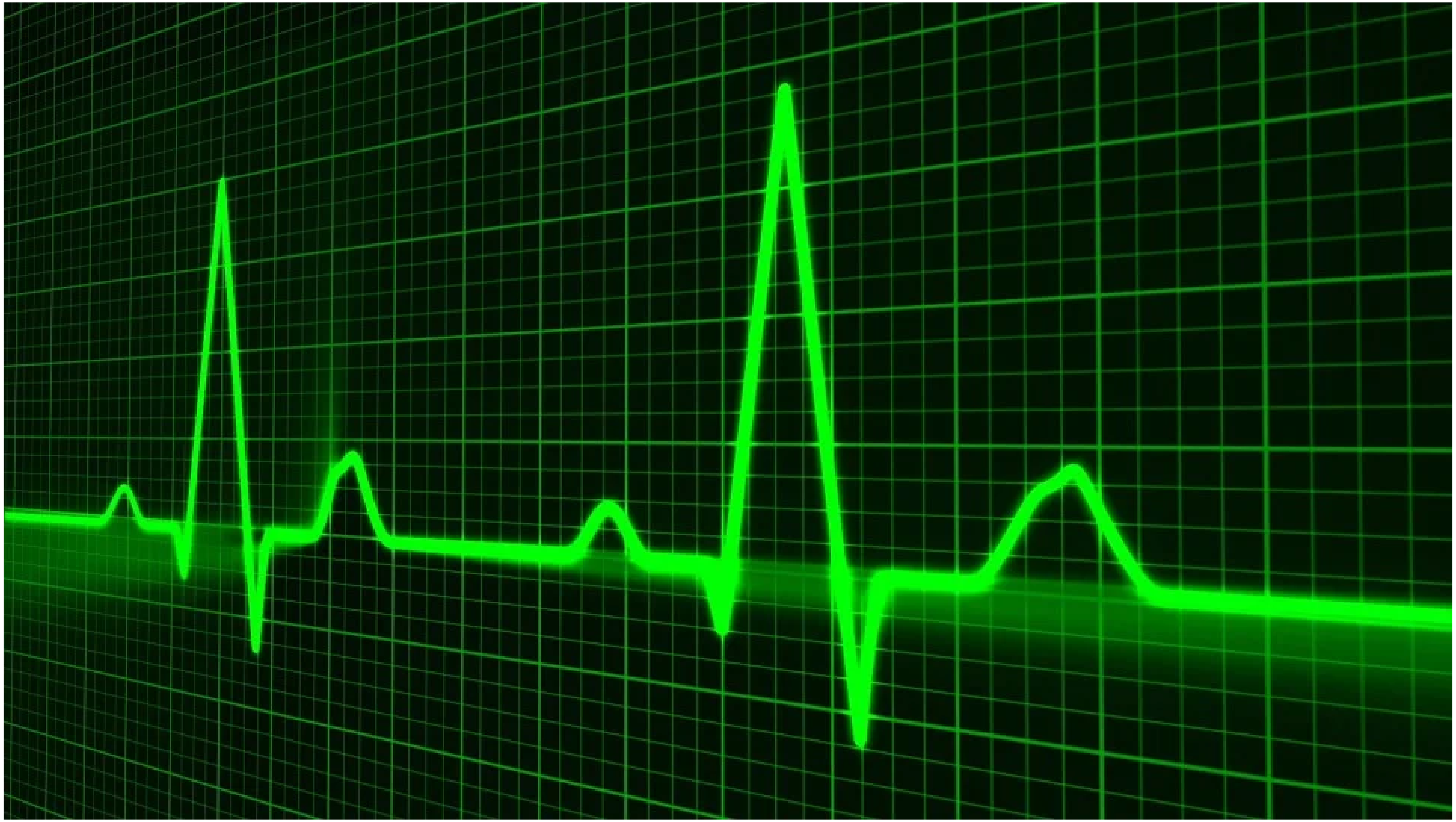
Insurance Price Prediction with Linear Regression

In this notebook, I'm going to show how to make insurance price prediction with Linear Regression for new customers, using information such as their age, sex, BMI, children, smoking habits and region of residence.. After doing the project, will enter the patient's information such as age, sex, region into the model and then predict the patient's charge.



