1-Understand the data :

- Upload the data and take a look of columns and data types
- Identfy the target Label

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random as rd
import warnings
import plotly.express as px
import plotly.io as pio
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv('insurance.csv')
```

Data size:

- how much the data size (columns and rows)
- shape function return the number of columns and rows

In [3]:
```
df.shape
```

Out[3]:
```
(1338, 7)
```

Data Preview:

- in this step we want to see how data Looks like
- head() display the first few rows of the dataset
- sample() display random sample of the dataset

In [4]:
```
df.head(7)
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960

In [5]:
```
df.sample(7)
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges
1019	21	female	32.68	2	no	northwest	26018.95052
527	51	female	25.80	1	no	southwest	9861.02500
1121	46	male	38.17	2	no	southeast	8347.16430
808	18	male	30.14	0	no	southeast	1131.50660
352	30	female	27.70	0	no	southwest	3554.20300
583	32	female	23.65	1	no	southeast	17626.23951
155	44	male	39.52	0	no	northwest	6948.70080

Data Types:

- check the types of the data using info() or dtypes
- info() provide information about dataset
- dtypes return the data type of each column

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [7]:
```
df.dtypes
```

Out[7]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

- Change In-correct Datatypes

In [8]:
```
cols = ["sex", "smoker", "region"]
df[cols] = df[cols].astype('category')
pd.DataFrame(df.dtypes).T
```

Out[8]:

	age	sex	bmi	children	smoker	region	charges
0	int64	category	float64	int64	category	category	float64

- check the value counts of catgorical columns

In [9]:
```
print(df['sex'].value_counts())
print('-----------------------')
print(df['smoker'].value_counts())
print('-----------------------')
print(df['region'].value_counts())
```

```
sex
male      676
female    662
Name: count, dtype: int64
-----------------------
smoker
no     1064
yes     274
Name: count, dtype: int64
-----------------------
region
southeast    364
northwest    325
southwest    325
northeast    324
Name: count, dtype: int64
```

Label encoding:

- convert string or text to numbers, so we can make analysis on it

In [10]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['sex_encoded'] = label_encoder.fit_transform(df['sex'])
df['smoker_encoded'] = label_encoder.fit_transform(df['smoker'])
```

```
df['region_encoded'] = label_encoder.fit_transform(df['region'])
df.head()
```

Out[10]:

	age	sex	bmi	children	smoker	region	charges	sex_encoded	smoker_encoded	region_encoded
0	19	female	27.900	0	yes	southwest	16884.92400	0	1	3
1	18	male	33.770	1	no	southeast	1725.55230	1	0	2
2	28	male	33.000	3	no	southeast	4449.46200	1	0	2
3	33	male	22.705	0	no	northwest	21984.47061	1	0	1
4	32	male	28.880	0	no	northwest	3866.85520	1	0	1

Missing Values:

- check if nulls or missing values is exist
- isnull().sum() gives the total number of missing values per column

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
age               0
sex               0
bmi               0
children          0
smoker            0
region            0
charges           0
sex_encoded       0
smoker_encoded    0
region_encoded    0
dtype: int64
```

- no null values of missing values are exist

Stastical Overview

- obtain stastical measure of the data
- describe() gives stastical measure of each column

In [12]:
```
df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
age	1338.0	39.207025	14.049960	18.0000	27.00000	39.000	51.000000	64.00000
bmi	1338.0	30.663397	6.098187	15.9600	26.29625	30.400	34.693750	53.13000
children	1338.0	1.094918	1.205493	0.0000	0.00000	1.000	2.000000	5.00000
charges	1338.0	13270.422265	12110.011237	1121.8739	4740.28715	9382.033	16639.912515	63770.42801
sex_encoded	1338.0	0.505232	0.500160	0.0000	0.00000	1.000	1.000000	1.00000
smoker_encoded	1338.0	0.204783	0.403694	0.0000	0.00000	0.000	0.000000	1.00000
region_encoded	1338.0	1.515695	1.104885	0.0000	1.00000	2.000	2.000000	3.00000

Duplicated Data

- check for dublicated values and remove it
- duplicated().sum() check for duplicated values

In [13]:
```
df.duplicated().sum()
```

Out[13]:
```
1
```

- remove duplicated

In [14]:
```
df.drop_duplicates(inplace=True)
```

In [15]:
```
df.duplicated().sum()
```

Out[15]:
```
0
```

Exploring Diversity:

- see how many unique values in the dataset
- nunique() return number of unique values

In [16]:
```
print(f"age:{df['age'].nunique()}")
print(f"sex:{df['sex'].nunique()}")
print(f"bmi:{df['bmi'].nunique()}")
print(f"children:{df['children'].nunique()}")
print(f"smoker:{df['smoker'].nunique()}")
print(f"region:{df['region'].nunique()}")
print(f"charges:{df['charges'].nunique()}")
```

```
age:47
sex:2
bmi:548
children:6
smoker:2
region:4
charges:1337
```

Correlation Analysis:

- check the Correlation between features and target Label
- corr() calculate the Correlation matrix

In [17]:
```
num_cols=df.select_dtypes('number')
corr_matrix=num_cols.corr()
corr_matrix
```

Out[17]:

	age	bmi	children	charges	sex_encoded	smoker_encoded	region_encoded
age	1.000000	0.109344	0.041536	0.298308	-0.019814	-0.025587	0.001626
bmi	0.109344	1.000000	0.012755	0.198401	0.046397	0.003746	0.157574
children	0.041536	0.012755	1.000000	0.067389	0.017848	0.007331	0.016258
charges	0.298308	0.198401	0.067389	1.000000	0.058044	0.787234	-0.006547
sex_encoded	-0.019814	0.046397	0.017848	0.058044	1.000000	0.076596	0.004936
smoker_encoded	-0.025587	0.003746	0.007331	0.787234	0.076596	1.000000	-0.002358
region_encoded	0.001626	0.157574	0.016258	-0.006547	0.004936	-0.002358	1.000000

- Sorting features by their correlation with the target variable (charges)

In [18]:
```
target_corr = corr_matrix['charges'].sort_values(ascending=False)
print("Correlation with target:\n", target_corr)
```

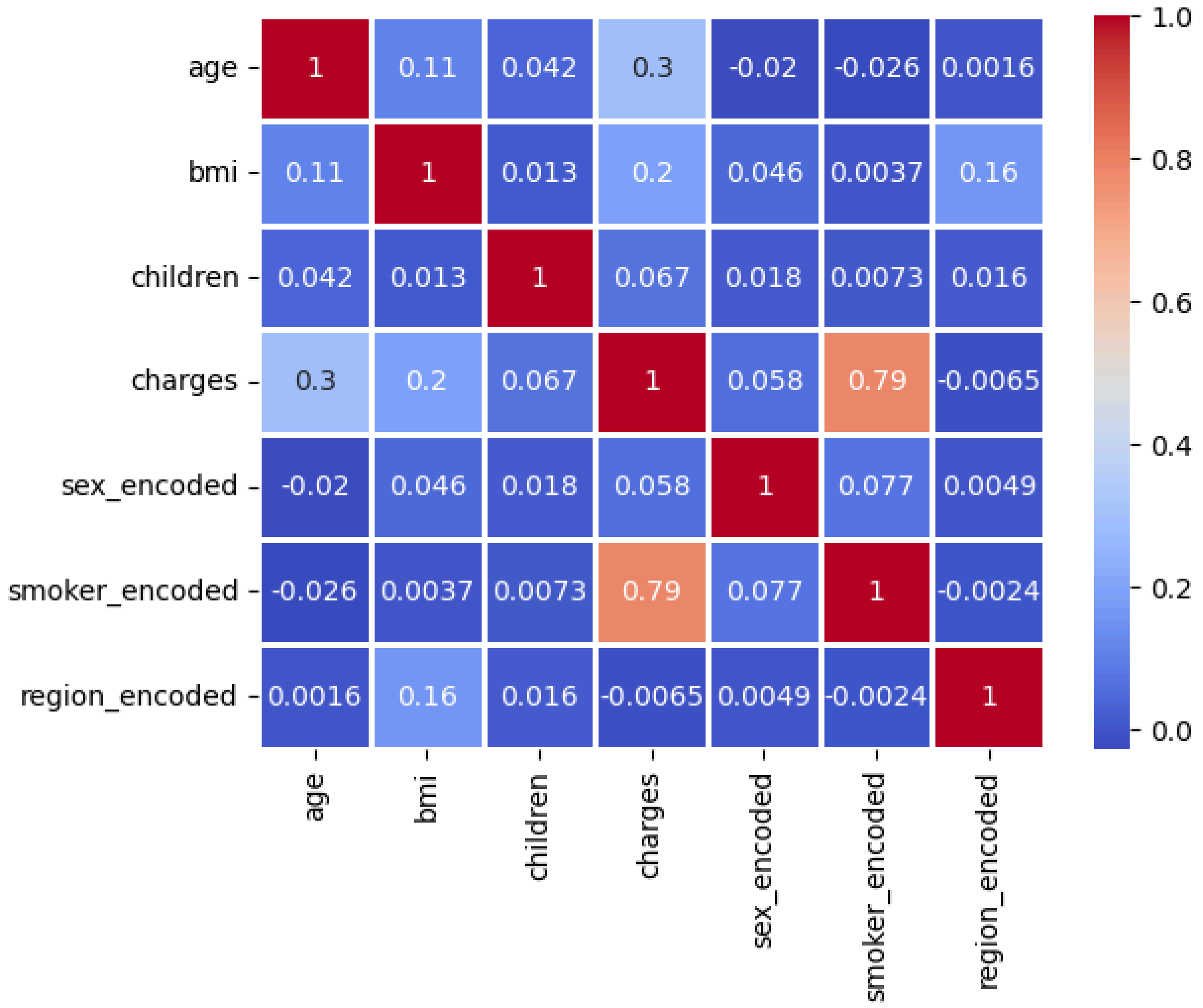
```
Correlation with target:
 charges           1.000000
smoker_encoded    0.787234
age               0.298308
bmi               0.198401
children          0.067389
sex_encoded       0.058044
region_encoded   -0.006547
Name: charges, dtype: float64
```

Heatmap:

- heatmap can show the corr matrix as visluzation

In [19]:
```
sns.heatmap(corr_matrix,annot=True,linewidths=2,cmap='coolwarm')
```

Out[19]:
```
<Axes: >
```



	age	bmi	children	charges	sex_encoded	smoker_encoded	region_encoded
age	1	0.11	0.042	0.3	-0.02	-0.026	0.0016
bmi	0.11	1	0.013	0.2	0.046	0.0037	0.16
children	0.042	0.013	1	0.067	0.018	0.0073	0.016
charges	0.3	0.2	0.067	1	0.058	0.79	-0.0065
sex_encoded	-0.02	0.046	0.018	0.058	1	0.077	0.0049
smoker_encoded	-0.026	0.0037	0.0073	0.79	0.077	1	-0.0024
region_encoded	0.0016	0.16	0.016	-0.0065	0.0049	-0.0024	1

