In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
df=pd.read_csv(r"C:\Users\santo\OneDrive\Desktop\Velocity\CSV Files\churn - churn.csv")
```

In [3]:

```
df.sample(5)
```

Out[3]:

	churn	accountlength	internationalplan	voicemailplan	numbervmailmessages	totalday
3651	No	42	no	yes	16	
53	No	96	no	no	0	
870	No	123	no	no	0	
552	Yes	44	no	no	0	
1756	No	118	yes	yes	39	

In [4]:

```
df["churn"]=df["churn"].map({"No":0,"Yes":1})
df["internationalplan"]=df["internationalplan"].map({"no":0,"yes":1})
df["voicemailplan"]=df["voicemailplan"].map({"no":0,"yes":1})
```

In [5]:

```
df.sample(5)
```

Out[5]:

	churn	accountlength	internationalplan	voicemailplan	numbervmailmessages	totalday
2623	0	134	0	0	0	
488	0	165	0	0	0	
3064	1	130	0	0	0	
817	0	243	0	0	0	
1950	1	52	0	0	0	

Exploratory data analysis

In [6]:

```
df.shape
```

Out[6]:

```
(5000, 18)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['churn', 'accountlength', 'internationalplan', 'voicemailplan',
       'numbervmailmessages', 'totaldayminutes', 'totaldaycalls',
       'totaldaycharge', 'totaleveminutes', 'totalevecalls', 'totalevechar
ge',
       'totalnightminutes', 'totalnightcalls', 'totalnightcharge',
       'totalintlminutes', 'totalintlcalls', 'totalintlcharge',
       'numbercustomerservicecalls'],
      dtype='object')
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 18 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   churn                       5000 non-null   int64  
 1   accountlength               5000 non-null   int64  
 2   internationalplan           5000 non-null   int64  
 3   voicemailplan               5000 non-null   int64  
 4   numbervmailmessages         5000 non-null   int64  
 5   totaldayminutes             5000 non-null   float64
 6   totaldaycalls               5000 non-null   int64  
 7   totaldaycharge              5000 non-null   float64
 8   totaleveminutes             5000 non-null   float64
 9   totalevecalls               5000 non-null   int64  
 10  totalevecharge              5000 non-null   float64
 11  totalnightminutes           5000 non-null   float64
 12  totalnightcalls             5000 non-null   int64  
 13  totalnightcharge            5000 non-null   float64
 14  totalintlminutes            5000 non-null   float64
 15  totalintlcalls              5000 non-null   int64  
 16  totalintlcharge             5000 non-null   float64
 17  numbercustomerservicecalls  5000 non-null   int64  
dtypes: float64(8), int64(10)
memory usage: 703.2 KB
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
churn                        0
accountlength                0
internationalplan            0
voicemailplan                0
numbervmailmessages          0
totaldayminutes              0
totaldaycalls                0
totaldaycharge               0
totaleveminutes              0
totalevecalls                0
totalevecharge               0
totalnightminutes            0
totalnightcalls              0
totalnightcharge             0
totalintlminutes             0
totalintlcalls               0
totalintlcharge              0
numbercustomerservicecalls   0
dtype: int64
```

In [10]:

```
df.duplicated().sum()
```

Out[10]:

```
0
```

In [11]:

```
df.describe()
```

Out[11]:

	churn	accountlength	internationalplan	voicemailplan	numbervmailmessages	t
count	5000.000000	5000.00000	5000.000000	5000.000000	5000.000000	
mean	0.141400	100.25860	0.094600	0.264600	7.755200	
std	0.348469	39.69456	0.292691	0.441164	13.546393	
min	0.000000	1.00000	0.000000	0.000000	0.000000	
25%	0.000000	73.00000	0.000000	0.000000	0.000000	
50%	0.000000	100.00000	0.000000	0.000000	0.000000	
75%	0.000000	127.00000	0.000000	1.000000	17.000000	
max	1.000000	243.00000	1.000000	1.000000	52.000000	

In [12]:

```
df["churn"].value_counts()
```

Out[12]:

```
0    4293
1     707
Name: churn, dtype: int64
```

In [13]:

```
from pandas_profiling import ProfileReport
prof=ProfileReport(df)
prof.to_file(output_file="Output.html")
```

```
C:\Users\santo\AppData\Local\Temp\ipykernel_2764\1332261464.py:1: Deprecat
ionWarning: `import pandas_profiling` is going to be deprecated by April 1
st. Please use `import ydata_profiling` instead.
  from pandas_profiling import ProfileReport

Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

seperate the features and target

In [14]:

```
x=df.iloc[:,1:]
y=df.iloc[:,0]
```

detecting outlier

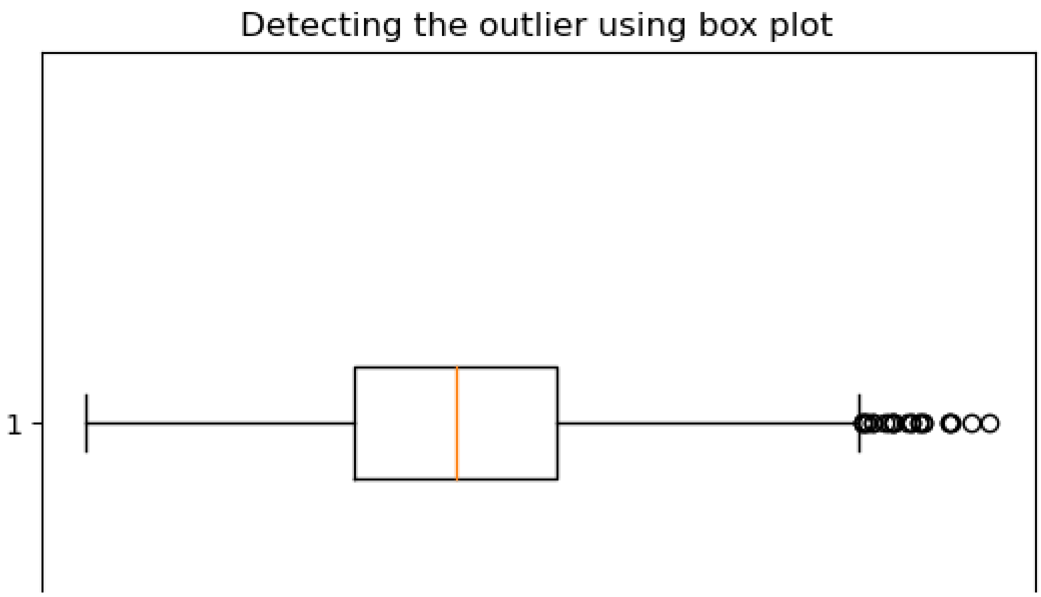
In [15]:

```
%matplotlib inline
```

In [16]:

```
import matplotlib.pyplot as plt
for i in x.columns:
    print("detecting outlier for {} column".format(i))
    plt.boxplot(df[i],vert=False)
    plt.title("Detecting the outlier using box plot")
    plt.xlabel("sample df data")
    plt.show()
```

```
detecting outlier for accountlength column
```



In [ ]:

Outlier handle

In [17]:

```
#trimming
#for i in x.columns:
   # Lower_bound=df[i].mean()- 3*(df[i].std())
   # Upper_bound=df[i].mean()+ 3*(df[i].std())
   # df[(df[i]>Upper_bound) | (df[i]<Lower_bound)]
   #new_df=df[(df[i]<Upper_bound) & (df[i]>Lower_bound)]
#new_df
```

In [18]:

```
#capping
for i in x.columns:
    percentile_25=df[i].quantile(0.25)
    percentile_75=df[i].quantile(0.75)
    IQR= percentile_75- percentile_25
    upper_limit=percentile_75+1.5*IQR
    lower_limit=percentile_25-1.5*IQR
    df[i]=np.where(df[i]>upper_limit,upper_limit,np.where(df[i]<lower_limit,lower_limit,
df
```

Out[18]:

	churn	accountlength	internationalplan	voicemailplan	numbervmailmessages	totaldayminutes	tota
0	0	128.0	0.0	1.0	25.0	265.1	
1	0	107.0	0.0	1.0	26.0	161.6	
2	0	137.0	0.0	0.0	0.0	243.4	
3	0	84.0	0.0	0.0	0.0	299.4	
4	0	75.0	0.0	0.0	0.0	166.7	
...	...	...	...	...	...	...	
4995	0	50.0	0.0	1.0	40.0	235.7	
4996	1	152.0	0.0	0.0	0.0	184.2	
4997	0	61.0	0.0	0.0	0.0	140.6	
4998	0	109.0	0.0	0.0	0.0	188.8	

train test split

In [19]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.70,random_state=7,strati
```

In [20]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
LG=LogisticRegression().fit(x_train,y_train)
y_pred=LG.predict(x_test)
print("Accuracy :\n",accuracy_score(y_test,y_pred))
print("Classification report :\n",classification_report(y_test,y_pred))
```

```
Accuracy :
 0.8666666666666667
Classification report :
               precision    recall  f1-score   support

           0       0.88      0.98      0.93      1288
           1       0.59      0.19      0.29       212

    accuracy                           0.87      1500
   macro avg       0.73      0.58      0.61      1500
weighted avg       0.84      0.87      0.84      1500
```

In [21]:

```
import pickle
pickle.dump(LG,open("churn.pkl","wb"))
```

scalling

In [22]:

```
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
x_train_scaled=scalar.fit_transform(x_train)
x_test_scaled=scalar.transform(x_test)
x_train_scaled=pd.DataFrame(x_train_scaled)
x_test_scaled=pd.DataFrame(x_test_scaled)
```

In [23]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
LG1=LogisticRegression().fit(x_train_scaled,y_train)
y_pred=LG1.predict(x_test_scaled)
print("Accuracy :\n",accuracy_score(y_test,y_pred))
print("Classification report :\n",classification_report(y_test,y_pred))
```

```
Accuracy :
 0.8666666666666667
Classification report :
               precision    recall  f1-score   support

           0       0.88      0.98      0.93      1288
           1       0.58      0.20      0.30       212

    accuracy                           0.87      1500
   macro avg       0.73      0.59      0.61      1500
weighted avg       0.84      0.87      0.84      1500
```

here is no effect of scalling on accuracy ,so we are only creating pkl file of without scalling model

In [ ]:

```

```