Assignment6

March 21, 2025

EDA

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

```
[ ]: df = pd.read_csv('churn.csv')
```

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   phoneno    5000 non-null   int64
 1   age        4994 non-null   float64
 2   gender     5000 non-null   object
 3   zipcode    5000 non-null   int64
 4   calls      5000 non-null   int64
 5   sms        5000 non-null   int64
 6   mms        5000 non-null   int64
 7   charges    5000 non-null   int64
 8   coverage   5000 non-null   int64
 9   complaint  5000 non-null   int64
 10  sim        5000 non-null   object
 11  phone      5000 non-null   object
 12  prepost    5000 non-null   object
 13  churn      5000 non-null   object
dtypes: float64(1), int64(8), object(5)
memory usage: 547.0+ KB
```

```
[ ]: df.head()
```

```
[ ]:    phoneno  age  gender  zipcode  calls  sms  mms  charges  coverage  \
     0     5974  1.0    Male    91107    160   25    1      490         0
     1     4535  1.0    Male    90089    150   45   19      340         0
```

```
2     4016  1.0    Male    94720    100   39   15       110          0
3     8523  2.0    Male    94112    270   35    9      1000          0
4     5052  2.0  Female    91330    100   35    8       450          0

   complaint         sim   phone  prepost     churn
0          4    Dual Sim  Andoid  Prepaid  No Churn
1          3    Dual Sim  Andoid  Prepaid  No Churn
2          1  Single Sim  Andoid  Prepaid  No Churn
3          1  Single Sim  Andoid  Prepaid  No Churn
4          4  Single Sim  Andoid  Prepaid  No Churn
```

```
[ ]: df.columns
```

```
[ ]: Index(['phoneno', 'age', 'gender', 'zipcode', 'calls', 'sms', 'mms', 'charges',
            'coverage', 'complaint', 'sim', 'phone', 'prepost', 'churn'],
           dtype='object')
```

```
[ ]: df.describe(include='all')
```

```
[ ]:            phoneno          age gender       zipcode        calls  \
     count  5000.000000  4994.000000   5000   5000.000000  5000.000000
     unique         NaN          NaN      2           NaN          NaN
     top            NaN          NaN   Male           NaN          NaN
     freq           NaN          NaN   3530           NaN          NaN
     mean   5497.188000     1.881057    NaN  93152.503000   193.793800
     std    2603.474018     0.839796    NaN   2121.852197   174.765898
     min    1000.000000     1.000000    NaN   9307.000000     0.000000
     25%    3266.500000     1.000000    NaN  91911.000000    70.000000
     50%    5457.500000     2.000000    NaN  93437.000000   150.000000
     75%    7779.250000     3.000000    NaN  94608.000000   250.000000
     max    9997.000000     3.000000    NaN  96651.000000  1000.000000

                    sms          mms      charges     coverage    complaint  \
     count  5000.000000  5000.000000  5000.000000  5000.000000  5000.000000
     unique         NaN          NaN          NaN          NaN          NaN
     top            NaN          NaN          NaN          NaN          NaN
     freq           NaN          NaN          NaN          NaN          NaN
     mean     45.338400    20.104600   737.742000     0.719000     2.388600
     std      11.463166    11.467954   460.337293     1.233184     1.154061
     min      23.000000    -3.000000    80.000000     0.000000     0.000000
     25%      35.000000    10.000000   390.000000     0.000000     1.000000
     50%      45.000000    20.000000   640.000000     0.000000     2.000000
     75%      55.000000    30.000000   980.000000     2.000000     3.000000
     max      67.000000    43.000000  2240.000000     7.000000     4.000000

                  sim   phone  prepost     churn
     count       5000    5000     5000      5000
```

```
unique          2       2         2         2
top     Single Sim  Andoid  Postpaid  No Churn
freq         4478    4698      2984      4520
mean          NaN     NaN       NaN       NaN
std           NaN     NaN       NaN       NaN
min           NaN     NaN       NaN       NaN
25%           NaN     NaN       NaN       NaN
50%           NaN     NaN       NaN       NaN
75%           NaN     NaN       NaN       NaN
max           NaN     NaN       NaN       NaN
```

[ ]:
```
df.shape
```

[ ]:
```
(5000, 14)
```

[ ]:
```
df.tail()
```

[ ]:
```
      phoneno  age  gender  zipcode  calls  sms  mms  charges  coverage  \
4995     4704  3.0    Male    92697    190   29    3      400         0
4996     3149  1.0    Male    92037     40   30    4      150         1
4997     7402  3.0    Male    93023     30   63   39      240         0
4998     5742  2.0    Male    90034     50   65   40      490         0
4999     3689  1.0  Female    92612     80   28    4      830         0

      complaint         sim   phone   prepost     churn
4995          1  Single Sim  Andoid  Postpaid  No Churn
4996          4  Single Sim  Andoid  Postpaid  No Churn
4997          2  Single Sim  Andoid   Prepaid  No Churn
4998          3  Single Sim  Andoid  Postpaid  No Churn
4999          3  Single Sim  Andoid  Postpaid  No Churn
```

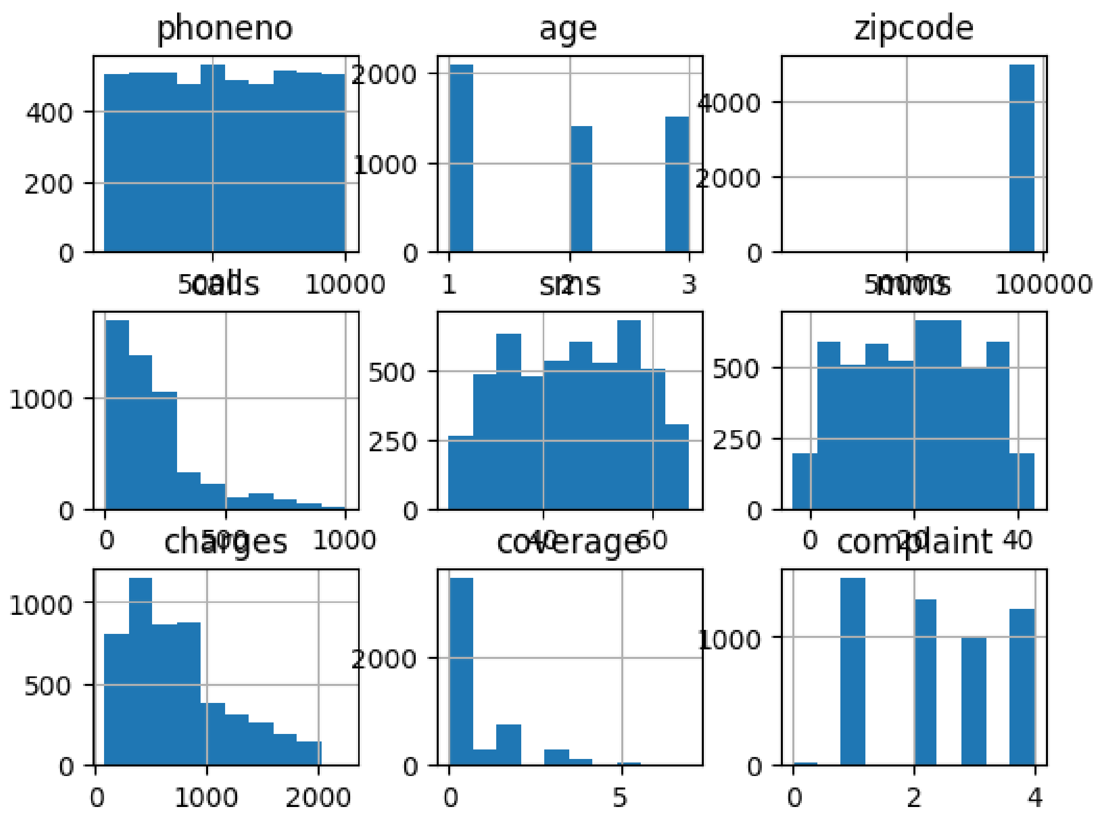
[ ]:
```
df.isnull().sum()
```

[ ]:
```
phoneno      0
age          6
gender       0
zipcode      0
calls        0
sms          0
mms          0
charges      0
coverage     0
complaint    0
sim          0
phone        0
prepost      0
churn        0
```

```
dtype: int64
```

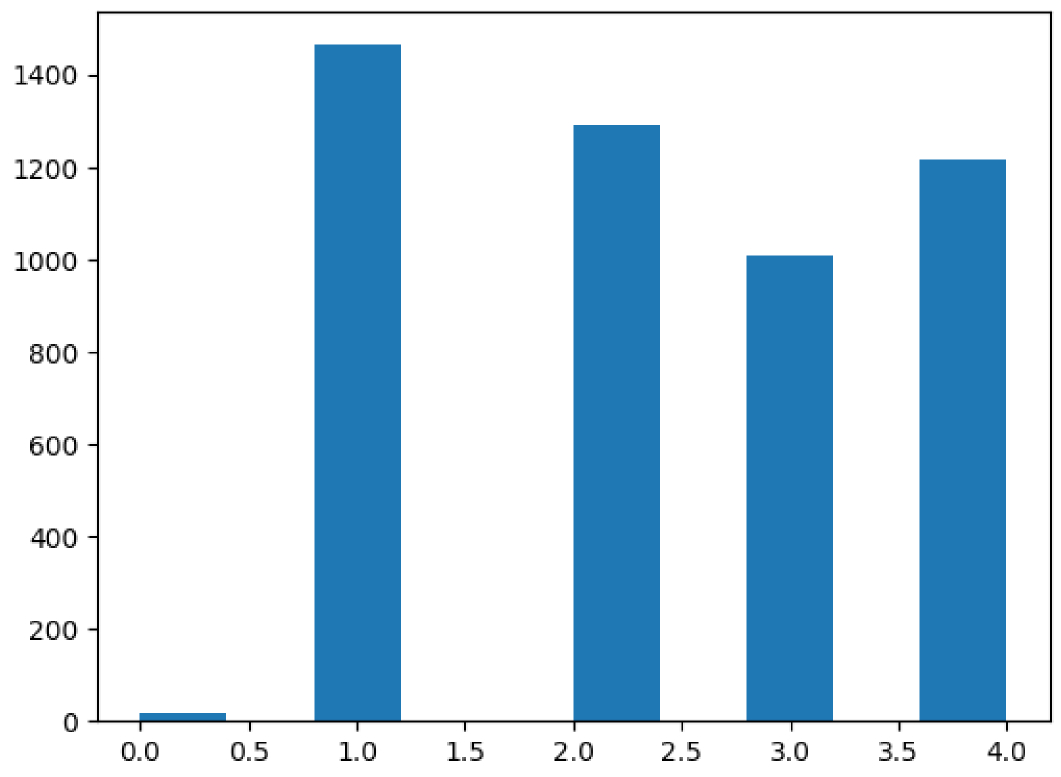
```
[ ]: df.hist()
```