2-Visulaization :

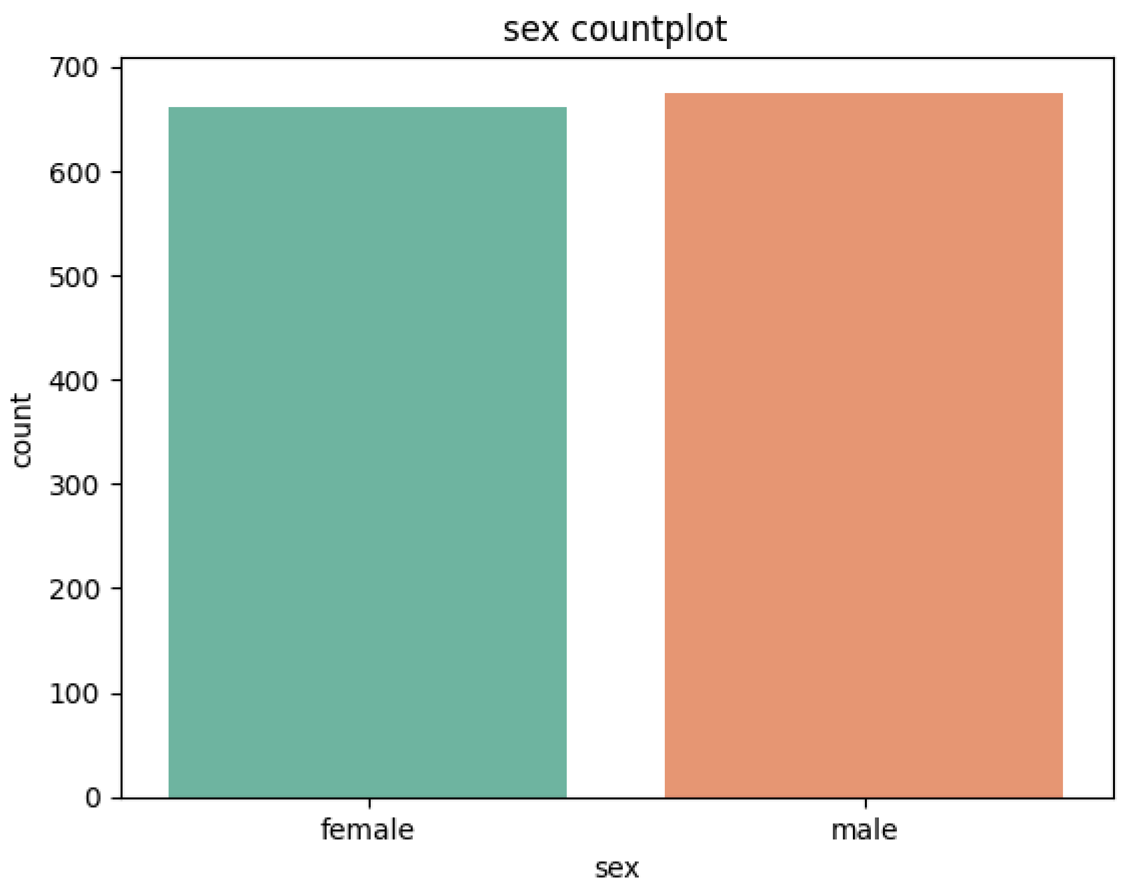
- Visualization allows us to quickly grasp complex data by presenting it in a visual format, making it easier to identify patterns, trends, and outliers that may not be apparent in raw data

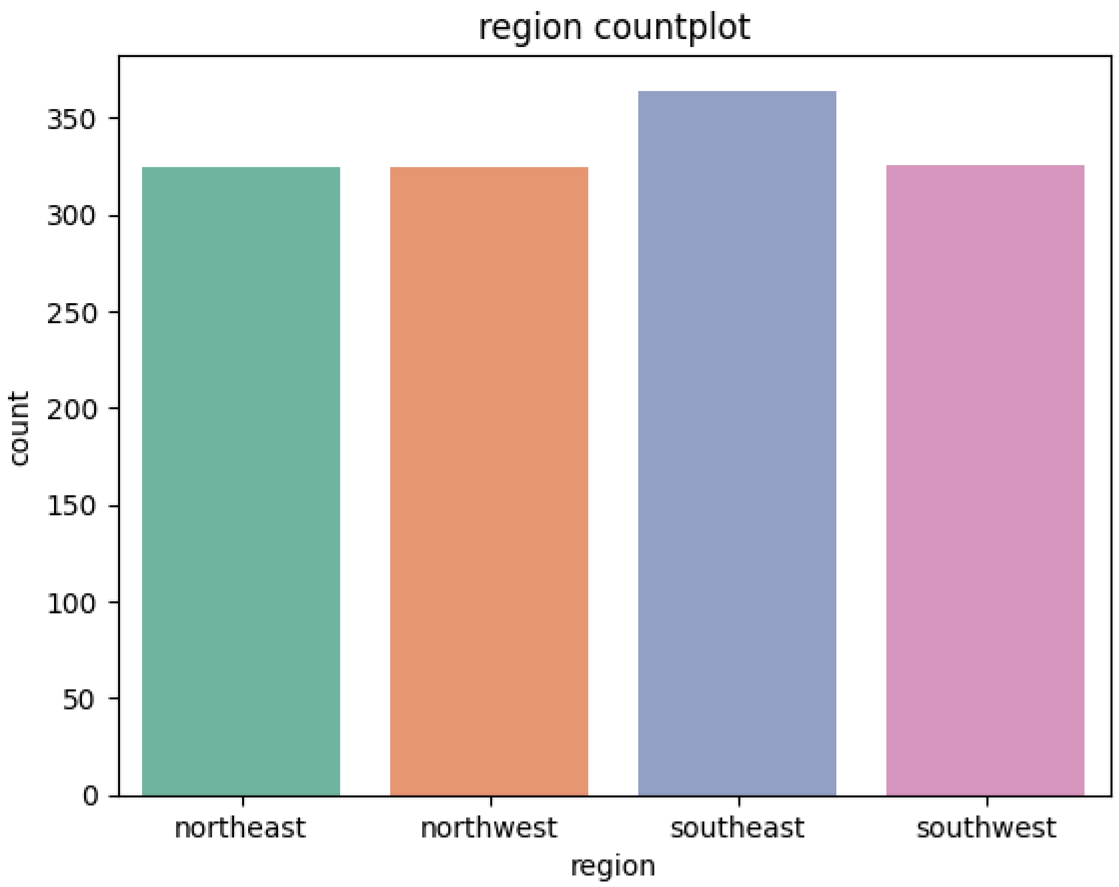
1-Categorical Data

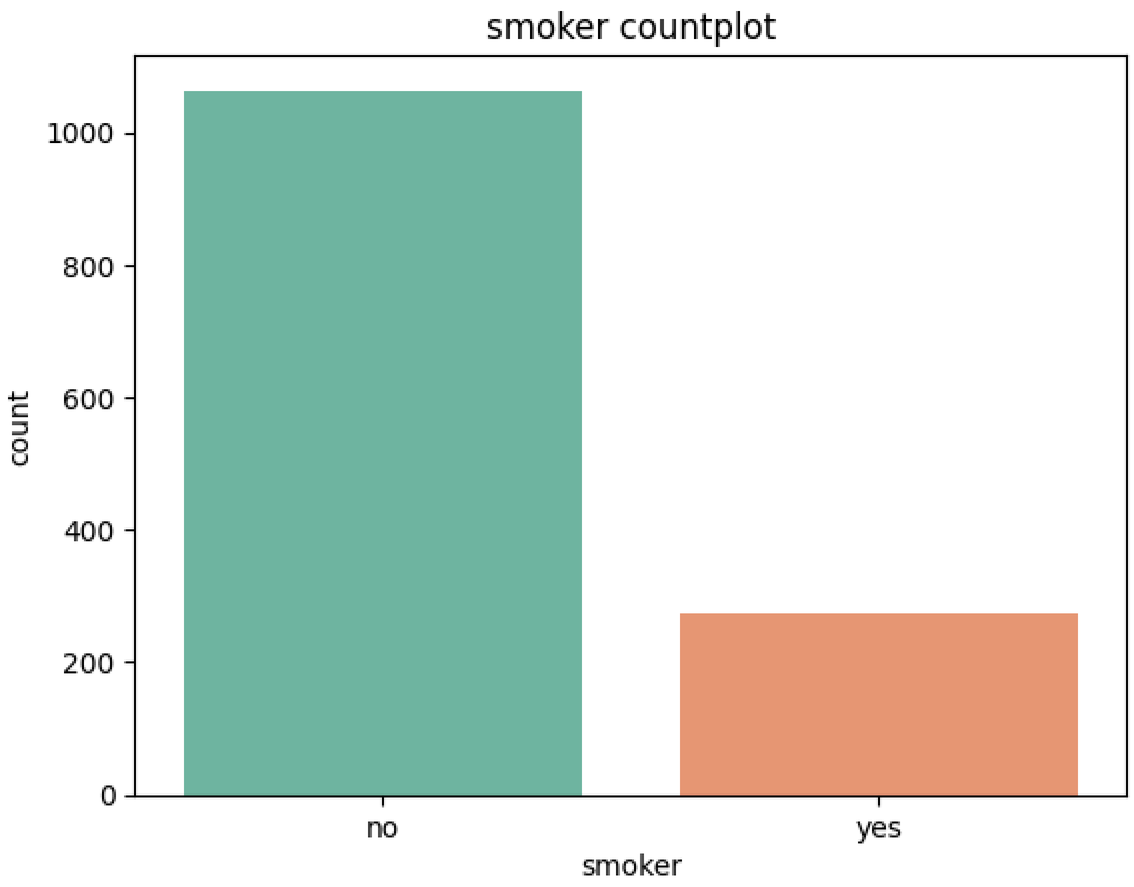
a. Countplot

- count occurrences of each category in categorical variable

In [20]:
```
sns.countplot(data=df,x='sex', palette='Set2')
plt.title('sex countplot')
plt.show()
sns.countplot(data=df,x='region', palette='Set2')
plt.title('region countplot')
plt.show()
sns.countplot(data=df,x='smoker', palette='Set2')
plt.title('smoker countplot')
plt.show()
```


sex countplot
count
700
600
500
400
300
200
100
0
female
male
sex


region countplot
350
300
250
200
150
100
50
0
count
northeast
northwest
southeast
southwest
region

