```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from keras.models import Sequential
from keras.layers import Dense
from enum import auto
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from matplotlib import pyplot
from sklearn.linear_model import LogisticRegression
import missingno as msno
from sklearn.metrics import roc_auc_score
from sklearn import metrics
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report,confusion_matrix, Confu
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridSearchCV
from sklearn.metrics import roc_curve
from matplotlib import pyplot
```

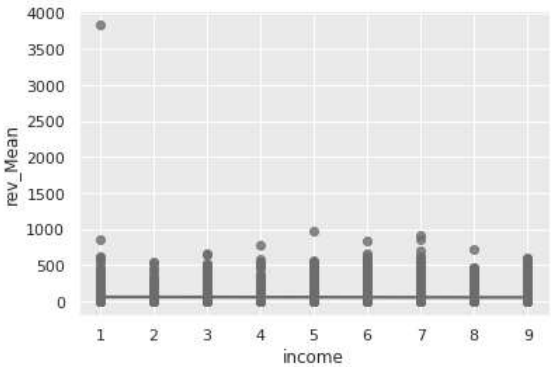
```
tel = pd.read_csv('Telecom_customer churn.csv')
data = tel.copy()
```

```
data.describe()
```

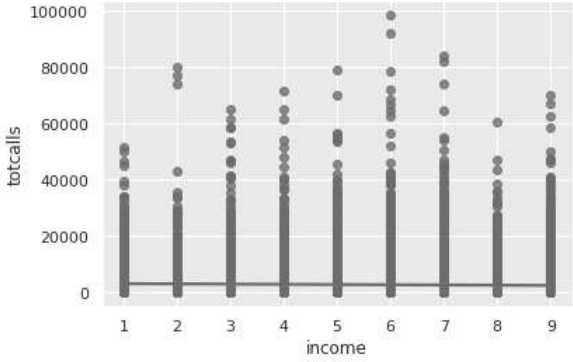
	rev_Mean	mou_Mean	totmrc_Mean	da_Mean	ovrmou_Mean	ovrrev_Mean	vceovr_Mean	datovr_Mean	roam_M
count	99643.000000	99643.000000	99643.000000	99643.000000	99643.000000	99643.000000	99643.000000	99643.000000	99643.000
mean	58.719985	513.559937	46.179136	0.888828	41.072247	13.559560	13.295062	0.261318	1.286
std	46.291677	525.168140	23.623489	2.177619	97.296150	30.500885	30.056089	3.126531	14.711
min	-6.167500	0.000000	-26.915000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	33.260000	150.750000	30.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	48.195000	355.500000	44.990000	0.247500	2.750000	1.000000	0.682500	0.000000	0.000
75%	70.750000	703.000000	59.990000	0.990000	42.000000	14.437500	14.025000	0.000000	0.235
max	3843.262500	12206.750000	409.990000	159.390000	4320.750000	1102.400000	896.087500	423.540000	3685.200

8 rows × 79 columns

```
 sns.set_theme(color_codes=True)
ax = sns.regplot(x="income", y="rev_Mean", data=data)
```

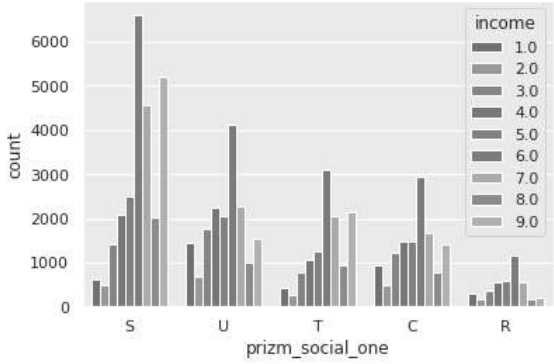


```
# totcalls vs income
sns.set_theme(color_codes=True)
ax = sns.regplot(x="income", y="totcalls", data=data)
```

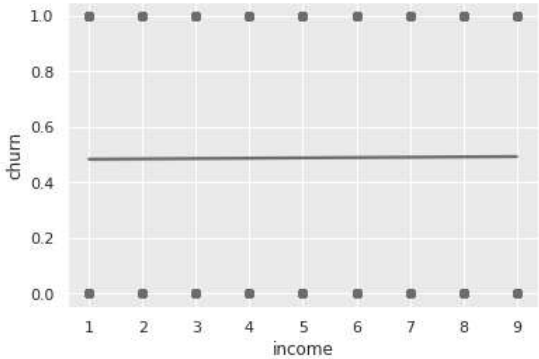



```
# income vs prizm
sns.countplot(x= "prizm_social_one", hue="income", data=data);
data.groupby('income')["prizm_social_one"].value_counts(normalize=True).unstack(fill_value=0)
```

prizm_social_one	C	R	S	T	U
income					
1.0	0.250998	0.082246	0.167953	0.114985	0.383817
2.0	0.233760	0.078236	0.230915	0.131816	0.325273
3.0	0.221295	0.064937	0.253401	0.139488	0.320878
4.0	0.199621	0.073944	0.282367	0.142199	0.301869
5.0	0.186486	0.075938	0.318458	0.158469	0.260649
6.0	0.164761	0.063975	0.368230	0.173518	0.229516
7.0	0.149550	0.048378	0.411441	0.185405	0.205225
8.0	0.157175	0.035722	0.412329	0.191468	0.203307
9.0	0.135546	0.020546	0.495577	0.202797	0.145534

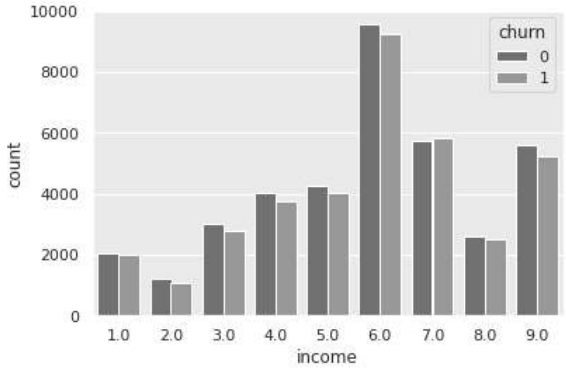



```
# totcalls vs income
sns.set_theme(color_codes=True)
ax = sns.regplot(x="income", y="churn", data=data)
```



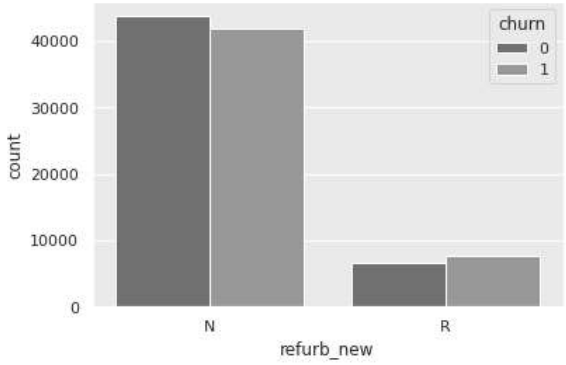
```
sns.countplot(x= "income", hue="churn", data=data);
data.groupby('income')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
income		
1.0	0.506819	0.493181
2.0	0.529204	0.470796
3.0	0.520926	0.479074
4.0	0.518357	0.481643
5.0	0.512988	0.487012
6.0	0.508882	0.491118
7.0	0.495645	0.504355
8.0	0.510113	0.489887
9.0	0.515554	0.484446



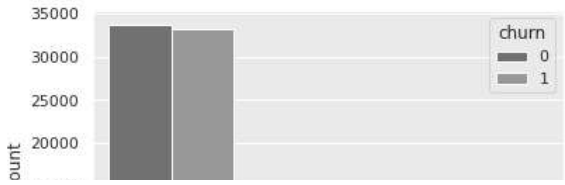

```
# refurb_new vs churn
sns.countplot(x= "refurb_new", hue="churn", data=data);
data.groupby('refurb_new')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
refurb_new		
N	0.510895	0.489105
R	0.465639	0.534361




```
#churn vs newcell
sns.countplot(x= "new_cell", hue="churn", data=data);
data.groupby('new_cell')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
new_cell		
N	0.503373	0.496627
U	0.503049	0.496951
Y	0.509715	0.490285

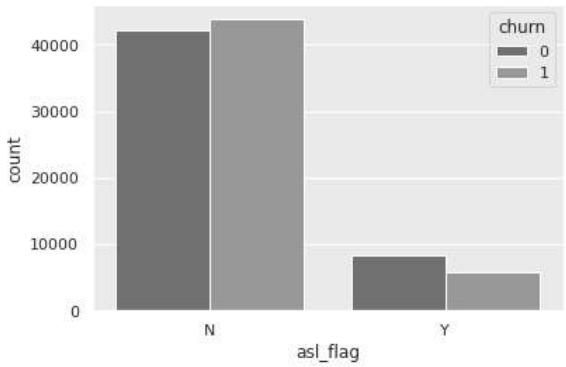



```
# crclscod vs churn
sns.countplot(x= "crclscod", hue="churn", data=data);
data.groupby('crclscod')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
crclscod		
A	0.482702	0.517298
A2	0.384437	0.615563
A3	0.333333	0.666667
AA	0.494316	0.505684
B	0.475669	0.524331
B2	0.467890	0.532110
BA	0.478724	0.521276
C	0.520604	0.479396
C2	0.527778	0.472222
C5	0.703704	0.296296
CA	0.531317	0.468683
CC	0.520000	0.480000
CY	0.547368	0.452632
D	0.550607	0.449393
D2	0.600000	0.400000
D4	0.627907	0.372093
D5	0.807018	0.192982
DA	0.551104	0.448896
E	0.592920	0.407080
E2	0.857143	0.142857
E4	0.693878	0.306122
EA	0.590279	0.409721
EC	0.645833	0.354167
EF	0.500000	0.500000
EM	0.421053	0.578947
G	0.460581	0.539419
GA	0.501779	0.498221
GY	0.441176	0.558824

```
# asl_flag vs churn
sns.countplot(x= "asl_flag", hue="churn", data=data);
data.groupby('asl_flag')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

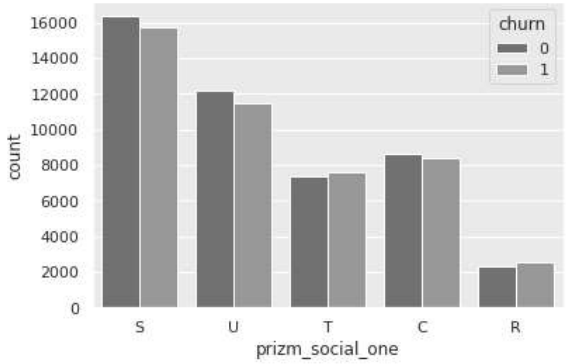
churn	0	1
asl_flag		
N	0.490402	0.509598
Y	0.590700	0.409300




