

DATA SOCIETY®

Regression - Day 1

*"One should look for what is and not what he thinks should be."
-Albert Einstein.*

Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	
Evaluate data using basic statistics such as summary statistics, covariance, correlation	
Explain hypothesis tests as needed for regression	
Prepare the dataset to run a linear regression model	
Implement and run the linear regression model on the given data	
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

Import packages

- Let's import the libraries we will be using today

```
import os
import pandas as pd
import numpy as np
import pickle

# New today - we will introduce it when we use it.
import matplotlib.pyplot as plt
from sklearn.feature_selection import VarianceThreshold
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).  
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).  
main_dir = "/Users/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).  
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and  
# remainder of the path to data directory.  
data_dir = main_dir + "/data"
```

Working directory

- Set working directory to the `data_dir` variable we set
- We do this using the `os.chdir` function, change directory
- We can then check the working directory using `.getcwd()`
- For complete documentation of the `os` package, [click here](#)

```
# Set working directory.  
os.chdir(data_dir)
```

```
# Check working directory.  
print(os.getcwd())
```

```
/home/[user-name]/af-werx/data
```

What questions can linear regression answer?

- Some questions that linear regression will help answer are:
 - How much does a *single factor* account for a final outcome?
 - How do *several variables together* account for a final outcome?
 - What set of variables together will best explain the final outcome?
- Today, we will start to understand how to answer those questions!

Motivation for linear regression

- Why would managers want to know how factors affect a final outcome?
 - To provide better quality products based on adjusting the factors
 - To anticipate maintenance expenditures and decrease costs by understanding each aspect of the final product

Linear regression: measuring relationships

- A linear relationship between 2 variables is:

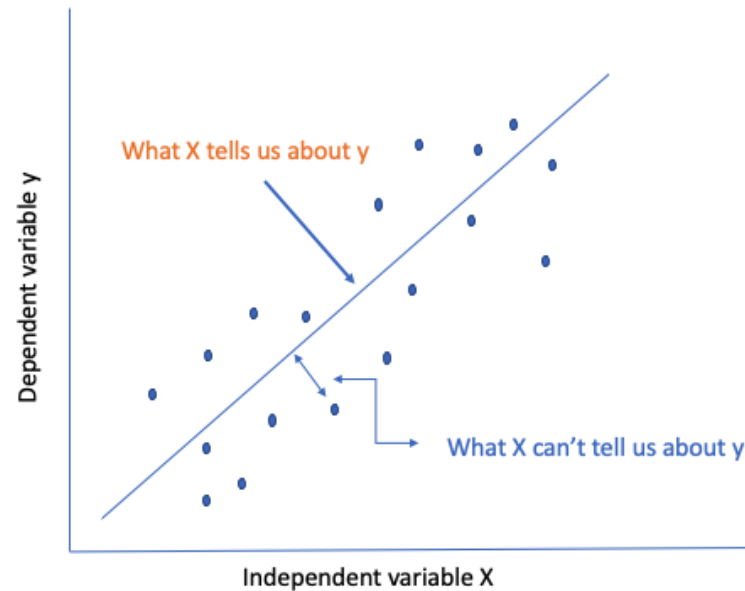
$$y = mx + b$$

$$m = \frac{\text{Change..in..}y}{\text{Change..in..}x} = \text{slope}$$

- y : variable 1 - dependent variable, what you want to predict - target variable
- x : variable 2 - independent variable, what you want to use to predict - predictor
- m : rate of change (slope)
- b : value of y when $x = 0$
- Linear regression is a **supervised learning method**
 - We have at least one predictor (x)
 - We have one target variable (y)


Linear regression: measuring relationships

- Linear regression is the first model many data scientists learn about
- It illustrates the basis of many more complex predictive models



- The main purpose is to find **linear trends** in your data so you can **predict outcomes!**

Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	
Evaluate data using basic statistics such as summary statistics, covariance, correlation	
Explain hypothesis tests as needed for regression	
Prepare the dataset to run a linear regression model	
Implement and run the linear regression model on the given data	
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

The statistics behind regression

- Throughout learning, executing, and analyzing linear regression, we will refer to various statistical concepts
 - Summary statistics
 - Correlation
 - Covariance
 - T-test, f-test, and p-value
- We are going to review these concepts now so that we can refer to them easily later on

Datasets for regression

- **We will be using two datasets total, we discussed each of the datasets already**
- **One dataset in class, to learn the concepts**
 - Temperature - heart rate dataset
- **One dataset for our in-class exercises**
 - Fast food dataset

Temp heart rate dataset

- We read in the temp_heart_rate dataset that we worked with before

```
# This dataset is of type dataframe. Let's assign this dataset to a variable, so that we can manipulate it freely.  
temp_heart_rate = pd.read_csv('temp_heart_rate.csv')
```

```
print(type(temp_heart_rate))  #<- a pandas dataframe!
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
print(len(temp_heart_rate))  #<- returns the number of rows
```

```
130
```

Temp heart rate dataset

```
# You can also save the shape of the dataframe as 2 variables  
# (since the returned object is a tuple with 2 values).  
nrows, ncols = temp_heart_rate.shape  
print(nrows)  
print(ncols)
```

```
130
```

```
3
```

Temp heart rate dataset - regression

- For the purpose of single variable regression, we are going to subset the dataset
- We will keep Body Temp and Heart Rate in the new dataset
 - Body Temp: body temperature of the individual
 - Heart Rate: heart rate of the individual

```
# Subset the two variables for single variable regression.  
temp_heart_regression = temp_heart_rate[['Body Temp', 'Heart Rate']]
```

- The purpose is to see if **there is a relationship between these two variables** and the **type of relationship** that exists

Pickle - what?

- We are going to pause for a moment to learn about the library `pickle`
- When we have objects we want to carry over and do not want to rerun code, we can `pickle` these objects
- In other words, `pickle` will help us save objects from one script/session and pull them up in new scripts
- How do we do that? We use a function in Python called `pickle`
- It is similar to **flattening** a file
 - **Pickle/saving: a Python object is converted into a byte stream**
 - **Unpickle/loading: the inverse operation where a byte stream is converted back into an object**



Implementing `pickle`

- Right now, we want to pickle `temp_heart_regression` for use in upcoming sessions

```
pickle.dump(temp_heart_regression, open("temp_heart_regression.sav", "wb" ))
```

Summary statistics

- Summary statistics help us understand our data better
- Here are the summary statistics that we'll review:

Summary statistic	Definition
Count	the count of non-null / NA observations
Mean	the central value or average of the dataset / vector
Standard deviation	the standard deviation of the observations
Min	the minimum number in the vector
1st quartile (25%)	the median of the lower half of the dataset / vector
3rd quartile (75%)	the median of the higher half of the dataset / vector
Max	the maximum number in the vector

Summary statistics: Body Temp and Heart Rate

Numerical summary: Body Temp

```
# Describe the `Body Temp` summary statistics.  
print(temp_heart_regression['Body  
Temp'].describe())
```

```
count      130.000000  
mean       98.249231  
std        0.733183  
min        96.300000  
25%        97.800000  
50%        98.300000  
75%        98.700000  
max        100.800000  
Name: Body Temp, dtype: float64
```

Numerical summary: Heart Rate

```
# Describe the `Heart Rate` summary statistics.  
print(temp_heart_regression['Heart  
Rate'].describe())
```

```
count      130.000000  
mean       73.761538  
std        7.062077  
min        57.000000  
25%        69.000000  
50%        74.000000  
75%        79.000000  
max        89.000000  
Name: Heart Rate, dtype: float64
```

Summary statistics: covariance

$$\text{Cov}(X, Y) = \frac{\sum_{i=1}^N (x_i - \text{mean}_x)(y_i - \text{mean}_y)}{(N - 1)}$$

Covariance of two variables (x, y) is a measure of **association between x and y**

- It is **positive** if y **increases with increasing** x
- It is **negative** if y **decreases with increasing** x
- It is **zero** if there is **no linear tendency** for y to change with x

Covariance is very **sensitive** to the scale of the two variables, which makes comparing covariance across variables **very difficult**



Summary statistics: covariance in Python

We can easily calculate covariance in Python

```
print(temp_heart_regression.cov())
```

	Body Temp	Heart Rate
Body Temp	0.537558	1.313381
Heart Rate	1.313381	49.872928

- The value of the covariance can be very large, simply because the data we are using (Body Temp) has a large value
- Thankfully, we can use correlation to determine relationships instead - scaled covariance

Summary statistics: correlation is scaled cov

We can use **correlation**:

$$r_{xy} = \frac{Cov(X, Y)}{\sqrt{var(X)var(Y)}}$$

This formula controls for the scale of the variables and guarantees that **correlation is always between -1 and +1**

We can easily calculate correlation in Python

```
temp_heart_regression.corr()
```