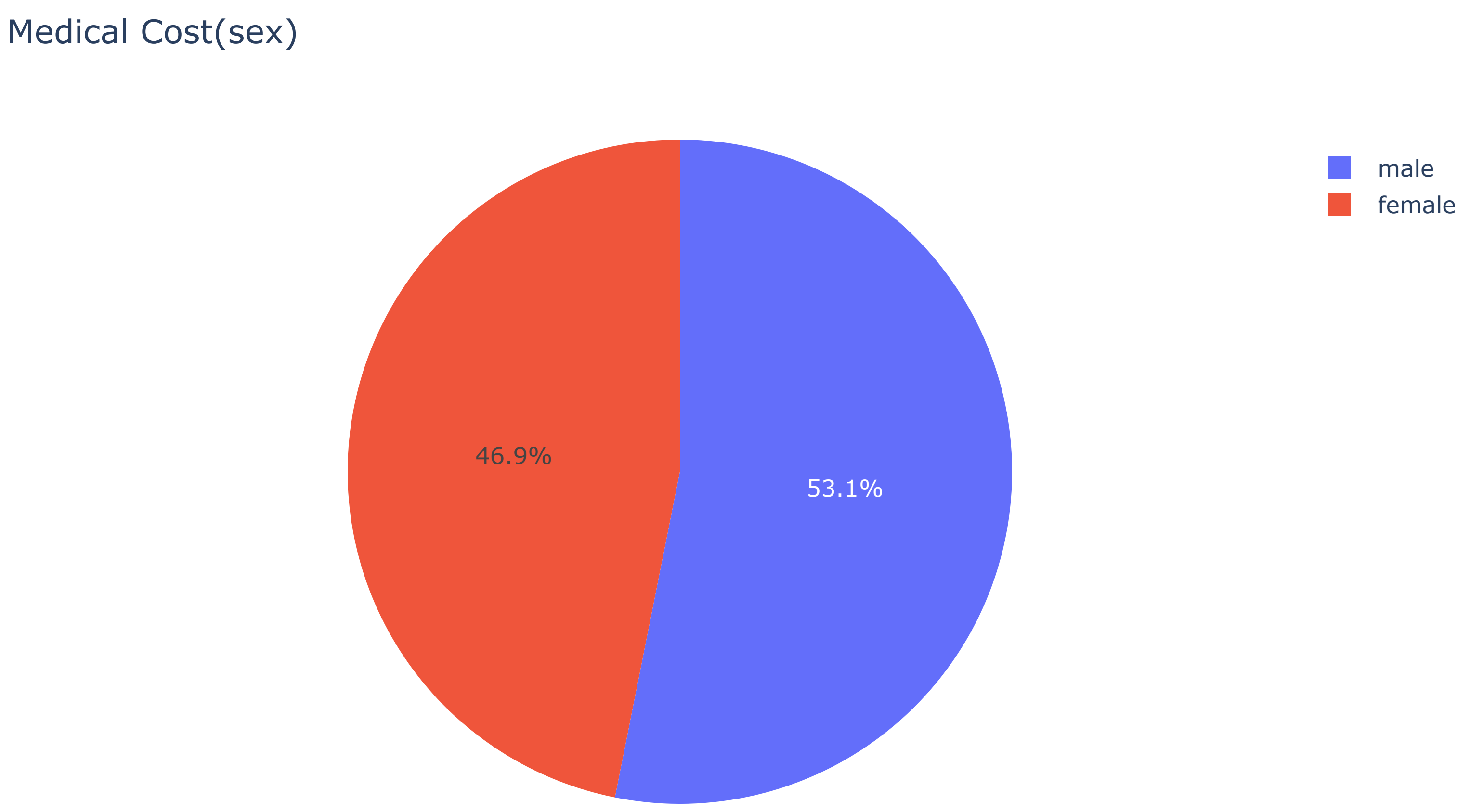


- **The number of smokers is fewer than non-smokers**
- **The number of males and females is nearly equal**

b.PieChart

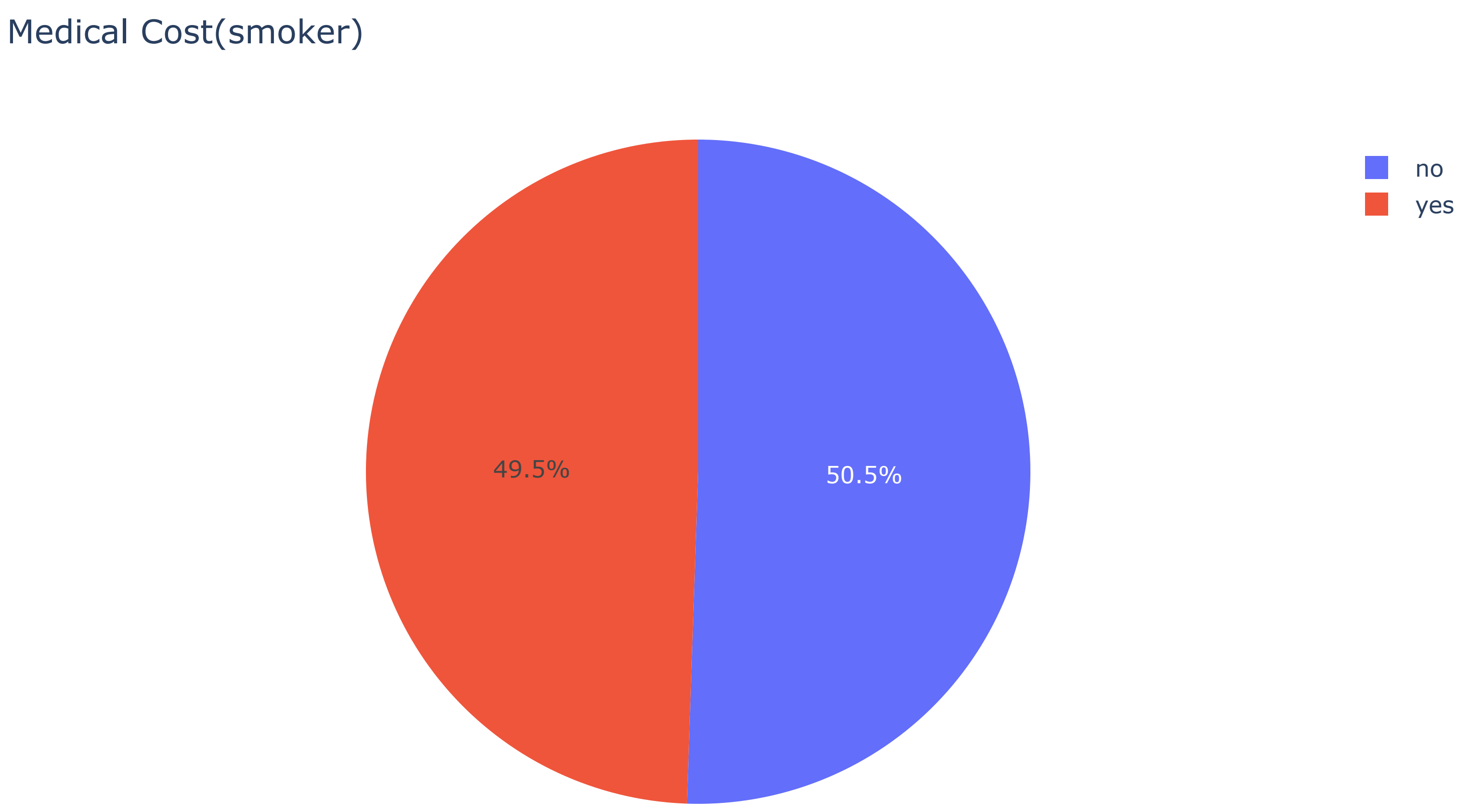
- Display the proportion of each category

```
In [21]: px.pie(df,values='charges', names = 'sex',title='Medical Cost(sex)')
```



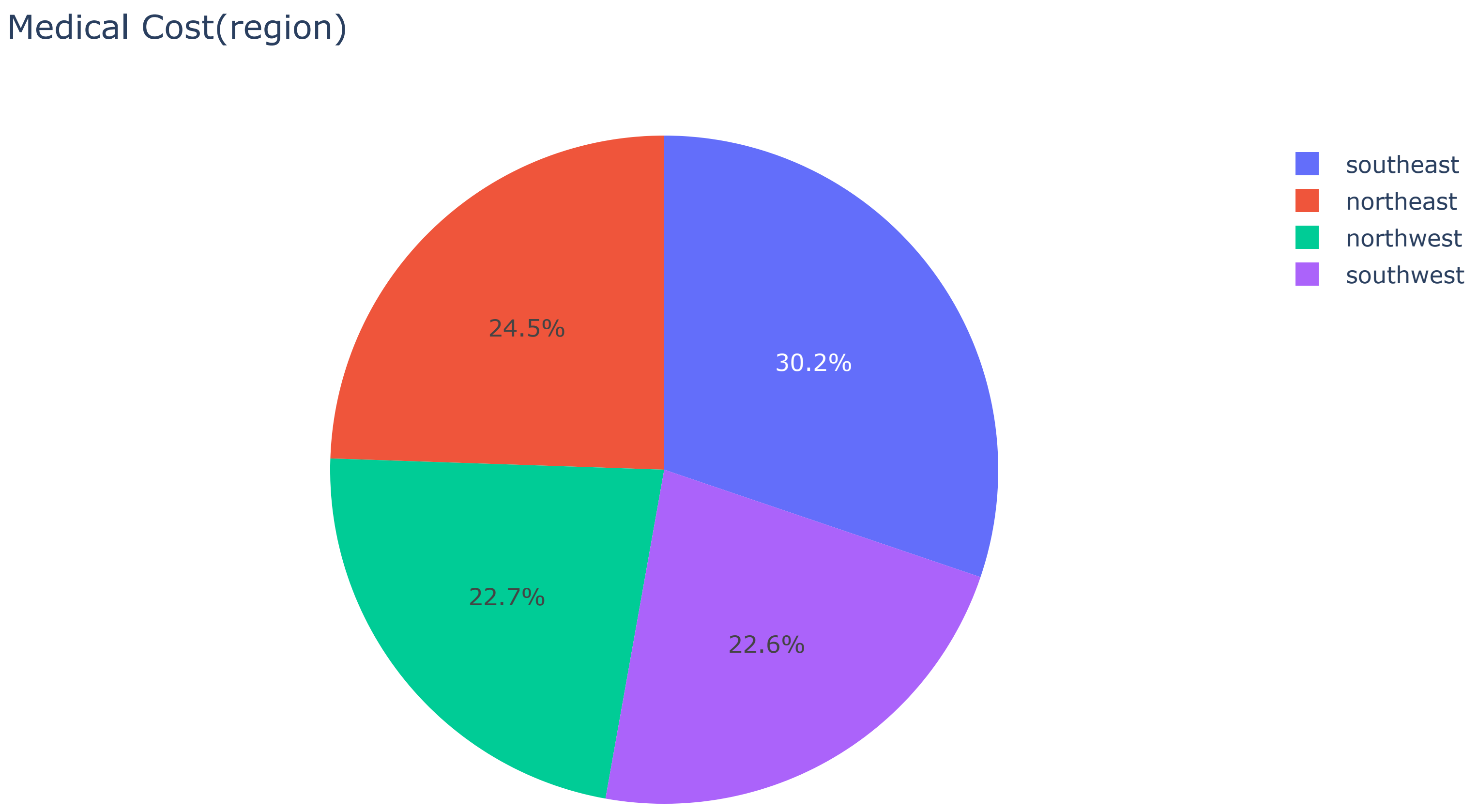
- **Males incur higher insurance costs than females**

```
In [22]: px.pie(df,values='charges', names = 'smoker',title='Medical Cost(smoker)')
```