```
# prizm_social_one vs churn
```

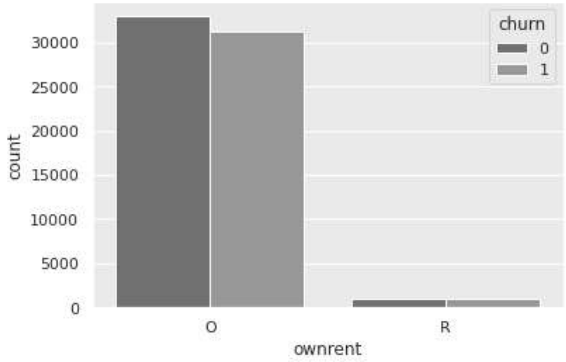
```
# prizm_social_one vs churn
sns.countplot(x= "prizm_social_one", hue="churn", data=data);
data.groupby('prizm_social_one')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
prizm_social_one		
C	0.509226	0.490774
R	0.472932	0.527068
S	0.509238	0.490762
T	0.491627	0.508373
U	0.515182	0.484818




```
# ownrent vs churn
sns.countplot(x= "ownrent", hue="churn", data=data);
data.groupby('ownrent')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
ownrent		
O	0.512803	0.487197
R	0.480597	0.519403

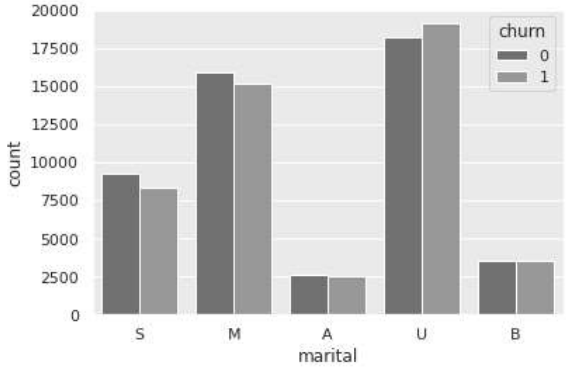



```
# dwellingtype vs churn
sns.countplot(x= "dwlltype", hue="churn", data=data);
data.groupby('dwlltype')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
dwlltype		
M	0.501345	0.498655

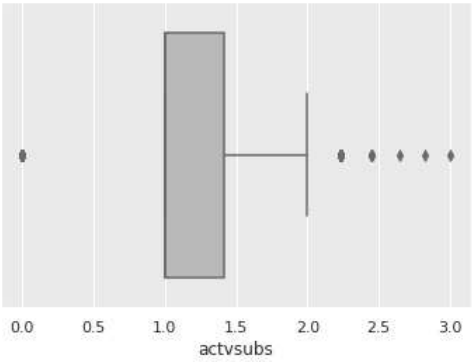
```
# martialstatus vs churn
sns.countplot(x= "marital", hue="churn", data=data);
data.groupby('marital')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
marital		
A	0.510311	0.489689
B	0.503092	0.496908
M	0.511336	0.488664
S	0.525330	0.474670
U	0.487344	0.512656




```
sns.boxplot(x = data['actvsubs'], palette = 'Set3')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fedaeb083d0>
```

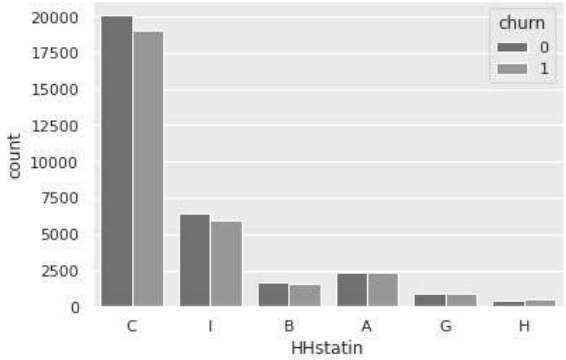



```
#infobase vs churn
sns.countplot(x= "infobase", hue="churn", data=data);
data.groupby('infobase')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
infobase		

```
#Household status indicator vs churn
sns.countplot(x= "HHstatin", hue="churn", data=data);
data.groupby('HHstatin')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
HHstatin		
A	0.500857	0.499143
B	0.520333	0.479667
C	0.513623	0.486377
G	0.497169	0.502831
H	0.477790	0.522210
I	0.519025	0.480975

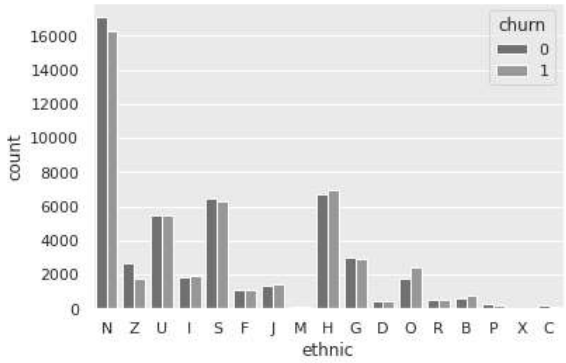



```
#dwelling size vs churn
sns.countplot(x= "dwllsize", hue="churn", data=data);
data.groupby('dwllsize')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
dwllsize		
A	0.515772	0.484228

```
#ethnicity vs churn
sns.countplot(x= "ethnic", hue="churn", data=data);
data.groupby('ethnic')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

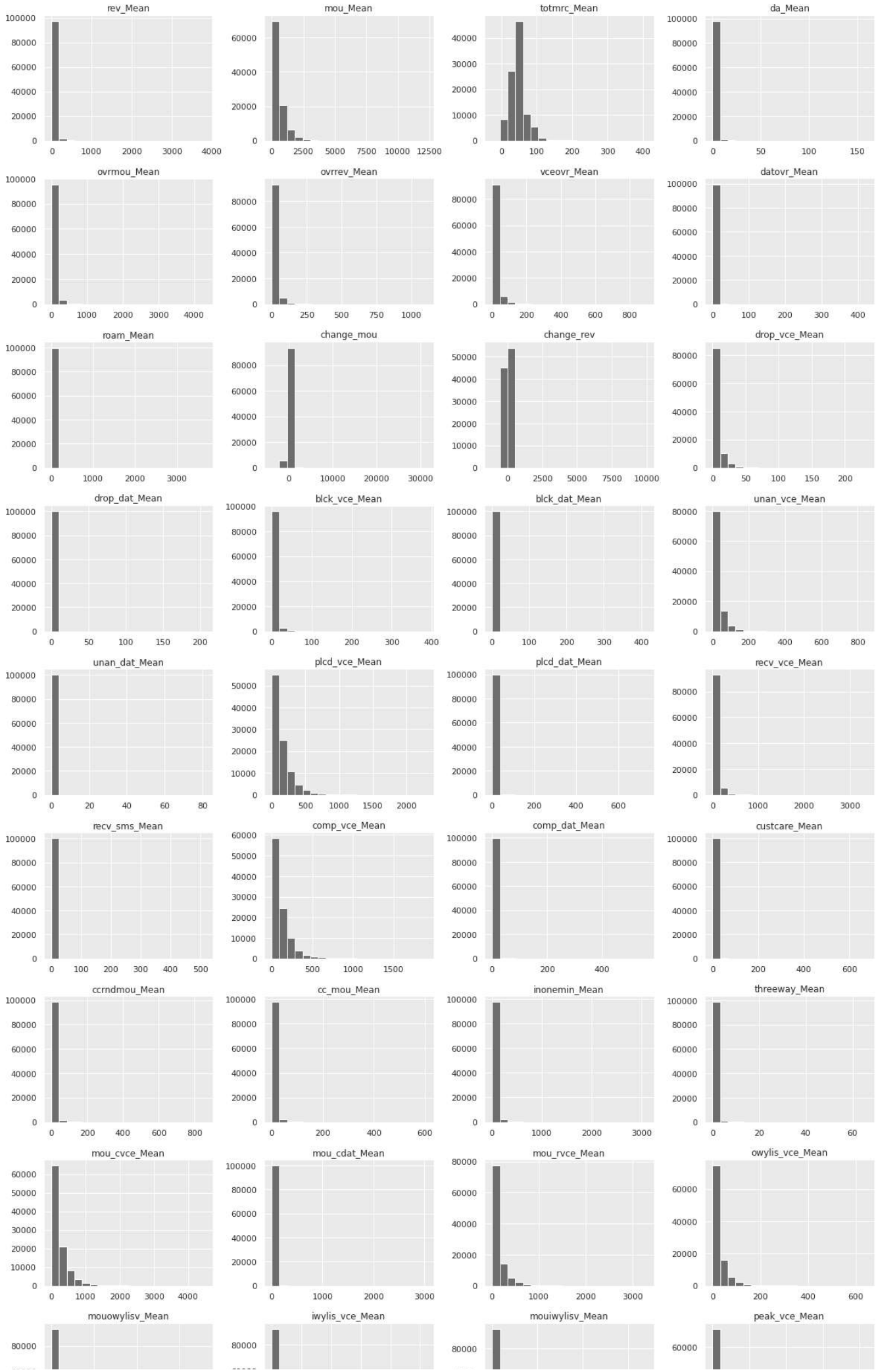
churn	0	1
ethnic		
B	0.446870	0.553130
C	0.684426	0.315574
D	0.466513	0.533487
F	0.496474	0.503526
G	0.504227	0.495773
H	0.491669	0.508331
I	0.487798	0.512202
J	0.483791	0.516209
M	0.463087	0.536913
N	0.511516	0.488484
O	0.419247	0.580753
P	0.625514	0.374486
R	0.468451	0.531549
S	0.508669	0.491331
U	0.499772	0.500228
X	0.674419	0.325581
Z	0.602938	0.397062

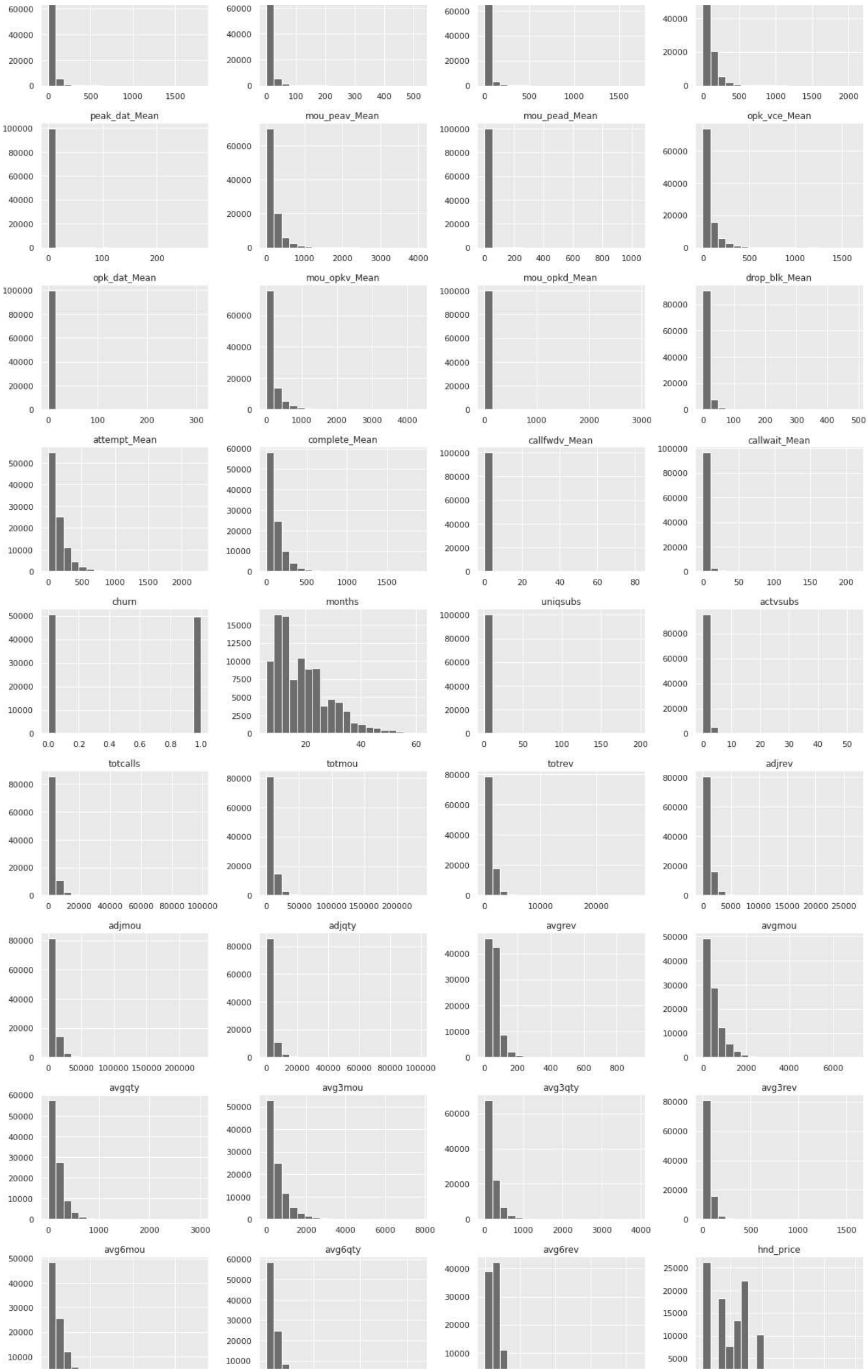


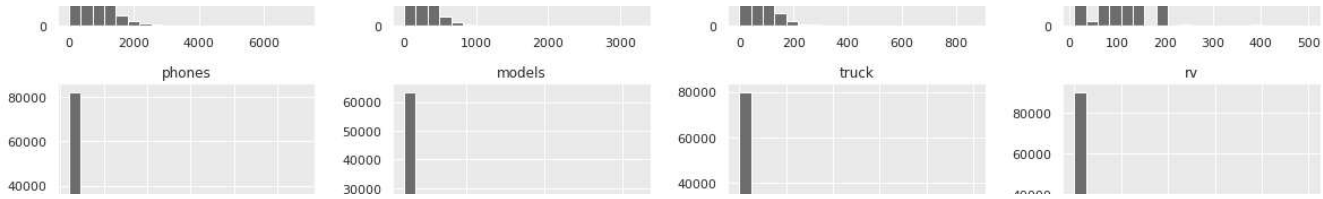

