Importing Libraries and Loading dataset

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

#loading dataset
df=pd.read_csv('/content/insurance_data.csv')
df
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospita
0	18.0	male	23.210	0.0	no	29087.54313	17.0	715428.0	4.720921e+06	
1	18.0	male	30.140	0.0	no	39053.67437	7.0	699157.0	4.329832e+06	
2	18.0	male	33.330	0.0	no	39023.62759	19.0	702341.0	6.884861e+06	
3	18.0	male	33.660	0.0	no	28185.39332	11.0	700250.0	4.274774e+06	
4	18.0	male	34.100	0.0	no	14697.85941	16.0	711584.0	3.787294e+06	
...	...	...	...	...	...	...	...	...	...	
1333	33.0	female	35.530	0.0	yes	63142.25346	32.0	1091267.0	1.703805e+08	
1334	31.0	female	38.095	1.0	yes	43419.95227	31.0	1107872.0	2.015152e+08	
1335	52.0	male	34.485	3.0	yes	52458.92353	25.0	1092005.0	2.236450e+08	
1336	45.0	male	30.360	0.0	yes	69927.51664	34.0	1106821.0	2.528924e+08	
1337	54.0	female	47.410	0.0	yes	63982.80926	31.0	1100328.0	2.616317e+08	

1338 rows × 13 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

EDA

```
#check for null values , duplicate values , get brief info of this dataset
#check number of rows and columns in the dataset

df.info() #info of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   age                           1329 non-null   float64
 1   sex                           1338 non-null   object 
 2   bmi                           1335 non-null   float64
 3   children                      1333 non-null   float64
 4   smoker                        1338 non-null   object 
 5   Claim_Amount                  1324 non-null   float64
 6   past_consultations            1332 non-null   float64
 7   num_of_steps                  1335 non-null   float64
 8   Hospital_expenditure          1334 non-null   float64
 9   NUmber_of_past_hospitalizations  1336 non-null   float64
 10  Anual_Salary                  1332 non-null   float64
 11  region                        1338 non-null   object 
 12  charges                       1338 non-null   float64
dtypes: float64(10), object(3)
memory usage: 136.0+ KB
```

```
df.isnull().sum() #check total no.of null values in each column
```

	0
age	9
sex	0
bmi	3
children	5
smoker	0
Claim_Amount	14
past_consultations	6
num_of_steps	3
Hospital_expenditure	4
NUmber_of_past_hospitalizations	2
Anual_Salary	6
region	0
charges	0

dtype: int64

```
df.isnull().sum().sum() #to check the total null of null values in the dataset
```

```
np.int64(52)
```

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

```
np.int64(0)
```

```
#if i want to know how many people are smoker and how many are not
df['smoker'].value_counts()
```

smoker	count
no	1064
yes	274

dtype: int64

```
#if we have null value in numerical column we will replace it with mean or median
#if in categorical column we will replace it with mode
```

```
col_list=list(df.columns)
print(col_list)

for x in col_list:
  if df[x].dtypes=='object':
    df[x].fillna(df[x].mode()[0],inplace=True)
  else:
    df[x].fillna(df[x].mean(),inplace=True)
```

```
['age', 'sex', 'bmi', 'children', 'smoker', 'Claim_Amount', 'past_consultations', 'num_of_steps', 'Hospital_expenditure', 'NUmber_of_pas
<ipython-input-8-e2eea41e5016>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignme
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[x].fillna(df[x].mean(),inplace=True)
<ipython-input-8-e2eea41e5016>:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignme
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[x].fillna(df[x].mode()[0],inplace=True)
```

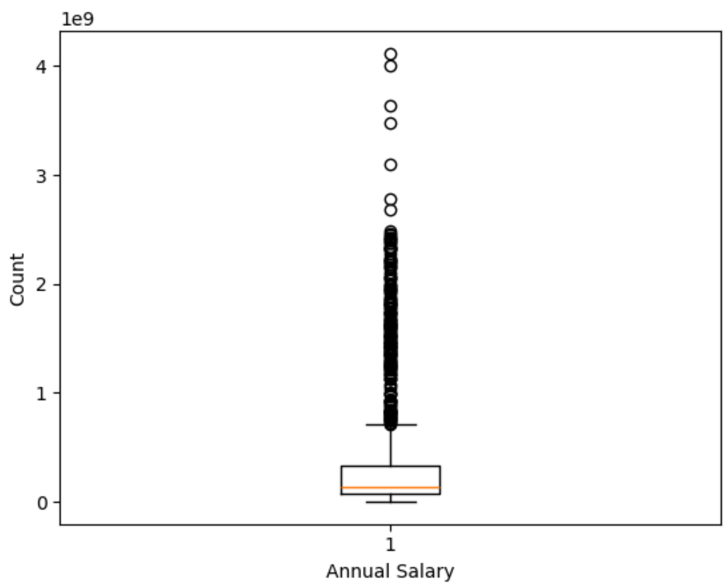
```
df.isnull().sum() #null values are replaced
```

	0
age	0
sex	0
bmi	0
children	0
smoker	0
Claim_Amount	0
past_consultations	0
num_of_steps	0
Hospital_expenditure	0
NUmber_of_past_hospitalizations	0
Anual_Salary	0
region	0
charges	0