```
[ ]: array([[<Axes: title={'center': 'phoneno'}>,
             <Axes: title={'center': 'age'}>,
             <Axes: title={'center': 'zipcode'}>],
            [<Axes: title={'center': 'calls'}>,
             <Axes: title={'center': 'sms'}>, <Axes: title={'center': 'mms'}>],
            [<Axes: title={'center': 'charges'}>,
             <Axes: title={'center': 'coverage'}>,
             <Axes: title={'center': 'complaint'}>]], dtype=object)
```



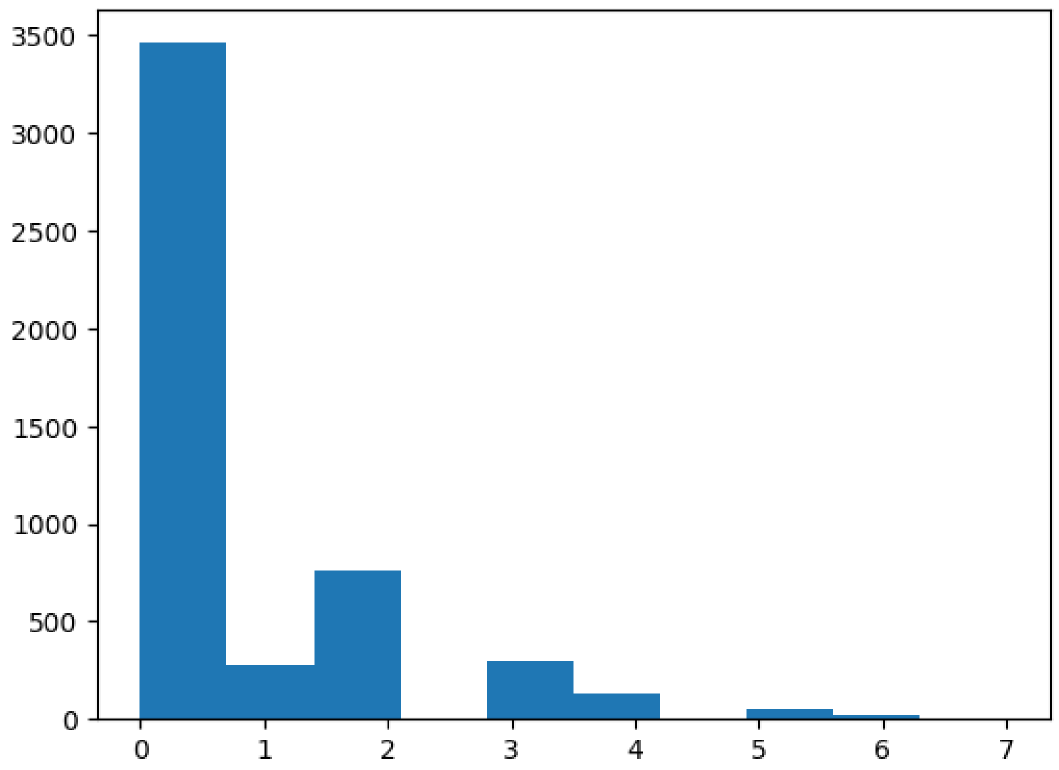
```
[ ]: plt.hist(df['complaint'])
```

```
[ ]: (array([  18.,    0., 1464.,    0.,    0., 1292.,    0., 1009.,    0.,
             1217.]),
      array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
      <BarContainer object of 10 artists>)
```



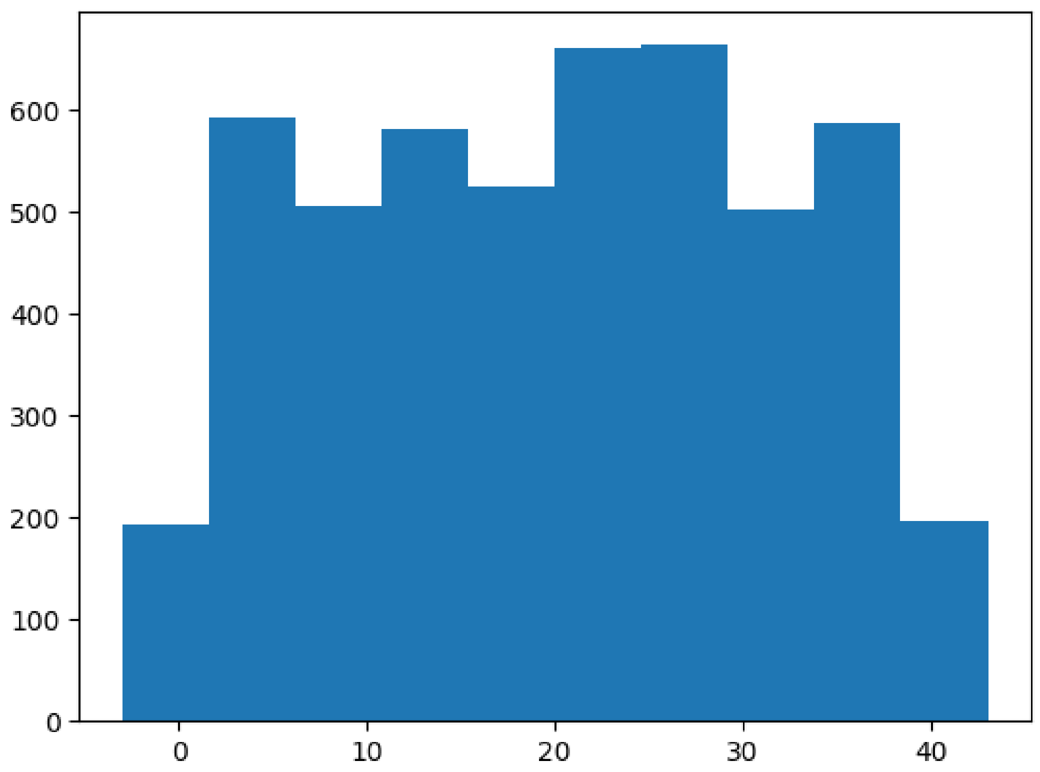
```
[ ]: plt.hist(df['coverage'])
```

```
[ ]: (array([3462.,  282.,  758.,    0.,  297.,  128.,    0.,   48.,   21.,
                4.]),
      array([0. , 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, 5.6, 6.3, 7. ]),
      <BarContainer object of 10 artists>)
```



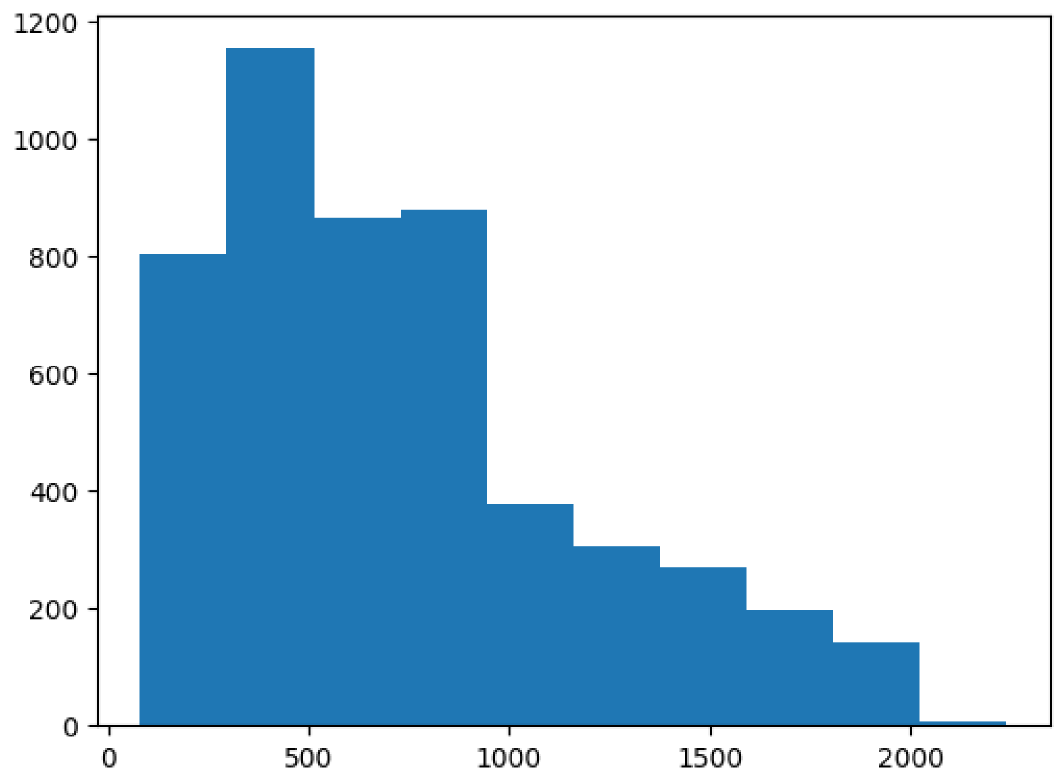
[ ]:
```
plt.hist(df['mms'])
```

[ ]:
```
(array([192., 592., 505., 581., 524., 660., 663., 501., 586., 196.]),
 array([-3. ,  1.6,  6.2, 10.8, 15.4, 20. , 24.6, 29.2, 33.8, 38.4, 43. ]),
 <BarContainer object of 10 artists>)
```



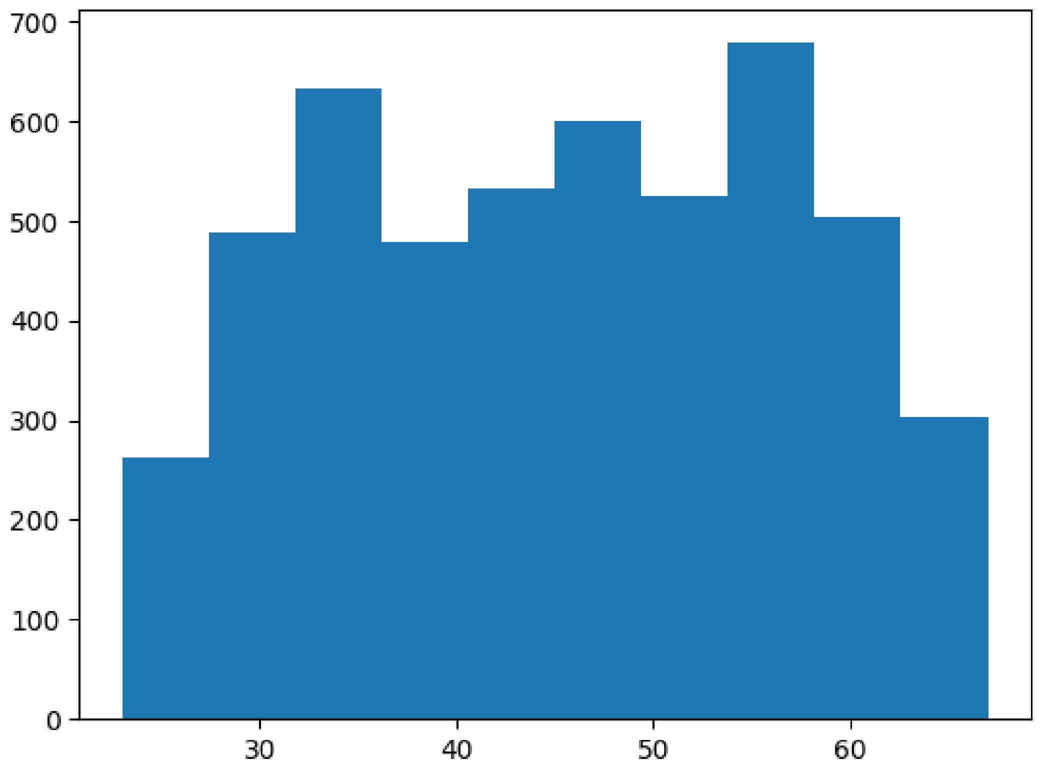
```
[ ]: plt.hist(df['charges'])
```

