# Step - 1 : Business Problem Understanding

In [1]:

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

# Step - 2 : Data Understanding

**Load Data & Understand every variable**

In [2]:

```
df = pd.read_excel("insurance.xlsx")
df.head()
```

Out[2]:

| | age | sex | bmi | children | smoker | region | expenses |
|---|---|---|---|---|---|---|---|
| **0** | 19 | female | 27.9 | 0 | yes | southwest | 16884.92 |
| **1** | 18 | male | 33.8 | 1 | no | southeast | 1725.55 |
| **2** | 28 | male | 33.0 | 3 | no | southeast | 4449.46 |
| **3** | 33 | male | 22.7 | 0 | no | northwest | 21984.47 |
| **4** | 32 | male | 28.9 | 0 | no | northwest | 3866.86 |

**Dataset Understanding**

In [3]:

```
df.shape
```

Out[3]:

```
(1338, 7)
```

In [4]:

```
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   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

**Exploratory Data Analysis**

In [5]:

```
categorical=[]
continous=[]
check =[]

d_types = dict(df.dtypes)
for name , type in d_types.items():
    if str(type) == 'object':
        categorical.append(name)
    elif str(type) == 'float64':
        continous.append(name)
    else:
        check.append(name)

print("categorical features:",categorical)
print("continous features:",continous)
print("features to be checked:",check)
```

```
categorical features: ['sex', 'smoker', 'region']
continous features: ['bmi', 'expenses']
features to be checked: ['age', 'children']
```

In [6]:

```
d_types = dict(df.dtypes)
for name , type_ in d_types.items():
    if str(type_) == 'object':
        print(f"<======== {name} ===========>")
        print(df[name].value_counts())
```

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

In [7]:

```
df.describe()
```

Out[7]:

| | age | bmi | children | expenses |
|---|---|---|---|---|
| **count** | 1338.000000 | 1338.000000 | 1338.000000 | 1338.000000 |
| **mean** | 39.207025 | 30.665471 | 1.094918 | 13270.422414 |
| **std** | 14.049960 | 6.098382 | 1.205493 | 12110.011240 |
| **min** | 18.000000 | 16.000000 | 0.000000 | 1121.870000 |
| **25%** | 27.000000 | 26.300000 | 0.000000 | 4740.287500 |
| **50%** | 39.000000 | 30.400000 | 1.000000 | 9382.030000 |
| **75%** | 51.000000 | 34.700000 | 2.000000 | 16639.915000 |
| **max** | 64.000000 | 53.100000 | 5.000000 | 63770.430000 |

In [8]:

```
df.corr()
```

Out[8]:

| | age | bmi | children | expenses |
|---|---|---|---|---|
| **age** | 1.000000 | 0.109341 | 0.042469 | 0.299008 |
| **bmi** | 0.109341 | 1.000000 | 0.012645 | 0.198576 |
| **children** | 0.042469 | 0.012645 | 1.000000 | 0.067998 |
| **expenses** | 0.299008 | 0.198576 | 0.067998 | 1.000000 |

# Step - 3 : Data Preprocessing

In [9]:

```
df.isnull().sum()
```

Out[9]:

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

In [10]:

```
#drop the region column
df.drop('region', axis=1,inplace=True)
```

In [11]:

```
# encoding sex column
df['sex'].replace({'female':0,'male':1}, inplace=True)

# encoding 'smoker' column
df['smoker'].replace({'no':0,'yes':1}, inplace=True)
```

In [12]:

```
X = df.drop('expenses', axis=1)
y = df['expenses']
```

In [13]:

```
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=9)
```

# Step - 4,5 : Modelling & Evaluation

**Lasso Regression with default parameters**

In [14]:

```
# Modelling
from sklearn.linear_model import Ridge
ridge_base = Ridge()
ridge_base.fit(X_train,y_train)

#predictions
train_predictions = ridge_base.predict(X_train)
test_predictions = ridge_base.predict(X_test)

print("Train R2:",ridge_base.score(X_train,y_train))
print("Test R2:",ridge_base.score(X_test,y_test))

from sklearn.model_selection import cross_val_score
print("Cross Validation Score:",cross_val_score(ridge_base,X,y,cv=5).mean())
```

```
Train R2: 0.7432963847740974
Test R2: 0.7754580997793767
Cross Validation Score: 0.7466523551462286
```

**Applying Hyperparameter tuning for Ridge Regression**

In [15]:

```
from sklearn.model_selection import GridSearchCV

# model
estimator = Ridge()

# parameters & values
param_grid = {"alpha":[0.1,0.2,0.5,0.7,1,10,50,100,1000]}

#Identifying the best value of the parameter within given values for the given data
model_hp = GridSearchCV(estimator,param_grid,cv=5,scoring='neg_mean_squared_error')
model_hp.fit(X_train,y_train)
model_hp.best_params_
```

Out[15]:

```
{'alpha': 0.1}
```

#### Rebuilt Ridge Model using best hyperparameters

In [16]:

```
#Modelling
ridge_best = Ridge(alpha=0.1)
ridge_best.fit(X_train,y_train)

print("Intercept:",ridge_best.intercept_)
print("coefficients:",ridge_best.coef_)

#predictions
train_predictions = ridge_best.predict(X_train)
test_predictions = ridge_best.predict(X_test)

#Evaluation
print("Train R2:",ridge_best.score(X_train,y_train))
print("Test R2:",ridge_best.score(X_test,y_test))
print("Cross Validation Score:",cross_val_score(ridge_best,X,y,cv=5).mean())
```

```
Intercept: -11456.794876086758
coefficients: [  256.8498504    -44.63820232   305.23708037   441.52233136
 23614.23737882]
Train R2: 0.7433160460644062
Test R2: 0.7757479380251502
Cross Validation Score: 0.7466599257222319
```

# Prediction on New Data

#### Data

In [17]:

```
input_data ={"age":31,
             "sex":"female",
             "bmi":25.74,
             "children":0,
             "smoker":"no",
             "region":"northeast"}
```

#### preprocessing the data

In [18]:

```
df_test = pd.DataFrame(input_data,index=[0])

df_test.drop('region',axis=1, inplace=True)
df_test['sex'].replace({'female':0,'male':1}, inplace=True)
df_test['smoker'].replace({'no':0,'yes':1}, inplace=True)

transormed_data = df_test
```

#### predict

In [19]:

```
ridge_best.predict(transormed_data)
```

Out[19]:

```
array([4362.35293501])
```