	Body Temp	Heart Rate
Body Temp	1.000000	0.253656
Heart Rate	0.253656	1.000000

Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	
Prepare the dataset to run a linear regression model	
Implement and run the linear regression model on the given data	
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

Hypothesis testing

There are a few test statistics that are used in regression - the **t-test** and the **f-test**:

T-tests are used to compare *two related samples*

- In regression, t-tests are used to test for a relationship between variables

F-tests are used to test the *equality of two populations*

- In regression, f-tests are used to test for if a model was found by chance



T-test vs F-test

T-test

- T-tests are used to compare **two populations**
- We look for this by using the **means** of the populations
- The t-test tells you if the **difference is significant** or not
 - To help us determine significance, we can look at the p-value of the t-test

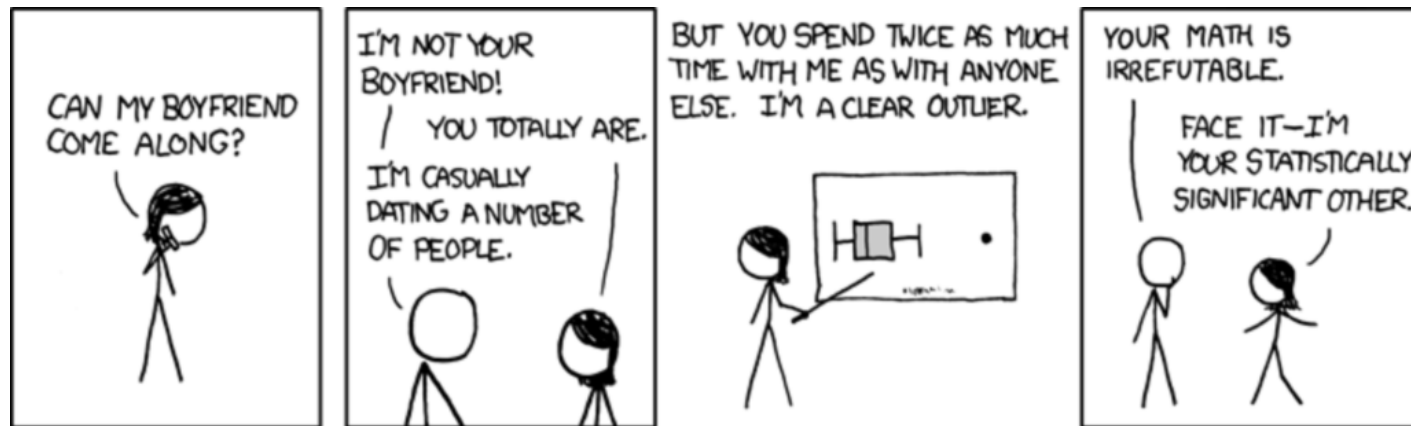
F-test

- F-tests are used to compare **two variances**
- We look for this by taking **two variances** and dividing them
 - If the variances are equal, the ratio of the variances will be 1
- The f-test tells you if the **variances of the populations are equal** or not

T-test and p-value in regression

In regression, a t-test is used to test the statistical **significance** of a *regression coefficient*

- Each t-test (usually) starts with the default *null* hypothesis that the relationship between a set of variables is **random** (coefficient = zero)
- p-values are a **probability**
 - a p-value is the probability of observing the given result if the null hypothesis is **true**
 - a small p-value will usually mean that you **reject** the default null hypothesis

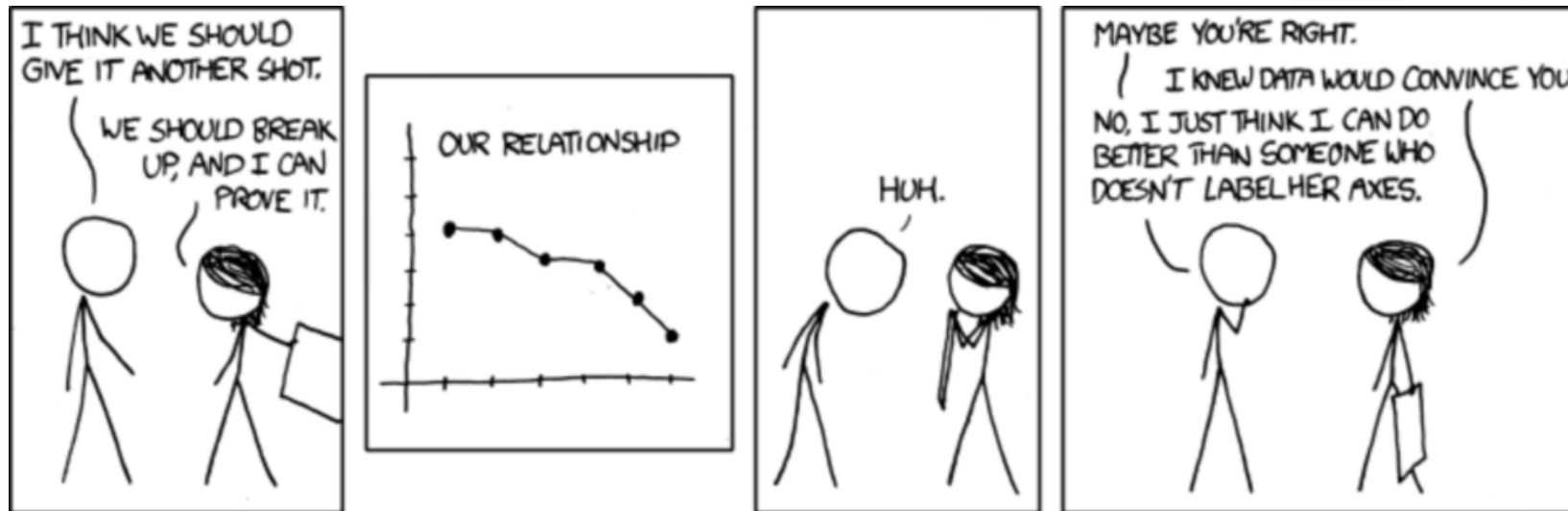


Comic Strip by: XKCD, From: <https://xkcd.com/539/>, Creative Commons, CC-BY-NC 2.5

F-test and p-value in regression

In regression, F-tests are used to test if the **model** was **found by chance**

- We look for this by comparing **a model with no predictors** to the **model that you specify**
- The f-test tells you if the fit of the model with no predictors is equal to the fit of your model
 - To help us determine significance, we can once again look at the **p-value** of the **f-test**
 - This time, we are looking at the significance of the **model** instead of the significance of **one coefficient**



Comic Strip by: XKCD From: <https://xkcd.com/833/> Creative Commons, CC-BY-NC 2.5

Knowledge Check 1



Exercise 1



Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	✓
Prepare the dataset to run a linear regression model	
Implement and run the linear regression model on the given data	
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

Linear regression: Temp heart rate data

- We will be using the temp heart rate data that we subsetting for simple linear regression

```
# Let's look at the data we will be working with.  
print(temp_heart_regression.head())
```

	Body Temp	Heart Rate
0	96.3	70
1	96.7	71
2	96.9	74
3	97.0	80
4	97.1	73

- We will be predicting Body Temp, this is our target variable (y)
- We will using Heart Rate to predict Body Temp, this is our predictor (x)

Linear regression: EDA

- We subset our dataset earlier, now let's get it ready to run a linear regression
- First, let's conduct simple exploratory data analysis (EDA) to understand our data better
- **EDA is a process used to understand your data**, and it's a best practice to perform EDA before diving into cleaning and wrangling your data for the modeling stage
- **EDA usually consists of various visualizations of:**
 - individual variables
 - interactions of predictor variables
 - interactions of predictor variables with the target
 - unsupervised learning methods such as clustering

Linear regression: EDA

- Today, we will be using **two visualizations for EDA**
- Scatterplot of the two variables, Body Temp and Heart Rate
 - This will show us the relationship between the predictor and target variable
 - We can eyeball it and see if there is no chance of linear relationship or if it is worth investigating further
- Histogram of each variable, Body Temp and Heart Rate
 - This will show us the distribution of each individual variable
 - We can look for highly skewed data, outliers, and missing values
 - This is helpful in identifying if we may need transformations when cleaning the data

Visualizing data with matplotlib for EDA



- `matplotlib` is a popular plotting library among scientists and data analysts
- It is one of the older Python plotting libraries, and for this reason, it has become quite flexible and *well-documented*
- Other plotting libraries you may come across are Seaborn (which is built on `matplotlib`), ggplot (the Python version of the popular R plotting library), Plotly, Bokeh, and many others
- Pandas also comes with some plotting capabilities, and these are actually just based on `matplotlib`
- You can begin to explore the different types of plots you can create with `matplotlib` by browsing their *gallery*