```
# credit card status vs churn
sns.countplot(x= "creditcd", hue="churn", data=data);
data.groupby('creditcd')["churn"].value_counts(normalize=True).unstack(fill_value=0)
```

churn	0	1
creditcd		
N	0.491719	0.508281

```
#Now we deal with Numberical Variables


data.iloc[:,:].hist(bins=20,figsize=(20,80),layout=(20,4));
```


rev_Mean
mou_Mean
totmrc_Mean
da_Mean
ovrmou_Mean
ovrrev_Mean
vceovr_Mean
datovr_Mean
roam_Mean
change_mou
change_rev
drop_vce_Mean
drop_dat_Mean
blck_vce_Mean
blck_dat_Mean
unan_vce_Mean
unan_dat_Mean
plcd_vce_Mean
plcd_dat_Mean
recv_vce_Mean
recv_sms_Mean
comp_vce_Mean
comp_dat_Mean
custcare_Mean
ccrndmou_Mean
cc_mou_Mean
inonemin_Mean
threeway_Mean
mou_cvce_Mean
mou_cdat_Mean
mou_rvce_Mean
owylis_vce_Mean
mouowylisv_Mean
iwylis_vce_Mean
mouiwylisv_Mean
peak_vce_Mean


peak_dat_Mean
mou_peav_Mean
mou_pead_Mean
opk_vce_Mean
opk_dat_Mean
mou_opkv_Mean
mou_opkd_Mean
drop_blk_Mean
attempt_Mean
complete_Mean
callfwdv_Mean
callwait_Mean
churn
months
uniqsubs
actvsubs
totcalls
totmou
totrev
adjrev
adjmou
adjqty
avgrev
avgmou
avgqty
avg3mou
avg3qty
avg3rev
avg6mou
avg6qty
avg6rev
hnd_price



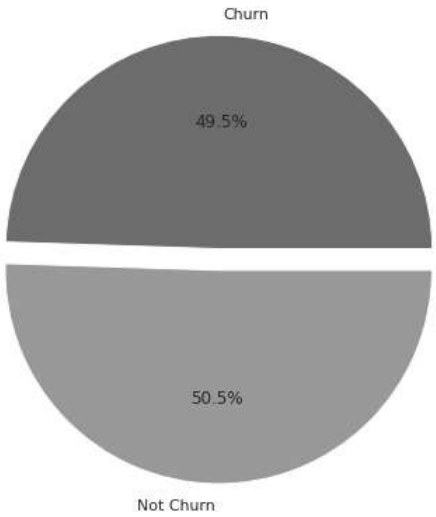

```
#understanding Data Set
```

```
#counting people who churn and who dont
stay = data[(data['churn'] ==0) ].count()[1]
churn = data[(data['churn'] ==1) ].count()[1]
print ("people who stay: "+ str(stay))
print ("people who churn: "+ str(churn))
```

```
people who stay: 50326
people who churn: 49317
```

```
label = ['Churn', 'Not Churn']
size=[(churn/(stay+churn))*100,(stay/(stay+churn))*100]
explode=(0,0.1)
fig = plt.figure(figsize =(10, 7))
plt.pie(size, labels = label,explode=explode,autopct='%1.1f%%')
fig.legend(label,
          title ="Churn ratio",
          loc ="center left",
          bbox_to_anchor =(1, 0, 0.5, 1))
```

```
<matplotlib.legend.Legend at 0x7fedb1874ed0>
```




```
#Now we start dealing with the missing values
```

```
#columns with missing values
missing = data.isnull().sum().sort_values(ascending = False).head(44)
miss_percentage = (missing/len(data))*100
# Percentage of missing values
pd.DataFrame({'No. missing values': missing, '% of missing data': miss_percentage.values})
```

	No. missing values	% of missing data
numbcars	49366	49.366
dwllsize	38308	38.308
HHstatin	37923	37.923
ownrent	33706	33.706
dwlltype	31909	31.909
lor	30190	30.190
income	25436	25.436
adults	23019	23.019
infobase	22079	22.079
hnd_webcap	10189	10.189
prizm_social_one	7388	7.388
avg6rev	2839	2.839
avg6mou	2839	2.839
avg6qty	2839	2.839
ethnic	1732	1.732
rv	1732	1.732
marital	1732	1.732
forgntvl	1732	1.732
kid0_2	1732	1.732
kid3_5	1732	1.732
kid6_10	1732	1.732
kid16_17	1732	1.732
creditcd	1732	1.732
truck	1732	1.732
kid11_15	1732	1.732
change_rev	891	0.891
change_mou	891	0.891
hnd_price	847	0.847
mou_Mean	357	0.357
rev_Mean	357	0.357
da_Mean	357	0.357
datovr_Mean	357	0.357
totmrc_Mean	357	0.357
ovrmou_Mean	357	0.357
ovrrev_Mean	357	0.357
vceovr_Mean	357	0.357
roam_Mean	357	0.357

```
data.corr
```

```
<bound method DataFrame.corr of        rev_Mean  mou_Mean  totmrc_Mean  da_Mean  ovrmou_Mean  ovrrev_Mean  \
0       23.9975    219.25       22.500   0.2475         0.00          0.0
1       57.4925    482.75       37.425   0.2475        22.75          9.1
2       16.9900     10.25       16.990   0.0000         0.00          0.0
3       38.0000      7.50       38.000   0.0000         0.00          0.0
4       55.2300    570.50       71.980   0.0000         0.00          0.0
...         ...       ...          ...      ...          ...          ...
99995       NaN       NaN          NaN      NaN          NaN          NaN
99996       NaN       NaN          NaN      NaN          NaN          NaN
99997       NaN       NaN          NaN      NaN          NaN          NaN
99998       NaN       NaN          NaN      NaN          NaN          NaN
99999       NaN       NaN          NaN      NaN          NaN          NaN

       vceovr_Mean  datovr_Mean  roam_Mean  change_mou  ...  forgntvl  ethnic  \
0              0.0          0.0        0.0     -157.25  ...       0.0       N
1              9.1          0.0        0.0      532.25  ...       0.0       Z
```

