Estimating Healthcare Insurance Expenses through Machine Learning.

1. Analyzing Insurance Costs

In [5]:
```
#--- Import Pandas ---
import pandas as pd
#--- Read in dataset ----
df = pd.read_csv('insurance.csv')

# ---WRITE YOUR CODE FOR TASK 1 ---
df.info()
df.describe()
df.shape
df.head(5)
df.tail(5)
```

```
<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
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

2. Unearthing Data Duplications

In [6]:
```
# --- WRITE YOUR CODE FOR TASK 2 ---
df.duplicated().sum()
duplicates = df.duplicated().sum()

#--- Inspect data ---

duplicates
```

Out[6]:
```
1
```

3. Eliminating data duplications

In [7]:
```
# --- WRITE YOUR CODE FOR TASK 3 ---
df.drop_duplicates(inplace=True)

#--- Inspect data ---
df
```

Out[7]:

	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

1337 rows × 7 columns

4. Check for null values

In [8]:
```
df.isnull()
```

Out[8]:

	age	sex	bmi	children	smoker	region	charges
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False

1337 rows × 7 columns

In [9]:
```
# --- WRITE YOUR CODE FOR TASK 4 ---
df.isnull()
null_values = df.isnull().sum()

#--- Inspect data ---
null_values
```

Out[9]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

5. Empowering Analysis with Encoded Insights.

In [10]:
```
# LabelEncoder is required to encode categorical variables to numerical variables before applying machine learning algorithm

from sklearn.preprocessing import LabelEncoder

# creating instance for the class LabelEncoder
lab_encode=LabelEncoder()

columns_to_encode=["sex","smoker"]
for column in columns_to_encode:
    df[column]=lab_encode.fit_transform(df[column])
```

In [11]:
```
df.head(5)
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520

6. Unleashing the Power of One-Hot Encoding.

In [12]:
```
#One hot encoding -categorical variable into binary representation
one_hot_encode = pd.get_dummies(df['region'])
```

In [13]:
```
df.head(5)
```

Out[13]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520

In [14]:
```
#one-hot encoding can lead to an increase in the dimensionality of the dataset,
one_hot_encode
```

Out[14]:

	northeast	northwest	southeast	southwest
0	0	0	0	1
1	0	0	1	0
2	0	0	1	0
3	0	1	0	0
4	0	1	0	0
...	...	...	...	...
1333	0	1	0	0
1334	1	0	0	0
1335	0	0	1	0
1336	0	0	0	1
1337	0	1	0	0

1337 rows × 4 columns

7. Concatenating a DataFrame with One-Hot Encoded Columns.

In [15]:
```
df1=pd.concat([df,one_hot_encode],axis=1)
```

In [16]:
```
df1.head(5)
```

Out[16]:

	age	sex	bmi	children	smoker	region	charges	northeast	northwest	southeast	southwest
0	19	0	27.900	0	1	southwest	16884.92400	0	0	0	1
1	18	1	33.770	1	0	southeast	1725.55230	0	0	1	0
2	28	1	33.000	3	0	southeast	4449.46200	0	0	1	0
3	33	1	22.705	0	0	northwest	21984.47061	0	1	0	0
4	32	1	28.880	0	0	northwest	3866.85520	0	1	0	0

8. Dropping region column to remove redundancy

In [17]:
```
df1.drop("region", axis=1,inplace=True)
```

In [18]:
```
df1.head(5)
```

Out[18]:

	age	sex	bmi	children	smoker	charges	northeast	northwest	southeast	southwest
0	19	0	27.900	0	1	16884.92400	0	0	0	1
1	18	1	33.770	1	0	1725.55230	0	0	1	0
2	28	1	33.000	3	0	4449.46200	0	0	1	0
3	33	1	22.705	0	0	21984.47061	0	1	0	0
4	32	1	28.880	0	0	3866.85520	0	1	0	0

9. Splitting a DataFrame into Train and Test Sets for Machine Learning.

In [19]:
```
from sklearn.model_selection import train_test_split
X=df1.drop("charges",axis=1)
Y=df1["charges"]
X_train, X_test, Y_train, Y_test=train_test_split(X,Y,test_size=0.2, random_state=42)
```

In [20]:
```
X_train
```

Out[20]:

	age	sex	bmi	children	smoker	northeast	northwest	southeast	southwest
1114	23	1	24.510	0	0	1	0	0	0
968	21	1	25.745	2	0	1	0	0	0
599	52	0	37.525	2	0	0	1	0	0
170	63	1	41.470	0	0	0	0	1	0
275	47	0	26.600	2	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
1096	51	0	34.960	2	1	1	0	0	0
1131	27	1	45.900	2	0	0	0	0	1
1295	20	1	22.000	1	0	0	0	0	1
861	38	0	28.000	3	0	0	0	0	1
1127	35	0	35.860	2	0	0	0	1	0

