Problem Statement

ACME Insurance Inc. offers affordable health insurance to thousands of customer all over the United States 2010. As the lead data scientist at ACME, you're tasked with creating an automated system to estimate the annual medical expenditure for new customers, using information such as their age, sex, BMI, children, smoking habits and region of residence.

Estimates from your system will be used to determine the annual insurance premium (amount paid every month) offered to the customer. Due to regulatory requirements, you must be able to explain why your system outputs a certain prediction.

Downloading Data

In [83]:
```
# Dataset url
acme_url = "https://raw.githubusercontent.com/JovianML/opendatasets/master/data/medical-cha
```

In [84]:
```
# Importing url retrieve method from urllib.request
from urllib.request import urlretrieve

urlretrieve(acme_url, "acme_dataset.csv")
```

Out[84]:
```
('acme_dataset.csv', <http.client.HTTPMessage at 0x217c6eeb850>)
```

In [85]:
```
# Now creating a Pandas dataframe using the downloaded file
import pandas as pd

acme_data = pd.read_csv("acme_dataset.csv")
acme_data
```

Out[85]:

	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

Data Understanding

In [86]:
```
# Checking information regarding dataset
acme_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
```

So, "age", "children", "bmi" (body mass index) and "charges" are numbers, whereas "sex", "smoker" and "region" are strings (possibly categories).

Further, None of the columns contain any missing values meaning our dataset is clean.

In [87]:
```
# Checking unique regions and customers count
acme_data["region"].value_counts()
```

Out[87]:
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

This shows wee have maximum customers from "southeast " region and minimum from "northeast".

In [88]:
```
# Checking sex and count
acme_data["sex"].value_counts()
```

Out[88]:
```
male      676
female    662
Name: sex, dtype: int64
```

Fairly the "sex" distribution is similar among both the sex. Later we can see if we want to standarize it for our model.

In [89]:
```
# Checking our Numerical Data
acme_data.describe()
```

Out[89]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900

25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Some references are:

1. age : a. The minimum age is 18. b. The maximum age is 64.

   It can be argued that maybe this Insurance Company can only take applications between the 18-64 ages.

2. bmi : The "bmi" lies from "underweight" i.e. 15.96 to "obesity range" i.e. 53.13

3. children: Some of our customers even have maximum 5 childrens which is fine as the mean of having children is 1.

4. charges: The "charges" column seems to be significantly skewed meaning there can be the outliers in this column as the minimum charges falls in 1K whereas maximum and even the the median (i.e. 50 percentile) is much lower than the maximum value

Exploratory Analysis and Visualization

In [90]:
```
# Importing viz libraries
import plotly.express as px
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

Age

Let's plot a histogram and a box plot for "age". As the minimum age in the dataset is 18 and the maximum age is 64. Thus, we can visualize the distribution of age using a histogram with 47 bins ((64-18) + 1) (one for each year).

In [91]:
```
acme_data["age"].describe()
```

Out[91]:
```
count    1338.000000
mean       39.207025
std        14.049960
min        18.000000
25%        27.000000
50%        39.000000
75%        51.000000
max        64.000000
Name: age, dtype: float64
```

In [92]:
```
fig = px.histogram(acme_data,
                   x = 'age',
                   marginal = 'box',
                   nbins = 47,
                   title = 'Distribution of Age',
                   color_discrete_sequence=['skyblue'])

fig.update_layout(bargap=0.1)
fig.show()
```

The "age" distribution in the dataset is almost uniform, with 20-30 customers at every age, except for the ages 18 and 19, which seem to have over twice as many customers as other ages. The uniform distribution might arise from the fact that there isn't a big variation in the number of people of any given age (between 18 & 64) in the USA.

Guess with reason:

> There are over twice as many customers with ages 18 and 19, compared to other ages because:
>
> > Since 18-19 is the legal age the insurance company may be asking for lower interest rates for people in this age group because of which the individuals are more interested in applying for Medical insurance.

Body Mass Index

Let's look at the distribution of BMI (Body Mass Index) of customers, using a histogram and box plot.

In [93]:
```
fig = px.histogram(acme_data,
                   x='bmi',
                   marginal='box',
                   color_discrete_sequence=['turquoise'],
                   title='Distribution of Body Mass Index (BMI)')
fig.update_layout(bargap=0.1)
fig.show()
```

The measurements of body mass index seem to form a Gaussian distribution centered around the value 30, with a few outliers towards the right.

> **Question**: Why the distribution of "ages" forms a "uniform distribution" while the distribution of "BMIs" forms a "gaussian distribution"?
>
> > It may be influenced by various factors including the characteristics of the populations being studied and the underlying processes that contribute to the variables. But the reasons that I can think are: The uniform distribution for ages can have relatively consistent representation of individuals across different age ranges. But the gaussian distribution for BMIs is influenced by multiple factors such as genetics, lifestyle, diet, and overall health.

The Central Limit Theorem suggests that the distribution of the sum (or average) of a large number of

independent, identically distributed random variables, each with finite mean and variance, approaches a normal distribution. The various factors influencing BMI could contribute to a Gaussian-like distribution.

Charges

Let's visualize the distribution of "charges" i.e. the annual medical charges for customers. This is the column we're trying to predict.

Let's also use the categorical column "smoker" to distinguish the charges for smokers and non-smokers.

In [94]:
```
fig = px.histogram(acme_data,
                   x='charges',
                   marginal='box',
                   color='smoker',
                   color_discrete_sequence=['green', 'grey'],
                   title='Annual Medical Charges: with Smoking Habit')
fig.update_layout(bargap=0.1)
fig.show()
```

We can make the following observations from the above graph:

- For most customers, the annual medical charges are under $10,000. Only a small fraction of customer have higher medical expenses, possibly due to major accidents, illnesses or genetical diseases. The distribution follows a "power law".
- There is a significant difference in medical expenses between smokers and non-smokers. While the median for non-smokers is $7300, the median for smokers is close to $35,000.

Let's also use the categorical column "region" to distinguish the charges for all the 4 regions.

In [95]:
```
fig = px.histogram(acme_data,
                   x='charges',
                   marginal='box',
                   color='region',
                   color_discrete_sequence=px.colors.sequential.Jet,
                   title='Annual Medical Charges: with Region')
fig.update_layout(bargap=0.1)
fig.show()
```

We can make the following observations from the above graph:

- For most customers, the annual medical charges are under $20,000. Only a small fraction of customer have higher medical expenses, possibly due to major accidents, illnesses or genetical diseases.
- There is no significant difference in medical expenses between the 4 regions as the median is between 8k- 10k expenses.

Let's also use the categorical column "sex" to distinguish the charges for males and females.

In [96]:
```
fig = px.histogram(acme_data,
```

```
                    x='charges',
                    marginal='box',
                    color='sex',
                    color_discrete_sequence=px.colors.sequential.Rainbow,
                    title='Annual Medical Charges: with Sex')
fig.update_layout(bargap=0.1)
fig.show()
```

We can make the following observations from the above graph:

- For most of our female customers, the annual medical charges are under $12K-14,000.

Charges based on sex and age

In [97]:
```
fig = px.histogram(acme_data,
                   x = 'age',
                   y = 'charges',
                   marginal='box',
                   color='sex',
                   color_discrete_sequence=px.colors.sequential.Rainbow,
                   title='Annual Medical Charges: Based on Age and Sex')
fig.update_layout(bargap=0.1)
fig.show()
```

We can make the following observations from the above graph:

- Customers whose age >= 40 are prone to pays the annual medical charges over double than <= 40 group. Maybe because of age group more prone to disease.

Smoker' Gender

Let's visualize the distribution of the "smoker" column (containing values "yes" and "no") using a histogram.

In [98]:
```
px.histogram(acme_data, x='smoker', color='sex', title='Smoker')
```

Inference can be drawn that 20% of customers have reported that they smoke. We can also see that smoking appears a more common habit among males.

Age and Charges

Let's visualize the relationship between "age" and "charges" using a scatter plot. Each point in the scatter plot represents one customer. We'll also use values in the "smoker" column to color the points.

In [99]:
```
fig = px.scatter(acme_data,
                 x='age',
                 y='charges',
                 color='smoker',
                 opacity=0.8,
                 hover_data=['sex'],
                 title='Age vs. Charges')
fig.update_traces(marker_size=5)
```

```
fig.show()
```

We can make the following observations from the above chart:

- The general trend seems to be that medical charges increase with age, as we might expect. However, there is significant variation at every age, and it's clear that age alone cannot be used to accurately determine medical charges.
- We can see three "clusters" of points, each of which seems to form a line with an increasing slope:
    1. The first and the largest cluster consists primary of presumably "healthy non-smokers" who have relatively low medical charges compared to others
    2. The second cluster contains a mix of smokers and non-smokers. It's possible that these are actually two distinct but overlapping clusters: "non-smokers with medical issues" and "smokers without major medical issues".
    3. The final cluster consists exclusively of smokers, presumably smokers with major medical issues that are possibly related to or worsened by smoking.

BMI and Charges

Let's visualize the relationship between BMI (body mass index) and charges using another scatter plot. Once again, we'll use the values from the "smoker" column to color the points.

In [100…

```
fig = px.scatter(acme_data,
                 x='bmi',
                 y='charges',
                 color='smoker',
                 opacity=0.8,
                 hover_data=['sex'],
                 title='BMI vs. Charges')
fig.update_traces(marker_size=5)
fig.show()
```

It appears that for non-smokers, an increase in BMI doesn't seem to be related to an increase in medical charges. However, medical charges seem to be significantly higher for smokers with a BMI greater than 30.

Children vs. Charges

In [101…