- **non smoker incur higher insurance costs than smokers**

```
In [23]: px.pie(df,values='charges', names = 'region',title='Medical Cost(region)')
```

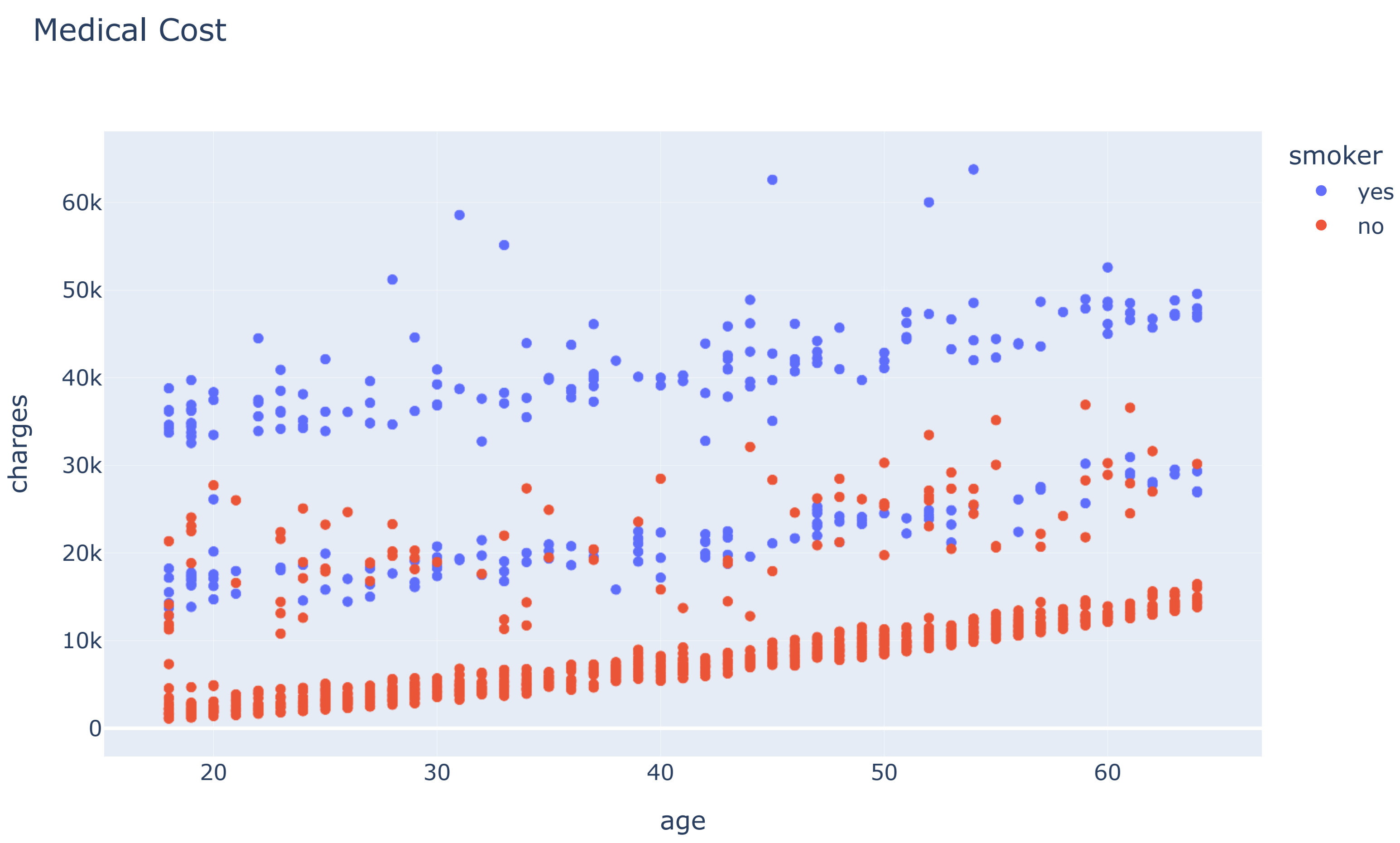


2-Numerical Data:

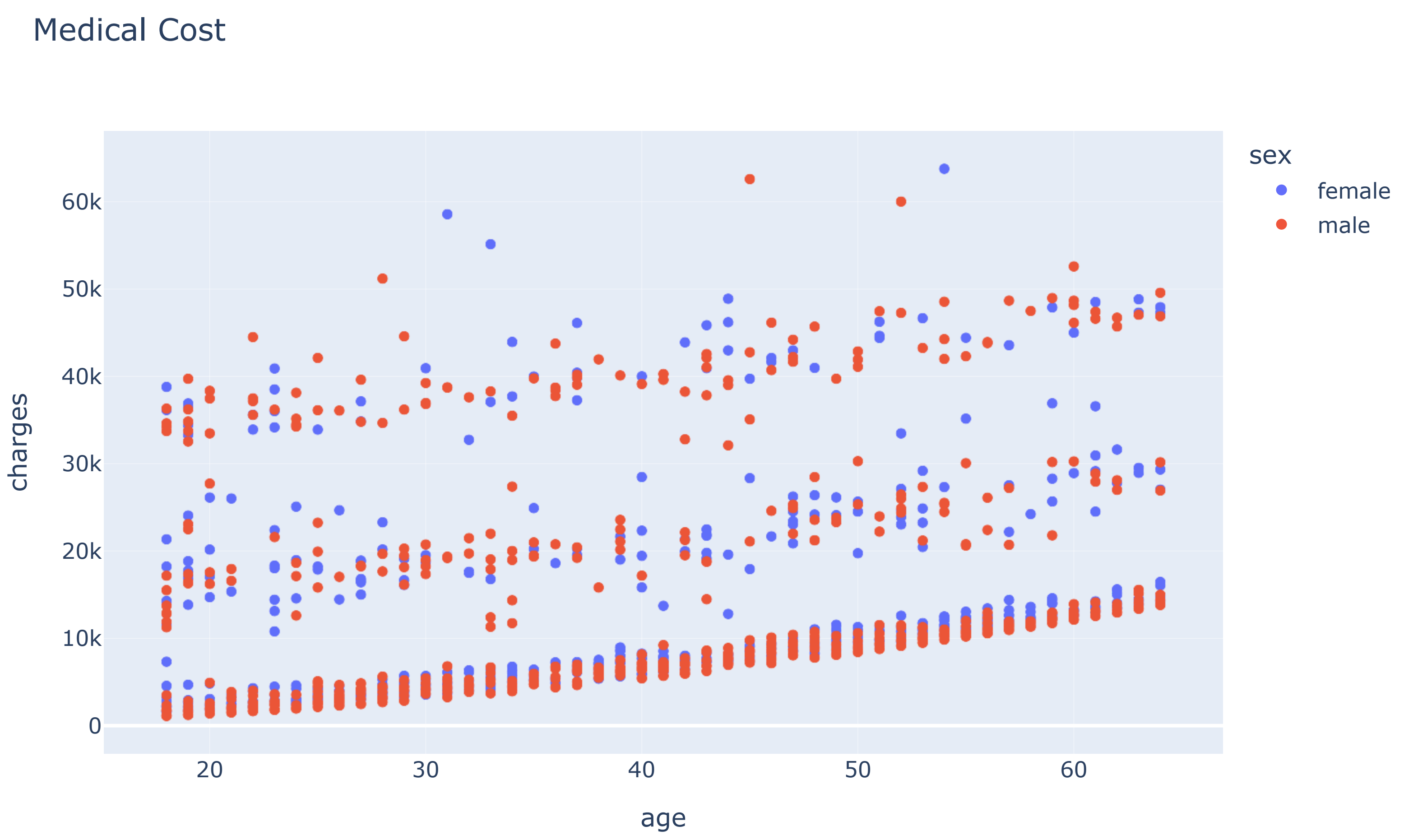
a-Scatter Diagram

- Visualize Relationships Between Features and Target Label:

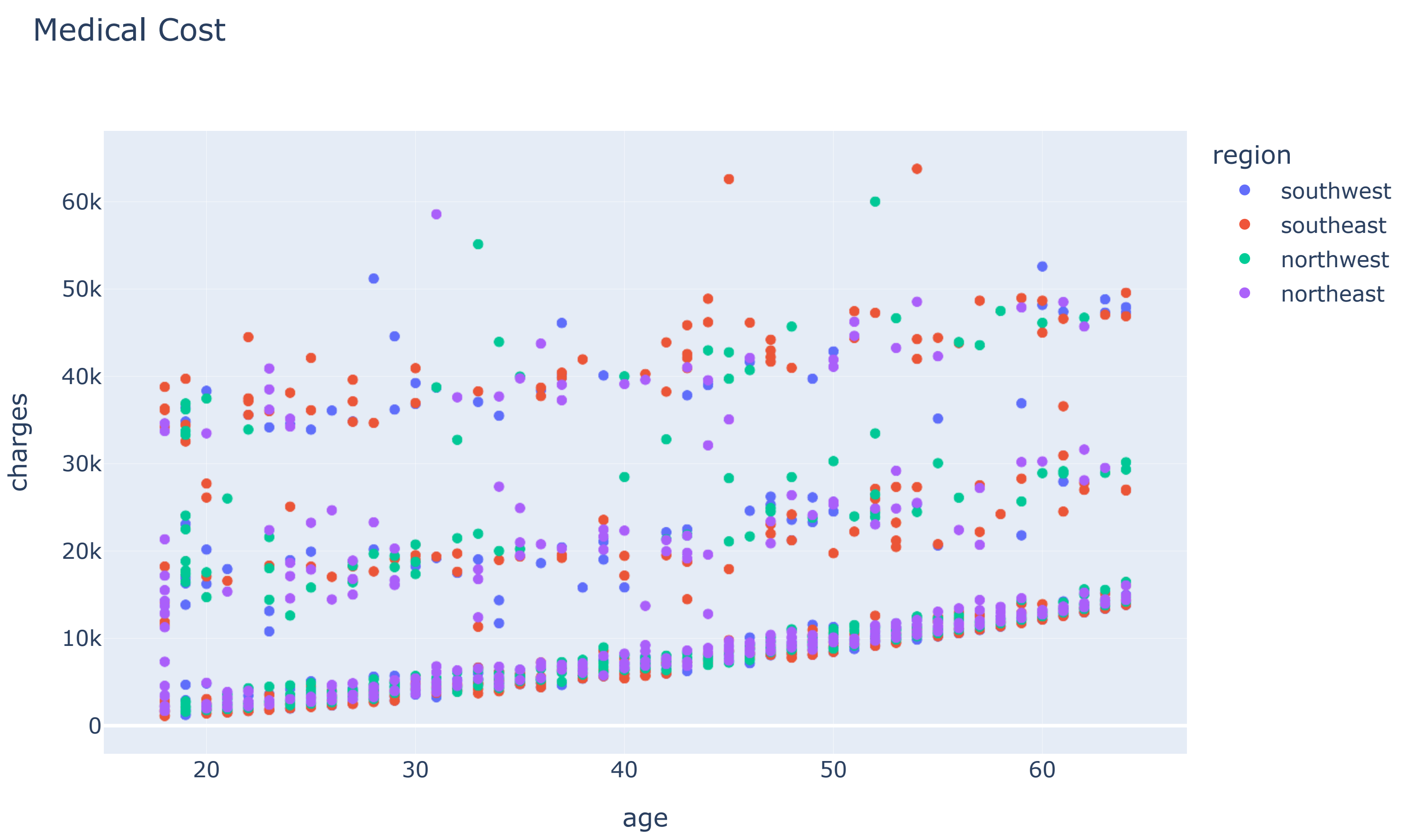
```
In [24]: px.scatter(data_frame=df,y='charges',x='age',hover_name='sex',color='smoker',title='Medical Cost')
```



```
In [25]: px.scatter(data_frame=df,y='charges',x='age',hover_name='sex',color='sex',title='Medical Cost')
```



```
In [26]: px.scatter(data_frame=df,y='charges',x='age',hover_name='bmi',color='region',title='Medical Cost')
```



Here are some key insights derived from the scatter plots:

- **1-Medical Cost vs. Age:** The scatter plot illustrated how medical charges vary with age. It showed a positive correlation, indicating that as age increases, the medical costs tend to rise as well. This suggests that older individuals may incur higher medical expenses.
- **2-Medical Cost vs. BMI:** Another scatter plot displayed the relationship between BMI (Body Mass Index) and medical charges. The visualization indicated a moderate positive correlation, suggesting that individuals with higher BMI may also have higher

medical costs.

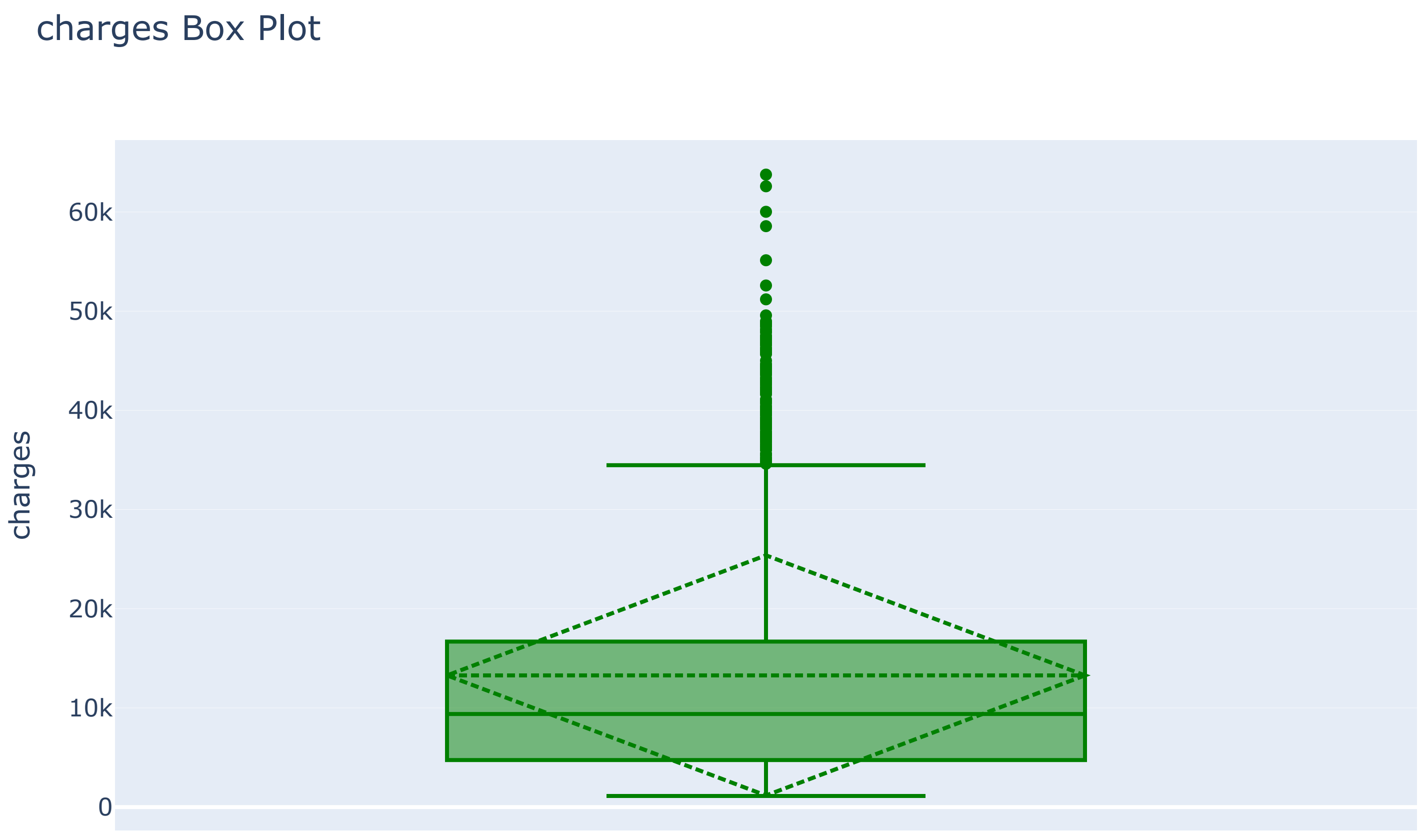
- **3-Medical Cost vs. Smoking Status:** The scatter plot differentiated between smokers and non-smokers, revealing a significant disparity in medical charges. Smokers generally had higher medical costs compared to non-smokers, highlighting the impact of smoking on healthcare expenses.

- **3-Medical Cost vs. Region:** By incorporating color coding for different regions, the scatter plot provided insights into how medical costs vary across geographical locations. It allowed for an easy comparison of charges among individuals from different regions, showcasing potential regional disparities in healthcare costs.

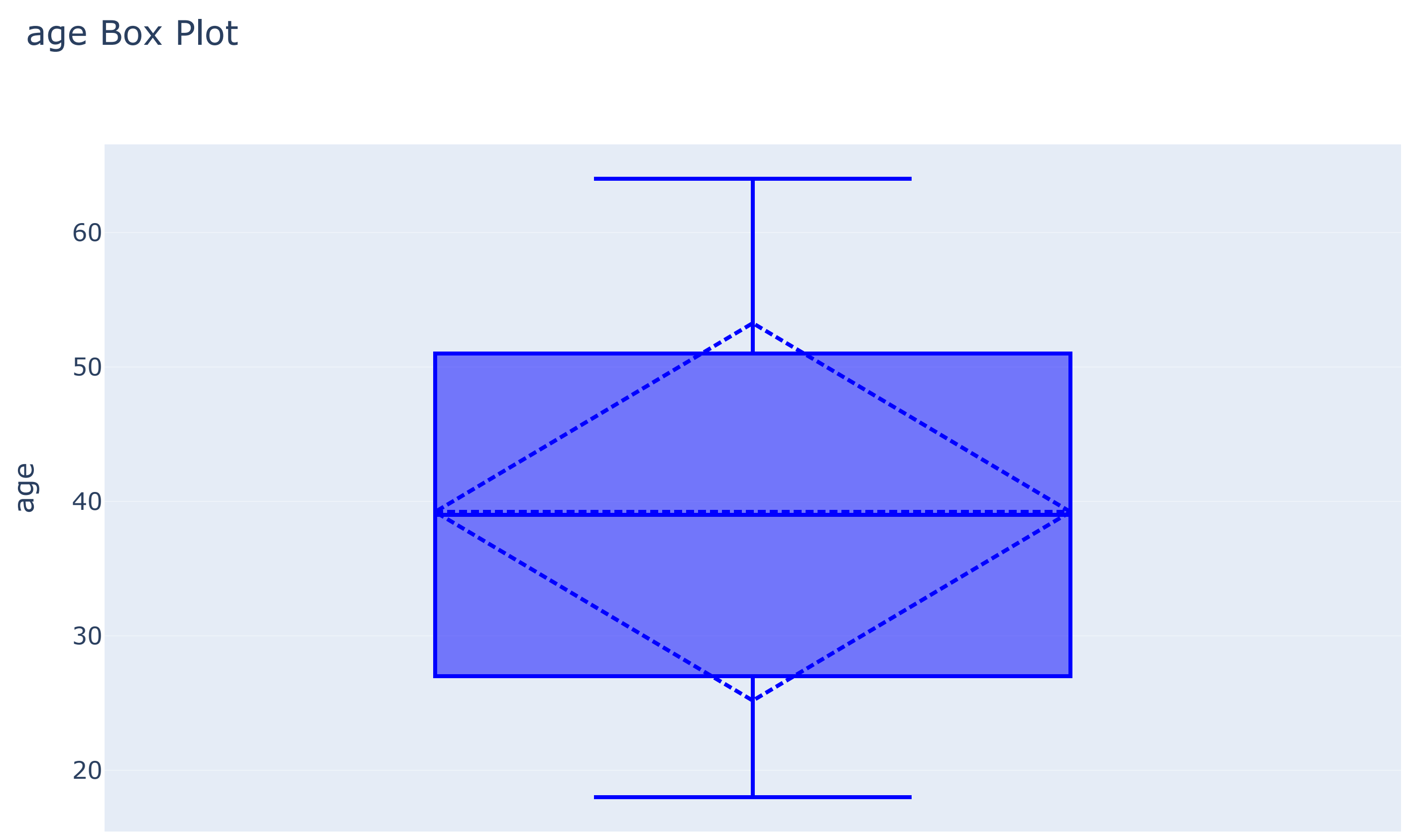
b-BoxPlot

- display five numbers summary (minumum,first quartile,median,third quartile and maximum)

