Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship, meaning that the change in the dependent variable is directly proportional to the changes in the independent variables.

The goal of linear regression is to estimate parameters that best fit the data, minimizing the difference between the predicted values of Y and the actual values. This estimation is typically done using the least squares method, where the sum of squared differences between the predicted and actual values is minimized.

DataSet

The Data set I will be analyzing here is one which discusses about the personal medical costs in different parts of the US.

This data set includes following features :

1) AGE 2) SEX 3) BMI 4) CHILDREN 5) SMOKER 6) REGION 7) CHARGES

The data set can be found here:

1) Github : https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/insurance.csv

I have also hosted a website for the predicted model which can be found here: https://yogeshmodel.pythonanywhere.com

GOAL

1) To find some valuebale insights from the data 2) To make a LR model to predict medical costs billed by health insurance

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

The warnings library in Python provides a way to handle warning messages issued by the Python interpreter or other libraries. It allows you to control the behavior of warnings, such as whether to ignore them, display them as errors, or filter specific types of warnings.

In [2]:
```
medical_data = pd.read_csv("insurance.csv")
medical_data
```

Out[2]:

	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
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

In [3]:
```
medical_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
```

1) we have 1338 rows with 7 unique columns with different datatypes

First we check for null values and duplicate values

In [4]:
```
# checking for nullvalues

null_values = medical_data.isnull().sum()
print(null_values)
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

We can see there are no null values

In [5]:
```
# checking for duplicate values
```

```
duplicates = medical_data.duplicated(keep=False)
duplicate_rows = medical_data[duplicates]
print(duplicate_rows)
```

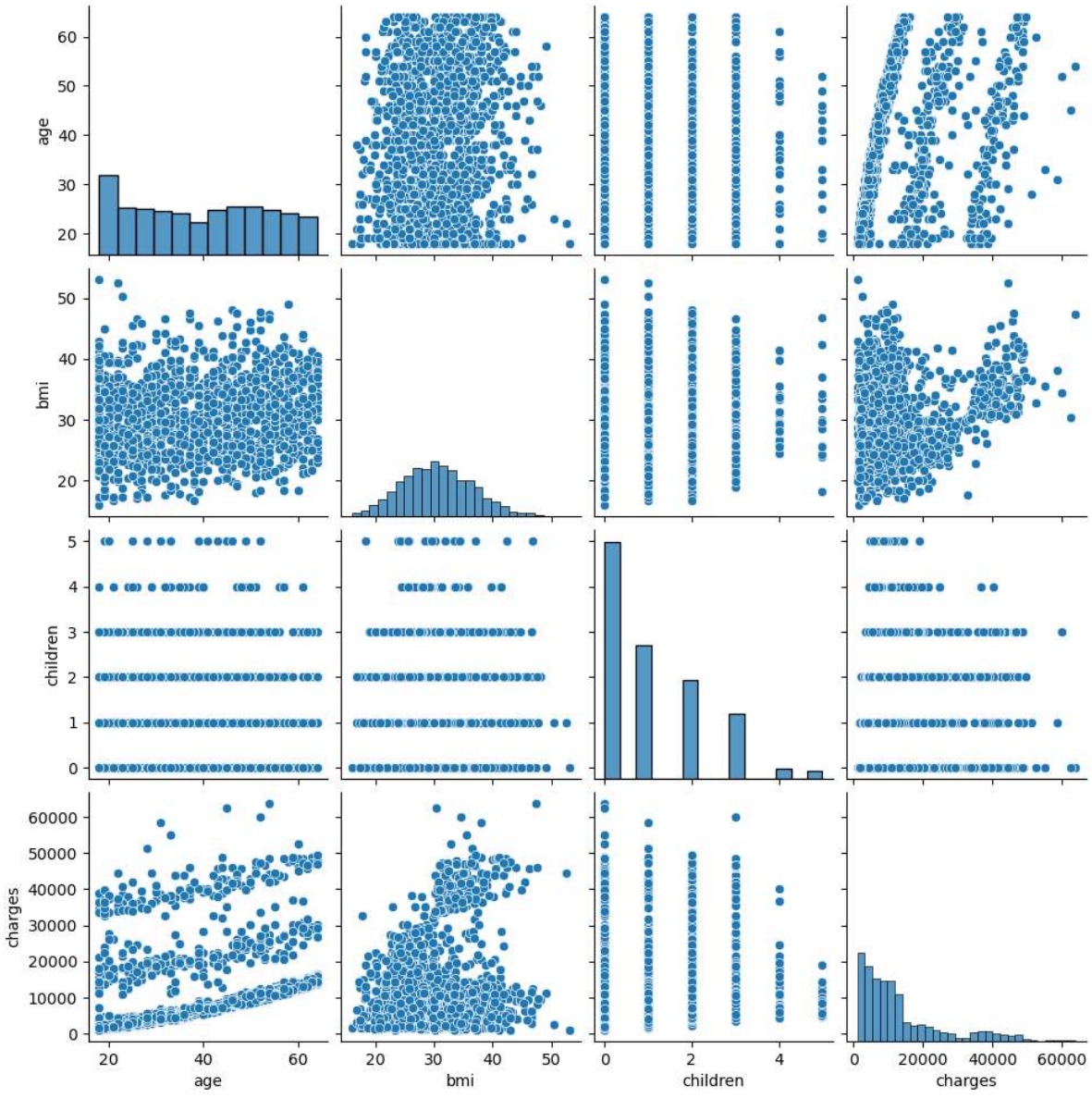
```
     age   sex    bmi  children smoker     region    charges
195   19  male  30.59         0     no  northwest  1639.5631
581   19  male  30.59         0     no  northwest  1639.5631
```

In [6]:
```
# we need to drop the duplicate row

medical_data = medical_data.drop_duplicates()
medical_data.shape
```

Out[6]:
```
(1337, 7)
```

In [7]:
```
# plotting a pair plot for a elementary analysis

sns.pairplot(medical_data)
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x1bbdb555f90>
```



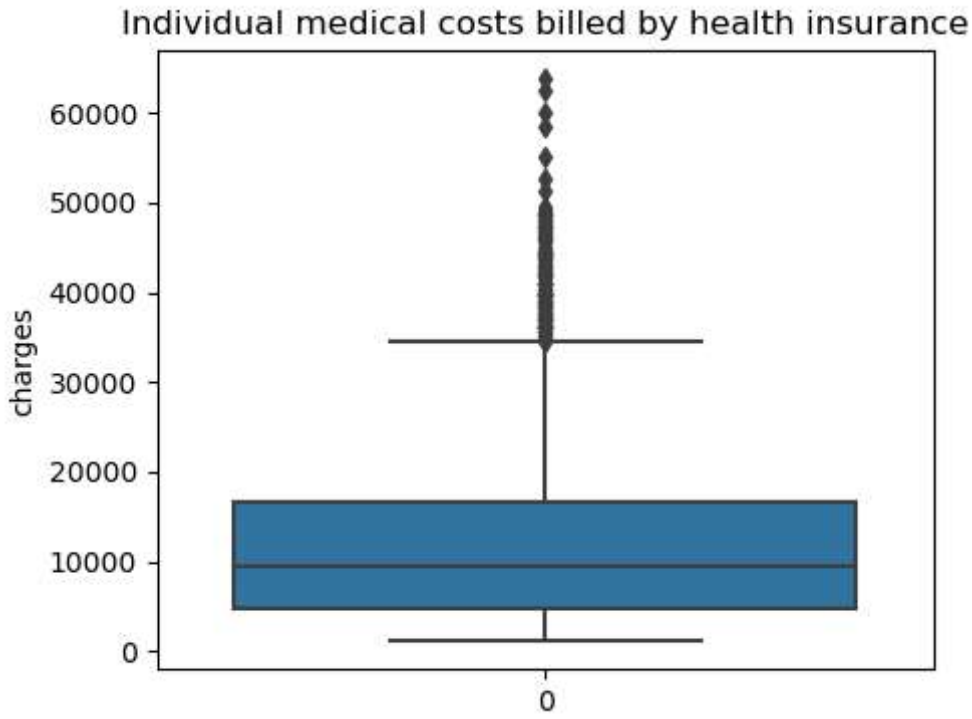
In [8]:
```
# let us make a index column for easier analysis using indexing

