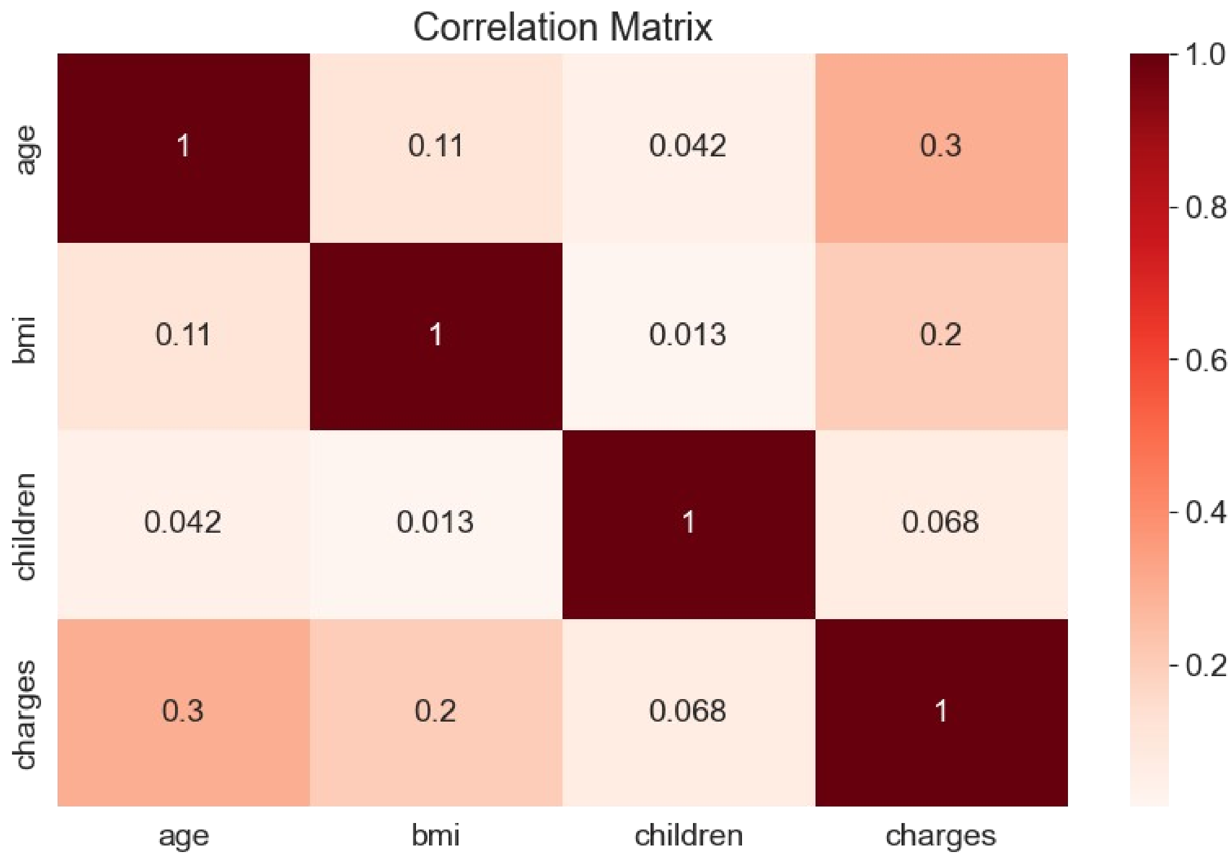


Note that a high correlation between two variables does not necessarily indicate a cause-effect relationship between features. They could potentially be impacted by another factor. It's therefore to understand the underlying data to determine if indeed there is a correlation.

Linear Regression using a Single Feature

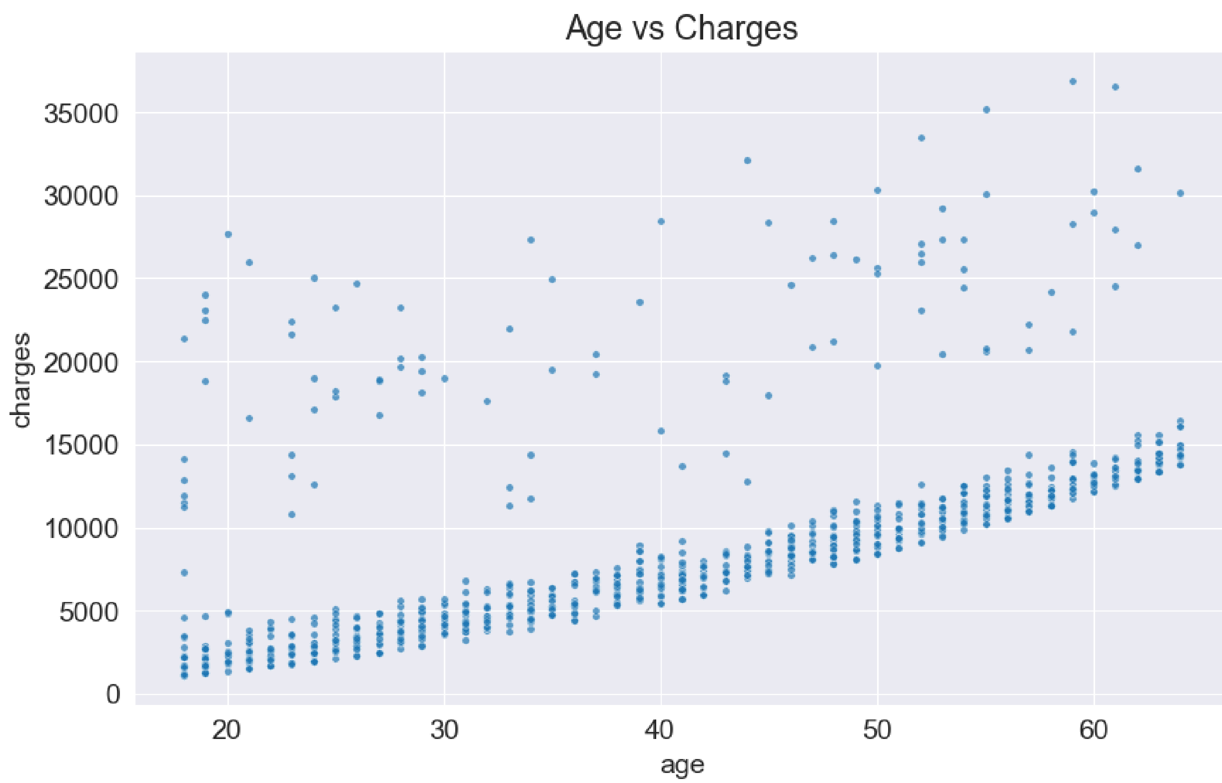
We have noted that "age" and "smoker" have a significant correlation with "charges".

We can try to estimate the value in "charges" using the value of "age" for non-smokers.

Let's first create a Pandas dataframe for the data on non-smokers

```
non_smoker_df = medical_df[medical_df.smoker == 'no']

plt.title('Age vs Charges')
sns.scatterplot(data=non_smoker_df, x='age', y='charges', alpha=0.7,
s=15)
```

```
<Axes: title={'center': 'Age vs Charges'}, xlabel='age',
ylabel='charges'>
```



Apart from some exceptions, the points seem to form a line. We will try to fit a line using the above points and use this line to predict the "charges" based on "age".

This line will have the formula: y = mx + b

- m = slope
- b = intercept (value of y when x=0)

Model

We assume that the relationship between the "charges" and "age can be represented as below: charges = m * age + b

- This is a **linear regression model** because it models the relationship between the "age" and "charges" as a straight line
- The values *m* and *b* are referred to as the **parameters/weights** of the model
- The "age" values are the **inputs** to the model and the values in the "charges" are the **targets**

We can create a helper function to calculate the charges given input(age) and parameters (m and b)

```
def estimate_charges(age, m, b):
    return m * age + b
```

The estimate_charges function is our very first model

Lets assign some random values to m and b

```
m = 50
b = 100

ages = non_smoker_df.age
ages
```

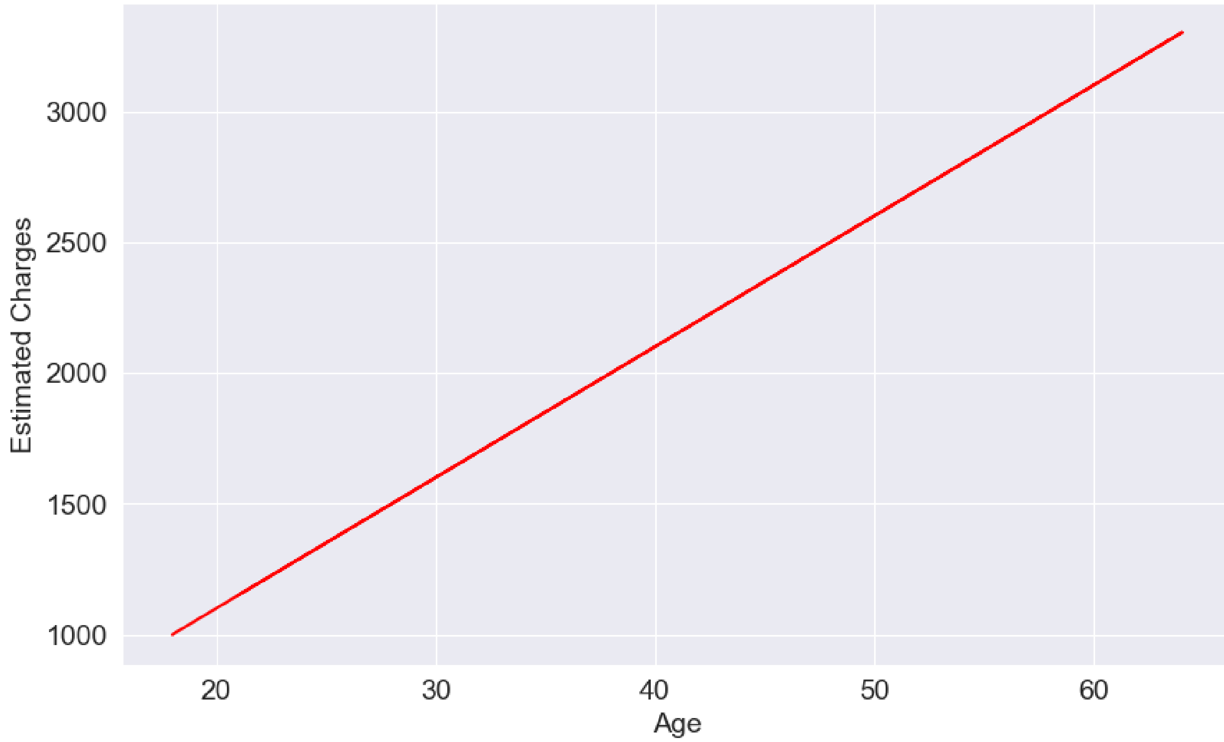
```
1       18
2       28
3       33
4       32
5       31
        ..
1332    52
1333    50
1334    18
1335    18
1336    21
Name: age, Length: 1064, dtype: int64
```

```
estimated_charges = estimate_charges(ages, m, b)
estimated_charges
```

```
1       1000
2       1500
3       1750
4       1700
5       1650
        ...
1332    2700
1333    2600
1334    1000
1335    1000
1336    1150
Name: age, Length: 1064, dtype: int64
```

