SCOPING A PROJECT

*Copyright Maven Analytics, LLC

SCOPING A PROJECT

In this section we'll discuss the process of **scoping a data science project**, from understanding your end users to deciding which tools and techniques to deploy

TOPICS WE'LL COVER:

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

GOALS FOR THIS SECTION:

- Outline the key steps for defining a clear and effective data science project scope
- Learn how to collaborate with end users to understand the business context, explore potential problems and solutions, and align on goals
- Identify which types of solutions would require supervised vs. unsupervised techniques
- Review common data requirements, including structure, features, sources and scope

*Copyright Maven Analytics, LLC

SCOPING A PROJECT

Scoping a data science project means clearly defining the goals, techniques, and data sources you plan to use for your analysis

Scoping steps:

1. Think like an end user
2. Brainstorm problems
3. Brainstorm solutions
4. Determine the techniques
5. Identify data requirements
6. Summarize the scope & objectives

Project scoping is one of the **most difficult yet important steps** of the data science workflow

If a project is not properly scoped, a lot of time can be wasted going down a path to create a solution that solves the wrong problem

MAVEN MUSIC
SINCE 1996

We'll be **scoping the course project** in this section together to set you up for success throughout the rest of the course

*Copyright Maven Analytics, LLC

THINK LIKE AN END USER

An **end user** (also known as a stakeholder) is a person, team, or business that will ultimately benefit from the results of your analysis

When introduced to a new project, start by asking questions that help you:

- **Empathize** with the end user – what do they care about?
- Focus on **impact** – what metrics are important to them?

Data Scientist

Debbie Dayda
Data Science Team

What's the situation?

People keep cancelling their streaming music subscriptions

Why is this a major issue?

Our monthly revenue growth is slowing

What would success look like for you?

Decreasing cancellation rate by 2% would be a big win

End User

Carter Careswell
Customer Care Team

*Copyright Maven Analytics, LLC

BRAINSTORM PROBLEMS

Before thinking about potential solutions, it helps to **brainstorm problems** with the end user to identify improvements and opportunities

- The ideas do not have to be data science related
- The sky is the limit – don't think about your data or resource limitations
- You want to start by thinking big and then whittle it down from there

Data Scientist

Debbie Dayda
Data Science Team

Is our product differentiated from competitors?

Are technical bugs or limitations to blame?

Is our product too expensive?

Do we need to expand our music library?

Do we even know WHY customers are cancelling?

That's a good one – let's focus on that problem first!

End User

Carter Careswell
Customer Care Team

*Copyright Maven Analytics, LLC

BRAINSTORM SOLUTIONS

Once you settle on a problem, the next step is to **brainstorm potential solutions**

Data science can be *one* potential solution, but it's not the only one:

- **PROS**: Solutions are backed by data, may lead to hidden insights, let you make predictions
- **CONS**: Projects take more time and specialized resources, and can lead to complex solutions

POTENTIAL SOLUTIONS

*Ask the product team to **add a survey** to capture cancellation feedback*

***Conduct customer interviews** to gather qualitative data and insights*

*Suggest that the leadership team **speak with other leaders** in the space to compare notes*

***Speak with customer reps** about any changes they've noticed in recent customer interactions*

***Research external factors** that may be impacting cancellations (competitive landscape, news, etc.)*

*Use data to **identify the top predictors** for account cancellations*

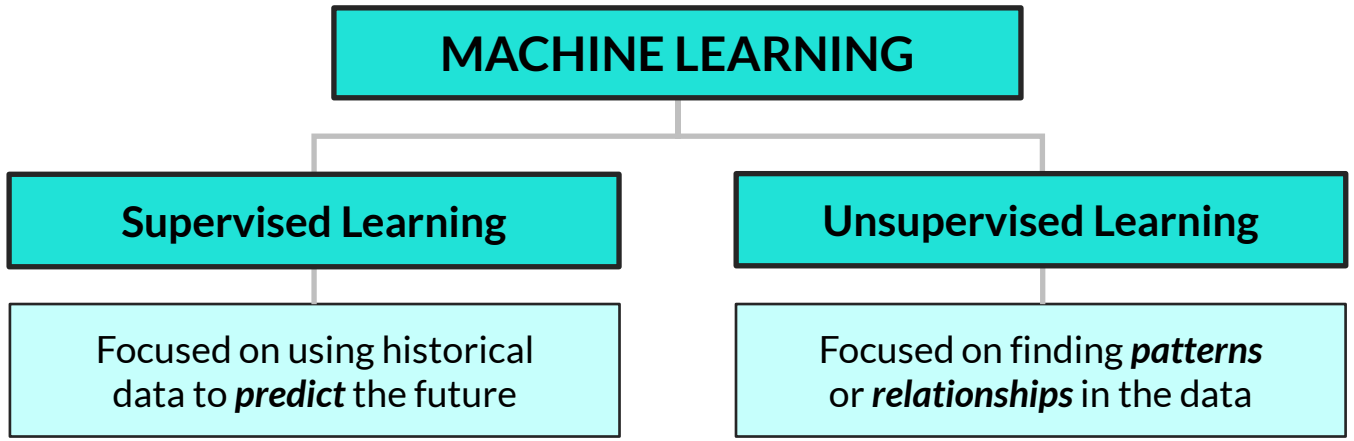
This is the only data science solution!

*Copyright Maven Analytics, LLC

DETERMINE THE TECHNIQUES

If you decide to take a data science approach to solving the problem, the next step is to **determine which techniques** are most suitable:

- Do you need supervised or unsupervised learning?



*Copyright Maven Analytics, LLC

KNOWLEDGE CHECK

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

Estimating how many customers will visit your website on New Year's Day

Looking at the products purchased by the highest-spend customers

Visualizing cancellation rate over time for various customer segments

Identifying the main themes mentioned in customer reviews

Clustering customers into different groups based on their preferences

Flagging which customers are most likely to cancel their membership

Supervised Learning

Unsupervised Learning

*Copyright Maven Analytics, LLC

KNOWLEDGE CHECK

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- Data Requirements
- Summarizing the Scope

Looking at the products purchased by the highest-spend customers

Visualizing cancellation rate over time for various customer segments

Supervised Learning

Estimating how many customers will visit your website on New Year's Day

Flagging which customers are most likely to cancel their membership

Unsupervised Learning

Identifying the main themes mentioned in customer reviews

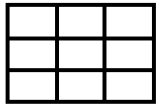
Clustering customers into different groups based on their preferences

*Copyright Maven Analytics, LLC

IDENTIFY DATA REQUIREMENTS

- Project Scoping Steps
- Thinking Like an End User
- Problems & Solutions
- Modeling Techniques
- **Data Requirements**
- Summarizing the Scope

After deciding on a technique, the next step is to **identify data requirements**, including structure, features, sources and scope

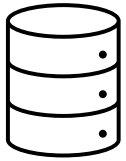


Structure

You will need to structure your data in different ways, whether you are using supervised or unsupervised techniques