medical_data.reset_index(inplace = True)
```

In [9]:
```
medical_data.head()
```

Out[9]:

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

In [10]:
```
# plotting charges to see behaviour of charges

plt.figure(figsize=(5,4))
sns.boxplot(medical_data['charges'])
plt.title("Individual medical costs billed by health insurance")
plt.ylabel("charges")
plt.show()
```



We have used a box plot here which helps us to visualize the outliers , Highest value, lowest value and the median of the data

This shows us that we have some outliers in our data set

In [11]:
```
medical_data.describe(include = "all")
```

Out[11]:

	index	age	sex	bmi	children	smoker	region	charge
count	1337.000000	1337.000000	1337	1337.000000	1337.000000	1337	1337	1337.00000
unique	NaN	NaN	2	NaN	NaN	2	4	Na
top	NaN	NaN	male	NaN	NaN	no	southeast	Na
freq	NaN	NaN	675	NaN	NaN	1063	364	Na
mean	668.565445	39.222139	NaN	30.663452	1.095737	NaN	NaN	13279.12148
std	386.528803	14.044333	NaN	6.100468	1.205571	NaN	NaN	12110.35965
min	0.000000	18.000000	NaN	15.960000	0.000000	NaN	NaN	1121.87390
25%	334.000000	27.000000	NaN	26.290000	0.000000	NaN	NaN	4746.34400
50%	669.000000	39.000000	NaN	30.400000	1.000000	NaN	NaN	9386.16130
75%	1003.000000	51.000000	NaN	34.700000	2.000000	NaN	NaN	16657.71745
max	1337.000000	64.000000	NaN	53.130000	5.000000	NaN	NaN	63770.42801

We can see from the graph and the data that the charges have high standard deviation and hence there must be outliers in the data , we need to take care of them but since our data set is small these outliers must not make a problem , Bills above 38000 can be considered outliers , lets see how many outliers we have

In [12]:
```
medical_data[medical_data['charges']>38000].count().reset_index()
```

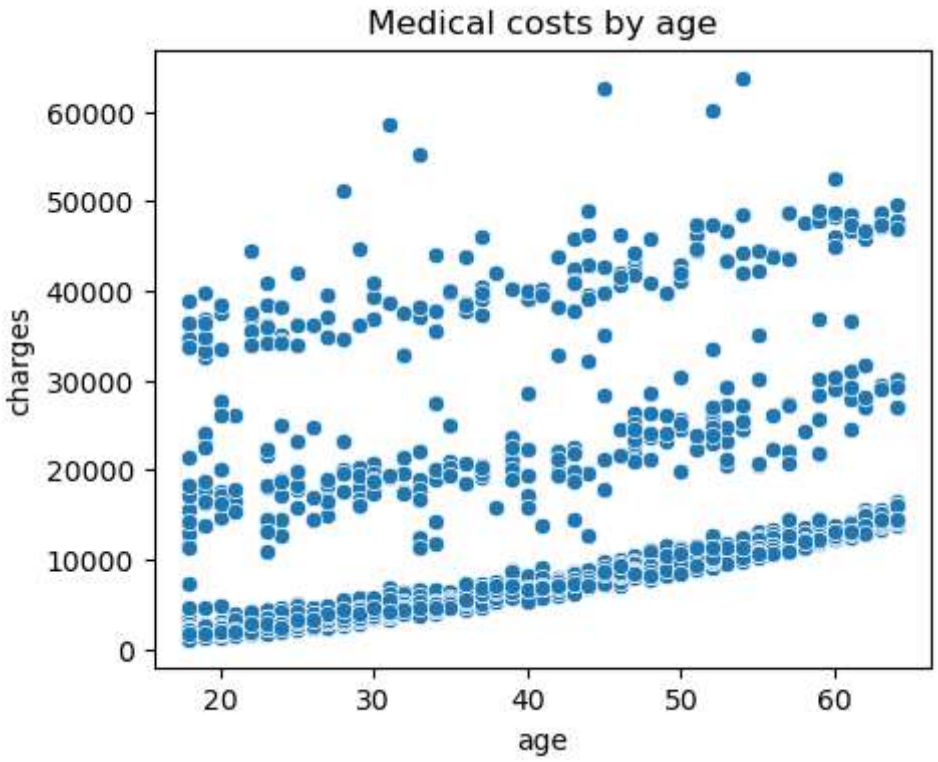
Out[12]:

	index	0
0	index	103
1	age	103
2	sex	103
3	bmi	103
4	children	103
5	smoker	103
6	region	103
7	charges	103

There are only 103 outliers hence we can use them for futheer anaylsis there is no need to drop them

Relation of gender and age on medical insurance bills

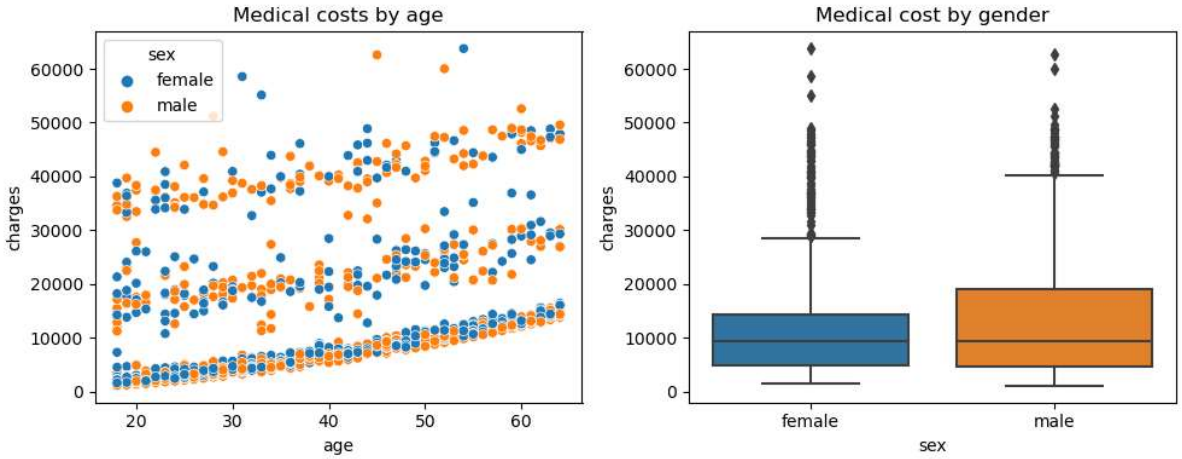
In [13]:
```
plt.figure(figsize=(5,4))
sns.scatterplot(data=medical_data, x='age', y='charges')
plt.title("Medical costs by age")
plt.show()
```



We can see how with age the medical expenses increase hence charges are related to age of the person

Let us now try to analyze the same using gender and age

In [14]:
```
fig, axes = plt.subplots(1, 2, figsize = (10,4))

sns.scatterplot(data=medical_data, x='age', y='charges', hue="sex", ax=axes[0])
axes[0].set_title("Medical costs by age")

sns.boxplot(data=medical_data, x='sex', y='charges', ax=axes[1])
axes[1].set_title("Medical cost by gender")

