm=None, para

test11 = Test_Data("No Balanced Normalization Switced ( ex. MinMax used instead of scalar)
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=False, algorithm=None, para

test12 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm=None, param

# When running these test these two variables need to be uncommented
# Also, once you get the dasta to show the runs copy to firstRun.csv
# Each time you want a new test modify remove_file and current_test.
# When running the test for parameters comment out remove_file_2 and
# current_test

# Remove the file and allow the user to start a new set of runs.
#remove_file = False

# The test to run.  Each test is an instantiation of Test_Data
#current_test = test12

```

executed in 16ms, finished 16:30:43 2023-04-24

3 Results of studying Accuracy

For my runs and since this is an classifiction problem I have focused on accuracy as my main heuristic fro success. The first results focused on preprocessing and running the alogirhtms. Some the preprocess ideas used were: balancing, Normalizatoin (scalar, Minmax), and Catecgorical Encoding (Integer Encoding, One Hot Encoding)

The second set of test will focus on the parameters

My goal is to have accuracy of 95% or over and I have selected the top 2 scores from the table blow Encoding

> K Nearest Neighbor with No Precessor except Integer Encoding run 0
> Decision Tree with run 0

In [20]:
```
pd.set_option('display.max_colwidth', None)


dataFrame = pd.read_csv("firstRun.csv")
dataFrame.index = algorithm_list

print("Descriptions of the test executed")
descriptions = dataFrame.iloc[0,:]
for index, description in enumerate(descriptions):
    print(index, "  -- ",description)

dataFrame = dataFrame[1:]
dataFrame
```

executed in 18ms, finished 16:30:43 2023-04-24

```
Descriptions of the test executed
0   --  Initial Test with Integer encoding
1   --  Initial Test with One Hot Encoding
2   --  Balanced the Data with Integer encoding
3   --  Balanced the Data with One Hot Encoding
4   --  Balanced, Normalization introduced with Integer Encoding
5   --  Balanced, Normalization introduced with One Hot Encoding
6   --  Balanced Normalization Reversed with Integer Encoding
7   --  Balanced Normalization Reversed with One Hot Encoding
8   --  No Balanced Normalization Introduced With Integer Encoding
9   --  No Balanced Normalization Introduced With One Hot Encoding
10   --  No Balanced Normalization Switced ( ex. MinMax used instead of scalar)
11   --  No Balanced Normalization Introduced ( ex. MinMax used instead of scalar)
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	10	11
SVC	0.88	0.88	0.68	0.57	0.78	0.77	0.68	0.57	0.88	0.87	0.88	0.88
GAUSSIAN	0.87	0.6	0.8	0.62	0.8	0.62	0.8	0.62	0.87	0.6	0.87	0.6
K Nearest Neighbor	0.89	0.89	0.67	0.66	0.79	0.69	0.67	0.66	0.89	0.88	0.89	0.89
Decision Tree	0.92	0.93	0.79	0.79	0.79	0.78	0.77	0.78	0.93	0.93	0.92	0.93
Logistic	0.87	0.87	0.68	0.7	0.78	0.78	0.68	0.7	0.88	0.87	0.87	0.87

In [21]:

```
# Run these test when you are analyzing the parameters for a machine
# algorithm.  Once satisified then copy the file paramters to
# seconds.cs

# Delete the file and start a new
remove_file_2 = False
file2 = "parameters.csv"

param_grid_1 = dict(n_neighbors=[i for i in range(1,20,2) ] )
param_grid_2 = dict(n_neighbors=[i for i in range(21,40,2) ] )
param_grid_3 = dict(max_depth=[i for i in range(1,30)])
param_grid_4 = dict(max_depth=[i for i in range(1,30)],\
                    min_samples_split=[i for i in range(1,19)])
param_grid_5 = dict(max_depth=[i for i in range(1,30)],\
                    min_samples_split=[i for i in range(1,19)])
param_grid_6 = dict(max_depth=[i for i in range(1,30)],\
                    min_samples_split=[i for i in range(1,19)],
                    min_samples_leaf=[i for i in range(1,19)])



test13 = Test_Data("Initial Test with One Hot Encoding", \
                   balanced=False, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=True,
                   algorithm="KNN", parameter_grid=param_grid_1)
test14 = Test_Data("Initial Test with One Hot Encoding", \
                   balanced=False, normalization=False, \
                   reverse_normalization=False, one_hot_encoding=True,
                   algorithm="KNN", parameter_grid=param_grid_2)
test15 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="KNN", para
test16 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="KNN", para
test17 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="DTREE", pa
test18 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="DTREE", pa
test19 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="DTREE", pa
test20 = Test_Data("No Balanced Normalization Introduced ( ex. MinMax used instead of scal
                  balanced=False, normalization=False, \
                  reverse_normalization=True, one_hot_encoding=True, algorithm="DTREE", pa
current_test = test20
```

executed in 13ms, finished 16:30:43 2023-04-24

In [64]:

```
# Show all the parameters
parameters = pd.read_csv('secondRun.csv')
parameters
```

executed in 13ms, finished 17:20:37 2023-04-24

Out[64]:

	Description	Algorithm	Parameters Tried	Best Parameters	Accuracy
0	Initial Test with One Hot Encoding	KNN	{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}	{'n_neighbors': 7}	0.90
1	Initial Test with One Hot Encoding	KNN	{'n_neighbors': [21, 23, 25, 27, 29, 31, 33, 35, 37, 39]}	{'n_neighbors': 27}	0.89
2	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	KNN	{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}	{'n_neighbors': 7}	0.90
3	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	KNN	{'n_neighbors': [21, 23, 25, 27, 29, 31, 33, 35, 37, 39]}	{'n_neighbors': 27}	0.89
4	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]}	{'max_depth': 6}	0.95
5	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 6, 'min_samples_split': 17}	0.95
6	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 6, 'min_samples_split': 17}	0.95
7	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 7, 'min_samples_leaf': 9, 'min_samples_split': 4}	0.95

In [82]:

```
yPredict
```

executed in 5ms, finished 18:22:38 2023-04-24

Out[82]: 0.9419419419419419

In [85]:

```
# Test 18 was choosen since it has the highest accuracy and the least
# Number of hyperparameters
decision_tree_classifier = DecisionTreeClassifier(max_depth=8)
decision_tree_classifier = decision_tree_classifier.fit(xtrain,yTrain)
y_predict = decision_tree_classifier.score(XTest,yTest)
print("The best score is ",y_predict)

```

executed in 36ms, finished 18:28:08 2023-04-24

```
The best score is  0.9409409409409409
```

4 Resusable Functions

In [22]:

```

def info_about_columns(dataframe, data_science_descriptions):
    '''
        A reusable function that will create a dataframe to contain in another
        dataframe the following : dataypes, Number of Unique Categories, Categories
        per sample and the type of variable missing values and missing values %

        input : A dataframe where data and categories will be retrieved
        series : The data science explamation for each data type
    '''

    if data_science_descriptions == None:
        data_science_descriptions = dataframe.copy().dtypes
        data_science_descriptions = \
            data_science_descriptions.replace(data_science_descriptions.to_list(), "NA")

    dataframe_info_about_columns = pd.concat([
          dataframe.dtypes,
          dataframe.nunique(),
          round(dataframe.nunique()*100/len(dataframe)),
          data_science_descriptions,
          dataframe.isna().sum(),
          dataframe.isna().sum() * 100 / len(dataframe)], axis=1)


    dataframe_info_about_columns.columns=[
                                     'DataType',
                                     '# of Categories',
                                     'categories/sample ratio',
                                     'Data Science Type',
                                     'missing values',
                                     'missing values %']

#     dataframe_info_about_columns = pd.DataFrame()
#     dataframe_info_about_columns['DataTypes'] = dataframe.dtypes
#     dataframe_info_about_columns['# of Categories'] = dataframe.nunique()
#     dataframe_info_about_columns['Data Science Description'] = data_science_descriptions
#     dataframe_info_about_columns['categories/sample ratio'] =  round(dataframe.nunique()
#     dataframe_info_about_columns['missing values'] = dataframe.isna().sum()
#     dataframe_info_about_columns['missing values %'] = dataframe.isna().sum() * 100 / le

    return dataframe_info_about_columns
```

executed in 7ms, finished 16:30:43 2023-04-24

In [23]:
```
from IPython.display import display, HTML

def show_examples_of_data(dataframe, data_information, description, catgeogry_cutoff):
    '''
       purpose To show the data to provide the data scientist an understand of the data

       input:
          dataframe          The data frame that contains the dataset
          data_information   Information about the categorical, missing values etc..
          descriptons        A series that contains a description for each filename
    '''

    data_dictionary = pd.DataFrame(columns=["Field", "Description", "Value"])

    for index, row in data_information.iterrows():

        values = ""
        column_cnt = data_information.loc[index, '# of Categories']
        if column_cnt <= catgeogry_cutoff:
               value = original_df[index].unique()
        else:
               value = str(original_df[index].min()) + " to "  + str(original_df[index].ma
        row_data = []
        row_data.append(index)
        row_data.append(description.loc[index])
        row_data.append(value)

        data_dictionary.loc[len(data_dictionary.index)] = row_data

    return data_dictionary

```

executed in 7ms, finished 16:30:43 2023-04-24

In [24]:
```
def create_dataframe_with_continuous(dataframe):
    '''
        Create a continous dataframe
    '''
    continous_df = dataframe.select_dtypes(include=np.number)
    continous_df = dataframe.drop(['area_code', 'churned', 'phone_number', 'state'], axis=
    return continous_df
```

executed in 4ms, finished 16:30:43 2023-04-24

5 Load Data

In [25]:
```
# Load the data provided.
original_df = pd.read_csv('churn.csv')
original_df.shape
```

executed in 22ms, finished 16:30:43 2023-04-24

Out[25]: (5000, 21)

In [26]:

```
# Descriptive information that seems to describe the columns.
# This is assumption, but the number of rows look correct and contain the same information
# The link where I got the information was https://www.kaggle.com/c/customer-churn-predict
column_descriptions = pd.read_csv('info.txt', header=None)
column_descriptions.columns = ['Field', 'Data Type', 'Description']
column_descriptions = column_descriptions.set_index('Field').reindex().sort_index()
column_descriptions
```

executed in 13ms, finished 16:30:43 2023-04-24

Out[26]:

Field	Data Type	Description
account_length	numerical	Number of months the customer has been with the current telco provider
area_code	string	"area_code_AAA" where AAA = 3 digit area code
churned	(yes/no)	Customer churn - target variable
intl_plan	(yes/no)	The customer has international plan
number_customer_service_calls	numerical	Number of calls to customer service
number_vmail_messages	numerical	Number of voice-mail messages
phone_number	string	The last 7 digits of the phone number
state	string	2-letter code of the US state of customer residence
total_day_calls	numerical	Total minutes of day calls
total_day_charge	numerical	Total charge of day calls
total_day_minutes	numerical	Total minutes of day calls
total_eve_calls	numerical	Total number of evening calls
total_eve_charge	numerical	Total charge of evening calls
total_eve_minutes	numerical	Total minutes of evening calls
total_intl_calls	numerical	Total number of international calls
total_intl_charge	numerical	Total charge of international calls
total_intl_minutes	numerical	Total minutes of international calls
total_night_calls	numerical	Total number of night calls
total_night_charge	numerical	Total charge of night calls
total_night_minutes	numerical	Total minutes of night calls
voice_mail_plan	(yes/no)	The customer has voice mail plan

5.1 Summarize Data Part 1 -- Overall View of Data

In [27]:

```
original_df.head(2)
```

executed in 20ms, finished 16:30:43 2023-04-24

Out[27]:

	state	account_length	area_code	phone_number	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	tot
0	KS	128	415	382-4657	no	yes	25	265.1	
1	OH	107	415	371-7191	no	yes	26	161.6	

2 rows × 21 columns

In [28]:

```
# Get the number of rows which will help to evaluate how missing values are handled, the m
# algorithm and other parts of the analysis.
print("The shape of the dataframe is ", original_df.shape)
```

executed in 4ms, finished 16:30:43 2023-04-24

```
The shape of the dataframe is  (5000, 21)
```

5.2 Univariate Data

In [29]:

```
# Basic Information about missing values, categories and datatype
initial_examination = info_about_columns(original_df, None).sort_index()
initial_examination
```

executed in 45ms, finished 16:30:43 2023-04-24

Out[29]:

	DataType	# of Categories	categories/sample ratio	Data Science Type	missing values	missing values %
account_length	int64	218	4.0	NA	0	0.00
area_code	int64	3	0.0	NA	0	0.00
churned	object	2	0.0	NA	0	0.00
intl_plan	object	2	0.0	NA	0	0.00
number_customer_service_calls	int64	10	0.0	NA	0	0.00
number_vmail_messages	int64	48	1.0	NA	0	0.00
phone_number	object	5000	100.0	NA	0	0.00
state	object	51	1.0	NA	0	0.00
total_day_calls	int64	123	2.0	NA	0	0.00
total_day_charge	float64	1961	39.0	NA	0	0.00
total_day_minutes	float64	1961	39.0	NA	0	0.00
total_eve_calls	int64	126	3.0	NA	0	0.00
total_eve_charge	object	1660	33.0	NA	0	0.00
total_eve_minutes	float64	1879	38.0	NA	0	0.00
total_intl_calls	int64	21	0.0	NA	0	0.00
total_intl_charge	float64	170	3.0	NA	1	0.02
total_intl_minutes	float64	170	3.0	NA	0	0.00
total_night_calls	int64	131	3.0	NA	0	0.00
total_night_charge	float64	1028	21.0	NA	0	0.00
total_night_minutes	float64	1853	37.0	NA	0	0.00
voice_mail_plan	object	2	0.0	NA	0	0.00

In [30]:

```
### Summarize Data Part 1 -- Contents of the Columns
```

executed in 2ms, finished 16:30:43 2023-04-24

In [31]:
```
# Understand the meaning of the data by linking the vlaue to the
# description and field name
display(HTML(show_examples_of_data(original_df, initial_examination, column_descriptions.De
```
executed in 52ms, finished 16:30:43 2023-04-24

	Field	Description	Value
0	account_length	Number of months the customer has been with the current telco provider	1 to 243
1	area_code	"area_code_AAA" where AAA = 3 digit area code	[415, 408, 510]
2	churned	Customer churn - target variable	[False., True.]
3	intl_plan	The customer has international plan	[no, yes]
4	number_customer_service_calls	Number of calls to customer service	[1, 0, 2, 3, 4, 5, 7, 9, 6, 8]
5	number_vmail_messages	Number of voice-mail messages	[25, 26, 0, 24, 37, 27, 33, 39, 30, 41, 28, 34, 46, 29, 35, 21, 32, 42, 36, 22, 23, 43, 31, 38, 40, 48, 18, 17, 45, 16, 20, 14, 19, 51, 15, 11, 12, 47, 8, 44, 49, 4, 10, 13, 50, 9, 6, 52]
6	phone_number	The last 7 digits of the phone number	327-1058 to 422-9964
7	state	2-letter code of the US state of customer residence	[KS, OH, NJ, OK, AL, MA, MO, LA, WV, IN, RI, IA, MT, NY, ID, VT, VA, TX, FL, CO, 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, ME, MS, TN, PA, CT, ND]
8	total_day_calls	Total minutes of day calls	0 to 165
9	total_day_charge	Total charge of day calls	0.0 to 59.76
10	total_day_minutes	Total minutes of day calls	0.0 to 351.5
11	total_eve_calls	Total number of evening calls	0 to 170
12	total_eve_charge	Total charge of evening calls	0.0 to ?
13	total_eve_minutes	Total minutes of evening calls	0.0 to 363.7
14	total_intl_calls	Total number of international calls	[3, 5, 7, 6, 4, 2, 9, 19, 1, 10, 15, 8, 11, 0, 12, 13, 18, 14, 16, 20, 17]
15	total_intl_charge	Total charge of international calls	0.0 to 5.4
16	total_intl_minutes	Total minutes of international calls	0.0 to 20.0
17	total_night_calls	Total number of night calls	0 to 175
18	total_night_charge	Total charge of night calls	0.0 to 17.77
19	total_night_minutes	Total minutes of night calls	0.0 to 395.0
20	voice_mail_plan	The customer has voice mail plan	[yes, no]

6 Data Cleanup -- Remove all missing values

- total_eve_charge has question marks and numbers
- total_intl_charge has one field that is not a number

Analysis

- The total dataset is 5000 observations so 6 observations will only lose .0001
- It is know that easy state and area code has at least 3 observations so we are not losing much.

In [32]:

```
# Convert a number to a float
def isFloat(num):
    try:
        float(num)
        return True
    except ValueError:
        return False

# Shows the missing data so we can analyze the data so we can understand what is being los
rows_with_missing_data = original_df.query('total_eve_charge == "?" | total_intl_charge.is
rows_with_missing_data

```

executed in 27ms, finished 16:30:43 2023-04-24

Out[32]:

	state	account_length	area_code	phone_number	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	tc
1	OH	107	415	371-7191	no	yes	26	161.6	
5	AL	118	510	391-8027	yes	no	0	223.4	
9	WV	141	415	330-8173	yes	yes	37	258.6	
20	FL	147	415	396-5800	no	no	0	155.1	
34	OK	57	408	395-2854	no	yes	25	176.8	
50	IA	52	408	413-4957	no	no	0	191.9	

6 rows × 21 columns

In [33]:

```
# Problem : The ? in the total_eve_charge is causing issues and so I since it was only fiv
# rows per state and one to rwows per area code.  The rows will be remmoved from the data
revision_1 = original_df.copy(deep=True)
revision_1 = revision_1.drop(rows_with_missing_data.index.to_list())

# Drop the na in total_intl_charge
revision_1 = revision_1.dropna(axis=1)
revision_1.shape

# revision_1 = revision_1.drop(index=bad_rows_in_eve_charge.index.to_list())
# revision_1 = revision_1.astype({"total_eve_charge":np.float64}, errors='ignore')
# revision_1 = revision_1.reset_index(drop=True)
# revision_1_continuous = create_dataframe_with_continuous(revision_1)
# revision_1_continuous
```

executed in 14ms, finished 16:30:43 2023-04-24

Out[33]: (4994, 21)

In [34]:

```
# Fix the types of each columns that is incorrect and massage the data
revision_2 = revision_1.copy(deep=True)

revision_2["intl_plan"] = np.where(revision_2["intl_plan"] == " yes", True, False)
revision_2["voice_mail_plan"] = np.where(revision_2["voice_mail_plan"] == " yes", True, Fa
revision_2["churned"] = np.where(revision_2["churned"] == " True.", True, False)

revision_2.phone_number = revision_2.phone_number.astype(dtype='string', copy=True)
revision_2.area_code = revision_2.area_code.astype(dtype="string", copy=True)
revision_2.state = revision_2.state.astype(dtype="string", copy=True)
revision_2.total_eve_charge = revision_2.total_eve_charge.astype(float, copy=True)
revision_2
```

executed in 47ms, finished 16:30:43 2023-04-24

Out[34]:

	state	account_length	area_code	phone_number	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
0	KS	128	415	382-4657	False	True	25	265.1
2	NJ	137	415	358-1921	False	False	0	243.4
3	OH	84	408	375-9999	True	False	0	299.4
4	OK	75	415	330-6626	True	False	0	166.7
6	MA	121	510	355-9993	False	True	24	218.2
...	...	...	...	...	...	...	...	...
4995	HI	50	408	365-8751	False	True	40	235.7
4996	WV	152	415	334-9736	False	False	0	184.2
4997	DC	61	415	333-6861	False	False	0	140.6
4998	DC	109	510	394-2206	False	False	0	188.8
4999	VT	86	415	373-8058	False	True	34	129.4

4994 rows × 21 columns

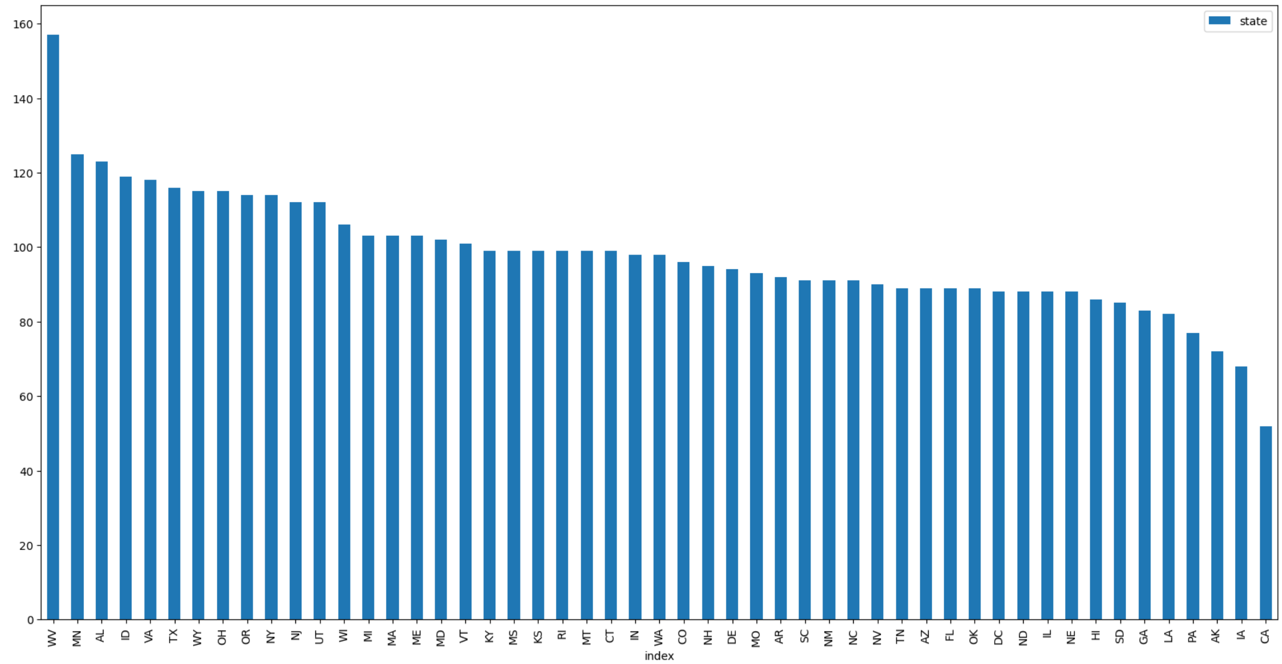
In [35]:

```
### Exaimine The Sring Columns -- State, phoneNumber
```

executed in 3ms, finished 16:30:43 2023-04-24

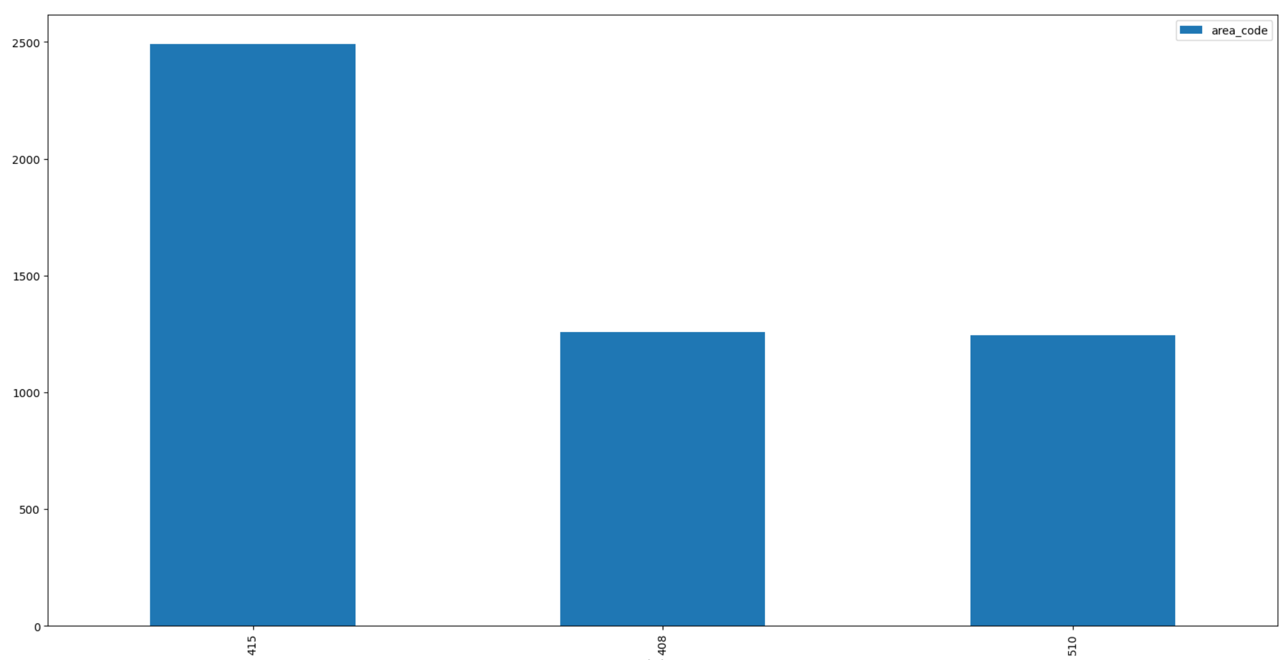
In [36]:
```
# Number of current and past customers
# No actionable results were found.
pd.DataFrame(revision_2.state.value_counts()).reset_index().plot( x='index', y='state', ki
```
executed in 696ms, finished 16:30:44 2023-04-24

Out[36]: <AxesSubplot:xlabel='index'>



In [37]:
```
# Number of people by area code
# Here I learnd that they are looking at three different area code
# and these are not the area of help centers, but of poeople's phone
# number since the column phone number has no area code.
pd.DataFrame(revision_2.area_code.value_counts()).reset_index().plot( x='index', y='area_co
```
executed in 181ms, finished 16:30:44 2023-04-24

Out[37]: <AxesSubplot:xlabel='index'>



In [38]:

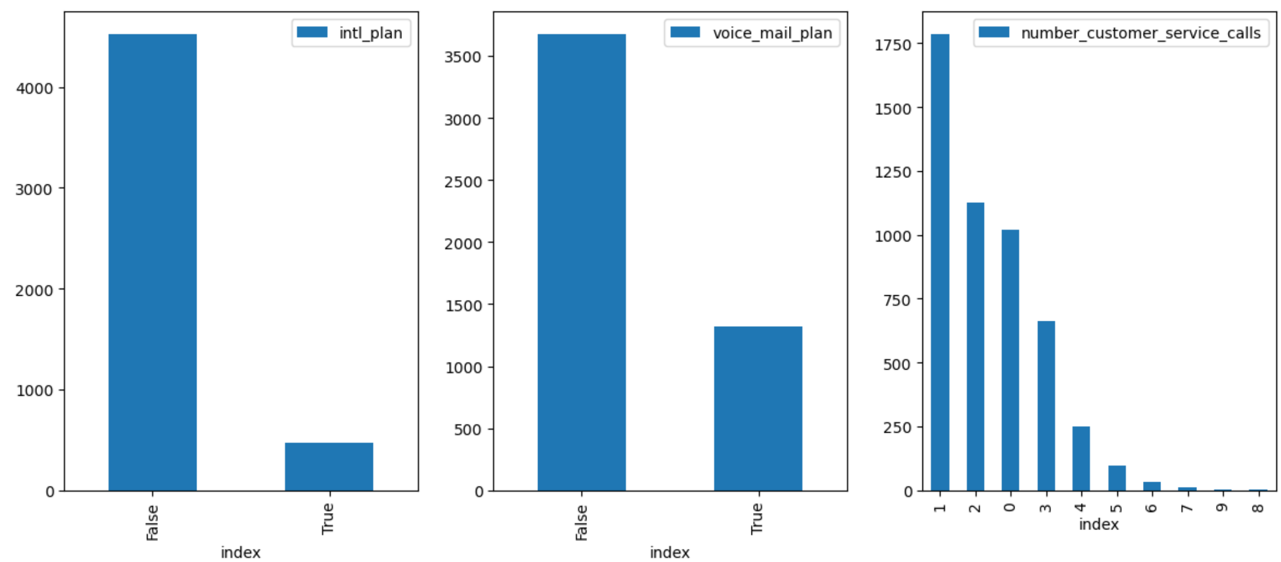
```
# Take a look at the plan by looking at the different parts such
# as the people on the international plan and voice mail plan
# and number of customer services.

# The numbers seem ok and the number of service calls looks reasonable.
# However, I would do more of an in depth analysis of the customer
# service calls to make sure that people are satisified.
categories = ['intl_plan','voice_mail_plan','number_customer_service_calls']
figure, axis = plt.subplots(1,3, figsize=(10,5))
figure.tight_layout()
axis_ravel = np.ravel(axis)
pd.DataFrame(revision_2.intl_plan.value_counts()).reset_index().plot( x='index', y='intl_pl
                                                                      figsize=(8,5), ax=axis
pd.DataFrame(revision_2.voice_mail_plan.value_counts()).reset_index().plot( x='index', y='v
                                                                      figsize=(8,5), ax=axis
pd.DataFrame(revision_2.number_customer_service_calls.value_counts()).reset_index().plot( x
                                                                      figsize=(12,5), ax=ax
plt.plot()