1069 rows × 9 columns

In [21]:
```
X_test
```

Out[21]:

	age	sex	bmi	children	smoker	northeast	northwest	southeast	southwest
900	49	1	22.515	0	0	1	0	0	0
1064	29	0	25.600	4	0	0	0	0	1
1256	51	0	36.385	3	0	0	1	0	0
298	31	1	34.390	3	1	0	1	0	0
237	31	1	38.390	2	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...
534	64	1	40.480	0	0	0	0	1	0
542	63	0	36.300	0	0	0	0	1	0
760	22	0	34.580	2	0	1	0	0	0
1284	61	1	36.300	1	1	0	0	0	1
1285	47	0	24.320	0	0	1	0	0	0

268 rows × 9 columns

In [22]:
```
Y_train
```

Out[22]:
```
1114     2396.09590
968      3279.86855
599     33471.97189
170     13405.39030
275      9715.84100
           ...
1096    44641.19740
1131     3693.42800
1295     1964.78000
861      7151.09200
1127     5836.52040
Name: charges, Length: 1069, dtype: float64
```

In [23]:
```
Y_test
```

Out[23]:
```
900      8688.85885
1064     5708.86700
1256    11436.73815
298     38746.35510
237      4463.20510
           ...
534     13831.11520
542     13887.20400
760      3925.75820
1284    47403.88000
1285     8534.67180
Name: charges, Length: 268, dtype: float64
```

10. Evaluating Model Performance

In [30]:
```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from math import sqrt

# Create an instance of the RandomForestRegressor class with the specified configurations
# n_estimators means the random forest will consist of 50 decision trees
# n_jobs=2 means two CPU cores utilized for parallel processing
# random_state=42 for attaining reproduciblity
rand_forest_model = RandomForestRegressor(n_estimators=50, n_jobs=2, random_state=42)

# Assuming you have X_train and Y_train as your training feature data and target values
# Use cross_val_score to evaluate the performance of the rand_forest_model
# higher negative values indicate better model performance in terms of mean squared error.
scores = cross_val_score(rand_forest_model, X_train, Y_train, scoring="neg_mean_squared_error", cv=10)

# Calculate RMSE (Root Mean Squared Error) values from the negative mean squared error scores
rmse_values = np.sqrt(-scores)

# Compute the standard deviation of RMSE values
std = np.std(rmse_values)

# Now 'rmse_values' contains the RMSE values for each fold of the cross-validation, and 'std' is the standard deviation.
```

In [24]:
```
scores
```

Out[24]:
```
array([-27424665.56591485, -31250318.70760591, -18286984.03659446,
       -29067518.06087001, -29830940.94321635, -15049369.3514271 ,
       -28294661.06064019, -17588383.180689  , -28925332.68580534,
       -23812680.15325704])
```

In [25]:
```
std
```

Out[25]:
```
586.8826484248
```

11. Forecasting Healthcare Costs

In [32]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

# Assuming you have X_train, y_train, X_test, and y_test as your training and testing datasets

# Create an instance of the RandomForestRegressor class with your desired configurations
rand_forest_model = RandomForestRegressor(n_estimators=50, n_jobs=2, random_state=42)

# Train the model using the training dataset
rand_forest_model.fit(X_train, Y_train)

# Make predictions on the X_test dataset
predictions = rand_forest_model.predict(X_test)

# Round the predictions to two decimal places
rounded_predictions = np.round(predictions, 2)

# Ensure y_test is a NumPy array
Y_test_values = Y_test.values

# Round the actual charges to two decimal places
rounded_Y_test = np.round(Y_test_values, 2)

# Create a DataFrame for comparison
compare_data = {'Actual Charges': rounded_Y_test[:10], 'Predicted Charges': rounded_predictions[:10]}
compare = pd.DataFrame(compare_data)

# Display the comparison DataFrame
print(compare)
```

```
   Actual Charges  Predicted Charges
0         8688.86           10334.48
1         5708.87            6651.49
2        11436.74           12295.40
3        38746.36           42608.01
4         4463.21            6853.03
5         9304.70            9366.65
6        38511.63           39526.20
7         2150.47            2194.72
8         7345.73            8516.75
9        10264.44           10708.99
```

In [ ]: