Which Factors Influence the Price of Health Insurance?

You work as a Data Scientist for the New York Times. They are doing a story titled, "Which Factors Influence the Price of Health Insurance?". Many factors that affect how much you pay for health insurance are not within your control. Nonetheless, it's good to have an understanding of what they are. Hence, you have collected data about individuals for the new story. Your data contains basic factors about the

Overall, you want to find how well the factors you collected can predict the individual insurance costs billed by the health insurance company.

The first step of any machine learning project is to understand your data. The dataset contains the variables/features below. You should look at each feature to understand what type of feature it is, i.e., is it a categorical feature, binary, numeric, etc?

This is what you will predict:

charges: Individual medical costs billed by health insurance What type of item are you predicting? (e.g., categorical, numeric, binary, ordinal, etc.)

In [3]:

```
# NOTES

# What is Machine Learning? f(x) = y

# Features and Scikit-Learn

## Methods of construction features

### DictVectorizer (Tabular Data)

### CountVectorizer (Text Data)

```

In [4]:
```
import csv
from sklearn.feature_extraction import DictVectorizer
from sklearn.model_selection import train_test_split

dataset = [] # Features go here
y = [] # What you want to predict goes here
with open('insurance.csv') as iFile:
    iCSV = csv.reader(iFile, delimiter=',')
    header = next(iCSV)
    for row in iCSV:
        item = {}
        item['age'] = float(row[0])
        item['sex'] = row[1]
        item['bmi'] = float(row[2])
        item['children'] = float(row[3])
        item['smoker'] = row[4]
        item['region'] = (row[5])
        dataset.append(item)
        y.append(float(row[6]))

# Here the lists are split into a 80% training portion and a 20% validatic
train_dataset, val_dataset, train_y, val_y = train_test_split(dataset, y,

# DictVectorizer will take a list of dictionaries and convert it into a ma
vec = DictVectorizer()

train_matrix = vec.fit_transform(train_dataset) # .fit_transform() should
val_matrix = vec.transform(val_dataset) # .transform() should only be apli
```

In [5]:
```
# Here is what one dictionary looks like in the dataset
dataset[0]
```

Out[5]:
```
{'age': 19.0,
 'sex': 'female',
 'bmi': 27.9,
 'children': 0.0,
 'smoker': 'yes',
 'region': 'southwest'}
```

In [6]:
```
print(vec.feature_names_) # This list contains the feature names for ea
```

```
['age', 'bmi', 'children', 'region=northeast', 'region=northwest', 'regio
n=southeast', 'region=southwest', 'sex=female', 'sex=male', 'smoker=no',
'smoker=yes']
```

In [7]:
```
print(train_matrix.toarray()[0:2,:]) # This prints the first two rows c
```

```
[[46.    19.95   2.     0.     1.     0.     0.     1.     0.     1.     0.   ]
 [47.    24.32   0.     1.     0.     0.     0.     1.     0.     1.     0.   ]]
```

In [9]:
```python
import numpy as np
# Modify the weights below to achieve the LOWEST MSE as possible.
weights = np.array([0., # 'age'
                    0., # 'bmi'
                    0., # 'children'
                    0., # 'region=northeast'
                    0., # 'region=northwest'
                    0., # 'region=southeast'
                    0., # 'region=southwest'
                    0., # 'sex=female'
                    0., # 'sex=male',
                    0., # 'smoker=no'
                    0.]) # 'smoker=yes'
```

In [10]:
```python
from sklearn.metrics import mean_squared_error
predictions = val_matrix.dot(weights)

print("MSE:", mean_squared_error(val_y, predictions))
```

```
MSE: 323425978.93488324
```

In [11]:
```python
# Finding the weights manually is HARD.
# Here we will use scikit-learn to find the weights for us. We are trai
# Run the cells below to see how well it performs and what weights it f
from sklearn.linear_model import LinearRegression

clf = LinearRegression(fit_intercept=False)

clf.fit(train_matrix, train_y)

lr_predictions = clf.predict(val_matrix)
print("MSE:", mean_squared_error(val_y, lr_predictions))
```

```
MSE: 33596625.74155255
```

