```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
import pandas as pd
import numpy as  np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

```
raw_data = pd.read_csv('/content/drive/MyDrive/Data science projects /churn.csv', delimiter=',')
data = raw_data.copy()
data
```

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	tota
0	16	128	415	2845	0	1	25	265.1	110	
1	35	107	415	2301	0	1	26	161.6	123	
2	31	137	415	1616	0	0	0	243.4	114	
3	35	84	408	2510	1	0	0	299.4	71	
4	36	75	415	155	1	0	0	166.7	113	
...	...	...	...	...	...	...	...	...	...	
4995	11	50	408	2000	0	1	40	235.7	127	
4996	49	152	415	394	0	0	0	184.2	90	
4997	7	61	415	313	0	0	0	140.6	89	
4998	7	109	510	3471	0	0	0	188.8	67	
4999	46	86	415	2412	0	1	34	129.4	102	

5000 rows × 21 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   state                          5000 non-null   int64  
 1   account_length                 5000 non-null   int64  
 2   area_code                      5000 non-null   int64  
 3   phone_number                   5000 non-null   int64  
 4   international_plan             5000 non-null   int64  
 5   voice_mail_plan                5000 non-null   int64  
 6   number_vmail_messages          5000 non-null   int64  
 7   total_day_minutes              5000 non-null   float64
 8   total_day_calls                5000 non-null   int64  
 9   total_day_charge               5000 non-null   float64
 10  total_eve_minutes              5000 non-null   float64
 11  total_eve_calls                5000 non-null   int64  
 12  total_eve_charge               5000 non-null   float64
 13  total_night_minutes            5000 non-null   float64
 14  total_night_calls              5000 non-null   int64  
 15  total_night_charge             5000 non-null   float64
 16  total_intl_minutes             5000 non-null   float64
 17  total_intl_calls               5000 non-null   int64  
 18  total_intl_charge              5000 non-null   float64
 19  number_customer_service_calls  5000 non-null   int64  
 20  class                          5000 non-null   int64  
dtypes: float64(8), int64(13)
memory usage: 820.4 KB
```

```
data.isnull().sum()
```

```
state                            0
account_length                   0
area_code                        0
phone_number                     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
class                         0
dtype: int64
```

```
data.describe()
```

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_call
count	5000.00000	5000.00000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000
mean	25.99840	100.25860	436.911400	2499.500000	0.094600	0.264600	7.755200	180.288900	100.02940
std	14.80348	39.69456	42.209182	1443.520003	0.292691	0.441164	13.546393	53.894699	19.83119
min	0.00000	1.00000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	13.00000	73.00000	408.000000	1249.750000	0.000000	0.000000	0.000000	143.700000	87.00000
50%	26.00000	100.00000	415.000000	2499.500000	0.000000	0.000000	0.000000	180.100000	100.00000
75%	39.00000	127.00000	415.000000	3749.250000	0.000000	1.000000	17.000000	216.200000	113.00000
max	50.00000	243.00000	510.000000	4999.000000	1.000000	1.000000	52.000000	351.500000	165.00000

8 rows × 21 columns

```
dataset = data.copy()
```

```
#exploring the dataset certain data like phone_number, area_code are not needed for this model, so it would be taken out
data_to_remove = ['phone_number', 'area_code']
```

```
dataset = dataset.drop(data_to_remove, axis=1)
```

```
dataset.head()
```

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minu
0	16	128	0	1	25	265.1	110	45.07	1
1	35	107	0	1	26	161.6	123	27.47	1
2	31	137	0	0	0	243.4	114	41.38	1
3	35	84	1	0	0	299.4	71	50.90	
4	36	75	1	0	0	166.7	113	28.34	1

Next steps: Generate code with dataset | View recommended plots

```
inputs_unbalanced = dataset.iloc[:,:-1]
targets_unbalanced = dataset.iloc[:,-1]
```

```
inputs_unbalanced
```