plt.tight_layout()
plt.show()
```



In [15]:
```
medical_data.groupby(['sex'])['charges'].mean()
```

Out[15]:
```
sex
female    12569.578844
male      13974.998864
Name: charges, dtype: float64
```

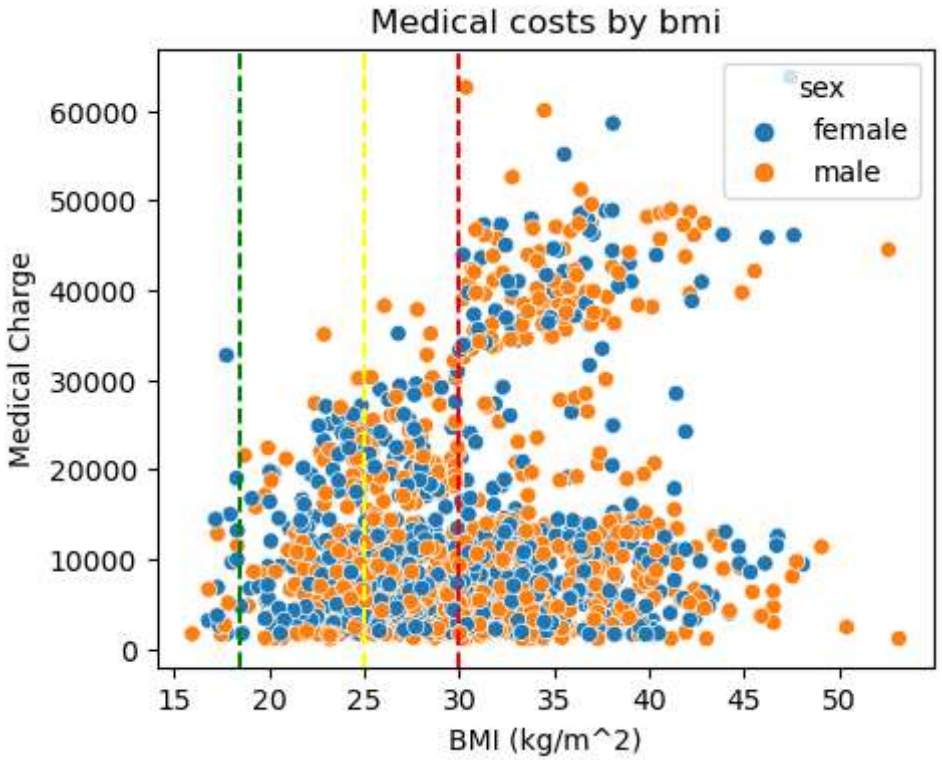
We can see men tend to pay more medical bills , this men tend to have more health concerns than females

Relation of BMI on medical insurance bills

we can refer to govt Data for BMI ranges and check if pople fall in them

1) If your BMI is less than 18.5, it falls within the underweight range. 2) If your BMI is 18.5 to 24.9, it falls within the Healthy Weight range. 3) If your BMI is 25.0 to 29.9, it falls within the overweight range. 4) If your BMI is 30.0 or higher, it falls within the obese range.

In [16]:
```
plt.figure(figsize=(5,4))
sns.scatterplot(data=medical_data, x='bmi', y='charges', hue='sex')
plt.axvline(x=18.5, color="green", linestyle = '--')  # under 18.5 - underweight, l
plt.axvline(x=25, color = "yellow", linestyle = '--')  # between 25~29.9 - Overweig
plt.axvline(x=30, color = "red", linestyle = '--')     # Over 30 - Obese
plt.title("Medical costs by bmi")
plt.xlabel("BMI (kg/m^2)")
plt.ylabel("Medical Charge")
plt.show()
```



In [17]:
```
bins_bmi = [medical_data['bmi'].min(), 18.5, 25, 30, medical_data['bmi'].max()]
charges_30k = medical_data[medical_data['charges']>30000]
charges_30k['bmi'].value_counts(bins=bins_bmi, sort=False).rename_axis('BMI Group'
```

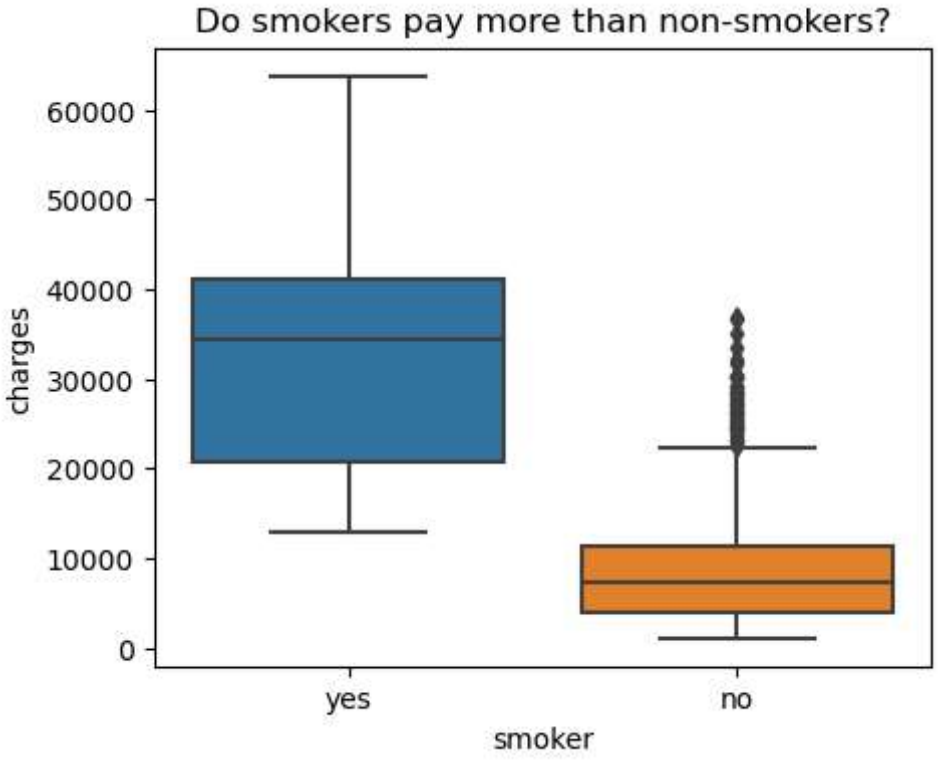
Out[17]:

	BMI Group	Numbers of Individuals
0	(15.959000000000001, 18.5]	1
1	(18.5, 25.0]	2
2	(25.0, 30.0]	10
3	(30.0, 53.13]	149

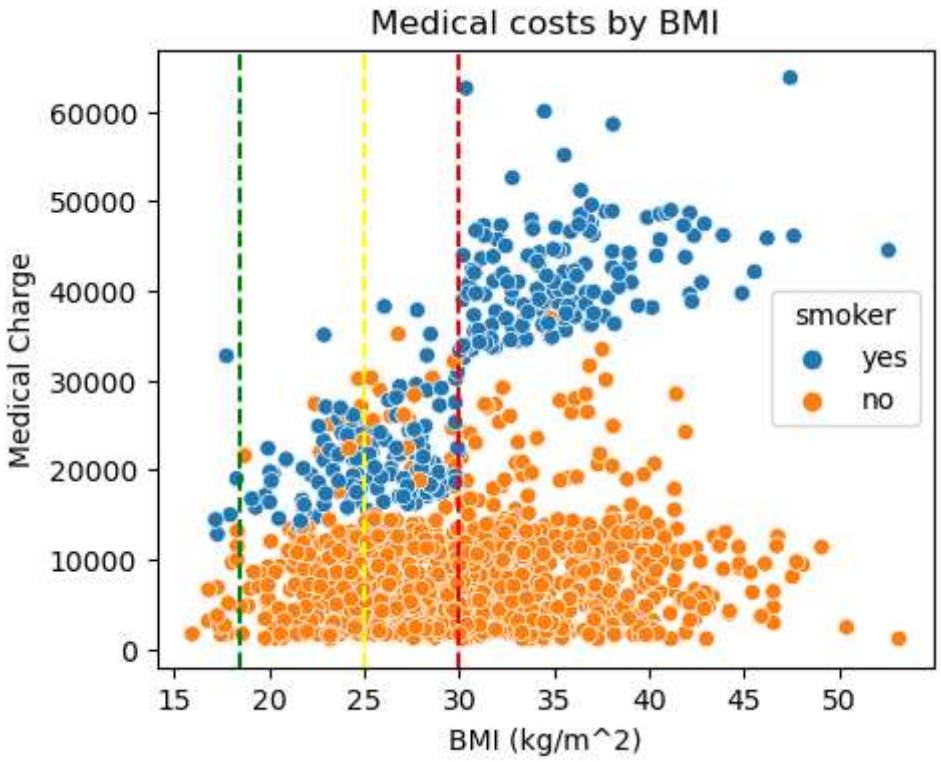
This shows individuals with higher BMI tend to pay more hospital bills , hence are more prone to diseases But why only some obese people tend to pay higher bills?

Relation of BMI on medical insurance bills

In [18]:
```
plt.figure(figsize=(5,4))
sns.boxplot(data=medical_data, x='smoker',y='charges')
plt.title("Do smokers pay more than non-smokers?")
plt.show()
```



In [19]:
```
plt.figure(figsize=(5,4))
sns.scatterplot(data=medical_data, x='bmi', y='charges', hue='smoker')
plt.axvline(x=18.5, color='green', linestyle = '--')  # under 18.5 - underweight, l
plt.axvline(x=25, color = 'yellow', linestyle = '--')  # between 25~29.9 - Overweig
plt.axvline(x=30, color = 'red', linestyle = '--')     # Over 30 - Obese
plt.title("Medical costs by BMI")
plt.xlabel("BMI (kg/m^2)")
plt.ylabel("Medical Charge")
plt.show()
```



In [20]:
```
medical_data.groupby(['smoker'])['charges'].mean()
```

Out[20]:
```
smoker
no      8440.660307
yes    32050.231832
Name: charges, dtype: float64
```

1) The above graph and data shows how smokers have higher bill data. 2) Also we get to know why all obese people dont pay high bills because smoking factor comes in

Applying the linear regression model

First we need to convert categorical values to intger values

In [21]:
```
medical_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1337 entries, 0 to 1336
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   index     1337 non-null   int64  
 1   age       1337 non-null   int64  
 2   sex       1337 non-null   object 
 3   bmi       1337 non-null   float64
 4   children  1337 non-null   int64  
 5   smoker    1337 non-null   object 
 6   region    1337 non-null   object 
 7   charges   1337 non-null   float64
dtypes: float64(2), int64(3), object(3)
memory usage: 83.7+ KB
```

We can see sex , smoker and region are categorical value , we need to convert them to binary

In [22]:
```
medical_data_model = pd.get_dummies(medical_data, columns = ['sex', 'smoker', 'reg
```

In [23]:
```