```
[ ]: (array([ 802., 1153.,  867.,  879.,  377.,  307.,  268.,  197.,  141.,
               9.]),
      array([  80.,  296.,  512.,  728.,  944., 1160., 1376., 1592., 1808.,
             2024., 2240.]),
      <BarContainer object of 10 artists>)
```



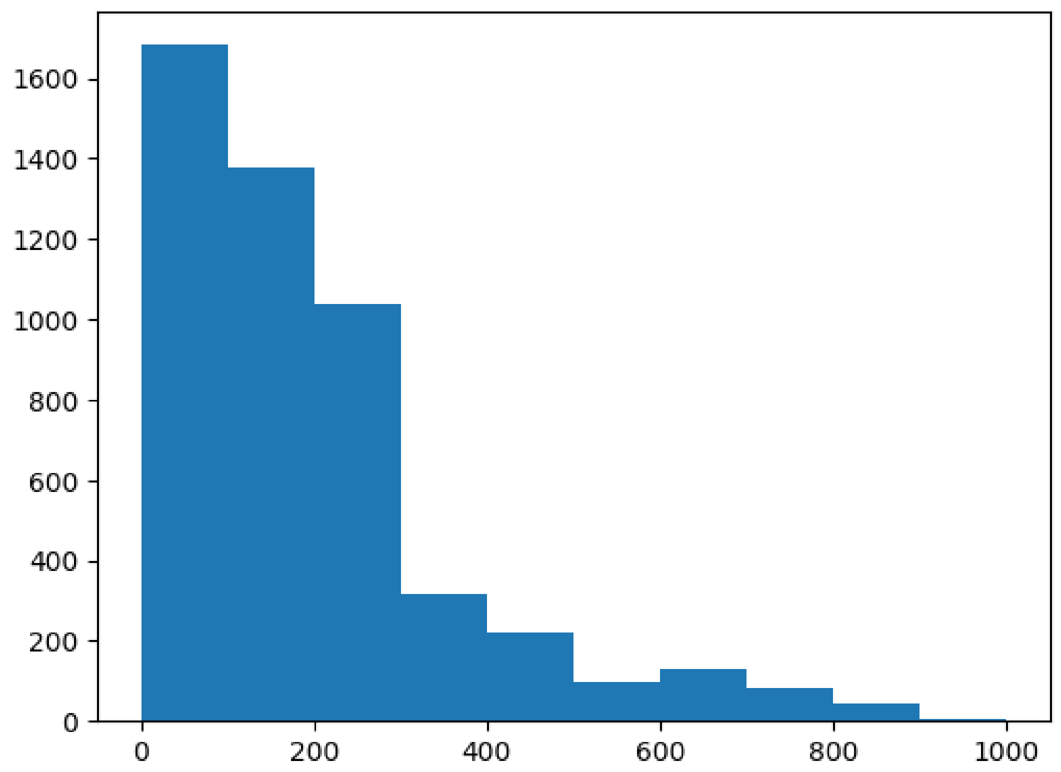
```
[ ]: plt.hist(df['sms'])
```

```
[ ]: (array([262., 487., 632., 479., 532., 600., 524., 678., 504., 302.]),
      array([23. , 27.4, 31.8, 36.2, 40.6, 45. , 49.4, 53.8, 58.2, 62.6, 67. ]),
      <BarContainer object of 10 artists>)
```



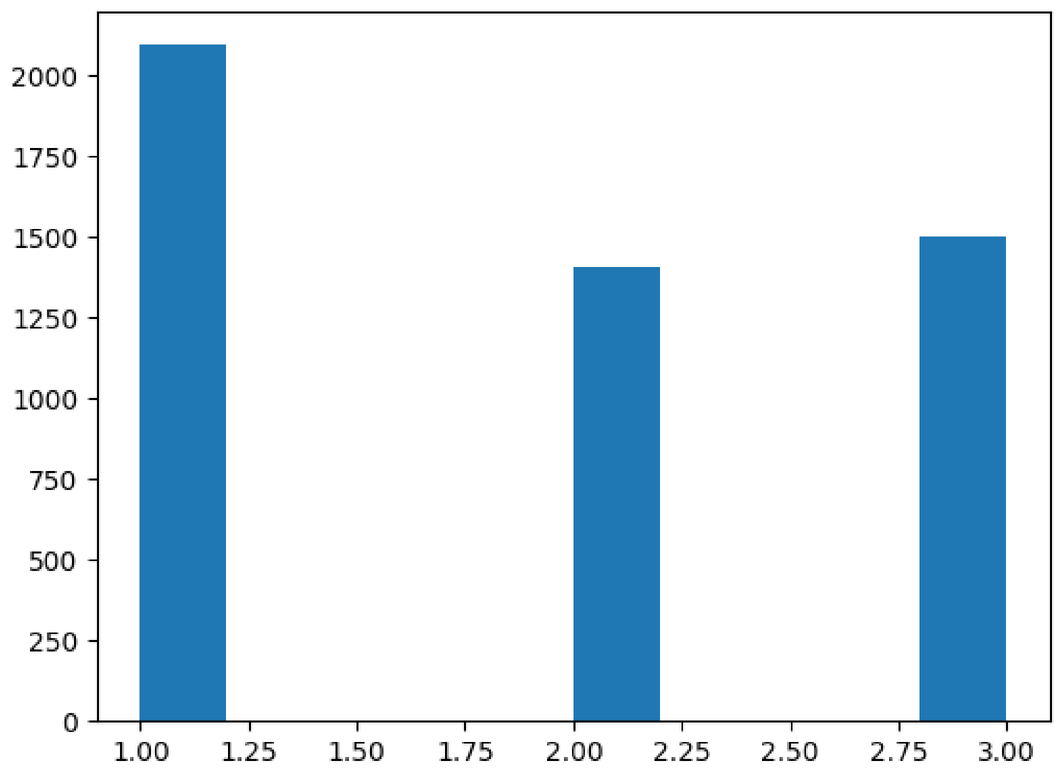
```
[ ]: plt.hist(df['calls'])
```

```
[ ]: (array([1683., 1376., 1039.,  319.,  219.,   97.,  132.,   84.,   45.,
                6.]),
      array([   0.,  100.,  200.,  300.,  400.,  500.,  600.,  700.,  800.,
              900., 1000.]),
      <BarContainer object of 10 artists>)
```



```
[ ]: plt.hist(df['age'])
```

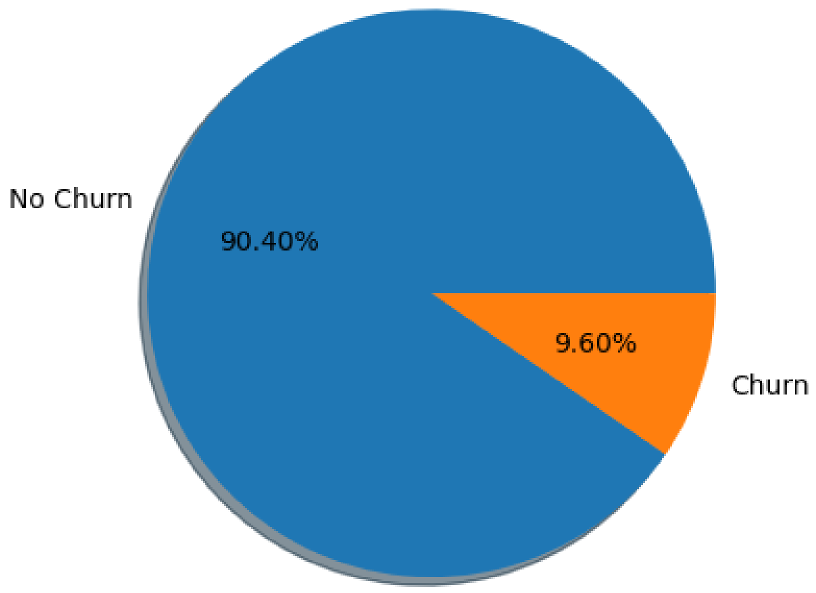
```
[ ]: (array([2093.,    0.,    0.,    0.,    0., 1402.,    0.,    0.,    0.,
            1499.]),
      array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),
      <BarContainer object of 10 artists>)
```



Inference: similar to count plot. Though have floating point value

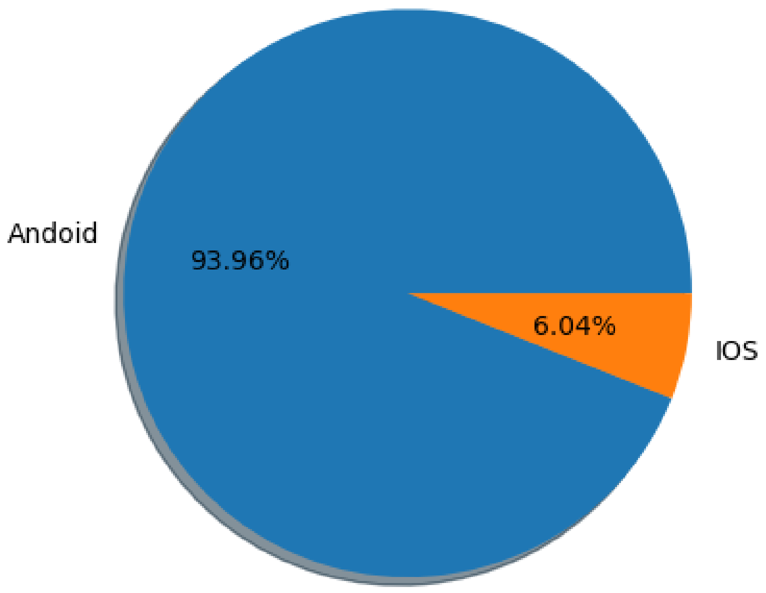
```
[ ]: plt.pie(df['churn'].value_counts().values,shadow=True , autopct='%1.2f%%',␣
     ↪labels=df.churn.value_counts().index)
```

```
[ ]: ([<matplotlib.patches.Wedge at 0x79b033e9e390>,
       <matplotlib.patches.Wedge at 0x79b03395d8d0>],
      [Text(-1.0503509720705493, 0.32674582701306604, 'No Churn'),
       Text(1.0503510785377979, -0.3267454847652605, 'Churn')],
      [Text(-0.5729187120384813, 0.17822499655258148, '90.40%'),
       Text(0.572918770111526, -0.17822480987196024, '9.60%')])
```



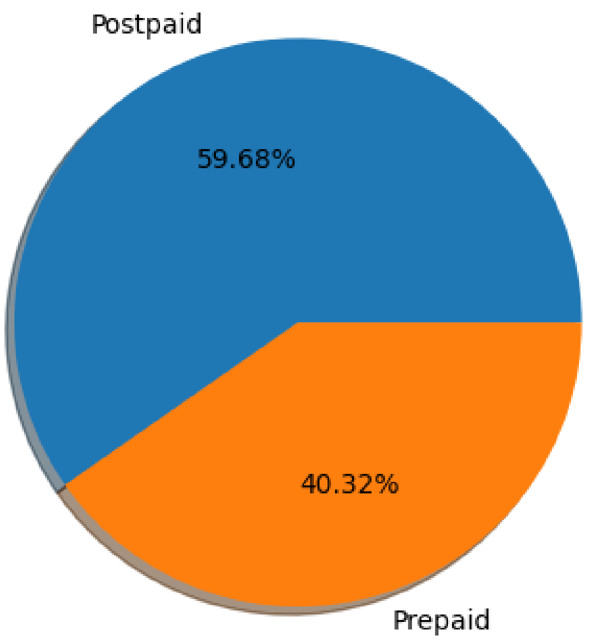
```
[ ]: plt.pie(df['phone'].value_counts().values,shadow=True , autopct='%1.2f%%',␣
     ↪labels=df.phone.value_counts().index)
```

