

DATA SOCIETY®

Regression - Day 3

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

Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	
Present cases where assumptions are violated	
Analyze data with log-log models	
Analyze data with log-lin models	
Analyze data with polynomial models	
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

Import packages

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

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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
```

Regression assumptions recap

- We learned how to use the regression equation $Y = b_0 + b_1 * X_1$ to make predictions
- There are **several assumptions that must be tested for**, otherwise this method is not appropriate for a given dataset
- They can be abbreviated into the acronym **'LINE'**
 - The relationship between the predictor and response variable must be **L**inear
 - The residuals must be **I**ndependent
 - The residuals must be **N**ormally distributed
 - The residuals must have **E**qual variance
- Additionally, in multiple linear regression, we also need to check that the **predictor variables are uncorrelated**

Revisiting temp_heart data

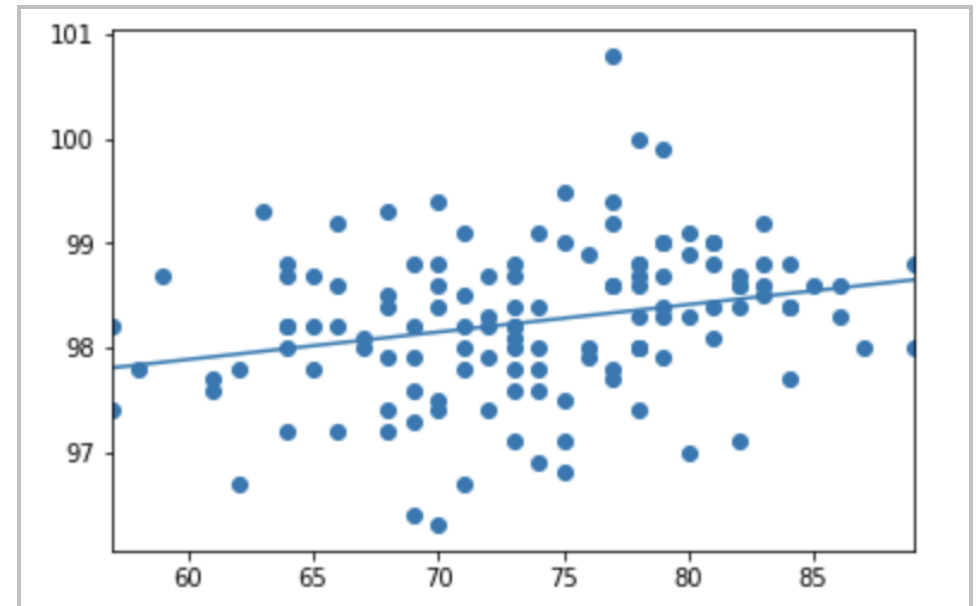
- In Regression Day 1, we used `temp_heart_data.csv` and predicted Body Temp based on Heart Rate
- The model output looked like this:

```
=====
                        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.
=====
```

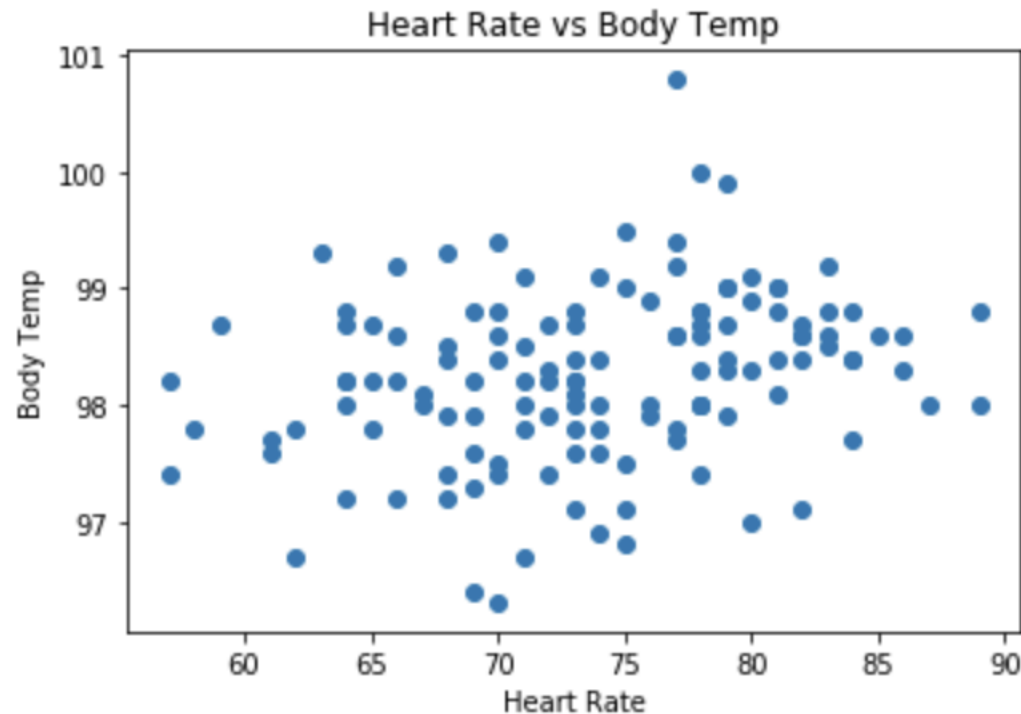
- The plot of Heart Rate VS. Body Temp looked like this:



- We will now check the modeling assumptions

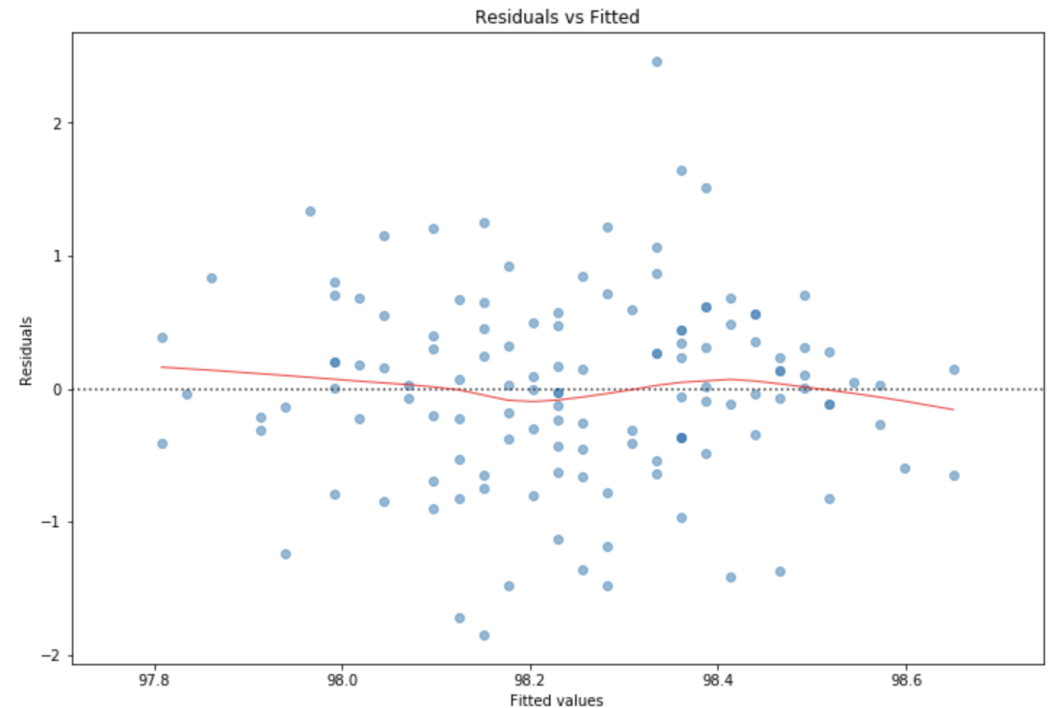
Relationship between x and y must be linear

- On one hand, we can eyeball this assumption by looking at the plot of the actual values:



- It does look like the relationship is linear

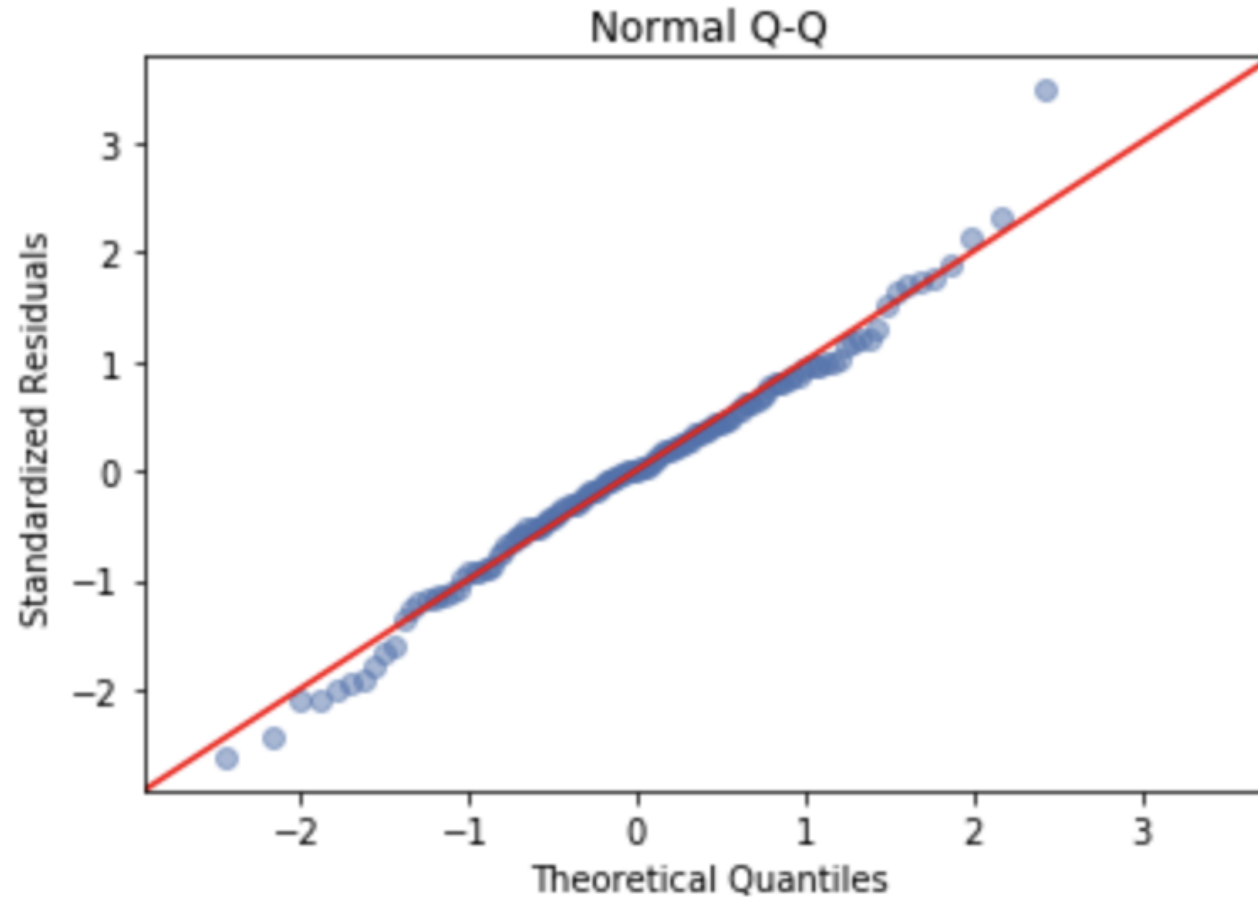
- We can also test this assumption by looking at the Residual vs. Fitted plot



- Because the red line is horizontal, the relationship is in fact linear

The residuals must be normally distributed

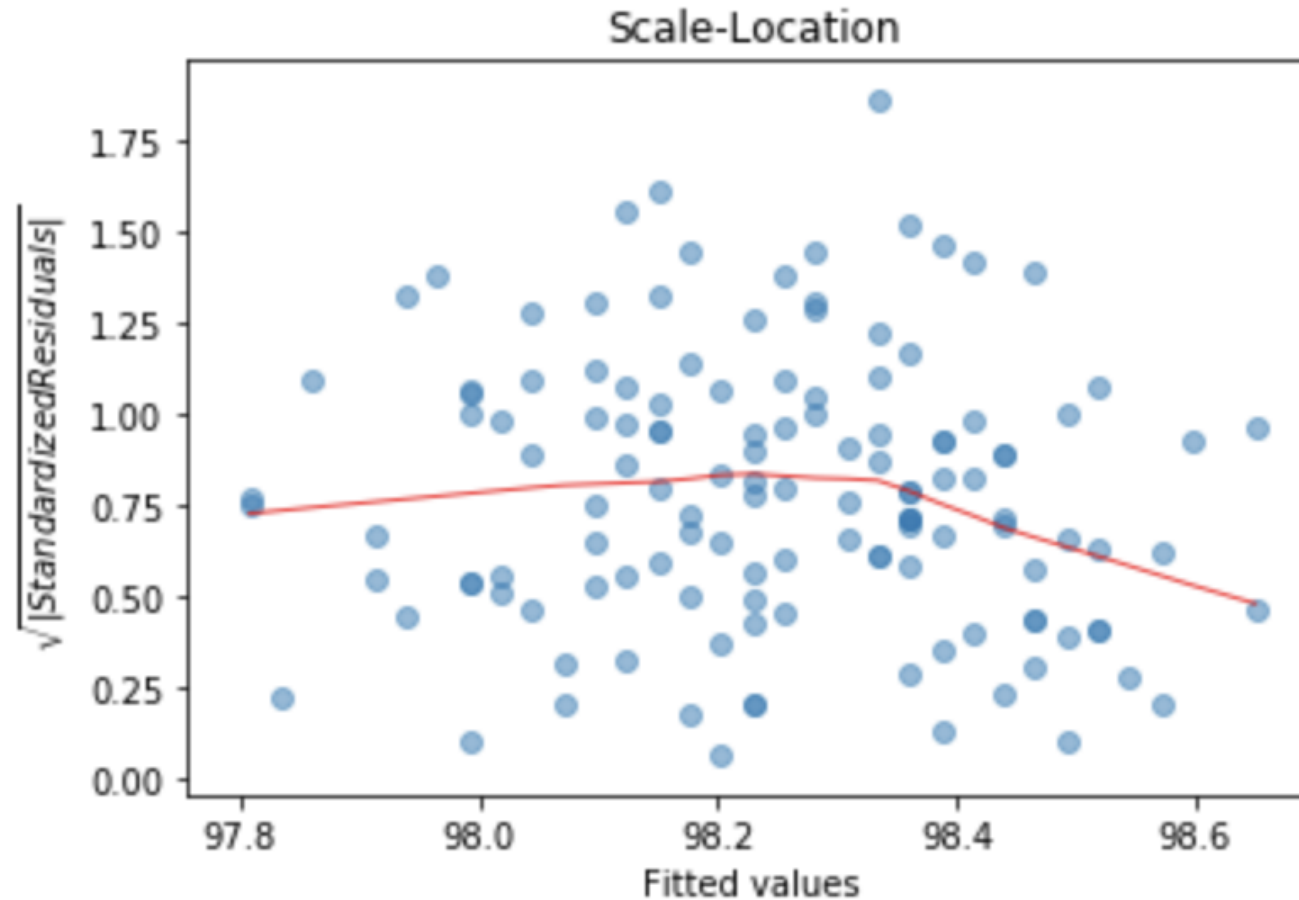
- We can test this assumption by looking at the Normal Q-Q plot



- Because the points are close to the red line, the residuals are in fact normally distributed

The residuals must have equal variance

- We can test this assumption by looking at a plot of the Standardized Residuals vs. Fitted Values




- Because the red line is horizontal, the residuals do in fact have equal variance

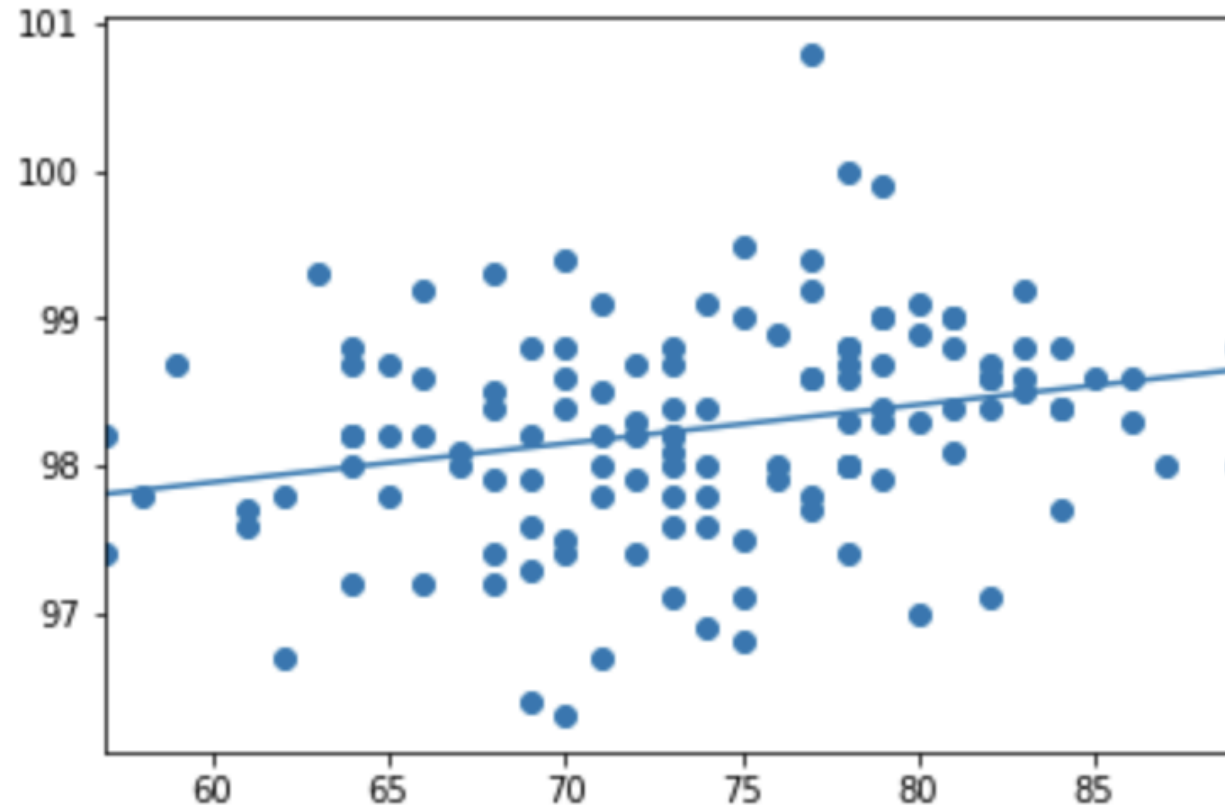
The assumptions are satisfied

- We established that the model satisfies the assumptions. Great!
- But unfortunately, the setup we chose for our model will not fit every dataset

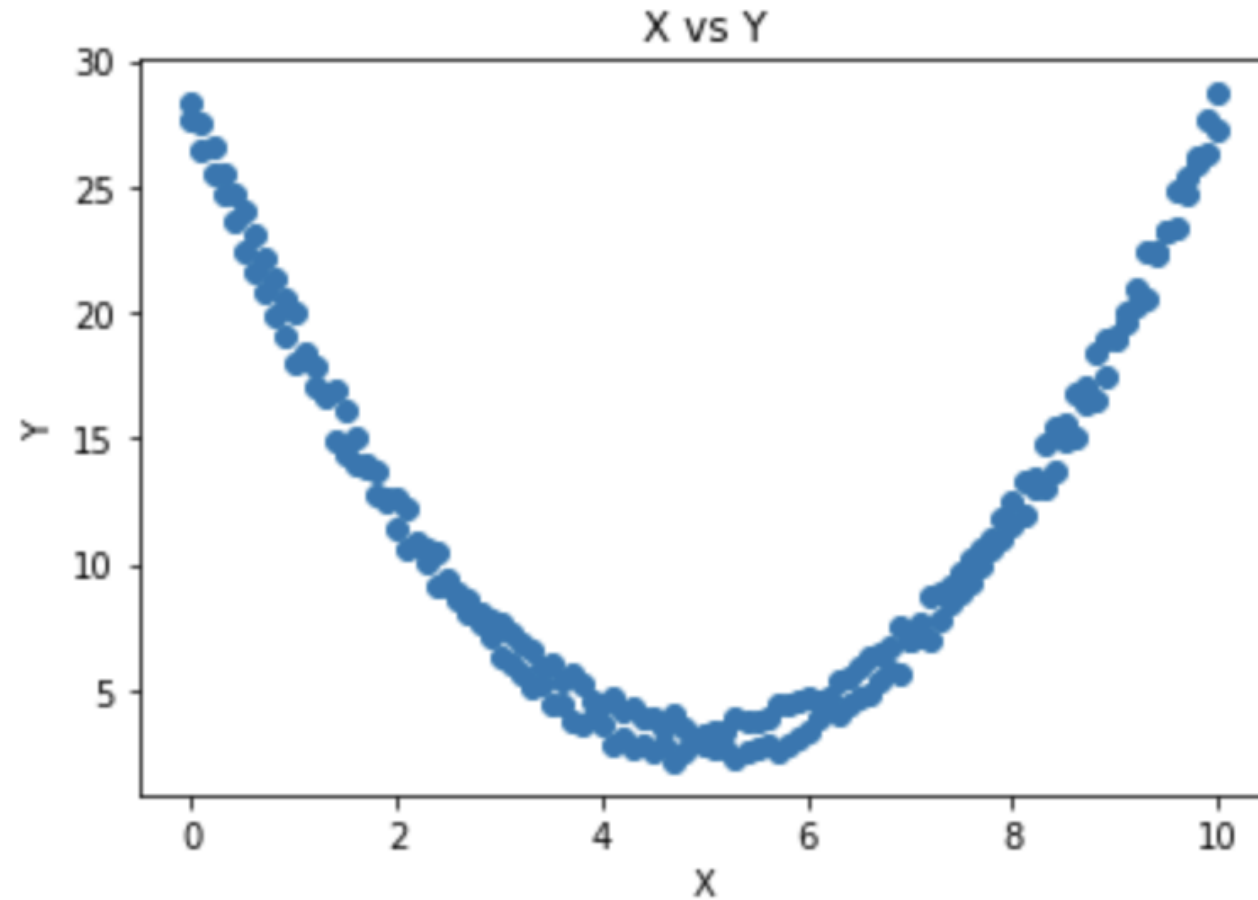
Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	
Present cases where assumptions are violated	
Analyze data with log-log models	
Analyze data with log-lin models	
Analyze data with polynomial models	
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

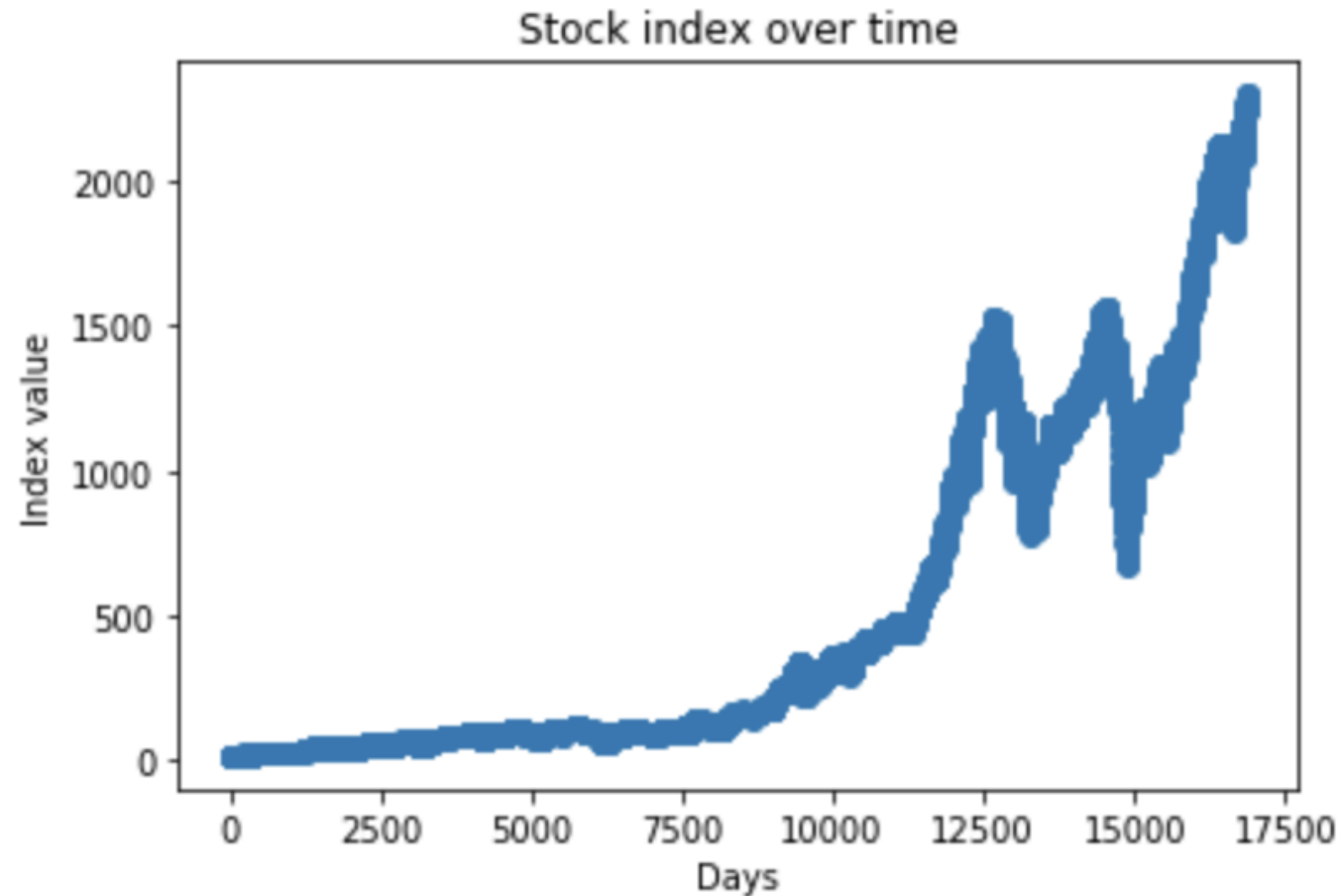
So far we analyzed data with a linear trend



But what if our data looks like this?



Or like this?



Today, we will learn how to handle non-linearity in data

Non-linearity in data

- Very often data is non-linear, meaning the change of the output is not proportional to the change of the input
- This means that we need to model it accordingly
- If there is non-linearity in the data and we model it using a linear model, two things happen:
 - The **assumptions are violated**
 - We **can improve the model metrics** (such as RMSE) by accounting for non-linearity

Models for non-linear data

- There are different techniques to account for non-linearity in the data
- We will discuss the following ones today:
- **Log-log models**
 - A percentage change in the predictor leads to a percentage change in the outcome variable