```

executed in 414ms, finished 16:30:45 2023-04-24

Out[38]: []

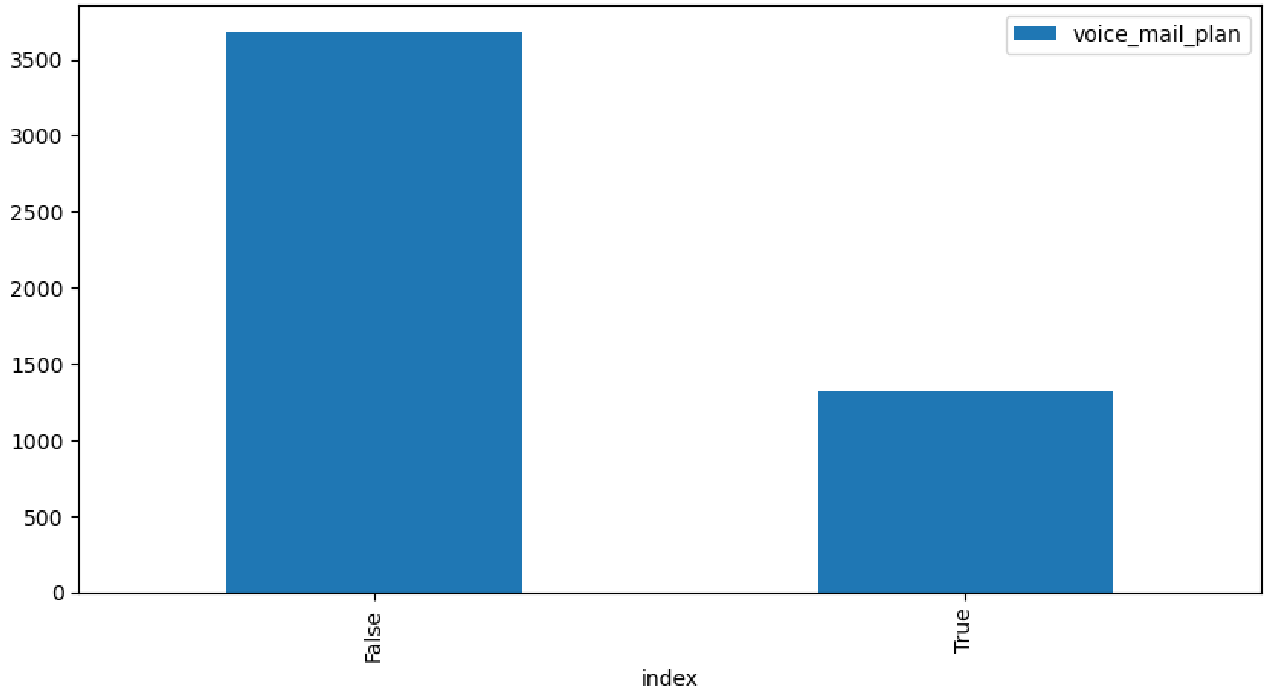


In [39]:

```
# Show a bar chart of the specific plan for the voice mail plan.
categories = ['number_customer_service_calls']
pd.DataFrame(revision_2.voice_mail_plan.value_counts()).reset_index().plot( x='index', y='
                                                                            figsize=(10,5))
plt.plot()
```

executed in 141ms, finished 16:30:45 2023-04-24

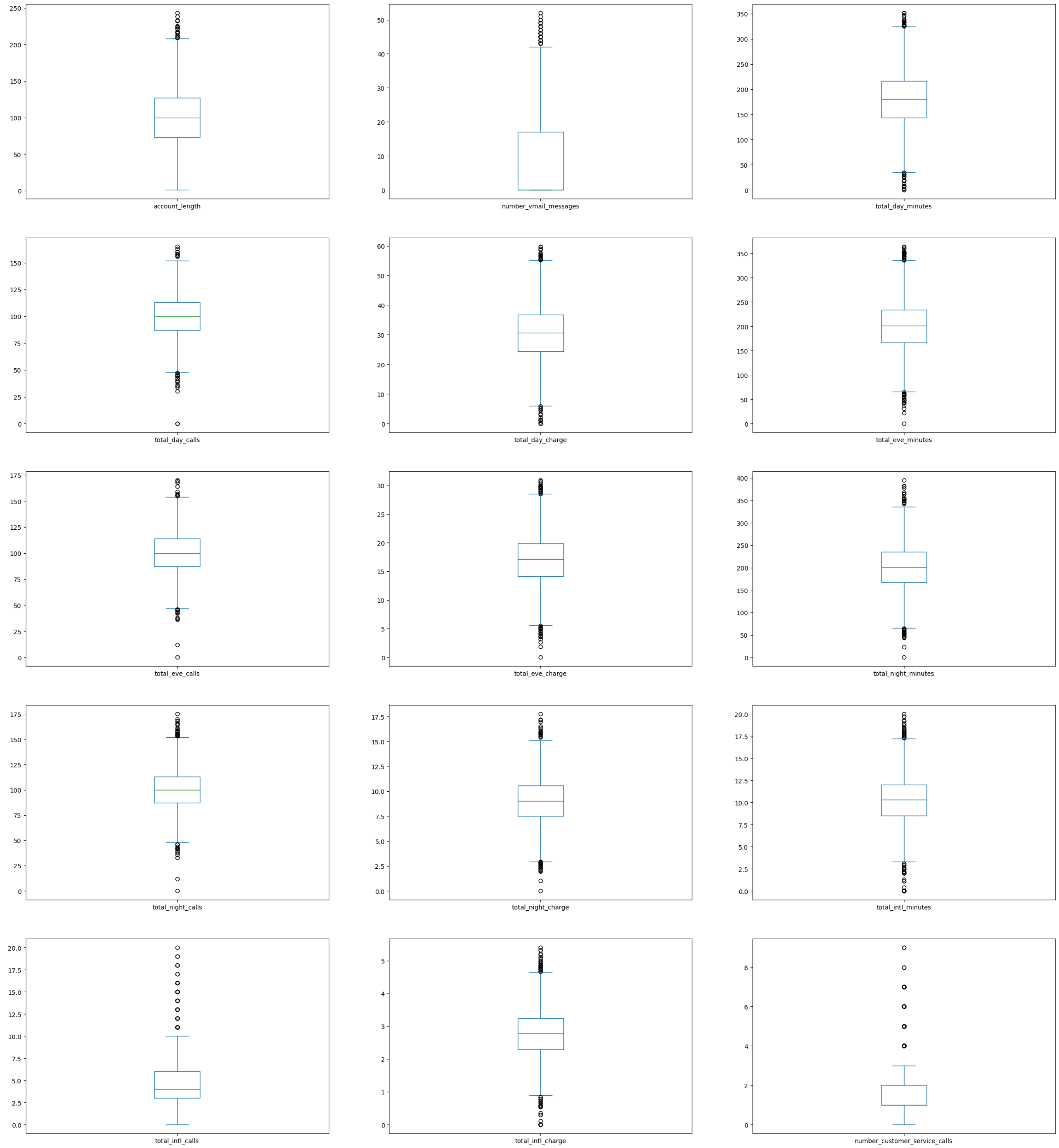
Out[39]: []



In [40]:

```
# Take a look at the outliers.  Later on a deicision will be made if
# they should be used.
temporary_dataframe = revision_2.select_dtypes(include=[np.number])
temporary_dataframe.drop(labels=['number_customer_service_calls', 'number_vmail_messages']
temporary_dataframe.plot(kind='box', subplots=True, layout=(6,3), figsize=(30,40))
plt.show()
```

executed in 1.72s, finished 16:30:46 2023-04-24



In [41]:

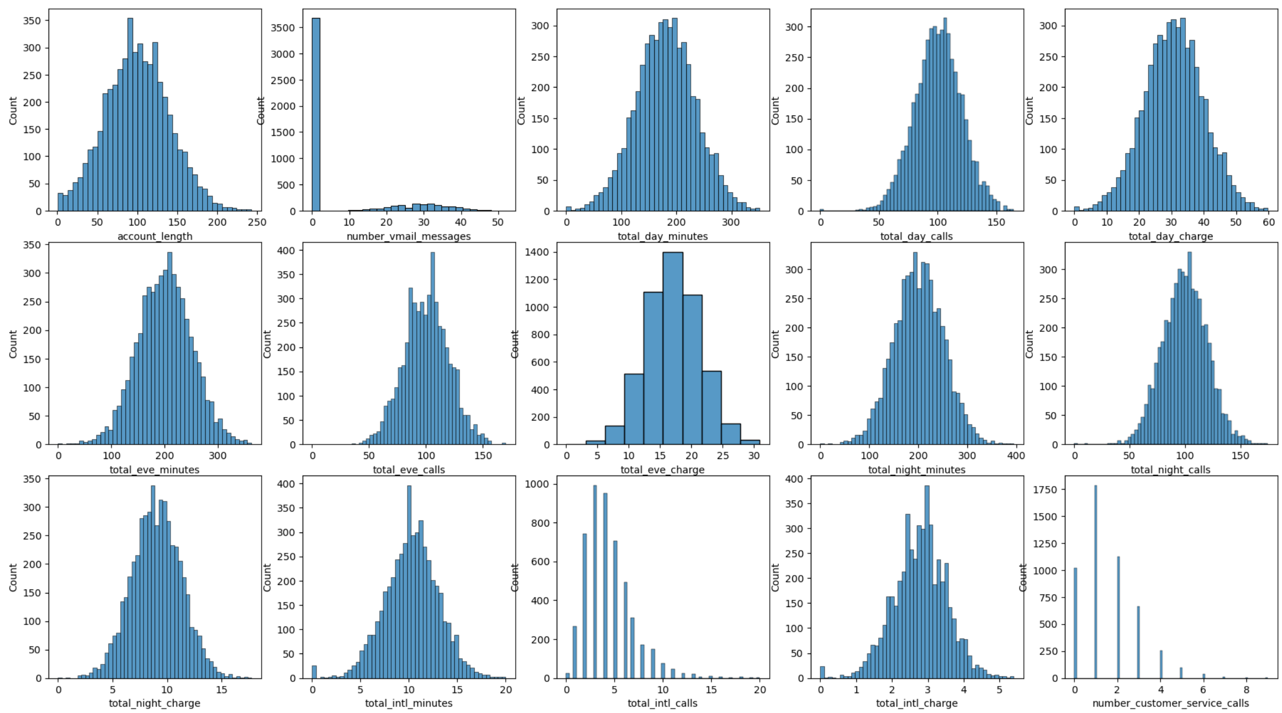
```
# show the Distributins for all the continous fields except
# address code
fig, ax = plt.subplots(3,5,figsize=(18,10))
fig.tight_layout()
ax_reval = np.ravel(ax)
for index, ax in enumerate(ax_reval):
    if (index > 14):
        ax.set_visible(False)
        continue

    if temporary_dataframe.columns[index] == 'total_eve_charge':
          print("Made it here")
          sns.histplot(data=temporary_dataframe, x=temporary_dataframe.columns[index], a
    else:
          sns.histplot(data=temporary_dataframe, x=temporary_dataframe.columns[index], a

plt.show()
```

executed in 3.29s, finished 16:30:50 2023-04-24

```
Made it here
```



In [42]:

```
# Shows the dataframe after cleaning the dta
revision_2
```

executed in 36ms, finished 16:30:50 2023-04-24

Out[42]:

	state	account_length	area_code	phone_number	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
0	KS	128	415	382-4657	False	True	25	265.1
2	NJ	137	415	358-1921	False	False	0	243.4
3	OH	84	408	375-9999	True	False	0	299.4
4	OK	75	415	330-6626	True	False	0	166.7
6	MA	121	510	355-9993	False	True	24	218.2
...	...	...	...	...	...	...	...	...
4995	HI	50	408	365-8751	False	True	40	235.7
4996	WV	152	415	334-9736	False	False	0	184.2
4997	DC	61	415	333-6861	False	False	0	140.6
4998	DC	109	510	394-2206	False	False	0	188.8
4999	VT	86	415	373-8058	False	True	34	129.4

4994 rows × 21 columns

In [43]:

```
# Display the number of unique values for the row that has over 4000
# values.  I wanted to make sure that were non unique values since
# I am considering getting columns since every value is unique and
# it could interfere with generalization of the classification
# algoirthm
print("columns that have too many values to show in a graph")
print("Phone Number: example=", revision_2['phone_number'][0], "and unique count is ", rev
print("I have decided to drop the phone number after a great deal of thought.  It interfere
revision_3 = revision_2.copy(deep=True)
revision_3.drop(labels=['phone_number'], axis=1, inplace=True)

```

executed in 10ms, finished 16:30:50 2023-04-24

```
columns that have too many values to show in a graph
Phone Number: example=  382-4657 and unique count is  4994
I have decided to drop the phone number after a great deal of thought.  It interferes with g
eneralization
```

In [44]:

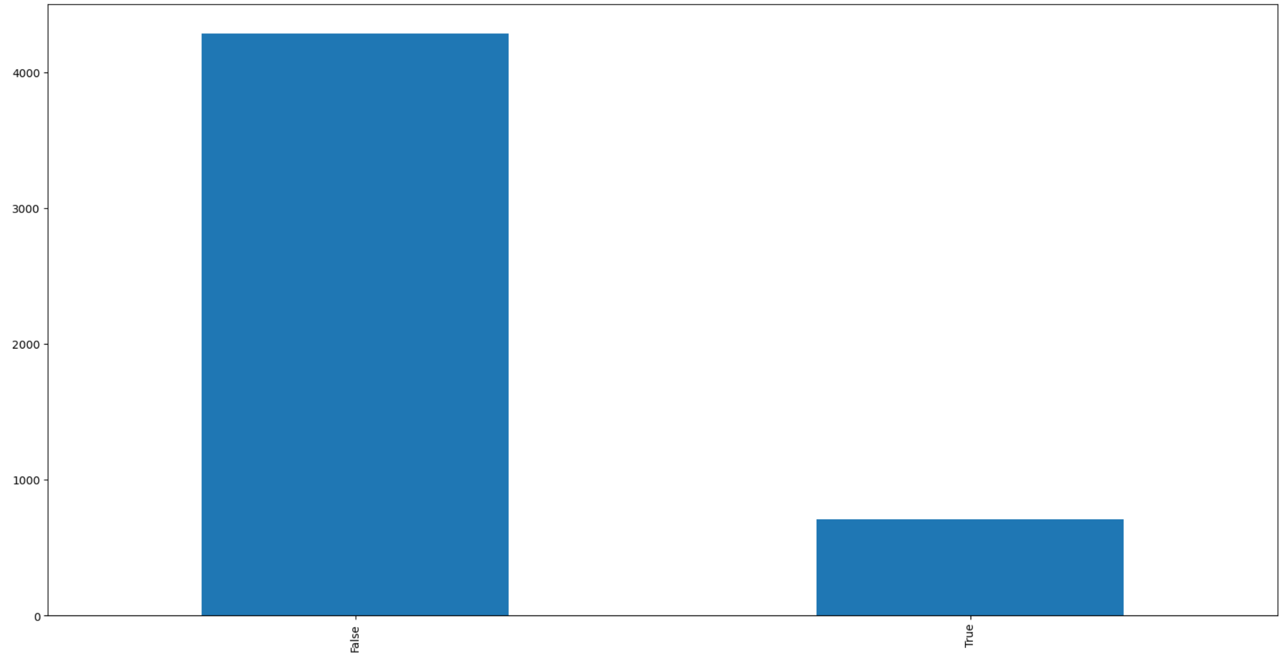
```
# Lets Look at the Target Values
```

executed in 3ms, finished 16:30:50 2023-04-24

In [45]:

```
# Looked at the number of people who were current and past cuostmers
# I was glad to see that tghe number of past customers was very small
revision_3.churned.value_counts().plot( x='index', y='state', kind='bar', figsize=(20,10))
plt.show()
# revision_2.churned.value_counts()
total_number = revision_3[['churned']].count().to_list()[0]
```

executed in 162ms, finished 16:30:50 2023-04-24



6.1 Analysis of Univariate Data

1. The target is churn.
2. The descriptions are a bit vague, but I following observations

> - The overall idea is to understand the churn for three different areacodes : 415,408,510.
> - The researchers thought most important was the plans, total_minutes and charges and charges and service calls.

6.2 Cleanup to be completed to move on to multivariate data

> - total_eve_charge is of type object but should be a float since it is dealing with currenncy
> - The rows with the Questions marks (invalid) will be removed since they are only five rows of data

In [46]:

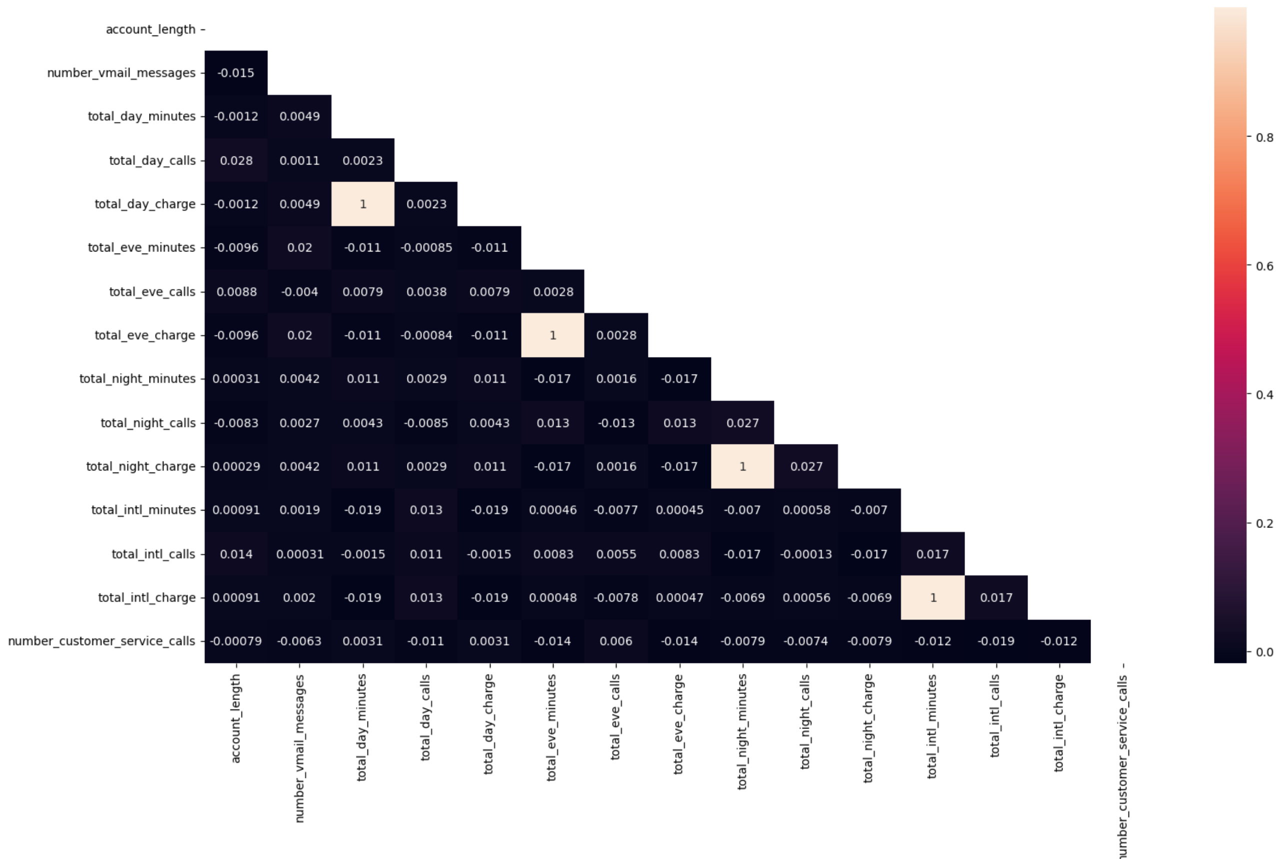
```
# The heatamp is a graphical representation of the correlation
# between two continuous varibles.  Once I get the paramters selected
# I wil try remove one of thwo correlations and see the effect that
# has on the machine algorithms

fig, ax = plt.subplots(figsize=(18,10))
# columns_to_drop = ['intl_plan', 'voice_mail_plan']

# heatmap = temporary_dataframe.drop(columns_to_drop, axis=1)
coorelation = temporary_dataframe.corr()
sns.heatmap(coorelation, annot=True,
            mask=np.triu(coorelation,0),
            ax=ax)
```

executed in 800ms, finished 16:30:51 2023-04-24

Out[46]: <AxesSubplot:>



6.3 Convert the Data Frame to be machine Friendly

- Convert all boolean: intl_plan, voice_mail_plan, churned to numeric
- Convert state, area_code, phone_number to categories

There are two ways to convert categorical to be machine friendly. There are two types of encoding Integer Encoding and One Hot Encoding.

- Integer Encoding
- - implies an order which could which could cause poor performance or (predictions halfway between categories)
- One Hot Encoding

I have decided not drop one column with One Hot Encoding since it could introduce a bias toward the dropped variable when regularizing

In [47]:

```
revision_4 = revision_3.copy(deep=True)

if current_test.get_one_hot_encoding() == False:
    states = revision_4.state.astype('category').copy(deep=True)
    revision_4.drop('state', axis=1)
    revision_4['state'] = states.cat.codes
    revision_4['state'] = revision_4['state'].astype(np.int64)

    areaCodes = revision_4.area_code.astype('category').copy(deep=True)
    revision_4.drop('area_code', axis=1)
    revision_4['area_code'] = areaCodes.cat.codes

    revision_4['intl_plan'] = revision_4["intl_plan"].astype(int)
    revision_4['voice_mail_plan'] = revision_4['voice_mail_plan'].astype(int)
else:
    revision_4 = pd.concat([revision_4,pd.get_dummies(revision_4.state,prefix="state")],ax
    revision_4 = pd.concat([revision_4,pd.get_dummies(revision_4.area_code,prefix="area")]
    revision_4 = pd.concat([revision_4,pd.get_dummies(revision_4.intl_plan,prefix="intl")]
    revision_4 = pd.concat([revision_4,pd.get_dummies(revision_4.voice_mail_plan,prefix="vo

    revision_4 = revision_4.drop('state',axis=1)
    revision_4 = revision_4.drop('area_code',axis=1)
    revision_4 = revision_4.drop('intl_plan',axis=1)
    revision_4 = revision_4.drop('voice_mail_plan',axis=1)

revision_4['churned'] = revision_4['churned'].astype(int)
```

executed in 28ms, finished 16:30:51 2023-04-24

6.4 Inbalanced Dataset

Inbalanced dataset causes the class in the target that has the majority to have bias. Meaning the classw with the majority will have more change of getting picked.

In [48]:

```
# When doing the analysis of the dataframe was confirmed to be
# inbalanced.  I have deicded to go with down sampling since it
# seemed the safest to implement I was not confident in building
# the data without inserting bias.

if current_test.get_balanced() == True:

    data_frame_majority = revision_4[revision_4.churned == False]
    data_frame_minority = revision_4[revision_4.churned == True]

    new_data = data_frame_majority.sample(n=len(data_frame_minority), \
                                          replace=True, random_state=132)
    revision_5 = pd.concat([data_frame_minority, new_data], axis=0)
else:
    revision_5 = revision_4.copy()

```

executed in 8ms, finished 16:30:51 2023-04-24

In [49]:

```
# Split the dataframe into a categorical data and numeric data.  The
# assumption is the order of the rows will change
categorical_set = set(revision_5.select_dtypes(np.uint8).columns)

non_categorical_set = revision_5.columns.difference(categorical_set)
non_categorical_set = non_categorical_set.drop('churned')

print("++++++++++++++++++++++++++++++++++++++++++")
categorical_dataframe_1 = revision_5[categorical_set]

numeric_dataframe_1 = revision_5[non_categorical_set]
numeric_dataframe_1.head(2)

# print(revision_5.info())
revision_4
```

executed in 34ms, finished 16:30:51 2023-04-24

```
++++++++++++++++++++++++++++++++++++++++++
```

Out[49]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_ev
0	128	25	265.1	110	45.07	197.4	
2	137	0	243.4	114	41.38	121.2	
3	84	0	299.4	71	50.90	61.9	
4	75	0	166.7	113	28.34	148.3	
6	121	24	218.2	88	37.09	348.5	
...	...	...	...	...	...	...	
4995	50	40	235.7	127	40.07	223.0	
4996	152	0	184.2	90	31.31	256.8	
4997	61	0	140.6	89	23.90	172.8	
4998	109	0	188.8	67	32.10	171.7	
4999	86	34	129.4	102	22.00	267.1	

4994 rows × 74 columns

7 Create the Training/Validation/Test Sets

In [50]:

```
features = revision_5.drop("churned", axis=1)
target = revision_5["churned"]

(XTrain, XTest, yTrain, yTest) = \
    train_test_split(features, target,\
                     test_size=.2, random_state=42,stratify=target)
(XTrain, VValidation, yTrain, vValidation) = \
    train_test_split(XTrain, yTrain, test_size=.2, random_state=42)

print("Testing/Training shapes:", XTrain.shape, yTrain.shape, VValidation.shape, vValidati

print(type(VValidation))
VValidation
```

executed in 37ms, finished 16:30:51 2023-04-24

```
Testing/Training shapes: (3196, 73) (3196,) (799, 73) (799,) (999, 73) (999,)
<class 'pandas.core.frame.DataFrame'>
```

Out[50]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_ev
1599	87	0	189.5	113	32.22	204.9	
3888	137	20	268.5	91	45.65	128.8	
2933	98	0	158.4	71	26.93	306.6	
4032	29	0	163.7	109	27.83	207.3	
1643	107	0	134.0	104	22.78	174.5	
...	...	...	...	...	...	...	
4960	128	41	184.8	76	31.42	160.6	
213	86	31	167.6	139	28.49	113.0	
1925	84	0	190.2	102	32.33	197.7	
3894	138	0	138.7	124	23.58	280.3	
1946	48	0	198.2	73	33.69	202.8	

799 rows × 73 columns

8 Preprocessing

Normalize or Standardize is feature scaling. The purpose is to ensure all features contribute equally to the the model and prevent featues with larget values from dominating the model.

standardization is that your data follows a Gaussian (bell curve) distribution. This isn't required, however, it helps the approach work better if your attribute distribution is Gaussian. -- Gaussian NB

Normalization is useful when your data has variable scales and the technique you're employing, such as k-nearest neighbors and artificial neural networks, doesn't make assumptions about the distribution of your data. -- Linear SVC, K-Neighbor Classifier, Logistic Regression

Algorithms that do not require featuer scaling : Decision Trees

Normalization	Standardization
Rescales values to a range between 0 and 1	Centers data around the mean and scales to a standard deviation of 1
Useful when the distribution of the data is unknown or not Gaussian	Useful when the distribution of the data is Gaussian or unknown
Sensitive to outliers	Less Sensitive to outliers

Retains the shape of the original distribution	Changes the shape of the distribution
May not preserve the relationships between the data points	Preserves the relationships between the data points
Equation: (x – min)/(max – min)	Equation: (x – mean)/standard deviation

In [51]:

```
# Since we are experimenting with different machine learning
# algorithms we need both Standard Scalar and normalization a
# dictionary will be created to store unique datasets (
# such as for StandardScalar, Normaliation)
preprocessed_data = {}

def splitDataframe(data_frame):
    '''
        splits the dataframe into categorical data/numbers for normalization/standard scala
    '''
    x_categorical = data_frame[categorical_set]
    x_numerical = data_frame[non_categorical_set]

    x_categorical.reset_index(drop=True, inplace=True)
    x_numerical.reset_index(drop=True, inplace=True)

    return x_categorical, x_numerical

def createDataframe(x_scalar_numerical, x_numerical, x_categorical):
    temp_dataframe = pd.DataFrame(x_scalar_numerical.transform(x_numerical), columns=x_num
    temp_dataframe = pd.concat([x_categorical, temp_dataframe], axis=1)
    return temp_dataframe

if ( current_test.get_do_normalization() == True ):

    # Standard Scalar

    (x_categorical, x_numerical ) = splitDataframe(XTrain)
    x_scalar_numerical = StandardScaler().fit(x_numerical)
    preprocessed_data['train_scaler'] = createDataframe(x_scalar_numerical, x_numerical,x_

    (x_categorical, x_numerical) = splitDataframe(VValidation)
    preprocessed_data['validate_scaler'] = createDataframe( x_scalar_numerical, x_numerica

    (x_categorical, x_numerical) = splitDataframe(XTest)
    preprocessed_data['test_scaler'] = createDataframe(x_scalar_numerical, x_numerical, x_

    # MinMax
    (x_categorical, x_numerical ) = splitDataframe(XTrain)
    x_scalar_numerical = MinMaxScaler().fit(x_numerical)
    preprocessed_data['train_minmax'] = createDataframe(x_scalar_numerical, x_numerical,x_

    (x_categorical, x_numerical) = splitDataframe(VValidation)
    preprocessed_data['validate_minmax'] = createDataframe( x_scalar_numerical, x_numerica

    (x_categorical, x_numerical) = splitDataframe(XTest)
    preprocessed_data['test_minmax'] = createDataframe(x_scalar_numerical, x_numerical, x_

else:
    preprocessed_data['train_scaler'] = XTrain
    preprocessed_data['validate_scaler'] = VValidation
    preprocessed_data['test_scaler'] = XTest
    preprocessed_data['train_minmax'] = XTrain
    preprocessed_data['validate_minmax'] = VValidation
    preprocessed_data['test_minmax'] = XTest

preprocessed_data['train'] = XTrain
preprocessed_data['validate'] = VValidation
preprocessed_data['test'] = XTest
```

executed in 13ms, finished 16:30:51 2023-04-24

9 Machine Learning

To choose the algorithms I used the following diagram.

9.1 Identificaton of algorithm

1. Start at start
2. The length of the data is 4994 and is > 50
3. Predicting a Catgory -- Churned Column
4. Labeled Data -- Yes, the labeled data is True or False
5. Less then 100,000 -- Start with Linear SVC and then took Naive Bayees and KNieghbors Classifier
6. Prepared for failure by Selecting Navie Bays as contingency
7. Select two random : KNeighbors Classifier since it close to other and DecisionTree since it explains systems

In [52]:

```
from IPython.display import Image
Image(filename="./DecisionTree.png")
```

executed in 12ms, finished 16:30:51 2023-04-24

Out[52]:

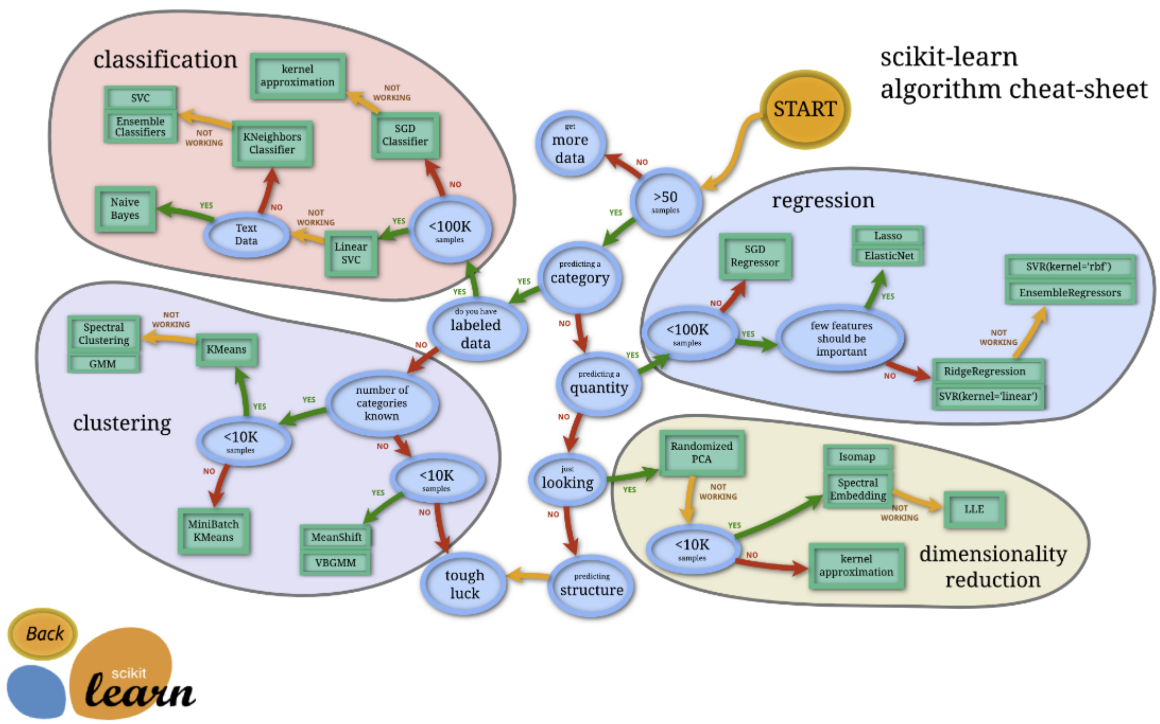


Image Source: Link

In [53]:
```
cv = 10
list_accuracy_list = []

def getStandardScaler():
    return preprocessed_data['train_scaler'], preprocessed_data['validate_scaler']

def getMinMax():
    return preprocessed_data['train_minmax'], preprocessed_data['validate_minmax']

def getRaw():
    return preprocessed_data['train'], preprocessed_data['validate']

def param_grid():
    if (current_test.get_algorithm() == None ):
        return {}
    else:
        return current_test.get_parameter_grid()

def getNormalization(isStandardScalar):
    '''
       Returns the Normalization to use can be the correct or other
       normalizaiton since this code is for experiments.
       Input A boolean (T) -- Scalar, (F) -- MinMax,  (None) -- No Normalization
       Output : The normalized or raw data
    '''
    print("Inside -- getNormalization")
    if isStandardScalar == None:
        return getRaw()
    elif ( isStandardScalar == True):
        if (current_test.get_reverse_normalization == False):
            return getStandardScaler()
        else:
            return getMinMax()
    elif isStandardScalar == False:
        print("isStandardScalar = False")
        if (current_test.get_reverse_normalization() == False):
            print("MinMAx()")
            return getMinMax()
        else:
            print("StandardScalar()")
            return getStandardScaler()
```

executed in 8ms, finished 16:30:51 2023-04-24

In [54]:
```
def displayTextualResults(grid,X,y_test, y_predict):
    '''
        Display the results of a run where we are looking at
        preprocessing and runnin the algorithm without any
        parameters
    '''
    print("best_index = ", grid.best_index_)
    print("best_estimator = ", grid.best_estimator_)
    print("best_params = ", grid.best_params_)
    print("best_score = ", grid.best_score_)
    print("Test Score = ", grid.score(X, y_predict))
    print("--------------------------------------------")

    # Explamation of classification report
    # Precision - Proportion of positives identifications were correct
    #     True Positive / (True Positive + False Positive)
    # Recall    - Proportion of positives identified correctly
    #     True Positive / ( True Positive + False Negative)
    # F1 Score
    print(classification_report(y_test, y_predict))

    # confusion_matrix(y_true=y_test, y_pred=y_predict)