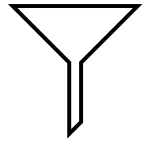
Features

Brainstorm specific columns or "features" that might provide insight (these will be good inputs for your models)



Sources

Determine which additional data sources (if any) are required to create or "engineer" those features



Scope

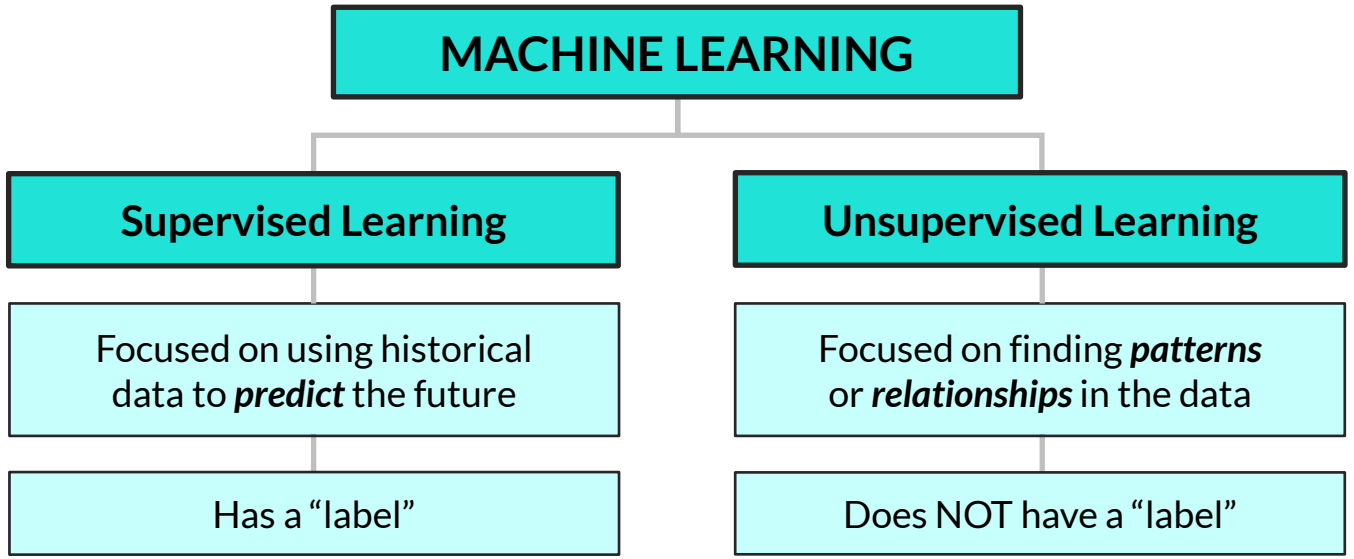
Narrow down your data to just what you need to get started (you can expand and iterate from there)

*Copyright Maven Analytics, LLC

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data



*A "label" is an **observed variable which you are trying to predict***

(house price ($), spam or not (1/0), chance of failure (probability), etc.)

*Copyright Maven Analytics, LLC

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data

EXAMPLE | ***Supervised Learning**: Predicting which customers are likely to cancel*

x variables = inputs = features
(what goes into a model)

y variable = output = target
(what you are trying to predict)

Customer	Months Active	Monthly Payment	Listened to Indie Artists?	Cancelled?
Aria	25	$9.99	Yes	No
Chord	2	$0	No	No
Melody	14	$14.99	No	Yes
When shown a new customer, predict whether they will cancel or not				
Rock	12	$9.99	Yes	**???**

*Each row represents a **customer***

*Each cancellation value is called the **label** for the row*

*Copyright Maven Analytics, LLC

DATA STRUCTURE

How you **structure your data** often depends on which technique you're using:

- A **supervised** learning model takes in **labeled** data *(the outcome you want to predict)*
- An **unsupervised** learning model takes in **unlabeled** data

EXAMPLE | ***Unsupervised Learning***: *Clustering customers based on listening behavior*

x variables = inputs = features
(what goes into a model)

Each row represents a ***customer*** →

Customer	Daily Listening Hours	# Pop Songs	# Soundtracks	# Podcasts
Aria	10	50	3	4
Chord	3	28	0	0
Melody	2	0	0	3

When shown a new customer, figure out which customers it's most similar to

Rock	1	4	0	2

*Copyright Maven Analytics, LLC

MODEL FEATURES

To identify relevant **model features**, brainstorm any potential variables that might be useful as model inputs based on your goal, for example:

- Supervised learning – features that will do a good job *predicting* cancellations
- Unsupervised learning – features that will do a good job *differentiating* customers

Data Scientist

Debbie Dayda
Data Science Team

What factors might help us predict cancellations?

Well, we recently increased the monthly subscription rate

It could also have something to do with auto-renewals

I wonder if certain demographics are more likely to cancel

Maybe a competitor launched a new offer or promotion?

End User

Carter Careswell
Customer Care Team

*Copyright Maven Analytics, LLC

DATA SOURCES

Once you've identified a list of relevant features, **brainstorm data sources** you can use to create or engineer those features

Features:	Potential Sources:	Ease of Access:
Monthly rate	Customer subscription history, customer listening history	Easy *(internal data)*
Auto-renew		
Age	Customer demographics	
Urban vs. Rural		
Competitor Promotion	Customer's other subcriptions, competitor promotion history	Hard *(external data)*

*Copyright Maven Analytics, LLC

DATA SCOPE

Next, **narrow the scope of your data** to remove sources that are difficult to obtain and prioritize the rest

- Remember that more data doesn't necessarily mean a better model!

Data Sources:

- Customer subscription history
- Customer listening history

Timeframe:
Past three months of data

Customers:
Only customers who actively subscribed, not those who were grandfathered in

PRO TIP: Aim to produce a **minimum viable product** (MVP) at this stage – if you go too deep without feedback, you risk heading down unproductive paths or over-complicating the solution

*Copyright Maven Analytics, LLC

SUMMARIZE THE SCOPE & OBJECTIVES

The final step is to **clearly summarize the project scope and objectives**:

- What techniques and data do you plan to leverage?
- What specific impact are you trying to make?

Data Scientist

Debbie Dayda
Data Science Team

*We plan to use **supervised learning** to predict which customers are likely to cancel their subscription, using **the past three months of subscription and listening history**. This will allow us to:*

- *Identify the **top predictors for cancellation** and figure out how to address them*
- *Use the model to **flag customers who are likely to cancel** and take proactive steps to keep them subscribed*

*Our goal is to **reduce cancellations by 2%** over the next year.*

End User

Carter Careswell
Customer Care Team

Sweet!