In [12]:
```python
print("LR Weights:\n", "\n".join([f"{x:.5f}" for x in clf.coef_]))
```

```
LR Weights:
 257.01791
337.34778
425.47128
342.99371
-27.60490
-315.85397
-467.22874
-224.54077
-243.15313
-12059.18785
11591.49395
```

*****Note that a lower MSE means the model is better (i.e., your prediction is closer to the real amount), How

You suspect that the Linear Regression model is overfitting (e.g., weights are too large). Moreover, you think other models may provide better predictive ability. Choosing a model is only half the battle. In scikit-learn trying a different model is as simple as importing and using a different classifier object. But, to get the best performance from any given model, you need to tune all of the "knobs" availble within the model. You can find all the "knobs" a model contains by looking at the scikit learn documentation under the section "Parameters"

[https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

You are going to experiment with the "alpha" parameter. The alpha parameter will control overfitting in the Ridge Regression model. Try the values 0, .00001, .0001, .001, .01, .1, 1, 10, 100. Which value gives the best results. You should try this manually, by setting the alpha value to the different values and seeing the results. What do you find? Specifically, how does the alpha value impact the parameters? Also, what alpha gives the best performance?

In [13]:

```
# Try the different alphas in this cell
from sklearn.linear_model import Ridge

clf = Ridge(alpha=0, fit_intercept=False)

clf.fit(train_matrix, train_y)

rr_predictions = clf.predict(val_matrix)
print("MSE:", mean_squared_error(val_y, rr_predictions))
print("\nRidge Weights:\n", "\n".join([f"{x:.5f}" for x in clf.coef_]))
```

```
MSE: 35516055.0927728

Ridge Weights:
 262.40381
331.86322
325.48535
173.70912
-474.11913
469.62882
-707.97476
-912.67017
373.91421
-11921.84742
11383.09146
```

Trying every hyper parameter one-by-one is time consuming. Instead, let us do it using GridSearchCV. Note that GridSearchCV will use cross-validation applied to the training dataset. So, the exact results may be different that what was found in the previous cell. Why is that?

In [14]:

```
from sklearn.model_selection import GridSearchCV

params = {"alpha": [0, .00001, .0001, .001, .01, .1, 1, 10, 100, 1000]}

rr = Ridge(fit_intercept=False)

clf = GridSearchCV(rr, param_grid=params, cv=10, scoring='neg_mean_squa

clf.fit(train_matrix, train_y)

rr_predictions = clf.predict(val_matrix)
print("MSE:", mean_squared_error(val_y, rr_predictions))
print("Best Alpha:", clf.best_params_['alpha'])
print("Cross-Validation Score:", -clf.best_score_)
print("\nRidge Weights:\n", "\n".join([f"{x:.5f}" for x in clf.best_est
```

```
MSE: 35516054.80993609
Best Alpha: 1e-05
Cross-Validation Score: 38399017.57406919

Ridge Weights:
 262.40380
331.86321
325.48535
173.70911
-474.11912
469.62880
-707.97474
-912.67014
373.91420
-11921.84709
11383.09115
```

Final Notes

```

So, where do you go from here? You can try different models such as:

Lasso: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
ElasticNet: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html
Random Forest Regression: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
SVM Regression: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
```

Each method has its own hyperparameters you must test to find what works best. Another option is to explore the use of "Feature Engineering". How will the model perform if you remove one or more features (columns)? What if you transform columns in some non-trival way, e.g., square the values in a column (e.g., age = age 2) or combine values via interaction terms (e.g., age*gender=Male). Overall, the combintions are endless. If you had access to the data at a specific company, then you could also try to collect more specific data, e.g., income, family history, etc. In the end, this is a creative endevor as much as it is a technical one.

In [ ]:

```

```