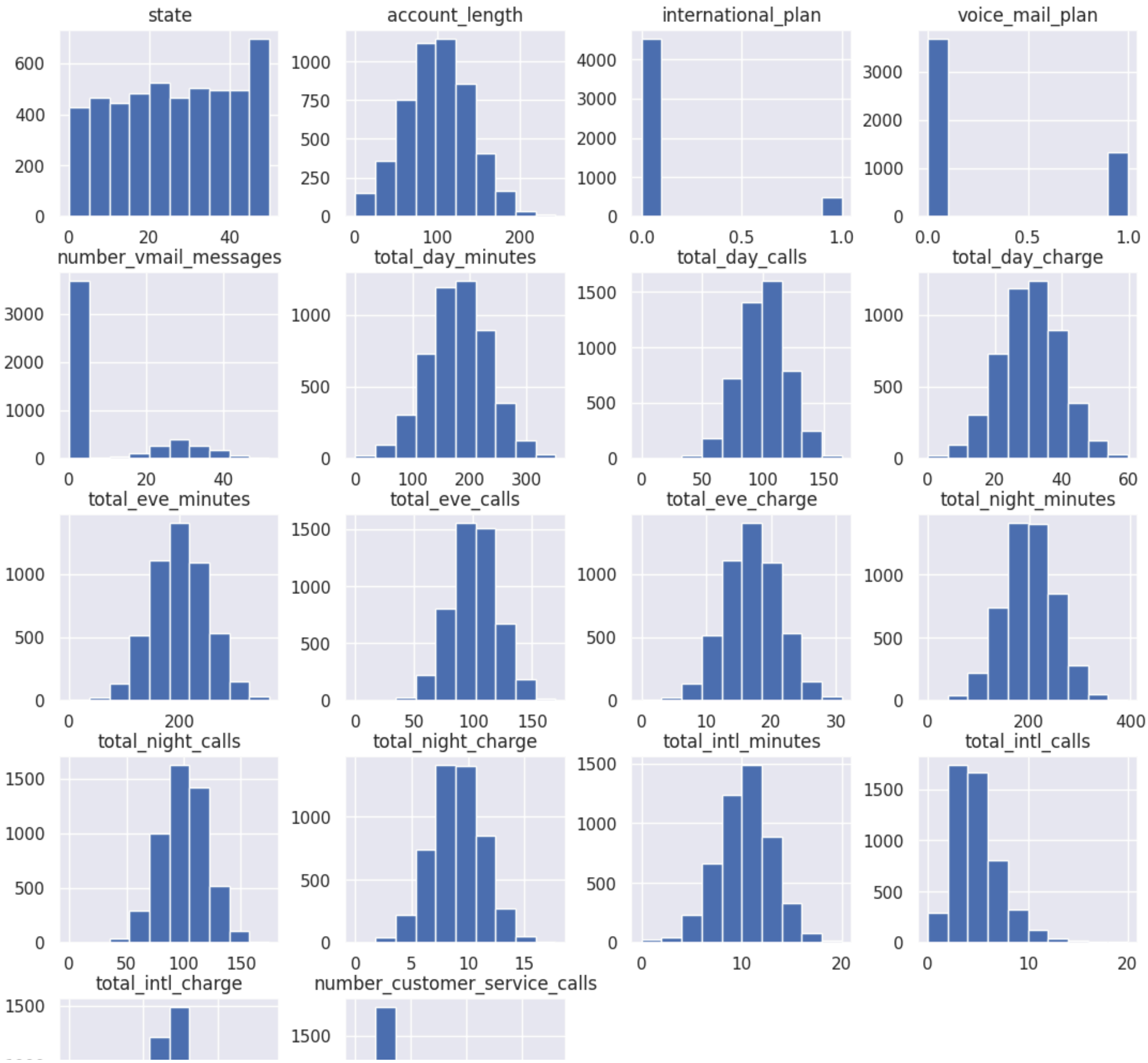
	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_m
0	16	128	0	1	25	265.1	110	45.07	
1	35	107	0	1	26	161.6	123	27.47	
2	31	137	0	0	0	243.4	114	41.38	
3	35	84	1	0	0	299.4	71	50.90	
4	36	75	1	0	0	166.7	113	28.34	
...	...	...	...	...	...	...	...	...	
4995	11	50	0	1	40	235.7	127	40.07	
4996	49	152	0	0	0	184.2	90	31.31	
4997	7	61	0	0	0	140.6	89	23.90	
4998	7	109	0	0	0	188.8	67	32.10	
4999	46	86	0	1	34	129.4	102	22.00	

5000 rows × 18 columns

Next steps: Generate code with `inputs_unbalanced` View recommended plots

Explolatory data analysis

```
inputs_unbalanced.hist(figsize=(13,14))
plt.show()
```




```
inputs_unbalanced['international_plan'].value_counts(),inputs_unbalanced['voice_mail_plan'].value_counts()
```

```
(0    4527
 1     473
 Name: international_plan, dtype: int64,
 0    3677
 1    1323
 Name: voice_mail_plan, dtype: int64)
```

```
targets_unbalanced
```