Lets plot the estimated charges using a line graph

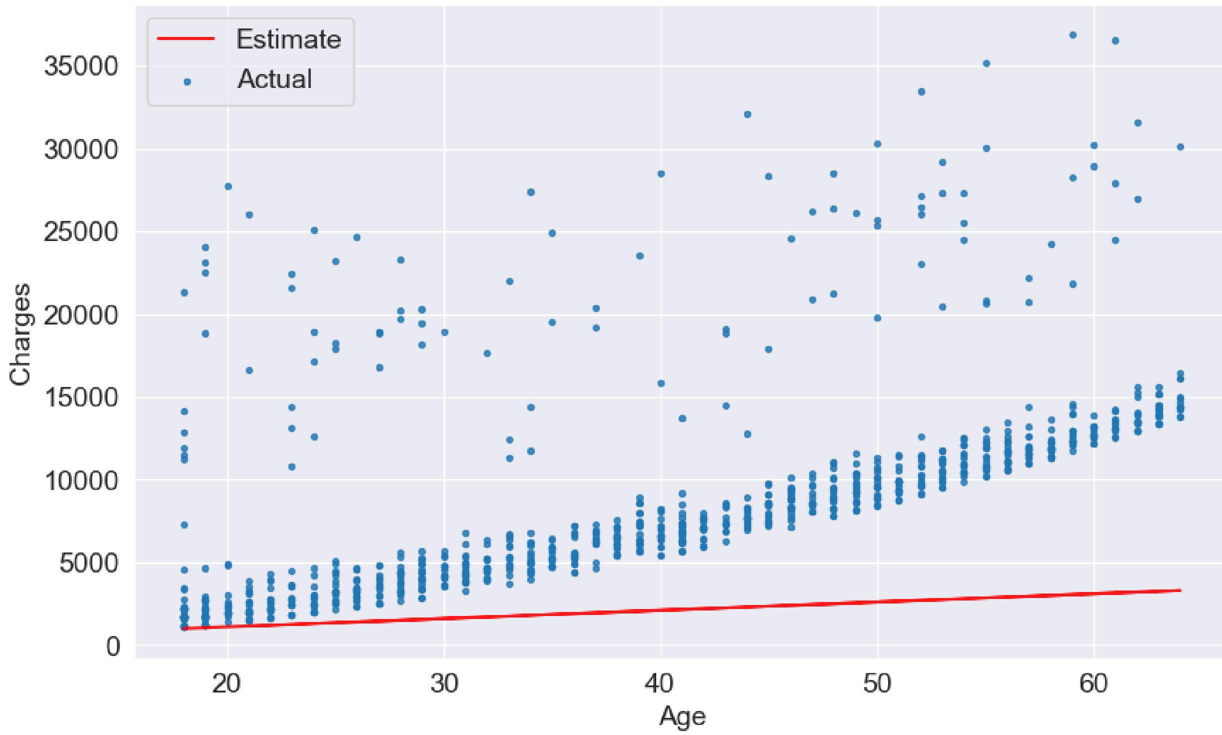
```
plt.plot(ages, estimated_charges, 'r-');
plt.xlabel('Age');
plt.ylabel('Estimated Charges');
```



We can fit this line on our data to see how well our model fits the data

```
target = non_smoker_df.charges

plt.plot(ages, estimated_charges, 'r', alpha=0.9);
plt.scatter(ages, target, s=8, alpha=0.8);
plt.xlabel('Age');
plt.ylabel('Charges');
plt.legend(['Estimate', 'Actual'])
```

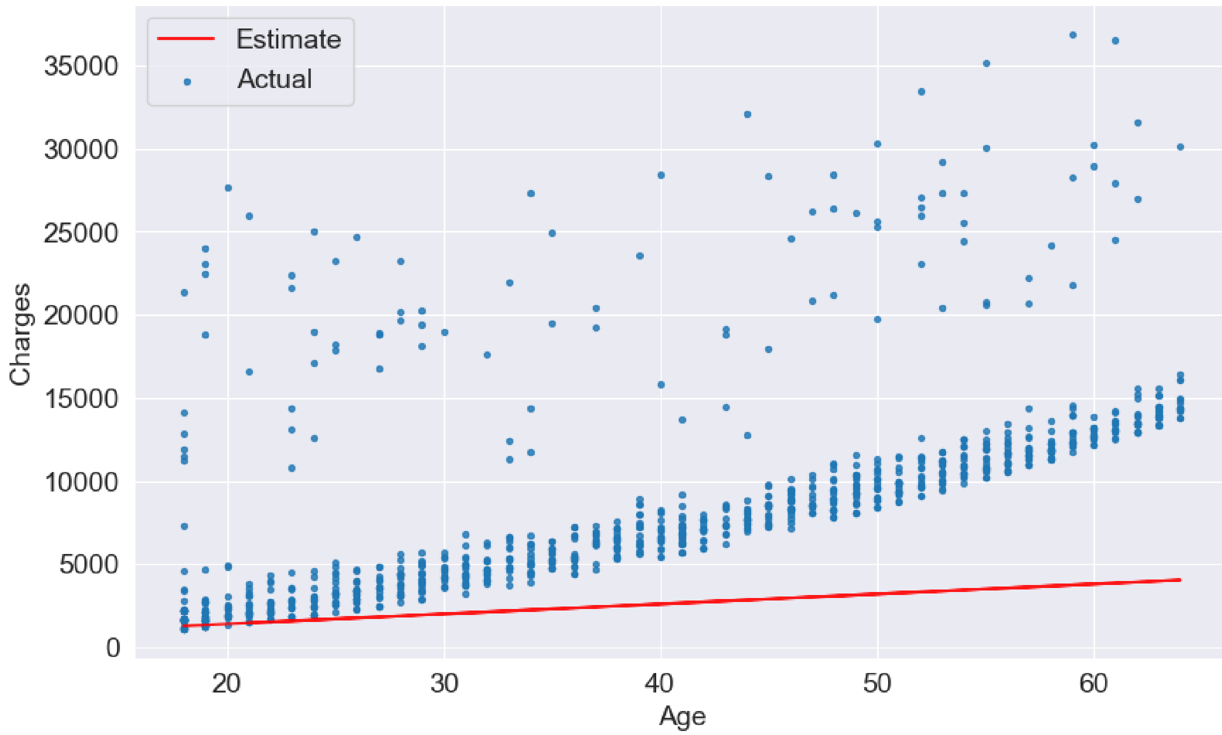
```
<matplotlib.legend.Legend at 0x1af87997a90>
```



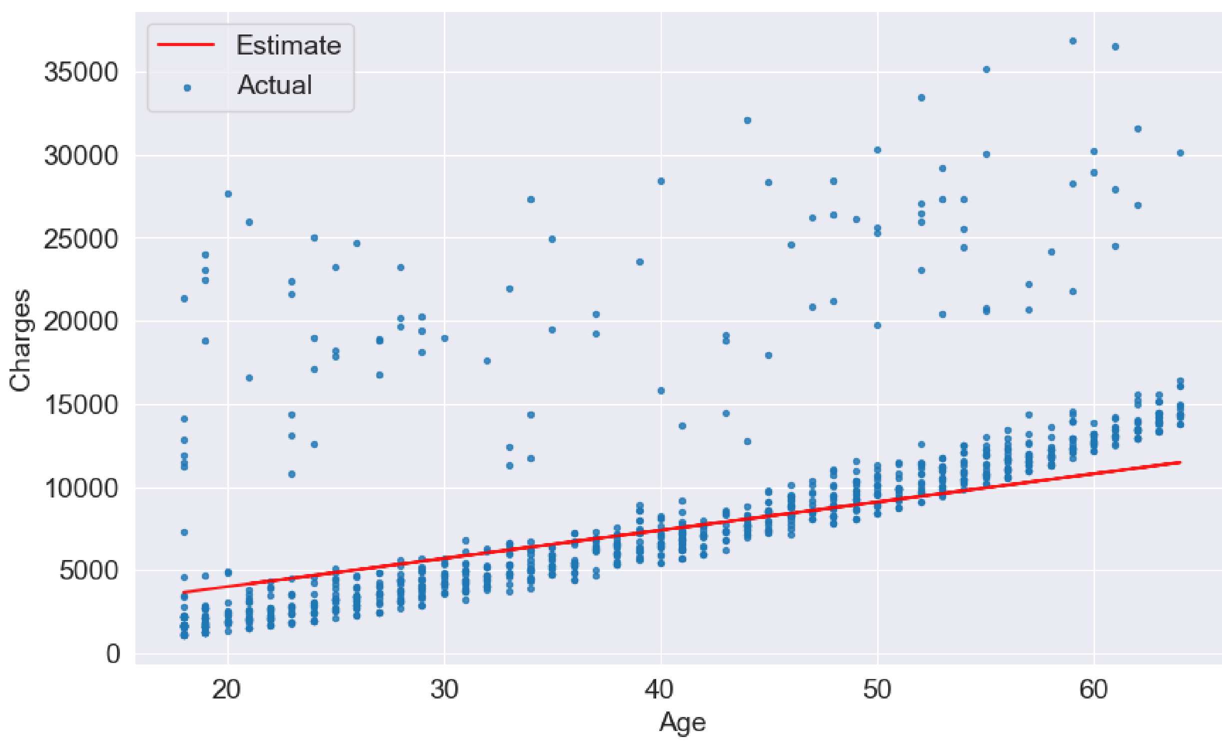
From the above plot, our model is clearly very inaccurate and does not fit the actual data. We can try different values of **m** and **b** to move the line around to get a better fit.