*Copyright Maven Analytics, LLC

KEY TAKEAWAYS

When scoping a project, start by **thinking like an end user**

- *Take time to sit down with end users or stakeholders to understand the situation and business context, brainstorm potential problems and solutions, and align on key objectives*

Don't limit yourself when brainstorming ideas for problems, solutions, or data

- *Start by thinking big and whittling ideas down from there, and keep in mind that many potential solutions likely won't require data science*

Supervised & unsupervised models require **different data structures**

- *Supervised models use labeled data which includes a target variable for the model to predict, while unsupervised models use unlabeled data to find patterns or relationships*

Leverage the **MVP approach** when working on data science projects

- *Start with something relatively quick and simple as a proof of concept, then continuously iterate to refine and improve your model once you know you're on the right track*

*Copyright Maven Analytics, LLC

GATHERING DATA

*Copyright Maven Analytics, LLC

GATHERING DATA

In this section we'll cover the steps for **gathering data**, including reading files into Python, connecting to databases within Python, and storing data in Pandas DataFrames

TOPICS WE'LL COVER:

- Data Gathering Process
- Reading Files Into Python
- Connecting to a Database
- Exploring DataFrames

GOALS FOR THIS SECTION:

- Understand the data gathering process and the various ways that data can be stored and structured
- Identify the characteristics of a Pandas DataFrame
- Use Pandas to read in flat files and Excel spreadsheets, and connect to SQL databases
- Quickly explore a Pandas DataFrame

*Copyright Maven Analytics, LLC

DATA GATHERING PROCESS

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

The **data gathering process** involves finding data, reading it into Python, transforming it if necessary, and storing the data in a Pandas DataFrame

1) Find the data

```
{
  "Name": ["Jim", "Dwight",
           "Angela", "Tobi"],
  "Age":  [26, 28, 27, 33],
  "Department": ["Sales", "Sales",
                 "Accounting",
                 "Human Resources"]
}
```

Raw data can come in many shapes and forms

2) Read in the data

Apply transformations using Python, if necessary

3) Store the data

	Name	Age	Department
0	Jim	26	Sales
1	Dwight	28	Sales
2	Angela	27	Accounting
3	Tobi	33	Human Resources

*Tables in Python are stored as **Pandas DataFrames** (more on this later!)*

*Copyright Maven Analytics, LLC

DATA SOURCES

Data Gathering Process

Reading Files into Python

Connecting to a Database

Exploring DataFrames

Data can come from a variety of **sources**:

Local files

You can read data from a file stored on your computer

Common examples:

- Flat files *(.csv, .txt, etc.)*
- Spreadsheets *(.xlsx, etc.)*

Databases

You can connect to a database and write queries

Common examples:

- SQL databases
- NoSQL databases

Web access

You can programmatically extract data from websites

Common examples:

- Web scraping *(.html, etc.)*
- API *(.json, .xml, etc.)*

Python is a great tool for data gathering due to its ability to read in and transform data coming from a **wide variety of sources** and formats

*Copyright Maven Analytics, LLC

STRUCTURED VS UNSTRUCTURED DATA

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

Structured

Already stored as tables

.xlsx

.db

Semi-Structured

Easily converted to tables

.csv

.json

Unstructured

Not easily converted to tables

.pdf

.jpg

To do analysis, data needs to be **structured as a table** with rows and columns

- Most data sources are already organized in this way and are ready for analysis
- Unstructured data sources require additional transformations before doing analysis

*Copyright Maven Analytics, LLC

THE PANDAS DATAFRAME

Data Gathering Process

Reading Files into Python

Connecting to a Database

Exploring DataFrames

To do analysis in Python, data must reside within a **Pandas DataFrame**

- DataFrames look just like regular tables with rows and columns, but with additional features
- External structured and semi-structured data can be read directly into a DataFrame

*The **row index** assigns a unique ID to each row*

(The index starts at 0)

	Course ID	Course Name	Instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

*Each column contains data of a **single data type***

(integer, text, etc.)

*Copyright Maven Analytics, LLC

READING DATA INTO PYTHON

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

Pandas allows you to **read in data** from multiple file formats:

pd.read_csv	*Reads in data from a delimiter-separated flat file*	**pd.read_csv(*file path*, *sep*, *header*)**
pd.read_excel	*Reads in data from a Microsoft Excel file*	**pd.read_excel(*file path*, *sheet_name*)**
pd.read_json	*Reads in data from a JSON file*	**pd.read_json(*file path*)**

'pd' is the standard alias for the Pandas library

*Copyright Maven Analytics, LLC

READING FLAT FILES

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

pd.read_csv() lets you read flat files by specifying the file path

```
pd.read_csv(file path, sep, header)
```

The file location, name and extension

Examples:

- *'data.csv'*
- *'sales/october.tsv'*

The column delimiter

Examples:

- *sep=',' (default)*
- *sep='\t'*

If the data has column headers in the first row

- *header='infer' (default)*
- *header=None (no headers)*

PRO TIP: Place the file in the same folder as the Jupyter Notebook so you don't have to specify the precise file location

For a full list of arguments, visit: ***https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html***

*Copyright Maven Analytics, LLC

READING FLAT FILES

pd.read_csv() lets you read flat files by specifying the file path

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

course_offerings.csv

```
course_id,course_name,instructor
101,Intro to Python,Chris Bruehl
102,Intro to SQL,John Pauler
201,Exploratory Data Analysis,Alice Zhao
301,Algorithms,Chris Bruehl
331,Natural Language Processing,Alice Zhao
```

```
import pandas as pd

pd.read_csv('course_offerings.csv')
```

	course_id	course_name	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

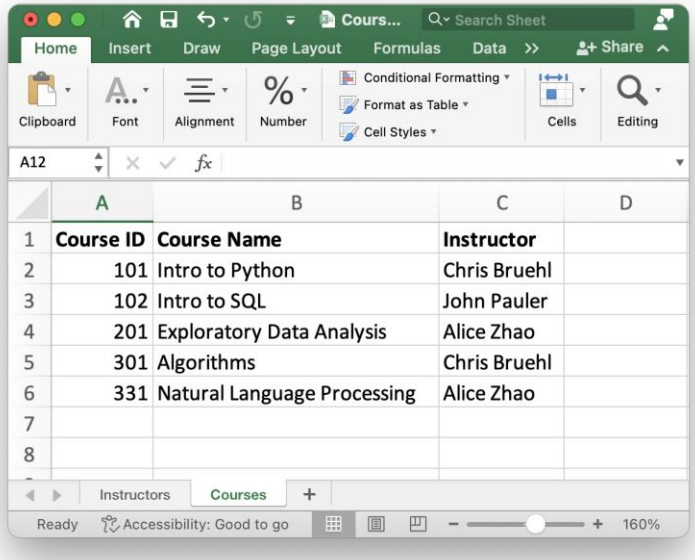
*Copyright Maven Analytics, LLC

READING EXCEL FILES

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

pd.read_excel() lets you read Excel files by specifying the file path

- You can use the "sheet_name" argument to specify the worksheet *(default is 0 – first sheet)*




```
import pandas as pd

pd.read_excel('course_offerings.xlsx', sheet_name=1)
```