 - Very often used for data that includes prices of products and services
- **Log-lin models**
 - A unit change in the predictor leads to a percentage change in the outcome variable
 - Very often used to model stock market indices
- **Polynomial models**
 - Use X as well as X^2 , X^3 , etc. as a predictor
 - Very often used to model general non-linearity in data

Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Analyze data with log-log models	
Analyze data with log-lin models	
Analyze data with polynomial models	
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

Log-log model

- **What it is**

- The log-log model allows you to model percentage changes in the predictor and the outcome
- A percentage change in the predictor leads to a percentage change in the outcome variable

- **How it works**

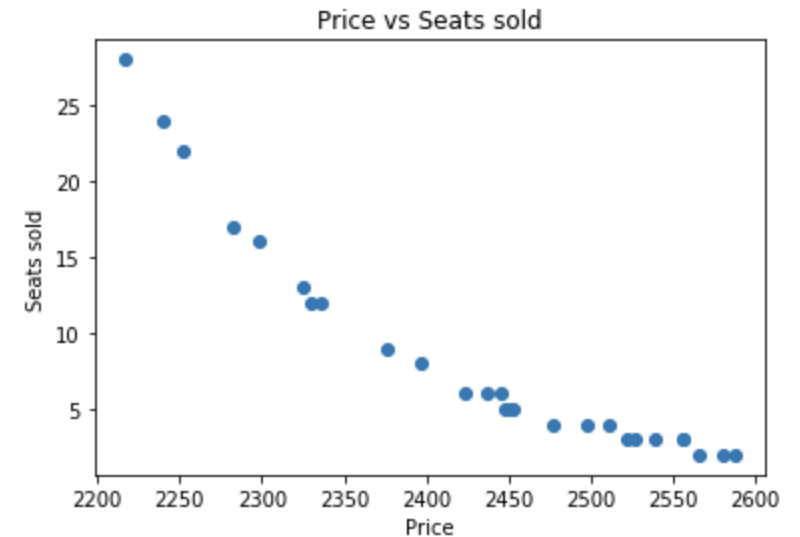
- Instead of fitting a regression of Y on X, you fit a regression of $\log(Y)$ on $\log(X)$:
$$\log(Y) = b_0 + b_1 * \log(X_1)$$
- You need to convert the variables Y and X into $\log(Y)$ and $\log(X)$ before fitting the regression
- When using log-log to predict, you get $\log(Y)$ back, so you need to exponentiate $\log(Y)$ to get the actual value of Y back

- **When it is used**

- Log-log models are popular in sales and marketing to model responses to prices
- You would use a log-log model for “How many more movie tickets will we sell, if we lower the price of tickets by 10%?”

Applying a dataset with a log-log model

- Let's use the sample dataset `IAI.csv`, which features airline ticket prices and seats sold
- You work for a flight booking website and your manager would like for you to use your skills to predict the number of seats that will be available for sale
- We are trying to predict the number of **first class** seats sold on a popular Monday morning commuter flight
- The prices were randomly varied between \$2200 and \$2600 on 27 consecutive weeks
- Target variable: **Number of seats sold (Seats)**
- Predictor variable: **Price of tickets (Price)**



Steps when using a log-log model

- 1) Import the data
 - 2) EDA and data cleaning
 - **3) Transform predictor and target to log of the actual values**
 - 4) Fit the log-log model
 - 5) Predict with the log-log model
 - **6) Exponentiate predictions**
 - 7) Evaluate model
-
- The steps are generally the same as for normal linear regression
 - However, when using a log-log model, we need to add extra steps: **step 3 and step 6**

Importing, EDA, and data cleaning

- Import data and check shape