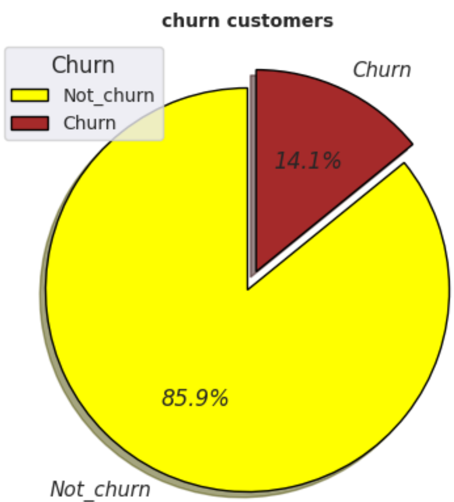
```
0       0
1       0
2       0
3       0
4       0
       ..
4995    0
4996    1
4997    0
4998    0
4999    0
Name: class, Length: 5000, dtype: int64
```

```
inputs_unbalanced.shape, targets_unbalanced.shape
```

```
((5000, 18), (5000,))
```

```
customer_churn = targets_unbalanced.value_counts()
```

```
#creating cusomer churn chart

wedgeprop = {'linewidth':1, 'edgecolor':'black', 'antialiased':True}
color = ['yellow', 'brown']
label = ['Not_churn', 'Churn']
textprops = {'fontstyle':'italic'}
explode = (0,0.1)


fig = plt.figure(figsize=(20, 5))

# Create the pie chart
plt.pie(customer_churn, colors=color, wedgeprops=wedgeprop, autopct="%0.1f%%", labels=label, startangle=90, explode=explode, shadow=True, textprops=textprops

# Add the legend
plt.legend(title='Churn', loc='upper left', fontsize='small')

# Add the title
plt.title('churn customers', fontsize='small', fontweight='bold')

# Show the pie chart
plt.show()
```



Balancing the dataset

```
#number of customers that churn
number_of_churn = int(np.sum(targets_unbalanced))
#count of non churn customers
zero_count = 0
#create empty list to append rows to be removed from data
row_to_remove = []

# Identify rows to remove
for i in range(len(targets_unbalanced)):
    if targets_unbalanced[i] == 0:
        zero_count += 1
        if zero_count > number_of_churn:
            row_to_remove.append(i)


# Remove rows from inputs and target
balanced_inputs = inputs_unbalanced.drop(row_to_remove)
balanced_targets = targets_unbalanced.drop(row_to_remove)


#checking if data is balanced
balanced_targets.sum()/balanced_targets.shape
```

```
array([0.5])
```

Shuffle the dataset

```
#shuffling the dataset
shuffle_indices = np.arange(balanced_targets.shape[0])
np.random.shuffle(shuffle_indices)
# Convert shuffle_indices to a NumPy array of integers
shuffle_indices = np.array(shuffle_indices, dtype=int)

# Use shuffle_indices to index the DataFrame
shuffled_inputs = balanced_inputs.iloc[shuffle_indices]
shuffled_targets = balanced_targets.iloc[shuffle_indices]
```

Scaling the variables

```
from sklearn.preprocessing import StandardScaler
```

```
 scale = StandardScaler()
scale.fit(balanced_inputs)
balanced_scale_inputs = scale.transform(balanced_inputs)
```

```
balanced_scale_inputs
```

```
array([[-0.65259473,  0.6956714 , -0.46344   , ..., -0.53225966,
        -0.15759705, -0.54759085],
       [ 0.64621334,  0.16154686, -0.46344   , ..., -0.53225966,
         1.17758703, -0.54759085],
       [ 0.37278006,  0.92458192, -0.46344   , ...,  0.25955008,
         0.63016156, -1.1812189 ],
       ...,
       [ 0.1677051 ,  1.00088543, -0.46344   , ...,  0.65545495,
        -1.05217038, -0.54759085],
       [-1.54125289, -0.09279816, -0.46344   , ...,  0.25955008,
        -0.58485595, -0.54759085],
       [ 1.60322981,  1.30609945, -0.46344   , ..., -0.92816453,
         1.53808673,  0.71966523]])
```

```
balanced_scale_inputs.shape
```

```
(1414, 18)
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,  x_test, y_train, y_test = train_test_split(balanced_scale_inputs, balanced_targets, test_size = 0.2, random_state=350)
```

```
from sklearn.linear_model import LogisticRegression #importing logistic regression
from sklearn.metrics import accuracy_score, precision_score #importing metrics to measure model accuracy and precision
```

```
pred = LogisticRegression()
```

```
pred.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
result = pred.predict(x_train)
```

```
accuracy = accuracy_score(y_train, result)
precision = precision_score(y_train, result)

print(f'Model accuracy {accuracy}, Model precision {precision}')
```

```
Model accuracy 0.7550839964633068, Model precision 0.7452667814113597
```

```
pred.intercept_
```

```
array([0.02042402])
```

```
pred.coef_
```

```
array([[ 0.12033786,  0.05649394,  0.84617471, -0.77824887,  0.37375208,
         0.38802795,  0.07139122,  0.38073672,  0.19111044,  0.05076029,
```

```
         0.18739011,  0.11816951, -0.07809806,  0.1144635 ,  0.13878716,
        -0.19342008,  0.08045209,  0.96727951]])
```

```
feature_names = inputs_unbalanced.columns.values
feature_names
```

```
array(['state', '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',
       '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'], dtype=object)
```

```
summary_table = pd.DataFrame(columns = ['Feature_names'], data = feature_names)
summary_table['Coefficient'] = np.transpose(pred.coef_)
summary_table
```