```
fig = px.violin(acme_data,
                x='children',
                y='charges',
                title='Children vs. Charges')
fig.update_traces(marker_size=5)
fig.show()
```

Inferences drawn are:

1. People with Zero Children are paying greater annual charges.
2. People with 5 Children have taken the insurance but the cost is even lesser than $10K.
3. The biggest cluster of violin lies in under $20K.

Correlation

As you can tell from the analysis, the values in some columns are more closely related to the values in "charges" compared to other columns. E.g. "age" and "charges" seem to grow together, whereas "bmi" and "charges" don't.

This relationship is often expressed numerically using a measure called the *correlation coefficient*, which can be computed using the `.corr` method of a Pandas series.

In [102…

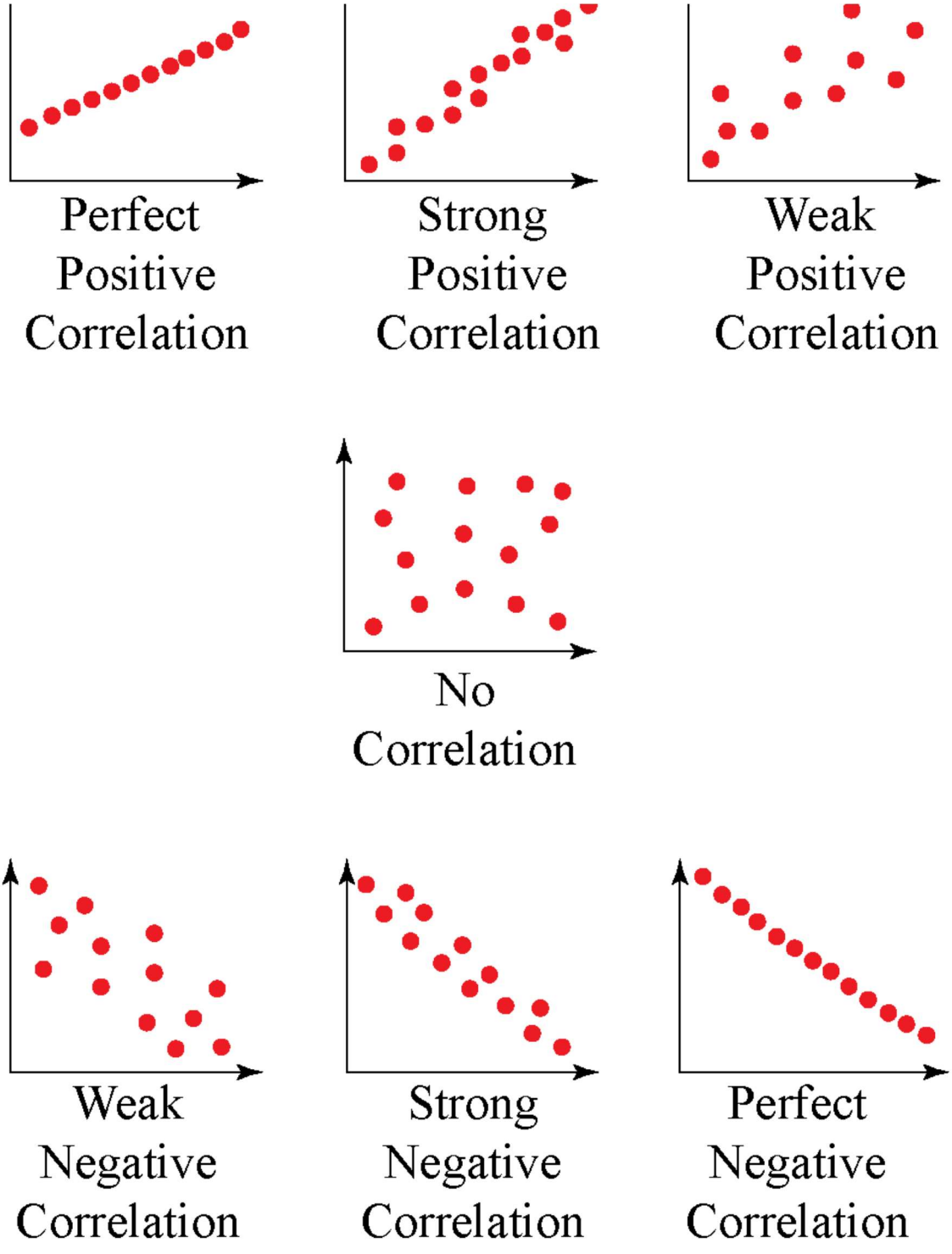
```
print("Correlation of charges and age: ", acme_data.charges.corr(acme_data.age))
print("Correlation of charges and bmi: ", acme_data.charges.corr(acme_data.bmi))
print("Correlation of charges and children: ", acme_data.charges.corr(acme_data.children))

# To compute the correlation for categorical columns, they must first be converted into num
smoker_values = {'no': 0, 'yes': 1}
smokers = acme_data.smoker.map(smoker_values)
print("Correlation of charges and smoker: ", acme_data.charges.corr(smokers))
```