Data Description

The data set includes the following variables:

Age: Age of the Customer
Sex: Gender of the Customer.
bmi: A person's weight in kilograms divided by the square of height in meters.
children: The number of children of the customer.
smoker: If the insured person is asmoker or not.
region: Region where the customer lived.
charges: The premium of insurance.
Data Source: Open Source

Loading The Data

In [1]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_csv("file:///C:/Users/angsh/OneDrive/Desktop/PRAXIS/Own%20Projects/ML/Regression%20project/insurance.c
```

In [3]:
```
data.head(4)
```

Out[3]:

	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

Understanding The Data

In [4]:
```
data.shape
```

Out[4]:
```
(1338, 7)
```

In [5]:
```
data.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
```

Dealing with Missing Data

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

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

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

Preprocessing The Data

Let's convert them to category type to use the model building later.

In [8]:
```
data["sex"] = data["sex"].astype("category")
data["smoker"] = data["smoker"].astype("category")
data["region"] = data["region"].astype("category")
```

In [9]:
```
data.dtypes
```

Out[9]:
```
age             int64
sex          category
bmi           float64
children        int64
smoker       category
region       category
charges       float64
dtype: object
```

Let's take a look at summary statistics.

In [10]:
```
data.describe().T
```

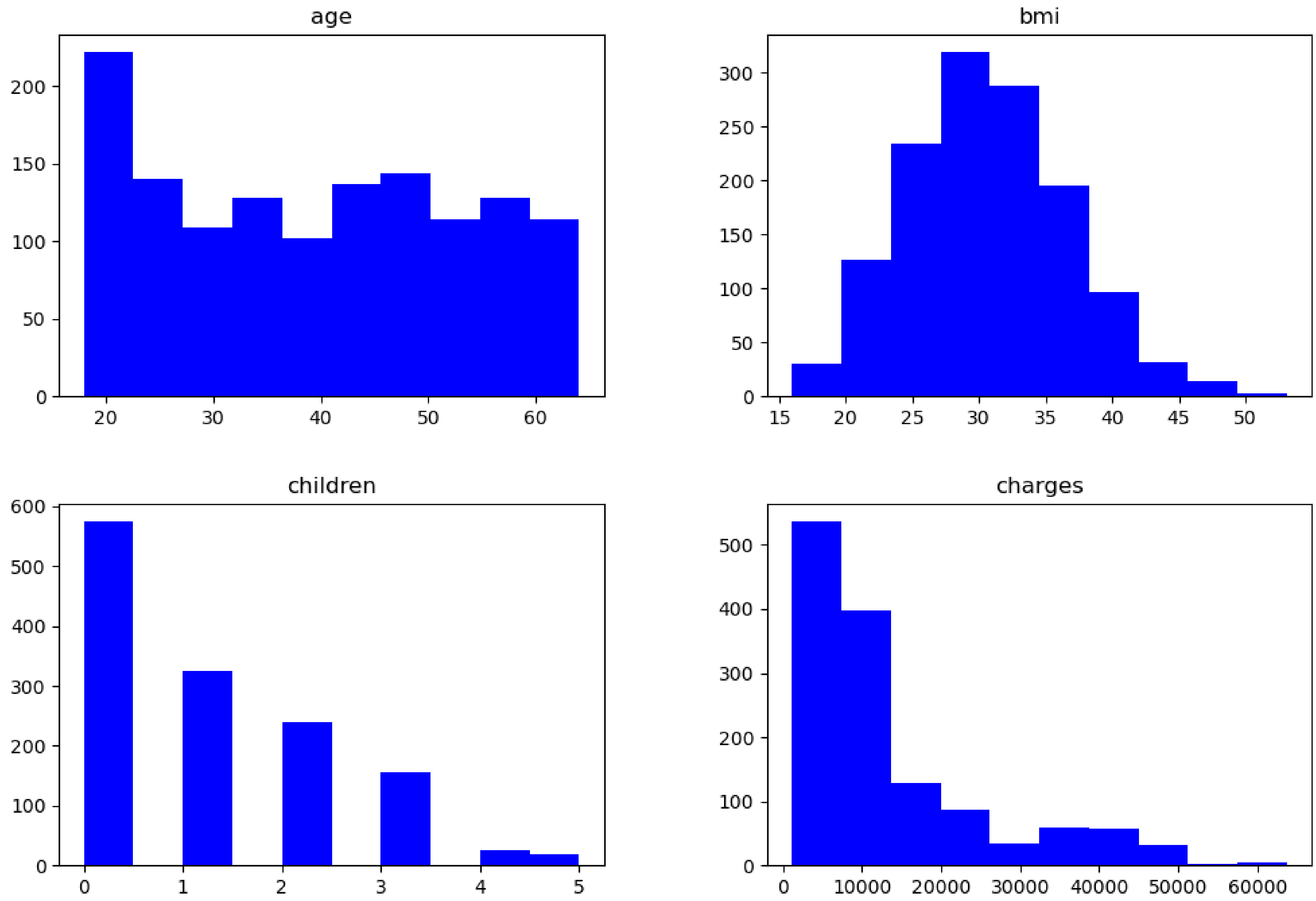
Out[10]:

	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

In [11]:
```
data.hist(
['age','bmi',
 'children',
 'charges'],figsize=(12,8),color='blue',grid=False)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14900\1812213538.py in <module>
      3  'children',
      4  'charges'],figsize=(12,8),color='blue',grid=False)
----> 5 plt.show()