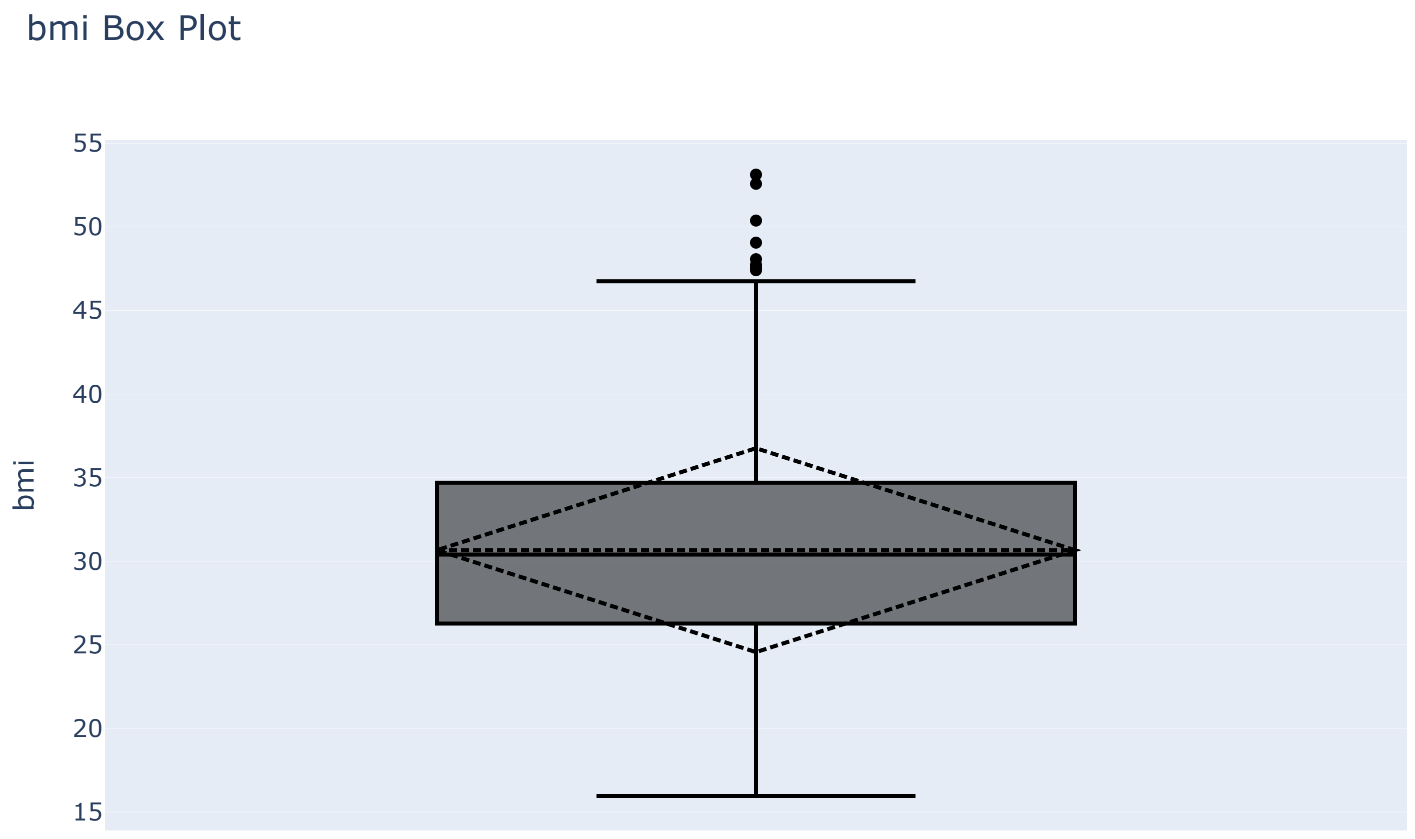
In [27]:
```
fig=px.box(df, y='charges', title='charges Box Plot ',orientation='v')
fig.update_traces(marker_color='green', boxmean='sd')
```



In [28]:
```
fig2=px.box(df, y='age', title='age Box Plot ',orientation='v')
fig2.update_traces(marker_color='blue', boxmean='sd')
```



```
In [29]: fig2=px.box(df, y='bmi', title='bmi Box Plot ',orientation='v')
         fig2.update_traces(marker_color='black', boxmean='sd')
```



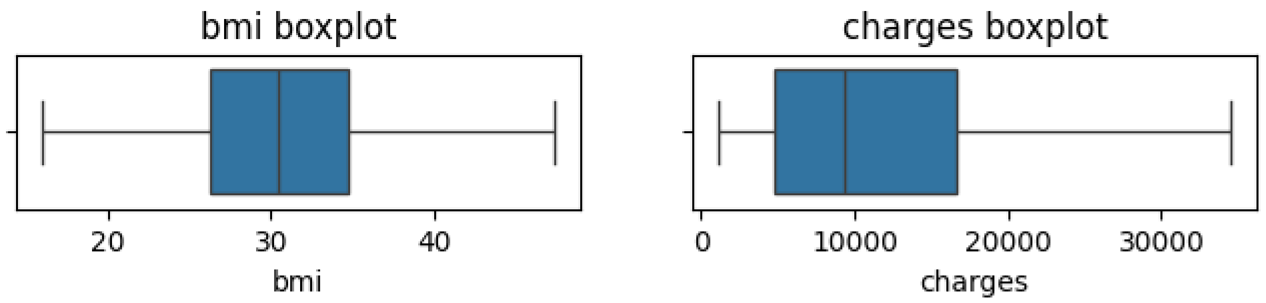
- **There's outliers in BMI and Charges columns**

Handle outliers

In [30]:
```
num_cols=['bmi','charges']
for col in num_cols:
    Q1 = df[col].quantile(.25)
    Q3 = df[col].quantile(.75)
    IQR = Q3 - Q1
    Lower_Fence = Q1 - 1.5 * IQR
    Upper_Fence = Q3 + 1.5 * IQR
    Lower_Outliers = df[df[col] < Lower_Fence][col].values
    Upper_Outliers = df[df[col] > Upper_Fence][col].values
    df[col].replace(Lower_Outliers, Lower_Fence, inplace=True)
    df[col].replace(Upper_Outliers, Upper_Fence, inplace=True)
```

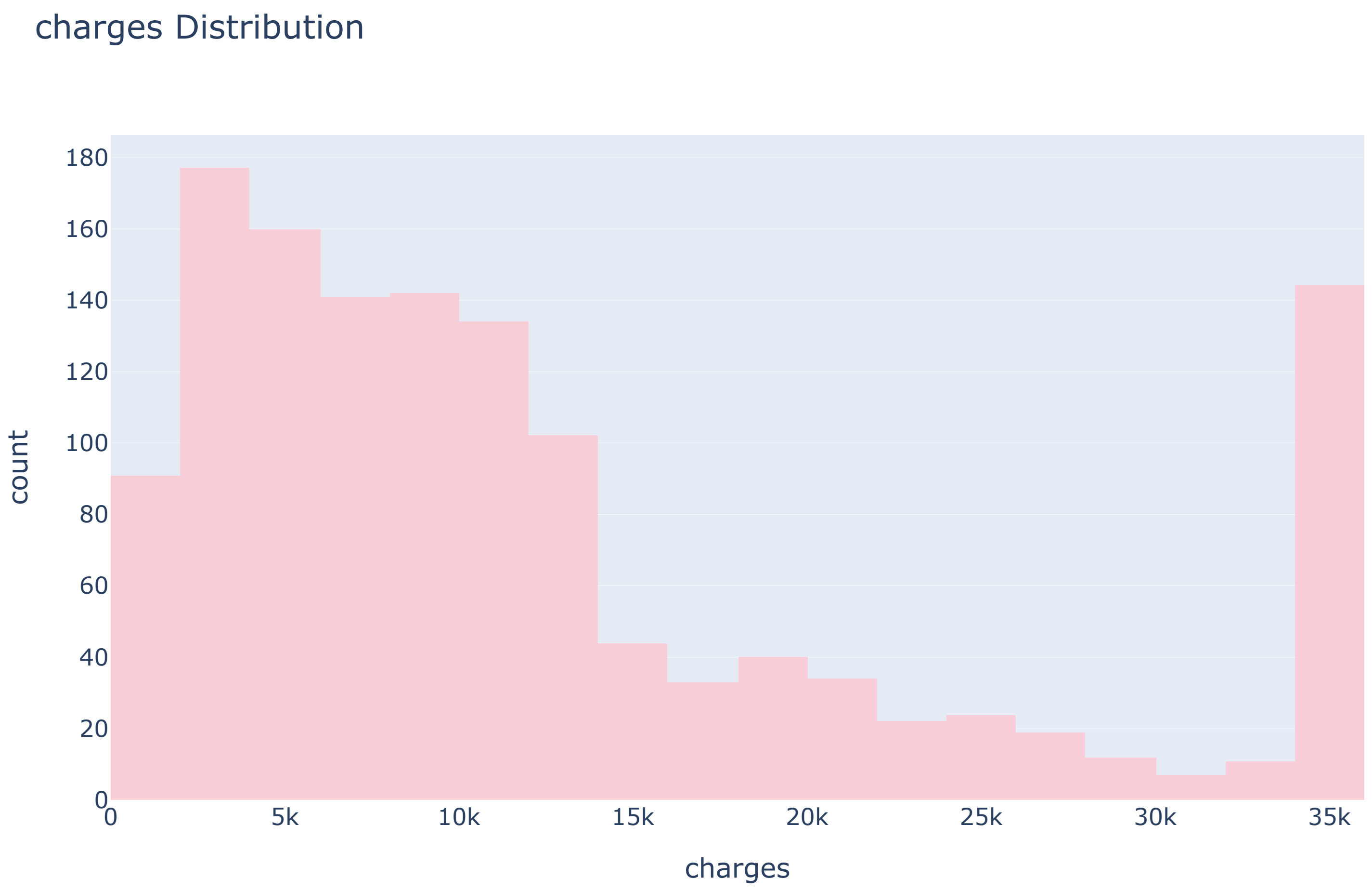
- **check outliers has been removed:**

In [31]:
```
plt.figure(figsize=(8, 1))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    sns.boxplot(df[col], orient="h")
    plt.title(f"{col} boxplot")
```