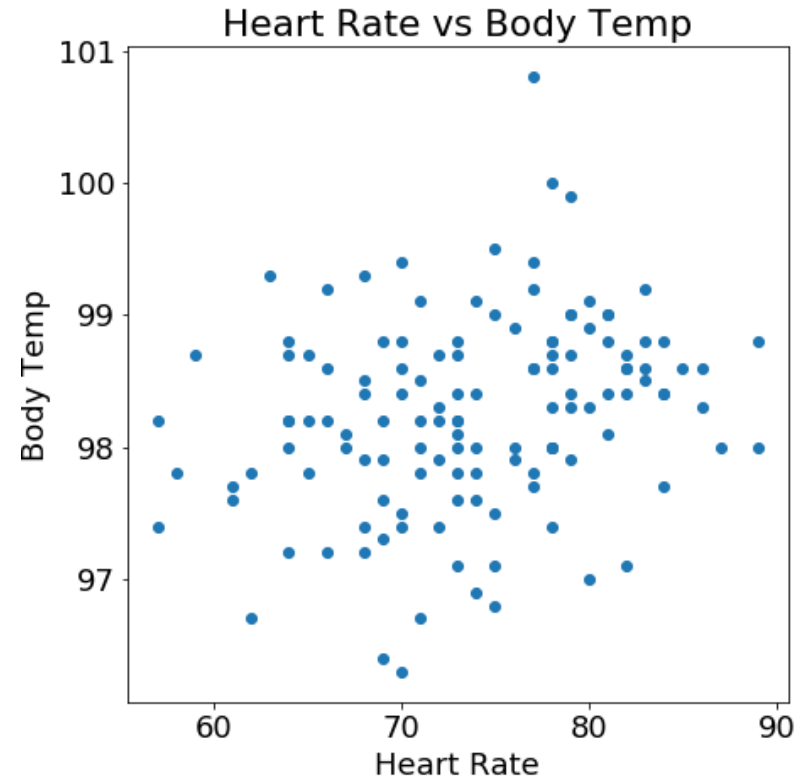
Importing matplotlib

- As with any other package in Python, in order to use it, we need to import it
- We'll import `pyplot` as `plt` so that we can call `plt.[any_function]()` with appropriate arguments to create a plot
- The `pyplot` module of the `matplotlib` library is a very big and diverse set of functions that allow to create any visualization out there!
- See documentation on `pyplot` [here](#)

EDA: scatterplot

- We will use `plt`, our alias for `matplotlib`, to build both the scatterplot and histogram
- In this module, we want to predict Body Temp based on Heart Rate
- In this visualization, we are looking at the interaction of the two variables
- **What do you think when you see the interaction of the two variables?**

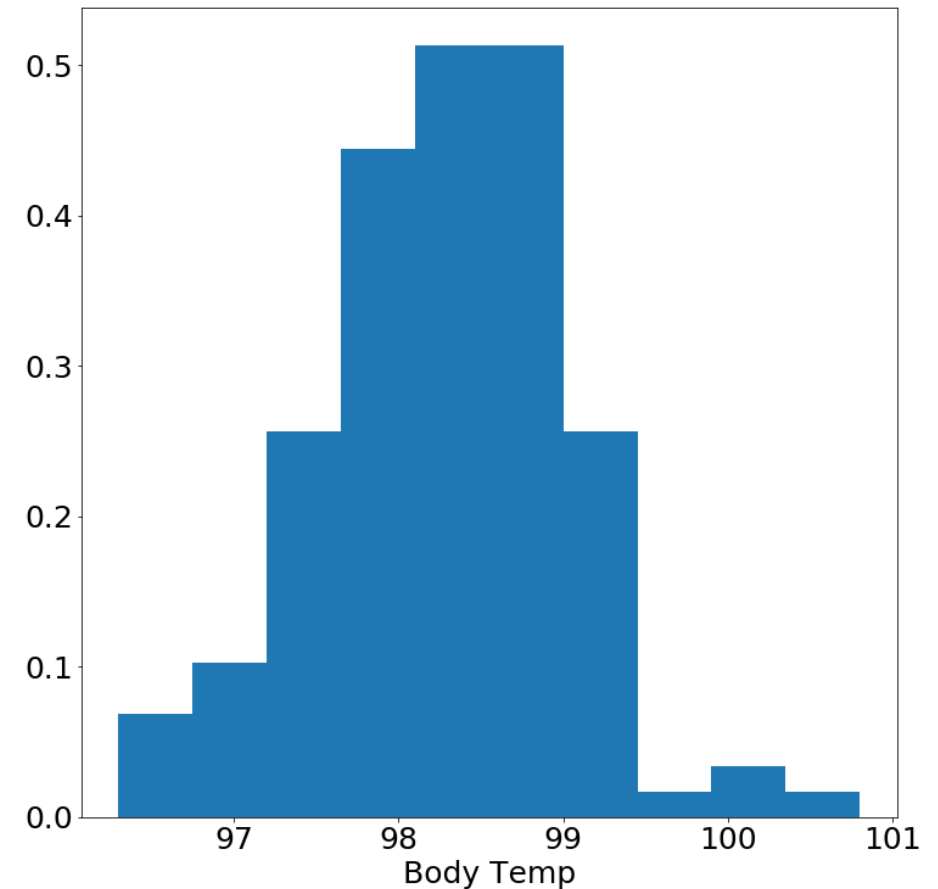
```
# Make scatterplot.  
plt.scatter(temp_heart_regression['Heart Rate'],  
            temp_heart_regression['Body Temp'])  
plt.title("Heart Rate vs Body Temp")  
plt.xlabel("Heart Rate")  
plt.ylabel("Body Temp")  
plt.show()
```



EDA: histogram `Body Temp`

- We will use `plt` to build a histogram
- In this visualization, we are looking at the distribution of `Body Temp`

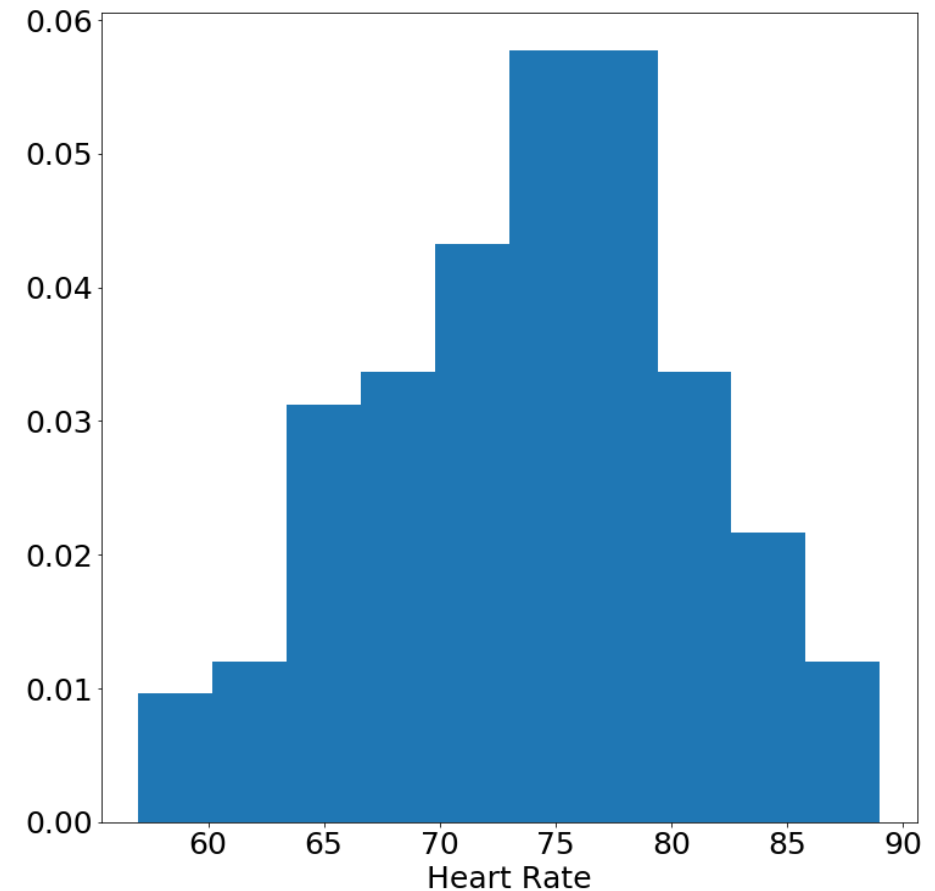
```
plt.hist(temp_heart_regression['Body Temp'],  
         normed = True, bins = 10)  
plt.xlabel('Body Temp')
```



EDA: histogram `Heart Rate`

- We will use `plt` to now build a histogram
- In this visualization, we are looking at the distribution of Heart Rate
- **What observations can we make about Heart Rate?**

```
plt.hist(temp_heart_regression['Heart Rate'],  
         normed = True, bins = 10)  
plt.xlabel('Heart Rate')
```



Linear regression: data cleaning

- So far, we have reviewed what we need to do to get our data ready for a model
- This is where we start incorporating it
- In linear regression, before actually running the regression, we look for:
 - **NAs**: since simple linear regression relies on numerical computations, NAs can produce non-numeric results
 - **Near zero variance (NZV)**: zero variance variables are variables with a single value. They are not predictive as the predictor variable's value is always the same

Data cleaning: fillna()

- First, let us look for NAs in our dataset
- We will be using a pandas built in function `fillna()`
- This function will allow us to fill NA values as well as choose what we want to impute them with
- Today, we will use the `mean()` to fill NA values
- There are many more complex ways to impute NAs however that is outside the scope of this course

pandas.DataFrame.fillna()

`DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, **kwargs)`

[\[source\]](#)

Fill NA/NaN values using the specified method.