	Course ID	Course Name	Instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

*Copyright Maven Analytics, LLC

CONNECTING TO A SQL DATABASE

To connect to a SQL database, import a database driver and specify the database connection, then use **pd.read_sql()** to query the database using SQL code


```
# Connect to a SQLite database

import sqlite3
conn = sqlite3.connect('maven.db')

pd.read_sql('SELECT * FROM courses', conn)
```

	course_id	course	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao
3	301	Algorithms	Chris Bruehl
4	331	Natural Language Processing	Alice Zhao

SQL Software	Database Driver
SQLite	sqlite3
MySQL	mysql.connector
Oracle	cx_Oracle
PostgreSQL	psycopg2
SQL Server	pyodbc

*Copyright Maven Analytics, LLC

QUICKLY EXPLORE A DATAFRAME

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

After reading data into Python, it's common to **quickly explore the DataFrame** to make sure the data was imported correctly

df.head()	*Display the first five rows of a DataFrame*
df.shape	*Display the number of rows and columns of a DataFrame*
df.count()	*Display the number of values in each column*
df.describe()	*Display summary statistics like mean, min and max*
df.info()	*Display the non-null values and data types of each column*

*Copyright Maven Analytics, LLC

QUICKLY EXPLORE A DATAFRAME

- Data Gathering Process
- Reading Files into Python
- Connecting to a Database
- Exploring DataFrames

After reading data into Python, it's common to **quickly explore the DataFrame** to make sure the data was imported correctly

```
df.head(3)
```

	course_id	course_name	instructor
0	101	Intro to Python	Chris Bruehl
1	102	Intro to SQL	John Pauler
2	201	Exploratory Data Analysis	Alice Zhao

```
df.shape
```

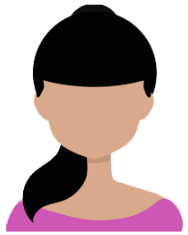
```
(5, 3)
```

```
df.describe()
```

	course_id
count	5.000000
mean	207.200000
std	107.829495
min	101.000000
25%	102.000000
50%	201.000000
75%	301.000000
max	331.000000

*Copyright Maven Analytics, LLC

ASSIGNMENT: READ A FILE INTO PYTHON



NEW MESSAGE

June 1, 2023

From: **Anna Analysis** *(Senior Data Scientist)*

Subject: **New Survey Data**

Hi there,

I hear you're the new junior data scientist on the team. Welcome!

Our polling partners just finished collecting survey data on the happiness levels in each country. I've attached the file.

Can you load the data into Python and confirm the number of rows in the data and the range of the happiness scores? Then we can walk through next steps together.

Thanks!
Anna

happiness_survey_data.csv

Reply Forward

Key Objectives

1. Read in data from a .csv file
2. Store the data in a DataFrame
3. Quickly explore the data in the DataFrame

*Copyright Maven Analytics, LLC

KEY TAKEAWAYS

Python can **read data** from your computer, a database, or the internet

- *Flat files & spreadsheets are typically stored locally on a computer, but as a data scientist you'll likely need to connect to and query a SQL database, and potentially collect data through web scraping or an API*

Data needs to be organized into **rows** & **columns** for analysis

- *Flat files, spreadsheets and SQL tables are already in this format, but if your data is unstructured you would need to extract the relevant data and transform it into rows and columns yourself*

Python tables are known as Pandas **DataFrames**

- *DataFrames include a unique row index and each column must be the same data type (integer, text, etc.)*
- *There are several ways to quickly explore the characteristics of a DataFrame (head, shape, describe, etc.)*

*Copyright Maven Analytics, LLC

CLEANING DATA

*Copyright Maven Analytics, LLC

CLEANING DATA

In this section we'll cover the steps for **cleaning data**, including converting columns to the correct data type, handling data issues and creating new columns for analysis

TOPICS WE'LL COVER:

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

GOALS FOR THIS SECTION:

- Identify and convert between Python data types
- Find common "messy" data issues and become familiar with the different ways to resolve them
- Create new calculated columns from existing data

*Copyright Maven Analytics, LLC

DATA CLEANING OVERVIEW

The goal of **data cleaning** is to get raw data into a format that's ready for analysis

This includes:

- Converting columns to the correct **data types** for analysis
- Handling **data issues** that could impact the results of your analysis
- **Creating new columns** from existing columns that are useful for analysis

The order in which you complete these data cleaning tasks will vary by dataset, but this is a good starting point

Even though there are automated tools available, doing some **manual data cleaning** provides a good opportunity to start understanding and getting a good feel for your data

*Copyright Maven Analytics, LLC

DATA TYPES

When using Pandas to read data, columns are automatically assigned a **data type**

- Use the **.dtypes** attribute to view the data type for each DataFrame column
- Note that sometimes numeric columns *(int, float)* and date & time columns *(datetime)* aren't recognized properly by Pandas and get read in as text columns *(object)*

```
df = pd.read_csv('customers.csv')
df.head()
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	$45,000	1970-09-12
1	Joseph Martinez	Portland	OR	$120,000	1960-12-10
2	Joseph Martinez	Portland	OR	$120,000	1960-12-10
3	Wayne Nielson	Dahlgren	VA	$75,000	1968-12-05
4	Beverly Nixon	Long Island	NY	$45,000,000	1962-03-05

Default data types

```
df.dtypes
```

```
Name          object
City          object
State         object
Income        object
Birthdate     object
dtype: object
```

*These need to be **converted** to be analyzed!*

Data Type	Description
bool	Boolean, True/False
int64	Integers
float64	Floating point, decimals
object	Text or mixed values
datetime64	Date and time values

PRO TIP: Use the **.info()** method as an alternative to show additional information along with the data types

*Copyright Maven Analytics, LLC

CONVERTING TO DATETIME

Use **pd.to_datetime()** to convert object columns into datetime columns

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df.Birthdate
```

```
0     1970-09-12
1     1960-12-10
2     1960-12-10
3     1968-12-05
4     1962-03-05
5     1993-09-23
6     1996-03-12
7     1994-01-13
8     1967-08-27
9            NaN
10           NaN
11           NaN
12    1968-05-19
Name: Birthdate, dtype: object
```

```
# convert to a datetime column
df.Birthdate = pd.to_datetime(df.Birthdate)
df.Birthdate
```

```
0    1970-09-12
1    1960-12-10
2    1960-12-10
3    1968-12-05
4    1962-03-05
5    1993-09-23
6    1996-03-12
7    1994-01-13
8    1967-08-27
9           NaT
10          NaT
11          NaT
12   1968-05-19
Name: Birthdate, dtype: datetime64[ns]
```

Note that the missing values are now NaT instead of NaN (more on missing data later!)