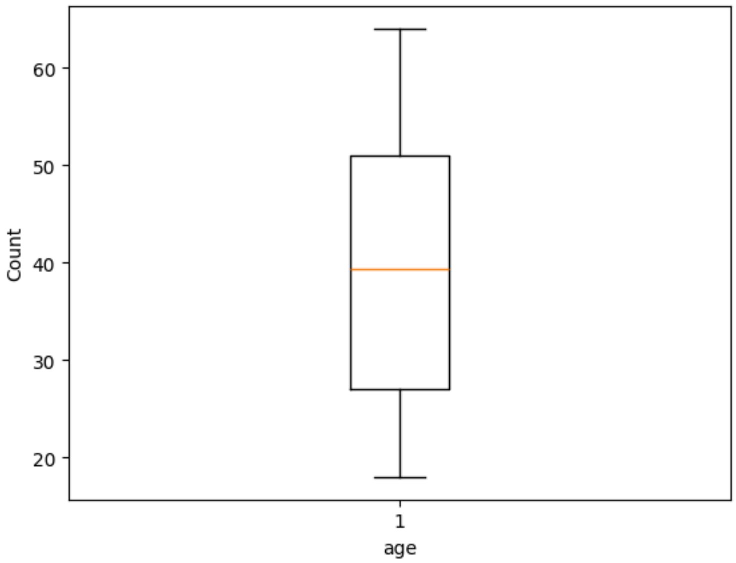
dtype: int64

Outliers Detection

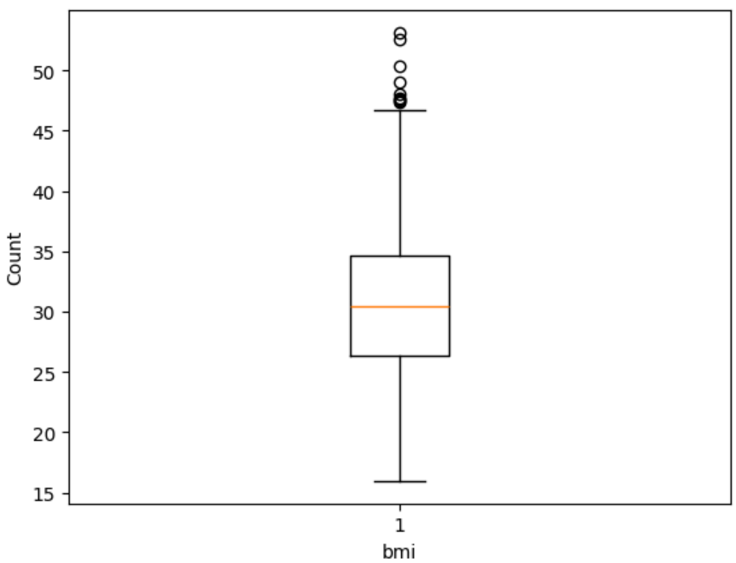
```
#outliers detection
plt.boxplot(df['Anual_Salary'])
plt.xlabel('Annual Salary')
plt.ylabel('Count')
plt.show()
```



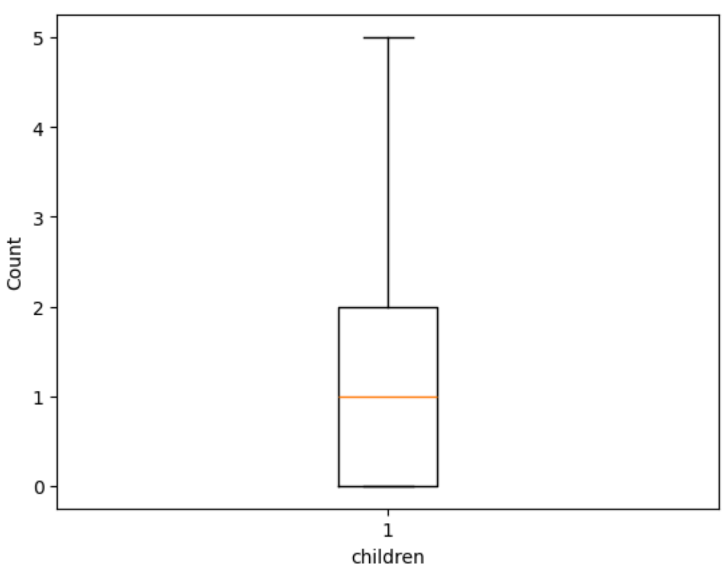

```
#outlier detection for all the columns in the dataset
for x in col_list:
  if df[x].dtypes=='int64' or df[x].dtypes=='float64':
    plt.boxplot(df[x])
    plt.xlabel(x)
    plt.ylabel('Count')
    plt.show()
```