```
IAI = pd.read_csv(data_dir + '/IAI.csv')  
IAI.shape
```

```
(27, 2)
```

- Look at first 3 columns

```
IAI.head(3)
```

	Price	Seats
0	2216.93	28
1	2240.27	24
2	2251.58	22

- Check if there are any missing values

```
IAI.isnull().values.any()
```

```
False
```

Transforming predictor and target to log

- We need to transform the values of our predictor and target variables to the log of the actual values
- We use the numpy function `np.log()` to achieve this
- We can pass the entire column to `np.log()`

```
IAI['Price_log'] = np.log(IAI['Price'])  
IAI['Seats_log'] = np.log(IAI['Seats'])
```

- Let us look at the first 3 rows again

```
IAI.head(3)
```

	Price	Seats	Price_log	Seats_log
0	2216.93	28	7.703879	3.332205
1	2240.27	24	7.714352	3.178054
2	2251.58	22	7.719387	3.091042

Fitting the log-log model

- Next, we fit the log-log model: $\log(\text{Seats}) = b_0 + b_1 * \log(\text{Price})$
- We first need to add a constant to our data using `sm.add_constant()`

```
IAI = sm.add_constant(IAI)
```

- Then, we fit the model using `sm.OLS` and pass it the columns we transformed to log in the step before

```
model_log = sm.OLS(IAI['Seats_log'], IAI.loc[:, ['const', 'Price_log']]).fit()
```


Predicting with the log-log model

- Now, we use the fitted log-log model to predict
- We are not using a training and test set here, so we predict using the same `Price_log` variable as predictor that we also used for fitting the model

```
prediction_log = model_log.predict(IAI.loc[:, ['const', 'Price_log']])
```

- Let us look at some of the predictions

```
prediction_log[1:10]
```

```
1    3.177108
2    3.091878
3    2.864078
4    2.745240
5    2.546320
6    2.511855
7    2.472315
8    2.183191
9    2.031986
dtype: float64
```

- But wait, aren't these values way too low? They are! **We need to exponentiate them!**

Exponentiating the predictions

- We need to exponentiate the predictions, because the log-log model predicts the log of the outcome variable
 - Remember that we fit the model giving it the log value of the outcome variable as a target
- To exponentiate the predictions, we use the numpy function `np.exp()`

```
prediction = np.exp(prediction_log)
```

- Let us look at the exponentiated predictions

```
prediction[1:5]
```

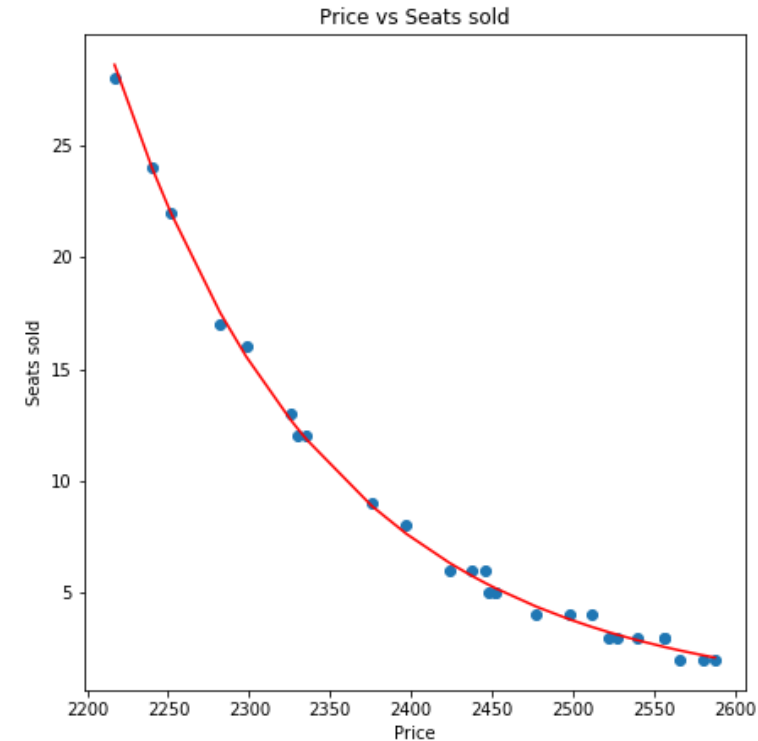
```
1    23.977317
2    22.018390
3    17.532874
4    15.568353
dtype: float64
```

- These look way more accurate!

Evaluating the log-log model: chart

- As a last step, we want to evaluate the model
- We do this with a chart and by calculating the RMSE
- Let us start by creating a chart that shows the actual values of Seats and the predicted values of Seats

```
plt.scatter(IAI['Price'], IAI['Seats'])  
plt.plot(IAI['Price'], prediction, 'red')  
plt.title("Price vs Seats sold")  
plt.xlabel("Price")  
plt.ylabel("Seats sold")  
plt.show()
```



- Wow! Our model fits the data extremely well

Evaluating the log-log model: RMSE

- Let us also evaluate the model using RMSE
- First, we get the residuals and put them into a dataframe, together with the actual values of Seats and the exponentiated predictions of Seats

```
actual = IAI['Seats']
prediction = prediction
residuals = actual - prediction
loglog_results = pd.concat([actual.rename('actual'),
                             prediction.rename('predicted'),
                             residuals.rename('residuals')], axis = 1)
```

- Then, we use our RMSE formula to get the RMSE of our log-log model

```
def rmse(predictions,actual):
    return np.sqrt(((prediction-actual) ** 2).mean())

print(rmse(loglog_results['predicted'],loglog_results['actual']))
```

```
0.33339266048990407
```

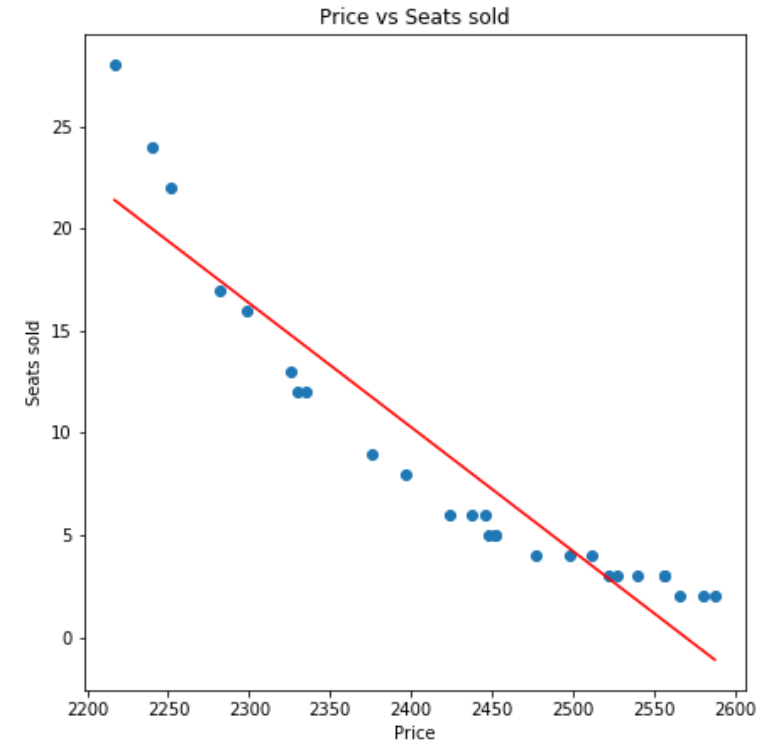
How would a linear model have performed?

- Let us see how a standard linear model would have performed

```
# Fit the model
model_lin = sm.OLS(IAI['Seats'], IAI.loc[:,
['const', 'Price']]).fit()
prediction = model_lin.predict(IAI.loc[:,
['const', 'Price']])
```

- Get a chart of actual vs. predicted

```
plt.scatter(IAI['Price'], IAI['Seats'])
plt.plot(IAI['Price'], prediction, 'red')
plt.title("Price vs Seats sold")
plt.xlabel("Price")
plt.ylabel("Seats sold")
plt.show()
```



- Way worse!

Knowledge Check 1



Exercise 1



Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Analyze data with log-log models	✓
Analyze data with log-lin models	
Analyze data with polynomial models	
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

Log-lin model

- **What it is**

- The log-lin model allows you to create a model where a unit change in the predictor causes a percent change in the target variable

- **How it works**

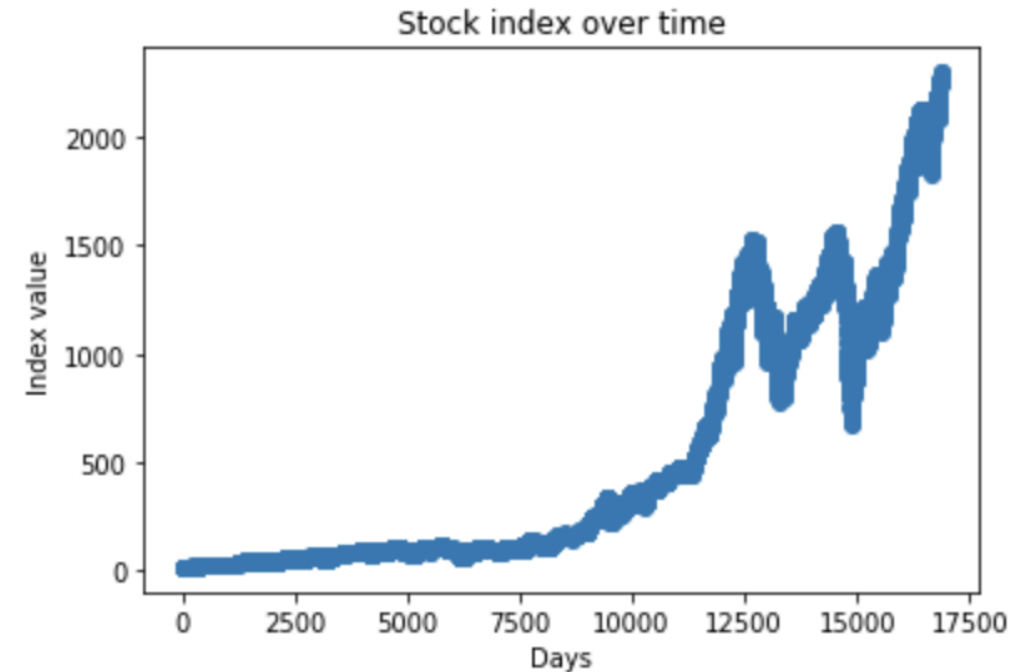
- Instead of fitting a regression of Y on X, you fit a regression of log(Y) on X:
$$\log(Y) = b_0 + b_1 * X_1$$
- You need to convert the variable Y into log(Y) before fitting the regression
- When using log-lin to predict, you get log(Y) back, so you need to exponentiate log(Y) to get the actual value of Y back

- **When it is used**

- Log-lin models are often used to model stocks
- For stocks, we often talk about the percentage they increase over a day/week/month/year
- For example, you would use a log-lin model for questions like “Given the history of the stock, what do we expect the value to be a year from now?”

Our dataset to work with a log-lin model

- Real-world stock dataset `GSPC_data.csv`, to understand the value of the S&P 500 index
- It includes the daily price of the S&P 500 index from 1/3/1950 to 1/27/2017
- Your boss asked you to develop a model that can predict prices for internal knowledge
- We want to fit a model that can accurately predict the price of the index at any given day
- Target variable: **Price of the index (GSPC)**
- Predictor variable: **Day (Dt)**



Steps when using a log-lin model

- 1) Import the data
 - 2) EDA and data cleaning
 - **3) Transform target to log of the actual values**
 - 4) Fit the log-log model
 - 5) Predict with the log-log model
 - **6) Exponentiate predictions**
 - 7) Evaluate model
-
- The steps are almost identical to using a log-log model, the only difference is that **we do not take the log of the predictor to fit the model**

Importing, EDA, and data cleaning

- Import data and check shape

```
GSPC = pd.read_csv(data_dir + '/GSPC_data.csv')  
GSPC.shape
```

```
(16877, 2)
```

- Look at first 3 columns

```
GSPC.head(3)
```

	Dt	GSPC
0	1	16.66
1	2	16.85
2	3	16.93

- Check if there are any missing values

```
GSPC.isnull().values.any()
```

```
False
```

Transforming target to log

- We need to transform the values of our target variable to the log of the actual values
- We use the numpy function `np.log()` to achieve this
- We can pass the entire column to `np.log()`

```
GSPC['GSPC_log'] = np.log(GSPC['GSPC'])
```

- Let us look at the first 3 rows again

```
GSPC.head(3)
```

	Dt	GSPC	GSPC_log
0	1	16.66	2.813011
1	2	16.85	2.824351
2	3	16.93	2.829087

Fitting the log-lin model

- Next, we fit the log-lin model: $\log(GSPC) = b_0 + b_1 * Dt$
- We first need to add a constant to our data using `sm.add_constant()`

```
GSPC = sm.add_constant(GSPC)
```

- Then, we fit the model using `sm.OLS` and pass it the target column we transformed to log in the step before and the predictor columns

```
model_lin = sm.OLS(GSPC['GSPC_log'], GSPC.loc[:, ['const', 'Dt']]).fit()
```

- We are using the whole dataset for modeling since the dataset is simulated and small

Predicting with the log-lin model

- Now, we use the fitted log-lin model to predict
- We are not using a training and test set here, so we predict using the same `Price_log` variable as predictor that we also used for fitting the model

```
prediction_lin = model_lin.predict(GSPC.loc[:, ['const', 'Dt']])
```

- Let us look at some of the predictions

```
prediction_lin[1:10]
```

```
1    3.103890
2    3.104164
3    3.104438
4    3.104712
5    3.104986
6    3.105260
7    3.105534
8    3.105808
9    3.106083
dtype: float64
```

- But wait, aren't these values way too low? They are! **We need to exponentiate them!**

Exponentiating the predictions

- We need to exponentiate the predictions, because the log-lin model predicts the log of the outcome variable
 - Remember that we fit the model giving it the log value of the outcome variable as a target
- To exponentiate the predictions, we use the numpy function `np.exp()`

```
prediction = np.exp(prediction_lin)
```

- Let us look at the exponentiated predictions

```
prediction[1:5]
```

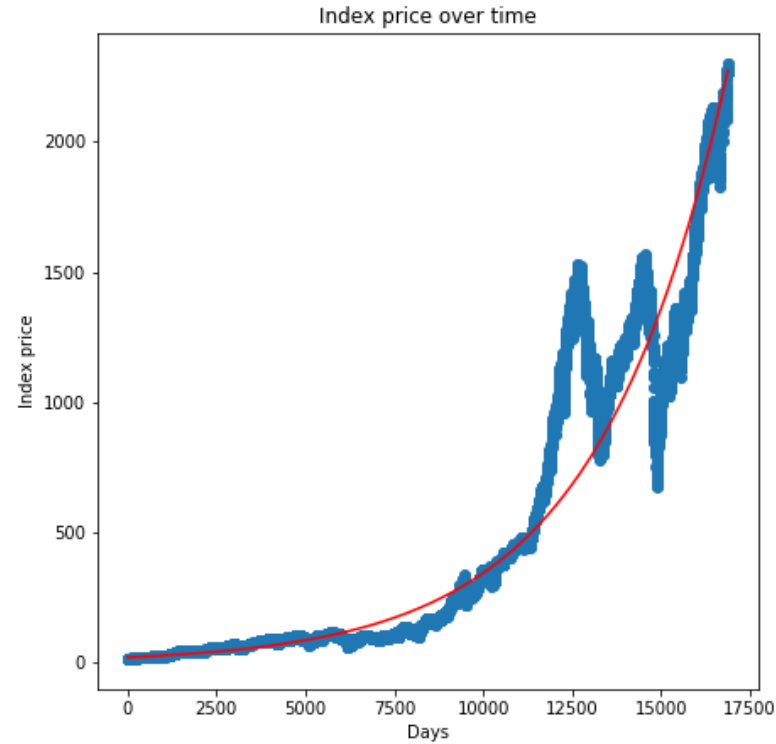
```
1    22.284469
2    22.290578
3    22.296688
4    22.302800
dtype: float64
```

- These values look much more accurate!

Evaluating the log-lin model: chart

- As a last step, we want to evaluate the model
- We do this with a chart and by calculating the RMSE
- Let us start by creating a chart that shows the actual values of GSPC and the predicted values of GSPC

```
plt.scatter(GSPC['Dt'], GSPC['GSPC'])  
plt.plot(GSPC['Dt'], prediction, 'red')  
plt.title("Index price over time")  
plt.xlabel("Days")  
plt.ylabel("Index price")  
plt.show()
```



- Wow! Our model fits the data extremely well

Evaluating the log-lin model: RMSE

- Let us also evaluate the model using RMSE
- First, we get the residuals and put them into a dataframe, together with the actual values of GSPC and the exponentiated predictions of GSPC

```
actual = GSPC['GSPC']
prediction = prediction
residuals = actual - prediction
loglin_results = pd.concat([actual.rename('actual'),
                           prediction.rename('predicted'),
                           residuals.rename('residuals')], axis = 1)
```

- Then, we use our RMSE formula to get the RMSE of our log-lin model

```
def rmse(predictions,actual):
    return np.sqrt(((prediction-actual) ** 2).mean())

print(rmse(loglin_results['predicted'],loglin_results['actual']))
```

```
183.39211235044104
```

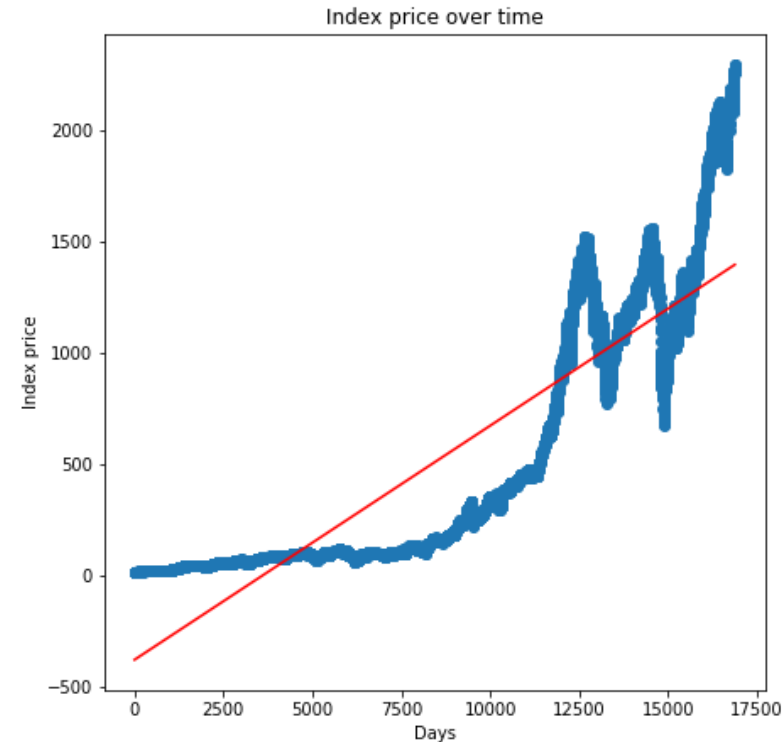
How would a linear model have performed?

- Let us see how a standard linear model would have performed

```
# Fit the model
model_lin = sm.OLS(GSPC['GSPC'], GSPC.loc[:,
['const', 'Dt'])).fit()
prediction = model_lin.predict(GSPC.loc[:,
['const', 'Dt']])
```

- Get a chart of actual vs. predicted

```
plt.scatter(GSPC['Dt'], GSPC['GSPC'])
plt.plot(GSPC['Dt'], prediction, 'red')
plt.title("Index price over time")
plt.xlabel("Days")
plt.ylabel("Index price")
plt.show()
```



- Way worse!

Knowledge Check 2



Exercise 2

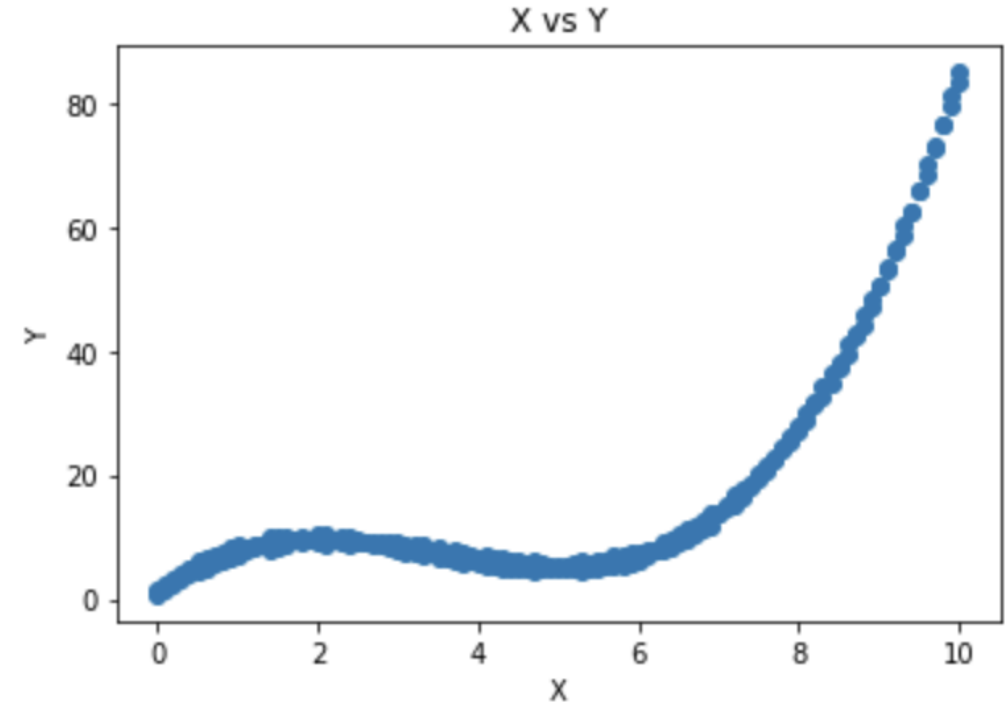


Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Analyze data with log-log models	✓
Analyze data with log-lin models	✓
Analyze data with polynomial models	
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

Polynomial models

- We learned that we can use log-log models and log-lin models when we have data that is based on percentage changes in the predictors or target or both
- However, there are also cases where we do not have a theoretical basis to assume a percentage change
- Moreover, there are also cases where the data has more complex patterns than the ones we saw with log-log and log-lin models
- In these cases, we use **polynomial models**



Polynomial models

- **What it is**

- Polynomial models allow us to model non-linearity in the data
- Polynomial models can fit many different kinds of curves

- **How it works**

- Instead of fitting a regression of Y on X , you fit a regression of Y on X, X^2, X^3 , etc.:
$$Y = b_0 + b_1X_1 + b_2X_1^2 + b_3X_1^3$$
- You need to add the variables X^2, X^3 , etc. to your dataset before fitting the regression

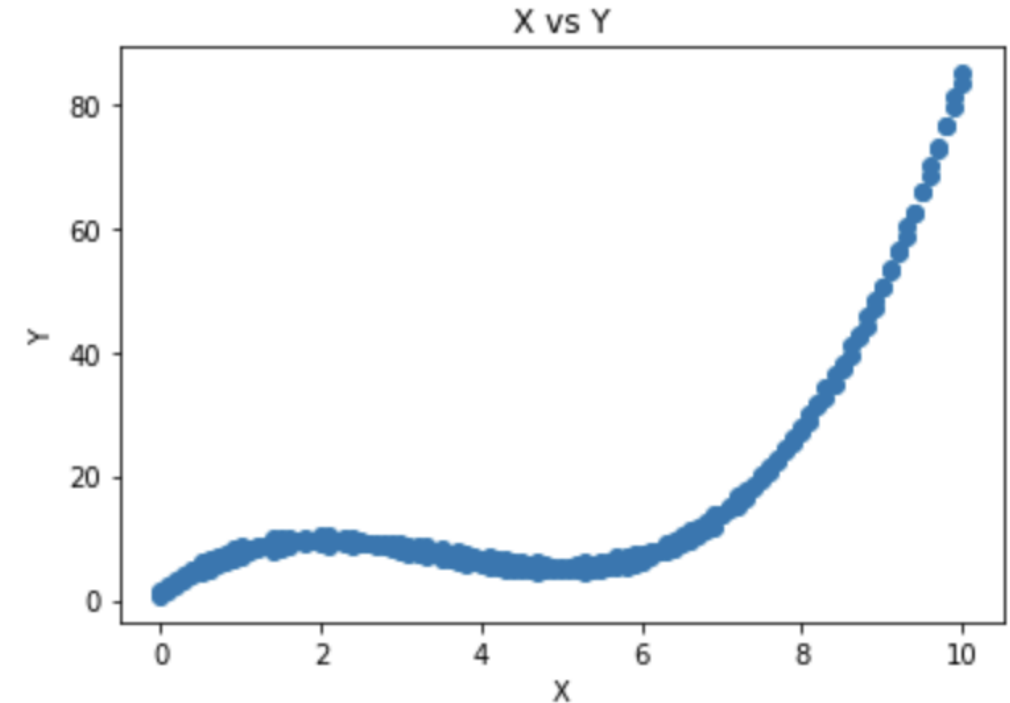
- **When it is used**

- Whenever you want to model non-linear data

- **Caution!** Be aware of overfitting - polynomials of an order higher than X^3 are rarely used

Applying our dataset with a polynomial model

- This is a fictitious dataset that warrants the use of a **3rd order polynomial model**
- The dataset is called `poly3` and has columns `y` and `x`
- `y` will be our target and `x` our predictors



Can you think of a case where you might see data like this?

Steps when using polynomial models

- 1) Import the data
- 2) EDA and data cleaning
- **3) Add polynomial terms to dataset**
- 4) Fit the log-log model
- 5) Predict with the log-log model
- 6) Evaluate model

Importing, EDA, and data cleaning

- Import data and check shape

```
poly3 = pd.read_csv(data_dir + '/poly3.csv')  
poly3.shape
```

```
(202, 2)
```

- Look at first 3 columns

```
poly3.head(3)
```

	X	Y
0	0.0	1.301945
1	0.0	0.698055
2	0.1	2.552809

- Check if there are any missing values

```
poly3.isnull().values.any()
```

```
False
```

Adding X^2 and X^3 to our dataset

- We need to add two new predictors to our dataset: x^2 and x^3
- We use the numpy function `np.power()` to achieve this
- We can pass the entire column to `np.power()`

```
poly3['X2'] = np.power(poly3['X'], 2)  
poly3['X3'] = np.power(poly3['X'], 3)
```

- Let us look at the first 3 rows again

```
poly3.head(3)
```

	X	Y	X2	X3
0	0.0	1.301945	0.00	0.000
1	0.0	0.698055	0.00	0.000
2	0.1	2.552809	0.01	0.001

Fitting the polynomial model

- Next, we fit the polynomial model: $Y = b_0 + b_1X_1 + b_2X_1^2 + b_3X_1^3$
- We first need to add a constant to our data using `sm.add_constant()`

```
poly3 = sm.add_constant(poly3)
```

- Then, we fit the model using `sm.OLS` and pass it the target column and the predictor columns

```
model_poly = sm.OLS(poly3['Y'], poly3.loc[:, ['const', 'X', 'X2', 'X3']]).fit()
```

Predicting with the polynomial model

- Now, we use the fitted polynomial model to predict
- We are not using a training and test set here, so we predict using the same x, x2, x3 variables as predictors that we also used for fitting the model

```
prediction_poly = model_poly.predict(poly3.loc[:, ['const', 'x', 'x2', 'x3']])
```

- Let us look at some of the predictions

```
prediction_poly[1:10]
```

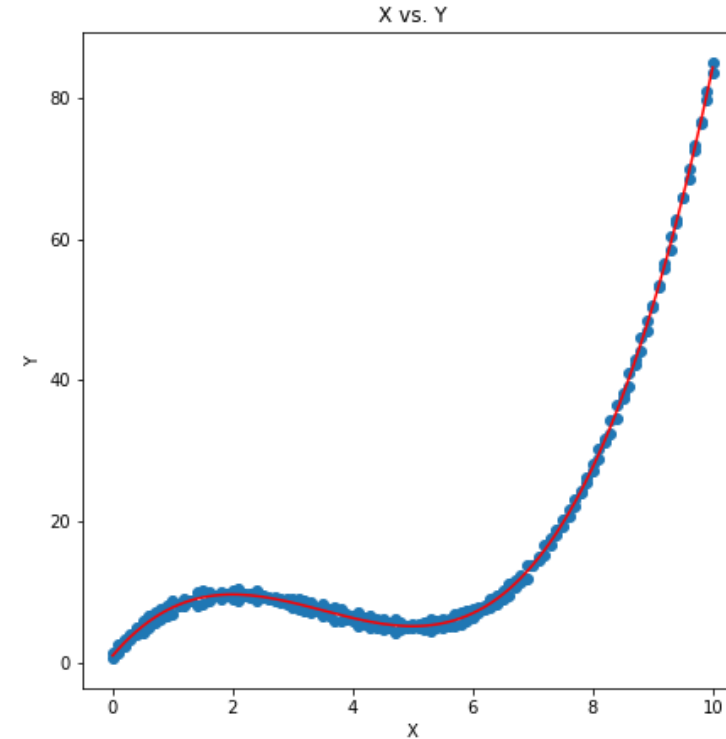
```
1    1.000000
2    1.965333
3    1.965333
4    2.862667
5    2.862667
6    3.694000
7    3.694000
8    4.461333
9    4.461333
dtype: float64
```

- We are using the whole dataset for modeling since the dataset is simulated and small

Evaluating the polynomial model: chart

- As a last step, we want to evaluate the model
- We do this with a chart and by calculating the RMSE
- Let us start by creating a chart that shows the actual values of Y and the predicted values of Y

```
plt.scatter(poly3['X'],poly3['Y'])
plt.plot(poly3['X'], prediction_poly, 'red')
plt.title("X vs. Y")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



- Wow! Our model fits the data extremely well

Evaluating the polynomial model: RMSE

- Let us also evaluate the model using RMSE
- First, we get the residuals and put them into a dataframe, together with the actual values of y and the predictions of y

```
actual = poly3['Y']
prediction = prediction_poly
residuals = actual - prediction
poly_results = pd.concat([actual.rename('actual'),
                          prediction.rename('predicted'),
                          residuals.rename('residuals')], axis = 1)
```

- Then, we use our RMSE formula to get the RMSE of our polynomial model

```
def rmse(predictions,actual):
    return np.sqrt(((prediction-actual) ** 2).mean())

print(rmse(poly_results['predicted'],poly_results['actual']))
```

```
0.5837434014764878
```

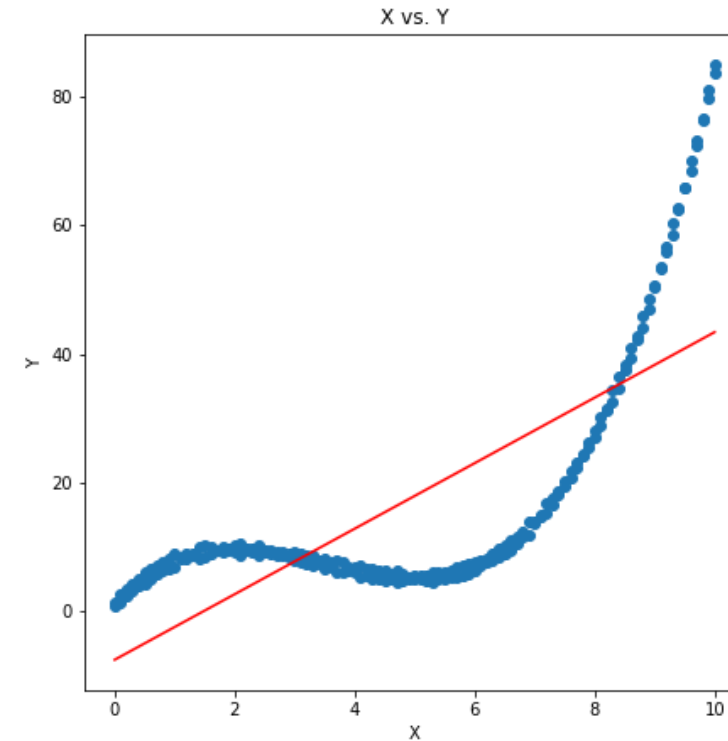

How would a linear model have performed?