Lets define a function that takes **m** and **b** as inputs and creates a plot

```
def try_parameters(m, b):
    ages = non_smoker_df.age
    target = non_smoker_df.charges

    estimated_charges = estimate_charges(ages, m, b)

    plt.plot(ages, estimated_charges, 'r', alpha=0.9);
    plt.scatter(ages, target, s=8, alpha=0.8);
    plt.xlabel('Age');
    plt.ylabel('Charges')
    plt.legend(['Estimate', 'Actual']);


try_parameters(60, 200)
```




```
try_parameters(170, 600)
```



From the plots above, we note that a change in the parameters leads to a change in the slope of the line

In the above functions, we are manually trying different values of the slope(m) and intercept(b) and approximating the relationship between the "age" and "charges" columns

It would be nice if a computer could help us figure out the values of m and b and learn the relationships between the different features

To do this we need to:

- Measure numerically how well the line fits the points
- Modify m and b to improve fit

```
targets = non_smoker_df.charges
targets
```

```
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
5        3756.62160
            ...
1332    11411.68500
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
Name: charges, Length: 1064, dtype: float64
```

```
predictions = estimated_charges
predictions
```

```
1       1000
2       1500
3       1750
4       1700
5       1650
        ...
1332    2700
1333    2600
1334    1000
1335    1000
1336    1150
Name: age, Length: 1064, dtype: int64
```

Loss Function

We can compare the predictions of our model with the actual targets using the following method:

- Calculating the **residual** : the difference between the targets and the predictions
- **Squaring** the residuals to remove negative values
- Calculate the **average** of the elements in the resulting matrix

- Finding the square root of the avg

The result of the above steps will give us the **Root Mean Squared Error(RMSE)**

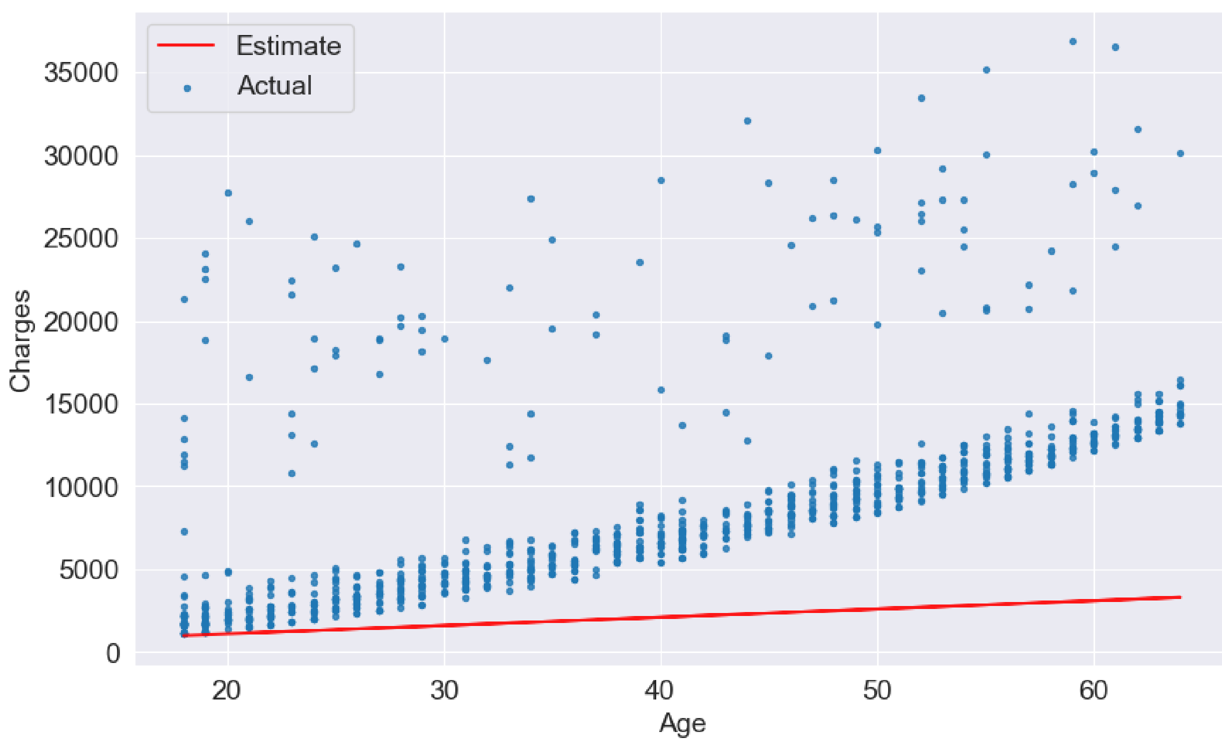
The RMSE can be respresented mathematically as follows:

The RMSE can be geometrically visualized as below:

```
def rmse(targets,  predictions):
    return np.sqrt(np.mean(np.square(targets - predictions)))
```

Lets try to calculate the RMSE using sample parameter values

```
m = 50
b = 100

try_parameters(m, b)
```




```
targets = non_smoker_df['charges']
predicted = estimate_charges(non_smoker_df.age, m, b)

rmse(targets, predicted)

8461.949562575493
```

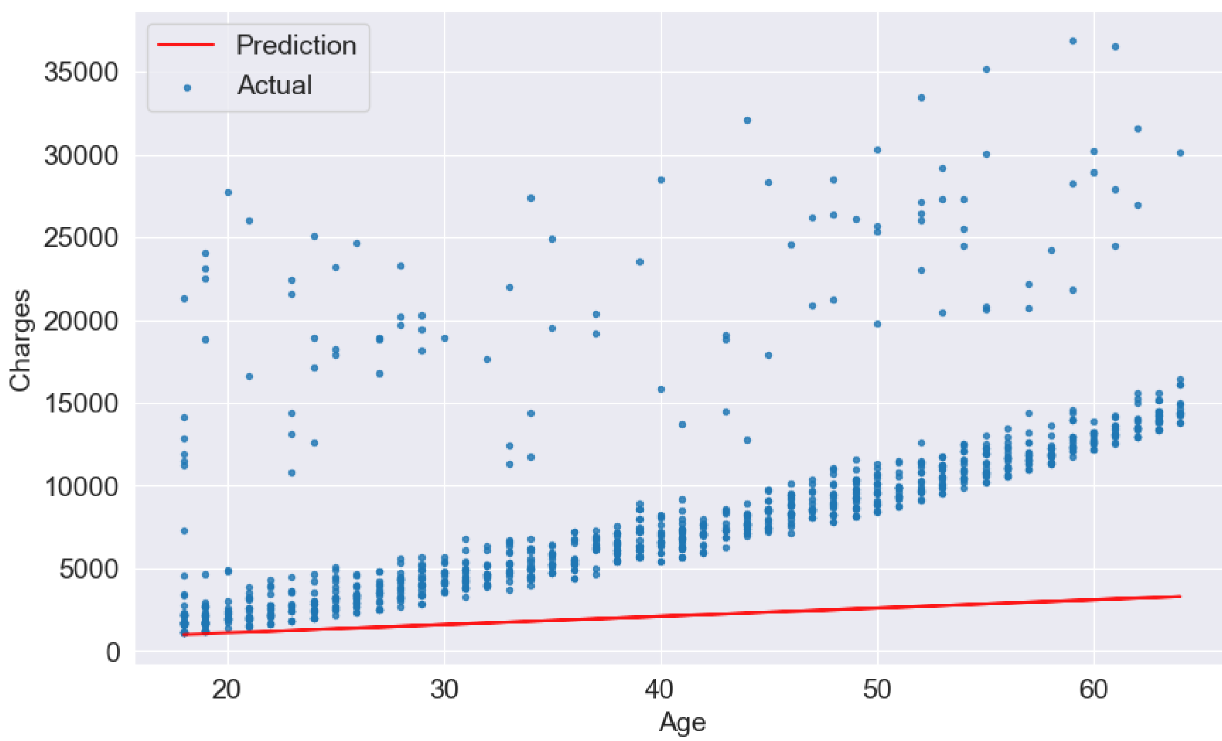
We can interpret the RMSE as follows: On average, each element in the prediction differs from the actual target by $8461

The result is called the *loss* because it indicates how bad the model is at predicting the target variables. It represents information loss in the model: the lower the loss, the better the model

Let's modify the try_parameters functions to also display the loss

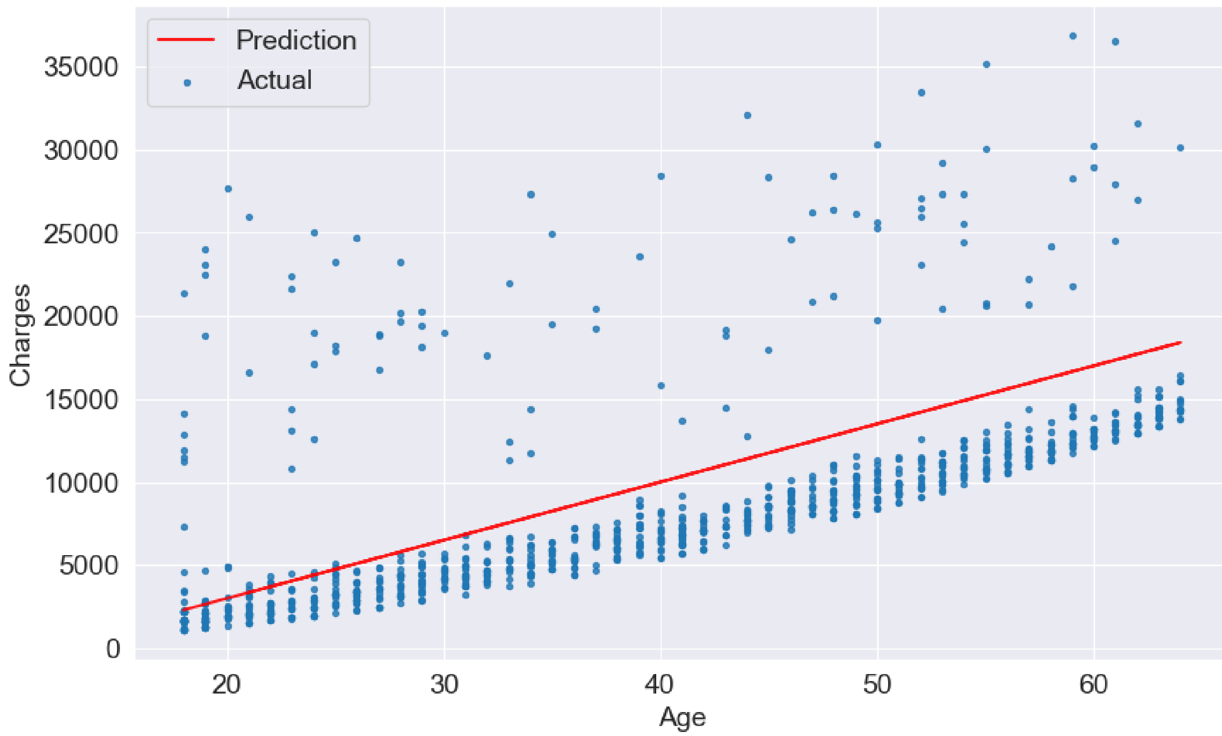
```
def try_parameters(m, b):
    ages = non_smoker_df.age
    target = non_smoker_df.charges
    predictions = estimate_charges(ages, m, b)

    plt.plot(ages, predictions, 'r', alpha=0.9);
    plt.scatter(ages, target, s=8, alpha=0.8);
    plt.xlabel('Age');
    plt.ylabel('Charges');
    plt.legend(['Prediction', 'Actual']);

    loss = rmse(target, predictions)
    print("RMSE Loss: ", loss)


try_parameters(50, 100)
```

```
RMSE Loss:  8461.949562575493
```



```
try_parameters(350, -4000)
```

```
RMSE Loss:  4991.993804156943
```