medical_data_model.head()
```

Out[23]:

	index	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	regi
0	0	19	27.900	0	16884.92400	1	0	0	1	
1	1	18	33.770	1	1725.55230	0	1	1	0	
2	2	28	33.000	3	4449.46200	0	1	1	0	
3	3	33	22.705	0	21984.47061	0	1	1	0	
4	4	32	28.880	0	3866.85520	0	1	1	0	

we need to drop redundant columns

In [24]:
```
medical_data_model = medical_data_model.drop(['index', 'sex_female', 'smoker_no'],
medical_data_model.rename(columns = {'sex_male':'sex', 'smoker_yes':'smoker'}, inp
```

In [25]:
```
medical_data_model.head()
```

Out[25]:

	age	bmi	children	charges	sex	smoker	region_northeast	region_northwest	region_sc
0	19	27.900	0	16884.92400	0	1	0	0	
1	18	33.770	1	1725.55230	1	0	0	0	
2	28	33.000	3	4449.46200	1	0	0	0	
3	33	22.705	0	21984.47061	1	0	0	1	
4	32	28.880	0	3866.85520	1	0	0	1	

Now we can plot a correlational heatmap to get more insights on dependent variables

Correlation in dataset

Correlation refers to the statistical relationship between two or more variables in a dataset. It measures the degree to which the variables are linearly related. Correlation is commonly used to understand the strength and direction of the relationship between variables.

A correlation value ranges from -1 to 1. A correlation coefficient of -1 indicates a perfect negative correlation, where the variables move in opposite directions. A correlation coefficient of 1 indicates a perfect positive correlation, where the variables move in the same direction. A correlation coefficient close to 0 suggests a weak or no correlation between the variables.

Heatmaps are graphical representations of data using a color-coded matrix. In the context of correlation, heatmaps are commonly used to visualize the correlation matrix, which shows the correlation coefficients between pairs of variables in a dataset. The heatmap uses a color gradient to represent the strength of the correlation: warmer colors (e.g., red) indicate

higher positive correlation, cooler colors (e.g., blue) indicate higher negative correlation, and neutral colors (e.g., white) represent no or weak correlation.

Handling correlation

Handling correlations in data preprocessing involves identifying and addressing potential issues that may arise due to correlated variables. Here are a few approaches to handle correlations:

1) Feature Selection: If you have highly correlated variables, you can select a subset of features based on their importance or relevance to the problem. This can help reduce the dimensionality of the dataset and eliminate redundant information.

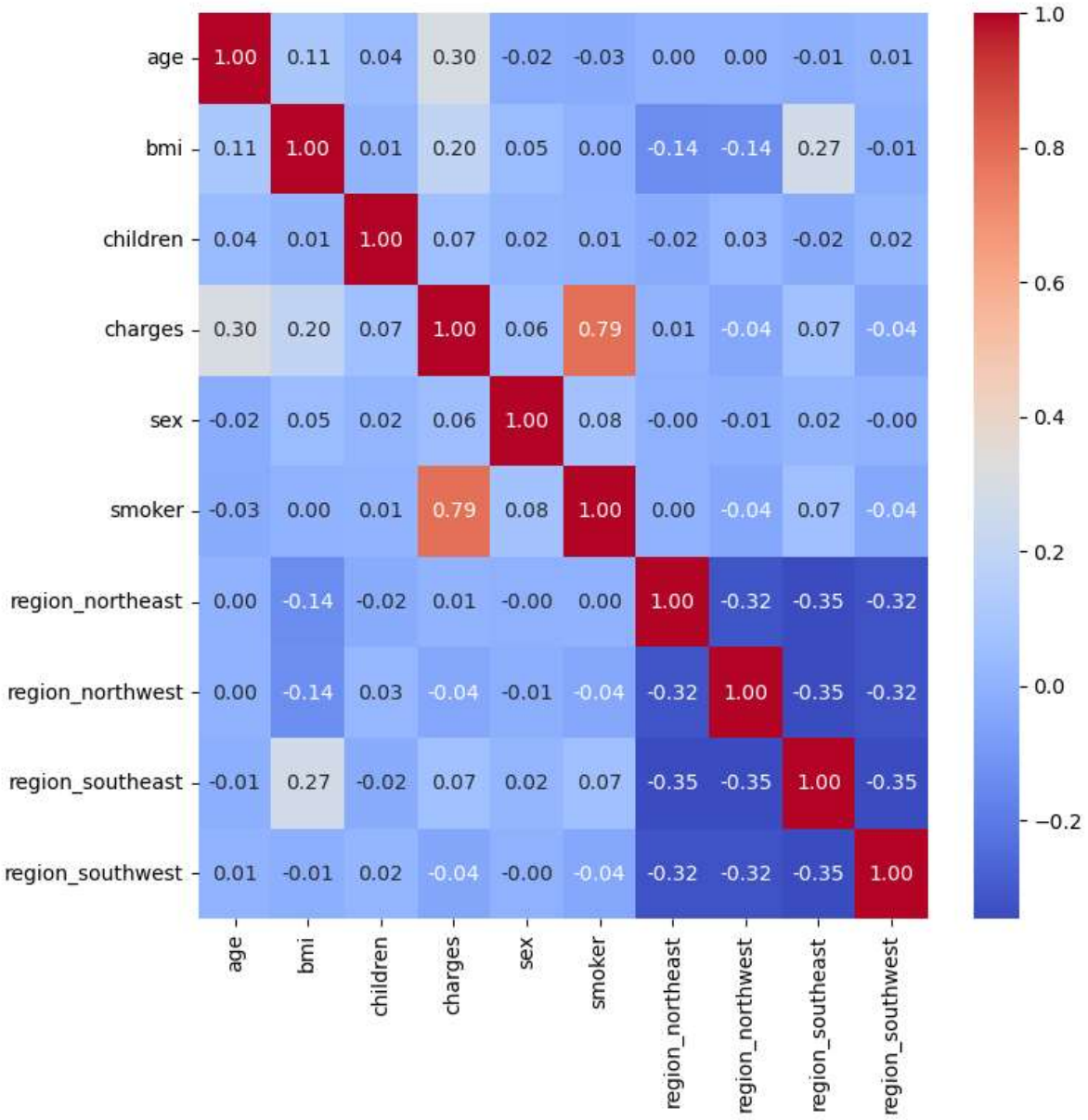
2) Principal Component Analysis (PCA): PCA is a dimensionality reduction technique that can be used to transform correlated variables into a set of uncorrelated variables called principal components. These components capture the maximum variance in the data, allowing you to represent the data in a lower-dimensional space while preserving the most important information.

3) Feature Engineering: You can create new features from the existing ones by combining or transforming them. For example, you can compute ratios, differences, or interaction terms between correlated variables to create new features that provide unique information. This can help mitigate the issues caused by multicollinearity.

4) Regularization: Regularization techniques, such as Ridge regression or Lasso regression, can be used to handle correlated variables in predictive modeling. These methods introduce penalties on the model coefficients, encouraging them to be small or zero, effectively reducing the impact of correlated variables.

In [26]:
```
df_corr = medical_data_model.corr()