Data cleaning: NAs

- First, we check how many NAs there are in each column

```
# Check how many values are null in the `Body Temp` column.  
print(temp_heart_regression['Body Temp'].isnull().sum())
```

0

```
# Check how many values are null in the `Heart Rate` column.  
print(temp_heart_regression['Heart Rate'].isnull().sum())
```

0

Data cleaning: NAs

- We see that we have no column that we will have to impute
- However, if there were NAs in Body Temp, then we could also check in which rows the NA values were in

```
print(temp_heart_regression[temp_heart_regression['Body Temp'].isnull()])
```

```
Empty DataFrame  
Columns: [Body Temp, Heart Rate]  
Index: []
```

Data cleaning: use fillna()

- Now, we'll use the pandas built-in function `fillna` and substitute NAs in `Body Temp` with the mean

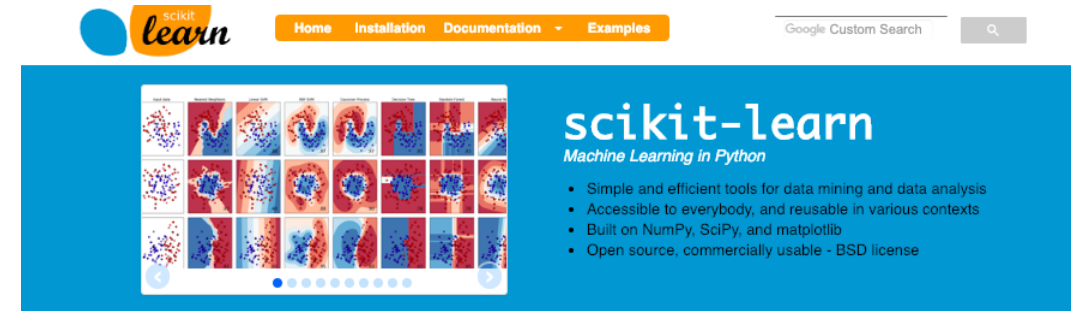
```
# Set the dataframe equal to the imputed dataset.  
temp_heart_regression = temp_heart_regression.fillna(temp_heart_regression.mean())  
# Check how many values are null in the `Body Temp` column.  
print(temp_heart_regression['Body Temp'].isnull().sum())
```

0

- We see that there aren't any NAs in `Body Temp`
- Now we move forward to look for zero variance

Introduce packages: sklearn

- We will be using `sklearn` throughout the next few weeks
- It is a powerful machine learning library in Python
- For complete documentation, go to <https://scikit-learn.org/stable/index.html>
- We will be using a function from `sklearn` for determining near zero variance



Data cleaning: near zero variance

- We are concerned with variables that have zero variance because they will not be helpful in prediction
- This is especially useful when you have high-dimension datasets
- We can use the function `VarianceThreshold` from `sklearn.feature_selection`

sklearn.feature_selection.VarianceThreshold

```
class sklearn.feature_selection.VarianceThreshold(threshold=0.0) ¶
```

[\[source\]](#)

«

Feature selector that removes all low-variance features.

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Read more in the [User Guide](#).

Parameters:	threshold : float, optional Features with a training-set variance lower than this threshold will be removed. The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.
Attributes:	variances_ : array, shape (n_features,) Variances of individual features.

Data cleaning: testing for near zero variance

- Now, let's run `VarianceThreshold` on the dataset

```
# Using sklearn, look for low variance within the columns.  
# We instantiate the function.  
selector = VarianceThreshold()  
# Name the cleaned dataset temp_heart_regression_clean.  
temp_heart_regression_clean = selector.fit_transform(temp_heart_regression)  
# Let's see if the dimensions changed.  
print(temp_heart_regression_clean.shape)
```

```
(130, 2)
```

- We see that the columns remained in the dataset
- Therefore, we can confirm that we do not have an issue with either variable having zero variance

Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	✓
Prepare the dataset to run a linear regression model	✓
Implement and run the linear regression model on the given data	
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

Linear regression: statsmodel

- There are many ways to perform linear regression in Python
- Two popular package to use are *sklearn's linear_model* and statsmodel
- Today, we will use statsmodel because we are able to dive deeper into the evaluation of the model with this package

statsmodels.regression.linear_model.OLS

```
class statsmodels.regression.linear_model.OLS(endog, exog=None, missing='none', hasconst=None, **kwargs)
```

[\[source\]](#)

A simple ordinary least squares model.

Parameters:

- **endog** (*array-like*) – 1-d endogenous response variable. The dependent variable.
- **exog** (*array-like*) – A *nobs* x *k* array where *nobs* is the number of observations and *k* is the number of regressors. An intercept is not included by default and should be added by the user. See `statsmodels.tools.add_constant`.
- **missing** (*str*) – Available options are 'none', 'drop', and 'raise'. If 'none', no nan checking is done. If 'drop', any observations with nans are dropped. If 'raise', an error is raised. Default is 'none.'
- **hasconst** (*None or bool*) – Indicates whether the RHS includes a user-supplied constant. If True, a constant is not checked for and *k_constant* is set to 1 and all result statistics are calculated as if a constant is present. If False, a constant is not checked for and *k_constant* is set to 0.

weights

scalar – Has an attribute `weights = array(1.0)` due to inheritance from WLS.

Implement linear regression

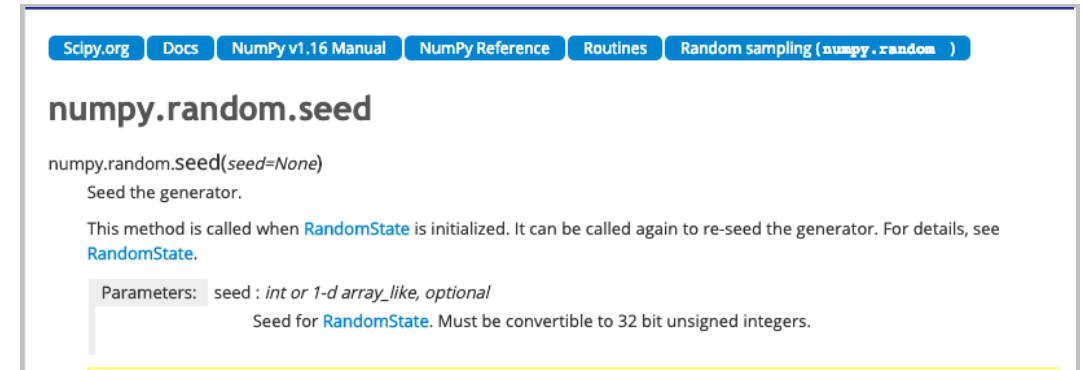
- To implement linear regression, we first need to separate the dependent and independent variable(s)
- In this case, there is **one independent variable** and **one dependent variable**

```
# Two variables for single regression.  
X = pd.DataFrame(temp_heart_regression_clean[:,1]) # independent variable  
# Make sure to add the constant term so that we have a column for the intercept.  
X = sm.add_constant(X)  
y = pd.DataFrame(temp_heart_regression_clean[:,0]) # dependent variable
```

Setting the seed

- Before we build our first model, make sure to set a seed
- This will set the **same random state** each time the model is run
- If you do not set the seed, your results will be a bit different every time, and might throw of reproducibility of your code / results in case someone else wants to run the analysis
- We will use numpy's `np.random.seed`

```
np.random.seed(1)
```



Implement: build a linear model

- We'll use `sm.OLS`, which is ordinary least squares or **simple linear regression**

```
# Build the model, note the difference in
argument order.
model = sm.OLS(y, X).fit()
```

- We will review the many components of this summary soon, here is a look at the output:

```
# Inspect the output of the `sm.OLS` function.
print(model.summary())
```

```
=====
                        OLS Regression Results
=====
Dep. Variable:          Body Temp      R-squared:                0.064
Model:                  OLS            Adj. R-squared:           0.057
Method:                 Least Squares   F-statistic:              8.802
Date:                  Mon, 22 Jul 2019 Prob (F-statistic):       0.00359
Time:                  11:44:37         Log-Likelihood:          -139.29
No. Observations:      130             AIC:                   282.6
Df Residuals:          128             BIC:                   288.3
Df Model:               1
Covariance Type:       nonrobust
=====
```

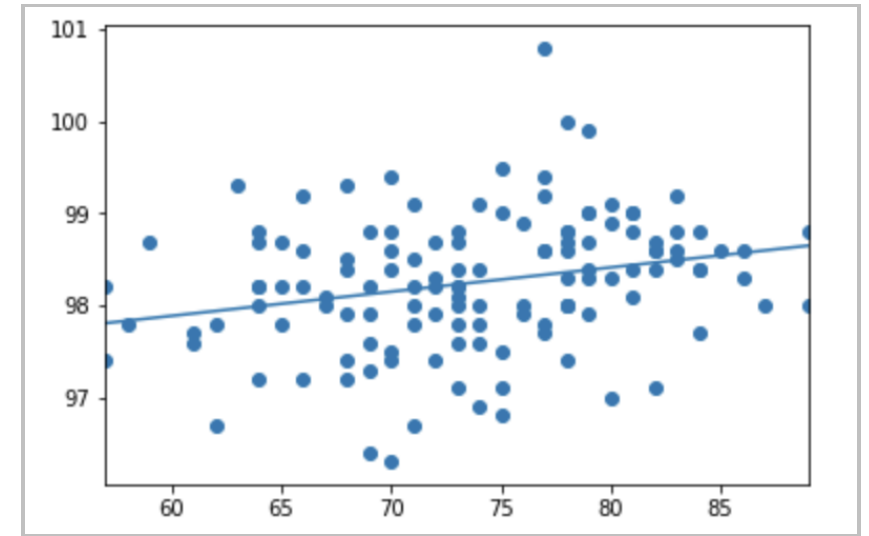
	coef	std err	t	P> t	[0.025	0.975]
const	96.3068	0.658	146.429	0.000	95.005	97.608
Heart Rate	0.0263	0.009	2.967	0.004	0.009	0.044

```
=====
Omnibus:                 2.917      Durbin-Watson:           0.278
Prob(Omnibus):            0.233      Jarque-Bera (JB):        2.986
Skew:                    0.051      Prob(JB):                0.225
Kurtosis:                3.735      Cond. No.                781.
=====
```

Implement: plot the fit

- We've already created a scatterplot for the two variables
- We can now add our linear model to the scatterplot and observe the fit

```
# Code to plot using matplotlib.pyplot and statsmodel
abline_plot.
fig = sm.graphics.abline_plot(model_results = model)
ax = fig.axes[0]
plt.scatter(temp_heart_regression['Heart Rate'],
            temp_heart_regression['Body Temp'])
plt.show()
```



Knowledge Check 2



Exercise 2



Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	✓
Prepare the dataset to run a linear regression model	✓
Implement and run the linear regression model on the given data	✓
Evaluate the linear model and analyze summary metrics	
Validate the linear regression model by looking at its assumptions	

Evaluate linear model: summary

- Let's now review each component of this detailed summary

Evaluate: coefficient

- In simple linear regression, coefficients are two unknown constants that represent the *intercept* and *slope* terms, which we have covered
- The first of four components is:
 - Estimate

	coef	std err	t	P> t	[0.025	0.975]
const	96.3068	0.658	146.429	0.000	95.005	97.608
Heart Rate	0.0263	0.009	2.967	0.004	0.009	0.044

- The coefficient *Estimate*:
 - Intercept: the expected value of x, in our example Heart Rate, for y to equal 0, e.g. someone to have a Body Temp of zero
 - Slope: the effect x has on y - in our example, the effect Heart Rate has on Body Temp

Evaluate: coefficients standard error

- The second of four components is:
 - Std. Error

	coef	std err	t	P> t	[0.025	0.975]
const	96.3068	0.658	146.429	0.000	95.005	97.608
Heart Rate	0.0263	0.009	2.967	0.004	0.009	0.044

- The coefficient *Standard Error*:
 - Each corresponding value measures the average amount that the coefficient estimates vary from the actual average value of our response variable
 - Ideally, we want a lower number relative to its coefficients

Evaluate: coefficient t-value

- The third component to the coefficient is:
 - t value

	coef	std err	t	P> t	[0.025	0.975]
const	96.3068	0.658	146.429	0.000	95.005	97.608
Heart Rate	0.0263	0.009	2.967	0.004	0.009	0.044

- The coefficient *t-value*:
 - Measure of how many standard deviations our coefficient estimate is far away from 0
 - The null hypothesis for the t-test is that the coefficient is 0 standard deviations away from 0
 - We want to be able to reject the null and say that the coefficient is far away from 0, this allows us to declare there is a relationship between the target and the predictor
 - t-values are also then used to compute the p-values

Evaluate: coefficients p-value

- The last component to the coefficient is:
 - $\Pr(> |t|)$

	coef	std err	t	P> t	[0.025	0.975]
const	96.3068	0.658	146.429	0.000	95.005	97.608
Heart Rate	0.0263	0.009	2.967	0.004	0.009	0.044

- The coefficient $\Pr(> |t|)$
 - This represents the **p-value** or the probability of observing any value equal or larger than t
 - A small p-value indicates it is unlikely the relationship between x and y is based on chance, we want a small p-value
 - Typically, a p-value of under **5%** is a good cut-off point, but all this depends on the confidence interval that has been set

Evaluate: residual standard error

- **Residual standard error** measures the quality of a linear regression fit
- Below is how to calculate residual standard error in Python

```
# Residual standard error  
print(np.sqrt(model.scale))
```

```
0.7119688909228121
```

- In theory, every linear model is assumed to contain an error term, ϵ
- The residual standard error is the average amount that the target variable will deviate from the true regression line
- The residual standard error should not be 0, this means it's a perfect model and won't generalize well
- If the residual standard error can't be shown, then the model probably does not have any predictive ability

Evaluate: R-squared

- R-squared and adjusted R-squared are the next two components to discuss

OLS Regression Results			
=====			
Dep. Variable:	Body Temp	R-squared:	0.064
Model:	OLS	Adj. R-squared:	0.057
Method:	Least Squares	F-statistic:	8.802
Date:	Mon, 22 Jul 2019	Prob (F-statistic):	0.00359
Time:	11:44:37	Log-Likelihood:	-139.29
No. Observations:	130	AIC:	282.6
Df Residuals:	128	BIC:	288.3
Df Model:	1		
Covariance Type:	nonrobust		

- **R-squared** is a statistic that provides a measure of how well the model is fitting the actual data
- It is the measure of the linear relationship between the predictor and the target
- It always lies between 0 and 1
- An R-squared of .064 means that around 6.4% of the variance found in the target variable can be explained by the predictor
- It is hard to say what a “good R-squared” is
- It depends heavily on the subject matter and application

Evaluate: Adjusted R-squared

- Adjusted R-squared is R-squared that accounts for multiple predictors

OLS Regression Results			
=====			
Dep. Variable:	Body Temp	R-squared:	0.064
Model:	OLS	Adj. R-squared:	0.057
Method:	Least Squares	F-statistic:	8.802
Date:	Mon, 22 Jul 2019	Prob (F-statistic):	0.00359
Time:	11:44:37	Log-Likelihood:	-139.29
No. Observations:	130	AIC:	282.6
Df Residuals:	128	BIC:	288.3
Df Model:	1		
Covariance Type:	nonrobust		

- Adjusted R-squared** is the same measure as R-squared
- In multiple regression, R-squared will constantly increase with added variables
- Adjusted R-squared will adjust for additional variables

Evaluate: F-statistic

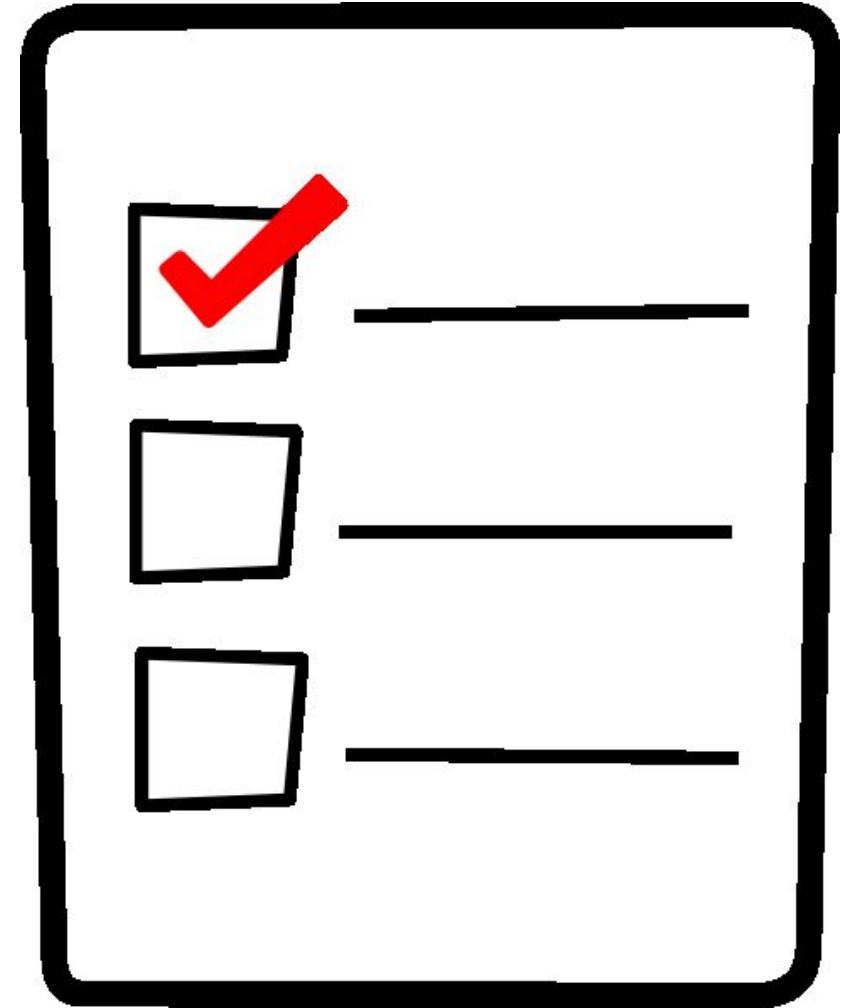
- The **F-statistic** is a good indicator of whether there is a relationship between our predictor and the target variable

OLS Regression Results			
=====			
Dep. Variable:	Body Temp	R-squared:	0.064
Model:	OLS	Adj. R-squared:	0.057
Method:	Least Squares	F-statistic:	8.802
Date:	Mon, 22 Jul 2019	Prob (F-statistic):	0.00359
Time:	11:44:37	Log-Likelihood:	-139.29
No. Observations:	130	AIC:	282.6
Df Residuals:	128	BIC:	288.3
Df Model:	1		
Covariance Type:	nonrobust		

- The further the F-statistic is from 1, the better it is
- F-statistic in regression is used to assess the overall performance of the model
- It is usually used in combination with the p-value

Final evaluation of summary metrics

- We have reviewed each component, now let's evaluate them all together and decide whether or not to move on to the next step of evaluating this simple linear regression model



Final evaluation: part 1

```
=====
                        OLS Regression Results
=====
Dep. Variable:          Body Temp    R-squared:                0.064
Model:                  OLS          Adj. R-squared:            0.057
Method:                 Least Squares  F-statistic:              8.802
Date:                   Mon, 22 Jul 2019  Prob (F-statistic):      0.00359
Time:                   11:44:37      Log-Likelihood:           -139.29
No. Observations:       130          AIC:                     282.6
Df Residuals:           128          BIC:                     288.3
Df Model:                1
Covariance Type:        nonrobust
=====
                        coef      std err          t      P>|t|      [0.025      0.975]
-----
const          96.3068         0.658     146.429     0.000     95.005     97.608
Heart Rate      0.0263         0.009       2.967     0.004      0.009      0.044
=====
Omnibus:                2.917    Durbin-Watson:           0.278
Prob(Omnibus):          0.233    Jarque-Bera (JB):         2.986
Skew:                   0.051    Prob(JB):                 0.225
Kurtosis:               3.735    Cond. No.                  781.
=====
```

- **Coefficients**: we looked over all four components of the two coefficients and determined that the intercept and Heart Rate are significant and should be kept in the model
- **R-squared**: the R-squared of .064 means that around 6.4% of the variance found in Body Temp can be explained by Heart Rate

Final evaluation: part 2

```
=====
                        OLS Regression Results
=====
Dep. Variable:          Body Temp      R-squared:                0.064
Model:                  OLS            Adj. R-squared:           0.057
Method:                 Least Squares   F-statistic:              8.802
Date:                  Mon, 22 Jul 2019 Prob (F-statistic):       0.00359
Time:                  11:44:37         Log-Likelihood:           -139.29
No. Observations:      130             AIC:                    282.6
Df Residuals:          128             BIC:                    288.3
Df Model:               1
Covariance Type:       nonrobust
=====
                        coef      std err          t      P>|t|      [0.025      0.975]
-----
const          96.3068      0.658      146.429      0.000      95.005      97.608
Heart Rate      0.0263      0.009       2.967      0.004       0.009      0.044
=====
Omnibus:                2.917      Durbin-Watson:           0.278
Prob(Omnibus):          0.233      Jarque-Bera (JB):        2.986
Skew:                   0.051      Prob(JB):                0.225
Kurtosis:               3.735      Cond. No.                781.
=====
```

- **Adjusted R-squared**: adjusted R-squared does not matter as much here since we are dealing with single linear regression and it is most helpful once multiple variables are added into the model
- **F-statistic**: we see a significant p-value of 0.00359, so we are able to reject the null hypothesis and say that the model does have predictive power

Knowledge Check 3



Exercise 3



Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	✓
Prepare the data set to run a linear regression model	✓
Implement and run the linear regression model on the given data	✓
Evaluate the linear model and analyze summary metrics	✓
Validate the linear regression model by looking at its assumptions	

Assumptions of linear regression

- We ran and analyzed our model: good fit based off the `statsmodel.OLS` model summary
- Now, let's double check that the model meets the assumptions of linear regression
- Remember **LINE** from earlier today?
 - The relationship between the predictor and response variable must be **L**inear
 - The residuals must be **I**ndependent (we will not be looking for this today)
 - The residuals must be **N**ormally distributed
 - The residuals must have **E**qual variance
- We will now review how to use the residuals to see how well or poorly the linear regression fits the data

Assumptions: plot

- First, let's find the residuals as outputted from the model
- The residuals are equal to the $\text{actual} - \text{predicted value}$
- In this case, we are not yet predicting and we will use the residuals from our given model

```
fitted = model.fittedvalues  
print(fitted.head())
```

```
0    98.150172  
1    98.176507  
2    98.255511  
3    98.413518  
4    98.229176  
dtype: float64
```

```
residuals = model.resid  
print(residuals.head())
```

```
0    -1.850172  
1    -1.476507  
2    -1.355511  
3    -1.413518  
4    -1.129176  
dtype: float64
```


Assumption: residuals vs. fitted be linear

The first assumption is:

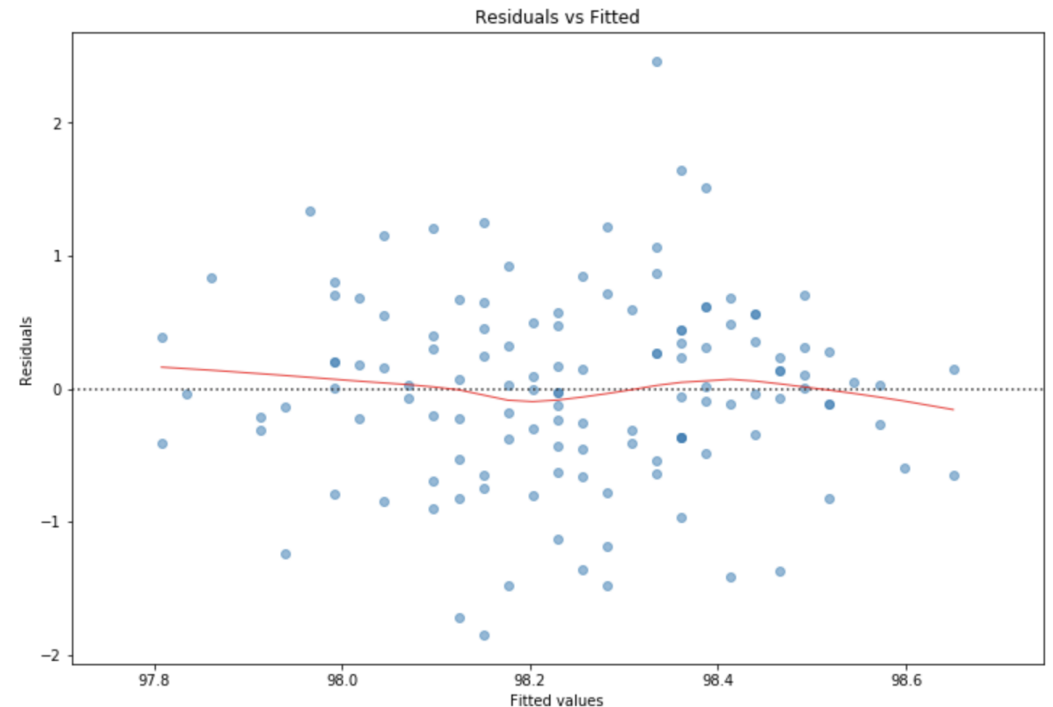
The relationship between the predictor and response variable must be [L]inear

```
import seaborn as sns

plot_lm_1 = plt.figure(1)
plot_lm_1.set_figheight(8)
plot_lm_1.set_figwidth(12)

plot_lm_1.axes[0] = sns.residplot(fitted, 'Body
Temp', data=temp_heart_regression,
                                lowess = True,
                                scatter_kws =
{'alpha': 0.5},
                                line_kws = {'color':
'red', 'lw': 1, 'alpha': 0.8})

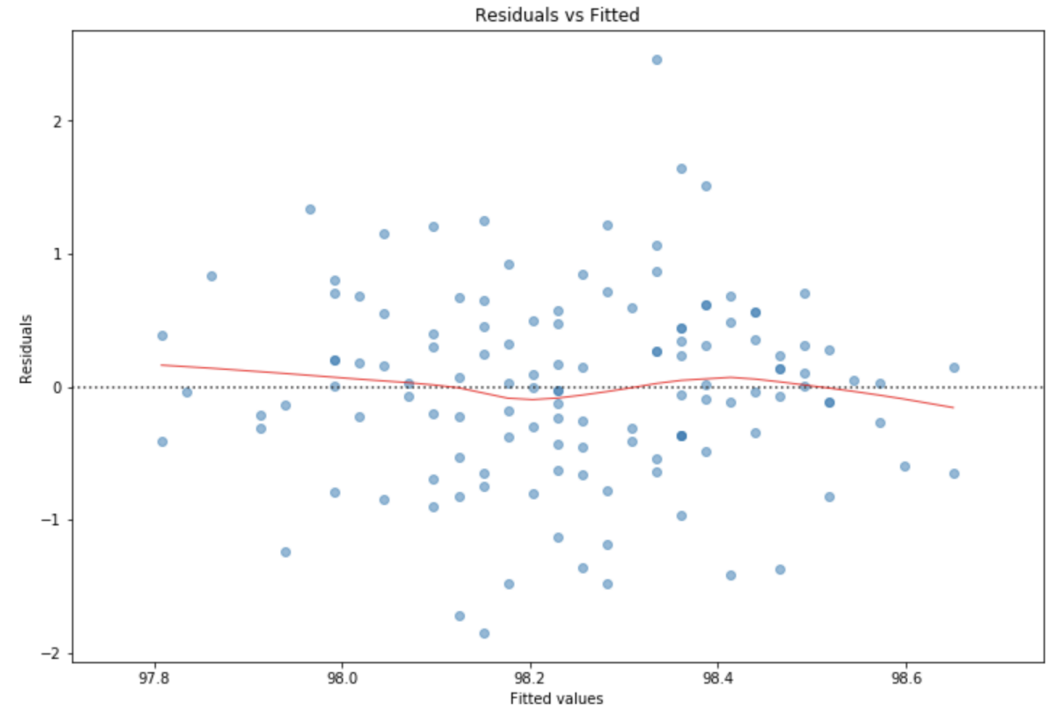
plot_lm_1.axes[0].set_title('Residuals vs
Fitted')
plot_lm_1.axes[0].set_xlabel('Fitted values')
plot_lm_1.axes[0].set_ylabel('Residuals')
plot_lm_1.axes
```



- Note: the code for this plot is using *seaborn* which is another popular Python library for visualizations

Assumption: linearity satisfied

- Our simple linear regression model **does meet the assumption**
- To meet the assumption, the residuals should be scattered evenly around the dashed line at 0



Assumption: normally distributed residuals

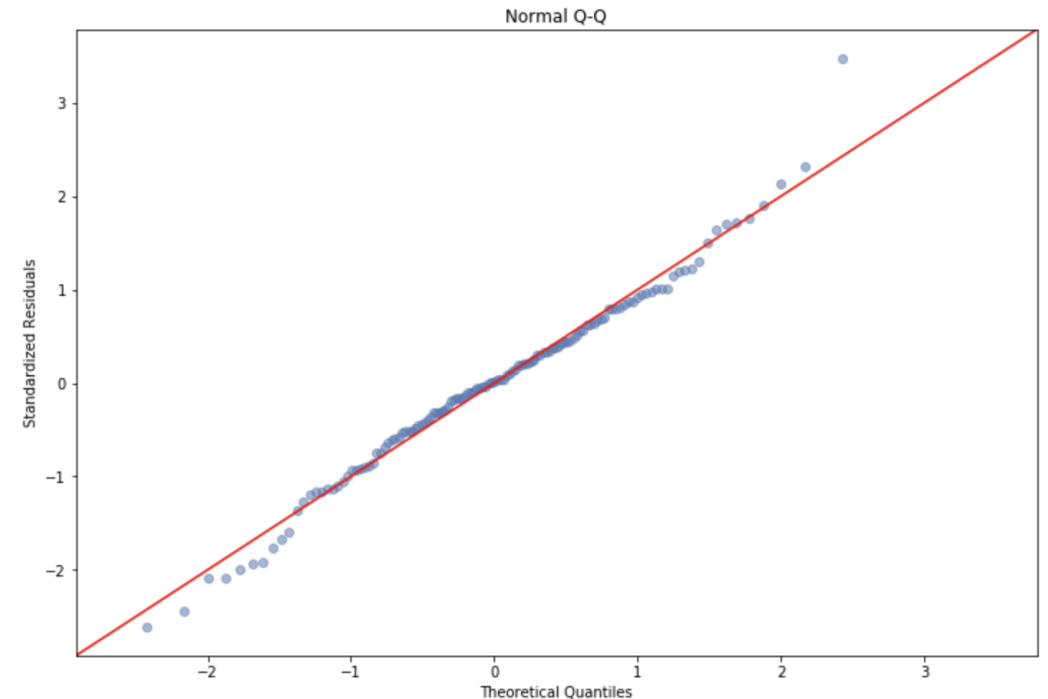
The second assumption is:

The residuals must be [N]ormally distributed

```
from statsmodels.graphics.gofplots import ProbPlot
model_norm_residuals =
model.get_influence().resid_studentized_internal
QQ = ProbPlot(model_norm_residuals)
plot_lm_2 = QQ.qqplot(line= '45', alpha = 0.5,
color = '#4C72B0', lw = 1)

plot_lm_2.set_figheight(8)
plot_lm_2.set_figwidth(12)

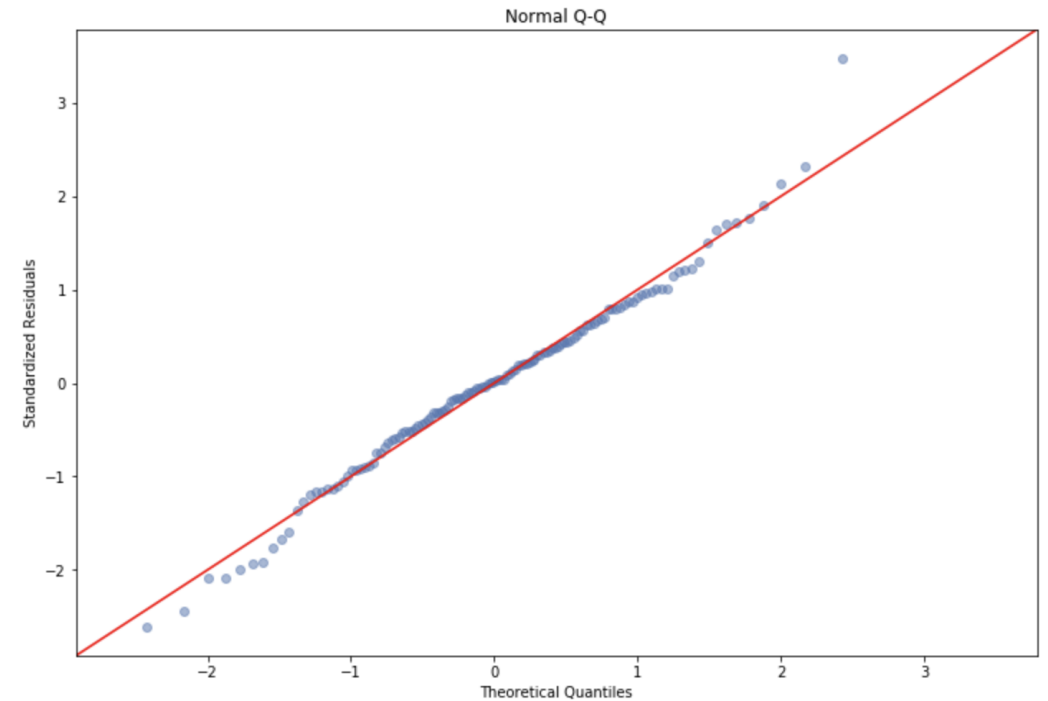
plot_lm_2.axes[0].set_title('Normal Q-Q')
plot_lm_2.axes[0].set_xlabel('Theoretical
Quantiles')
plot_lm_2.axes[0].set_ylabel('Standardized
Residuals')
```