Optimizer

We need a way to adjust the parameters `m` and `b` to reduce the loss and improve the "fit" of the line

We hve two methods for adjusting the parameters:

- Ordinary Least Squares method: Uses matrix operations to compute the best parameter values (combines calculus & linear algebra)
- Stochastic Gradient Descent: Uses an iterative approach to calculate the parameters by slowly improving them using derivatives starting from a random value

Linear Regression using SciKit-Learn

Rather than implementing a linear regression model by ourselves, we can use the `scikit-learn` library

We will use the `LinearRegression` class from `sklearn` to find the line of best fit between "age" and "charges" using the *Ordinary Least Squares Method* for optimization

First we create a new **model object**

```
model = LinearRegression()
```

We can use the `fit` method of the model to find the line of best fit for the input and targets

```
help(model.fit)
```

```
Help on method fit in module sklearn.linear_model._base:

fit(X, y, sample_weight=None) method of
sklearn.linear_model._base.LinearRegression instance
    Fit linear model.

    Parameters
    ----------
    X : {array-like, sparse matrix} of shape (n_samples, n_features)
        Training data.

    y : array-like of shape (n_samples,) or (n_samples, n_targets)
        Target values. Will be cast to X's dtype if necessary.

    sample_weight : array-like of shape (n_samples,), default=None
        Individual weights for each sample.

        .. versionadded:: 0.17
           parameter *sample_weight* support to LinearRegression.

    Returns
    -------
    self : object
        Fitted Estimator.
```

NB: X must be a 2D Array, so we pass a dataframe instead of a single column

```
inputs = non_smoker_df[['age']]
targets = non_smoker_df.charges
print('inputs.shape:', inputs.shape)
print('targets.shape:', targets.shape)
```

```
inputs.shape: (1064, 1)
targets.shape: (1064,)
```

```
type(inputs)
```

```
pandas.core.frame.DataFrame
```

Let's fit the model to the data

```
model.fit(inputs, targets)
```

```
LinearRegression()
```

We can make predictions using the created model. Let's try predicting the charges for the ages 23, 37 and 61

```
model.predict(np.array([[23],
                        [37],
                        [61]]))
```

```
C:\Users\admin\miniconda3\envs\alxenv\Lib\site-packages\sklearn\
base.py:464: UserWarning:

X does not have valid feature names, but LinearRegression was fitted
with feature names


array([ 4055.30443855,  7796.78921819, 14210.76312614])
```

The results of the model seem relevant

Lets compute the predictions for the entire set of inputs

```
predictions = model.predict(inputs)
```

```
predictions
```

```
array([2719.0598744 , 5391.54900271, 6727.79356686, ...,
2719.0598744 ,
       2719.0598744 , 3520.80661289])
```

```
targets
```

```
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
5        3756.62160
           ...
1332    11411.68500
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
Name: charges, Length: 1064, dtype: float64
```

Lets compute the RMSE loss to evaluate the model

```
rmse(targets, predictions)
```

```
4662.505766636395
```

From the RMSE, it is clear that our predictions are off by an average of $4000, which is reasonable since we have outliers

The **parameters** of the model are stored in the `coef_` and `intercept_` properties

```
# m
model.coef_
```

```
array([267.24891283])
```

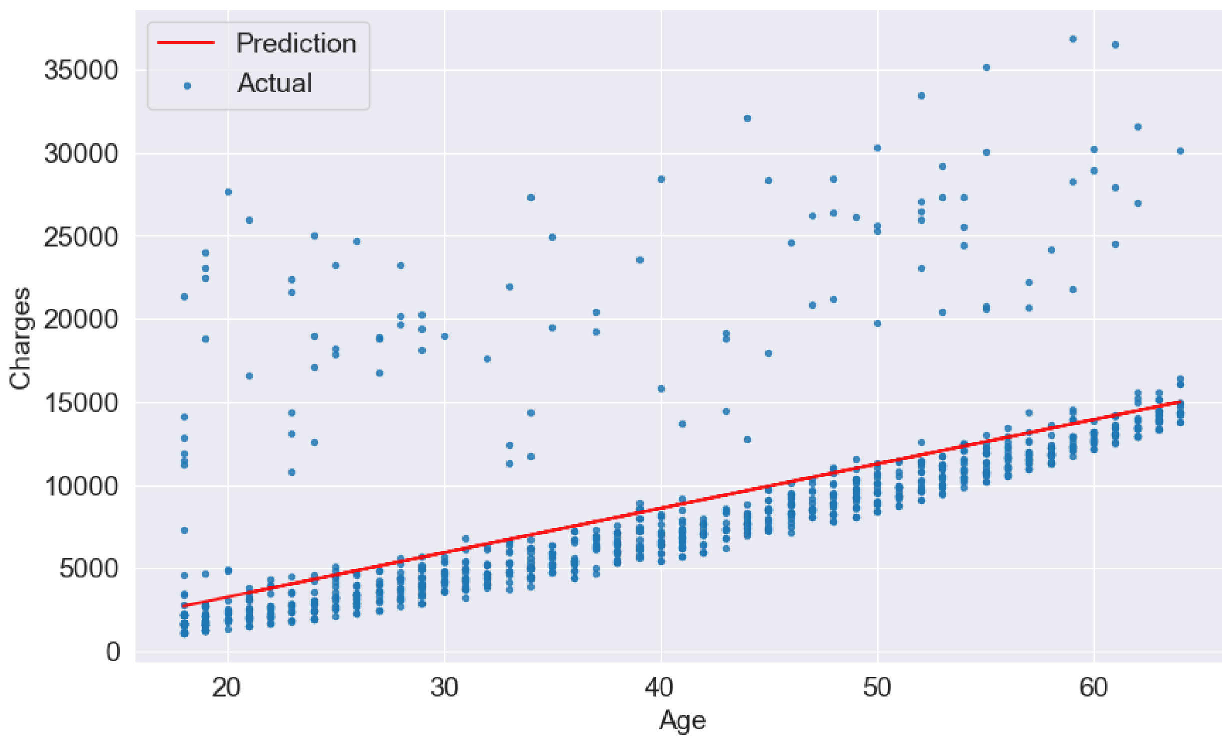
```
# b
model.intercept_
```

```
-2091.4205565650827
```

Lets visualize the line created by the above parameters

```
try_parameters(model.coef_, model.intercept_)
```

```
RMSE Loss:  4662.505766636395
```



The line is close to the data points

It is slightly above the cluster of points because it is trying to account for the outliers

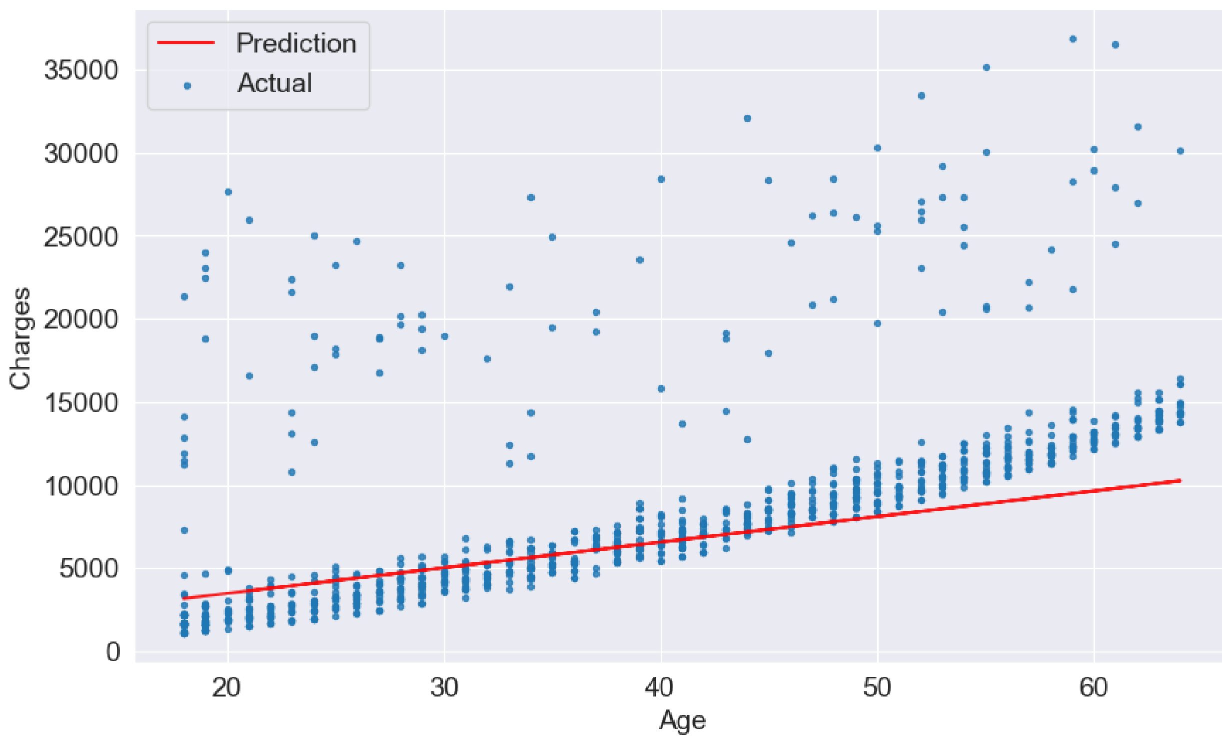
Stochastic Gradient Descent

Lets create a linear regression model that is trained using the stochastic gradient descent technique

We will use the **`SGDRegressor`** class from `SKlearn`

```
model_two = SGDRegressor()

help(model_two.fit)

Help on method fit in module
sklearn.linear_model._stochastic_gradient:

fit(X, y, coef_init=None, intercept_init=None, sample_weight=None)
method of sklearn.linear_model._stochastic_gradient.SGDRegressor
instance
    Fit linear model with Stochastic Gradient Descent.

    Parameters
    ----------
    X : {array-like, sparse matrix}, shape (n_samples, n_features)
        Training data.

    y : ndarray of shape (n_samples,)
        Target values.

    coef_init : ndarray of shape (n_features,), default=None
        The initial coefficients to warm-start the optimization.

    intercept_init : ndarray of shape (1,), default=None
        The initial intercept to warm-start the optimization.

    sample_weight : array-like, shape (n_samples,), default=None
        Weights applied to individual samples (1. for unweighted).

    Returns
    -------
    self : object
        Fitted `SGDRegressor` estimator.


model_two.fit(inputs, targets)

SGDRegressor()

model_two.predict(np.array([[23],
                        [37],
                        [61]]))

C:\Users\admin\miniconda3\envs\alxenv\Lib\site-packages\sklearn\
base.py:464: UserWarning:

X does not have valid feature names, but SGDRegressor was fitted with
feature names


array([3940.2871713 , 6099.21741984, 9800.24070307])
```

```
stoch_predictions = model_two.predict(inputs)

stoch_predictions

array([3169.24065396, 4711.33368863, 5482.38020597, ...,
3169.24065396,
       3169.24065396, 3631.86856436])

rmse(targets, stoch_predictions)

5304.79987665297

# m
model_two.coef_

array([154.20930347])

# b
model_two.intercept_

array([393.47319154])

try_parameters(model_two.coef_, model_two.intercept_)

RMSE Loss:  5304.79987665297
```