```
[ ]: ([<matplotlib.patches.Wedge at 0x79b033991f10>,
       <matplotlib.patches.Wedge at 0x79b0339a0590>],
      [Text(-1.0802560936468724, 0.2074771605232722, 'Andoid'),
       Text(1.080256161251455, -0.20747680853114742, 'IOS')],
      [Text(-0.5892305965346576, 0.11316936028542118, '93.96%'),
       Text(0.5892306334098845, -0.11316916828971677, '6.04%')])
```



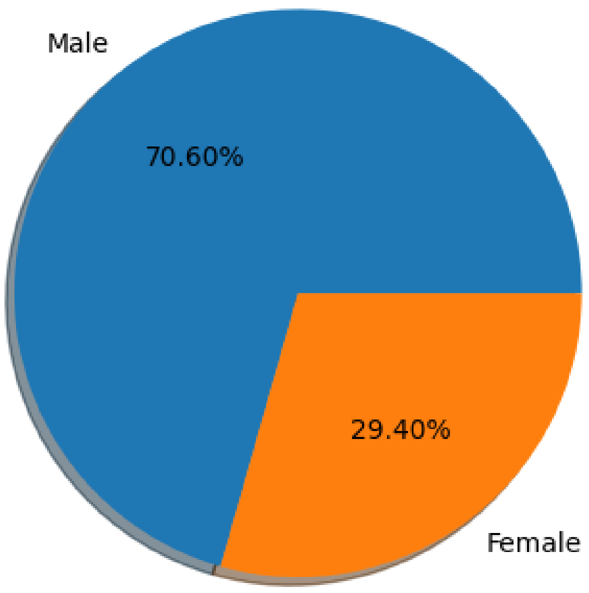
```
[ ]: plt.pie(df['prepost'].value_counts().values,shadow=True , autopct='%1.2f%%' ,␣
      ↪labels=df.prepost.value_counts().index)
```

```
[ ]: ([<matplotlib.patches.Wedge at 0x79b0339d9450>,
       <matplotlib.patches.Wedge at 0x79b0339cdc50>],
      [Text(-0.3293846408032828, 1.0495264448325696, 'Postpaid'),
       Text(0.3293851078956752, -1.0495262982396176, 'Prepaid')],
      [Text(-0.17966434952906332, 0.5724689699086742, '59.68%'),
       Text(0.1796646043067319, -0.5724688899488822, '40.32%')])
```



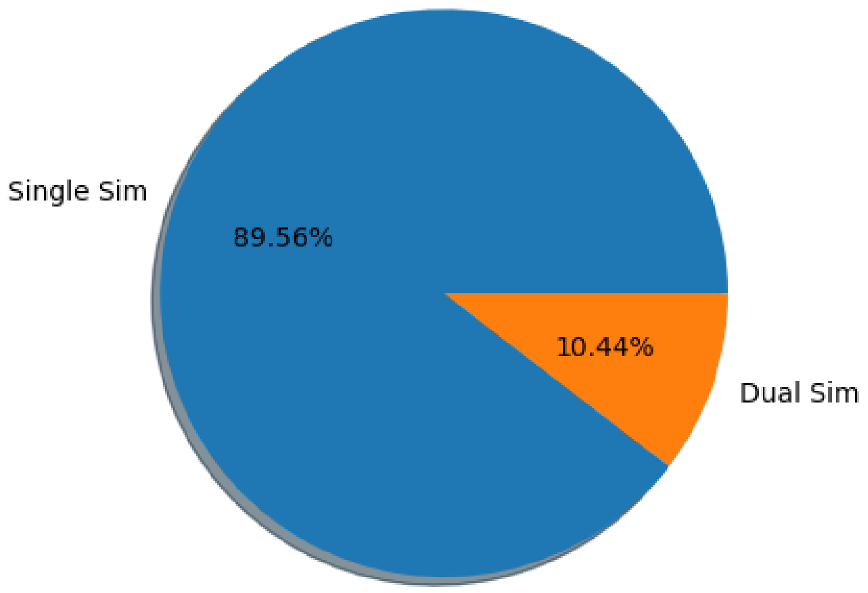
```
[ ]: plt.pie(df['gender'].value_counts().values,shadow=True , autopct='%1.2f%%',␣
     ↪labels=df.gender.value_counts().index)
```

```
[ ]: ([<matplotlib.patches.Wedge at 0x79b033a16990>,
       <matplotlib.patches.Wedge at 0x79b033a24d10>],
      [Text(-0.663222493530133, 0.8775738852516481, 'Male'),
       Text(0.6632225702500791, -0.8775738272708907, 'Female')],
      [Text(-0.36175772374370885, 0.47867666468271713, '70.60%'),
       Text(0.3617577655909522, -0.4786766330568494, '29.40%')])
```



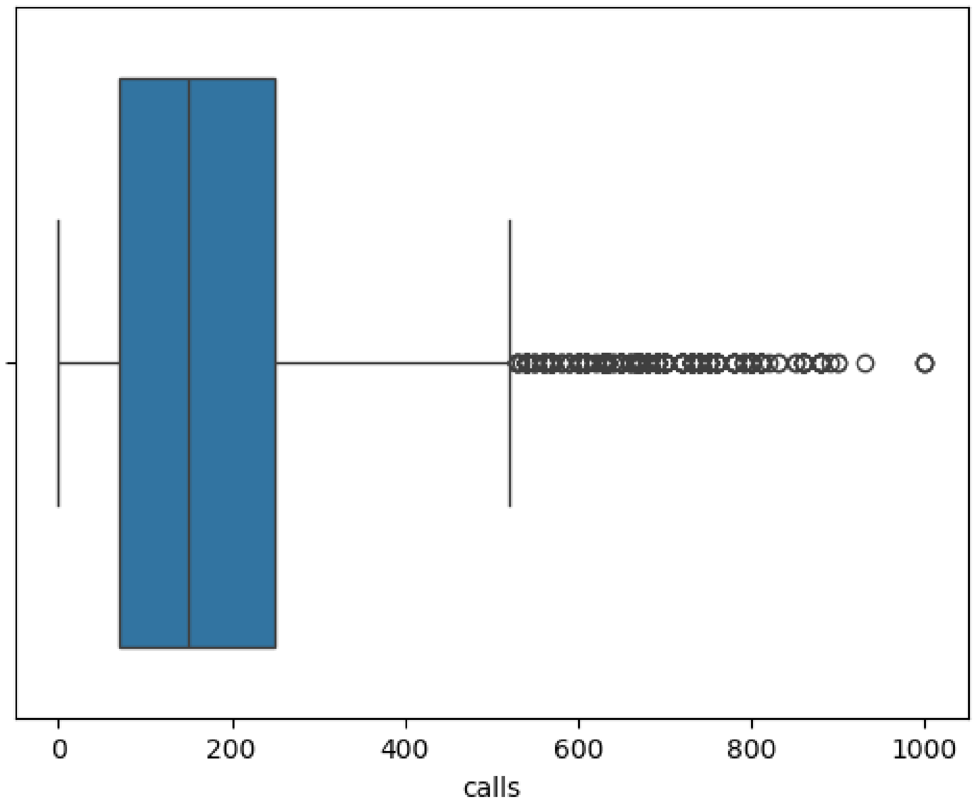
```
[ ]: plt.pie(df['sim'].value_counts().values,shadow=True , autopct='%1.2f%%',␣
     ↪labels=df.sim.value_counts().index)
```

```
[ ]: ([<matplotlib.patches.Wedge at 0x79b033864310>,
       <matplotlib.patches.Wedge at 0x79b033866a50>],
      [Text(-1.0413637338022248, 0.35434668605969627, 'Single Sim'),
       Text(1.0413638492629755, -0.35434634674030313, 'Dual Sim')],
      [Text(-0.5680165820739408, 0.1932800105780161, '89.56%'),
       Text(0.568016645052532, -0.19327982549471076, '10.44%')])
```



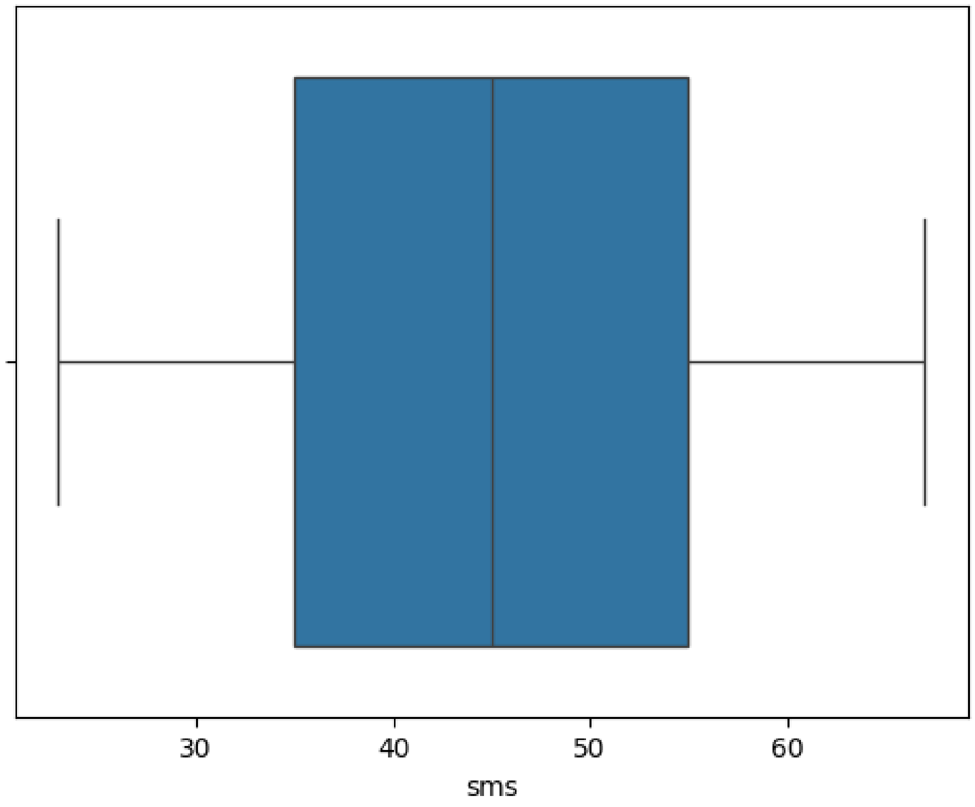
```
[ ]: sns.boxplot(x='calls', data=df)
```