plt.figure(figsize = (8,8))
sns.heatmap(df_corr, annot=True, fmt=".2f", cmap="coolwarm" )
```

Out[26]:
```
<Axes: >
```



1) we can see smoker and charges have a positive correlation between them. 2) There is weak correlation between charge and bmi as well as with age 3) southwest region has also positive correlation with bmi

Fitting Data into Linear Regression Model

In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
import lightgbm as ltb
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [28]:
```
medical_data.head()
```

Out[28]:

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

In [29]:
```
# first we need no drop the charges column to analyize the data

X = medical_data.drop(["charges"] , axis=1)
X.head()
```

Out[29]:

	index	age	sex	bmi	children	smoker	region
0	0	19	female	27.900	0	yes	southwest
1	1	18	male	33.770	1	no	southeast
2	2	28	male	33.000	3	no	southeast
3	3	33	male	22.705	0	no	northwest
4	4	32	male	28.880	0	no	northwest

In [30]:
```
y = medical_data['charges']
y.head()
```

Out[30]:
```
0    16884.92400
1     1725.55230
2     4449.46200
3    21984.47061
4     3866.85520
Name: charges, dtype: float64
```

In [31]:
```
# we will know split the data into training set and the test set to check accuracy

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_s
```

In this example, X represents the feature data, y represents the target variable, and test_size indicates the proportion of the data to be allocated to the test set (33% in this case). The random_state parameter is used to ensure reproducibility, so that the split is consistent across multiple runs.

In [32]:
```
# we need to take out categorical values into integer values for out model to work
```

The ColumnTransformer class is particularly useful when dealing with datasets that have heterogeneous data types or require different preprocessing steps for different subsets of features.

This is how we apply the class:

1) Define the transformers: Specify the transformers to be applied to specific subsets of columns. For example, you might want to apply one-hot encoding to categorical features and standard scaling to numerical features. 2) Fit and transform the data: The fit_transform method fits the transformers to the training data and applies the transformations to

produce the transformed data. The transform method applies the previously fitted transformers to the test data.

In [33]:
```
numerical_columns = ['age','bmi' , 'children']
categorical_columns = ['sex','smoker', 'region']
```

In [34]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', "passthrough", numerical_columns),
        ('cat', OneHotEncoder(sparse_output = False, drop = "if_binary"), categori
    ])

# here we have defined how to process the data
```

1) The StandardScaler is a preprocessing class in scikit-learn that is used for standardizing numerical features in a dataset. It is a commonly used technique in machine learning to transform numerical data so that it has a mean of 0 and a standard deviation of 1.

2) The OneHotEncoder is a preprocessing class in scikit-learn that is used to transform categorical features into a binary one-hot encoded representation. It is a common technique used in machine learning to convert categorical variables into a numerical format that can be used by machine learning algorithms.

3) fit_transform is used on the training data (X_train) to learn the unique categories and transform the categorical features into one-hot encoded representation. Then, the same encoder object is used to transform the test data (X_test) using the learned categories.

In [35]:
```
X_train = preprocessor.fit_transform(X_train) # transforming the data according to
```

In [36]:
```
X_train.shape
```

Out[36]:
```
(895, 9)
```

In [37]:
```
X_train[0]
```

Out[37]:
```
array([45.   , 30.495,  1.   ,  0.   ,  1.   ,  0.   ,  1.   ,  0.   ,
        0.   ])
```

In [38]:
```
models = pd.DataFrame(columns=["Model","MSE","R2 Score"])
```

Linear regression

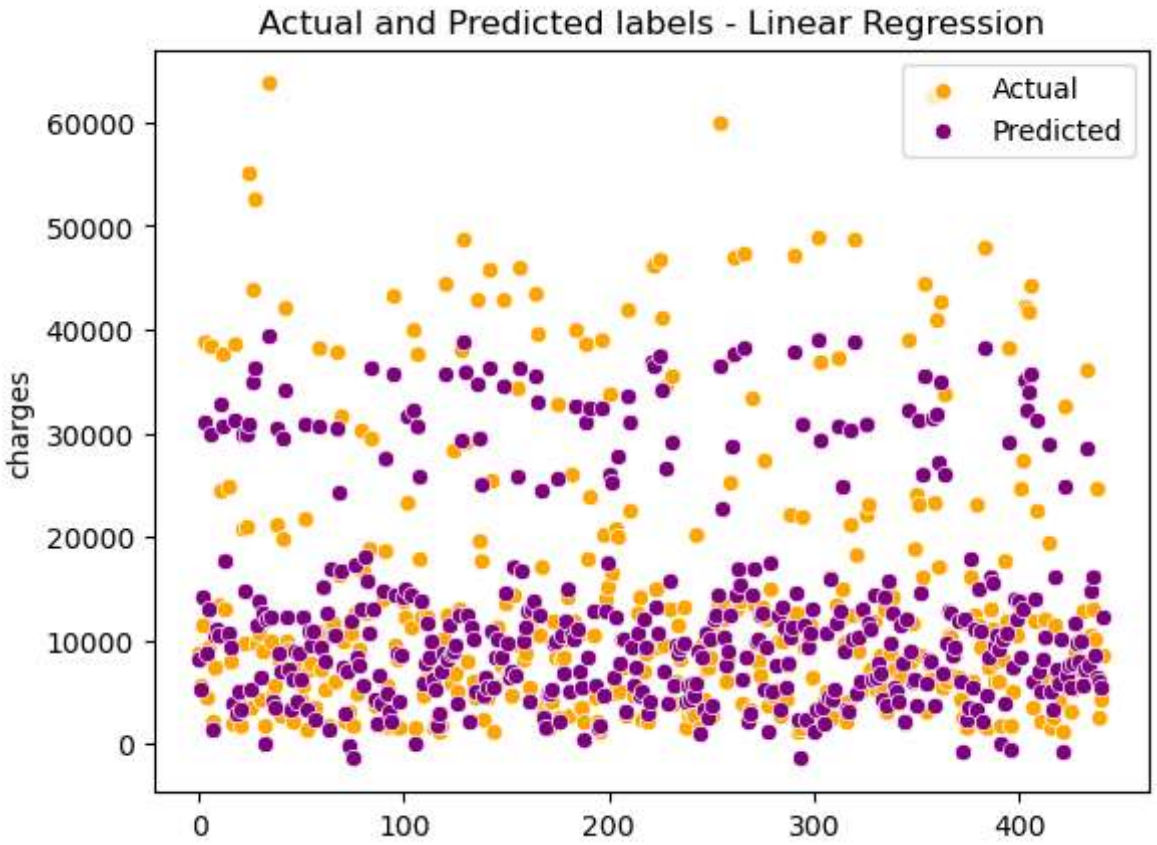
In [39]:
```
linear_model = LinearRegression()
results = linear_model.fit(X_train, y_train)
X_test = preprocessor.transform(X_test)
y_predict = linear_model.predict(X_test)
```

In [40]:
```
mse = mean_squared_error(y_test, y_predict)
r2 = r2_score(y_test, y_predict)
mae = mean_absolute_error(y_test, y_predict)
rmse = mean_squared_error(y_test, y_predict, squared=False)
print(f"Mean Squared error: {mse}" )
print(f"R squared: {r2}" )
print(f"mean absoulte error: {mae}" )
print(f"mean squared error: {rmse}" )
```

```
new_row = {"Model": "Linear Regression","MSE": mse, "R2 Score": r2 , "MAE": mae, "
models = models.append(new_row, ignore_index=True)
```

```
Mean Squared error: 38565934.50286132
R squared: 0.7732755908459318
mean absoulte error: 4190.934869866732
mean squared error: 6210.147703787835
```

In [41]:
```
sns.scatterplot(y = y_test, x = range(len(y_test)), color  ='orange', label = "Actu
sns.scatterplot(y = y_predict, x = range(len(y_predict)), color = 'purple', label =
plt.title("Actual and Predicted labels - Linear Regression")
plt.show()
```



XGboost Regression

1) Regularization: XGBoost uses regularization techniques to prevent overfitting and improve generalization.