Linear Regression Model for Smokers

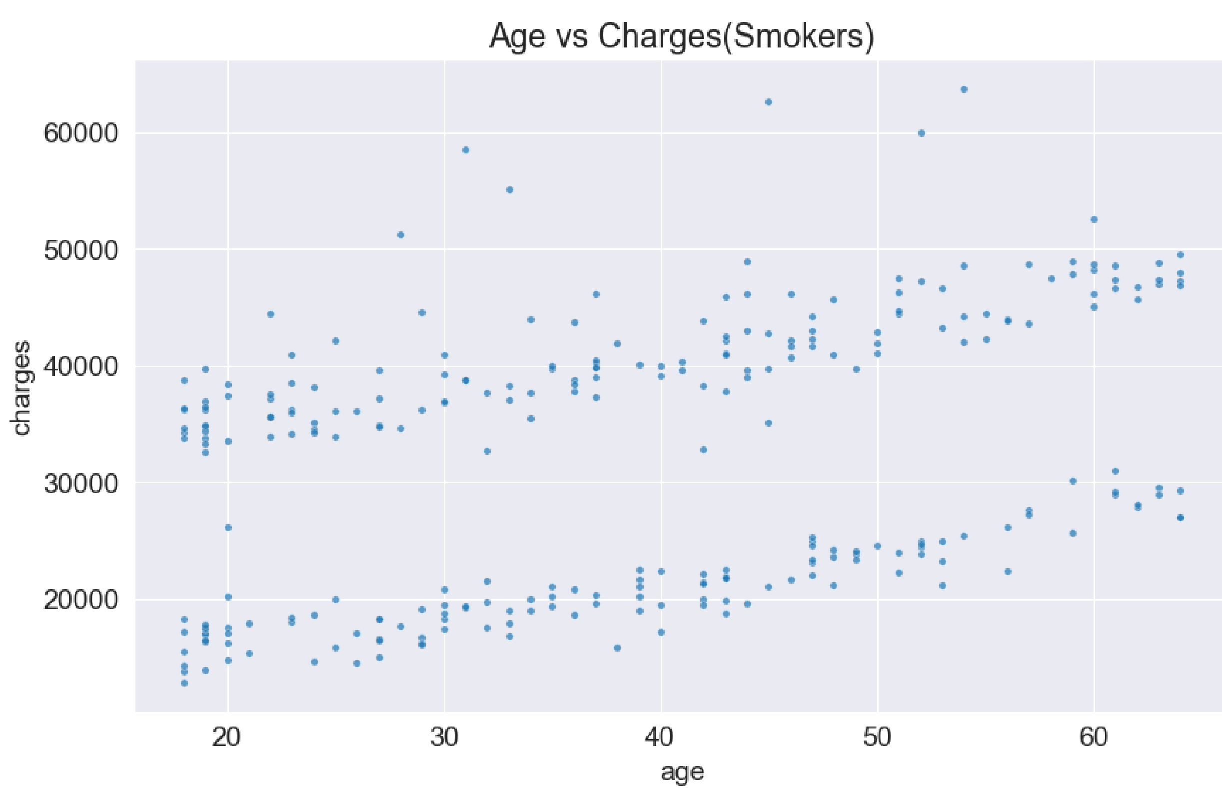
Lets now train a model to estimate the charges for smokers

```
smoker_df = medical_df[medical_df.smoker == 'yes']

smoker_df
```

```
      age     sex     bmi  children smoker     region      charges
0      19  female  27.900         0    yes  southwest  16884.92400
11     62  female  26.290         0    yes  southeast  27808.72510
14     27    male  42.130         0    yes  southeast  39611.75770
19     30    male  35.300         0    yes  southwest  36837.46700
23     34  female  31.920         1    yes  northeast  37701.87680
...   ...     ...     ...       ...    ...        ...          ...
1313   19  female  34.700         2    yes  southwest  36397.57600
1314   30  female  23.655         3    yes  northwest  18765.87545
1321   62    male  26.695         0    yes  northeast  28101.33305
1323   42  female  40.370         2    yes  southeast  43896.37630
1337   61  female  29.070         0    yes  northwest  29141.36030

[274 rows x 7 columns]
```

Let us visualize the relationship between Age and Charges for Smokers

```
plt.title('Age vs Charges(Smokers)')
sns.scatterplot(data=smoker_df, x='age', y='charges', alpha=0.7, s=15)

<Axes: title={'center': 'Age vs Charges(Smokers)'}, xlabel='age',
ylabel='charges'>
```



```
smoker_ages = smoker_df.age
smoker_ages
```

```
0       19
11      62
14      27
19      30
23      34
        ..
1313    19
1314    30
1321    62
1323    42
1337    61
Name: age, Length: 274, dtype: int64
```

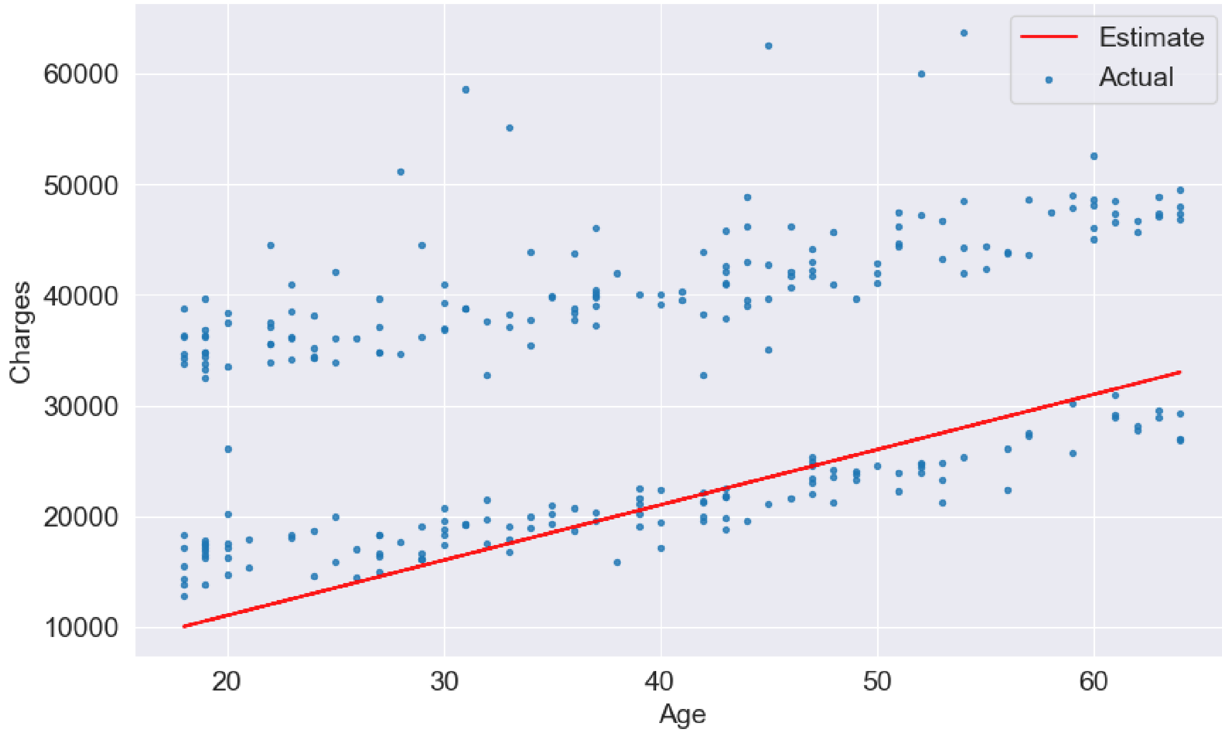
```
smoker_est_charges = estimate_charges(smoker_ages, m, b)
smoker_est_charges
```

```
0       1050
11      3200
14      1450
19      1600
23      1800
        ...
1313    1050
1314    1600
1321    3200
1323    2200
1337    3150
Name: age, Length: 274, dtype: int64
```