```
[ ]: <Axes: xlabel='calls'>
```



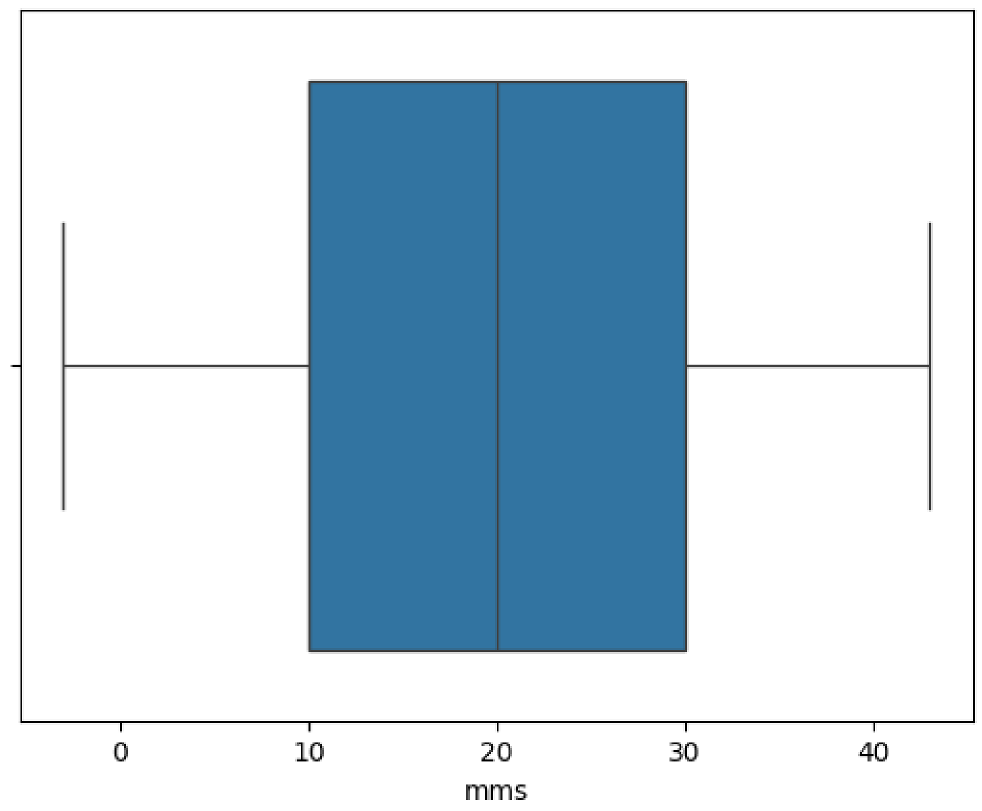
```
[ ]: sns.boxplot(x='sms', data=df)
```

```
[ ]: <Axes: xlabel='sms'>
```



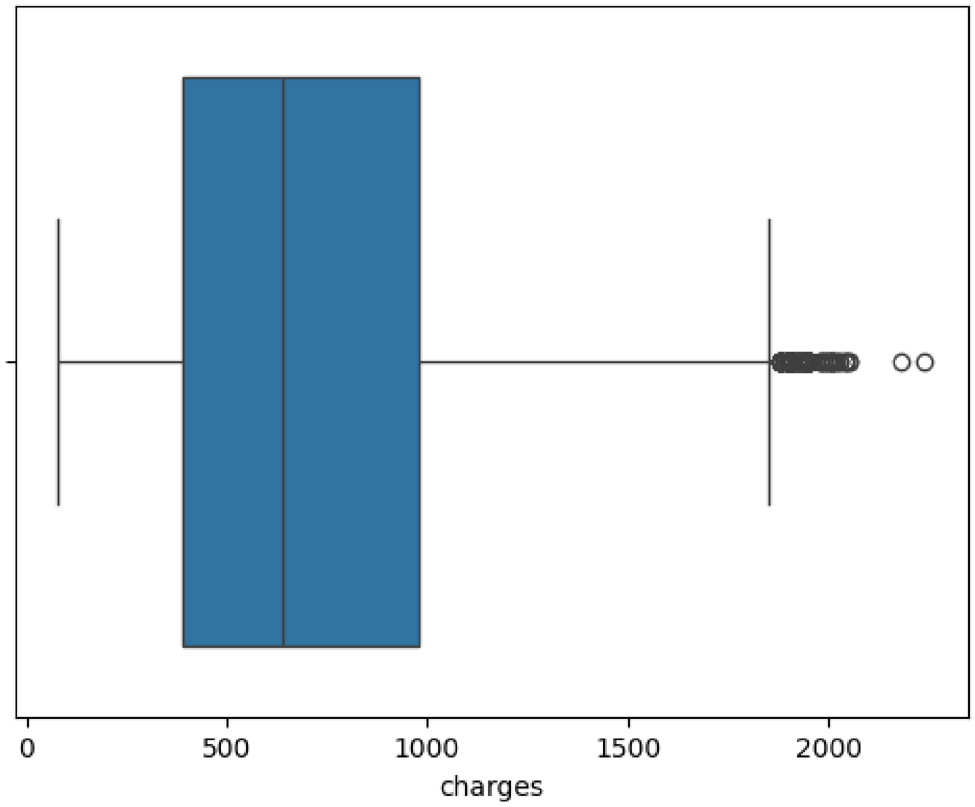
```
[ ]: sns.boxplot(x='mms', data=df)
```

```
[ ]: <Axes: xlabel='mms'>
```



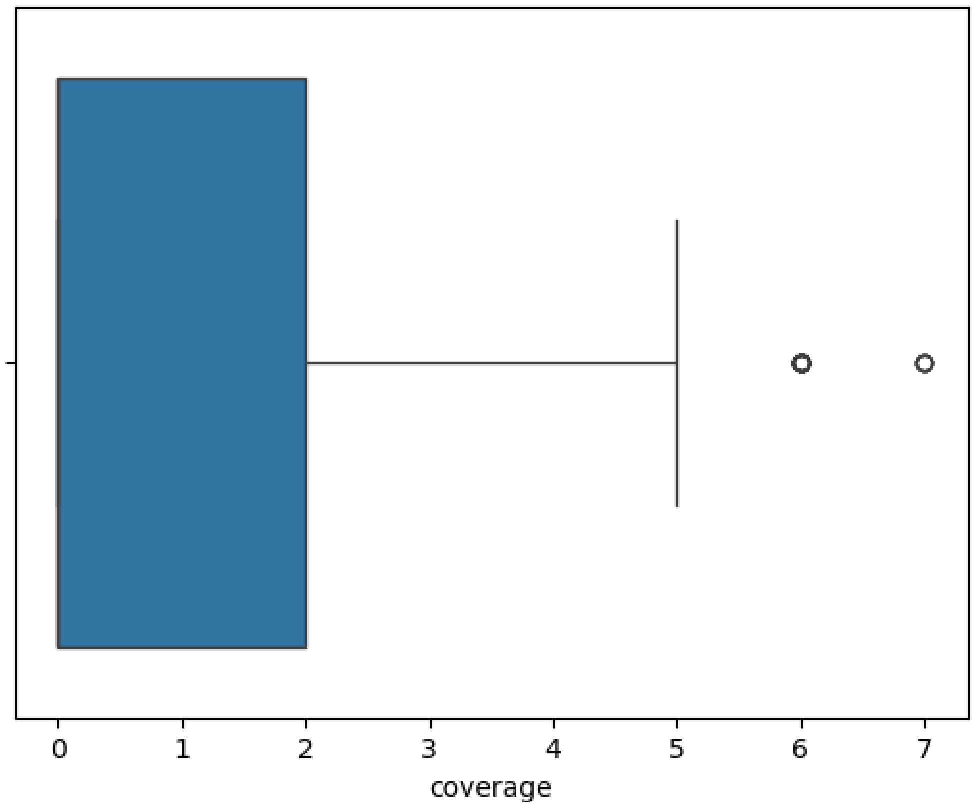
```
[ ]: sns.boxplot(x='charges', data=df)
```

```
[ ]: <Axes: xlabel='charges'>
```



```
[ ]: sns.boxplot(x='coverage', data=df)
```

```
[ ]: <Axes: xlabel='coverage'>
```



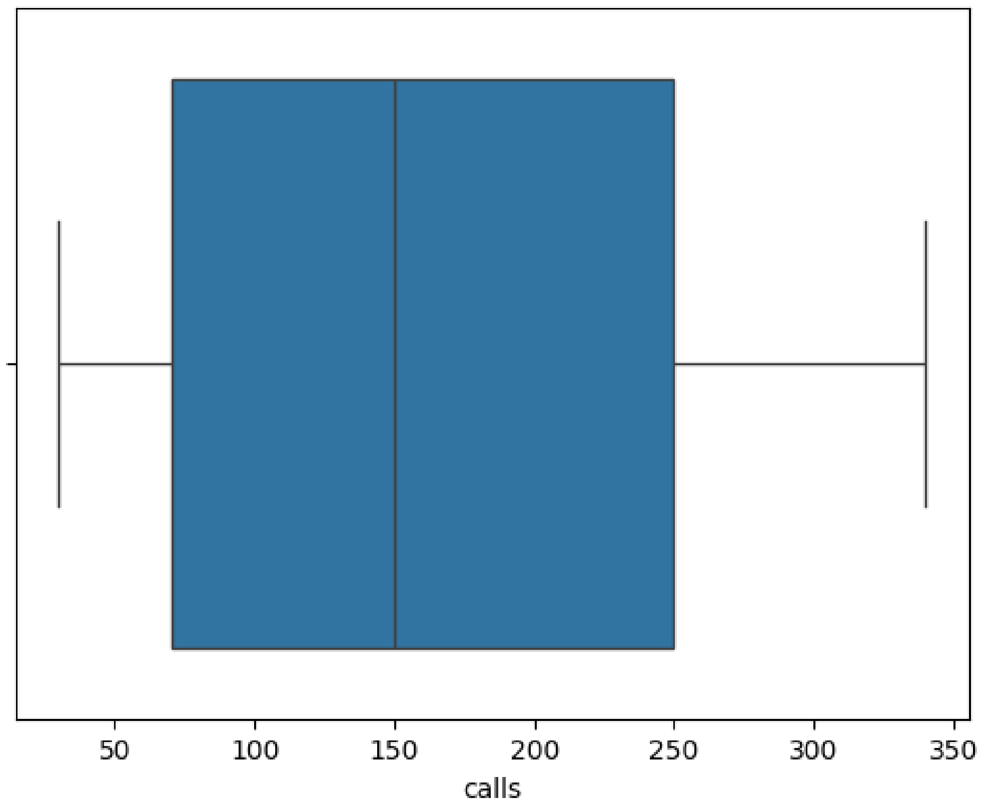
[ ]:
```
print(df['calls'].quantile(0.10))
print(df['calls'].quantile(0.85))
```