2) Handling missing values: XGBoost can automatically handle missing values in the dataset, reducing the need for data preprocessing.

3) Feature importance: XGBoost provides insights into the importance of features in the prediction task, helping to identify the most influential variables.

4) Handling imbalanced datasets: XGBoost includes strategies to handle imbalanced datasets, where the number of instances in different classes is disproportionate.

5) Scalability: XGBoost is designed for efficiency and scalability, making it suitable for large datasets and parallel processing.

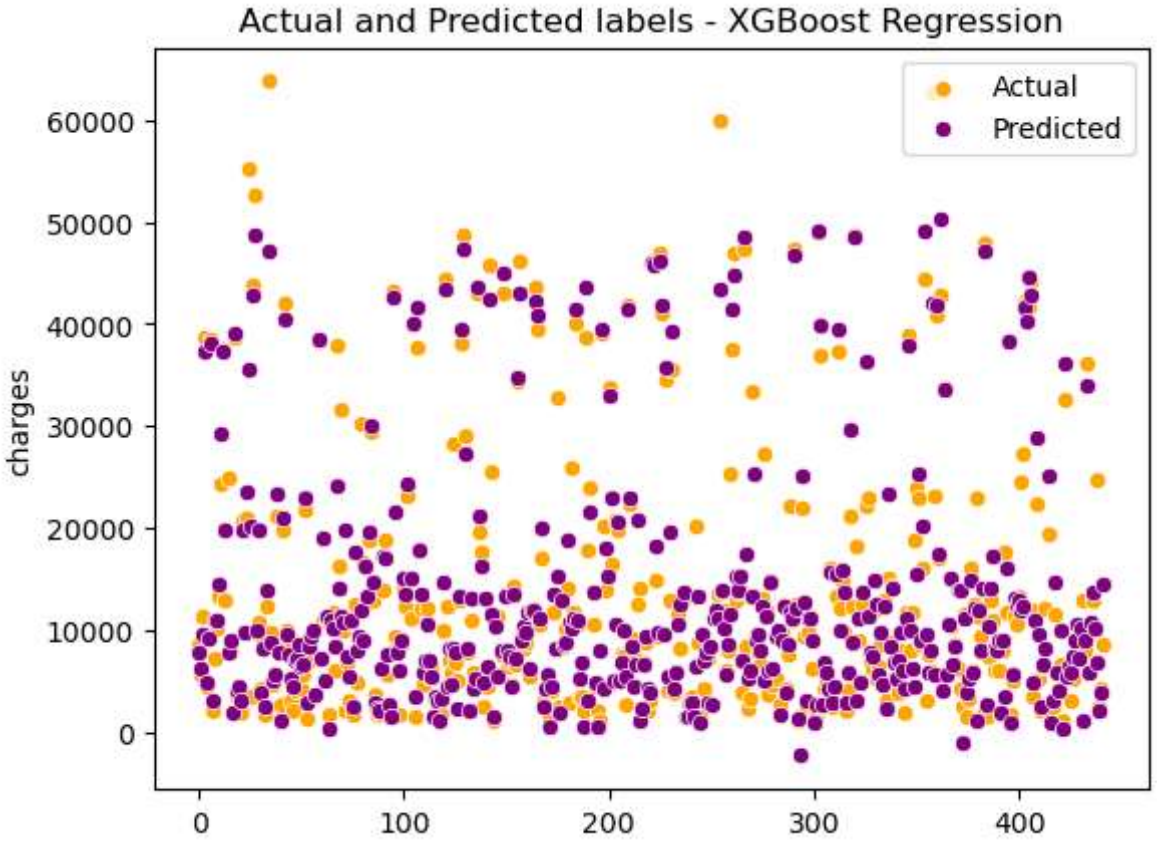
In [42]:
```
xgbr = xgb.XGBRegressor(random_state =0)
xgbr.fit(X_train, y_train)
```

```
y_xgbr = xgbr.predict(X_test)
```

In [43]:
```
mse = mean_squared_error(y_test, y_xgbr)
r2 = r2_score(y_test, y_xgbr)
mae = mean_absolute_error(y_test, y_xgbr)
rmse = mean_squared_error(y_test, y_xgbr, squared=False)
print(f"Mean Squared error: {mse}" )
print(f"R squared: {r2}" )
print(f"mean absoulte error: {mae}" )
print(f"mean squared error: {rmse}" )

new_row = {"Model": "XGBoost Regression","MSE": mse, "R2 Score": r2 , "MAE": mae,
models = models.append(new_row, ignore_index=True)
```

```
Mean Squared error: 27538033.0559948
R squared: 0.8381072738319775
mean absoulte error: 2890.0148363279836
mean squared error: 5247.669297506733
```

In [44]:
```
sns.scatterplot(y = y_test, x = range(len(y_test)), color  ='orange', label = "Actu
sns.scatterplot(y = y_xgbr, x = range(len(y_xgbr)), color = 'purple', label = "Pred
plt.title("Actual and Predicted labels - XGBoost Regression")
plt.show()
```



LightGBM regression

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework that uses tree-based learning algorithms. LightGBM regression is a variant of LightGBM specifically designed for regression tasks.

1) Efficiency and speed: LightGBM is designed to be highly efficient and can handle large-scale datasets with millions of instances and thousands of features. It employs techniques such as histogram-based binning and parallel learning to achieve faster training and prediction times.

2) Handling categorical features: LightGBM natively supports categorical features without requiring one-hot encoding. It can directly work with categorical variables and can automatically find the best split points for categorical features during tree construction.

3) Flexibility and customization: LightGBM provides numerous hyperparameters that can be tuned to customize the model's behavior and optimize performance. It offers options to control tree growth, regularization, learning rates, and more.

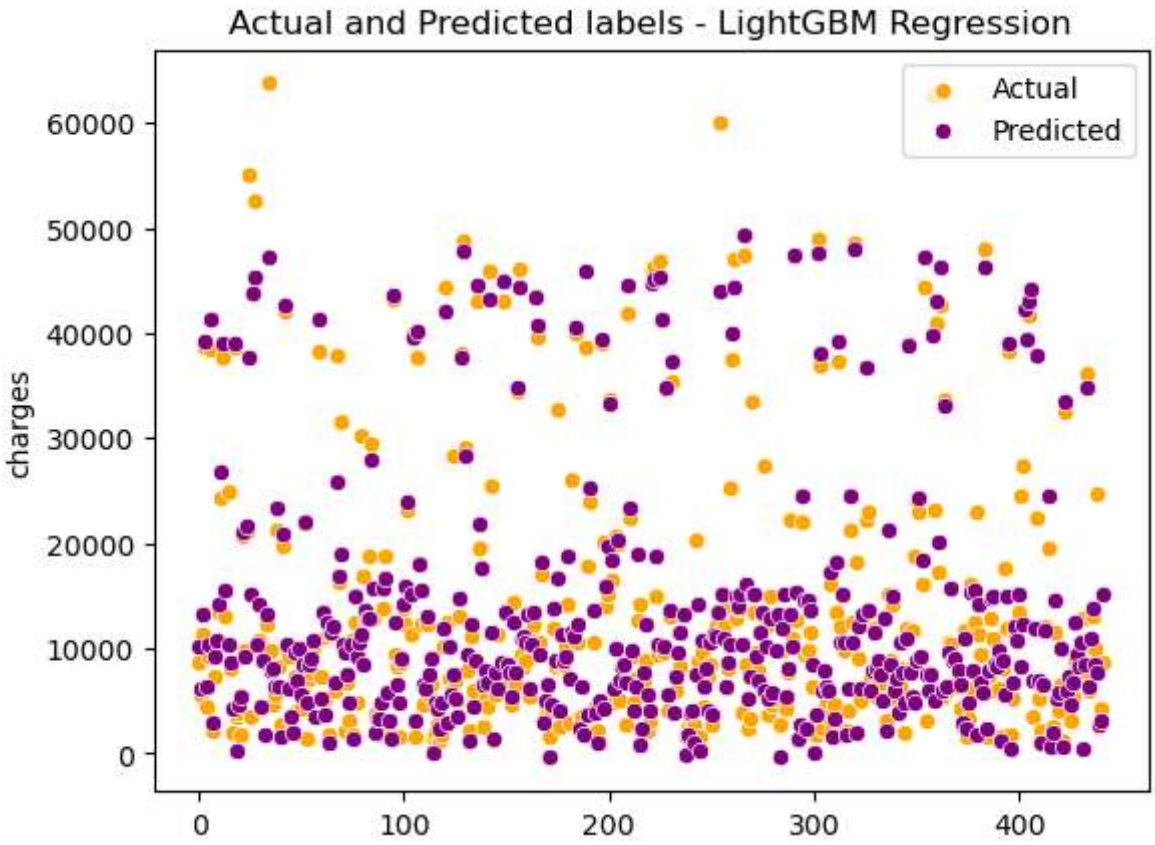
4) High accuracy: LightGBM has shown competitive performance on various benchmark datasets and is known for its ability to handle complex patterns and capture nonlinear relationships between features and the target variable.