```
2              0.0          0.0        0.0       -4.25  ...       0.0       N
3              0.0          0.0        0.0       -1.50  ...       0.0       U
4              0.0          0.0        0.0       38.50  ...       0.0       I
...            ...          ...        ...         ...  ...       ...     ...
99995          NaN          NaN        NaN         NaN  ...       0.0       S
99996          NaN          NaN        NaN         NaN  ...       0.0       N
99997          NaN          NaN        NaN         NaN  ...       0.0       U
99998          NaN          NaN        NaN         NaN  ...       0.0       S
99999          NaN          NaN        NaN         NaN  ...       0.0       H

       kid0_2  kid3_5  kid6_10  kid11_15  kid16_17  creditcd  eqpdays  \
0           U       U        U         U         U         Y    361.0
1           U       U        U         U         U         Y    240.0
2           U       Y        U         U         U         Y   1504.0
3           Y       U        U         U         U         Y   1812.0
4           U       U        U         U         U         Y    434.0
...       ...     ...      ...       ...       ...       ...      ...
99995       U       U        U         Y         U         Y    773.0
99996       U       U        Y         Y         Y         Y    835.0
99997       Y       Y        U         U         U         N    433.0
99998       U       U        U         U         U         N     75.0
99999       U       U        U         U         U         N      5.0

       Customer_ID
0          1000001
1          1000002
2          1000003
3          1000004
4          1000005
...            ...
99995      1099996
99996      1099997
99997      1099998
99998      1099999
99999      1100000

[100000 rows x 100 columns]>
```

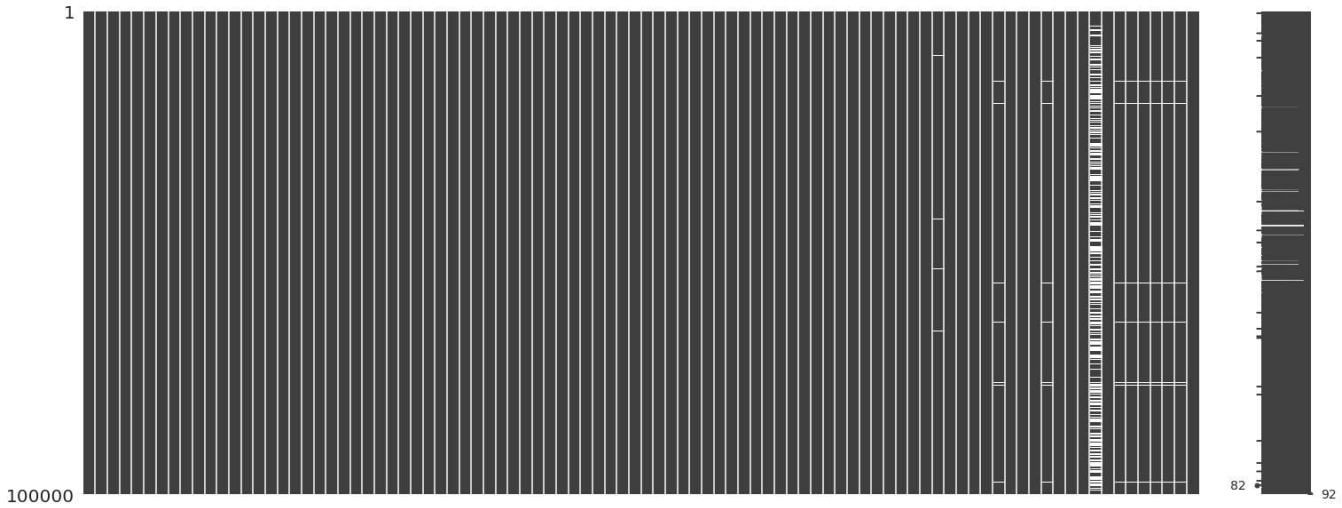
```
data.drop(["Customer_ID"], axis = 1, inplace=True)
data.shape
```

```
(100000, 99)
```

```
# We dropped the columns that seem to have no significant contribution to the model.
data.drop(['prizm_social_one','infobase','crclscod'],axis=1,inplace=True)
data.drop(["area"], axis = 1, inplace=True)
data.drop(["ethnic"], axis = 1, inplace=True)
data.drop(["forgntvl"], axis = 1, inplace=True)
data.drop(["asl_flag"], axis = 1, inplace=True)
```

```
data['numbcars']=data['numbcars'].fillna(data['numbcars'].mean())
data['dwllsize']=data['dwllsize'].fillna('UNKW')
data['ownrent']=data['ownrent'].fillna('UNKW')
data['dwlltype']=data['dwlltype'].fillna('UNKW')
data['lor']=data['lor'].fillna('UNKW')
data['truck']=data['truck'].fillna(0)
data['income']=data['income'].fillna(data['income'].mean())
data['adults']=data['adults'].fillna(data['adults'].mean())
data['hnd_webcap']=data['hnd_webcap'].fillna('UNKW')
data['avg6qty']=data['avg6qty'].fillna(data['avg6qty'].mean())
data['avg6rev']=data['avg6rev'].fillna(data['avg6rev'].mean())
data['avg6mou']=data['avg6mou'].fillna(data['avg6mou'].mean())
data['change_mou']=data['change_mou'].fillna(data['change_mou'].mean())
data['change_rev']=data['change_rev'].fillna(data['change_rev'].mean())
data['rev_Mean']=data['rev_Mean'].fillna(data['rev_Mean'].mean())
data['totmrc_Mean']=data['totmrc_Mean'].fillna(data['totmrc_Mean'].mean())
data['da_Mean']=data['da_Mean'].fillna(data['da_Mean'].mean())
data['ovrmou_Mean']=data['ovrmou_Mean'].fillna(data['ovrmou_Mean'].mean())
data['ovrrev_Mean']=data['ovrrev_Mean'].fillna(data['ovrrev_Mean'].mean())
data['vceovr_Mean']=data['vceovr_Mean'].fillna(data['vceovr_Mean'].mean())
data['datovr_Mean']=data['datovr_Mean'].fillna(data['datovr_Mean'].mean())
data['roam_Mean']=data['roam_Mean'].fillna(data['roam_Mean'].mean())
data['mou_Mean']=data['mou_Mean'].fillna(data['mou_Mean'].mean())
```