NameError: name 'plt' is not defined
```


age
0
50
100
150
200
20
30
40
50
60
bmi
0
50
100
150
200
250
300
15
20
25
30
35
40
45
50
children
0
100
200
300
400
500
600
0
1
2
3
4
5
charges
0
100
200
300
400
500
0
10000
20000
30000
40000
50000
60000

In [12]:
```
smoke_data = data.groupby("smoker").mean().round(2)
```

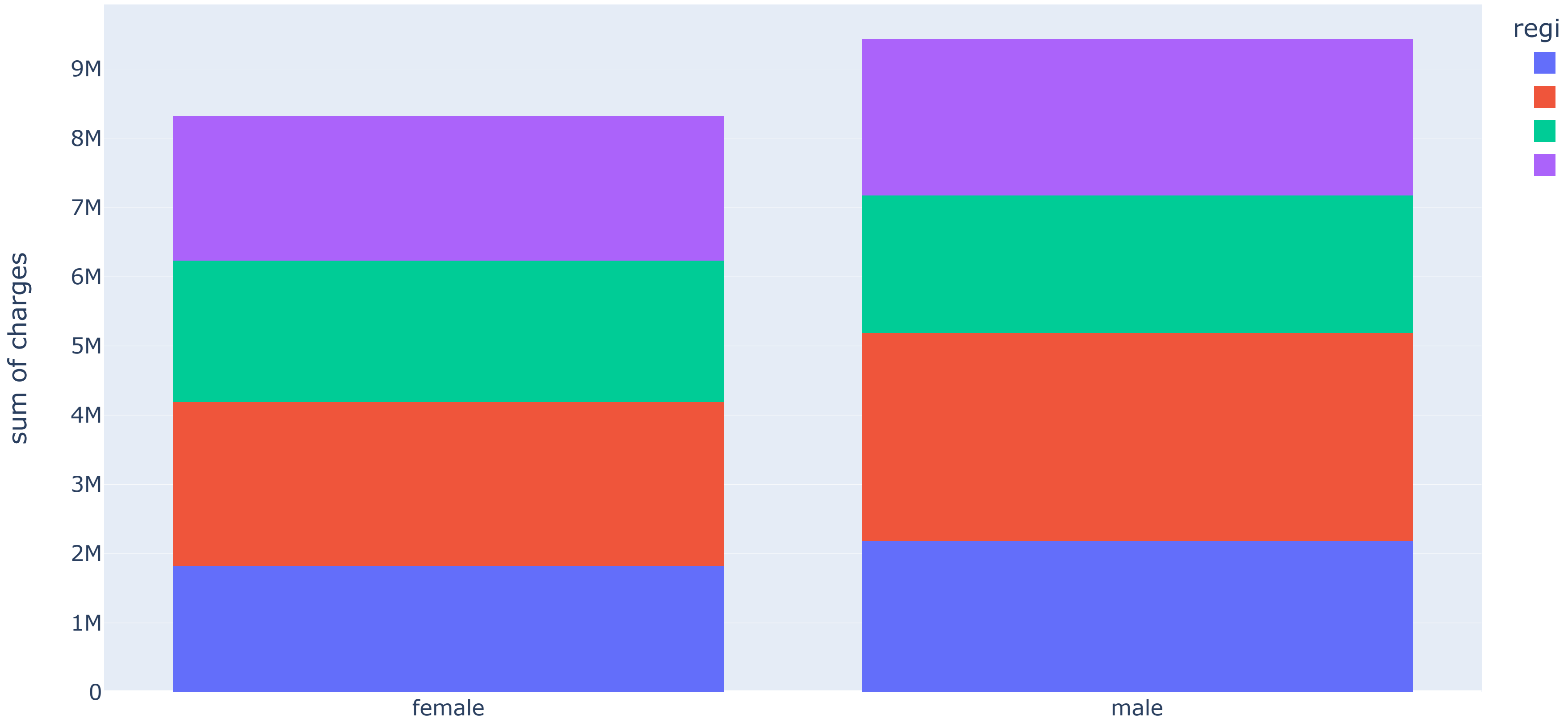
In [13]:
```
smoke_data
```

Out[13]:

	age	bmi	children	charges
smoker				
no	39.39	30.65	1.09	8434.27
yes	38.51	30.71	1.11	32050.23

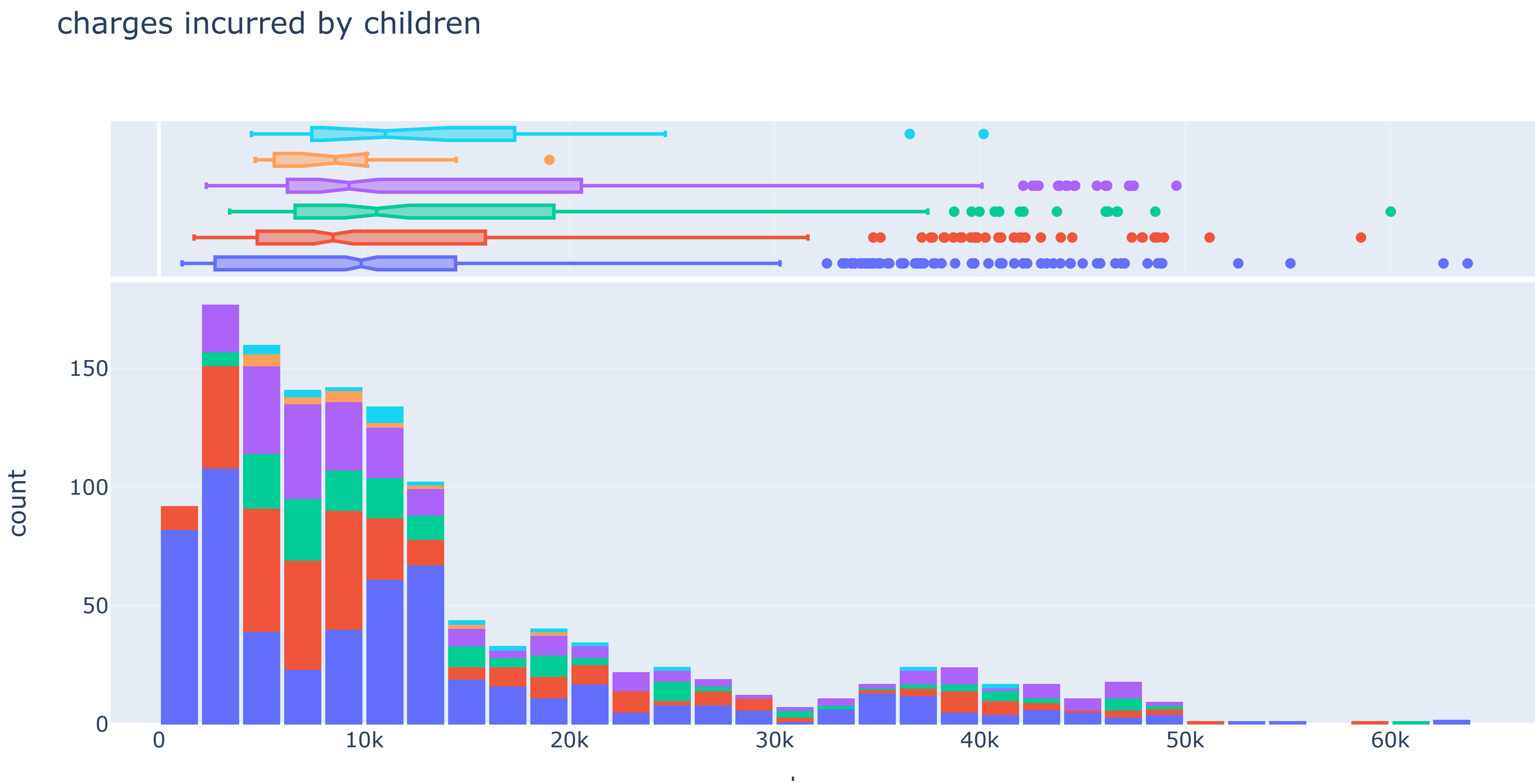
Preprocessing The Data

In [14]:
```
import plotly.express as px
px.histogram(data,x='sex',y = 'charges',color = 'region')
```



We can infer that from every region our customer base which has males are incurring more bills but interestingly females of northwest region are having more medical bills.

In [15]:
```
fig = px.histogram(data,
                  x = 'charges',
                   marginal ="box",