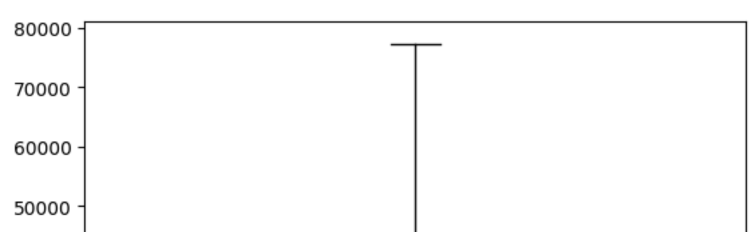
60
50
40
30
20
Count
1
age



50
45
40
35
30
25
20
15
Count
1
bmi

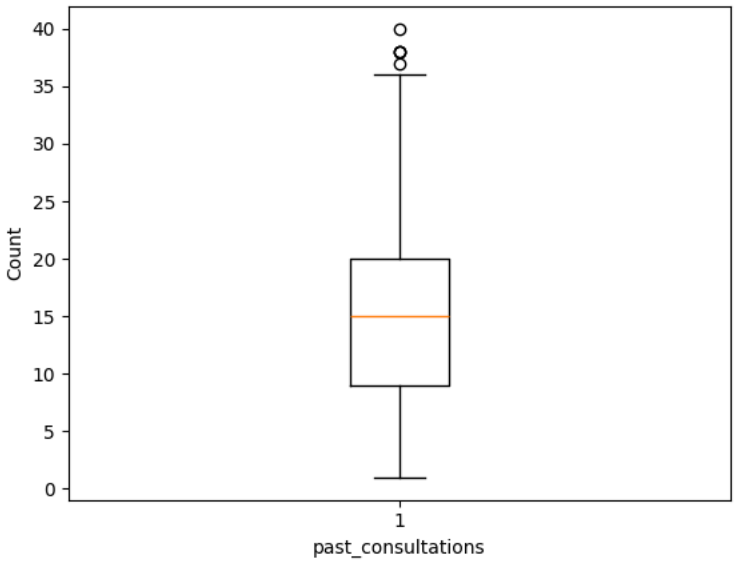


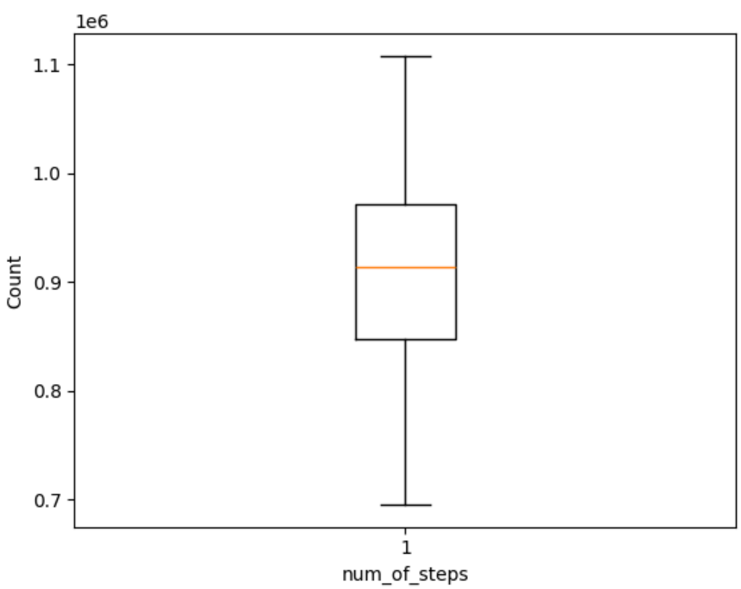
5
4
3
2
1
0
Count
1
children

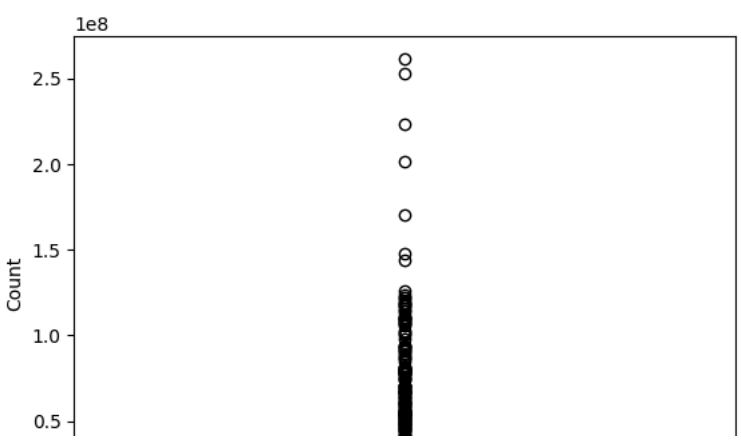


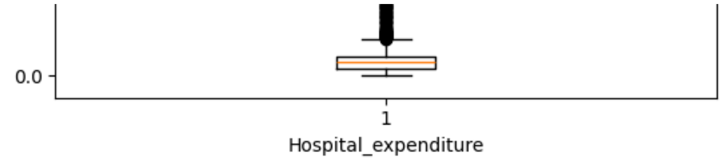
80000
70000
60000
50000

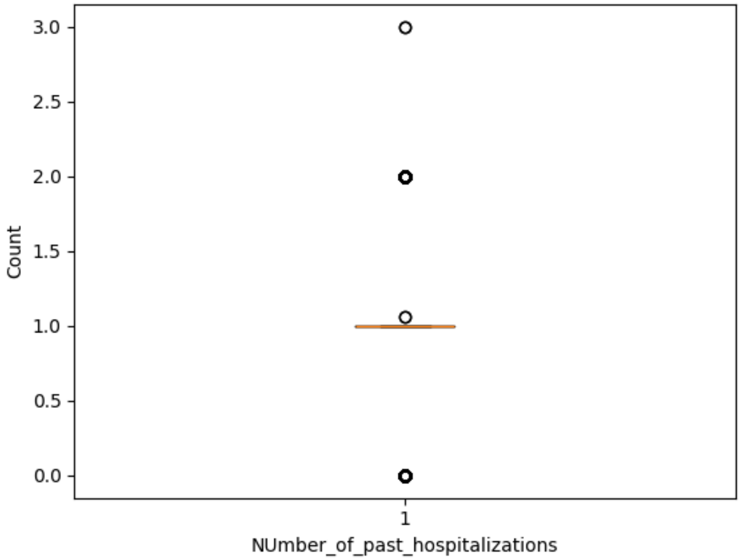

Count
40000
30000
20000
10000
0
1
Claim_Amount

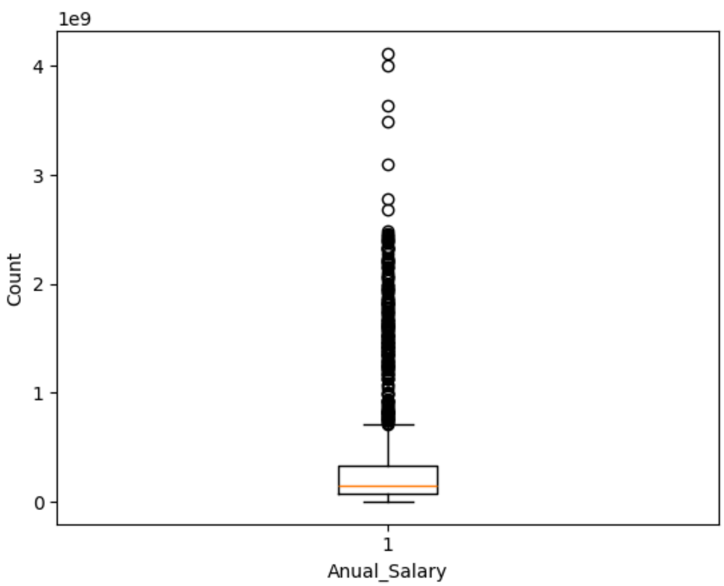

40
35
30
25
20
15
10
5
0
Count
1
past_consultations

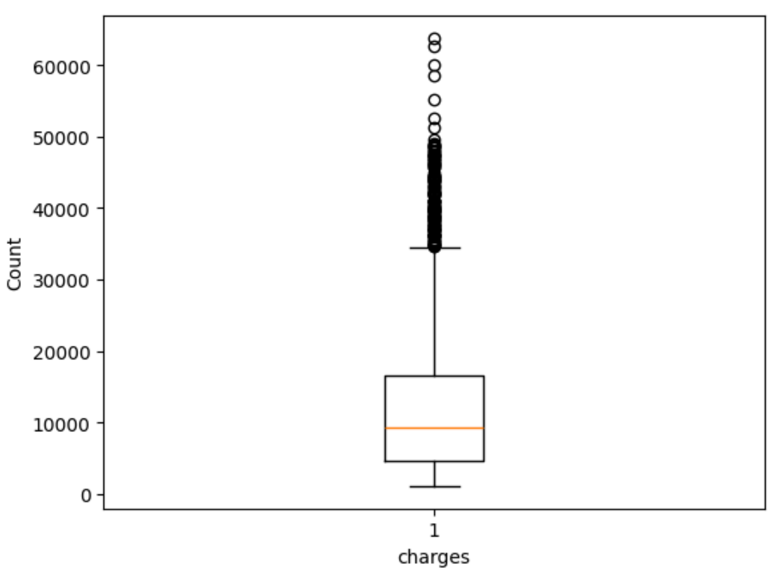

1e6
1.1
1.0
0.9
0.8
0.7
Count
1
num_of_steps


1e8
2.5
2.0
1.5
1.0
0.5
Count


0.0
1
Hospital_expenditure


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Count
1
NUmber_of_past_hospitalizations


1e9
4
3
2
1
0
Count
1
Anual_Salary


60000
50000
40000
30000
20000
10000
0
Count
1
charges

Now we have detected the outliers, we have to deal with them or remove them

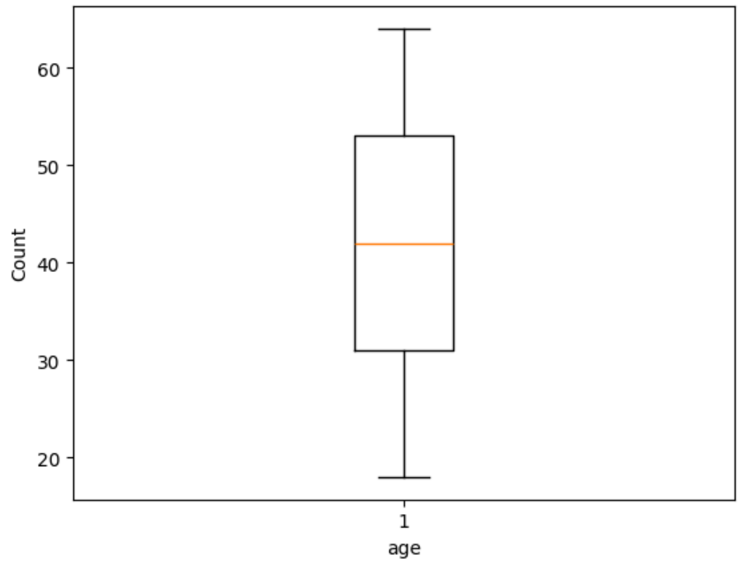
```
#finding the lower_bound and upper_bound values of bmi column

Q1= df.bmi.quantile(0.25)
Q3=df.bmi.quantile(0.75)

IQR= Q3-Q1

lower_bound= Q1-1.5*IQR
upper_bound= Q3+1.5*IQR

print(lower_bound)
print(upper_bound)
```

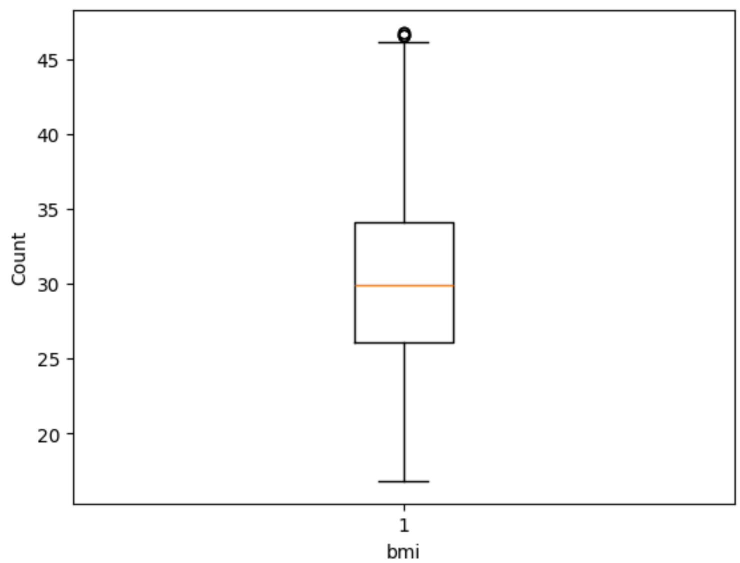
```
13.803125000000003
47.168124999999996
```

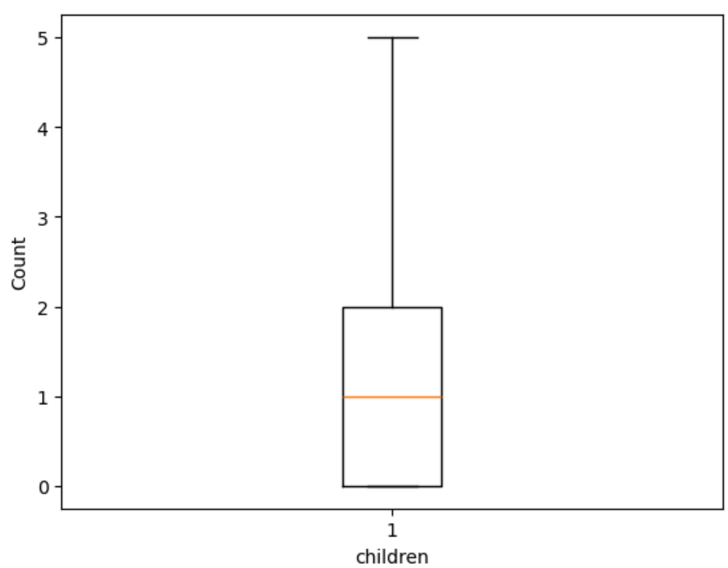
```
#finding all the lower_bound and upper_bound values of all the numerical columns and removing outliers

col_list=list(df.columns)
print(col_list)
for x in col_list:
  if df[x].dtypes=='object' or x=='charges':
    continue
  else:
    Q1=df[x].quantile(0.25)
    Q3=df[x].quantile(0.75)
    IQR=Q3 - Q1
    lower_bound=Q1-1.5*IQR
    upper_bound= Q3+1.5*IQR
    print(f"lower bound of {x} is {lower_bound}")
    print(f"upper bound of {x} is {upper_bound}")
    df = df[(df[x]>=lower_bound) & (df[x]<=upper_bound)]
```

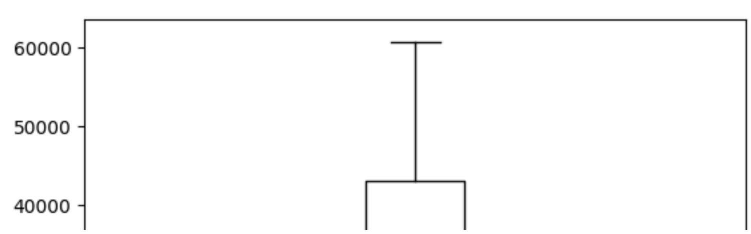
```
['age', 'sex', 'bmi', 'children', 'smoker', 'Claim_Amount', 'past_consultations', 'num_of_steps', 'Hospital_expenditure', 'NUmber_of_pas
lower bound of age is -9.0
upper bound of age is 87.0
lower bound of bmi is 13.803125000000003
upper bound of bmi is 47.168124999999996
lower bound of children is -3.0
upper bound of children is 5.0
lower bound of Claim_Amount is -15079.880879999993
upper bound of Claim_Amount is 80991.7952
lower bound of past_consultations is -7.5
upper bound of past_consultations is 36.5
lower bound of num_of_steps is 662100.125
upper bound of num_of_steps is 1155245.125
lower bound of Hospital_expenditure is -6066324.56875
upper bound of Hospital_expenditure is 20930808.357249998
lower bound of NUmber_of_past_hospitalizations is 1.0
upper bound of NUmber_of_past_hospitalizations is 1.0
lower bound of Anual_Salary is -99592205.85625002
upper bound of Anual_Salary is 384246561.35375
```

```
#vizual representation of the dataset after removing outliers

import matplotlib.pyplot as plt

for x in col_list:
  if df[x].dtypes=='object' or x=='charges':
    continue
  else:
    plt.boxplot(df[x])
    plt.xlabel(x)
    plt.ylabel('Count')
    plt.show()
```

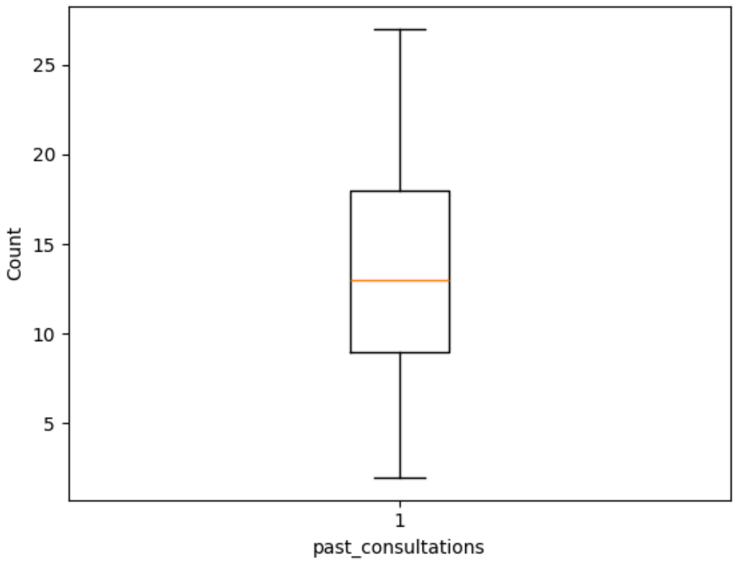

60
50
40
30
20
Count
1
age

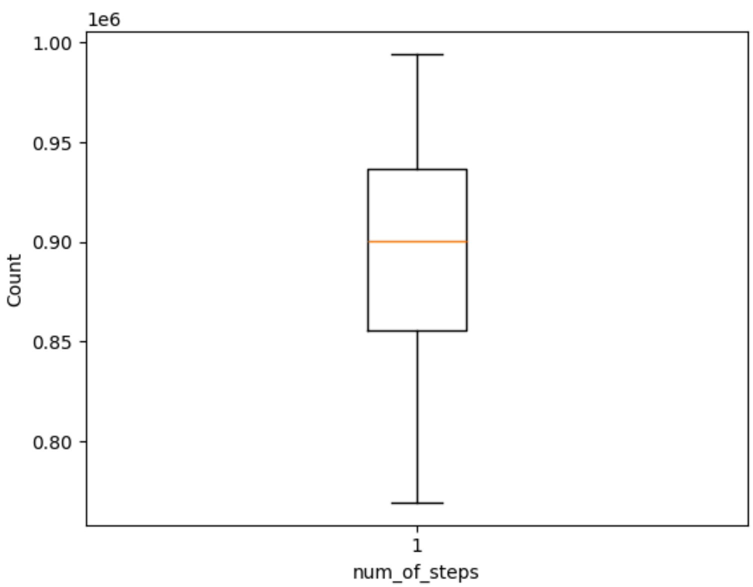

45
40
35
30
25
20
Count
1
bmi

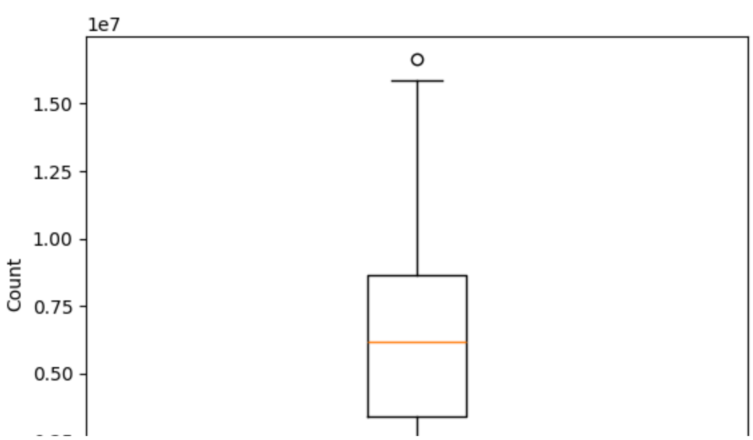

5
4
3
2
1
0
Count
1
children

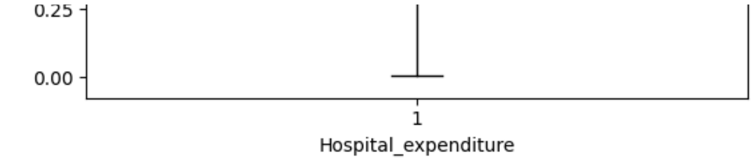

60000
50000
40000

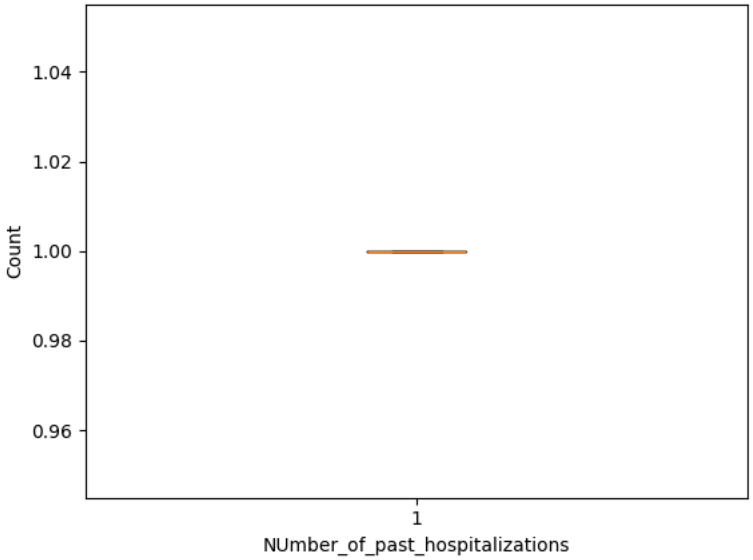

Count
30000
20000
10000
1
Claim_Amount

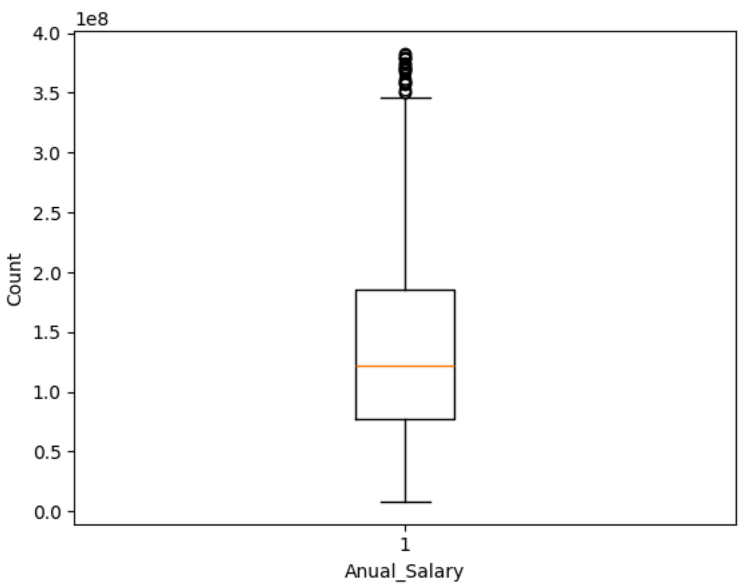

25
20
15
10
5
Count
1
past_consultations


1e6
1.00
0.95
0.90
0.85
0.80
Count
1
num_of_steps


1e7
1.50
1.25
1.00
0.75
0.50
Count


0.25
0.00
1
Hospital_expenditure


1.04
1.02
1.00
0.98
0.96
Count
1
NUmber_of_past_hospitalizations


1e8
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Count
1
Anual_Salary

```
#There can be a possibility that catergorical columns can also be affecting my model
#But my model cannot understand the catergorical columns
#so i should convert the catergorical columns into numerical columns.
```