```
msno.matrix(data);
```




```
data.dropna(inplace=True)
data.shape
```

```
(61549, 92)
```

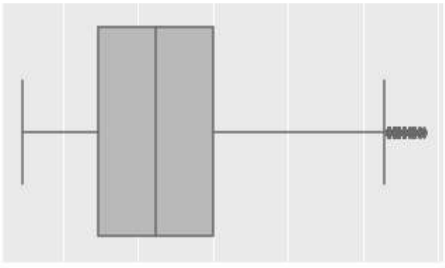
```
obj_col = data.select_dtypes(include = 'object').columns
# obj_col['churn'] = data['churn']
obj_col
```

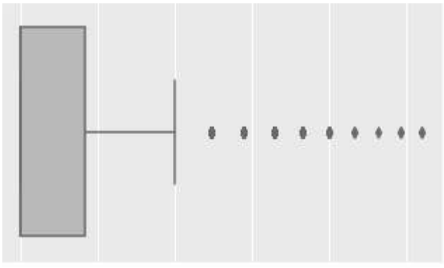
```
Index(['new_cell', 'dualband', 'refurb_new', 'hnd_webcap', 'ownrent', 'lor',
       'dwlltype', 'marital', 'HHstatin', 'dwllsize', 'kid0_2', 'kid3_5',
       'kid6_10', 'kid11_15', 'kid16_17', 'creditcd'],
      dtype='object')
```

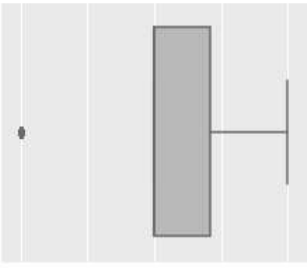
```
num_ft = ['months', 'uniqsubs', 'actvsubs', 'totcalls', 'avg3qty', 'avg3rev','rev_Mean', 'mou_Mean', 'totmrc_Mean', 'da_Mean', 'ov
                      'roam_Mean', 'change_mou', 'change_rev', 'drop_vce_Mean', 'drop_dat_Mean', 'blck_vce_Mean', 'blck_dat_Mean',
                      'plcd_vce_Mean', 'plcd_dat_Mean', 'recv_vce_Mean', 'recv_sms_Mean', 'custcare_Mean', 'ccrndmou_Mean', 'three
                      'mou_cdat_Mean', 'mou_rvce_Mean', 'owylis_vce_Mean', 'mouowylisv_Mean', 'iwylis_vce_Mean', 'mouiwylisv_Mean'
                      'mou_peav_Mean', 'mou_pead_Mean', 'opk_vce_Mean', 'opk_dat_Mean', 'mou_opkv_Mean', 'drop_blk_Mean', 'callfwd
                      'totrev', 'avgrev', 'avgmou', 'avgqty', 'avg6mou', 'avg6rev', 'hnd_price', 'phones', 'models', 'truck', 'rv'
```

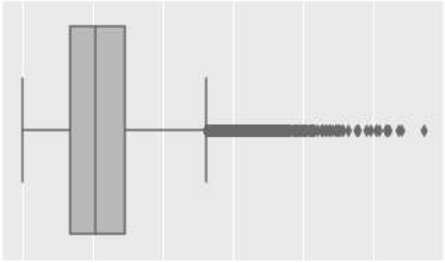
```
for i in num_ft:
    f_sqrt= (lambda x: np.sqrt(abs(x)) if (x>=1) or (x<=-1) else x)
    data[i] = data[i].apply(f_sqrt)
```

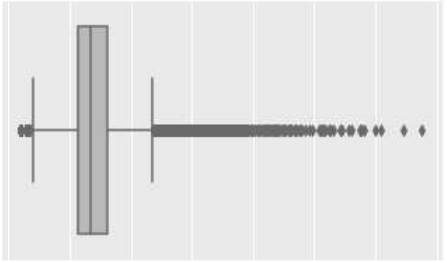
```
# Box plots of all numerical variables
fig, ax = plt.subplots(15, 4, figsize = (20, 50))
ax = ax.flatten()
for i, c in enumerate(num_ft):
    sns.boxplot(x = data[c], ax = ax[i], palette = 'Set3')
# plt.suptitle('Box Plot', fontsize = 25)
fig.tight_layout()
```

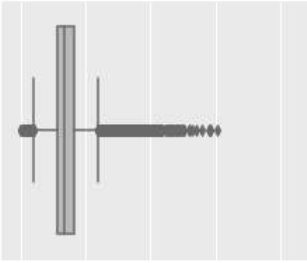

3 4 5 6 7 8
months

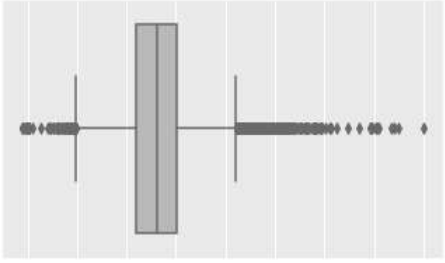

1.0 1.5 2.0 2.5 3.0 3.5
uniqsubs

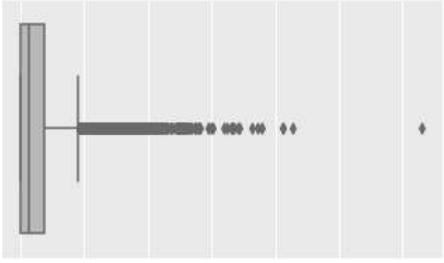

0.0 0.5 1.0 1.5 2.0
actvsubs

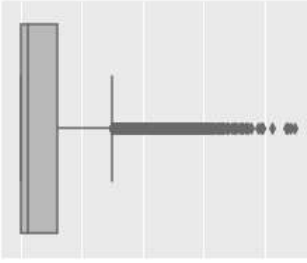

0 10 20 30 40 50
avg3qty


0 5 10 15 20 25 30 35
avg3rev

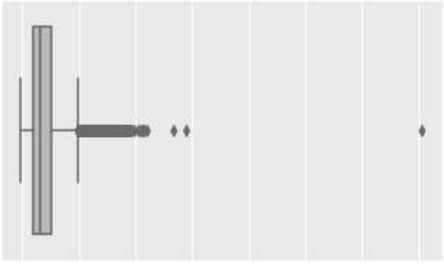

0 10 20 30 40
rev_Mean

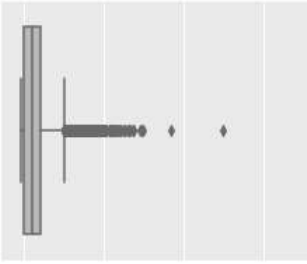

0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
totmrc_Mean

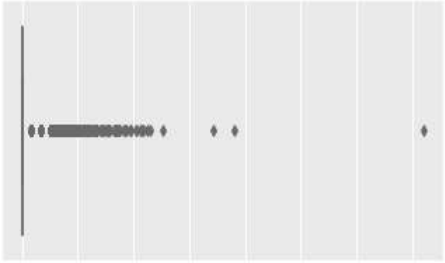

0 2 4 6 8 10 12
da_Mean

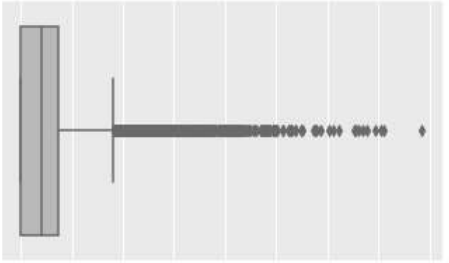

0 10 20 30 40
ovrmou_Mean

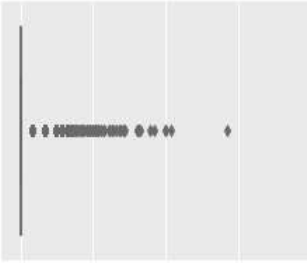
0 10 20 30 40 50 60
roam_Mean

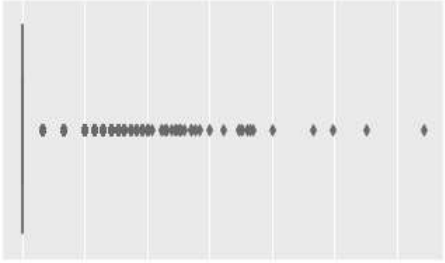

0 25 50 75 100 125 150 175
change_mou

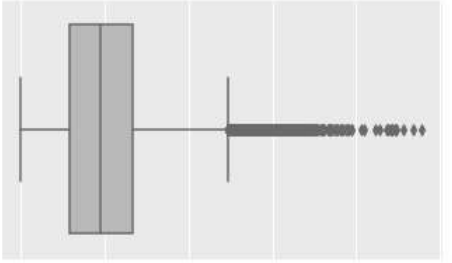

0 20 40 60
change_rev

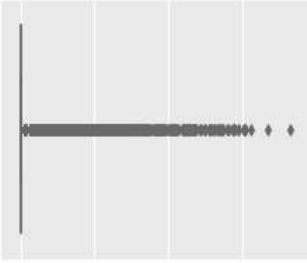

0 2 4 6 8 10 12 14
drop_dat_Mean

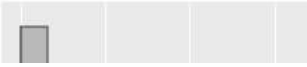

0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
blck_vce_Mean

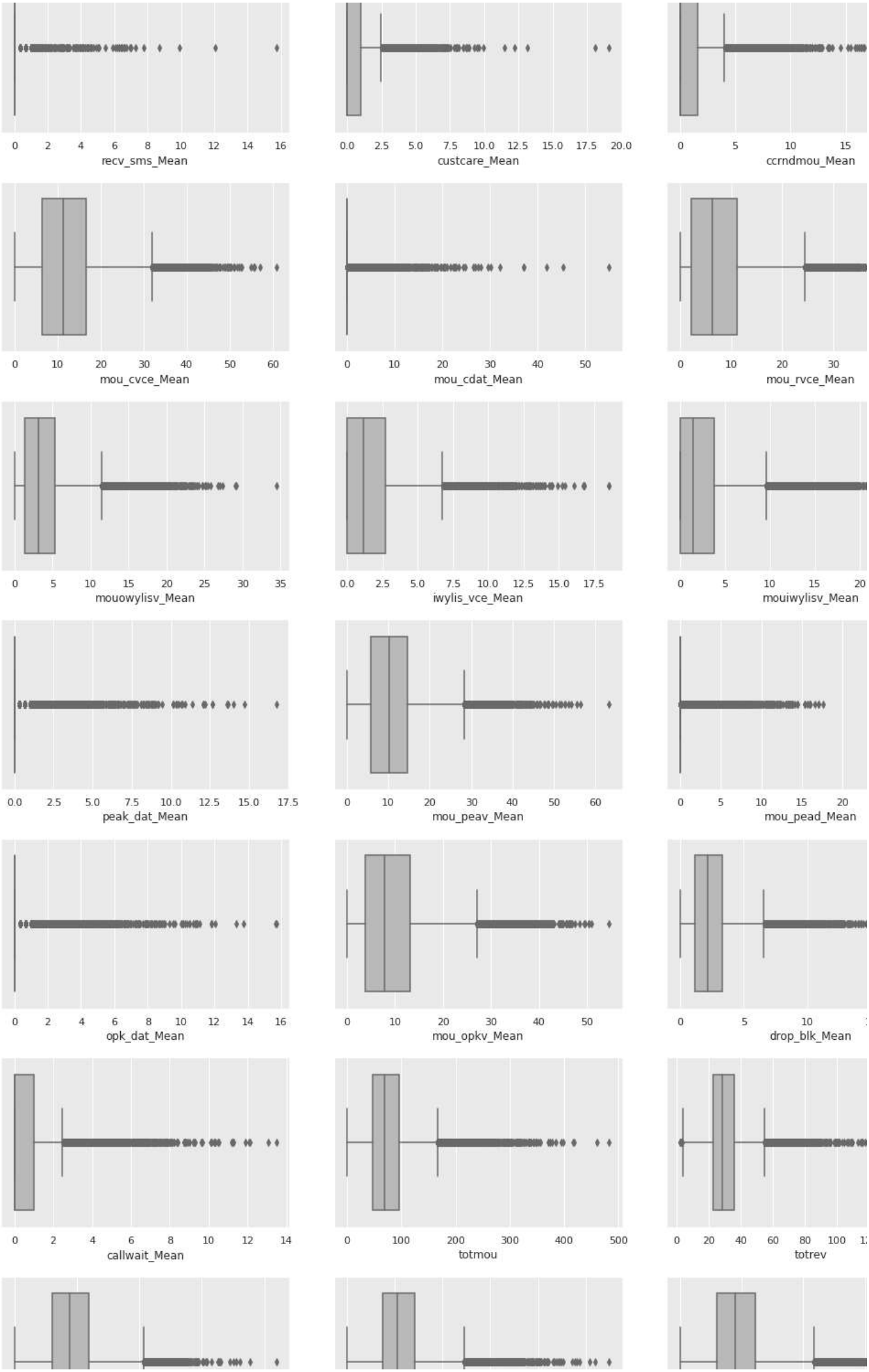

0 2 4 6
blck_dat_Mean

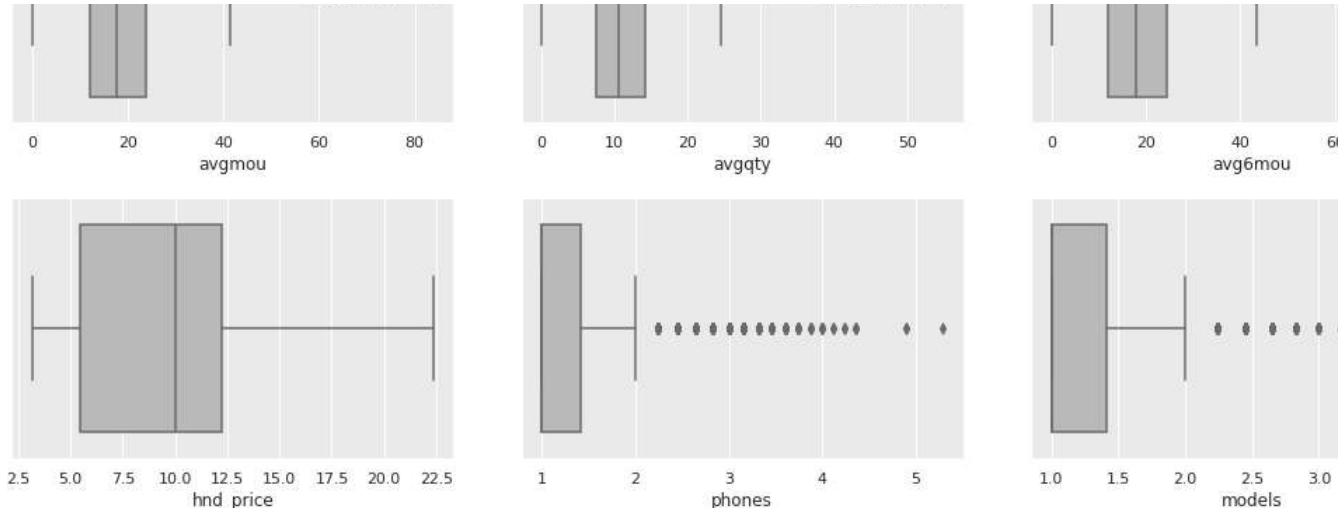

0 1 2 3 4 5 6
unan_dat_Mean


0 10 20 30 40 50
plcd_vce_Mean


0 5 10 15
plcd_dat_Mean




recv_sms_Mean
custcare_Mean
ccrndmou_Mean
mou_cvce_Mean
mou_cdat_Mean
mou_rvce_Mean
mouowylisv_Mean
iwylis_vce_Mean
mouiwylisv_Mean
peak_dat_Mean
mou_peav_Mean
mou_pead_Mean
opk_dat_Mean
mou_opkv_Mean
drop_blk_Mean
callwait_Mean
totmou
totrev

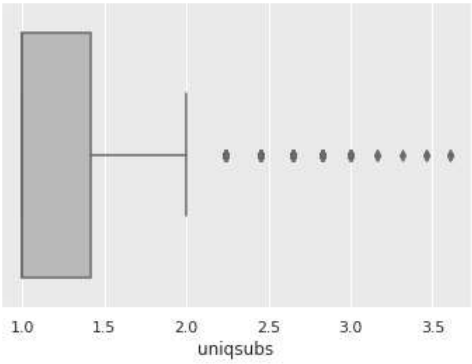