                  color = 'children',
                  title = 'charges incurred by children')
fig.update_layout(bargap=0.1)
fig.show()
```


charges incurred by children
count
150
100
50
0
0
10k
20k
30k
40k
50k
60k

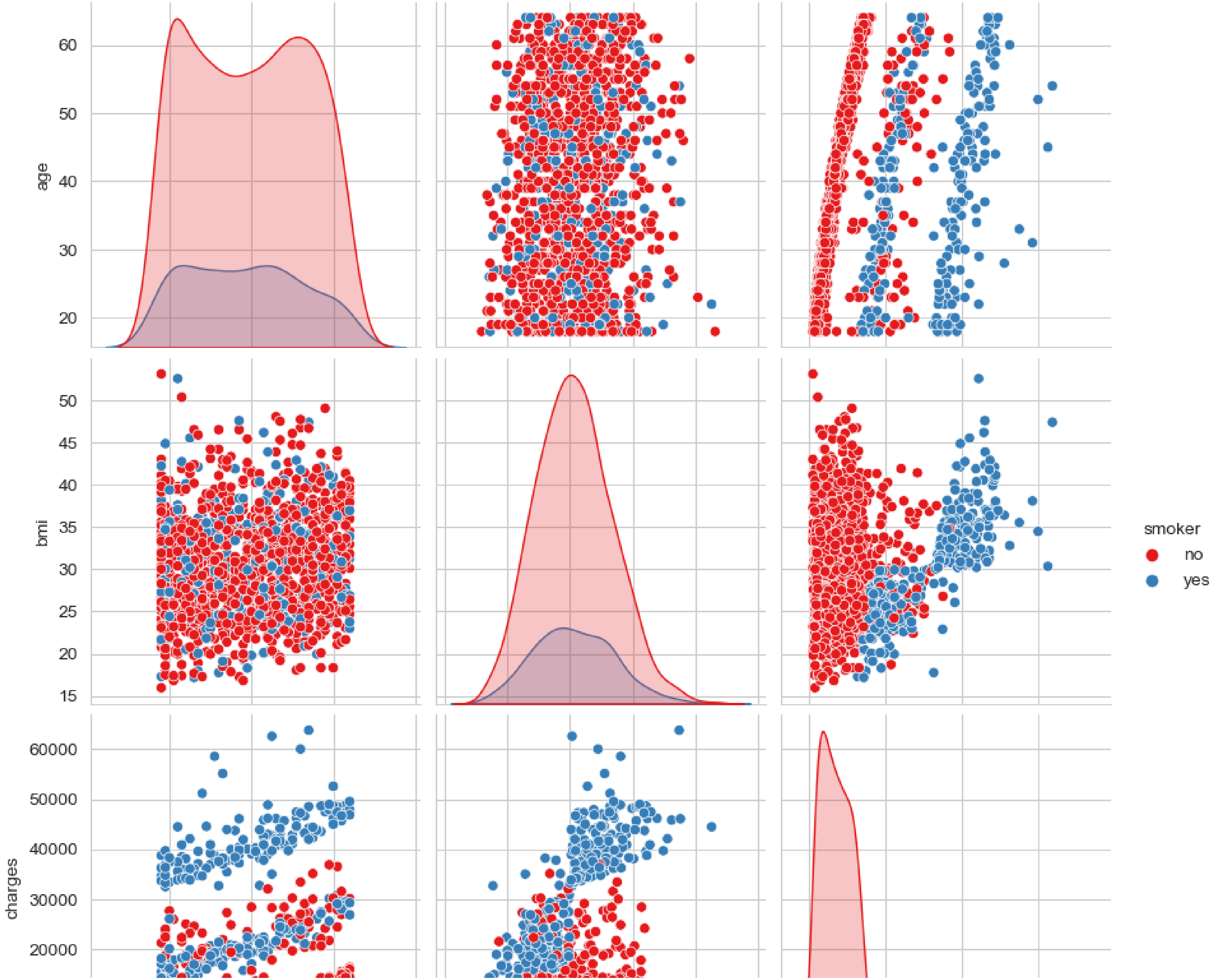
It seems that majority of our customers have 0 or 1 child and median charges vary between 8.5k to 11k dollars.

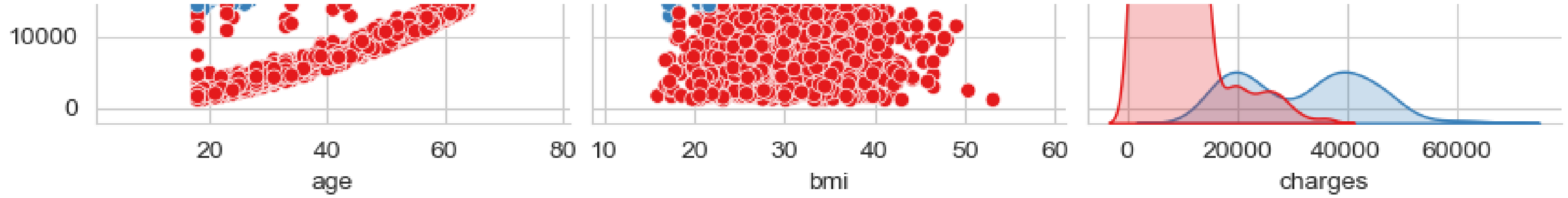
We can also conclude that people who have more children are given less priority in terms of pricing discounts.

In [16]:
```
import seaborn as sns
```

In [17]:
```
sns.set_style("whitegrid")
```

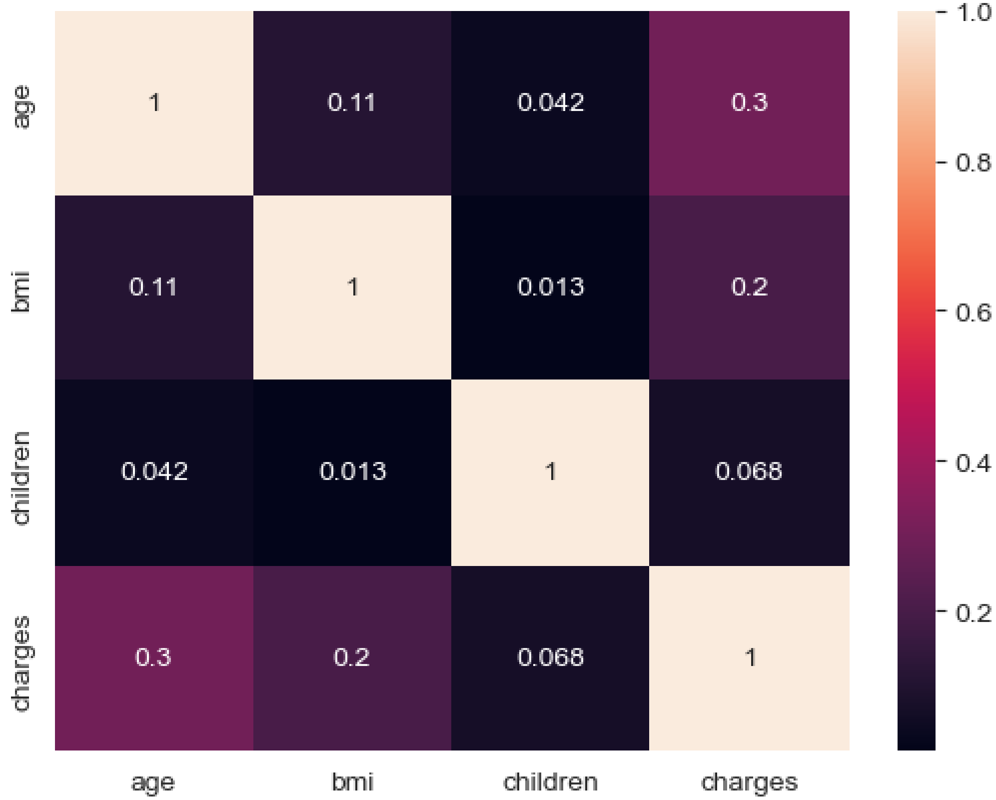
In [18]:
```
import matplotlib.pyplot as plt
sns.pairplot(data[["age", "bmi", "charges", "smoker"]],
            hue = "smoker",
            height = 3,
            palette="Set1")
plt.show()
```


age
bmi
charges
smoker
no
yes
60
50
40
30
20
50
45
40
35
30
25
20
15
60000
50000
40000
30000