```
Correlation of charges and age:  0.2990081933306476
Correlation of charges and bmi:  0.1983409688336288
Correlation of charges and children:  0.0679982268479047
Correlation of charges and smoker:  0.7872514304984785
```

Here's how correlation coefficients can be interpreted (source):

- **Strength**: The greater the absolute value of the correlation coefficient, the stronger the relationship.
  - The extreme values of -1 and 1 indicate a perfectly linear relationship where a change in one variable is accompanied by a perfectly consistent change in the other. For these relationships, all of the data points fall on a line. In practice, you won't see either type of perfect relationship.
  - A coefficient of zero represents no linear relationship. As one variable increases, there is no tendency in the other variable to either increase or decrease.
  - When the value is in-between 0 and +1/-1, there is a relationship, but the points don't all fall on a line. As r approaches -1 or 1, the strength of the relationship increases and the data points tend to fall closer to a line.
- **Direction**: The sign of the correlation coefficient represents the direction of the relationship.
  - Positive coefficients indicate that when the value of one variable increases, the value of the other variable also tends to increase. Positive relationships produce an upward slope on a scatterplot.
  - Negative coefficients represent cases when the value of one variable increases, the value of the other variable tends to decrease. Negative relationships produce a downward slope.

Here's the same relationship expressed visually (source):



The correlation coefficient has the following formula:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

You can learn more about the mathematical definition and geometric interpretation of correlation here: https://www.youtube.com/watch?v=xZ_z8KWkhXE

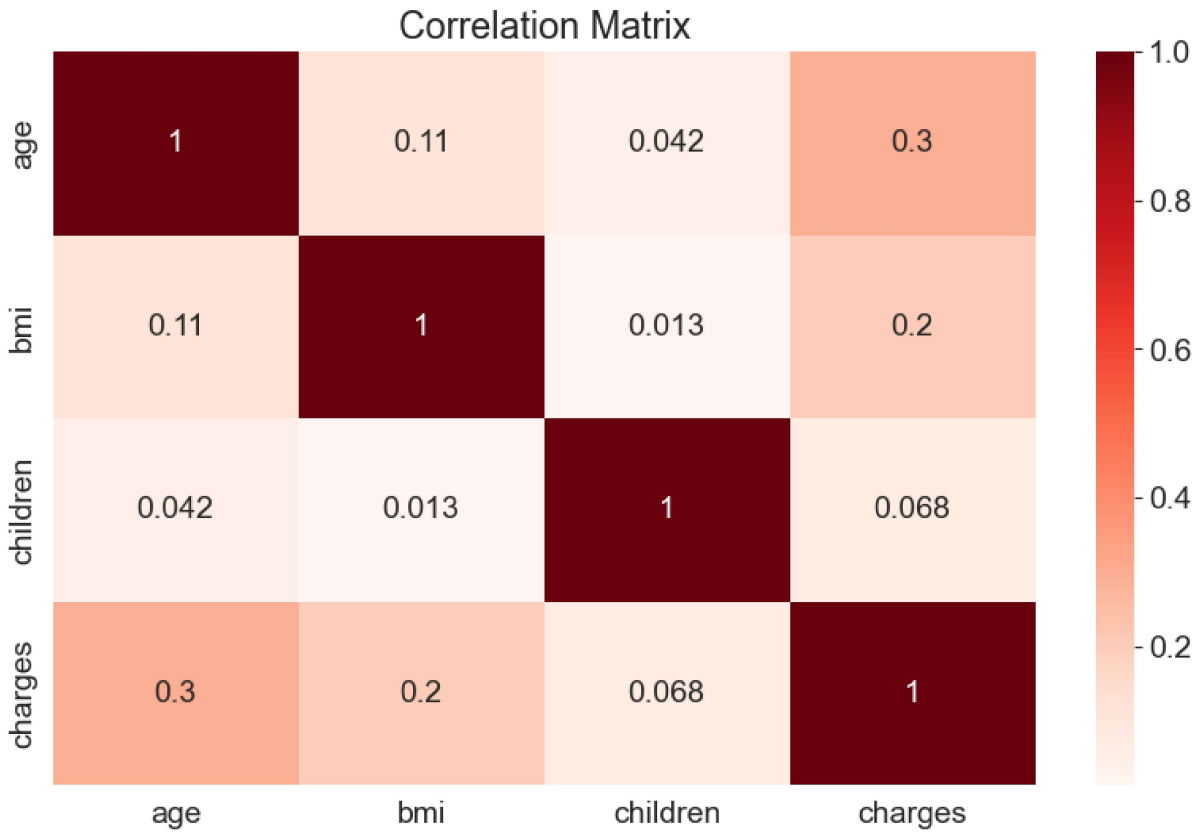
In [103…
```
# Pandas dataframes also provide a `.corr` method to compute the correlation coefficients b
acme_data.corr()
```

Out[103…

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

In [104…
```
# The result of .corr is called a correlation matrix and is often visualized using a heatma
sns.heatmap(acme_data.corr(), cmap='Reds', annot=True)
plt.title('Correlation Matrix')
```

Out[104… `Text(0.5, 1.0, 'Correlation Matrix')`



Correlation vs causation fallacy: Note that a high correlation cannot be used to interpret a cause-effect relationship between features. Two features X and Y can be correlated if X causes Y or if Y causes X, or if both are caused independently by some other factor Z, and the correlation will no longer hold true if one of the cause-effect relationships is broken. It's also possible that X are Y simply appear to be correlated because the sample is too small.

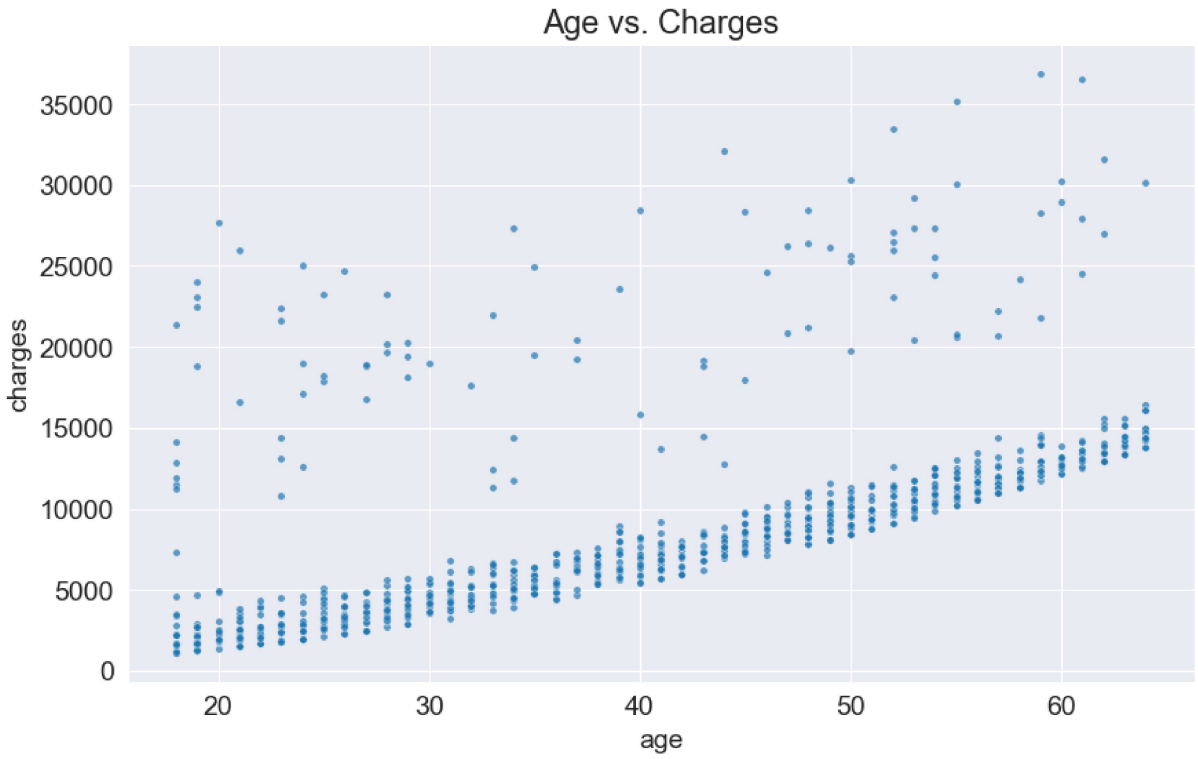
While this may seem obvious, computers can't differentiate between correlation and causation, and decisions based on automated system can often have major consequences on society, so it's important to study why automated systems lead to a given result. Determining cause-effect relationships requires human insight.

Linear Regression using a Single Feature

We now know that the "smoker" and "age" columns have the strongest correlation with "charges". Let's try to find a way of estimating the value of "charges" using the value of "age" for non-smokers. First, let's create a data frame containing just the data for non-smokers.

In [105…
```
non_smoker_df = acme_data[acme_data.smoker == 'no']
```

In [106…
```
# Let's visualize the relationship between "age" and "charges"
plt.title('Age vs. Charges')
sns.scatterplot(data=non_smoker_df, x='age', y='charges', alpha=0.7, s=15);
```



Apart from a few exceptions, the points seem to form a line. We'll try and "fit" a line using this points, and use the line to predict charges for a given age. A line on the X&Y coordinates has the following formula:

$y = wx + b$

The line is characterized two numbers: w (called "slope") and b (called "intercept").

Model

In the above case, the x axis shows "age" and the y axis shows "charges". Thus, we're assuming the following relationship between the two:

$$charges = w \times age + b$$

We'll try determine w and b for the line that best fits the data.

- This technique is called *linear regression*, and we call the above equation a *linear regression model*, because it models the relationship between "age" and "charges" as a straight line.
- The numbers w and b are called the *parameters* or *weights* of the model.
- The values in the "age" column of the dataset are called the *inputs* to the model and the values in the charges column are called "targets".

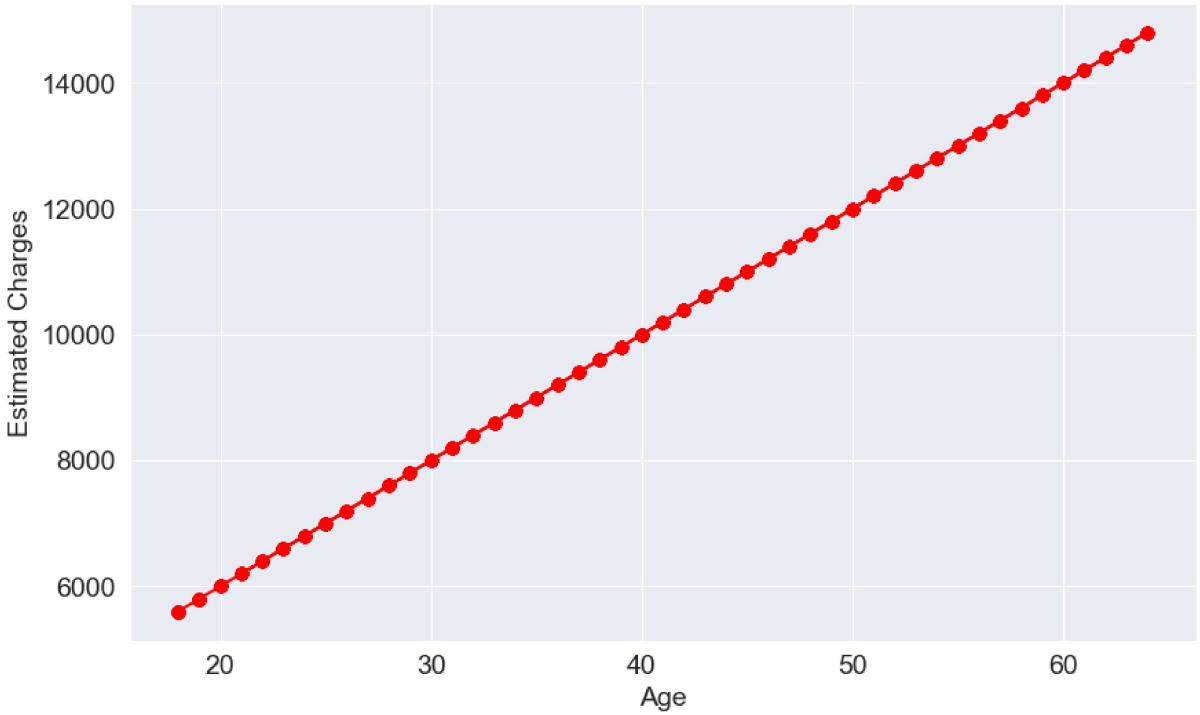
Let define a helper function `estimate_charges` , to compute $charges$, given age, w and b.

In [107…
```
# The estimate_charges function is our very first model
def estimate_charges(age, w, b):
    return w * age + b
```

In [108…
```
w = 200
b = 2000
```

In [109…
```
ages = non_smoker_df.age
estimated_charges = estimate_charges(ages, w, b)
```

In [110…
```
plt.plot(ages, estimated_charges, 'r-o');
plt.xlabel('Age');
plt.ylabel('Estimated Charges');
```

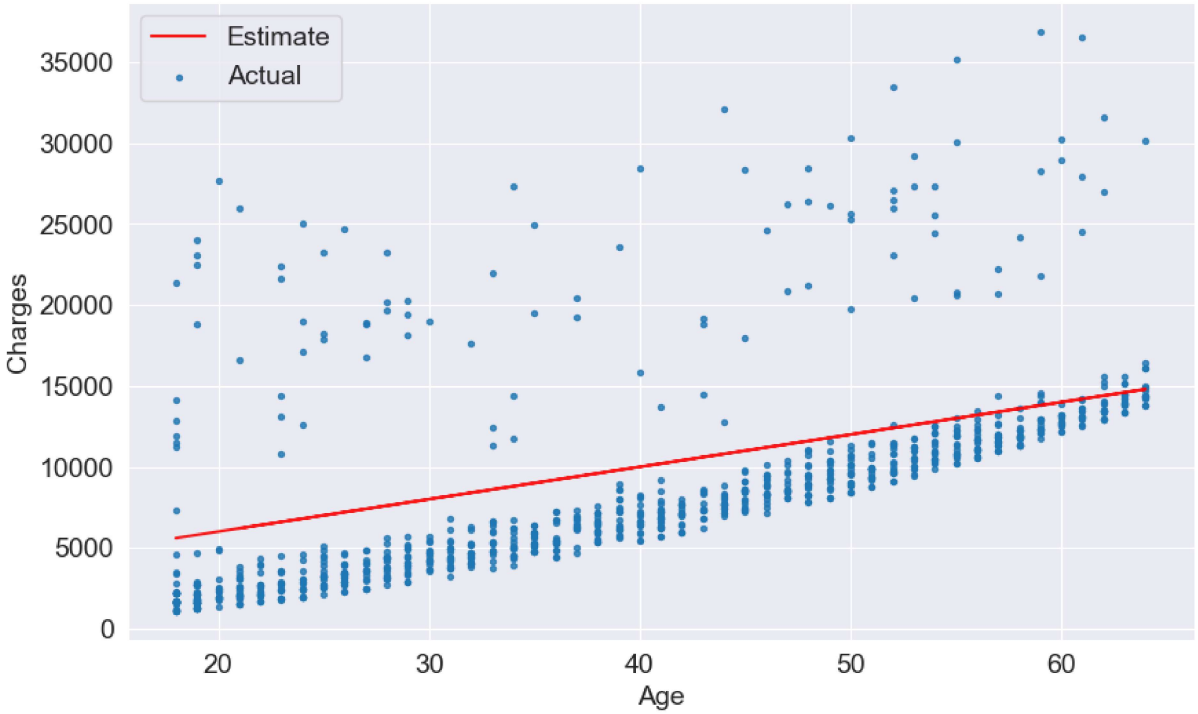


As expected, the points lie on a straight line.

We can overlay this line on the actual data, so see how well our *model* fits the *data*.

In [111…

```
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']);
```

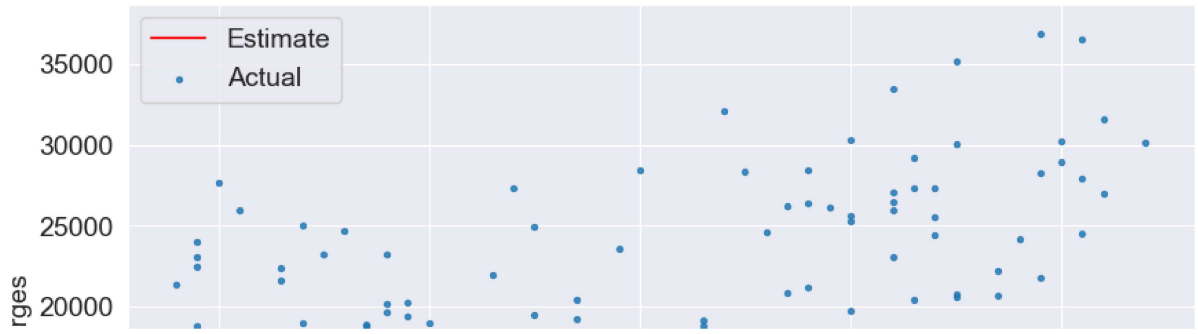


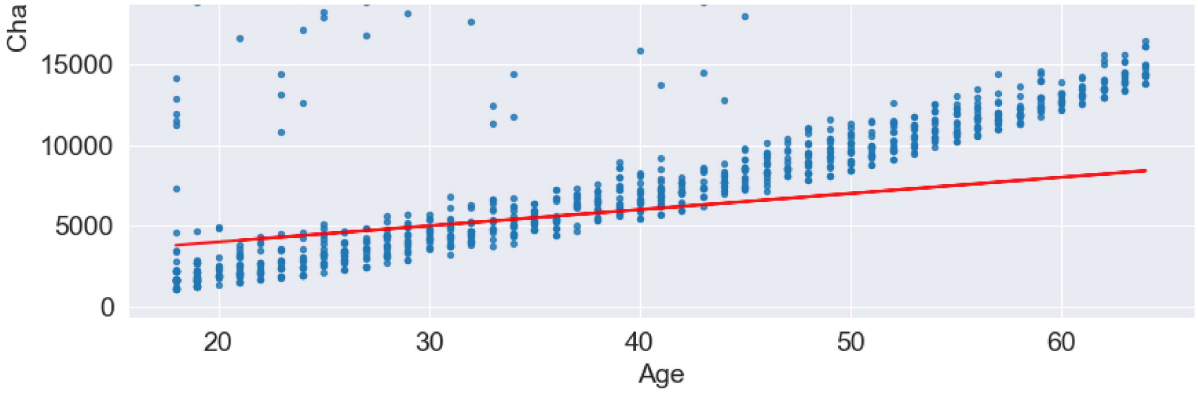
In [112…

```
"""Clearly, the our estimates are quite poor and the line does not "fit" the data.
   However, we can try different values of $w$ and $b$ to move the line around.
   Let's define a helper function `try_parameters` which takes `w` and `b` as
   inputs and creates the above plot."""

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

    estimated_charges = estimate_charges(ages, w, 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']);
```

In [113…

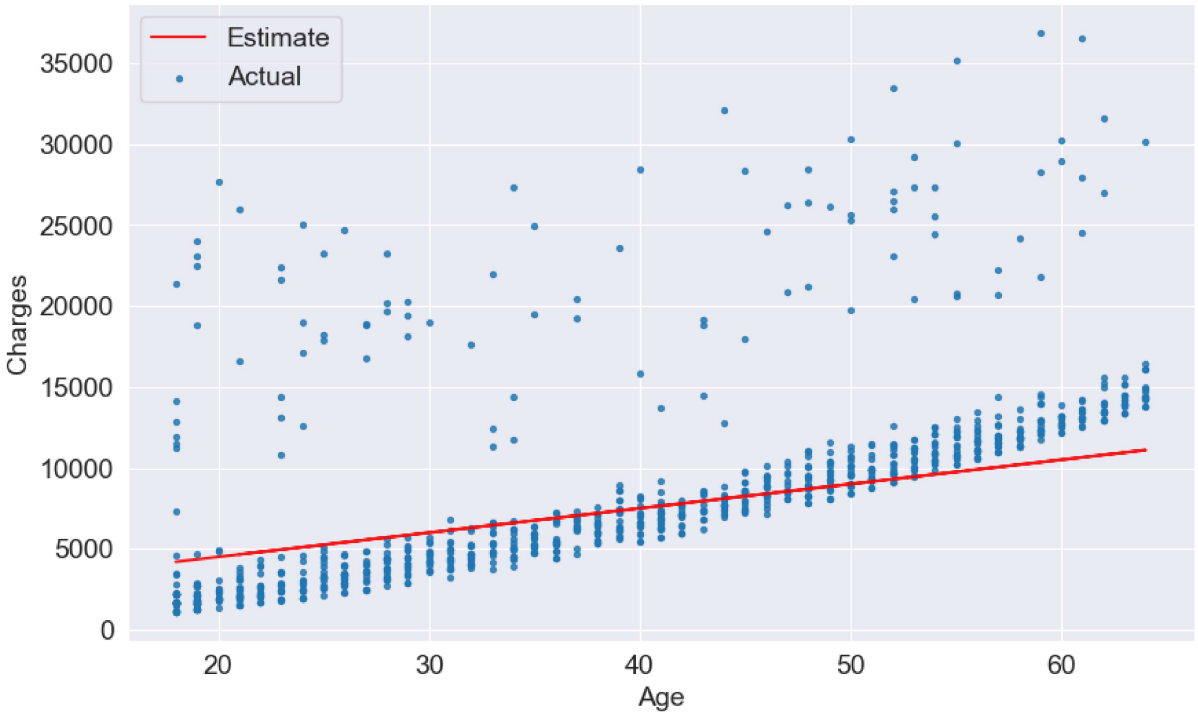
```
try_parameters(100, 2000)
```





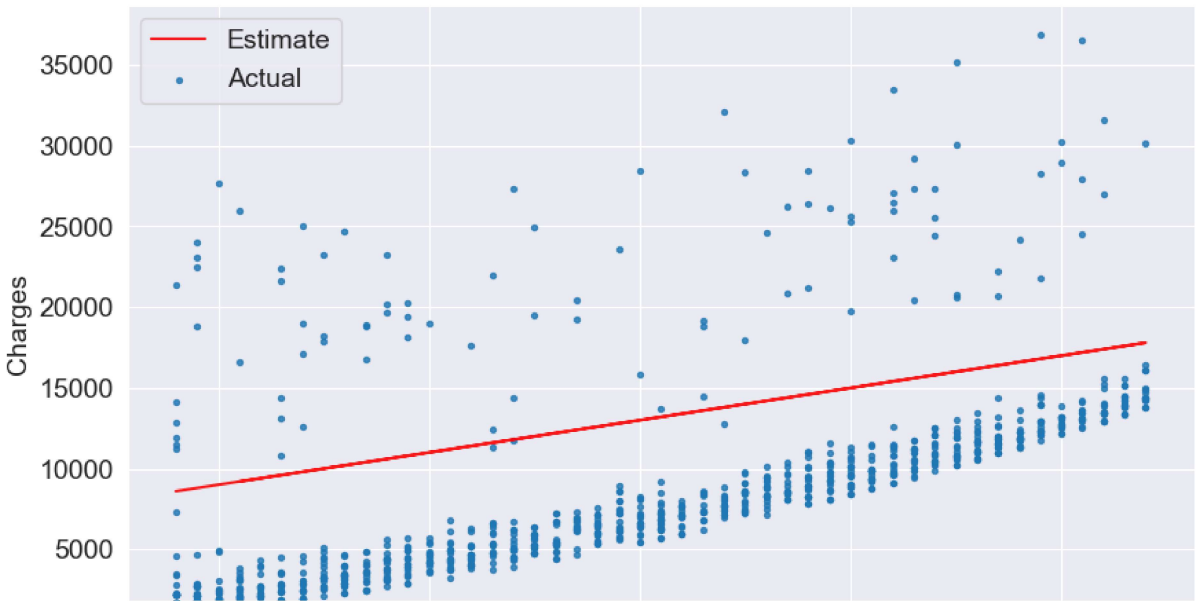
In [114…

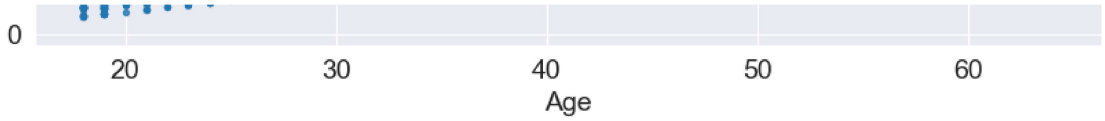
```
try_parameters(150, 1500)
```



In [115…

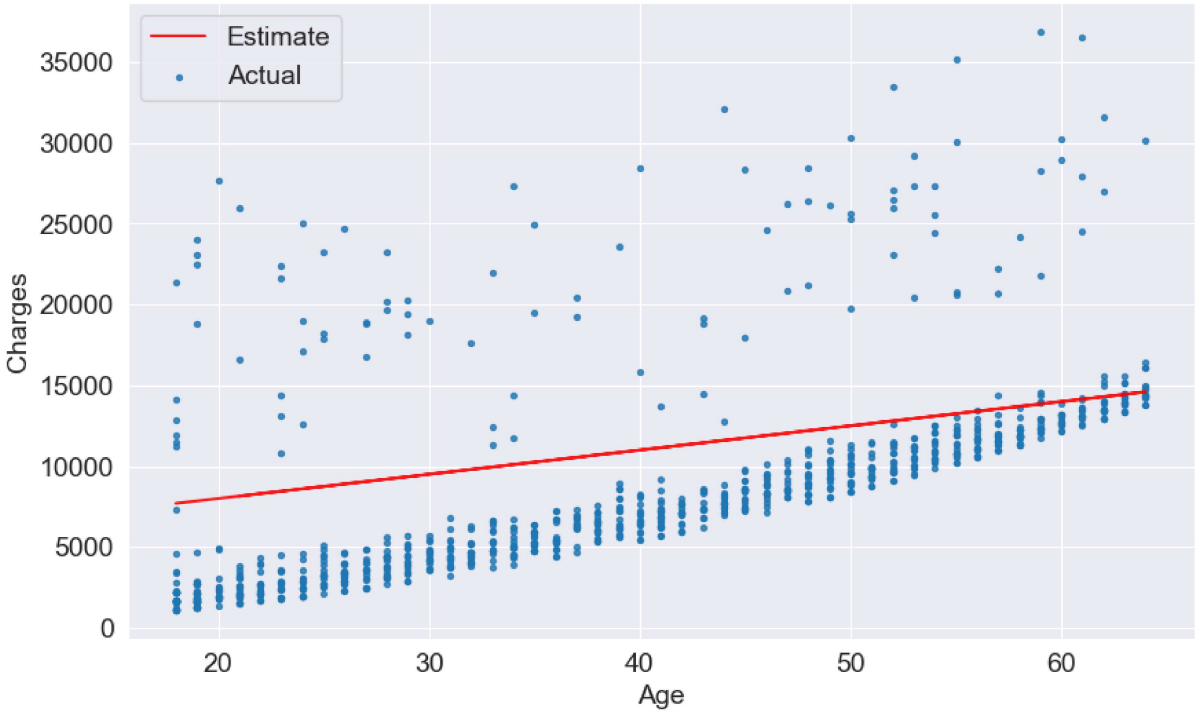
```
try_parameters(200, 5000)
```





In [116…