```
30.0
340.0
```

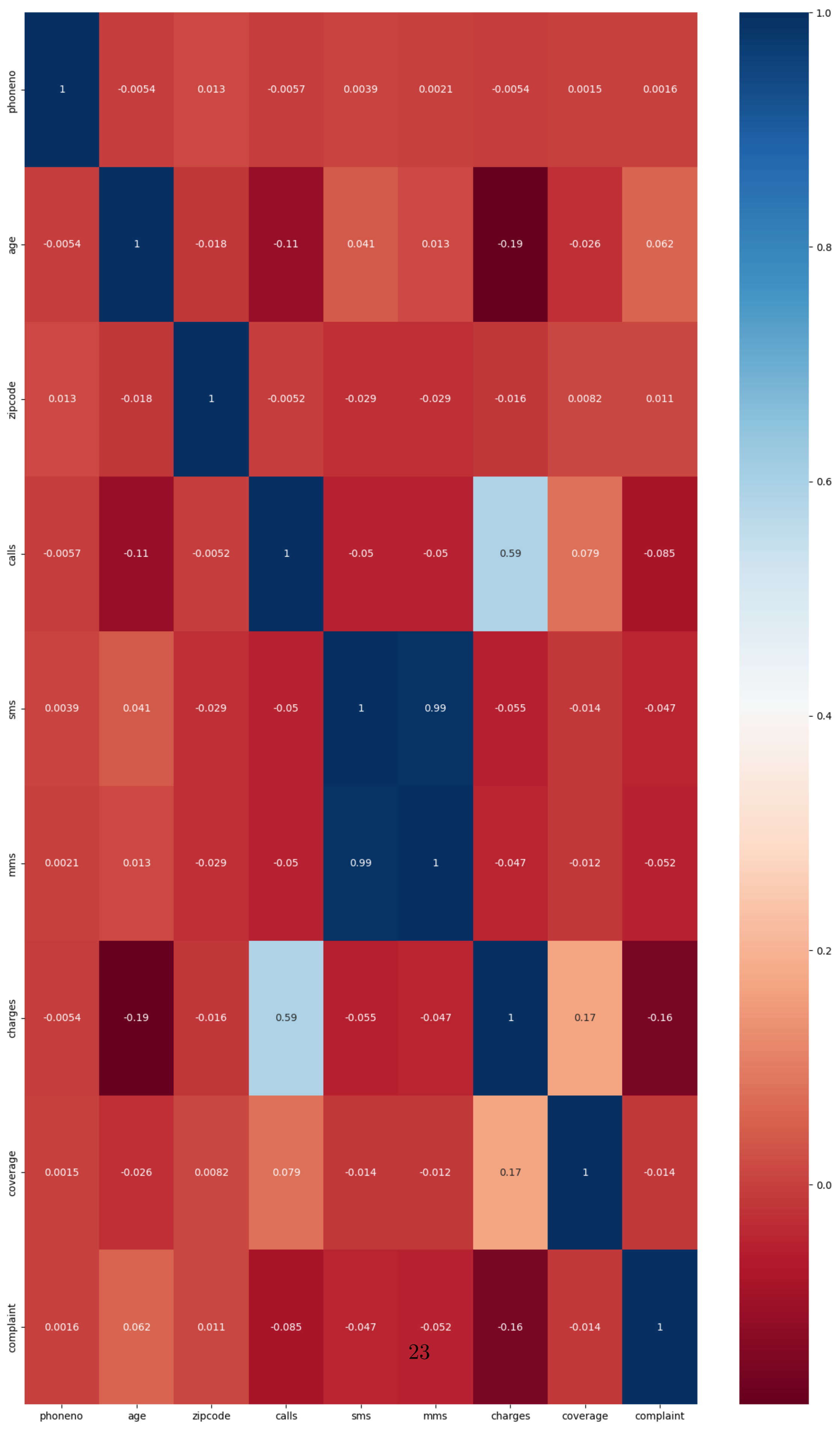
[ ]:
```
df['calls'] = np.where(df['calls'] > 340 , 340 , df['calls'])
df['calls'] = np.where(df['calls'] < 30 , 30 , df['calls'])
```

[ ]:
```
sns.boxplot(data=df , x='calls')
```

[ ]:
```
<Axes: xlabel='calls'>
```



```
[ ]: fig,ax = plt.subplots(figsize=(15,25))
     sns.heatmap(df.corr(numeric_only = True) , annot=True ,  cmap='RdBu')
     plt.show()
```


phoneno
age
zipcode
calls
sms
mms
charges
coverage
complaint
1 -0.0054 0.013 -0.0057 0.0039 0.0021 -0.0054 0.0015 0.0016
-0.0054 1 -0.018 -0.11 0.041 0.013 -0.19 -0.026 0.062
0.013 -0.018 1 -0.0052 -0.029 -0.029 -0.016 0.0082 0.011
-0.0057 -0.11 -0.0052 1 -0.05 -0.05 0.59 0.079 -0.085
0.0039 0.041 -0.029 -0.05 1 0.99 -0.055 -0.014 -0.047
0.0021 0.013 -0.029 -0.05 0.99 1 -0.047 -0.012 -0.052
-0.0054 -0.19 -0.016 0.59 -0.055 -0.047 1 0.17 -0.16
0.0015 -0.026 0.0082 0.079 -0.014 -0.012 0.17 1 -0.014
0.0016 0.062 0.011 -0.085 -0.047 -0.052 -0.16 -0.014 1
1.0
0.8
0.6
0.4
0.2
0.0

```
[ ]: df.drop(axis=1 , columns = ['phoneno' , 'zipcode'] , inplace=True)
```

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

```
[ ]: churn
     No Churn    4520
     Churn        480
     Name: count, dtype: int64
```

Handling Missing

```
[ ]: df['age'].fillna(df['age'].median())
```

```
[ ]: 0       1.0
     1       1.0
     2       1.0
     3       2.0
     4       2.0
            …
     4995    3.0
     4996    1.0
     4997    3.0
     4998    2.0
     4999    1.0
     Name: age, Length: 5000, dtype: float64
```

Handling Categorical

```
[ ]: df.head()
```

```
[ ]:    age  gender  calls  sms  mms  charges  coverage  complaint         sim  \
     0  1.0    Male    160   25    1      490         0          4    Dual Sim
     1  1.0    Male    150   45   19      340         0          3    Dual Sim
     2  1.0    Male    100   39   15      110         0          1  Single Sim
     3  2.0    Male    270   35    9     1000         0          1  Single Sim
     4  2.0  Female    100   35    8      450         0          4  Single Sim

         phone  prepost     churn
     0  Andoid  Prepaid  No Churn
     1  Andoid  Prepaid  No Churn
     2  Andoid  Prepaid  No Churn
     3  Andoid  Prepaid  No Churn
     4  Andoid  Prepaid  No Churn
```

```
[ ]: df['phone'] = df.apply(lambda x: 1 if x['phone'] == 'Android' else 0, axis=1)
[ ]: df['gender'] = df.apply(lambda x: 1 if x['gender'] == 'Female' else 0, axis=1)
```

```
[ ]: df['prepost'] = df.apply(lambda x: 1 if x['prepost'] == 'Prepaid' else 0,␣
     ↪axis=1)
```

```
[ ]: df['sim'] = df.apply(lambda x: 1 if x['sim'] == 'Dual Sim' else 0, axis=1)
```

```
[ ]: df.head()
```

```
[ ]:    age  gender  calls  sms  mms  charges  coverage  complaint  sim  phone  \
     0  1.0       0    160   25    1      490         0          4    1      0
     1  1.0       0    150   45   19      340         0          3    1      0
     2  1.0       0    100   39   15      110         0          1    0      0
     3  2.0       0    270   35    9     1000         0          1    0      0
     4  2.0       1    100   35    8      450         0          4    0      0

        prepost     churn
     0        1  No Churn
     1        1  No Churn
     2        1  No Churn
     3        1  No Churn
     4        1  No Churn
```

Normalization

```
[ ]: df['charges'] =((df['charges'] - df['charges'].mean() )/ (df['charges'].std()))
```

```
[ ]: df['calls'] =((df['calls'] - df['calls'].mean() )/ (df['calls'].std()))
```

```
[ ]: df.head()
```

```
[ ]:    age  gender     calls  sms  mms   charges  coverage  complaint  sim  phone  \
     0  1.0       0 -0.060834   25    1 -0.538175         0          4    1      0
     1  1.0       0 -0.153574   45   19 -0.864023         0          3    1      0
     2  1.0       0 -0.617274   39   15 -1.363657         0          1    0      0
     3  2.0       0  0.959306   35    9  0.569708         0          1    0      0
     4  2.0       1 -0.617274   35    8 -0.625068         0          4    0      0

        prepost     churn
     0        1  No Churn
     1        1  No Churn
     2        1  No Churn
     3        1  No Churn
     4        1  No Churn
```

```
[ ]: from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.cluster import KMeans
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import accuracy_score, classification_report
```

[ ]:
```
y = df['churn']
x = df.drop('churn' , axis=1)
```

[ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪2,random_state=1)
```

[ ]:
```
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train, y_train)
```

[ ]:
```
DecisionTreeClassifier(random_state=42)
```

[ ]:
```
y_pred = model.predict(x_test)
```

[ ]:
```
print("\nBaseline Model (No Sampling) Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```

Baseline Model (No Sampling) Performance:
Accuracy: 0.981
              precision    recall  f1-score   support

       Churn       0.95      0.86      0.90       100
    No Churn       0.98      0.99      0.99       900

    accuracy                           0.98      1000
   macro avg       0.96      0.93      0.95      1000
weighted avg       0.98      0.98      0.98      1000
```

Simple Random Sampling

[ ]:
```
churn_sample = df.sample(frac=0.3, random_state=42)
```

[ ]:
```
x_sample = churn_sample.drop(columns=['churn'])
y_sample = churn_sample['churn']
```

[ ]:
```
x_train_srs, x_test_srs, y_train_srs, y_test_srs = train_test_split(x_sample,␣
 ↪y_sample, test_size=0.3, random_state=42)
```

[ ]:
```
model.fit(x_train_srs, y_train_srs)
y_pred_srs = model.predict(x_test_srs)
```

```
[ ]: print("\nSimple Random Sampling Performance:")
     print("Accuracy:", accuracy_score(y_test_srs, y_pred_srs))
     print(classification_report(y_test_srs, y_pred_srs))
```

```
Simple Random Sampling Performance:
Accuracy: 0.9711111111111111
              precision    recall  f1-score   support

       Churn       0.80      0.86      0.83        37
    No Churn       0.99      0.98      0.98       413

    accuracy                           0.97       450
   macro avg       0.89      0.92      0.91       450
weighted avg       0.97      0.97      0.97       450
```

Stratified Sampling

```
[ ]: x_train_strat, x_test_strat, y_train_strat, y_test_strat = train_test_split(x,␣
      ↪y, train_size=0.3, stratify=y, random_state=42)
```

```
[ ]: model.fit(x_train_strat, y_train_strat)
     y_pred_strat = model.predict(x_test_strat)
```

```
[ ]: print("\nStratified Sampling Performance:")
     print("Accuracy:", accuracy_score(y_test_strat, y_pred_strat))
     print(classification_report(y_test_strat, y_pred_strat))
```

```
Stratified Sampling Performance:
Accuracy: 0.97
              precision    recall  f1-score   support

       Churn       0.83      0.87      0.85       336
    No Churn       0.99      0.98      0.98      3164

    accuracy                           0.97      3500
   macro avg       0.91      0.93      0.92      3500
weighted avg       0.97      0.97      0.97      3500
```

```
[ ]: from sklearn.preprocessing import StandardScaler
     from sklearn.impute import SimpleImputer
```

```
[ ]: scaler = StandardScaler()
     x_scaled = scaler.fit_transform(x)
```

[ ]:
```
imputer = SimpleImputer(strategy='mean')
x_scaled_imputed = imputer.fit_transform(x_scaled)
```

[ ]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(x_scaled_imputed)
```

[ ]:
```
adaptive_sample = df.groupby('Cluster').sample(frac=0.3, random_state=42)
```

[ ]:
```
x_adaptive = adaptive_sample.drop(columns=['churn', 'Cluster'])
y_adaptive = adaptive_sample['churn']
```

[ ]:
```
X_train_adapt, X_test_adapt, y_train_adapt, y_test_adapt =␣
↪train_test_split(x_adaptive, y_adaptive, test_size=0.3, random_state=42)
```

[ ]:
```
model.fit(X_train_adapt, y_train_adapt)
y_pred_adapt = model.predict(X_test_adapt)
```

[ ]:
```
print("\nAdaptive Sampling Performance:")
print("Accuracy:", accuracy_score(y_test_adapt, y_pred_adapt))
print(classification_report(y_test_adapt, y_pred_adapt))
```