Lets create a function to visualize the results for the smokers

```
def try_smoker_parameters(m, b):
    smoker_ages = smoker_df.age
    smoker_target = smoker_df.charges

    estimated_charges = estimate_charges(smoker_ages, m, b)

    plt.plot(smoker_ages, estimated_charges, 'r', alpha=0.9);
    plt.scatter(smoker_ages, smoker_target, s=8, alpha=0.8);
    plt.xlabel('Age');
    plt.ylabel('Charges')
    plt.legend(['Estimate', 'Actual']);

    loss = rmse(smoker_target, estimated_charges)
    print("RMSE Loss: ", loss)

try_smoker_parameters(500, 1000)
```

```
RMSE Loss:  16159.375031795793
```




```
smoker_inputs = smoker_df[['age']]
smoker_targets = smoker_df.charges
print('smoker inputs.shape:', smoker_inputs.shape)
print('smoker targets.shape:', smoker_targets.shape)
```

```
smoker inputs.shape: (274, 1)
smoker targets.shape: (274,)
```

```
smoker_model = model.fit(smoker_inputs, smoker_targets)
```

```
smoker_model
```

```
LinearRegression()
```

```
smoker_predictions = smoker_model.predict(smoker_inputs)
```

```
smoker_predictions
```

```
array([26093.642567  , 39218.85945773, 28535.54338388, 29451.25619021,
       30672.20659865, 29756.49379232, 27009.35537333, 28840.78098599,
       30977.44420076, 38608.38425351, 31282.68180287, 34945.53302819,
       31282.68180287, 37997.90904929, 25788.40496489, 36471.72103874,
       26398.88016911, 28840.78098599, 28535.54338388, 27009.35537333,
       31587.91940498, 34029.82022186, 37692.67144718, 38303.1466514 ,
       39829.33466195, 37387.43384507, 31893.15700709, 38913.62185562,
```

```
       26398.88016911, 39524.09705984, 29146.0185881 , 33724.58261975,
       26093.642567  , 30061.73139443, 30672.20659865, 29451.25619021,
       34335.05782397, 33114.10741553, 34945.53302819, 25788.40496489,
       29451.25619021, 33114.10741553, 25788.40496489, 39524.09705984,
       31282.68180287, 28535.54338388, 30977.44420076, 26093.642567  ,
       33114.10741553, 32503.63221131, 26093.642567  , 27314.59297544,
       39524.09705984, 25788.40496489, 39524.09705984, 36776.95864085,
       35556.00823241, 37387.43384507, 26093.642567  , 26398.88016911,
       36166.48343663, 26093.642567  , 34335.05782397, 32503.63221131,
       35556.00823241, 32503.63221131, 36776.95864085, 38303.1466514 ,
       27925.06817966, 26093.642567  , 34640.29542608, 29756.49379232,
       36471.72103874, 33419.34501764, 28535.54338388, 30672.20659865,
       34029.82022186, 39829.33466195, 38913.62185562, 36166.48343663,
       35556.00823241, 26093.642567  , 28230.30578177, 27314.59297544,
       32198.3946092 , 27619.83057755, 28535.54338388, 37082.19624296,
       33724.58261975, 28230.30578177, 31282.68180287, 39524.09705984,
       39829.33466195, 38913.62185562, 32503.63221131, 30366.96899654,
       37387.43384507, 33114.10741553, 29451.25619021, 36776.95864085,
       38913.62185562, 27619.83057755, 33724.58261975, 26704.11777122,
       29146.0185881 , 35861.24583452, 26093.642567  , 32198.3946092 ,
       33114.10741553, 37692.67144718, 36776.95864085, 35250.7706303 ,
       33419.34501764, 30977.44420076, 34945.53302819, 29756.49379232,
       30672.20659865, 26704.11777122, 26093.642567  , 38303.1466514 ,
       29451.25619021, 34640.29542608, 35250.7706303 , 26093.642567  ,
       31587.91940498, 25788.40496489, 33724.58261975, 32198.3946092 ,
       33114.10741553, 36166.48343663, 39829.33466195, 33419.34501764,
       32503.63221131, 39218.85945773, 33724.58261975, 38608.38425351,
       32198.3946092 , 28535.54338388, 32808.86981342, 35861.24583452,
       29451.25619021, 29146.0185881 , 30977.44420076, 31587.91940498,
       27314.59297544, 29146.0185881 , 28535.54338388, 36471.72103874,
       31587.91940498, 34640.29542608, 25788.40496489, 30366.96899654,
       26093.642567  , 29451.25619021, 35556.00823241, 36471.72103874,
       28535.54338388, 30366.96899654, 25788.40496489, 34640.29542608,
       30366.96899654, 37387.43384507, 31282.68180287, 32808.86981342,
       27314.59297544, 37692.67144718, 38608.38425351, 31587.91940498,
       34335.05782397, 35250.7706303 , 34945.53302819, 27925.06817966,
       31587.91940498, 35861.24583452, 30061.73139443, 37692.67144718,
       39829.33466195, 34640.29542608, 33419.34501764, 38608.38425351,
       30061.73139443, 25788.40496489, 33419.34501764, 34029.82022186,
       31587.91940498, 27925.06817966, 35861.24583452, 33724.58261975,
       30672.20659865, 36776.95864085, 33419.34501764, 35861.24583452,
       29146.0185881 , 29756.49379232, 27619.83057755, 28535.54338388,
       29451.25619021, 27619.83057755, 34640.29542608, 33419.34501764,
       27009.35537333, 34640.29542608, 26093.642567  , 34335.05782397,
       37082.19624296, 25788.40496489, 27009.35537333, 34029.82022186,
       30977.44420076, 26398.88016911, 33419.34501764, 27009.35537333,
       35250.7706303 , 34640.29542608, 38303.1466514 , 31587.91940498,
       28840.78098599, 32198.3946092 , 34640.29542608, 27009.35537333,
       35861.24583452, 30366.96899654, 31893.15700709, 34945.53302819,
```

```
       27925.06817966, 30366.96899654, 27314.59297544, 36471.72103874,
       27314.59297544, 26093.642567  , 38608.38425351, 33419.34501764,
       26093.642567  , 25788.40496489, 33419.34501764, 36166.48343663,
       29756.49379232, 27314.59297544, 26398.88016911, 33419.34501764,
       26093.642567  , 25788.40496489, 31282.68180287, 31587.91940498,
       34335.05782397, 26398.88016911, 36166.48343663, 26398.88016911,
       36166.48343663, 39829.33466195, 30061.73139443, 27619.83057755,
       26398.88016911, 39829.33466195, 27619.83057755, 28230.30578177,
       32198.3946092 , 34640.29542608, 25788.40496489, 38913.62185562,
       26398.88016911, 26093.642567  , 34029.82022186, 39218.85945773,
       33419.34501764, 33114.10741553, 29146.0185881 , 30061.73139443,
       27925.06817966, 26093.642567  , 29451.25619021, 39218.85945773,
       33114.10741553, 38913.62185562])
```

```
rmse(smoker_targets, smoker_predictions)
```

```
10711.00334810241
```

```
# smoker m
smoker_model.coef_
```

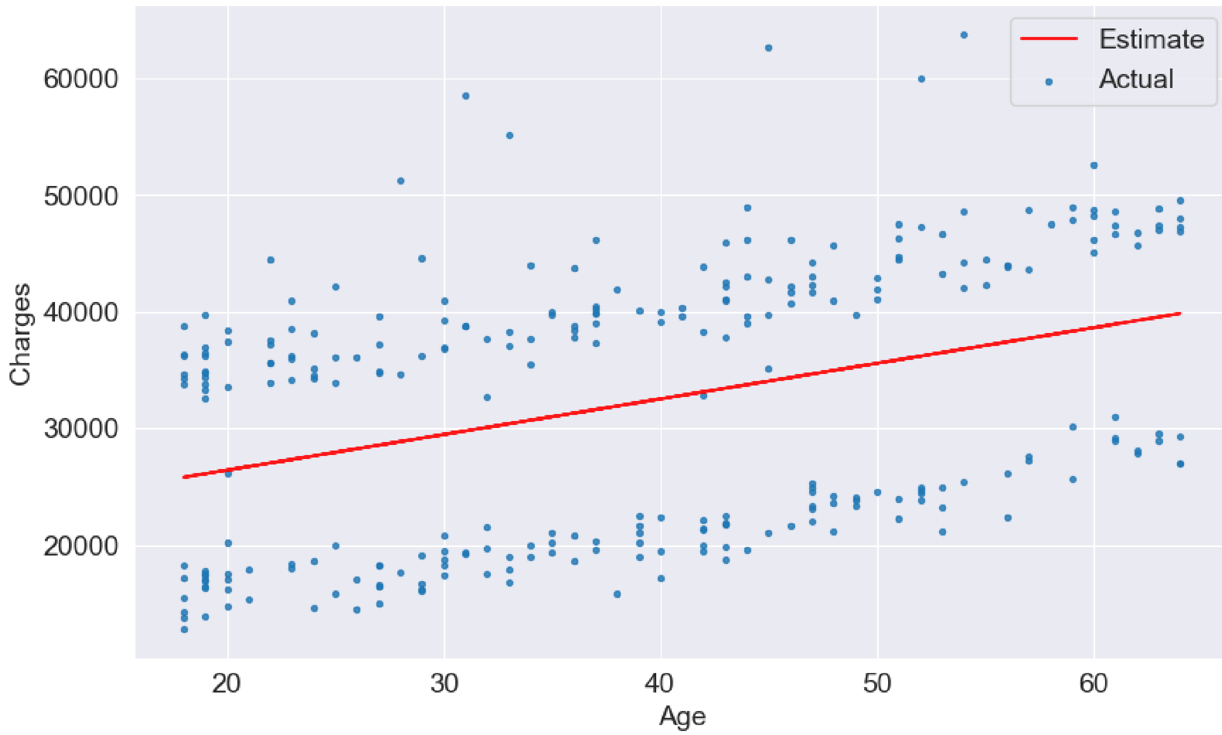
```
array([305.23760211])
```

```
# smoker b
smoker_model.intercept_
```

```
20294.128126915966
```

```
try_smoker_parameters(smoker_model.coef_, smoker_model.intercept_)
```

```
RMSE Loss:  10711.00334810241
```



Linear Regression using Multiple Features

We can use multiple features e.g "age" and "bmi" to predict the "charges"

We simply assume the following relationship:

- charges = w_1 * age + w_2 * bmi + b

```
# Create inputs and targets
inputs, targets = non_smoker_df[['age', 'bmi']],
non_smoker_df['charges']
# Create and train the model
model = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)

Loss:  4662.3128354612945
```

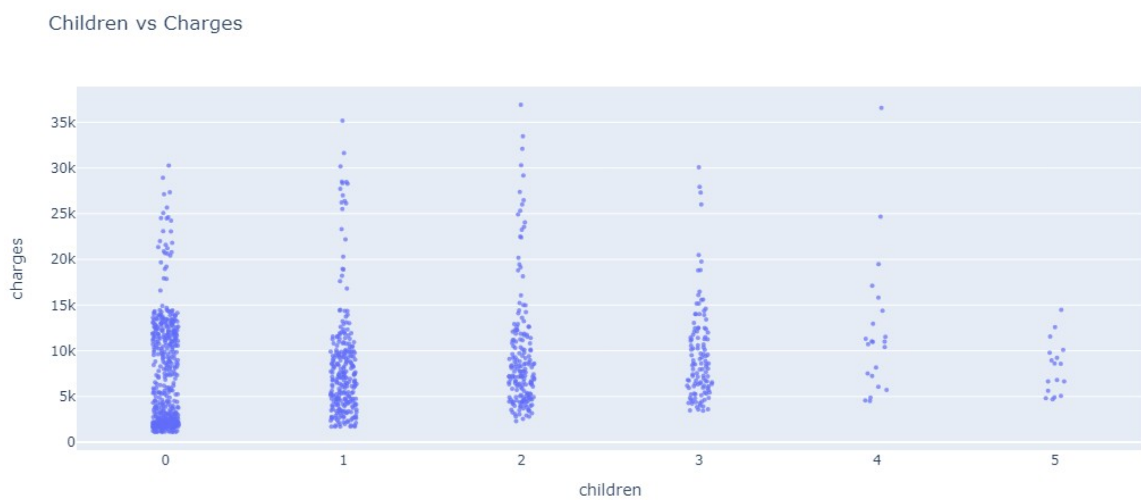
Adding BMI to the model does not really reduce the loss because of the weak correlation between BMI and Charges

```
non_smoker_df.charges.corr(non_smoker_df.bmi)

0.08403654312833272
```

```
non_smoker_df.charges.corr(non_smoker_df.children)
```

```
0.138928704535422
```

```
fig = px.strip(non_smoker_df, x='children', y='charges',
title='Children vs Charges')
fig.update_traces(marker_size = 4, marker_opacity = 0.7)
fig.update_layout(width=700, height=500)
fig.show()
```



Using three features

```
# Create inputs and targets
inputs, targets = non_smoker_df[['age', 'bmi', 'children']],
non_smoker_df['charges']
# Create and train the model
model = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)
```

```
Loss:  4608.470405038246
```

```
model.coef_
```

```
array([265.2938443 ,   5.27956313, 580.65965053])
```

```
model.intercept_
```

```
-2809.2976032235892
```

Lets create a model for the **smokers**

```
# Create inputs and targets
inputs, targets = smoker_df[['age', 'bmi', 'children']],
smoker_df['charges']
# Create and train the model
model_smoker = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model_smoker.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)
```

```
Loss:  5718.202480524154
```

```
model_smoker.coef_
```

```
array([ 264.93316919, 1438.72926245,  198.88027911])
```

```
model_smoker.intercept_
```

```
-22556.088196491593
```

Let's create a model for **all customers**

```
# Create inputs and targets
inputs, targets = medical_df[['age', 'bmi', 'children']],
medical_df['charges']
# Create and train the model
model_all = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model_all.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)
```

```
Loss:  11355.317901125973
```

```
model_all.coef_
```

```
array([239.99447429, 332.0833645 , 542.86465225])
```

```
model_all.intercept_
```

```
-6916.243347787033
```

Linear Model using Categorical Features

To use categorical columns in our model, we need to convert them into numerical formats.