PRO TIP: Pandas does a pretty good job at detecting the datetime values from an object if the text is in a standard format (like "YYYY-MM-DD"), but you can also manually specify the format using the format argument within the pd.to_datetime() function: **pd.to_datetime(dt_col, format='%Y-%M-%D')**

For a full list of formats, visit: ***https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior***

*Copyright Maven Analytics, LLC

CONVERTING TO NUMERIC

Use **pd.to_numeric()** to convert object columns into numeric columns

- To remove non-numeric characters ($, %, etc.), use **str.replace()**

```
df.Income
```

```
0         $45,000
1        $120,000
2        $120,000
3         $75,000
4     $45,000,000
5         $50,000
6        $105,000
7         $62,000
8         $60,000
9             NaN
10        $80,000
11            NaN
12       $110,000
Name: Income, dtype: object
```

Currency data is read in as an object by Python due to the dollar sign ($) and comma separator (,)

```
# remove all dollar signs and commas
clean_income = df.Income.str.replace('$','').str.replace(',','')
clean_income
```

```
0        45000
1       120000
2       120000
3        75000
4     45000000
5        50000
6       105000
7        62000
8        60000
9          NaN
10       80000
11         NaN
12      110000
Name: Income, dtype: object
```

```
# convert to a numeric column
df.Income = pd.to_numeric(clean_income)
```

```
df.dtypes
```

```
Name                    object
City                    object
State                   object
Income                 float64
Birthdate    datetime64[ns]
dtype: object
```

An alternative is to use Series.astype() to convert to more specific data types like 'int', 'float', 'object' and 'bool', but **pd.to_numeric() can handle missing values** (NaN), while Series.astype() cannot

*Copyright Maven Analytics, LLC

ASSIGNMENT: CONVERTING DATA TYPES

NEW MESSAGE

June 12, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Data in Python Request**

Hi there,

We just finished collecting survey data from a few thousand customers who've purchased our alarm clocks (see attached).

Can you read the data into Python and make sure the data type of each column makes sense? We'd like to do quite a few calculations using the data, so if a column can be converted to a numeric or datetime column, please do so.

Thanks!
Alan

Alarm Survey Data.xlsx

Reply Forward

Key Objectives

1. Read in data from the Excel spreadsheet and store it in a Pandas DataFrame
2. Check the data type of each column
3. Convert object columns into numeric or datetime columns as needed

*Copyright Maven Analytics, LLC

DATA ISSUES OVERVIEW

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Data issues need to be identified and corrected upfront in order to not impact or skew the results of your analysis

Common "messy" data issues include:

Duplicates

Inconsistent text & typos

Name	City	State	Income	Birthdate
Susan Rodriguez	Maplewood	NJ	$45,000	1970-09-12
Joseph Martinez	Portland	OR	$120,000	1960-12-10
Joseph Martinez	Portland	OR	$120,000	1960-12-10
Wayne Nielson	Dahlgren	VA	$75,000	1968-12-05
Beverly Nixon	Long Island	NY	$45,000,000	1962-03-05
Veronica Comerford	Tuskegee	AL	$50,000	1993-09-23
Ivan Layton	Albany	New York	$105,000	1996-03-12
Laura Hailey	Ephraim	Utah	$62,000	1994-01-13
John Depaul	Baton Rouge	LA	$60,000	1967-08-27
James Weiss	Houston	TX		
Addie Stevenson	Jacksonville		$80,000	
Daniel Long				
Queen Watson	Los Angeles	CA	$110,000	1968-05-19

Outliers

Missing data

*Copyright Maven Analytics, LLC

MISSING DATA

There are various ways to represent **missing data** in Python

- **np.NaN** – NumPy's NaN is the most common representation *(values are stored as floats)*
- **pd.NA** – Pandas' NA is a newer missing data type *(values can be stored as integers)*
- **None** – Base Python's default missing data type *(doesn't allow numerical calculations)*

```
df = pd.read_csv("customers.csv")
df.tail()
```

	Name	City	State	Income	Age
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Missing values are treated as **np.NaN** when data is read into Pandas*

*Copyright Maven Analytics, LLC

IDENTIFYING MISSING DATA

The easiest way to identify missing data is with the **.isna()** method

- You can also use **.info()** or **.value_counts(dropna=False)**

```
df.isna()
```

	Name	City	State	Income	Age
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	True	True
10	False	False	True	False	True
11	False	True	True	True	True
12	False	False	False	False	False

*This returns **True** for any missing values*

```
df.isna().sum()
```

```
Name      0
City      1
State     2
Income    2
Age       3
dtype: int64
```

*Use **sum()** to return the missing values by column*

```
df[df.isna().any(axis=1)]
```

	Name	City	State	Income	Age
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN

*Or use **any(axis=1)** to select the rows with missing values*

*Copyright Maven Analytics, LLC

HANDLING MISSING DATA

There are multiple ways to **handle missing data**:

- Keep the missing data as is
- Remove an entire row or column with missing data
- Impute missing numerical data with a 0 or a substitute like the average, mode, etc.
- Resolve the missing data based on your domain expertise

```
df[df.isna().any(axis=1)]
```

	Name	City	State	Income	Age
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN

Missing age values are likely ok to keep

We have no data on this customer

This is likely Jacksonville, FL

Can we calculate an approximate income?

There is no right or wrong way to deal with missing data, which is why it's important to **be thoughtful and deliberate** in how you handle it

*Copyright Maven Analytics, LLC

KEEPING MISSING DATA

You can still perform calculations if you choose to **keep missing data**

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df["Age"].mean()
```

```
48.5
```

```
df["Age"].count()
```

```
10
```

These ignore the missing values

*Copyright Maven Analytics, LLC

REMOVING MISSING DATA

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

The **.dropna()** method removes rows with missing data

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

Our original data set with all of its rows, including all missing values

*Copyright Maven Analytics, LLC

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Note that the row index is now skipping values, but you can reset the index with **df.dropna().reset_index()***

*Copyright Maven Analytics, LLC

REMOVING MISSING DATA

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Daniel Long	NaN	NaN	NaN	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

Removes rows that don't have at least "n" values

```
df.dropna(thresh=2)
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

REMOVING MISSING DATA

The **.dropna()** method removes rows with missing data

Removes any rows with NaN values

```
df.dropna()
```

Removes rows that only have NaN values

```
df.dropna(how="all")
```

Removes rows that don't have at least "n" values

```
df.dropna(thresh=2)
```

Removes rows with NaN values in a specified column

```
df.dropna(subset=["City"])
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

PRO TIP: KEEPING NON-MISSING DATA

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

You can use **.notna()** to keep non-missing data instead

```
df[df["City"].notna()]
```

Note that using .dropna() or .notna() to remove rows with missing data **does not make permanent changes** to the DataFrame, so you need to save the output to a new DataFrame (or the same one) or set the argument inplace=True

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	NaN	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
12	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

IMPUTING MISSING DATA

The **.fillna()** method imputes missing data with an appropriate value

- There are many ways to impute data in a column *(zero, mean, mode, etc.)*, so take a moment to decide the best one – if you're unsure, reach out to a subject matter expert

```
df[["Income"]]
```

	Income
0	45000.0
1	120000.0
2	120000.0
3	75000.0
4	45000000.0
5	50000.0
6	105000.0
7	62000.0
8	60000.0
9	NaN
10	80000.0
11	110000.0

What value can be used here?