- Let us see how a standard linear model would have performed

```
# Fit the model
model_lin = sm.OLS(poly3['Y'], poly3.loc[:,
['const', 'X']]).fit()
prediction = model_lin.predict(poly3.loc[:,
['const', 'X']])
```

- Get a chart of actual vs. predicted

```
plt.scatter(poly3['X'], poly3['Y'])
plt.plot(poly3['X'], prediction, 'red')
plt.title("X vs. Y")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



- Way worse!

Knowledge Check 3



Exercise 3



Module completion checklist

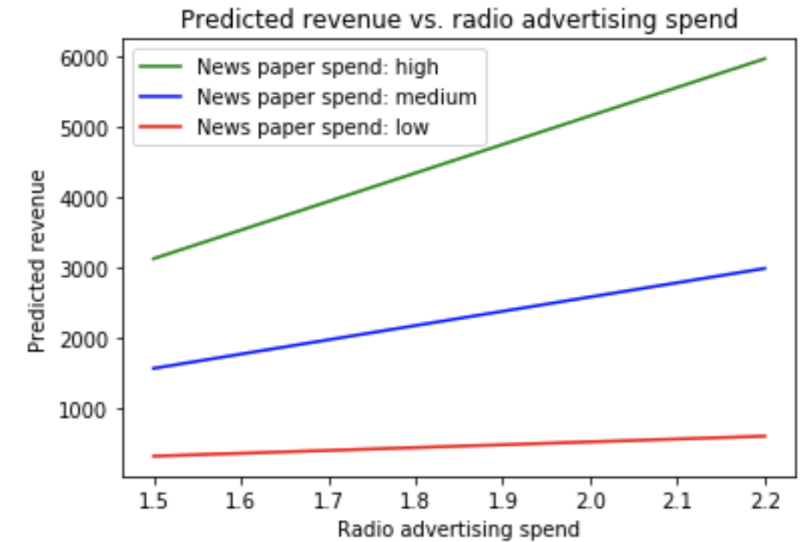
Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Discuss implications of non-linearity	✓
Analyze data with log-log models	✓
Analyze data with log-lin models	✓
Analyze data with polynomial models	✓
Understanding how to model interactions	
Analyze data using interactions in the predictor variables	

Non-additivity: modeling interactions

- So far today, we have looked at simple linear regression models where the relationship between an outcome variable and a predictor variable was non-linear
- Now, let us look at multiple linear regressions where the relationship between an outcome variable and **a set of predictors** is non-linear
- We are going to look at **interactions**
- **Definition of an interaction:**
 - An interaction occurs when the effect of an independent variable on a dependent variable changes, depending on the value of another independent variable
- **Example:**
 - Sales & Marketing: a company uses advertising in newspapers and radio to increase sales. The more they spend on advertising in newspapers, the more every additional dollar spent on advertising for radio will add to sales

Modeling interactions: example

- The chart shows the **relationship between radio ads and predicted revenue for different values of newspaper ads**
 - If spending on newspaper ads is high, increasing radio ads will increase predicted revenue by a lot
 - If spending on newspaper ads is medium, increasing radio ads will increase predicted revenue by quite a bit
 - If spending on newspaper ads is low, increasing radio ads will not increase predicted revenue by much



Modeling interactions: more examples

Healthcare

- How the severity of an injury affects healing time for varying ages of patients
- The effects of a drug on health for different genders

Sales & marketing

- The effect of additional sales personnel on sales for different values of sales area size
- The effect of variable salary percentage on sales for different values of total compensation

Politics

- The effect of income on attitude towards gun control for different community types (e.g., rural, urban, suburban)
- The effect of education on attitude towards climate change for different values of age

Sports

- The effect of additional hours of practice on performance depending on the values of years of practicing

Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Analyze data with log-log models	✓
Analyze data with log-lin models	✓
Analyze data with polynomial models	✓
Understanding how to model interactions	✓
Analyze data using interactions in the predictor variables	

Models with interactions

- **What it is**

- Interactions allow us to model data where the effect of an independent variable on a dependent variable changes, depending on the value of another independent variable

- **How it works**

- Instead of fitting a regression of Y on U and V, you fit a regression of Y on U, V, and U*V:

$$Y = b_0 + b_1 U + b_2 V + b_3 UV$$

- We need to create the values for U*V in our dataset before fitting the regression

- **When it is used**

- Sales & Marketing: A company uses advertising in newspapers and radio to increase sales. The more they spend on advertising in newspapers, the more every additional dollar spent on advertising for radio will add to sales

Our dataset to work with interactions

- This is a fictitious dataset from the Sales & Marketing department of a grocery store chain
- The dataset is called `grocery.csv` and has columns `sales`, `news` and `radio`
- `sales` will be our target, `news` and `radio` our predictors
- `sales` refers to the total sales by the grocery store chain in a particular month
- `news` refers to the dollar amount spend on advertising in newspapers in a particular month
- `radio` refers to the dollar amount spend on advertising on the radio in a particular month

Steps when using interaction models

- 1) Import the data
- 2) EDA and data cleaning
- **3) Add interaction term to dataset**
- 4) Fit the model
- 5) Predict with the model
- 6) Evaluate model

Importing, EDA, and data cleaning

- Import data and check shape

```
grocery = pd.read_csv(data_dir + '/grocery.csv')  
grocery.shape
```

```
(31, 3)
```

- Look at first 3 columns

```
grocery.head(3)
```

	sales	news	radio
0	1600.00	1.000000	1.500000
1	1640.25	1.033333	1.523333
2	1681.00	1.066667	1.546667

- Check if there are any missing values

```
grocery.isnull().values.any()
```

```
False
```

Adding the interaction to our dataset

- We need to add the interaction between `news` and `radio` to our dataset
- We create a new variable called `news:radio` and create it by multiplying `news` with `radio`

```
grocery['news:radio'] = grocery['news'] * grocery['radio']
```

- Let us look at the first 3 rows again

```
grocery.head(3)
```

	sales	news	radio	news:radio
0	1600.00	1.000000	1.500000	1.500000
1	1640.25	1.033333	1.523333	1.574111
2	1681.00	1.066667	1.546667	1.649778

Fitting the model

- Next, we fit the model $sales = b_0 + b_1news + b_2radio + b_3news : radio$
- We first need to add a constant to our data using `sm.add_constant()`

```
grocery = sm.add_constant(grocery)
```

- Then, we fit the model using `sm.OLS` and pass it the target column and the predictor columns

```
model_interact = sm.OLS(grocery['sales'], grocery.loc[:, ['const', 'news', 'radio', 'news:radio']]).fit()
```

- We are using the whole dataset for modeling since the dataset is simulated and small

Predicting with the model

- Now, we use the fitted model to predict
- We are not using a training and test set here, so we predict using the same news, radio, news:radio variables as predictors that we also used for fitting the model

```
prediction_interact = model_interact.predict(grocery.loc[:, ['const', 'news', 'radio', 'news:radio']])
```

- Let us look at some of the predictions

```
prediction_interact[1:10]
```

```
1    1640.25
2    1681.00
3    1722.25
4    1764.00
5    1806.25
6    1849.00
7    1892.25
8    1936.00
9    1980.25
dtype: float64
```

Evaluating the model: RMSE

- Let us evaluate the model using RMSE
- First, we get the residuals and put them into a dataframe, together with the actual values of sales and the predictions of sales

```
actual = grocery['sales']
prediction = prediction_interact
residuals = actual - prediction
interact_results = pd.concat([actual.rename('actual'),
                             prediction.rename('predicted'),
                             residuals.rename('residuals')], axis = 1)
```

- Then, we use our RMSE formula to get the RMSE of our interactions model

```
def rmse(predictions,actual):
    return np.sqrt(((prediction-actual) ** 2).mean())

print(rmse(interact_results['predicted'],interact_results['actual']))
```

```
5.5960811357290745e-12
```


What if we left out the interaction?

- Let us see how a model without the interaction would have performed

```
model_lin = sm.OLS(grocery['sales'], grocery.loc[:, ['const', 'news', 'radio']]).fit()
prediction_lin = model_lin.predict(grocery.loc[:, ['const', 'news', 'radio']])
actual = grocery['sales']
prediction = prediction_lin
residuals = actual - prediction
lin_results = pd.concat([actual.rename('actual'),
                        prediction.rename('predicted'),
                        residuals.rename('residuals')], axis = 1)

def rmse(predictions, actual):
    return np.sqrt(((prediction-actual) ** 2).mean())

print(round(rmse(lin_results['predicted'], lin_results['actual'])))
```

18.0

- Way worse!

Another way to evaluate the interaction

- To evaluate if including an interaction term in a model is warranted, we have two options:
 - 1. Evaluate model error and compare to model without interactions
 - **2. Look at p-value of regression coefficient of interaction term**
- Let us look at the p-value of the regression coefficient of the interaction term

	coef	std err	t	P> t	[0.025	0.975]
const	307.6291	3.15e-11	9.76e+12	0.000	307.629	307.629
news	215.1576	4.53e-11	4.75e+12	0.000	215.158	215.158
radio	396.7136	6.65e-12	5.96e+13	0.000	396.714	396.714
news:radio	321.4286	1.71e-11	1.87e+13	0.000	321.429	321.429

- The **p-value is extremely small** and therefore the coefficient of **the interaction term is significant**
- We conclude that **we should include the interaction term** in this model

Knowledge Check 4



Exercise 4



Module completion checklist

Objective	Complete
Recap model assumptions and ways to evaluate models	✓
Present cases where assumptions are violated	✓
Analyze data with log-log models	✓
Analyze data with log-lin models	✓
Analyze data with polynomial models	✓
Understanding how to model interactions	✓
Analyze data using interactions in the predictor variables	✓

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
- This is an exploratory exercise to get you comfortable with the content we discussed today
- **Tasks for today:**
 - Evaluate whether the dataset you chose has any non-linearity or interactions
 - Try different model specifications with polynomials and interactions
 - Evaluate whether the inclusion of polynomials and interactions improves your model

This completes module!
Congratulations!