There are three techniques for doing this;

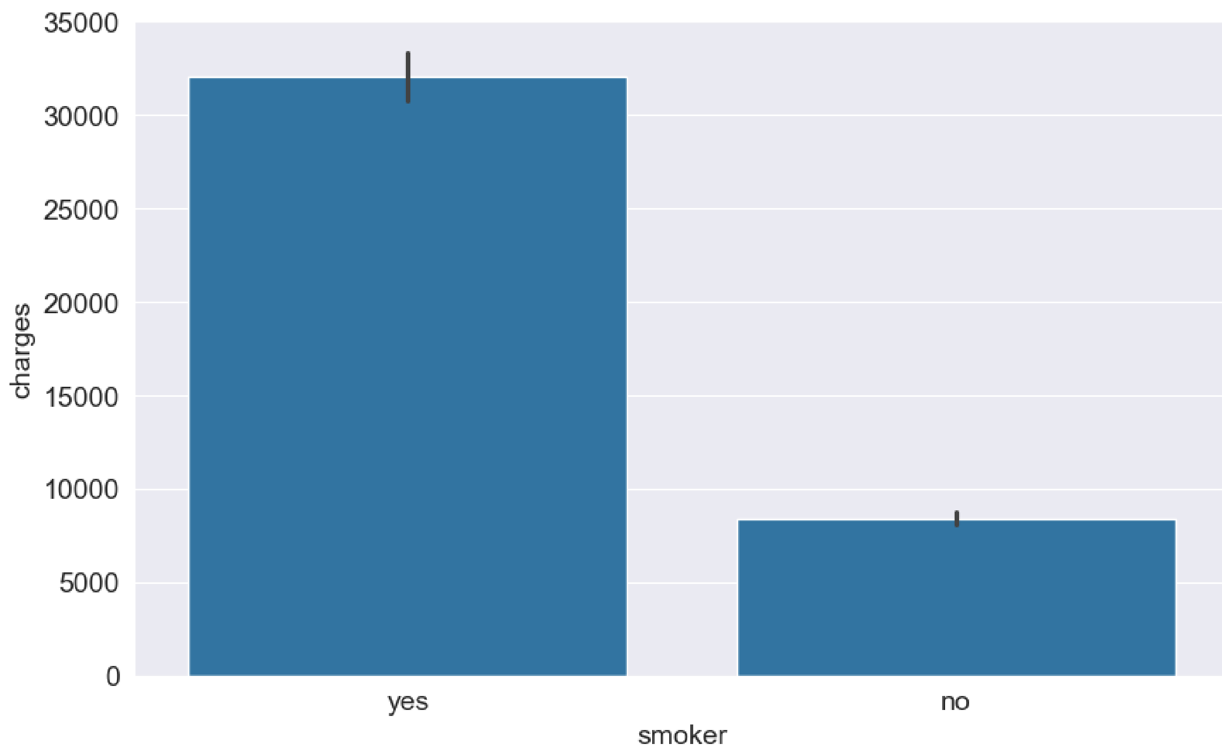
- If the categorical column has two categories (binary category), we replace their values with 0 and 1
- We can use **`one-hot encoding`** for categorical columns with more than two categories
- If the categories have a natural order (e.g. cold, neutral, warm, hot), we can convert them into numbers (e.g. 1, 2, 3, 4) preserving the order. These are called **ordinals**

Binary Categories

The smoker category has two values: "yes" and "no".

We can create a new column "smoker_code" with 0 for "no" and 1 for "yes"

```
sns.barplot(medical_df, x='smoker', y='charges');
```




```
smoker_codes = {'no': 0, 'yes': 1}
medical_df['smoker_code'] = medical_df.smoker.map(smoker_codes)
medical_df
```

```
      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

      smoker_code
0               1
1               0
2               0
3               0
4               0
...           ...
1333            0
1334            0
1335            0
1336            0
1337            1

[1338 rows x 8 columns]
```

```
medical_df.charges.corr(medical_df.smoker_code)
```

```
0.7872514304984772
```

We can now use the `smoker_code` column for linear regression

charges = w1* age + w2 * bmi + w3 * children + w4 * smoker + b

```
# Create inputs and targets
inputs, targets = medical_df[['age', 'bmi', 'children',
'smoker_code']], medical_df['charges']
# Create and train the model
model_full = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model_full.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)
```

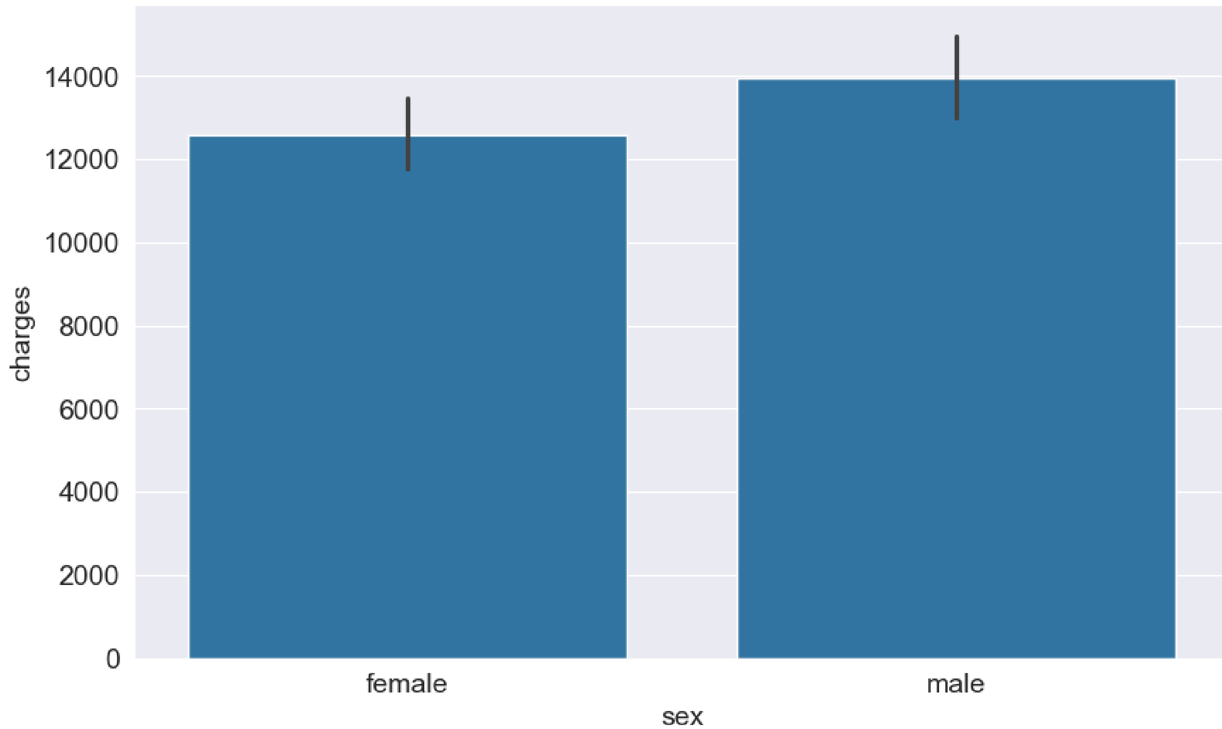
```
Loss:  6056.439217188081
```

The loss reduces from `11355` to `6056` which signifies the importance of not ignoring the categorical data

Lets add the **sex** column as well

```
sns.barplot(medical_df, x='sex', y='charges')
```

```
<Axes: xlabel='sex', ylabel='charges'>
```




```
sex_codes = {'female': 0, 'male': 1}
medical_df['sex_code'] = medical_df.sex.map(sex_codes)
medical_df
```

	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	

	smoker_code	sex_code
0	1	0
1	0	1
2	0	1
3	0	1
4	0	1
...	...	...
1333	0	1
1334	0	0
1335	0	0

```
1336              0          0
1337              1          0

[1338 rows x 9 columns]
```

```
medical_df.charges.corr(medical_df.sex_code)
```

```
0.057292062202025415
```

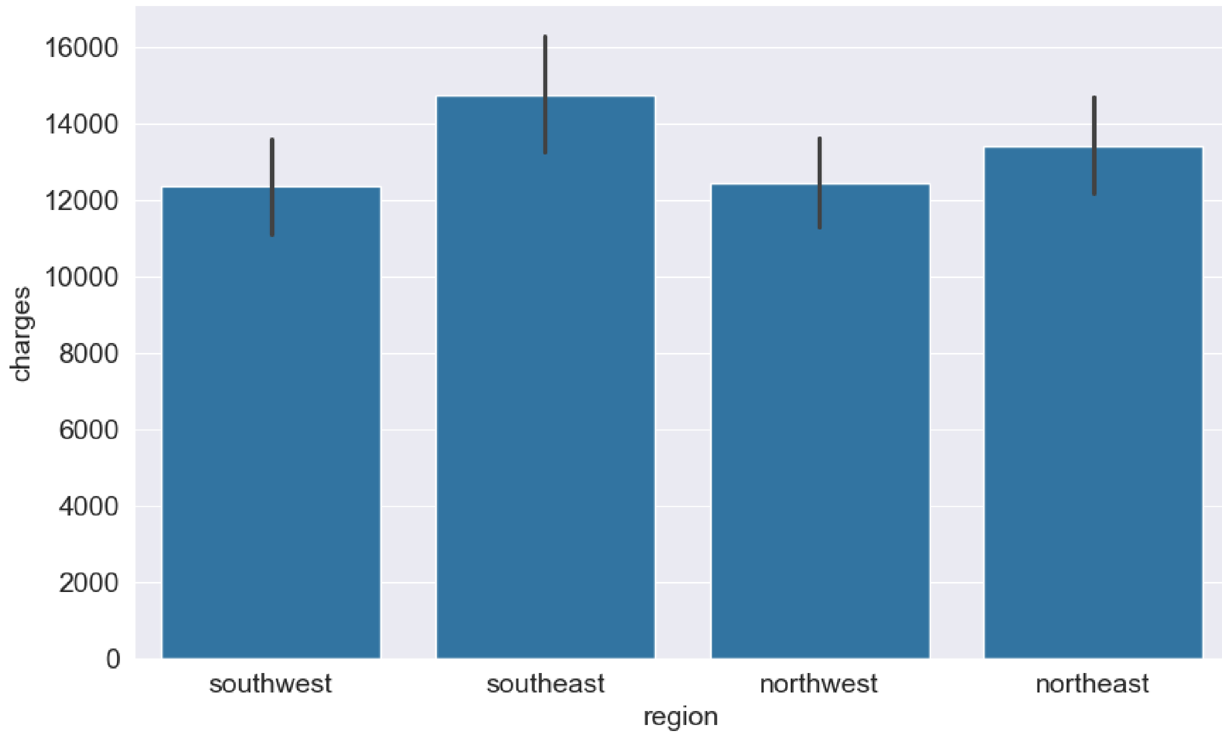
```
# Create inputs and targets
inputs, targets = medical_df[['age', 'bmi', 'children', 'smoker_code',
'sex_code']], medical_df['charges']
# Create and train the model
model_whole = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model_whole.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)
```

```
Loss:  6056.100708754546
```