```
#Now we treat the outliers in our numerical variables
```

```
def detect_outliers(data,features):
    outlier_index = []

    for c in features:
        # 1st quartile
        Q1 = np.percentile(data[c],25)
        # 3rd quartile
        Q3 = np.percentile(data[c],75)
        # IQR
        IQR = Q3 - Q1
        # Outlier step
        outlier_step = IQR * 1.5
        outlier_list_col = data[(data[c] < Q1 - outlier_step) | (data[c] > Q3 + outlier_step)].index
        outlier_index.extend(outlier_list_col)

    outlier_index = Counter(outlier_index)
    multiple_outliers = list(i for i, v in outlier_index.items() if v > 2)

    return outlier_index
```

```
sns.boxplot(x = data['uniqsubs'], palette = 'Set3')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fedb366a1d0>
```




```
num_col = data.select_dtypes(include = 'number').columns
num_cols = list(num_col)
num_cols.remove('churn')
```

```
# one-hot encoding for variables with more than 2 categories
datac = data.copy()
datac = pd.get_dummies(datac, drop_first=True, columns = obj_col, prefix = obj_col)
datac
```

	rev_Mean	mou_Mean	totmrc_Mean	da_Mean	ovrmou_Mean	ovrrev_Mean	vceovr_Mean	datovr_Mean	roam_Mean	change_mou
0	4.898724	14.807093	4.743416	0.247500	0.000000	0.00000	0.000000	0.000000	0.000000	12.539936
2	4.121893	3.201562	4.121893	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	2.061553
3	6.164414	2.738613	6.164414	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	1.224745
4	7.431689	23.885142	8.484103	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	6.204837
5	9.070557	36.224991	8.660254	1.112430	0.000000	0.00000	0.000000	0.000000	0.000000	12.519984
...	...	...	...	...	...	...	...	...	...	...
99990	5.476313	13.564660	5.476313	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	3.732803
99991	7.662897	22.661861	6.795523	0.888828	6.408763	13.55956	13.295062	0.261318	1.134198	3.732803
99992	7.662897	22.661861	6.795523	0.888828	6.408763	13.55956	13.295062	0.261318	1.134198	3.732803

```
#corr
```

```
c = data.corr()['churn'].abs()
sc = c.sort_values()
sc
```

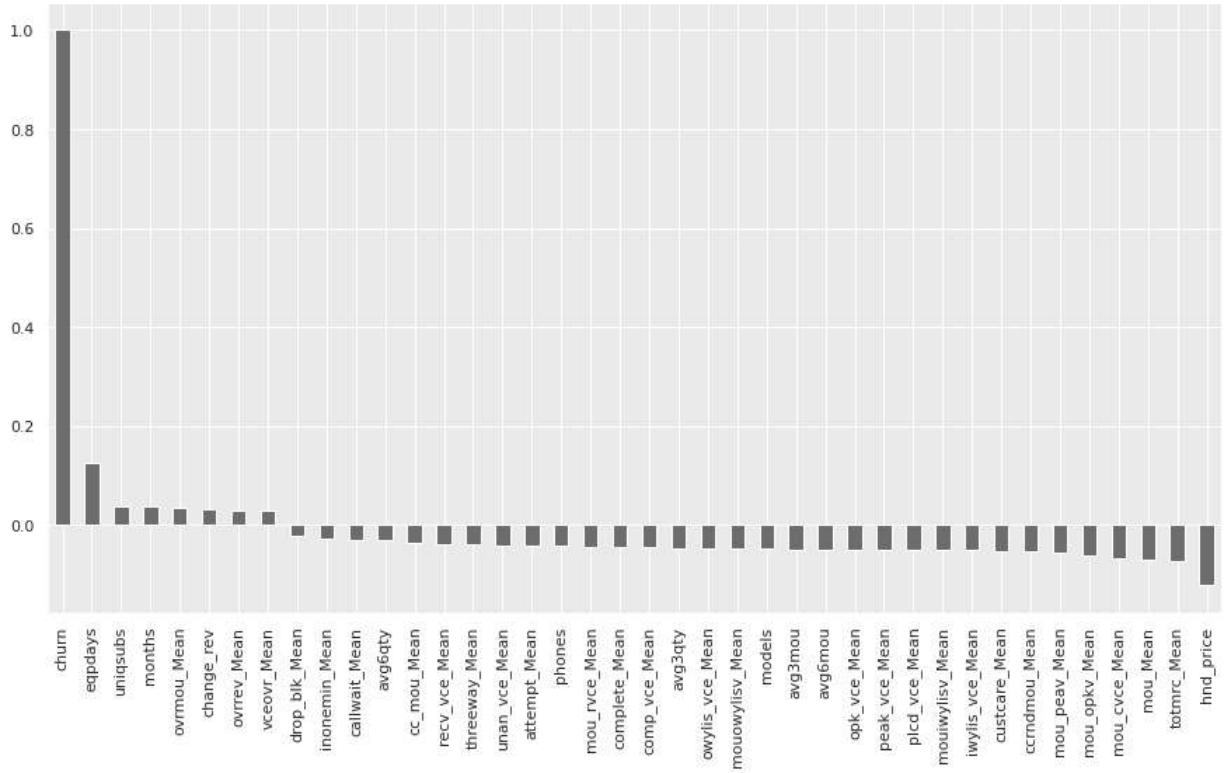
```
truck            0.000249
numbcars         0.000563
recv_sms_Mean    0.001460
income           0.001672
adjrev           0.002023
                   ...
mou_Mean         0.069320
totmrc_Mean      0.072302
hnd_price        0.120929
eqpdays          0.124737
churn            1.000000
Name: churn, Length: 76, dtype: float64
```

```
a = dict(sc.tail(40))
b = a.keys()
print(sorted(b))
```

```
['attempt_Mean', 'avg3mou', 'avg3qty', 'avg6mou', 'avg6qty', 'callwait_Mean', 'cc_mou_Mean', 'ccrndmou_Mean', 'change_rev',
```

```
# Get Correlation of "churn" with other variables:
plt.figure(figsize=(15,8))
datac[b].corr()['churn'].sort_values(ascending = False).plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fedad1805d0>
```



```
 #modelling our dataset


# generate dataset
X = datac.drop('churn', axis=1)
y = datac['churn']


sc = StandardScaler()
X_sc = sc.fit_transform(X)
X = pd.DataFrame(X_sc, columns = X.columns)

X_num = X[num_cols]
X_obj = X.drop(num_cols, axis=1)



transformer = FunctionTransformer(np.log1p)
X_log = transformer.fit_transform(X_num)
X_logged_df = pd.DataFrame(X_log, columns = X_num.columns)
X_logged_df.shape
```

```
(61549, 75)
```

```
X_logged_df2 = X_logged_df.fillna(0)
X_logged_df2.shape
```

```
(61549, 75)
```

```
Xs = [X_obj, X_logged_df2]
X_final = pd.concat(Xs, axis=1)
X_final.shape
```

```
(61549, 132)
```

```
# define feature selection
fs = SelectKBest(score_func=f_classif, k='all')

# apply feature selection
# X_final = X_final.astype('long')
X_selected = fs.fit_transform(X_final, y)
print(X_selected.shape)
```

```
(61549, 132)
```

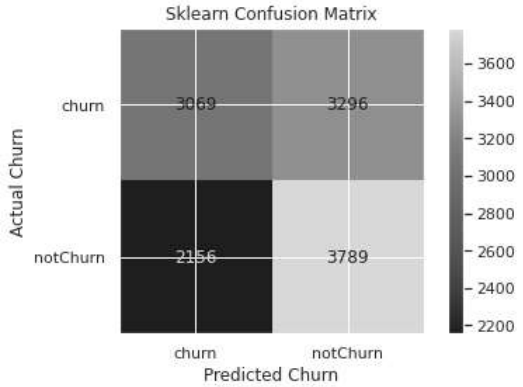
```
# dependent and independent variables were determined.
# X = datac.drop('churn', axis=1)
# y = data['churn']