```
df["Income"] = df["Income"].fillna(df["Income"].median())
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

Using the median removes the impact of the outlier

*Copyright Maven Analytics, LLC

RESOLVING MISSING DATA

An alternative to imputing is to **resolve missing data** with domain expertise

- You can use the **.loc[]** accessor to update a specific value

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	NaN	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df.loc[10,"State"] = "FL"
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

ASSIGNMENT: MISSING DATA

NEW MESSAGE

June 13, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Missing Data Check**

Hi again,

Can you check the file I sent you yesterday for missing data?

Please use your best judgement when choosing how to handle the missing data and let me know what approaches you decide to take.

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any missing data
2. Deal with the missing data

*Copyright Maven Analytics, LLC

INCONSISTENT TEXT & TYPOS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Inconsistent text & typos in a data set are represented by values that are either:

- Incorrect by a few digits or characters
- Inconsistent with the rest of a column

```
df.head(8)
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	New York	105000.0	27.0
7	Laura Hailey	Ephraim	Utah	62000.0	29.0

These are full state names, while the rest of the column has abbreviations

Finding inconsistent text and typos within a large data set in Python is **not a straightforward approach**, as there is no function that will automatically identify these situations

*Copyright Maven Analytics, LLC

IDENTIFYING INCONSISTENT TEXT & TYPOS

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

While there is no specific method to **identify inconsistent text & typos**, you can take the following two approaches to check a column depending on its data type:

Categorical data

Look at the unique values in the column

```
df["State"].value_counts()
```

```
OR          2
NJ          1
VA          1
NY          1
AL          1
New York    1
Utah        1
LA          1
TX          1
FL          1
CA          1
Name: State, dtype: int64
```

These represent the same thing!

Numerical data

Look at the descriptive stats of the column

```
df["Age"].describe()
```

```
count     10.000000
mean      48.500000
std       14.370108
min       27.000000
25%       34.750000
50%       54.000000
75%       59.500000
max       62.000000
Name: Age, dtype: float64
```

This looks like a realistic age range

*Copyright Maven Analytics, LLC

HANDLING INCONSISTENT TEXT & TYPOS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

You can fix **inconsistent text & typos** by using:

1. **.loc[]** to update a value at a particular location
2. **np.where()** to update values in a column based on a conditional statement
3. **.map()** to map a set of values to another set of values
4. **String methods** like str.lower(), str.strip() & str.replace() to clean text data

We've already covered the **loc[] accessor** to resolve missing data using domain expertise

*Copyright Maven Analytics, LLC

UPDATE VALUES BASED ON A LOGICAL CONDITION

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Use **np.where()** to update values based on a logical condition

```
np.where(condition, if_true, if_false)
```

Calls the NumPy library

A logical expression that evaluates to True or False

Value to return when the expression is True

Value to return when the expression is False

This is **different from the Pandas where method**, which has similar functionality, but different syntax

The NumPy function is used more often than the Pandas method because np.where is vectorized, meaning it executes faster

*Copyright Maven Analytics, LLC

UPDATE VALUES BASED ON A LOGICAL CONDITION

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

Use **np.where()** to update values based on a logical condition

```
df
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	45000.0	1970-09-12
1	Joseph Martinez	Portland	OR	120000.0	1960-12-10
2	Joseph Martinez	Portland	OR	120000.0	1960-12-10
3	Wayne Nielson	Dahlgren	VA	75000.0	1968-12-05
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23
6	Ivan Layton	Albany	New York	105000.0	1996-03-12
7	Laura Hailey	Ephraim	Utah	62000.0	1994-01-13

If a State value is equal to 'Utah', then replace it with 'UT',
Otherwise, keep the value that was already in the State column

```
import numpy as np

df.State = np.where(df.State == 'Utah', 'UT', df.State)
df
```

	Name	City	State	Income	Birthdate
0	Susan Rodriguez	Maplewood	NJ	45000.0	1970-09-12
1	Joseph Martinez	Portland	OR	120000.0	1960-12-10
2	Joseph Martinez	Portland	OR	120000.0	1960-12-10
3	Wayne Nielson	Dahlgren	VA	75000.0	1968-12-05
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23
6	Ivan Layton	Albany	New York	105000.0	1996-03-12
7	Laura Hailey	Ephraim	UT	62000.0	1994-01-13

*Copyright Maven Analytics, LLC

MAP VALUES

Use **.map()** to map values from one set of values to another set of values


```
# map multiple state values to one abbreviation
state_mapping = {'AL': 'AL',
                 'Alabama': 'AL',
                 'NY': 'NY',
                 'New York': 'NY'}
```

You can pass a dictionary with existing values as the keys, and new values as the values

```
# use the map method to create a new column
df['State_Clean'] = df.State.map(state_mapping)
df
```

	Name	City	State	Income	Birthdate	State_Clean
4	Beverly Nixon	Long Island	NY	45000000.0	1962-03-05	NY
5	Veronica Comerford	Tuskegee	AL	50000.0	1993-09-23	AL
6	Ivan Layton	Albany	New York	105000.0	1996-03-12	NY

These states were mapped to these state abbreviations

This is similar to creating a **lookup table** in Excel and using VLOOKUP to search for a value in a column of the table and retrieve a corresponding value from another column

*Copyright Maven Analytics, LLC

PRO TIP: CLEANING TEXT

String methods are commonly used to **clean text** and standardize it for analysis

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
run_times
```

	Name	Run Time	Warm Up Time	Location
0	Alexis	9.2343	3.5	“school”
1	Alexis	10.3842	3.5	School
2	Alexis	8.1209	3 min	“the gym”
3	David	7.2123	2.2	“school”
4	David	6.8342	2	“gym”

There is a lot more you can do to clean text data, which will be covered in the **Natural Language Processing** course

```
# strip away the leading and trailing characters
run_times.Location.str.strip('“”')
```

```
0     school
1     School
2    the gym
3     school
4        gym
Name: Location, dtype: object
```

```
# make everything lowercase
run_times.Location.str.strip('“”').str.lower()
```

```
0     school
1     school
2    the gym
3     school
4        gym
Name: Location, dtype: object
```

```
# replace or remove characters
run_times.Location.str.strip('“”').str.lower().str.replace('the ', '')
```

```
0    school
1    school
2       gym
3    school
4       gym
Name: Location, dtype: object
```

*Copyright Maven Analytics, LLC

ASSIGNMENT: INCONSISTENT TEXT & TYPOS

NEW MESSAGE

June 14, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Inconsistent Text Check**

Hi again,

Thanks for your help with the missing data yesterday. I like how you decided to handle those missing values.

Can you check the same file for inconsistencies in the text and resolve any issues that you find?

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any inconsistent text and typos
2. Deal with the inconsistent text and typos

*Copyright Maven Analytics, LLC

DUPLICATE DATA

Duplicate data represents the presence of one or more redundant rows that contain the same information as another, and can therefore be removed

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

This is duplicate data on the same customer

These are the same values, but they're not considered duplicate data

*Copyright Maven Analytics, LLC

IDENTIFYING DUPLICATE DATA

Use **.duplicated()** to identify duplicate rows of data

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