Encoding

```
#Encoding - the process of converting categorical (non-numerical) data into a numerical format that machine learning algorithms can understa
#Common Encoding Techniques
# 1.One-Hot Encoding
# 2.Label Encoding
# 3.Ordinal Encoding
# 4.Mean Encoding
# 5.Frequency Encoding
```

```
#Label Encoding - Assigns a unique numerical label to each category
#We have to import Labelencoding sklearn.preprocessing
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for x in col_list:
  if df[x].dtypes=='object':
    df[x]=le.fit_transform(df[x])
```

```
df
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospitali:
151	25.0	1	27.550	0.0	0	39148.95495	10.0	780652.0	8.614147e+06	
152	22.0	0	20.235	0.0	0	41547.52536	13.0	802627.0	2.491594e+05	
153	25.0	1	35.625	0.0	0	39660.60193	12.0	770773.0	3.043323e+06	
154	20.0	1	31.130	2.0	0	16032.87148	7.0	769255.0	1.599069e+06	
155	21.0	0	17.400	1.0	0	31090.98977	21.0	778769.0	3.015365e+06	
...	...	...	...	...	...	...	...	...	...	
1046	29.0	0	27.940	1.0	1	51168.25474	23.0	993751.0	1.665982e+07	
1048	31.0	1	25.900	3.0	1	46619.40230	27.0	989387.0	1.361938e+07	
1050	31.0	1	29.810	0.0	1	24382.58056	21.0	973924.0	1.028991e+07	
1062	43.0	0	20.045	2.0	1	21596.43846	10.0	994419.0	1.083030e+07	
1069	35.0	0	28.025	0.0	1	17200.14586	15.0	993979.0	1.247744e+07	

881 rows × 13 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Model Building

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
# x_train -> Training questions  independent columns
# y_train -> Training answers    dependent column
# x_test -> Testing questions of independent columns (some rows that I won't pass to the model)
# y_test -> Testing answers of dependent column (To analyse the performance of the model)
```

```
X=df.iloc[:,:12]
Y=df.iloc[:,-1]
```

```
X #independent columns
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospitali
151	25.0	1	27.550	0.0	0	39148.95495	10.0	780652.0	8.614147e+06	
152	22.0	0	20.235	0.0	0	41547.52536	13.0	802627.0	2.491594e+05	
153	25.0	1	35.625	0.0	0	39660.60193	12.0	770773.0	3.043323e+06	
154	20.0	1	31.130	2.0	0	16032.87148	7.0	769255.0	1.599069e+06	
155	21.0	0	17.400	1.0	0	31090.98977	21.0	778769.0	3.015365e+06	
...	...	...	...	...	...	...	...	...	...	
1046	29.0	0	27.940	1.0	1	51168.25474	23.0	993751.0	1.665982e+07	
1048	31.0	1	25.900	3.0	1	46619.40230	27.0	989387.0	1.361938e+07	
1050	31.0	1	29.810	0.0	1	24382.58056	21.0	973924.0	1.028991e+07	
1062	43.0	0	20.045	2.0	1	21596.43846	10.0	994419.0	1.083030e+07	
1069	35.0	0	28.025	0.0	1	17200.14586	15.0	993979.0	1.247744e+07	

881 rows × 12 columns

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
Y #dependent columns
```

	charges
151	2523.16950
152	2527.81865
153	2534.39375
154	2566.47070
155	2585.26900
...	...
1046	19107.77960
1048	19199.94400
1050	19350.36890
1062	19798.05455
1069	20234.85475

881 rows × 1 columns

dtype: float64

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,train_size=0.75)
```

```
X_train #Training data (questions)
```

	age	sex	bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospitaliz
838	55.0	0	30.140	2.0	0	9056.42148	24.0	944311.0	6.361727e+06	
698	54.0	1	31.600	0.0	0	24661.25182	5.0	922267.0	7.844040e+06	
236	27.0	0	23.210	1.0	0	11003.85343	4.0	815134.0	4.585253e+06	
779	56.0	0	39.820	0.0	0	28716.57206	22.0	942062.0	8.381702e+06	
851	54.0	0	30.800	3.0	0	36160.15855	9.0	933932.0	1.093410e+07	
...	...	...	...	...	...	...	...	...	...	
180	24.0	0	29.925	0.0	0	47697.21927	12.0	801145.0	9.343137e+06	
677	46.0	1	24.795	3.0	0	53204.28189	15.0	928337.0	1.256063e+06	
755	55.0	1	21.500	1.0	0	12121.20912	17.0	925692.0	3.808804e+06	
610	47.0	0	24.320	0.0	0	12207.16184	7.0	914552.0	3.474461e+06	
885	60.0	0	35.100	0.0	0	51236.61127	7.0	936964.0	8.250142e+06	

660 rows × 12 columns

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
Y_train #training data answers.
```

	charges
838	11881.96960
698	9850.43200
236	3561.88890
779	11090.71780
851	12105.32000
...	...
180	2850.68375
677	9500.57305
755	10791.96000
610	8534.67180
885	12644.58900

660 rows × 1 columns

dtype: float64

```
X_test #Testing data. after building the model I will test my model with this data.
```

bmi	children	smoker	Claim_Amount	past_consultations	num_of_steps	Hospital_expenditure	NUmber_of_past_hospitaliza
29.370	1.0	0	34628.746980	4.0	892575.0	5.763826e+06	
28.310	1.0	0	33295.679460	15.0	919699.0	8.146261e+06	
27.170	0.0	0	54067.641030	22.0	900391.0	6.156272e+06	
38.830	0.0	0	18763.140540	10.0	939384.0	5.550482e+06	
46.700	2.0	0	32446.156670	11.0	925256.0	1.123172e+07	
...	...	...	...	...	...	...	
35.625	4.0	0	17368.246560	14.0	931847.0	4.724863e+06	
31.730	2.0	0	45087.100540	14.0	915453.0	1.004051e+07	
27.455	1.0	0	32845.669720	15.0	906079.0	2.587267e+06	
41.230	4.0	0	45345.774660	11.0	927845.0	9.955031e+06	
34.800	1.0	0	6863.265468	11.0	843105.0	4.091292e+06	