X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
print("X_train",len(X_train))
print("X_test",len(X_test))
print("y_train",len(y_train))
print("y_test",len(y_test))
```

```
X_train 49239
X_test 12310
y_train 49239
y_test 12310
```

```
#Defining the modelling function
def modeling(alg, alg_name, params={}):
    model = alg(**params) #Instantiating the algorithm class and unpacking parameters if any
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    #Performance evaluation
    def print_scores(alg, y_true, y_pred):
        print(alg_name)
        acc_score = accuracy_score(y_true, y_pred)
        print("accuracy: ",acc_score)
        pre_score = precision_score(y_true, y_pred)
        print("precision: ",pre_score)
        rec_score = recall_score(y_true, y_pred)
        print("recall: ",rec_score)
        f_score = f1_score(y_true, y_pred, average='weighted')
```

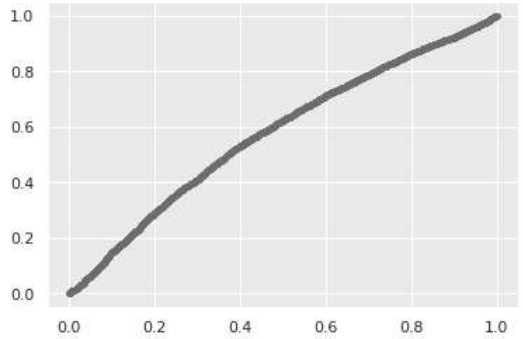
```
        print("f1_score: ",f_score)
    print_scores(alg, y_test, y_pred)


    cm = confusion_matrix(y_test, y_pred)
    cmd_obj = ConfusionMatrixDisplay(cm, display_labels=['churn', 'notChurn'])
    cmd_obj.plot()

    cmd_obj.ax_.set(
                    title='Sklearn Confusion Matrix',
                    xlabel='Predicted Churn',
                    ylabel='Actual Churn')


    return model


def showroc(model, model_name):
    model.fit(X_train,y_train)
    md_probs = model.predict_proba(X_test)
    md_probs = md_probs[:,1]
    md_auc = roc_auc_score(y_test, md_probs)
    md_fpr, md_tpr, _ = roc_curve(y_test, md_probs)
    plt.plot(md_fpr, md_tpr, marker='.', label=model_name)
    print(md_auc)


GNB_model = modeling(GaussianNB, 'Gaussian Naive Bayes')
```

```
Gaussian Naive Bayes
accuracy:  0.5571080422420797
precision:  0.5347918136908962
recall:  0.6373423044575274
f1_score:  0.5547008693151466
```



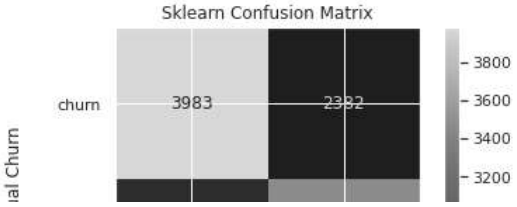

```
showroc(GNB_model, "Gaussian Naive Bayes")
```

```
0.5781628134833776
```



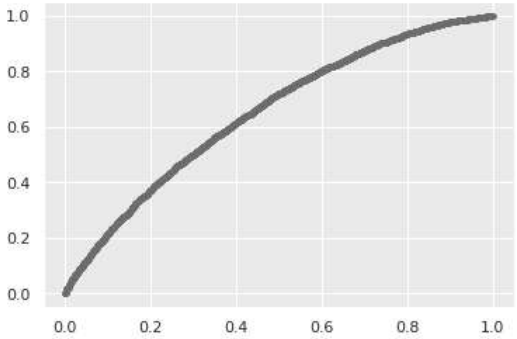

```
Ada_boost_model = modeling(AdaBoostClassifier, 'Ada Boost')
```

```
Ada Boost
accuracy:  0.6056864337936637
precision:  0.5931682322801025
recall:  0.5841883936080741
f1_score:  0.6055667915986767
```



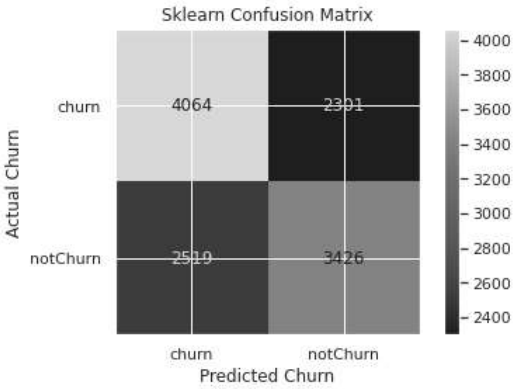

```
showroc(Ada_boost_model, "Ada Boost")
```

```
0.6522134491545636
```



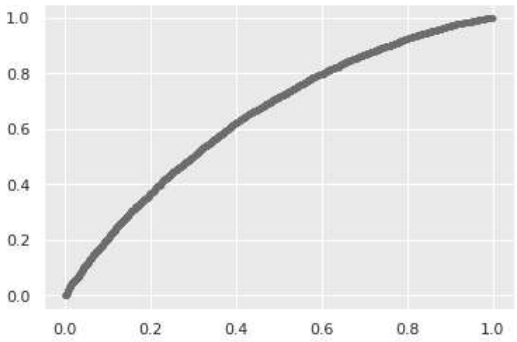

```
a={'C':1.0, 'class_weight':None, 'dual':False, 'fit_intercept':True,
                    'intercept_scaling':1, 'l1_ratio':None,
                    'multi_class':'auto', 'n_jobs':None, 'penalty':'l2',
                    'random_state':None, 'solver':'lbfgs', 'tol':0.0001, 'verbose':0,
                    'warm_start':False,'max_iter':1000}
Log_reg_model = modeling(LogisticRegression, 'Logistic Regression',a)
```

```
Logistic Regression
accuracy:  0.6084484159220146
precision:  0.5982189628077528
recall:  0.576282590412111
f1_score:  0.6080880711811602
```



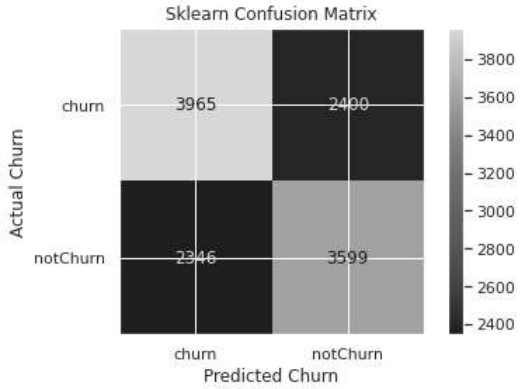

```
showroc(Log_reg_model, "Logistic Regression")
```

```
0.6493044053337843
```



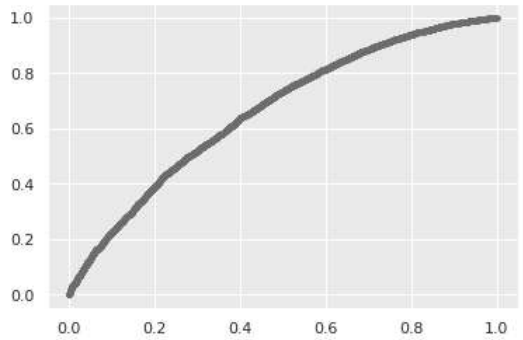

```
j={'max_depth':2,'n_estimators':100, 'random_state':0}
GB model = modeling(GradientBoostingClassifier, 'Gradient Boosting',j)
```

```
Gradient Boosting
accuracy:  0.614459788789602
precision:  0.5999333222203701
recall:  0.6053826745164004
f1_score:  0.6145101171174119
```



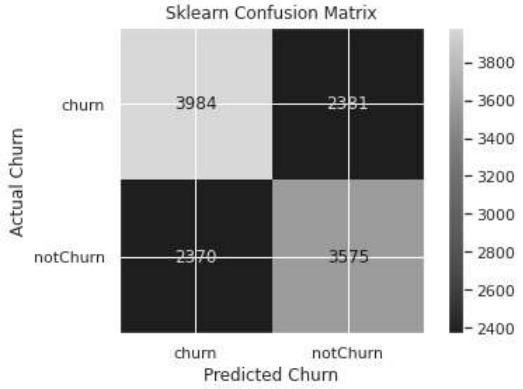

```
showroc(GB_model, "Graident Boosting")
```

```
0.6641704892385489
```




```
g={'base_score':0.5,'colsample_bylevel':1,'colsample_bynode':1,'colsample_bytree':1,'gamma':0,'importance_type':'gain','gpu_id':0,
XGB_model = modeling(XGBClassifier, 'XG Boosting',g)
```

```
XG Boosting
accuracy:  0.6140536149471973
precision:  0.6002350570852921
recall:  0.6013456686291001
f1_score:  0.6140650860863516
```

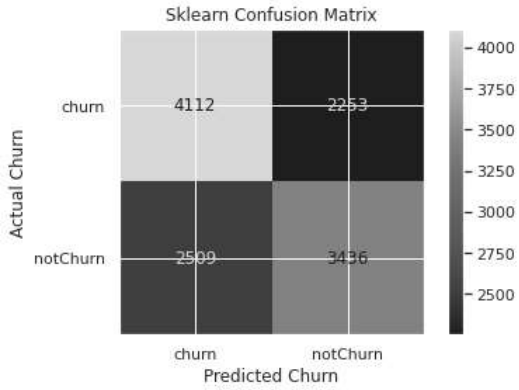