```

executed in 5ms, finished 16:30:51 2023-04-24

In [55]:
```
def workflow(algorithm, xTrain, VValidation):
    '''
        Creates the grid search and display the results to the user
        and retrieve the accuracy to store as part of the record of
        the run
    '''
    grid = GridSearchCV(algorithm, param_grid=param_grid(), cv=cv,verbose=5)
    grid = grid.fit(xTrain,yTrain)
    predictions = grid.predict(VValidation)
    displayTextualResults(grid, VValidation, vValidation, predictions)
    accuracy = (accuracy_score(y_true=vValidation, y_pred=predictions))
    accuracy = np.round(accuracy, 2)
    return grid, accuracy
```

executed in 7ms, finished 16:30:51 2023-04-24

In [56]:
```
### Linear SVC
###    SVC -- Separates data points data points with a large margin
###           between each set of points
###    Linear SVC uses the linear kerenel
###    Hyperparameters
###        C -- Regularization.  Applied to the linera kernal function
###        and inverly proportional to regularization
###        Penalty which one to L1 (Ridge) or L2(Lasso)
###
# parameter_grid = {"C": [0,1,2,3,4,5],
#                     "penalty": ['l1', 'l2'],
#                     "max_iter":[100,1000, 10000]}
# parameter_grid = {"C": [0,1,2,3,4,5],
#                      "penalty": ['l1', 'l2'],
#                      "max_iter":[1000, 100000, 100000]}
# parameter_grid = {"C": [3,4,5],
#                      "penalty": ['l2'],
#                      "max_iter":[1000000]}
# grid = GridSearchCV(LinearSVC(random_state=42), \
#                       param_grid=parameter_grid, cv=cv,verbose=5)
if current_test.get_algorithm() == 'SVC' or current_test.get_algorithm() == None:
    (Xtrain, VValidation) = getNormalization(False)
    grid, accuracy = workflow(LinearSVC(random_state=42),Xtrain,VValidation)
    list_accuracy_list.append(accuracy)

```

executed in 5ms, finished 16:30:51 2023-04-24

In [57]:
```
# GaussianNB  -- Assumes data is drawn from simple Gaussian
# distribution.  Can be fed partial data in chucn.  The dividing
# line is a parabola rather than a dividing line
#    Hyperparameters
#       priors -- Represents the proior probabilities fo the class
#       var_smoothing -- Give the portion of the largetst variance
#                        of the feature that is added inorder to
#
if current_test.get_algorithm() == "GAUSS" or current_test.get_algorithm() == None:
    (Xtrain, VValidation) = getNormalization(True)
    grid, accuracy =  workflow(GaussianNB(),Xtrain, VValidation)
    list_accuracy_list.append(accuracy)

```

executed in 5ms, finished 16:30:51 2023-04-24

In [58]:
```
# KNeighborClassifier
# hyperparameters -- Number of Point considered neighbors
if ( current_test.get_algorithm() == 'KNN'  or current_test.get_algorithm() == None):
    (xtrain, VValidation) = getNormalization(True)
    grid, accuracy = workflow(KNeighborsClassifier(),xtrain, VValidation)
    list_accuracy_list.append(accuracy)
```

executed in 6ms, finished 16:30:51 2023-04-24

In [59]:
```
# Decision Tree -- Predicts the value of a target by learning simple
# decision rules inferred from the data features
# HyperParameters
#    criterion -- A function to measure the quality of the pslit
#    max_depth -- Choose the best or the best random split
#    min_samples_split -- The minimal number of samples to split a node
#    min_samples_leaf -- The mininium number of samples to be a leaf node
#    min_weight_fraction_leaf -- Sum total of weights required to be a leaf node
#    max_featuers -- Number of featrues to determine when looking at the correct split
if ( current_test.get_algorithm() == 'DTREE'  or current_test.get_algorithm() == None):
    (xtrain, VValidation) = getNormalization(None)
    grid, accuracy = workflow(DecisionTreeClassifier(),xtrain, VValidation)
    list_accuracy_list.append(accuracy)
    print("final list =", list_accuracy_list)
```

executed in 42m 42s, finished 17:13:33 2023-04-24

```
e=   0.0s
[CV 9/10] END max_depth=27, min_samples_leaf=8, min_samples_split=2;, score=0.922 total tim
e=   0.0s
[CV 10/10] END max_depth=27, min_samples_leaf=8, min_samples_split=2;, score=0.925 total ti
me=   0.0s
[CV 1/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.950 total tim
e=   0.0s
[CV 2/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.950 total tim
e=   0.0s
[CV 3/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.941 total tim
e=   0.0s
[CV 4/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.938 total tim
e=   0.0s
[CV 5/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.928 total tim
e=   0.0s
[CV 6/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.938 total tim
e=   0.0s
[CV 7/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.931 total tim
e=   0.0s
[CV 8/10] END max_depth=27, min_samples_leaf=8, min_samples_split=3;, score=0.944 total tim
```

In [60]:
```
# Logisitic  -- modeling the probability of a discrete outcome given an input variable.
# hyperparameters
#    parameters -- Specify the norm of the panalty
#    C -- Inverse of regularization strength
#    Algoirthm -- Used in the optimiation problem
if ( current_test.get_algorithm() == 'DECISION'  or current_test.get_algorithm() == None):
    (xtrain, VValidation) = getNormalization(False)
    grid, accuracy = workflow(LogisticRegression(),xtrain,VValidation)
    list_accuracy_list.append(accuracy)

```

executed in 4ms, finished 17:13:33 2023-04-24

10 Show all runs

This section wil create the dataframes of the runs so we understand which run were better and can see the conditions need to reproduce the run.

In [61]:
```python

import os
import shutil

dataFrame_2 = None
if (current_test.get_algorithm() == None):
    print("The filename is ", file)
    print("The remove is ", remove_file)
    dataFrame_2 = None

    # If the user decide to remove the file remove and create a new
    # dataframe
    #
    # If the user does not remove file the data frame is rea from the
    # file
    if ( remove_file == True ):
       if os.path.isfile(file):
            print("*** Deleting file ***")
            new_file_name = file + ".orig"
            shutil.copy(file, new_file_name)
            os.remove(file)
       dataFrame_2 = pd.DataFrame(index=algorithm_list)
       print(dataFrame_2)
    else:
        print("Setting Dataframe")
        print("File = ", file)
        dataFrame_2 = pd.read_csv(file)
        print("The size is ", dataFrame_2.shape)
        dataFrame_2.index = algorithm_list

    # Add a new run to the data frame
    column = len(dataFrame_2.columns)
    print("columns = ", column)
    list_accuracy_list.insert(0, current_test.get_description())
    dataFrame_2[column] = list_accuracy_list
    dataFrame_2.to_csv(file, index=False)

    # Since the descirptions are too long, put the descriptions
    # above the dataframe so they are easily readable
    descriptions = dataFrame_2.iloc[0,:]
    for index, description in enumerate(descriptions):
         print(index, "  -- ",description)

    # Add the index and remove the row with descriptions
    dataFrame_2.index = algorithm_list
    dataFrame_2.to_csv(file, index=False)
    dataFrame_2 = dataFrame_2[1:]
    print("Went through for loo")

dataFrame_2


```
executed in 9ms, finished 17:13:33 2023-04-24

In [62]:

```
import os
import shutil

columns=["Description", "Algorithm", "Parameters Tried", "Best Parameters", "Accuracy"]

# Create the row to be stored
row = [ current_test.get_description(),
       current_test.get_algorithm(),current_test.get_parameter_grid(), grid.best_params_,

#
# If requested removes the file and creates a new files
#
# If requested to add a row to an exiting data then read the
# cs file from disk and adds the header
dataFrame_3 = None
if (current_test.get_algorithm() != None):
    print("The filename is ", file2)
    dataFrame_3 = None
    if ( remove_file_2 == True ):
       if os.path.isfile(file2):
            print("*** Deleting file ***")
            new_file_name = file2 + ".orig"
            shutil.copy(file2, new_file_name)
            os.remove(file2)
       dataFrame_3 = pd.DataFrame(columns=columns)
       print(dataFrame_3)
    else:
        print("Setting Dataframe")
        print("File = ", file2)
        if ( os.path.isfile(file2)):
            dataFrame_3 = pd.read_csv(file2,index_col=False)
        else:
            dataFrame_3 = pd.DataFrame(columns=columns)
        print("The size is ", dataFrame_3.shape)

#Add the row and write the file to the disk.
print("size of dataframe ", dataFrame_3.size)
print("row = ",row)
next_row = dataFrame_3.shape[0]
print("next row = ", next_row)
dataFrame_3.loc[next_row] = row
dataFrame_3.to_csv(file2, index=False)

# Show the output here.
dataFrame_3
```

executed in 35ms, finished 17:13:33 2023-04-24

```
The filename is  parameters.csv
Setting Dataframe
File =  parameters.csv
The size is  (7, 5)
size of dataframe  35
row =  ['No Balanced Normalization Introduced ( ex. MinMax used instead of scalar)', 'DTRE
E', {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18]}, {'max_depth': 7, 'min_samples_leaf': 9, 'min_samples_split': 4}, 0.95]
next row =  7
```

Out[62]:

	Description	Algorithm	Parameters Tried	Best Parameters	Accuracy
0	Initial Test with One Hot Encoding	KNN	{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}	{'n_neighbors': 7}	0.90
1	Initial Test with One Hot Encoding	KNN	{'n_neighbors': [21, 23, 25, 27, 29, 31, 33, 35, 37, 39]}	{'n_neighbors': 27}	0.89
2	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	KNN	{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}	{'n_neighbors': 7}	0.90
3	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	KNN	{'n_neighbors': [21, 23, 25, 27, 29, 31, 33, 35, 37, 39]}	{'n_neighbors': 27}	0.89
4	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]}	{'max_depth': 6}	0.95
5	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 6, 'min_samples_split': 17}	0.95
6	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 6, 'min_samples_split': 17}	0.95
7	No Balanced Normalization Introduced (ex. MinMax used instead of scalar)	DTREE	{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], 'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]}	{'max_depth': 7, 'min_samples_leaf': 9, 'min_samples_split': 4}	0.95