ıns

Next steps: Generate code with X_test | View recommended plots | New interactive sheet

```
Y_test   #When the model will predict the values for us we will compare those values with y_test
```

	charges
613	8547.69130
788	11272.33139
619	8601.32930
901	12981.34570
812	11538.42100
...	...
754	10736.87075
784	11187.65670
684	9617.66245
774	11033.66170
375	5246.04700

221 rows × 1 columns

dtype: float64

```
linear_model=LinearRegression()
```

```
linear_model.fit(X_train,Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#now we need to test the data to check what values my model cal predict
```

```
model_predicted=linear_model.predict(X_test)
model_predicted
```

```
array([ 7765.42800061, 10376.27028334,  9034.68372458, 11716.62335944,
       10540.53235134,  3026.91067583, 10060.03543412,  6323.23519115,
       18896.4833641 , 13561.37003915,  8899.70851398,  2886.61644702,
       13006.21673345,  4003.55249521,  5442.04230199,  5882.29813661,
        4328.15882684, 12669.78023405, 11967.73673585, 16549.85176121,
        5210.91700187, 10302.74888698, 13147.0005343 ,  9971.48719266,
        9189.96048415,  3646.08454009, 14286.10326176, 11850.54630126,
        8274.30961962, 12584.44832136,  2394.31416877,  7220.84066505,
       13930.9669238 , 10276.36342053, 12849.61240149,  2551.33183055,
        7541.68463732, 11229.96172675,  7259.26979883,  7133.88021428,
```

```
17024.4147842 ,  4890.42743718,  4013.91224279,  3713.80129508,
 7393.69582227,  3134.34266248, 10872.21519587,  9292.47645094,
 8043.03114431, 11245.14603346, 11144.61420721,  7371.43140547,
11367.316831  ,  5292.87462721,  3025.42025135, 12159.09650708,
 8210.81604537,  6635.47309122,  8283.17546523,  5855.17408451,
 7082.62216918,   251.14396791,  9939.71314907,  9464.06666656,
 7390.7250398 , 10371.41001422,  6057.29097056,  6197.74884523,
 3999.10177598,  5185.82520283,  7446.5105116 ,  9163.27485816,
 9114.70630788, 12512.56080967,  7258.51559356, 13745.69310614,
 7028.20402291,  9620.29148158,  3551.14708833, 15413.74538124,
10383.8511418 ,  3229.14631246,  5644.94052637,  4410.19254374,
 8077.18022236,  6129.62539532,  6943.81590127,  8206.74418216,
 4829.73743192, 12746.64351425,  6958.69600868,  6052.29035191,
 6820.98554607,  9697.38532498,  6227.90494608,  4940.82617289,
 8153.25565356,  3203.13109048, 10535.19814707,  9415.58967579,
10661.49854251,  9175.72113436,  2850.11183887,  4728.48953844,
 8523.15958899,  7720.849622  ,  4877.74568442,  5143.70399683,
 3799.01340793,  6114.73131333,  6341.36545273, 10569.27509776,
12571.42403974, 11583.55326366, 12770.550434  ,  4998.70428695,
 3760.53476931,  9606.91442814,  6582.30801594,  2591.72279963,
 8277.98019793, 12879.52594815,  7620.61452162,  7455.56830074,
 6436.88016066,  8953.51619832,  3354.90653953, 10329.15075043,
 6116.43783156, 12999.78385295, 10410.22568834,  8246.12980801,
17024.39944319, 10713.5988916 , 16791.4401858 , 11187.39452276,
 8276.41293833,  7956.77340667, 10293.04514522,  7203.34301159,
 6768.30898229, 11343.64642423, 13338.81306625, 11575.93455769,
 2509.53210964,  9055.09912073, 12974.6114663 , 13427.94061517,
 9392.76599556, 10497.43402238, 10457.51352811, 13883.92642259,
 4739.44057094,  9237.57960582,  9733.86387244,  5041.22174967,
 5053.46660849,  9995.68992688, 17254.58819689,  1945.91406609,
10481.30167612,  5346.75540731,  8514.98372949,  5826.75007779,
12475.97531153, 14790.89381039,  9017.48285313,  7063.96833347,
 8601.02737374,  5502.85146328,  6244.68134238, 12807.03733869,
 7474.13136304,  9321.87727851,  4593.77659354,  6824.26512314,
 9250.10092418,  3392.56608659,  6918.47161105,  8818.69426094,
13454.29031684, 13304.64582447,  5711.78927429,  6032.38757978,
16815.087611  ,  2026.98665235,  9374.91008574,  3537.85107475,
12244.56419213,  4259.90805102, 12520.76633316, 14573.29619232,
 7396.26719337, 10964.14210032, 16364.3126161 ,  4931.12888833,
11898.78069533, 11672.75713649,  2635.8066275 , 17930.64847678,
12937.69435443,  5417.71562203,  2744.69454115, 12228.86164288,
 9153.16917741,  8212.82106447, 12341.40642688,  5514.61551633,
10424.24938338, 14952.66777259,  4461.01707816, 11105.73725873,
13800.84381578,  7291.64024165, 11027.8937917 , 11233.71021851,
11227.65554077, 10175.20287243,  9711.8142677 , 11244.33877183,
```

MSE,R2_score

```
#MSE -> Mean Squared Error
#RMSE -> Root Mean Squared Error
#r2_score
```

```
from sklearn.metrics import mean_squared_error

mse=mean_squared_error(Y_test,model_predicted)
mse
```

```
587936.1964791534
```

```
rmse=np.sqrt(mse)
rmse
```

```
np.float64(766.7699762504745)
```

```
from sklearn.metrics import r2_score

r2_score=r2_score(Y_test,model_predicted)
r2_score*100
```

```
95.92307201834575
```

```
sns.regplot(x=model_predicted,y=Y_test)
plt.xlabel('Predictions')
plt.ylabel('Actual Values')
plt.title('Regression Plot')
plt.show()
```