```
try_parameters(150, 5000)
```



As we change the values, of w and b manually, trying to move the line visually closer to the points, we are *learning* the approximate relationship between "age" and "charges".

Wouldn't it be nice if a computer could try several different values of `w` and `b` and *learn* the relationship between "age" and "charges"? To do this, we need to solve a couple of problems:

1. We need a way to measure numerically how well the line fits the points.
2. Once the "measure of fit" has been computed, we need a way to modify `w` and `b` to improve the the fit.

If we can solve the above problems, it should be possible for a computer to determine `w` and `b` for the best fit line, starting from a random guess.

Loss/Cost Function

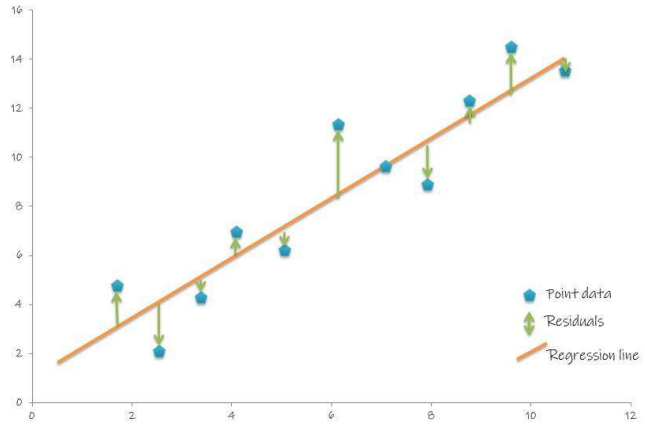
We can compare our model's predictions with the actual targets using the following method:

- Calculate the difference between the targets and predictions (the differenced is called the "residual")
- Square all elements of the difference matrix to remove negative values.
- Calculate the average of the elements in the resulting matrix.
- Take the square root of the result

The result is a single number, known as the **root mean squared error** (RMSE). The above description can be stated mathematically as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Geometrically, the residuals can be visualized as follows:

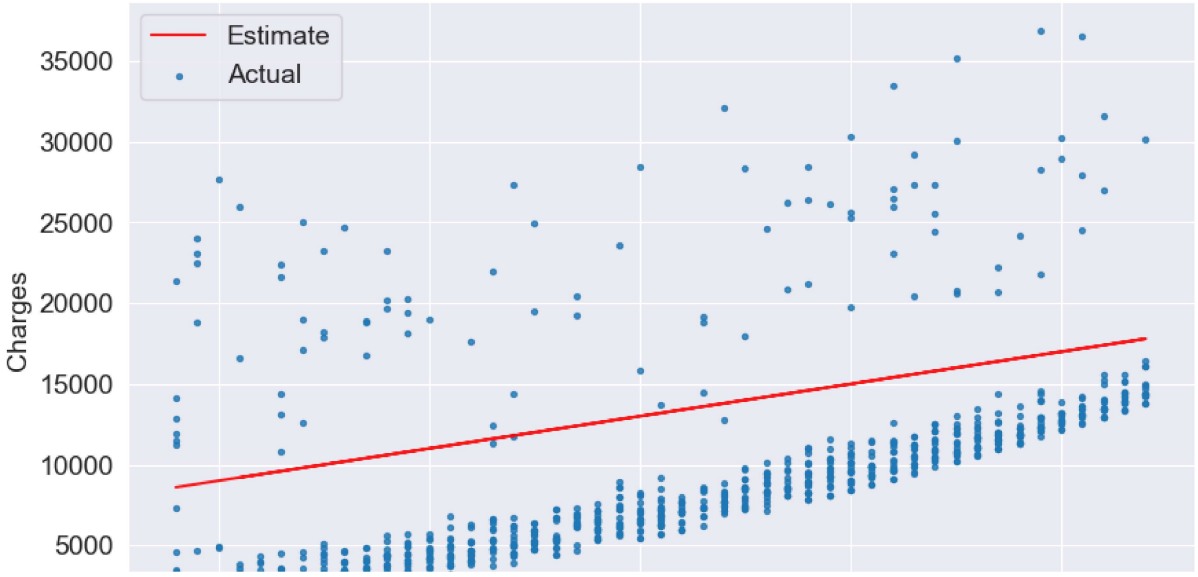


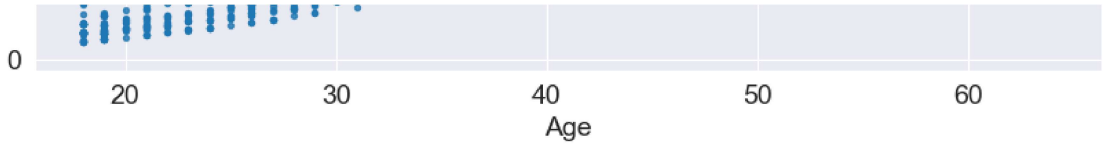
Let's define a function to compute the RMSE.

In [117…
```
import numpy as np
```

In [118…
```
def rmse(targets, predictions):
    return np.sqrt(np.mean(np.square(targets-predictions)))
```

In [119…
```
# Let's compute the RMSE for our model with a sample set of weights
w = 200
b = 5000
try_parameters(w, b)
```





In [120…
```
targets = non_smoker_df['charges']
predicted = estimate_charges(non_smoker_df.age, w, b)
```

In [121…
```
rmse(targets, predicted)
```

Out[121… 6509.498468654397

Here's how we can interpret the above number: *On average, each element in the prediction differs from the actual target by $6509.*

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.

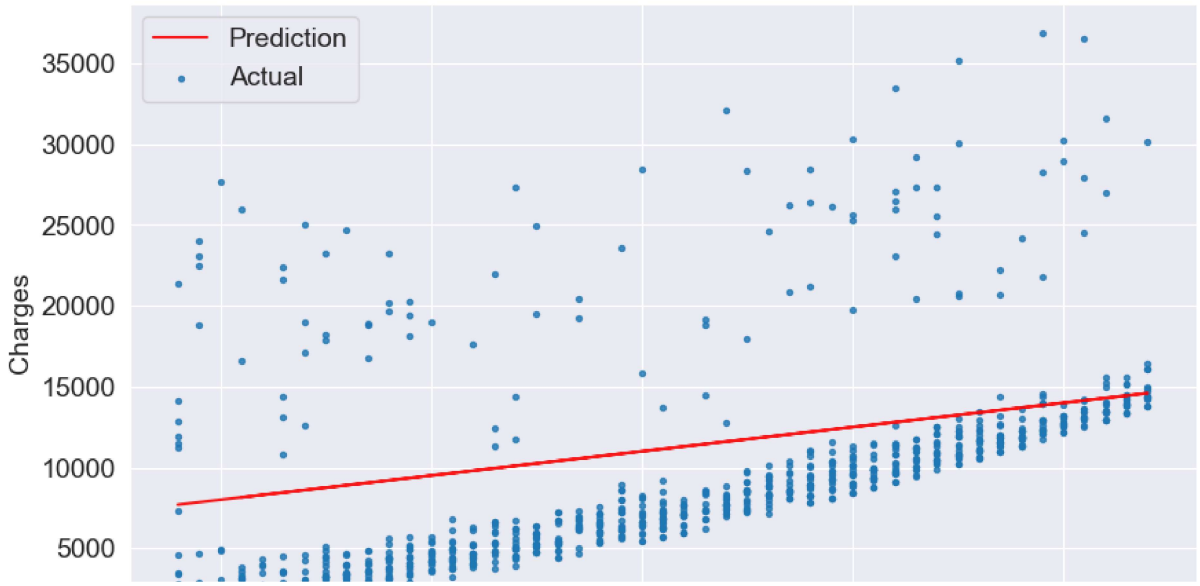
If rmse(loss) is high means model is bad.

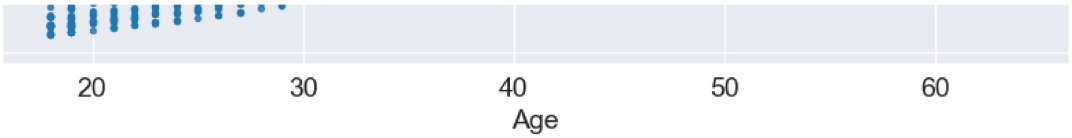
Let's modify the `try_parameters` functions to also display the loss.

In [122…
```
def try_parameters(w, b):
    ages = non_smoker_df.age
    target = non_smoker_df.charges
    predictions = estimate_charges(ages, w, 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)
```

In [123…
```
try_parameters(150, 5000)
```

```
RMSE Loss:  5530.04834104994
```

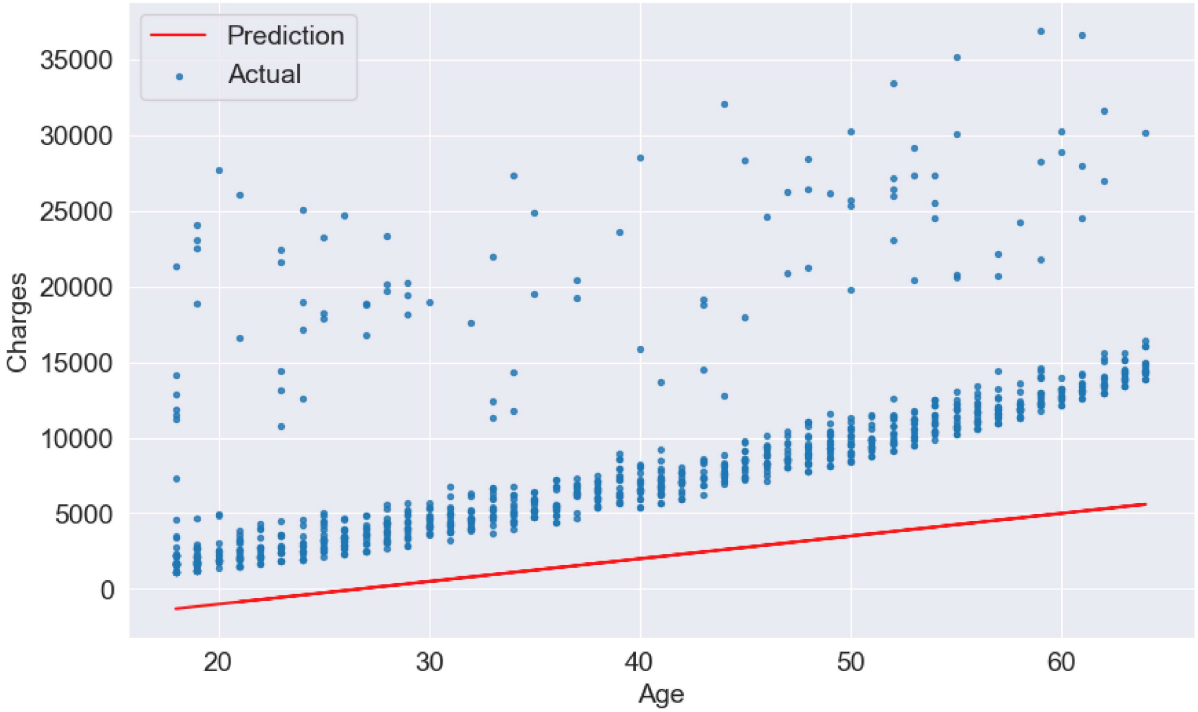




In [124…

```
try_parameters(150, -4000)
```

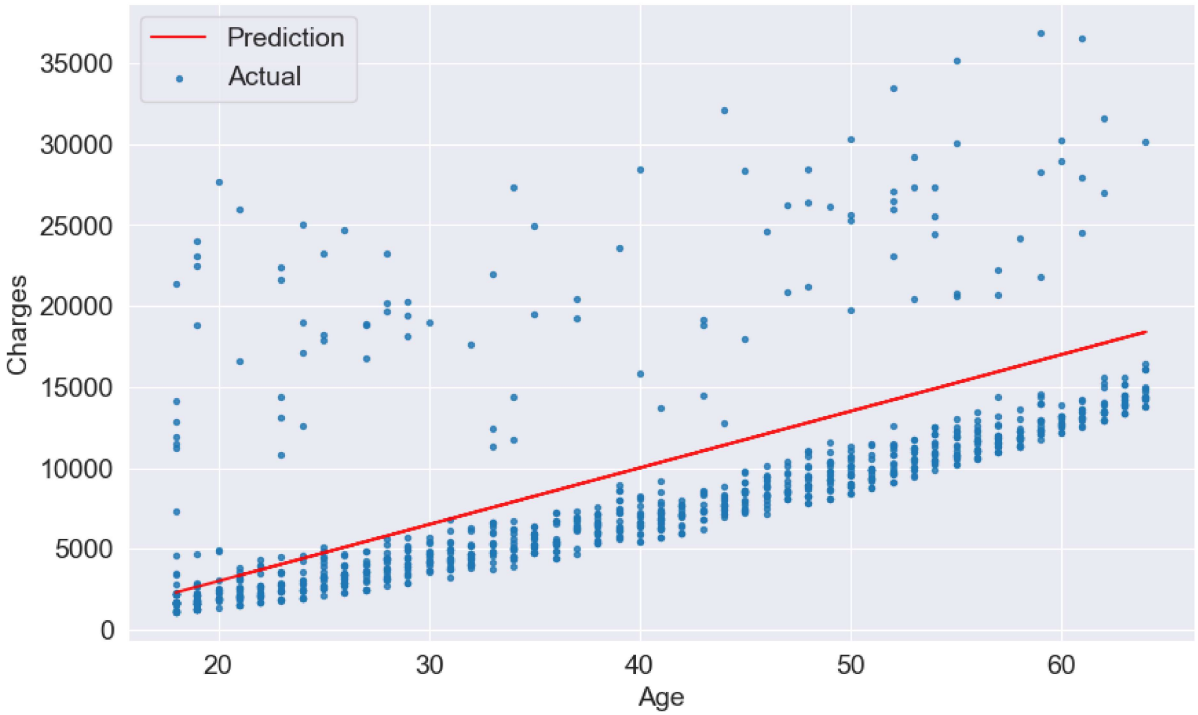
```
RMSE Loss:  8188.885790016007
```



In [125…

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

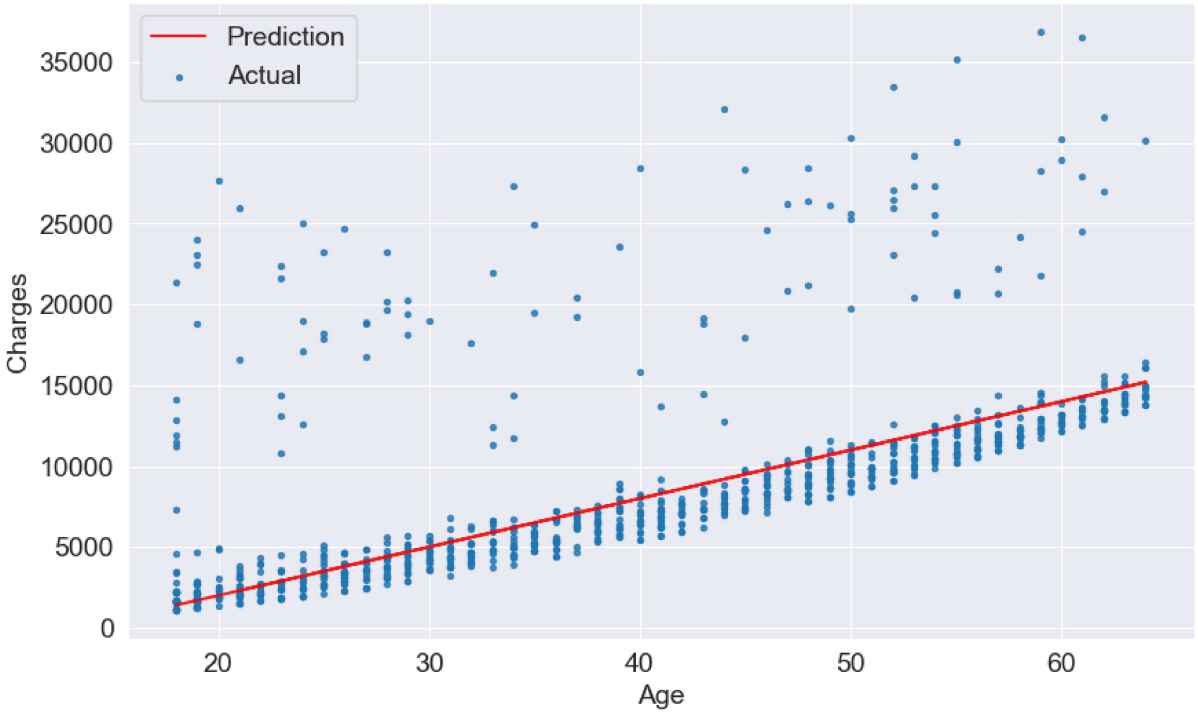
```
RMSE Loss:  4991.993804156945
```



In [126…

```
try_parameters(300, -4000)
```

```
RMSE Loss:  4725.9133994520325
```



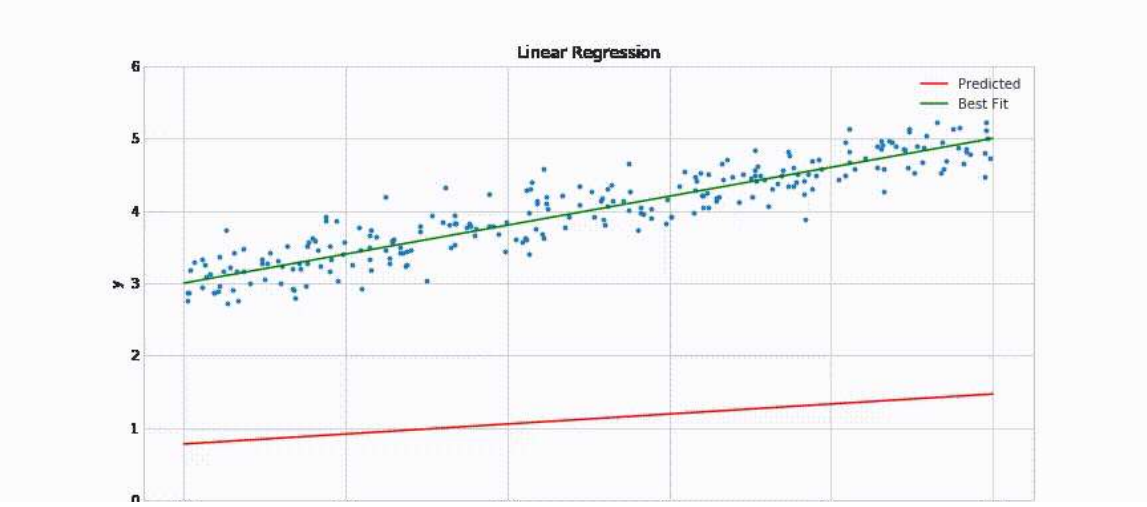
Optimizer

Next, we need a strategy to modify weights `w` and `b` to reduce the loss and improve the "fit" of the line to the data.

- Ordinary Least Squares: https://www.youtube.com/watch?v=szXbuO3bVRk (better for smaller datasets)
- Stochastic gradient descent: https://www.youtube.com/watch?v=sDv4f4s2SB8 (better for larger datasets)

Both of these have the same objective: to minimize the loss, however, while ordinary least squares directly computes the best values for `w` and `b` using matrix operations, while gradient descent uses a iterative approach, starting with a random values of `w` and `b` and slowly improving them using derivatives.

Here's a visualization of how gradient descent works:



Doesn't it look similar to our own strategy of gradually moving the line closer to the points?

Linear Regression using Scikit-learn

In practice, you'll never need to implement either of the above methods yourself. You can use a library like `scikit-learn` to do this for you.

In [127…
```
from sklearn.linear_model import LinearRegression
```

In [128…
```
# New Model object
model = LinearRegression()
```

Next, we can use the `fit` method of the model to find the best fit line for the inputs and targets.

In [129…
```
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.
```

In [130…
```
# Not that the input `X` must be a 2-d array, so we'll need to pass a dataframe, instead of

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

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

In [131…
```
model.fit(inputs, targets)
```

Out[131… `LinearRegression()`

In [132…
```
# We can now make predictions using the model. Let's try predicting the charges for the age

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

```
c:\Users\lenovo\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:

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

Out[132…
```
array([ 4055.30443855,  7796.78921819, 14210.76312614])
```

In [133…
```
# Let compute the predictions for the entire set of inputs

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

Out[133…
```
array([2719.0598744 , 5391.54900271, 6727.79356686, ..., 2719.0598744 ,
       2719.0598744 , 3520.80661289])
```

In [134…
```
inputs
```

Out[134…

	age
1	18
2	28
3	33
4	32
5	31
…	…
1332	52
1333	50
1334	18
1335	18
1336	21

1064 rows × 1 columns

In [135…
```
predictions
```

Out[135…
```
array([2719.0598744 , 5391.54900271, 6727.79356686, ..., 2719.0598744 ,
       2719.0598744 , 3520.80661289])
```

In [136…
```
targets
```

Out[136…
```
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
```

In [137…

```
# Let's compute the RMSE loss to evaluate the model.
rmse(targets, predictions)
```

Out[137… 4662.505766636391

Seems like our prediction is off by $4000 on average, which is not too bad considering the fact that there are several outliers.

In [138…

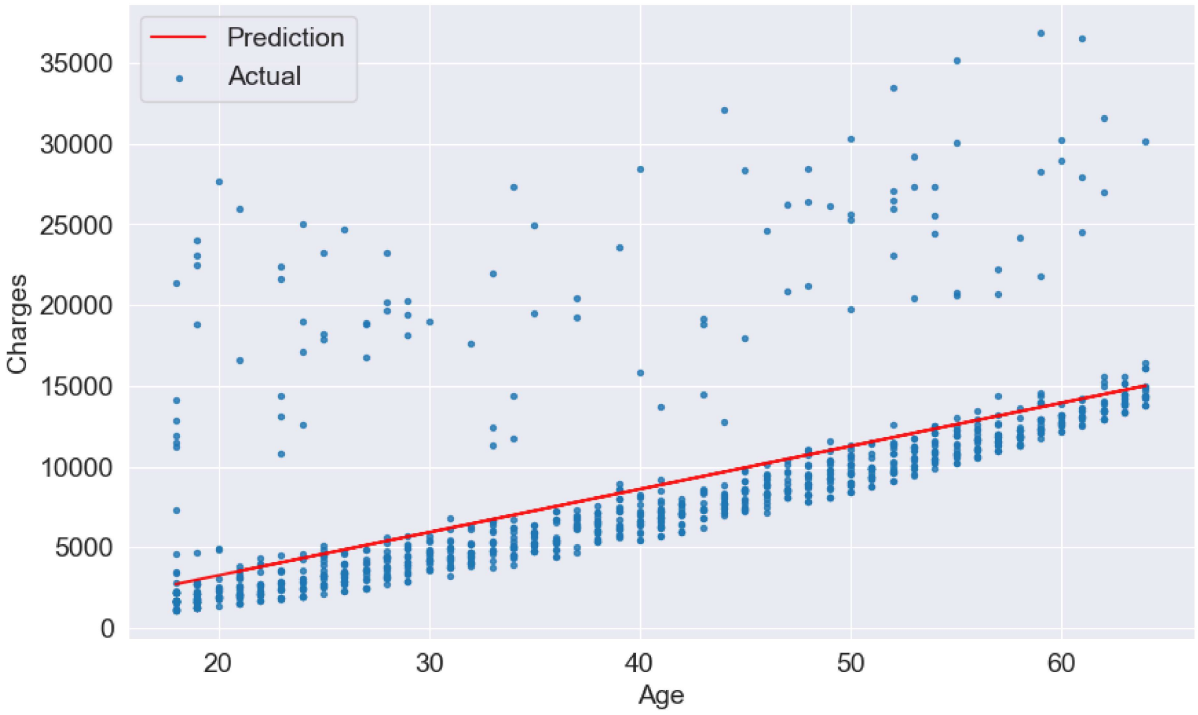
```
# The parameters of the model are stored in the `coef_` and `intercept_` properties.