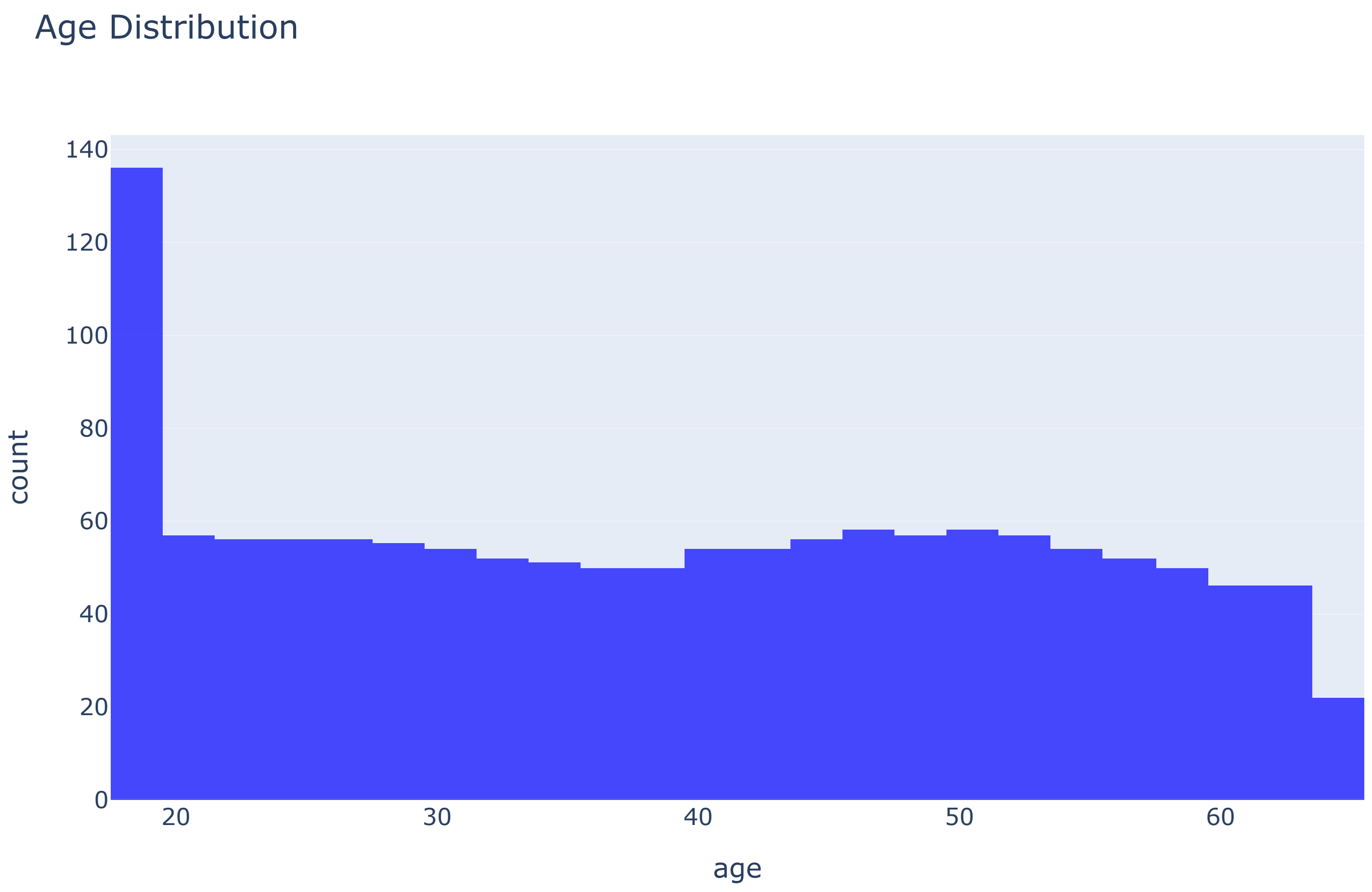


c-Histogram

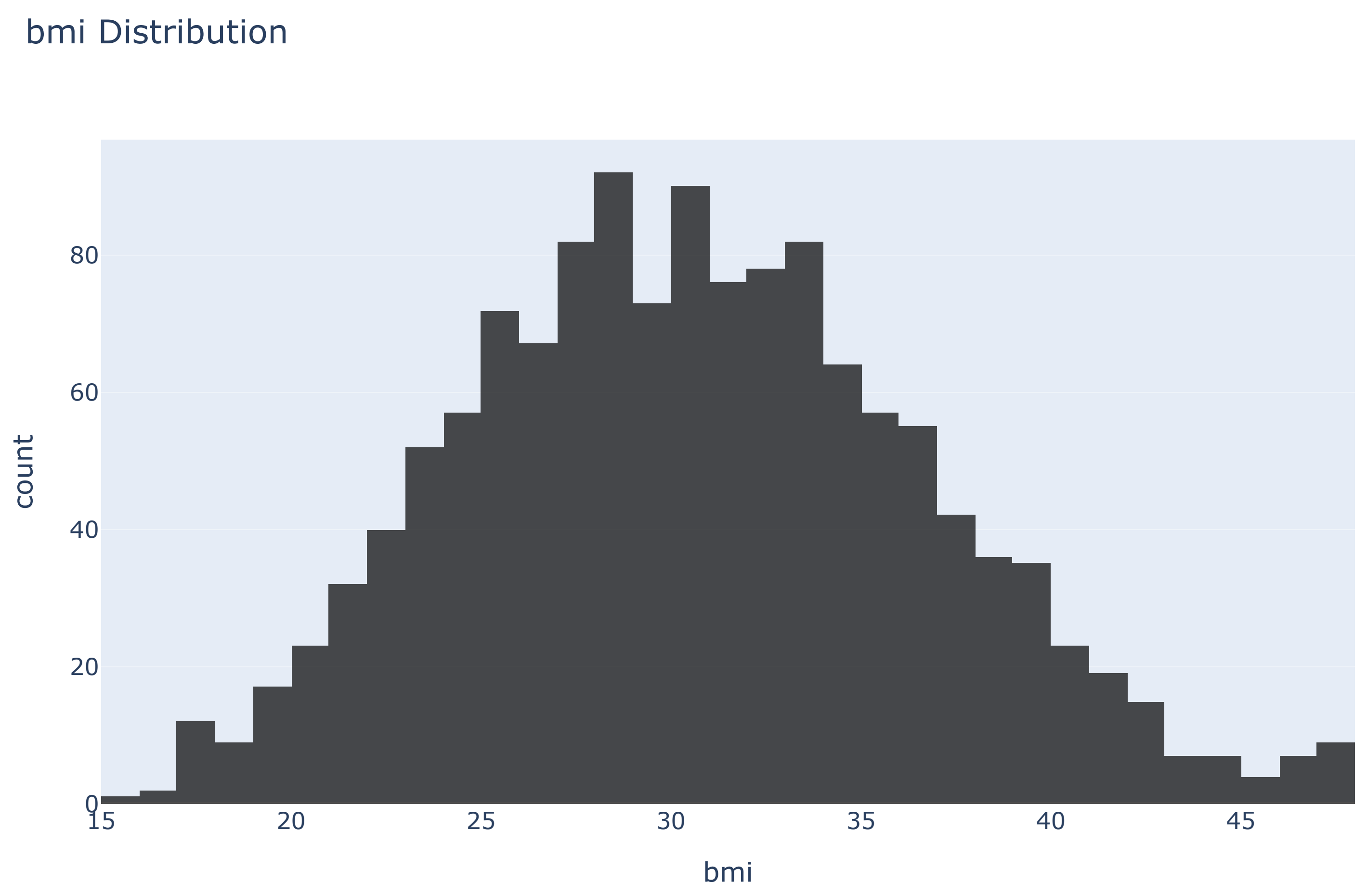
In [32]:
```
hist_fig= px.histogram(df, x='charges', title='charges Distribution')
hist_fig.update_traces(marker_color='pink', opacity=0.7)
```



In [33]:
```
hist_fig2= px.histogram(df, x='age', title='Age Distribution')
hist_fig2.update_traces(marker_color='blue', opacity=0.7)
```



In [34]:
```
hist_fig3= px.histogram(df, x='bmi', title='bmi Distribution')
hist_fig3.update_traces(marker_color='black', opacity=0.7)
```