	Feature_names	Coefficient
0	state	0.120338
1	account_length	0.056494
2	international_plan	0.846175
3	voice_mail_plan	-0.778249
4	number_vmail_messages	0.373752
5	total_day_minutes	0.388028
6	total_day_calls	0.071391
7	total_day_charge	0.380737
8	total_eve_minutes	0.191110
9	total_eve_calls	0.050760
10	total_eve_charge	0.187390
11	total_night_minutes	0.118170
12	total_night_calls	-0.078098
13	total_night_charge	0.114463
14	total_intl_minutes	0.138787
15	total_intl_calls	-0.193420
16	total_intl_charge	0.080452
17	number_customer_service_calls	0.967280

Next steps: Generate code with `summary_table` View recommended plots

```
summary_table.index = summary_table.index + 1
summary_table.loc[0] = ['Intercept', pred.intercept_[0]]
summary_table = summary_table.sort_index()
summary_table
```

	Feature_names	Coefficient
0	Intercept	0.020424
1	state	0.120338
2	account_length	0.056494
3	international_plan	0.846175
4	voice_mail_plan	-0.778249
5	number_vmail_messages	0.373752
6	total_day_minutes	0.388028
7	total_day_calls	0.071391
8	total_day_charge	0.380737
9	total_eve_minutes	0.191110
10	total_eve_calls	0.050760
11	total_eve_charge	0.187390
12	total_night_minutes	0.118170
13	total_night_calls	-0.078098
14	total_night_charge	0.114463
15	total_intl_minutes	0.138787
16	total_intl_calls	-0.193420
17	total_intl_charge	0.080452
18	number_customer_service_calls	0.967280

Next steps: Generate code with `summary_table` View recommended plots

```
summary_table['Odds_ratio'] = np.exp(summary_table.Coefficient)
summary_table
```

	Feature_names	Coefficient	Odds_ratio
0	Intercept	0.020424	1.020634
1	state	0.120338	1.127878
2	account_length	0.056494	1.058120
3	international_plan	0.846175	2.330714
4	voice_mail_plan	-0.778249	0.459209
5	number_vmail_messages	0.373752	1.453177
6	total_day_minutes	0.388028	1.474071
7	total_day_calls	0.071391	1.074001
8	total_day_charge	0.380737	1.463362
9	total_eve_minutes	0.191110	1.210593
10	total_eve_calls	0.050760	1.052071
11	total_eve_charge	0.187390	1.206098
12	total_night_minutes	0.118170	1.125435
13	total_night_calls	-0.078098	0.924874
14	total_night_charge	0.114463	1.121272
15	total_intl_minutes	0.138787	1.148880
16	total_intl_calls	-0.193420	0.824136
17	total_intl_charge	0.080452	1.083777
18	number_customer_service_calls	0.967280	2.630778

Next steps: Generate code with `summary_table` View recommended plots

```
#sorting the summary table by its odds ratio
summary_table = summary_table.sort_values('Odds_ratio', ascending=False)
summary_table
```

	Feature_names	Coefficient	Odds_ratio
18	number_customer_service_calls	0.967280	2.630778
3	international_plan	0.846175	2.330714
6	total_day_minutes	0.388028	1.474071
8	total_day_charge	0.380737	1.463362
5	number_vmail_messages	0.373752	1.453177
9	total_eve_minutes	0.191110	1.210593
11	total_eve_charge	0.187390	1.206098
15	total_intl_minutes	0.138787	1.148880
1	state	0.120338	1.127878
12	total_night_minutes	0.118170	1.125435
14	total_night_charge	0.114463	1.121272
17	total_intl_charge	0.080452	1.083777
7	total_day_calls	0.071391	1.074001
2	account_length	0.056494	1.058120
10	total_eve_calls	0.050760	1.052071
0	Intercept	0.020424	1.020634
13	total_night_calls	-0.078098	0.924874
16	total_intl_calls	-0.193420	0.824136
4	voice_mail_plan	-0.778249	0.459209

Next steps: Generate code with summary_table View recommended plots

Using the odds_ratio column it shows that all of the variables used have an impact on the model as most of the variable odds_ratio is not close to zero hence removing any variable from the model will affect the accuracy of the model.

Testing the model

```
#checking the accuracy of the model
```