One Hot Encoding

The "region" column has 4 values so we will use one hot encoding to create a new column for each region

```
sns.barplot(medical_df, x='region', y='charges');
```



Let's use **onehotencoding** for the four regions

We can get the preprocessing library from `sklearn`

```
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
enc.fit(medical_df[['region']])
enc.categories_
```

```
[array(['northeast', 'northwest', 'southeast', 'southwest'],
dtype=object)]
```

```
enc.transform([['northeast'],
               ['northwest']]).toarray()
```

```
C:\Users\admin\miniconda3\envs\alxenv\Lib\site-packages\sklearn\
base.py:464: UserWarning:

X does not have valid feature names, but OneHotEncoder was fitted with
feature names


array([[1., 0., 0., 0.],
       [0., 1., 0., 0.]])
```

```
medical_df[['region']]
```

```
         region
0     southwest
```

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

[1338 rows x 1 columns]
```

```
one_hot = enc.transform(medical_df[['region']]).toarray()
one_hot
```

```
array([[0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       ...,
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 1., 0., 0.]])
```

We can then add the one_hot_encoded keys to the dataframe

```
medical_df[['northeast', 'northwest', 'southeast', 'southwest']] =
one_hot
medical_df
```

```
      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

      smoker_code  sex_code  northeast  northwest  southeast
southwest
0               1         0        0.0        0.0        0.0
1.0
1               0         1        0.0        0.0        1.0
0.0
2               0         1        0.0        0.0        1.0
```

```
0.0
3                0         1        0.0        1.0        0.0
0.0
4                0         1        0.0        1.0        0.0
0.0
...             ...       ...       ...        ...        ...        ..
.
1333             0         1        0.0        1.0        0.0
0.0
1334             0         0        1.0        0.0        0.0
0.0
1335             0         0        0.0        0.0        1.0
0.0
1336             0         0        0.0        0.0        0.0
1.0
1337             1         0        0.0        1.0        0.0
0.0

[1338 rows x 13 columns]
```

We can now include the Region column in our linear regression model

The formula of our model will be as follows:

$$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + w_4 \times smoker + w_5 \times sex + w_6 \times region + b$$

For our specific model, the formula is:

$$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + w_4 \times smoker + w_5 \times sex + w_6 \times northeast + w_7 \times northwe$$

```
# Create inputs and targets
input_cols = ['age', 'bmi', 'children', 'smoker_code', 'sex_code',
'northeast', 'northwest', 'southeast', 'southwest']
inputs, targets = medical_df[['age', 'bmi', 'children', 'smoker_code',
'sex_code', 'northeast', 'northwest', 'southeast', 'southwest']],
medical_df['charges']
# Create and train the model
model_e = LinearRegression().fit(inputs, targets)
# Generate Predictions
predictions = model_e.predict(inputs)
# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)

Loss:  6041.6796511744515

medical_df.charges.corr(medical_df.northeast)

0.006348771280156069
```

Model Improvements

Let's apply some improvements to our model

Feature Scaling

We need to explain the rationale behind the predictions of our models

$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + w_4 \times smoker + w_5 \times sex + w_6 \times northeast + w_7 \times northwe$

To compare the importance of each feature in the model, our first instinct might be to compare their weights

```
model_e.coef_
```

```
array([  256.85635254,   339.19345361,   475.50054515, 23848.53454191,
        -131.3143594 ,   587.00923503,   234.0453356 ,  -448.01281436,
        -373.04175627])
```

```
model_e.intercept_
```

```
-12525.547811195462
```

```
medical_df[input_cols].loc[10]
```

```
age            25.00
bmi            26.22
children        0.00
smoker_code     0.00
sex_code        1.00
northeast       1.00
northwest       0.00
southeast       0.00
southwest       0.00
Name: 10, dtype: float64
```

```
weights_df = pd.DataFrame({
    'feature': np.append(input_cols, 1),
    'weight': np.append(model_e.coef_, model_e.intercept_)
})
weights_df
```

	feature	weight
0	age	256.856353
1	bmi	339.193454
2	children	475.500545
3	smoker_code	23848.534542
4	sex_code	-131.314359
5	northeast	587.009235
6	northwest	234.045336
7	southeast	-448.012814

```
8    southwest   -373.041756
9            1 -12525.547811
```

From our dataframe above, BMI and Northeast have more weight than age. We should remember that the range of values for BMI is limited (15-40) and the "Northeast" column only takes two values: 0 and 1

The different ranges of the different columns cause certain problems:

- We cannot compare the weights of different columns to identify which features are important
- A column with a larger range of inputs may disproportionately affect the loss and dominate the optimization process

For this reason, it's common practice to scale (standardize) the values in numeric column by subtracting the mean and dividing by the standard deviation

We can apply scaling using the **StandardScaler** class from `scikit-learn`

```
medical_df

      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

      smoker_code  sex_code  northeast  northwest  southeast  
southwest
0               1         0        0.0        0.0        0.0
1.0
1               0         1        0.0        0.0        1.0
0.0
2               0         1        0.0        0.0        1.0
0.0
3               0         1        0.0        1.0        0.0
0.0
4               0         1        0.0        1.0        0.0
0.0
...           ...       ...        ...        ...        ...          ..
.
```

```
1333              0          1         0.0          1.0          0.0
0.0
1334              0          0         1.0          0.0          0.0
0.0
1335              0          0         0.0          0.0          1.0
0.0
1336              0          0         0.0          0.0          0.0
1.0
1337              1          0         0.0          1.0          0.0
0.0

[1338 rows x 13 columns]
```

```
from sklearn.preprocessing import StandardScaler
```

```
medical_df[numeric_cols]
```

```
      age     bmi  children
0      19  27.900         0
1      18  33.770         1
2      28  33.000         3
3      33  22.705         0
4      32  28.880         0
...   ...     ...       ...
1333   50  30.970         3
1334   18  31.920         0
1335   18  36.850         0
1336   21  25.800         0
1337   61  29.070         0

[1338 rows x 3 columns]
```

```
numeric_cols = ['age', 'bmi', 'children']
scaler = StandardScaler()
scaler.fit(medical_df[numeric_cols])
```

```
StandardScaler()
```

```
scaler.mean_
```

```
array([39.20702541, 30.66339686,  1.09491779])
```

```
scaler.var_
```

```
array([197.25385199,  37.16008997,   1.45212664])
```

We can now scale the data as follows:

```
scaled_inputs = scaler.transform(medical_df[numeric_cols])
scaled_inputs
```

```
array([[-1.43876426, -0.45332   , -0.90861367],
       [-1.50996545,  0.5096211 , -0.07876719],
       [-0.79795355,  0.38330685,  1.58092576],
       ...,
       [-1.50996545,  1.0148781 , -0.90861367],
       [-1.29636188, -0.79781341, -0.90861367],
       [ 1.55168573, -0.26138796, -0.90861367]])
```

These can now be combined with the categorical data

```
cat_cols = ['smoker_code', 'sex_code', 'northeast', 'northwest',
'southeast', 'southwest']
categorical_data = medical_df[cat_cols].values

inputs[0]

array([-1.43876426, -0.45332   , -0.90861367,  1.        ,
0.        ,
        0.        ,  0.        ,  0.        ,  1.        ])

inputs = np.concatenate((scaled_inputs, categorical_data), axis=1)
targets = medical_df.charges

# Create and train the model
model_a = LinearRegression().fit(inputs, targets)

# Generate predictions
predictions = model_a.predict(inputs)

# Compute loss to evaluate the model
loss = rmse(targets, predictions)
print('Loss: ', loss)

Loss:  6041.6796511744515
```

We can now compare the weights in the formula

$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + w_4 \times smoker + w_5 \times sex + w_6 \times northeast + w_7 \times northwe$

```
weights_df = pd.DataFrame({
    'feature': np.append(numeric_cols + cat_cols, 1),
    'weight': np.append(model_a.coef_, model_a.intercept_)
})
weights_df.sort_values('weight', ascending=False)

       feature        weight
3  smoker_code  23848.534542
9            1   8466.483215
0          age   3607.472736
1          bmi   2067.691966
```

```
5    northeast    587.009235
2     children    572.998210
6    northwest    234.045336
4     sex_code   -131.314359
8    southwest   -373.041756
7    southeast   -448.012814
```

```
new_customer =  [[28, 30, 2, 1, 0, 0, 1, 0, 0.]]
```

```
scaler.transform([[28, 30, 2]])
```

```
C:\Users\admin\miniconda3\envs\alxenv\Lib\site-packages\sklearn\
base.py:464: UserWarning:

X does not have valid feature names, but StandardScaler was fitted
with feature names


array([[-0.79795355, -0.10882659,  0.75107928]])
```

```
model_a.predict([[-0.79795355, -0.10882659,  0.75107928, 1, 0, 0, 1,
0, 0.]])
```

```
array([29875.81463371])
```

Creating a Test-Set

It is common practice to set aside a small fraction of the data for testing and reporting the results of the model

```
from sklearn.model_selection import train_test_split

inputs_train, inputs_test, targets_train, targets_test =
train_test_split(inputs, targets, test_size=0.1)

# Create and train the model
model_f = LinearRegression().fit(inputs_train, targets_train)

# Generate Predictions
predictions_test = model_f.predict(inputs_test)

# Compute the loss to evaluate the model
loss = rmse(targets_test, predictions_test)
print('Test Loss:', loss)
```

```
Test Loss: 6380.875134700934
```

Let's compare with the training loss

```
# Generate predictions
predictions_train = model_f.predict(inputs_train)
```

```python
# Compute loss to evaluate the model
loss = rmse(targets_train, predictions_train)
print('Training Loss:', loss)
```

```
Training Loss: 6010.286018718855
```