20000



From the pairplot we can see that the charges to smokers is realtively high than non-smkoers.

In [19]:
```
sns.heatmap(data.corr(), annot = True)
plt.show()
```



One-Hot Encoding

Some categorical variables have subcategories such as sex and smoker. We need to convert these categorical variables into a form that the scikit-learn library can understand.

In [20]:
```
data.columns
```

Out[20]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

In [21]:
```
data = pd.get_dummies(data)
```

In [22]:
```
data.columns
```

Out[22]:
```
Index(['age', 'bmi', 'children', 'charges', 'sex_female', 'sex_male',
       'smoker_no', 'smoker_yes', 'region_northeast', 'region_northwest',
       'region_southeast', 'region_southwest'],
      dtype='object')
```

Creating Input and Output Variables

In [23]:
```
X = data.drop("charges", axis = 1)
```

In [24]:
```
y = data["charges"]
```

In [25]:
```
target_range = y.max() - y.min()
target_range
```

Out[25]:
```
62648.554110000005
```

Splitting The Dataset

In [26]:
```
from sklearn.model_selection import train_test_split
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size= 0.80, random_state=1)
```

Building The Linear Regression Model

In [28]:
```
from sklearn.linear_model import LinearRegression
```

In [29]:
```
lr = LinearRegression()
```

In [30]:
```
lr.fit(X_train, y_train)
```

Out[30]:
```
LinearRegression()
```

Evaluating The Model

In [31]:
```
lr.score(X_train, y_train).round(3)
```

Out[31]:
```
0.748
```

In [32]:
```
lr.score(X_test, y_test).round(3)
```

Out[32]:
```
0.762
```

The model prediction is realtively near with its training set.

Now let's take a look at another metric, Root mean squared error, to evaluate the model.

In [33]:
```
y_pred = lr.predict(X_test)
```

In [34]:
```
from sklearn.metrics import mean_squared_error
```

RMSLE

In [54]:
```
import numpy as np