# w
print("w : " , model.coef_)

# b
print("b :", model.intercept_)
```

```
w :  [267.24891283]
b : -2091.4205565650827
```

In [139…

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

```
RMSE Loss:  4662.505766636391
```



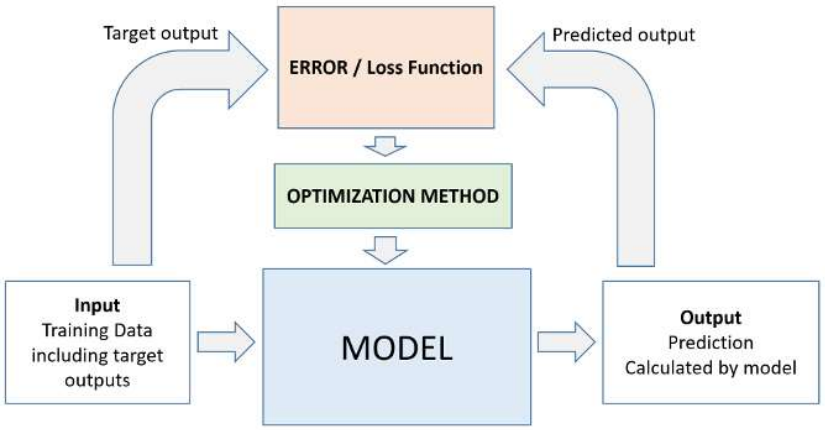
Machine Learning

Congratulations, you've just trained your first *machine learning model!* Machine learning is simply the process of computing the best parameters to model the relationship between some feature and targets.

Every machine learning problem has three components:

1. **Model**
2. **Cost Function** Parameter (cost function= Lower cost, best model)
3. **Optimizer**

We'll look at several examples of each of the above in future tutorials. Here's how the relationship between these three components can be visualized:



In [140…

```
# Create inputs and targets
inputs, targets = non_smoker_df[['age']], non_smoker_df['charges']

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 4662.505766636391
```

Linear Regression using Multiple Features

So far, we've used on the "age" feature to estimate "charges". Adding another feature like "bmi" is fairly straightforward. We simply assume the following relationship:

$$charges = w_1 \times age + w_2 \times bmi + b$$

We need to change just one line of code to include the BMI.

In [141…

```
# 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 evalute the model
loss = rmse(targets, predictions)
```

```
print('Loss:', loss)
```

```
Loss: 4662.312835461298
```

As you can see, adding the BMI doesn't seem to reduce the loss by much, as the BMI has a very weak correlation with charges, especially for non smokers.

In [142…
```
non_smoker_df.charges.corr(non_smoker_df.bmi)
```

Out[142…
```
0.08403654312833271
```

In [143…
```
fig = px.scatter(non_smoker_df, x='bmi', y='charges', title='BMI vs. Charges')
fig.update_traces(marker_size=5)
fig.show()
```

We can also visualize the relationship between all 3 variables "age", "bmi" and "charges" using a 3D scatter plot.

In [144…
```
fig = px.scatter_3d(non_smoker_df, x='age', y='bmi', z='charges')
fig.update_traces(marker_size=3, marker_opacity=0.5)
fig.show()
```

You can see that it's harder to interpret a 3D scatter plot compared to a 2D scatter plot. As we add more features, it becomes impossible to visualize all feature at once, which is why we use measures like correlation and loss.

Let's also check the parameters of the model.

In [145…
```
model.coef_, model.intercept_
```

Out[145…
```
(array([266.87657817,   7.07547666]), -2293.6320906488672)
```

Clearly, BMI has a much lower weightage, and you can see why. It has a tiny contribution, and even that is probably accidental. This is an important thing to keep in mind: you can't find a relationship that doesn't exist, no matter what machine learning technique or optimization algorithm you apply.

Let's go one step further, and add the final numeric column: "children", which seems to have some correlation with "charges".

$$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + b$$

In [146…
```
non_smoker_df.charges.corr(non_smoker_df.children)
```

Out[146…
```
0.13892870453542186
```

In [147…
```
# Stripe Plot
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.show()
```

In [148…

```
# 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 evalute the model
loss = rmse(targets, predictions)
print('Loss:', loss)
```

```
Loss: 4608.470405038245
```

Once again, we don't see a big reduction in the loss, even though it's greater than in the case of BMI.

To train a linear regression model to estimate medical charges for all customers.

In [149…

```
# Create inputs and targets
inputs, targets = acme_data[['age', 'bmi', 'children']], acme_data['charges']

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 11355.317901125969
```

Using Categorical Features for Machine Learning

So far we've been using only numeric columns, since we can only perform computations with numbers. If we could use categorical columns like "smoker", we can train a single model for the entire dataset.

To use the categorical columns, we simply need to convert them to numbers. There are three common techniques for doing this:

1. If a categorical column has just two categories (it's called a binary category), then we can replace their values with 0 and 1.
2. If a categorical column has more than 2 categories, we can perform one-hot encoding i.e. create a new column for each category with 1s and 0s.
3. If the categories have a natural order (e.g. cold, neutral, warm, hot), then they can be converted to numbers (e.g. 1, 2, 3, 4) preserving the order. These are called ordinals.

Binary Categories

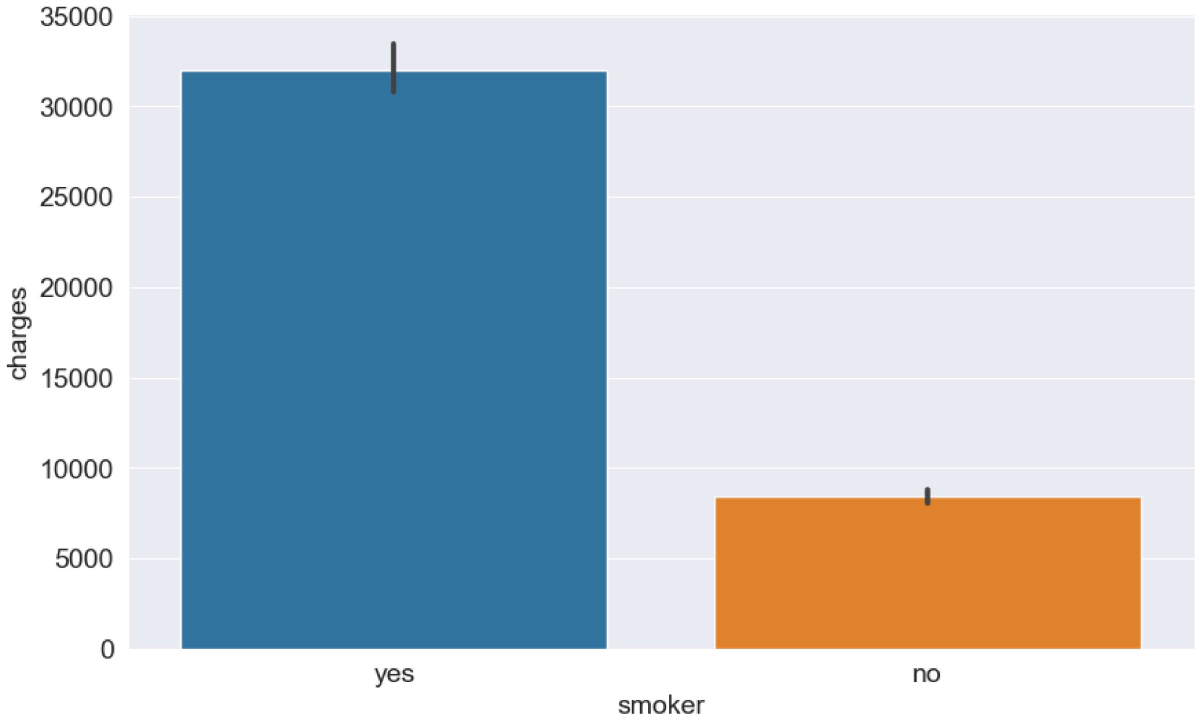
The "smoker" category has just two values "yes" and "no". Let's create a new column "smoker_code" containing 0 for "no" and 1 for "yes".

In [150…

```
sns.barplot(data=acme_data, x='smoker', y='charges')
```

Out[150…

```
<AxesSubplot:xlabel='smoker', ylabel='charges'>
```



In [151…

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

In [152…

```
acme_data.charges.corr(acme_data.smoker_code)
```

Out[152…

```
0.7872514304984785
```

We can now use the `smoker_df` column for linear regression.

$$charges = w_1 \times age + w_2 \times bmi + w_3 \times children + w_4 \times smoker + b$$

In [153…

```
# Create inputs and targets
inputs, targets = acme_data[['age', 'bmi', 'children', 'smoker_code']], acme_data['charges'