```
df.duplicated()
```

```
0     False
1     False
2      True
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
dtype: bool
```

This returns True for every row that is a duplicate of a previous row

```
df.duplicated().sum()
```

```
1
```

*You can use **keep=False** to return all the duplicate rows*

```
df[df.duplicated(keep=False)]
```

	Name	City	State	Income	Age
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Joseph Martinez	Portland	OR	120000.0	62.0

*Copyright Maven Analytics, LLC

REMOVING DUPLICATE DATA

Use **.drop_duplicates()** to remove duplicate rows of data

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df.drop_duplicates()
```

You may need to reset the index

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
3	Wayne Nielson	Dahlgren	VA	75000.0	54.0
4	Beverly Nixon	Long Island	NY	45000000.0	61.0
5	Veronica Comerford	Tuskegee	AL	50000.0	29.0
6	Ivan Layton	Albany	NY	105000.0	27.0
7	Laura Hailey	Ephraim	UT	62000.0	29.0
8	John Depaul	Baton Rouge	LA	60000.0	55.0
9	James Weiss	Houston	TX	80000.0	NaN
10	Addie Stevenson	Jacksonville	FL	80000.0	NaN
11	Queen Watson	Los Angeles	CA	110000.0	54.0

*Copyright Maven Analytics, LLC

ASSIGNMENT: DUPLICATE DATA

NEW MESSAGE

June 15, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Duplicate Data Check**

Hi again,

I know the last task around finding inconsistent text was a bit tricky. This should be more straightforward!

Can you check the same file for duplicate data and resolve any issues?

Thanks!
Alan

Reply Forward

Key Objectives

1. Find any duplicate data
2. Deal with the duplicate data

*Copyright Maven Analytics, LLC

OUTLIERS

An **outlier** is a value in a data set that is much bigger or smaller than the others

- Data Cleaning Overview
- Data Types
- Data Issues
- Creating New Columns

```
df
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Wayne Nielson	Dahlgren	VA	75000.0	54.0
3	Beverly Nixon	Long Island	NY	45000000.0	61.0
4	Veronica Comerford	Tuskegee	AL	50000.0	29.0
5	Ivan Layton	Albany	NY	105000.0	27.0
6	Laura Hailey	Ephraim	UT	62000.0	29.0
7	John Depaul	Baton Rouge	LA	60000.0	55.0
8	James Weiss	Houston	TX	80000.0	NaN
9	Addie Stevenson	Jacksonville	FL	80000.0	NaN
10	Queen Watson	Los Angeles	CA	110000.0	54.0

*Average income (**including** outlier) = **$4.1M***

*Average income (**excluding** outlier) = **$82K***

If outliers are not identified and dealt with, they can have a **notable impact** on calculations and models

*Copyright Maven Analytics, LLC

IDENTIFYING OUTLIERS

Data Cleaning Overview

Data Types

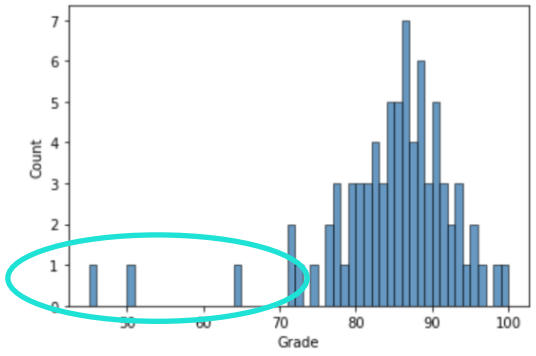
Data Issues

Creating New Columns

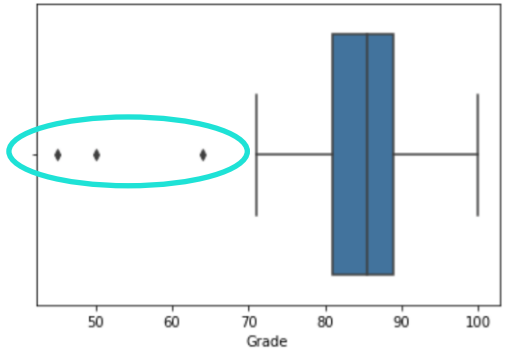
You can **identify outliers** in different ways using plots and statistics

EXAMPLE | *Identifying outliers in student grades from a college class*

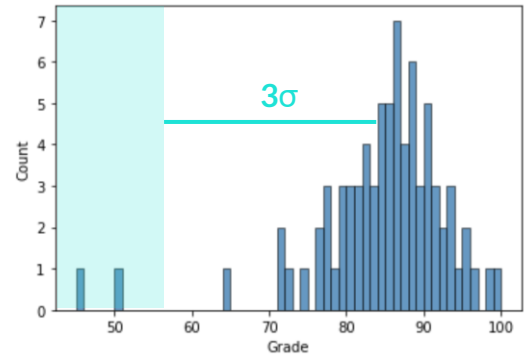
Histogram



Box Plot



Standard Deviation



It's also important to define what you'll consider an outlier in each scenario

Should we flag 3 or 2 outliers?

Use your domain expertise!

*Copyright Maven Analytics, LLC

HISTOGRAMS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Histograms are used to visualize the distribution *(or shape)* of a numerical column

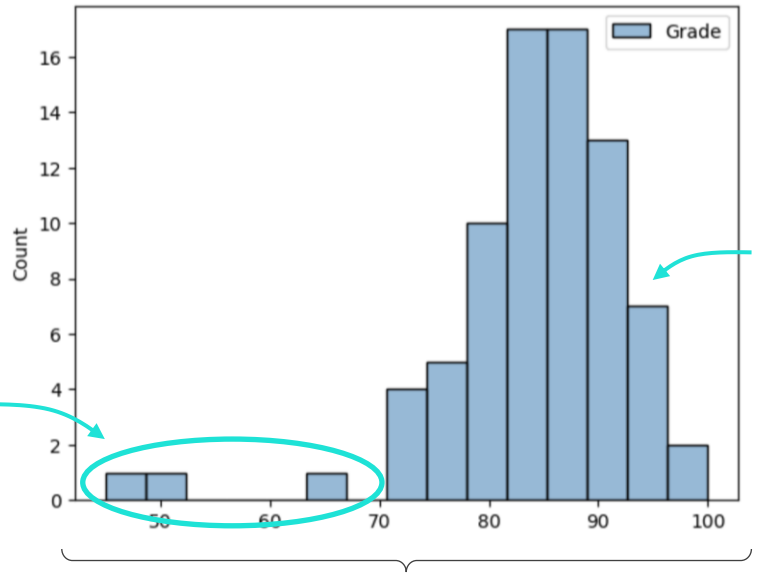
- They help identify outliers by showing which values fall outside of the normal range