```
showroc(XGB_model, "XG Boosting")
```

```
0.658441897017502
```

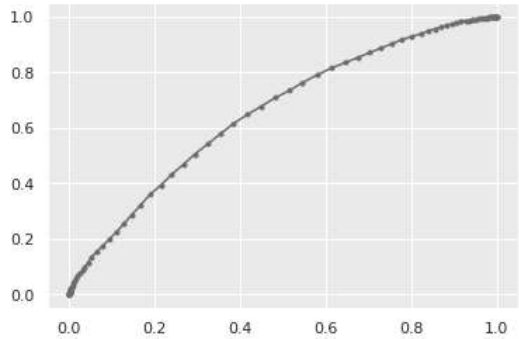
```
# Running RandomForestClassifier model
v={'bootstrap':True, 'ccp_alpha':0.0, 'class_weight':None,
                        'criterion':'gini', 'max_depth':None, 'max_features':'auto',
                        'max_leaf_nodes':None, 'max_samples':None,
                        'min_impurity_decrease':0.0,
                        'min_samples_leaf':1, 'min_samples_split':2,
                        'min_weight_fraction_leaf':0.0, 'n_estimators':100,
                        'n_jobs':-1, 'oob_score':False, 'random_state':123, 'verbose':0,
                        'warm_start':False}
RF_model = modeling(RandomForestClassifier, 'Random Forest',v)
```

```
Random Forest
accuracy:  0.6131600324939074
precision:  0.6039725786605731
recall:  0.5779646761984861
f1_score:  0.6127169204633086
```



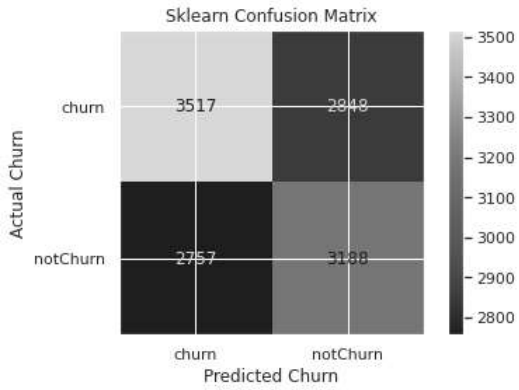

```
showroc(RF_model, "Random Forest")
```

```
0.655120682718055
```



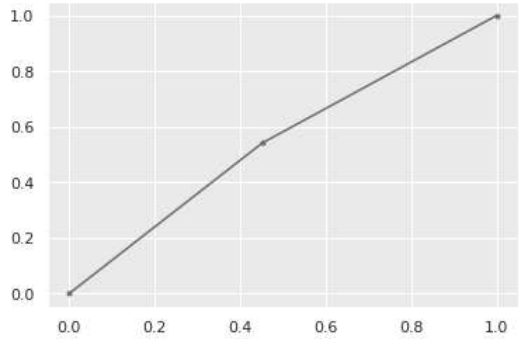

```
#Decision tree
dt_model = modeling(DecisionTreeClassifier, "Decision Tree Classification")
```

```
Decision Tree Classification
accuracy:  0.5446791226645004
precision:  0.5281643472498343
recall:  0.5362489486963835
f1_score:  0.544769144750506
```




```
showroc(dt_model, "Decision Tree Classification")
```

```
0.5448927951099268
```




```
# define the  model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu', kernel_initializer='he_normal'))
model.add(Dense(256, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
fittedm=model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=2,validation_data=(X_test, y_test))
# evaluate the keras model
_, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 54/100
3078/3078 - 13s - loss: 0.0887 - accuracy: 0.9691 - val_loss: 3.6787 - val_accuracy: 0.5534 - 13s/epoch - 4ms/step
Epoch 55/100
3078/3078 - 14s - loss: 0.0861 - accuracy: 0.9706 - val_loss: 3.7452 - val_accuracy: 0.5474 - 14s/epoch - 5ms/step
Epoch 56/100
3078/3078 - 12s - loss: 0.0877 - accuracy: 0.9707 - val_loss: 3.6489 - val_accuracy: 0.5566 - 12s/epoch - 4ms/step
Epoch 57/100
3078/3078 - 12s - loss: 0.0866 - accuracy: 0.9708 - val_loss: 3.7335 - val_accuracy: 0.5535 - 12s/epoch - 4ms/step
Epoch 58/100
3078/3078 - 14s - loss: 0.0842 - accuracy: 0.9723 - val_loss: 3.6259 - val_accuracy: 0.5569 - 14s/epoch - 5ms/step
Epoch 59/100
3078/3078 - 13s - loss: 0.0814 - accuracy: 0.9719 - val_loss: 4.1593 - val_accuracy: 0.5559 - 13s/epoch - 4ms/step
Epoch 60/100
3078/3078 - 17s - loss: 0.0822 - accuracy: 0.9729 - val_loss: 4.0241 - val_accuracy: 0.5611 - 17s/epoch - 6ms/step
Epoch 61/100
3078/3078 - 15s - loss: 0.0803 - accuracy: 0.9733 - val_loss: 4.0689 - val_accuracy: 0.5550 - 15s/epoch - 5ms/step
Epoch 62/100
3078/3078 - 19s - loss: 0.0797 - accuracy: 0.9733 - val_loss: 3.9606 - val_accuracy: 0.5556 - 19s/epoch - 6ms/step
Epoch 63/100
3078/3078 - 19s - loss: 0.0811 - accuracy: 0.9740 - val_loss: 4.1553 - val_accuracy: 0.5535 - 19s/epoch - 6ms/step
Epoch 64/100
3078/3078 - 20s - loss: 0.0750 - accuracy: 0.9758 - val_loss: 4.1912 - val_accuracy: 0.5536 - 20s/epoch - 6ms/step
Epoch 65/100
3078/3078 - 19s - loss: 0.0778 - accuracy: 0.9745 - val_loss: 4.2020 - val_accuracy: 0.5569 - 19s/epoch - 6ms/step
Epoch 66/100
3078/3078 - 20s - loss: 0.0755 - accuracy: 0.9741 - val_loss: 4.4035 - val_accuracy: 0.5607 - 20s/epoch - 6ms/step
Epoch 67/100
3078/3078 - 20s - loss: 0.0718 - accuracy: 0.9763 - val_loss: 4.0836 - val_accuracy: 0.5516 - 20s/epoch - 7ms/step
Epoch 68/100
3078/3078 - 17s - loss: 0.0757 - accuracy: 0.9751 - val_loss: 4.1573 - val_accuracy: 0.5562 - 17s/epoch - 5ms/step
Epoch 69/100
3078/3078 - 16s - loss: 0.0758 - accuracy: 0.9759 - val_loss: 4.4116 - val_accuracy: 0.5527 - 16s/epoch - 5ms/step
Epoch 70/100
3078/3078 - 16s - loss: 0.0741 - accuracy: 0.9753 - val_loss: 4.5021 - val_accuracy: 0.5577 - 16s/epoch - 5ms/step
Epoch 71/100
3078/3078 - 17s - loss: 0.0684 - accuracy: 0.9773 - val_loss: 4.1417 - val_accuracy: 0.5544 - 17s/epoch - 5ms/step
Epoch 72/100
3078/3078 - 17s - loss: 0.0765 - accuracy: 0.9759 - val_loss: 4.3002 - val_accuracy: 0.5515 - 17s/epoch - 5ms/step
Epoch 73/100
3078/3078 - 17s - loss: 0.0689 - accuracy: 0.9775 - val_loss: 4.3418 - val_accuracy: 0.5569 - 17s/epoch - 5ms/step
Epoch 74/100
3078/3078 - 18s - loss: 0.0707 - accuracy: 0.9769 - val_loss: 4.2534 - val_accuracy: 0.5593 - 18s/epoch - 6ms/step
Epoch 75/100
3078/3078 - 19s - loss: 0.0667 - accuracy: 0.9788 - val_loss: 4.4500 - val_accuracy: 0.5551 - 19s/epoch - 6ms/step
Epoch 76/100
3078/3078 - 19s - loss: 0.0725 - accuracy: 0.9763 - val_loss: 4.4838 - val_accuracy: 0.5556 - 19s/epoch - 6ms/step
Epoch 77/100
3078/3078 - 19s - loss: 0.0687 - accuracy: 0.9783 - val_loss: 4.5001 - val_accuracy: 0.5558 - 19s/epoch - 6ms/step
Epoch 78/100
3078/3078 - 20s - loss: 0.0756 - accuracy: 0.9763 - val_loss: 4.3539 - val_accuracy: 0.5590 - 20s/epoch - 6ms/step
Epoch 79/100
3078/3078 - 21s - loss: 0.0669 - accuracy: 0.9789 - val_loss: 4.6994 - val_accuracy: 0.5572 - 21s/epoch - 7ms/step
Epoch 80/100
```

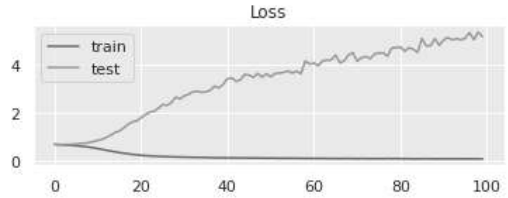
```
Epoch 80/100
3078/3078 - 19s - loss: 0.0744 - accuracy: 0.9768 - val_loss: 4.7182 - val_accuracy: 0.5574 - 19s/epoch - 6ms/step
Epoch 81/100
3078/3078 - 20s - loss: 0.0682 - accuracy: 0.9783 - val_loss: 4.7473 - val_accuracy: 0.5595 - 20s/epoch - 7ms/step
Epoch 82/100
3078/3078 - 19s - loss: 0.0665 - accuracy: 0.9787 - val_loss: 4.5687 - val_accuracy: 0.5542 - 19s/epoch - 6ms/step
Epoch 83/100
```

```
train_acc = model.evaluate(X_train, y_train, verbose=0)
test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Training accuracy {:.4f}',train_acc)
print('Testing accuracy {:.4f}',test_acc)
```

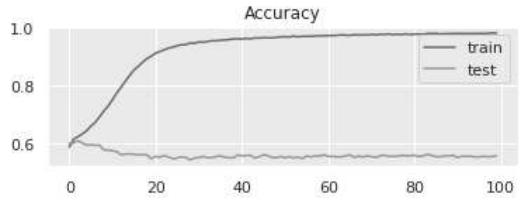
```
Training accuracy {:.4f} [0.04683384671807289, 0.9857227206230164]
Testing accuracy {:.4f} [5.173447132110596, 0.5571892857551575]
```

```
# plot loss during training
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(fittedm.history['loss'], label='train')
pyplot.plot(fittedm.history['val_loss'], label='test')
pyplot.legend()
```

```
<matplotlib.legend.Legend at 0x7fee2face710>
```




```
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(fittedm.history['accuracy'], label='train')
pyplot.plot(fittedm.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```



✓ 11s completed at 1:10 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.