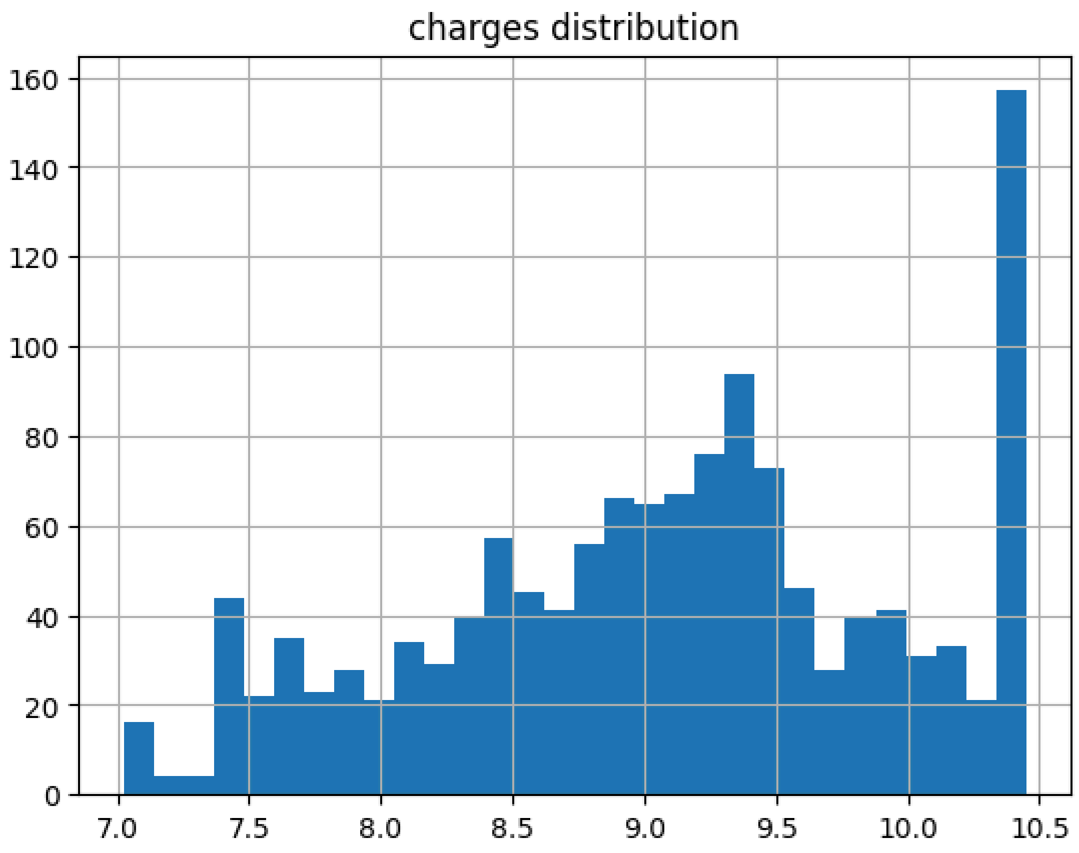


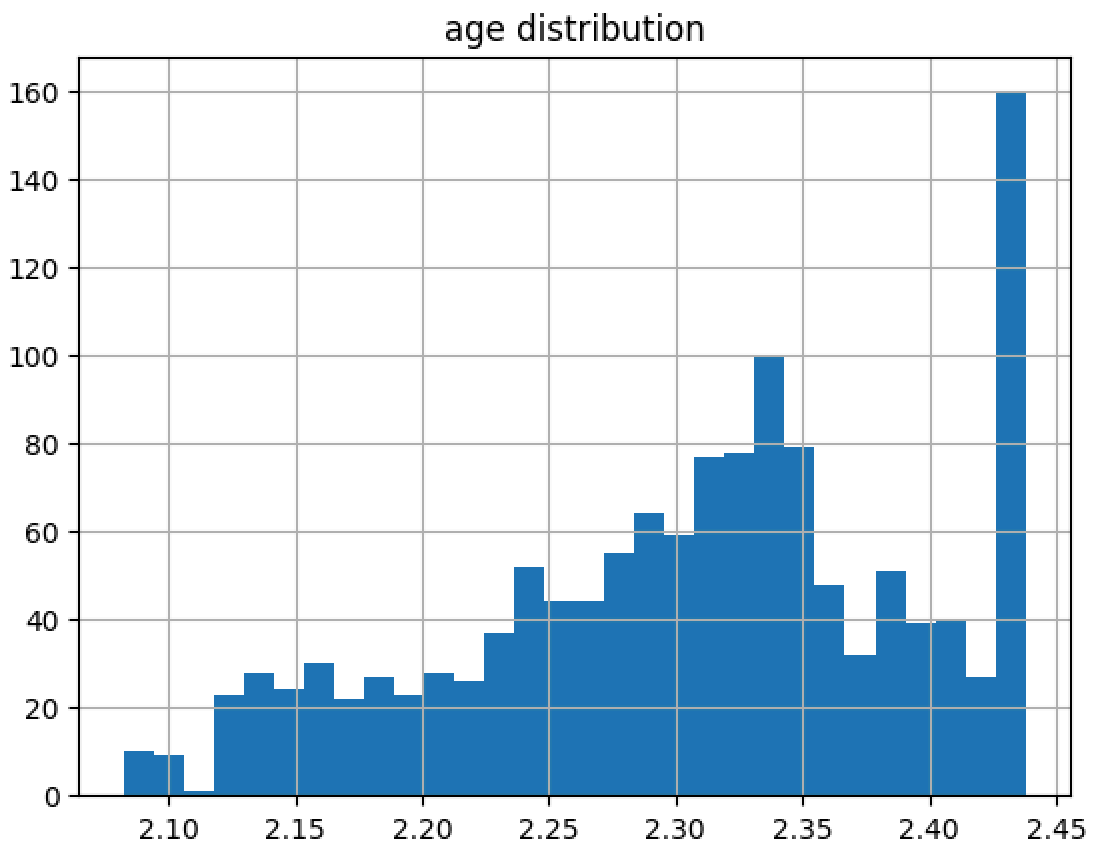
- **The previous histogram shows that BMI is normally distributed, while age and charges are right-skewed. To improve the distribution of charges and age and avoid errors in the model, we will apply a logarithmic transformation**

In [35]:
```
#  Logarithmic Transformation
df['charges'] = np.log(df['charges'] + 1) # Adding 1 to avoid log(0) issue
df['age'] = np.log(df['charges'] + 1)  # Adding 1 to avoid log(0) issue

# Checking the distribution
```

```
df['charges'].hist(bins=30)
plt.title('charges distribution')
plt.show()
df['age'].hist(bins=30)
plt.title('age distribution')
plt.show()
```





3-Feature Engineering

- We will drop columns with low correlation to the target label (charges)

In [36]:
```
target_corr = corr_matrix['charges'].sort_values(ascending=False)
print("Correlation with target:\n", target_corr)
```

```
Correlation with target:
 charges           1.000000
smoker_encoded    0.787234
age               0.298308
bmi               0.198401
children          0.067389
sex_encoded       0.058044
region_encoded   -0.006547
Name: charges, dtype: float64
```

In [37]:
```
cols_to_drop=['children','sex_encoded','region_encoded','sex','region','smoker']
df.drop(cols_to_drop,axis=1,inplace=True)
```

In [38]:
```
df.head()
```

Out[38]:

	age	bmi	charges	smoker_encoded
0	2.373438	27.900	9.734236	1
1	2.134626	33.770	7.453882	0
2	2.240791	33.000	8.400763	0
3	2.397726	22.705	9.998137	0
4	2.225753	28.880	8.260455	0

4-Normalization

-Scaling Features to a Common Range using MinMax Scaler

In [39]:
```
from sklearn.preprocessing import MinMaxScaler
num_cols=df.select_dtypes('number').columns
scaler=MinMaxScaler()
scaler.fit_transform(df[num_cols])
```

Out[39]:
```
array([[0.81857818, 0.38080051, 0.7912246 , 1.        ],
       [0.14690692, 0.56801148, 0.12558622, 0.        ],
       [0.44550134, 0.54345399, 0.40198218, 0.        ],
       ...,
       [0.12786738, 0.66624143, 0.10893754, 0.        ],
       [0.19685809, 0.31382555, 0.16980437, 0.        ],
       [0.9580479 , 0.41811513, 0.9505188 , 1.        ]])
```

5- Split the Data:

- Split the dataset into training and testing sets:
- x: features , y: target label

In [40]:
```
y=df[['charges']]
x=df.drop('charges',axis=1)
```

In [41]:
```
x
```

Out[41]:

	age	bmi	smoker_encoded
0	2.373438	27.900	1
1	2.134626	33.770	0
2	2.240791	33.000	0
3	2.397726	22.705	0
4	2.225753	28.880	0
...	...	...	...
1333	2.329106	30.970	0
1334	2.163252	31.920	0
1335	2.127856	36.850	0
1336	2.152386	25.800	0
1337	2.423027	29.070	1

1337 rows × 3 columns

In [42]:

```
y
```

Out[42]:

	charges
0	9.734236
1	7.453882
2	8.400763
3	9.998137
4	8.260455
...	...
1333	9.268755
1334	7.699381
1335	7.396847
1336	7.605365
1337	10.279948

1337 rows × 1 columns

5-Machine Learning Model :

a-Split the data into train and test models

In [43]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

- check the shape of the train and test models:

In [44]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1069, 3)
(268, 3)
(1069, 1)
(268, 1)
```

b-Import Linear Regression Model

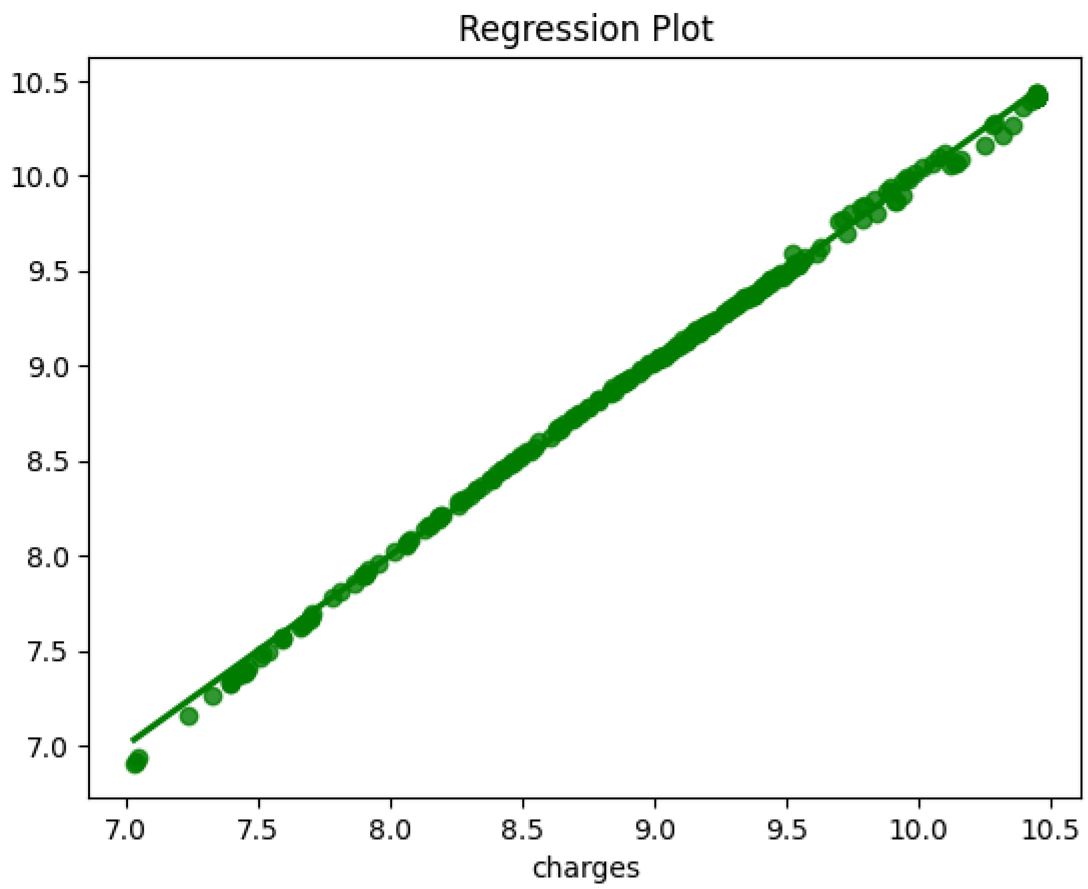
- importing the LR model from sklearn library
- train the data using fit()
- predict the output using predict()

In [45]:
```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train, y_train)
y_test_pred=LR.predict(x_test)
```

c-Testing model accuarcy

- Create a scatter plot to visualize how the linear regression model fits the data
- calculate :
- 1-mean_absolute_error
- 2-mean_squared_error
- 3-Score Matrix

In [59]:
```
sns.regplot(x=y_test,y=y_test_pred,ci=None,color='green')
plt.title('Regression Plot')
plt.show()
```



In [57]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_test_pred)
print(f"Mean Absolute Error: {mae:.2f}")

mse = mean_squared_error(y_test, y_test_pred)
print(f"Mean Squared Error: {mse:.2f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error: {rmse:.2f}")

r2 = r2_score(y_test, y_test_pred)
print(f"R-squared: {r2:.2f}")
```

```
Mean Absolute Error: 0.03
Mean Squared Error: 0.00
Root Mean Squared Error: 0.03
R-squared: 1.00
```

- **Model has accuarcy 100%**
- **accuarcy has been improved form 83% to 100%**