In [45]:
```
lgbm = ltb.LGBMRegressor()
lgbm.fit(X_train, y_train)
y_lgbm = lgbm.predict(X_test)
```

In [46]:
```
mse = mean_squared_error(y_test, y_lgbm)
r2 = r2_score(y_test, y_lgbm)
mae = mean_absolute_error(y_test, y_lgbm)
rmse = mean_squared_error(y_test, y_lgbm, squared=False)
print(f"Mean Squared error: {mse}" )
print(f"R squared: {r2}" )
print(f"mean absoulte error: {mae}" )
print(f"mean squared error: {rmse}" )

new_row = {"Model": "LightGBM regression","MSE": mse, "R2 Score": r2 , "MAE": mae,
models = models.append(new_row, ignore_index=True)
```

```
Mean Squared error: 24206940.918975655
R squared: 0.8576903568387537
mean absoulte error: 2810.7799838499323
mean squared error: 4920.054971133519
```

In [47]:
```
sns.scatterplot(y = y_test, x = range(len(y_test)), color  ='orange', label = "Act
sns.scatterplot(y = y_lgbm, x = range(len(y_lgbm)), color = 'purple', label = "Pre
plt.title("Actual and Predicted labels - LightGBM Regression")
plt.show()
```



Random Forest Regression

Random Forest Regression is a machine learning algorithm that uses an ensemble of decision trees for regression tasks. It is an extension of the Random Forest algorithm, which is widely used for classification tasks.

1) Robustness to noise and outliers: Random Forest Regression can handle noisy and outlier-prone data due to the averaging effect of multiple trees. Outliers have less impact on the final prediction compared to individual decision trees.

2) Nonlinearity and interactions: Random Forest Regression can capture complex nonlinear relationships and interactions between features and the target variable. It can handle both categorical and numerical features without requiring feature scaling or one-hot encoding.

3) Feature importance: Random Forest Regression provides a measure of feature importance, indicating which features are most influential in making predictions. This information can be valuable for feature selection and understanding the underlying patterns in the data.

4) Handling missing values: Random Forest Regression can handle missing values in the dataset by using surrogate splits. It leverages other features to make accurate predictions even when some values are missing.

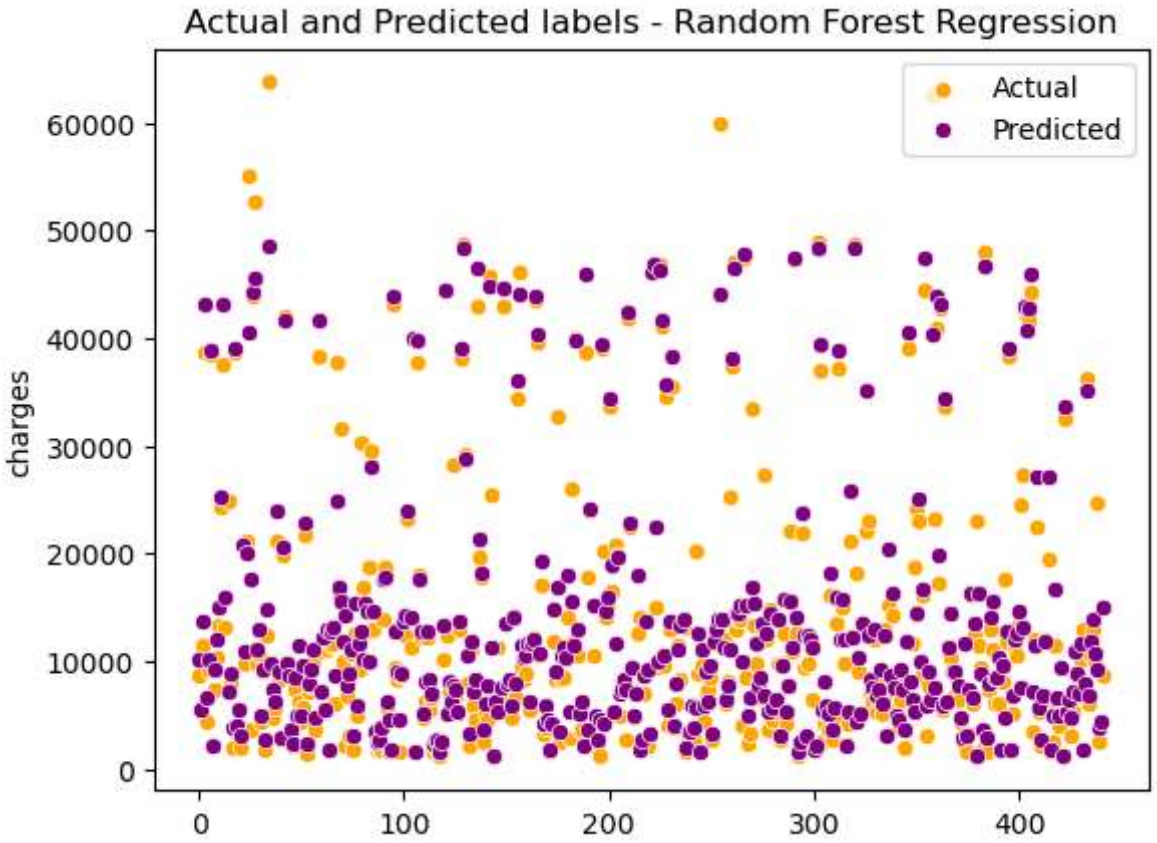
In [48]:
```
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)
y_rf = random_forest.predict(X_test)
```

In [49]:
```
mse = mean_squared_error(y_test, y_rf)
r2 = r2_score(y_test, y_rf)
mae = mean_absolute_error(y_test, y_rf)
```

```
rmse = mean_squared_error(y_test, y_rf, squared=False)
print(f"Mean Squared error: {mse}" )
print(f"R squared: {r2}" )
print(f"mean absoulte error: {mae}" )
print(f"mean squared error: {rmse}" )

new_row = {"Model": "Random Forest Regression","MSE": mse, "R2 Score": r2 , "MAE":
models = models.append(new_row, ignore_index=True)
```

```
Mean Squared error: 22092850.821999095
R squared: 0.8701188337916437
mean absoulte error: 2517.2268961056566
mean squared error: 4700.3032691518
```

In [50]:
```
sns.scatterplot(y = y_test, x = range(len(y_test)), color  ='orange', label = "Actu
sns.scatterplot(y = y_rf, x = range(len(y_rf)), color = 'purple', label = "Predicte
plt.title("Actual and Predicted labels - Random Forest Regression")
plt.show()
```



In [51]:
```
models.sort_values(by="R2 Score", ascending=False)
```

Out[51]:

	Model	MSE	R2 Score	MAE	RMSE
3	Random Forest Regression	2.209285e+07	0.870119	2517.226896	4700.303269
2	LightGBM regression	2.420694e+07	0.857690	2810.779984	4920.054971
1	XGBoost Regression	2.753803e+07	0.838107	2890.014836	5247.669298
0	Linear Regression	3.856593e+07	0.773276	4190.934870	6210.147704