```
Adaptive Sampling Performance:
Accuracy: 0.9733333333333334
              precision    recall  f1-score   support

       Churn       0.89      0.85      0.87        47
    No Churn       0.98      0.99      0.99       403

    accuracy                           0.97       450
   macro avg       0.94      0.92      0.93       450
weighted avg       0.97      0.97      0.97       450
```

[ ]:
```
print("\nOriginal Dataset Statistics:")
print(df.describe())
print("\nSimple Random Sampling Statistics:")
print(churn_sample.describe())
print("\nStratified Sampling Statistics:")
print(pd.DataFrame(x_train_strat).describe())
print("\nAdaptive Sampling (K-Means) Statistics:")
print(adaptive_sample.describe())
```

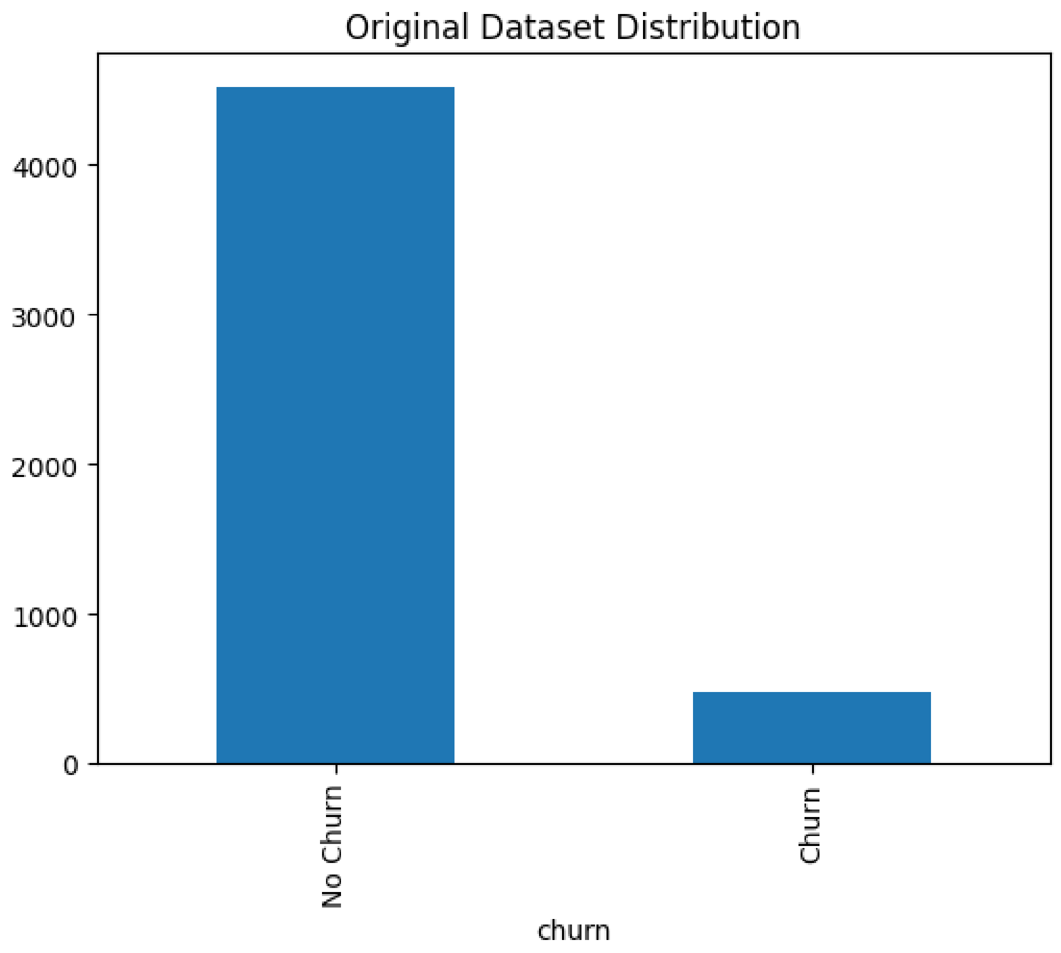
```
Original Dataset Statistics:
               age       gender         calls          sms          mms  \
count  4994.000000  5000.000000  5.000000e+03  5000.000000  5000.000000
```

```
mean      1.881057     0.294000  8.739676e-17    45.338400    20.104600
std       0.839796     0.455637  1.000000e+00    11.463166    11.467954
min       1.000000     0.000000 -1.266454e+00    23.000000    -3.000000
25%       1.000000     0.000000 -8.954939e-01    35.000000    10.000000
50%       2.000000     0.000000 -1.535738e-01    45.000000    20.000000
75%       3.000000     1.000000  7.738264e-01    55.000000    30.000000
max       3.000000     1.000000  1.608487e+00    67.000000    43.000000

            charges     coverage    complaint          sim   phone  \
count  5.000000e+03  5000.000000  5000.000000  5000.000000  5000.0
mean   8.526513e-17     0.719000     2.388600     0.104400     0.0
std    1.000000e+00     1.233184     1.154061     0.305809     0.0
min   -1.428826e+00     0.000000     0.000000     0.000000     0.0
25%   -7.554070e-01     0.000000     1.000000     0.000000     0.0
50%   -2.123269e-01     0.000000     2.000000     0.000000     0.0
75%    5.262619e-01     2.000000     3.000000     0.000000     0.0
max    3.263385e+00     7.000000     4.000000     1.000000     0.0

           prepost      Cluster
count  5000.000000  5000.000000
mean      0.403200     0.999200
std       0.490589     0.946456
min       0.000000     0.000000
25%       0.000000     0.000000
50%       0.000000     1.000000
75%       1.000000     2.000000
max       1.000000     2.000000

Simple Random Sampling Statistics:
               age       gender        calls          sms          mms  \
count  1498.000000  1500.000000  1500.000000  1500.000000  1500.000000
mean      1.875167     0.284667     0.009729    45.924667    20.707333
std       0.844510     0.451406     1.004375    11.409489    11.426083
min       1.000000     0.000000    -1.266454    23.000000    -3.000000
25%       1.000000     0.000000    -0.895494    36.000000    11.000000
50%       2.000000     0.000000    -0.060834    46.000000    21.000000
75%       3.000000     1.000000     0.866566    55.000000    30.000000
max       3.000000     1.000000     1.608487    67.000000    43.000000

           charges     coverage    complaint          sim   phone      prepost
count  1500.000000  1500.000000  1500.000000  1500.000000  1500.0  1500.000000
mean      0.009626     0.702667     2.418667     0.106667     0.0     0.405333
std       1.016614     1.198562     1.151348     0.308792     0.0     0.491120
min      -1.428826     0.000000     0.000000     0.000000     0.0     0.000000
25%      -0.755407     0.000000     1.000000     0.000000     0.0     0.000000
50%      -0.212327     0.000000     2.000000     0.000000     0.0     0.000000
75%       0.591432     1.000000     4.000000     0.000000     0.0     1.000000
max       2.828921     7.000000     4.000000     1.000000     0.0     1.000000
```

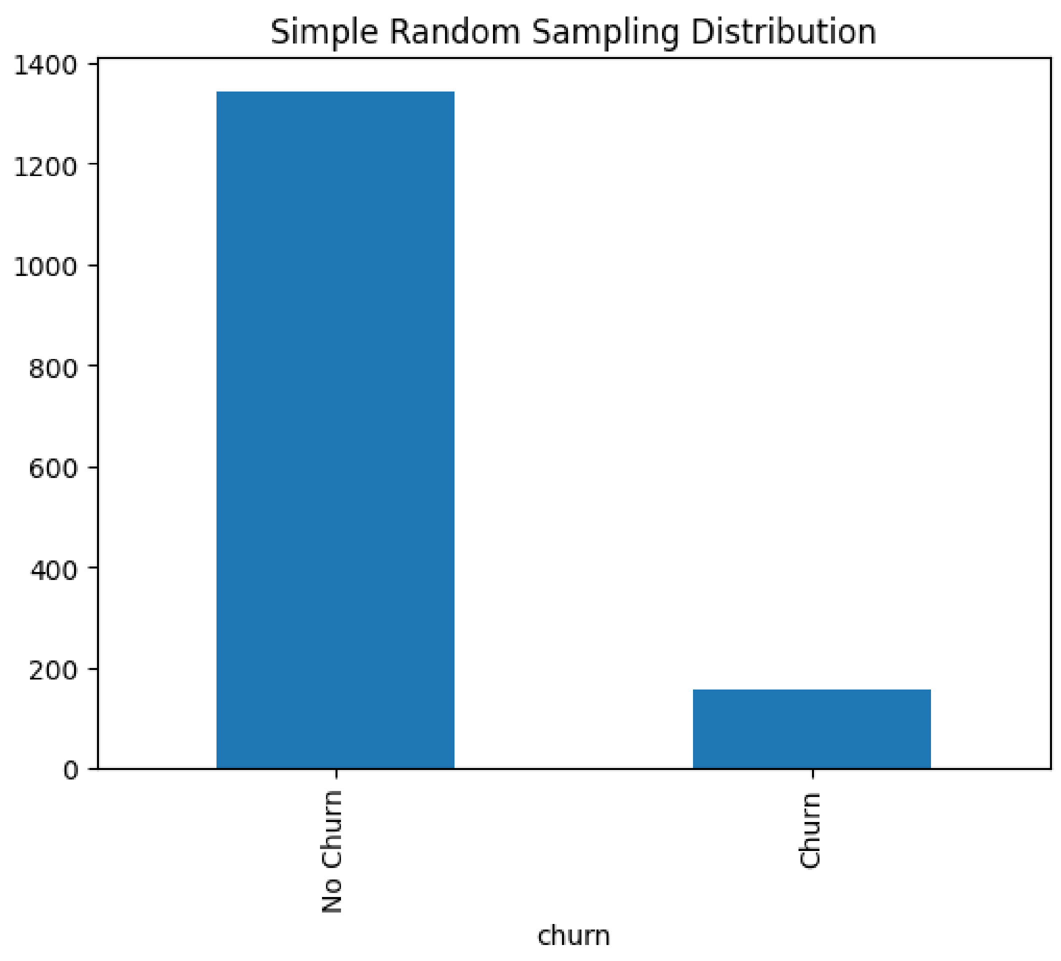
```
Stratified Sampling Statistics:
               age       gender        calls          sms          mms  \
count  1497.000000  1500.000000  1500.000000  1500.000000  1500.000000
mean      1.888444     0.291333    -0.032171    44.896667    19.667333
std       0.834728     0.454528     1.014331    11.089754    11.087288
min       1.000000     0.000000    -1.266454    23.000000    -3.000000
25%       1.000000     0.000000    -0.988234    35.000000    10.000000
50%       2.000000     0.000000    -0.153574    45.000000    20.000000
75%       3.000000     1.000000     0.773826    54.000000    29.000000
max       3.000000     1.000000     1.608487    66.000000    41.000000

           charges     coverage   complaint          sim   phone      prepost
count  1500.000000  1500.000000  1500.00000  1500.000000  1500.0  1500.000000
mean     -0.008476     0.718667     2.36200     0.092667     0.0     0.408667
std       1.021317     1.242798     1.14826     0.290061     0.0     0.491751
min      -1.428826     0.000000     0.00000     0.000000     0.0     0.000000
25%      -0.777130     0.000000     1.00000     0.000000     0.0     0.000000
50%      -0.255773     0.000000     2.00000     0.000000     0.0     0.000000
75%       0.553416     2.000000     3.00000     0.000000     0.0     1.000000
max       2.850645     6.000000     4.00000     1.000000     0.0     1.000000