- Note: the code for this plot is using *ProbPlot*, also part of *thstatsmodel* package

Assumption: normality satisfied

- Our simple linear regression model **does meet the assumption**
- The residuals do fit the QQ-line that signifies normality

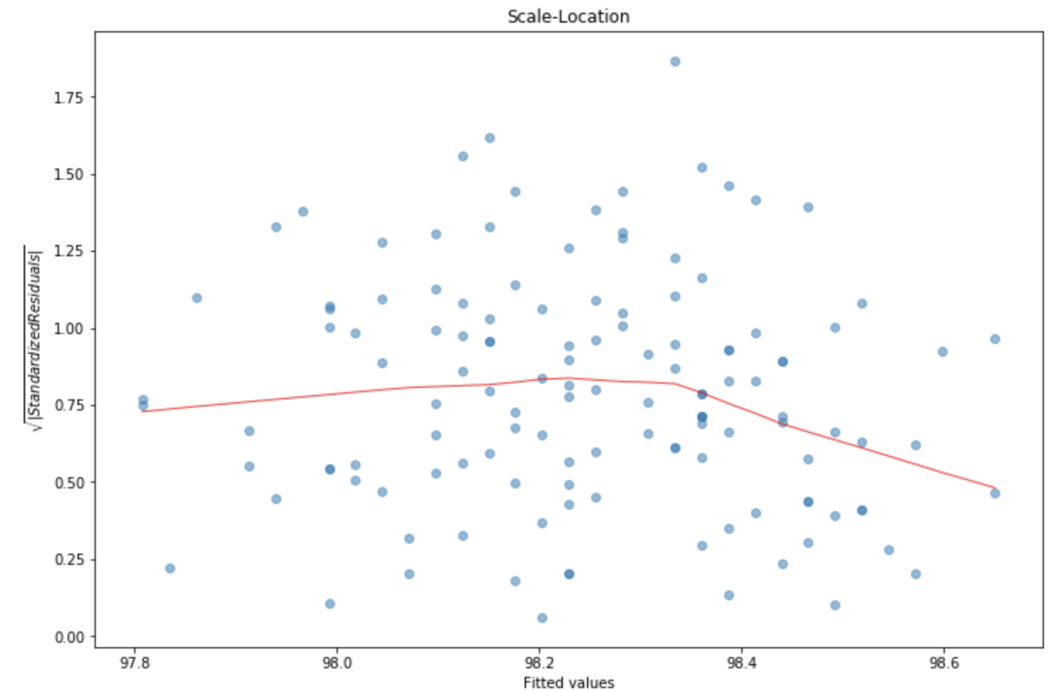


Assumption: equal residual variance

The third assumption is:

The residuals must have [E]qual variance

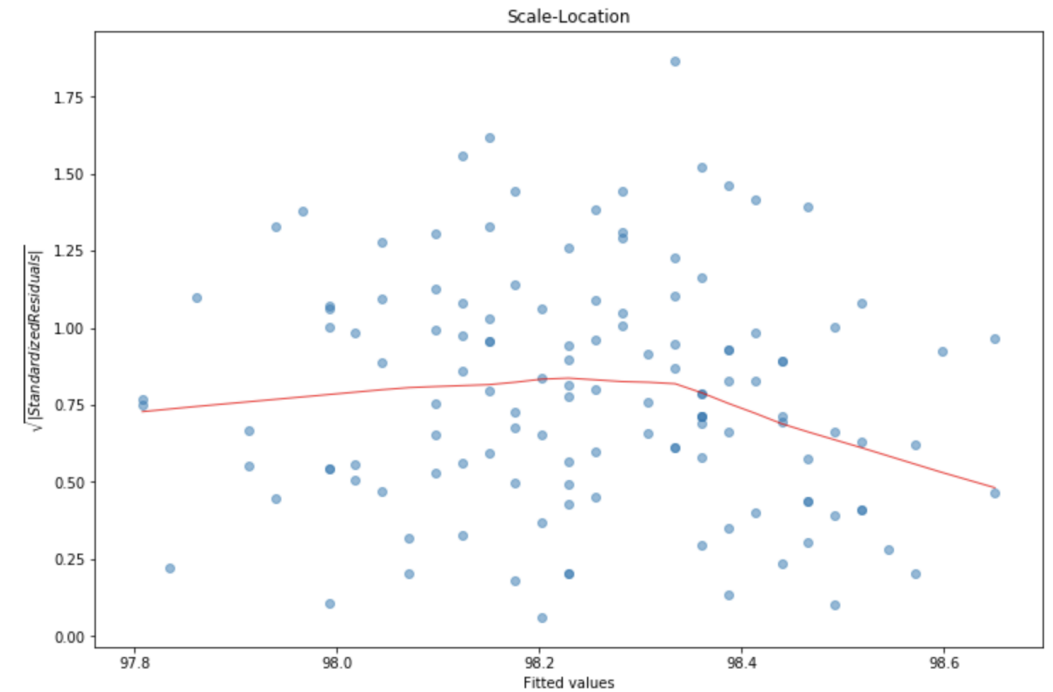
```
model_norm_residuals_abs_sqrt =  
np.sqrt(np.abs(model_norm_residuals))  
plot_lm_3 = plt.figure(3)  
plot_lm_3.set_figheight(8)  
plot_lm_3.set_figwidth(12)  
  
plt.scatter(fitted,  
model_norm_residuals_abs_sqrt, alpha = 0.5)  
sns.regplot(fitted,  
model_norm_residuals_abs_sqrt,  
            scatter = False,  
            ci = False,  
            lowess = True,  
            line_kws = {'color': 'red', 'lw': 1,  
                        'alpha': 0.8})  
  
plot_lm_3.axes[0].set_title('Scale-Location')  
plot_lm_3.axes[0].set_xlabel('Fitted values')  
plot_lm_3.axes[0].set_ylabel('$\sqrt{|Standardized  
Residuals|}$')
```



- Note: the code for this plot is using *seaborn* which is another popular Python library for visualizations

Assumption: equal residual variance satisfied

- Our simple linear regression model **does meet the assumption**
- It looks like the residuals are spread equally along the line (except for a few values on the right side)



Influential cases: residuals vs. leverage

- We just evaluated our model against three of the assumptions
- We saw that it does seem to meet all of them
- Now we look for outliers
- We use `statsmodel.outlier_test()` to test for outliers which uses the *Bonferroni outlier test*

```
# Let's find influential points.  
test = model.outlier_test()  
print('Bad data points (bonf(p) < 0.05):')
```

```
Bad data points (bonf(p) < 0.05):
```

```
print(test[test['bonf(p)'] < 0.05])  
  
# Save the final outliers.
```

```
Empty DataFrame  
Columns: [student_resid, unadj_p, bonf(p)]  
Index: []
```

```
test_final = test[test['bonf(p)'] < 0.05]
```

Removing outliers from regression dataset

- Now that we have a saved dataframe of outliers, let's remove them from our original dataset and save it as `temp_heart_no_outliers`
- Note that in our case, there were no outliers according to the Bonferroni outlier test

```
test_final = test[test['bonf(p)'] < 0.05]
temp_heart_no_outliers = temp_heart_regression.drop(test_final.index)
# Look at the shape of the new dataframe, notice that no rows have actually been dropped.
print(temp_heart_no_outliers.shape)
```

```
(130, 2)
```


Importance of checking assumptions and outliers

- We can see if it makes sense to use a linear approach
 - There may be more in the data
 - We can use a more complex model
- This is the final step in the go/no go test for linear regression
- We have analyzed and understood how to validate our simple linear regression model
- **Single variable linear regression is the base of machine learning, it teaches many concepts**
- *However, it is very finicky and often can not be used because of all the assumptions*
- **In the next class, we will learn about multiple regression, followed by more complex methods which you will most probably end up using in your projects**

Knowledge Check 4



Exercise 4



Module completion checklist

Objective	Complete
Summarize the basics of linear regression and how it can help in a business setting	✓
Evaluate data using basic statistics such as summary statistics, covariance, correlation	✓
Explain hypothesis tests as needed for regression	✓
Prepare the dataset to run a linear regression model	✓
Implement and run the linear regression model on the given data	✓
Evaluate the linear model and analyze summary metrics	✓
Validate the linear regression model by looking at its assumptions	✓

Workshop: Next steps!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- **Today you will:**
 - Subset your dataset into two variables, the target must be continuous
 - Explore your data using basic summary statistics
 - Prepare your data to run a linear regression
 - Run a linear model
 - Check if assumptions are met. If the assumptions are not met, rerun if anything more is uncovered (outliers)
 - Save your final single regression model as `workshop_final_single_regression.sav`

This completes the module!
Congratulations!