Pickling the Model

In [52]:
```
import pickle
```

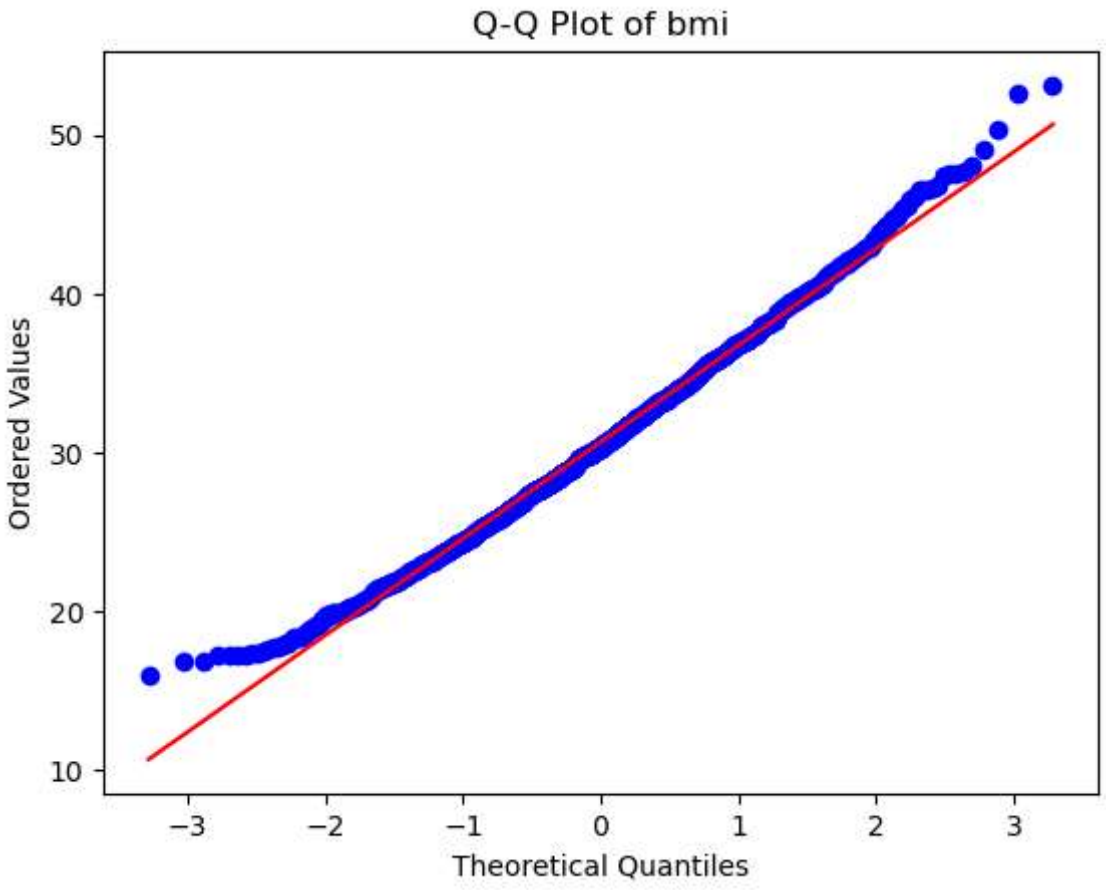
In [53]:
```
# we will pickle our trained model with best fit that is Random Forest Regression

with open('RFmodel.pkl', 'wb') as file:
    pickle.dump(random_forest, file)
```

Predicting the new Dataset using the pickled Dataset

we will generate a synthetic data set for our model:

1) for age we will choose a random age between 18-65 as min age in our data is 18 and highest is 65 2) sex and smoker is binary hence we choose a random number between 0,1 3) similarly region also contains only 4 values 4) no of children is a random integer from 0-5

In [54]:
```
import scipy.stats as stats

# Create a Q-Q plot of the column against a normal distribution
stats.probplot(medical_data['bmi'], dist="norm", plot=plt)
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Ordered Values')
plt.title('Q-Q Plot of bmi')
plt.show()
```



we plot our bmi in a Q-Q plot showing distribution deviation from the normal distribution hence we can safely assume normal distribution of BMI from 15-55

In [55]:
```
# now we can start making our new synthetic dataset
import numpy as np
num_samples = 5000 #number of samples

# synthetic data for each dataset
```

```
age = np.random.randint(18, 65, size=num_samples)  # Random ages between 18 and 65
sex = np.random.choice(['male', 'female'], size=num_samples)
smoker = np.random.choice(['yes', 'no'], size=num_samples)
bmi = np.random.normal(30, 6, size=num_samples)  # Normal distribution with mean 3
children = np.random.randint(0, 5, size=num_samples)  # Random number of children
region = np.random.choice(['northeast', 'northwest', 'southeast', 'southwest'], si

# Create a DataFrame to store the synthetic data
synthetic_medical_data = pd.DataFrame({
    'age': age,
    'sex': sex,
    'smoker': smoker,
    'BMI': bmi,
    'children': children,
    'region': region
})

# Display the first few rows of the new dataset
print(synthetic_medical_data.head())

# Save the synthetic dataset to a CSV file
synthetic_medical_data.to_csv('synthetic_medical_data.csv', index=False)
```

```
   age     sex smoker        BMI  children     region
0   23    male     no  30.593565         3  northeast
1   59    male     no  38.012515         3  southwest
2   63    male     no  18.999836         2  southeast
3   44  female    yes  25.125429         3  southeast
4   27    male    yes  19.783942         4  southeast
```

Now we can repeat the above procedure to predict the model

In [56]:
```
synthetic_medical_data = pd.read_csv("synthetic_medical_data.csv")
```

In [57]:
```
synthetic_medical_data.head()
```

Out[57]:

	age	sex	smoker	BMI	children	region
0	23	male	no	30.593565	3	northeast
1	59	male	no	38.012515	3	southwest
2	63	male	no	18.999836	2	southeast
3	44	female	yes	25.125429	3	southeast
4	27	male	yes	19.783942	4	southeast

In [58]:
```
synthetic_medical_data.describe(include = "all")
```

Out[58]:

	age	sex	smoker	BMI	children	region
count	5000.000000	5000	5000	5000.000000	5000.000000	5000
unique	NaN	2	2	NaN	NaN	4
top	NaN	female	yes	NaN	NaN	northwest
freq	NaN	2522	2535	NaN	NaN	1284
mean	40.964600	NaN	NaN	29.871220	1.985000	NaN
std	13.440933	NaN	NaN	6.016956	1.417313	NaN
min	18.000000	NaN	NaN	10.910630	0.000000	NaN
25%	29.000000	NaN	NaN	25.798050	1.000000	NaN
50%	41.000000	NaN	NaN	29.971962	2.000000	NaN
75%	53.000000	NaN	NaN	33.828259	3.000000	NaN
max	64.000000	NaN	NaN	53.553131	4.000000	NaN

In [59]:
```
numerical_columns = ['age','BMI' , 'children']
categorical_columns = ['sex','smoker', 'region']
```

In [60]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', "passthrough", numerical_columns),
        ('cat', OneHotEncoder(sparse_output = False, drop = "if_binary"), categori
    ])
```

In [61]:
```
X_syntest = synthetic_medical_data
X_syntest.head()
```

Out[61]:

	age	sex	smoker	BMI	children	region
0	23	male	no	30.593565	3	northeast
1	59	male	no	38.012515	3	southwest
2	63	male	no	18.999836	2	southeast
3	44	female	yes	25.125429	3	southeast
4	27	male	yes	19.783942	4	southeast

In [62]:
```
X_syntest = preprocessor.fit_transform(X_syntest)
```

In [63]:
```
X_syntest.shape
```

Out[63]:
```
(5000, 9)
```

In [64]:
```
X_syntest[0]
```

Out[64]:
```
array([23.       , 30.5935652,  3.       ,  1.       ,  0.       ,
        1.       ,  0.       ,  0.       ,  0.       ])
```

In [65]:
```
#loading our model
with open('RFmodel.pkl', 'rb') as file:
    RFmodel = pickle.load(file)
```

In [66]:
```
y_charges_predict = RFmodel.predict(X_syntest)
```

In [67]:
```
y_charges_predict.shape
```

Out[67]:
```
(5000,)
```

In [68]:
```
synthetic_medical_data["predicted_charges"] = y_charges_predict
```

In [69]:
```
synthetic_medical_data.head()
```

Out[69]:

	age	sex	smoker	BMI	children	region	predicted_charges
0	23	male	no	30.593565	3	northeast	8932.165747
1	59	male	no	38.012515	3	southwest	17828.371664
2	63	male	no	18.999836	2	southeast	19261.955873
3	44	female	yes	25.125429	3	southeast	23543.059122
4	27	male	yes	19.783942	4	southeast	16339.903036