Adaptive Sampling (K-Means) Statistics:
               age       gender        calls          sms          mms  \
count  1499.000000  1500.000000  1500.000000  1500.000000  1500.000000
mean      1.895264     0.302667     0.010533    45.291333    20.045333
std       0.843002     0.459565     1.002715    11.490684    11.529089
min       1.000000     0.000000    -1.266454    23.000000    -2.000000
25%       1.000000     0.000000    -0.895494    35.000000    10.000000
50%       2.000000     0.000000    -0.060834    45.000000    20.000000
75%       3.000000     1.000000     0.866566    55.000000    30.000000
max       3.000000     1.000000     1.608487    67.000000    43.000000

           charges     coverage    complaint          sim   phone  \
count  1500.000000  1500.000000  1500.000000  1500.000000  1500.0
mean     -0.000149     0.743333     2.387333     0.104667     0.0
std       1.017994     1.263271     1.163705     0.306226     0.0
min      -1.428826     0.000000     0.000000     0.000000     0.0
25%      -0.755407     0.000000     1.000000     0.000000     0.0
50%      -0.255773     0.000000     2.000000     0.000000     0.0
75%       0.461092     2.000000     3.000000     0.000000     0.0
max       2.850645     6.000000     4.000000     1.000000     0.0

           prepost      Cluster
count  1500.000000  1500.000000
mean      0.398667     0.999333
std       0.489787     0.946536
min       0.000000     0.000000
```

```
25%        0.000000     0.000000
50%        0.000000     1.000000
75%        1.000000     2.000000
max        1.000000     2.000000
```

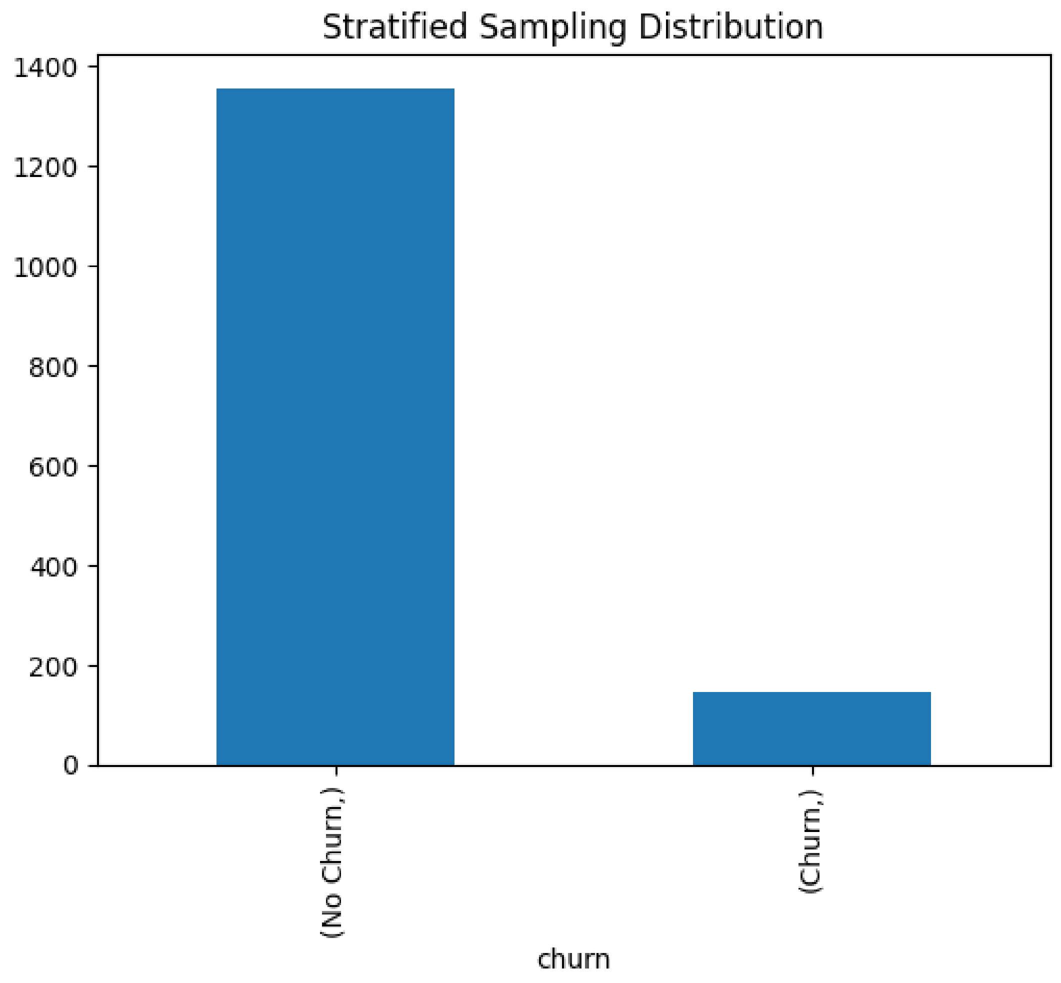
[ ]:
```
import matplotlib.pyplot as plt
df['churn'].value_counts().plot(kind='bar', title='Original Dataset␣
  ↪Distribution')
plt.show()
```



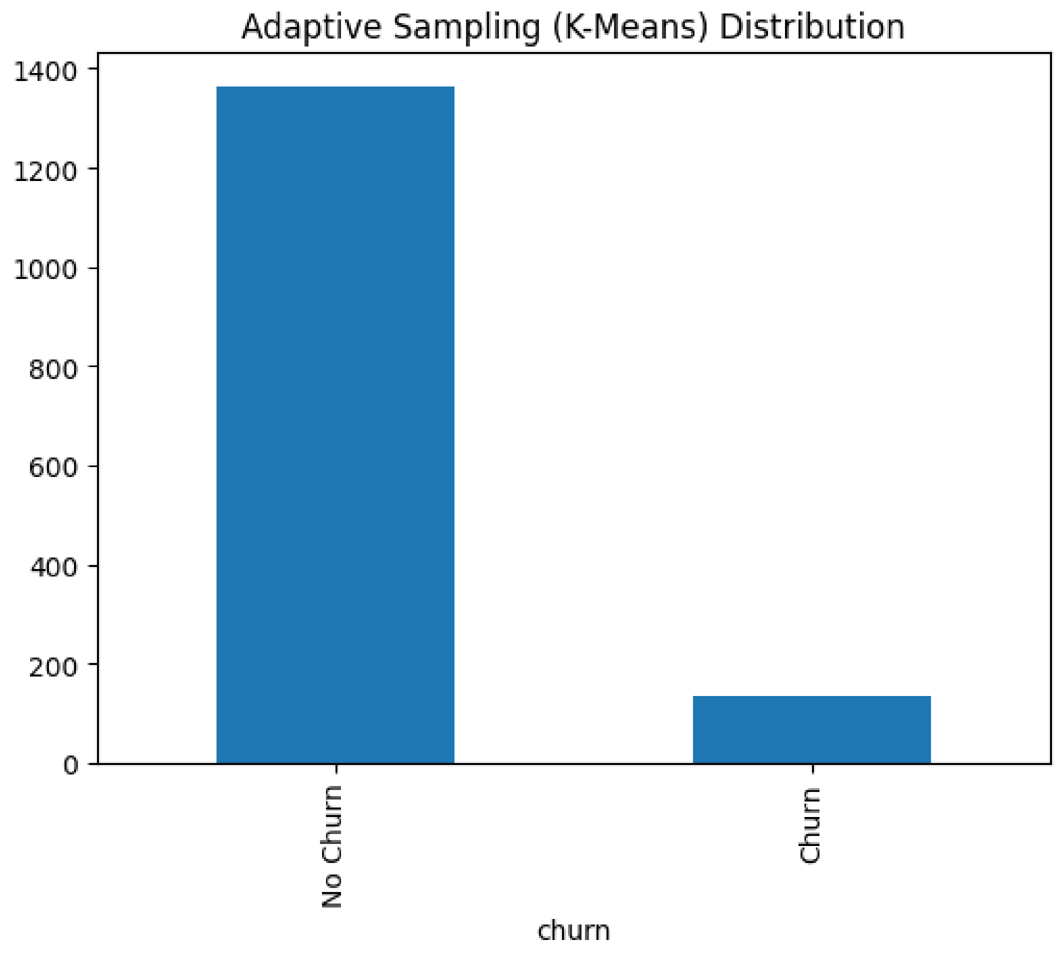
[ ]:
```
churn_sample['churn'].value_counts().plot(kind='bar', title='Simple Random␣
  ↪Sampling Distribution')
plt.show()
```



```
[ ]: pd.DataFrame({'churn': y_train_strat}).value_counts().plot(kind='bar',␣
       ↪title='Stratified Sampling Distribution')
     plt.show()
```



```
[ ]: adaptive_sample['churn'].value_counts().plot(kind='bar', title='Adaptive␣
       ↪Sampling (K-Means) Distribution')
     plt.show()
```



```
[ ]: import numpy as np
     from sklearn.metrics import pairwise_distances

     original_variance = np.var(x, axis=0)
     random_variance = np.var(x_sample, axis=0)
     stratified_variance = np.var(x_train_strat, axis=0)
     adaptive_variance = np.var(x_adaptive, axis=0)
     print("\nFeature Variance (Original Dataset):", original_variance)
     print("Feature Variance (Random Sampling):", random_variance)
     print("Feature Variance (Stratified Sampling):", stratified_variance)
     print("Feature Variance (Adaptive Sampling):", adaptive_variance)
     adaptive_spread = np.mean(pairwise_distances(x_scaled_imputed)[kmeans.labels_␣
      ↪== kmeans.labels_[:, None]])
     print("\nAdaptive Sampling (K-Means) Cluster Spread:", adaptive_spread)
```

```
Feature Variance (Original Dataset): age            0.705116
```

```
gender         0.207564
calls          0.999800
sms          131.377885
mms          131.487659
charges        0.999800
coverage       1.520439
complaint      1.331590
sim            0.093501
phone          0.000000
prepost        0.240630
dtype: float64
Feature Variance (Random Sampling): age            0.712721
gender         0.203632
calls          1.008096
sms          130.089658
mms          130.468346
charges        1.032814
coverage       1.435593
complaint      1.324718
sim            0.095289
phone          0.000000
prepost        0.241038
dtype: float64
Feature Variance (Stratified Sampling): age            0.696306
gender         0.206458
calls          1.028182
sms          122.900656
mms          122.846000
charges        1.042394
coverage       1.543518
complaint      1.317623
sim            0.084080
phone          0.000000
prepost        0.241658
dtype: float64
Feature Variance (Adaptive Sampling): age            0.710178
gender         0.211060
calls          1.004767
sms          131.947792
mms          132.831278
charges        1.035622
coverage       1.594789
complaint      1.353306
sim            0.093712
phone          0.000000
prepost        0.239732
dtype: float64
```

```
Adaptive Sampling (K-Means) Cluster Spread: 3.757266708546486
```