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 6056.439217188077
```

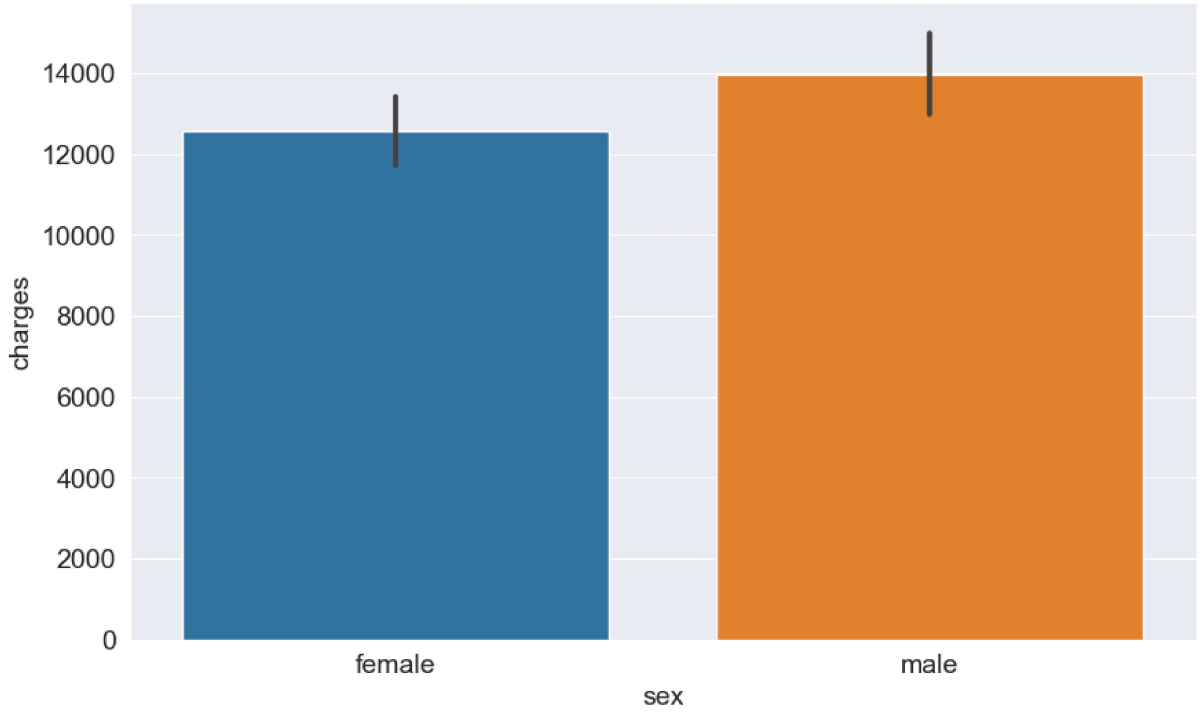
The loss reduces from `11355` to `6056`, almost by 50%! This is an important lesson: never ignore categorical data.

Let's try adding the "sex" column as well.

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

In [155…
```
sns.barplot(data=acme_data, x='sex', y='charges')
```

Out[155… <AxesSubplot:xlabel='sex', ylabel='charges'>



In [156…
```
sex_codes = {'female': 0, 'male': 1}
acme_data['sex_code'] = acme_data.sex.map(sex_codes)

# Correaltion btw charges and sex
acme_data.charges.corr(acme_data.sex_code)
```

Out[156… 0.05729206220202527

In [157…
```
# Create inputs and targets
inputs, targets = acme_data[['age', 'bmi', 'children', 'smoker_code', 'sex_code']], acme_da

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 6056.100708754538
```

As you might expect, this does have a significant impact on the loss.

One-hot Encoding

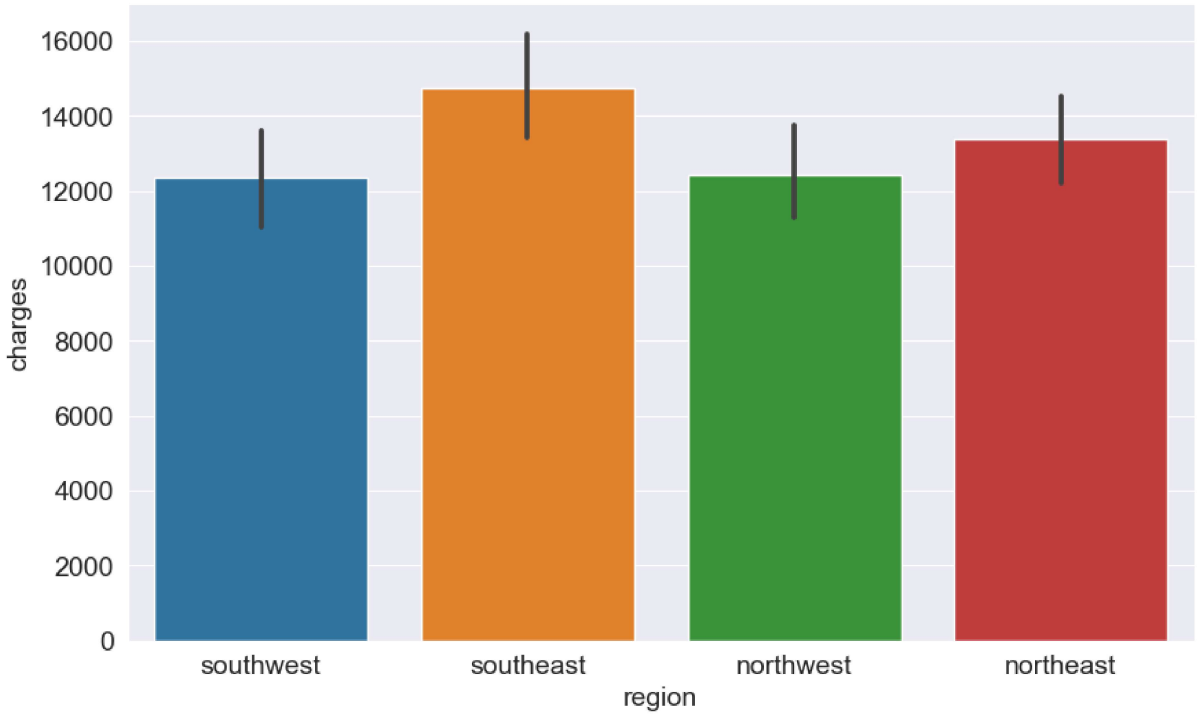
The "region" column contains 4 values, so we'll need to use hot encoding and create a new column for each region.

1	Cat A
2	Cat B
3	Cat C

→

1	1	0	0
2	0	1	0
3	0	0	1

In [158…
```
sns.barplot(data=acme_data, x='region', y='charges');
```



In [159…
```
from sklearn import preprocessing

enc = preprocessing.OneHotEncoder()
enc.fit(acme_data[['region']])
enc.categories_
```

Out[159… `[array(['northeast', 'northwest', 'southeast', 'southwest'], dtype=object)]`

In [160…
```
one_hot = enc.transform(acme_data[['region']]).toarray()
one_hot
```

Out[160…
```
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.]])
```

In [161…
```
acme_data[['northeast', 'northwest', 'southeast', 'southwest']] = one_hot
```

In [162…
```
acme_data
```

Out[162…

Out[162…

	age	sex	bmi	children	smoker	region	charges	smoker_code	sex_code	northi
0	19	female	27.900	0	yes	southwest	16884.92400	1	0	
1	18	male	33.770	1	no	southeast	1725.55230	0	1	
2	28	male	33.000	3	no	southeast	4449.46200	0	1	
3	33	male	22.705	0	no	northwest	21984.47061	0	1	
4	32	male	28.880	0	no	northwest	3866.85520	0	1	
…	…	…	…	…	…	…	…	…	…	
1333	50	male	30.970	3	no	northwest	10600.54830	0	1	
1334	18	female	31.920	0	no	northeast	2205.98080	0	0	
1335	18	female	36.850	0	no	southeast	1629.83350	0	0	
1336	21	female	25.800	0	no	southwest	2007.94500	0	0	
1337	61	female	29.070	0	yes	northwest	29141.36030	1	0	

1338 rows × 13 columns

Let's include the region columns into our linear regression model.

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

In [163…

```
# Create inputs and targets
input_cols = ['age', 'bmi', 'children', 'smoker_code', 'sex_code', 'northeast', 'northwest'
inputs, targets = acme_data[input_cols], acme_data['charges']

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 6041.679651174453
```

Once again, this leads to a fairly small reduction in the loss.

Model Improvements

Let's discuss and apply some more improvements to our model.

Feature Scaling

Recall that due to regulatory requirements, we also need to explain the rationale behind the predictions our model.

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

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

In [164…
```
model.coef_
```

Out[164…
```
array([  256.85635254,   339.19345361,   475.50054515, 23848.53454191,
        -131.3143594 ,   587.00923503,   234.0453356 ,  -448.01281436,
        -373.04175627])
```

In [165…
```
model.intercept_
```

Out[165…
```
-12525.547811195444
```

In [166…
```
weights_df = pd.DataFrame({
    'feature': np.append(input_cols, 1),
    'weight': np.append(model.coef_, model.intercept_)
})
weights_df
```

Out[166…

	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

While it seems like BMI and the "northeast" have a higher weight than age, keep in mind that the range of values for BMI is limited (15 to 40) and the "northeast" column only takes the values 0 and 1.

Because different columns have different ranges, we run into two issues:

1. We can't compare the weights of different column to identify which features are important
2. 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 (or standardize) the values in numeric column by subtracting the mean and dividing by the standard deviation.

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\frac{1}{N}\sum$$

$$\mu = \frac{1}{N}\sum_{i=1}^{}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

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

In [167…
```
from sklearn.preprocessing import StandardScaler

numeric_cols = ['age', 'bmi', 'children']
scaler = StandardScaler()
scaler.fit(acme_data[numeric_cols])
print(scaler.mean_)
print(scaler.var_)
```

```
[39.20702541 30.66339686  1.09491779]
[197.25385199  37.16008997   1.45212664]
```

In [168…
```
scaled_inputs = scaler.transform(acme_data[numeric_cols])
scaled_inputs
```

Out[168…
```
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]])
```

In [169…
```
cat_cols = ['smoker_code', 'sex_code', 'northeast', 'northwest', 'southeast', 'southwest']
categorical_data = acme_data[cat_cols].values
```

In [170…
```
inputs = np.concatenate((scaled_inputs, categorical_data), axis=1)
targets = acme_data.charges

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

# Generate predictions
predictions = model.predict(inputs)

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

```
Loss: 6041.679651174457
```

We can now compare the weights in the formula:

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

In [171…
```
weights_df = pd.DataFrame({
    'feature': np.append(numeric_cols + cat_cols, 1),
    'weight': np.append(model.coef_, model.intercept_)
})
```

```
})
weights_df.sort_values('weight', ascending=False)
```

Out[171…

	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

As you can see now, the most important feature are:

1. Smoker
2. Age
3. BMI

Creating a Test Set

Models like the one we've created in this tutorial are designed to be used in the real world. It's common practice to set aside a small fraction of the data (e.g. 10%) just for testing and reporting the results of the model.

In [172…

```
from sklearn.model_selection import train_test_split
inputs_train, inputs_test, targets_train, targets_test = train_test_split(inputs, targets,
```

In [173…

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

# Generate predictions
predictions_test = model.predict(inputs_test)

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

```
Test Loss: 6495.514915036123
```

In [174…

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

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

```
Training Loss: 5991.788627367449
```