# Train your model using X_train and y_train

# Make predictions on the test data
y_pred = lr.predict(X_test)

# Calculate RMSLE
log_y_pred = np.log1p(y_pred)  # Take the logarithm of the predicted values
log_y_test = np.log1p(y_test)  # Take the logarithm of the actual values
squared_diff = (log_y_pred - log_y_test) ** 2  # Calculate squared difference
mean_squared_log_error = np.mean(squared_diff)  # Average of squared differences
rmsle = np.sqrt(mean_squared_log_error)  # Square root of the average

print("RMSLE:", rmsle)
```

```
RMSLE: 0.49544319054018776

C:\Users\angsh\AppData\Local\Temp\ipykernel_14900\1292476574.py:9: RuntimeWarning:

invalid value encountered in log1p
```

We take here RMSLE as the target range is high.

Now try to make my model prediction more good with Random Forest Regressor.

Building The Random Forest Model

In [55]:
```
import numpy as np

# Train your model using X_train and y_train

# Make predictions on the test data
y_pred = rf.predict(X_test)

# Calculate RMSLE
log_y_pred = np.log1p(y_pred)  # Take the logarithm of the predicted values
log_y_test = np.log1p(y_test)  # Take the logarithm of the actual values
squared_diff = (log_y_pred - log_y_test) ** 2  # Calculate squared difference
mean_squared_log_error = np.mean(squared_diff)  # Average of squared differences
rmsle = np.sqrt(mean_squared_log_error)  # Square root of the average

print("RMSLE:", rmsle)
```

```
RMSLE: 0.43228881831505095
```

Here RMSLE is relatively more smaller than the linear regression model , which suggests that the model's predictive performance imporved with it than LR.

Evaluating The Model

In [56]:
```
rf.score(X_train, y_train).round(3)
```

Out[56]:
```
0.976
```

In [57]:
```
rf.score(X_test, y_test).round(3)
```

Out[57]:
```
0.855
```

The Random Forest model score got better than Linear Regression

Building The KNR Model

In [58]:
```
knr.score(X_train, y_train).round(3)
```

Out[58]:
```
0.525
```

In [59]:
```
knr.score(X_test, y_test).round(3)
```

Out[59]:
```
0.284
```

In [60]:
```
import numpy as np

# Train your model using X_train and y_train

# Make predictions on the test data
y_pred = knr.predict(X_test)

# Calculate RMSLE
log_y_pred = np.log1p(y_pred)  # Take the logarithm of the predicted values
log_y_test = np.log1p(y_test)  # Take the logarithm of the actual values
squared_diff = (log_y_pred - log_y_test) ** 2  # Calculate squared difference
mean_squared_log_error = np.mean(squared_diff)  # Average of squared differences
rmsle = np.sqrt(mean_squared_log_error)  # Square root of the average

print("RMSLE:", rmsle)
```

```
RMSLE: 0.7451786340817319
```

Predicting a New Data

I'm going to predict the first row as an example.

In [46]:
```
data_new = X_train[:4]
```

In [47]:
```
data_new
```

Out[47]:

	age	bmi	children	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southw
216	53	26.600	0	1	0	1	0	0	1	0	
731	53	21.400	1	0	1	1	0	0	0	0	
866	18	37.290	0	0	1	1	0	0	0	1	
202	60	24.035	0	1	0	1	0	0	1	0	

Predict with Linear Regressor

In [48]:
```
lr.predict(data_new)
```

Out[48]:
```
array([10508.41885042,  8494.95651816,  4049.96586839, 11485.89042227])
```

Predict with Random Forest Regressor

In [49]:
```
rf.predict(data_new)
```

Out[49]:
```
array([11780.1045341, 11407.0849706,  1325.3675585, 14226.998494 ])
```

Predict with K- Nearest Regressor

In [50]:
```
knr.predict(data_new)
```

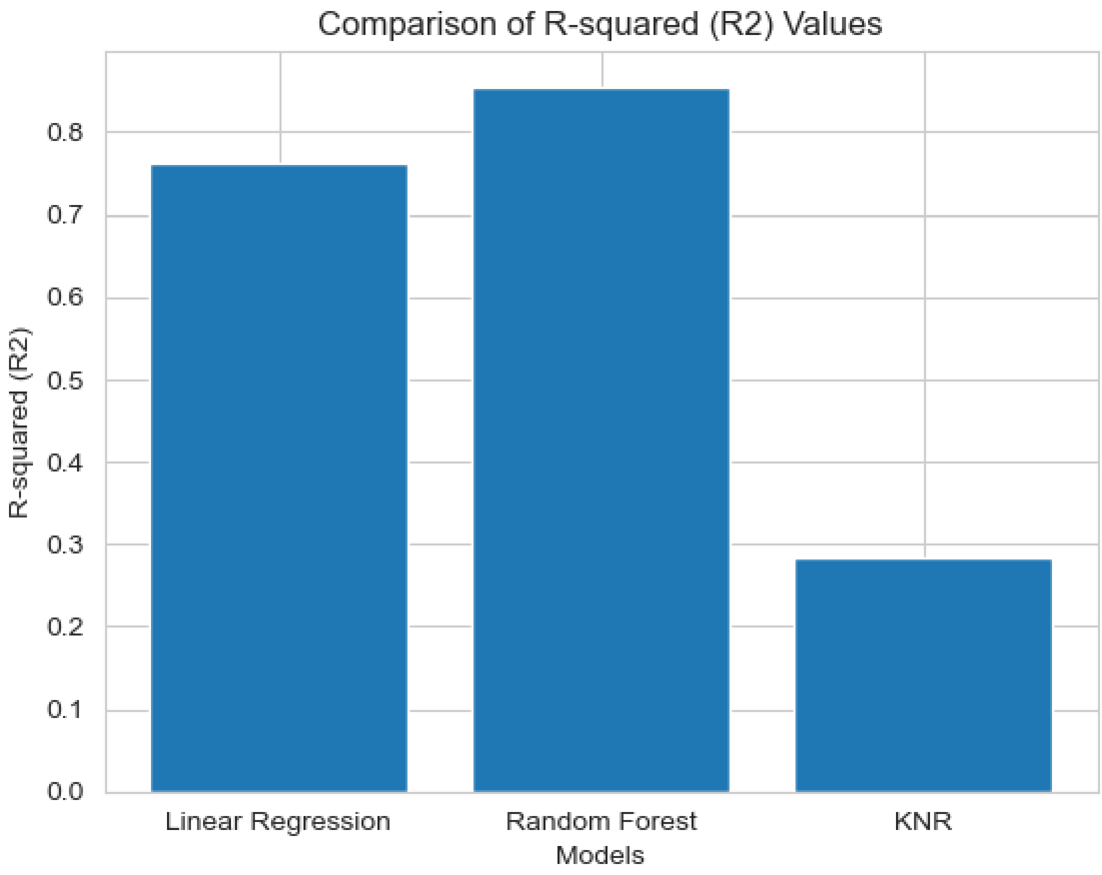
Out[50]:
```
array([13657.71103 , 10217.50862 , 10937.09363 , 18416.929286])
```

So let's see the real value.

In [51]:
```
y_train[:4]
```

Out[51]:
```
216    10355.64100
731    10065.41300
866     1141.44510
202    13012.20865
Name: charges, dtype: float64
```

In [61]:
```
import matplotlib.pyplot as plt
models = ['Linear Regression', 'Random Forest','KNR']
r2_values = [0.762, 0.855, 0.284]
plt.bar(models, r2_values)
plt.xlabel('Models')
plt.ylabel('R-squared (R2)')
plt.title('Comparison of R-squared (R2) Values')
plt.show()
```



As you can see, that out of all the models , Random Forest is giving the most accurate result compare to others. And Hence we can say that Random Forest is the best regressor to us to work on this dataset.

In [ ]:

```

```