```
df = pd.read_csv("student_grades.csv")
df
```

	Student	Grade
0	Emma	86
1	Olivia	86
2	Noah	86
3	Sophia	87
4	Liam	90
...	...	...
73	Zoey	91
74	Aaron	85
75	Charles	93
76	Connor	91
77	Riley	87

*You can create them with the **seaborn** library*

```
import seaborn as sns
sns.histplot(df);
```



These are the potential outliers

The height of each bar is how often the value occurs

This is the range of values in the data set

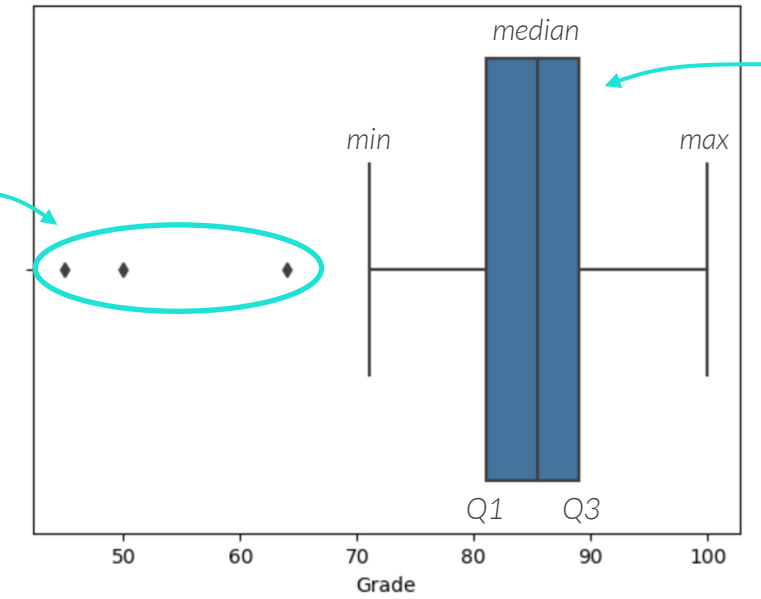
*Copyright Maven Analytics, LLC

BOXPLOTS

Boxplots are used to visualize the descriptive statistics of a numerical column

- They automatically plot outliers as dots outside of the min/max data range


```
sns.boxplot(x=df.Grade);
```



These are the outliers

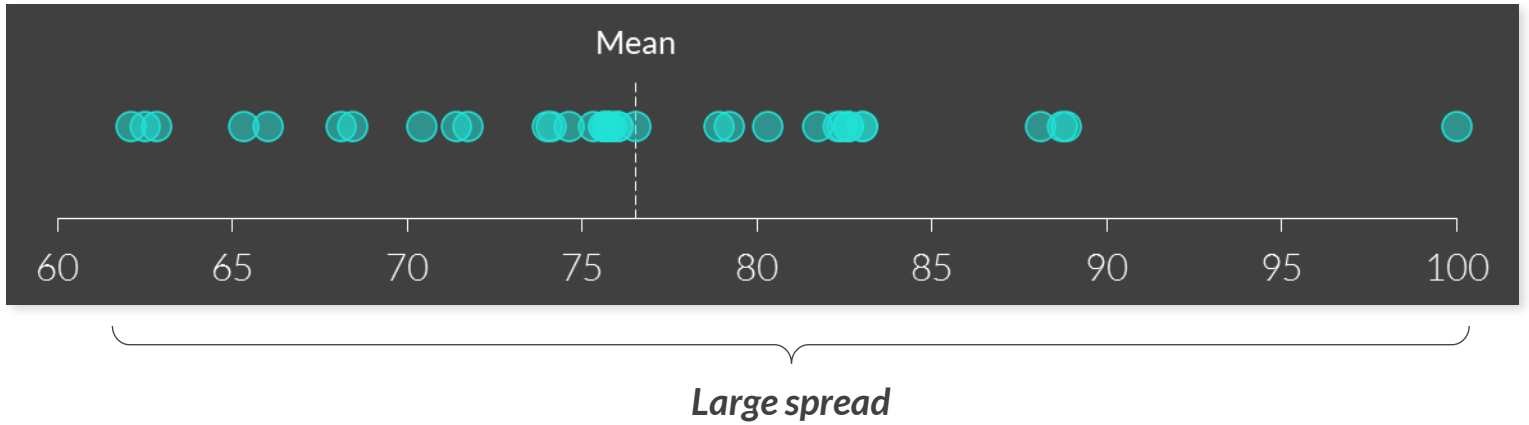
*The width of the "box" is the **interquartile range** (IQR), which is the middle 50% of the data*

*Any value farther away than 1.5*IQR from each side of the box is considered an outlier*

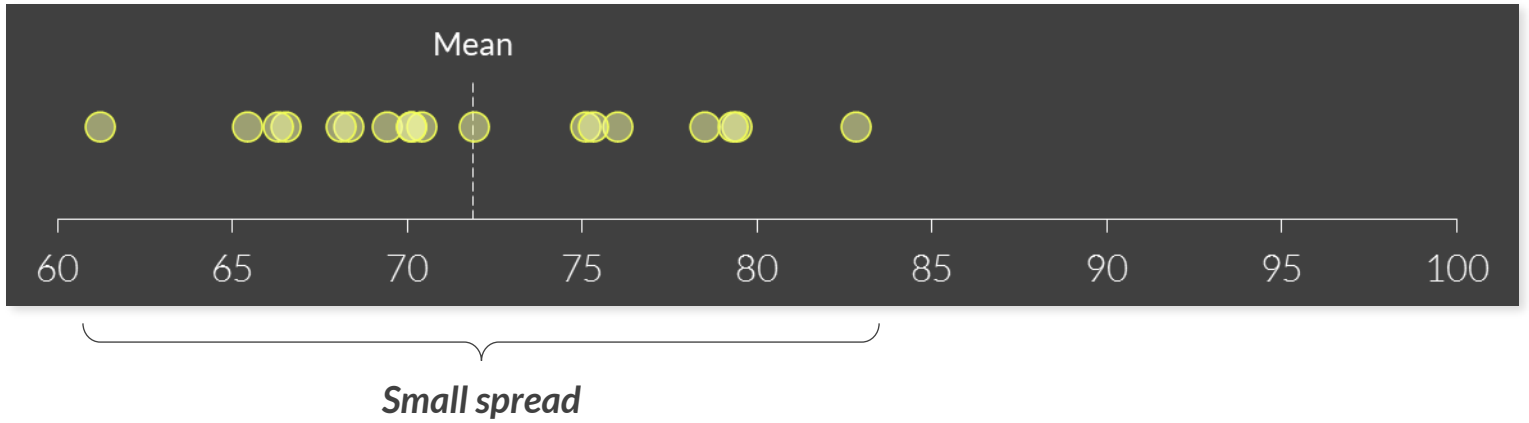
*Copyright Maven Analytics, LLC

STANDARD DEVIATION

The **standard deviation** is a measure of the spread of a data set from the mean



Standard Deviation = **8.2**



Standard Deviation = **5.8**

*Copyright Maven Analytics, LLC

STANDARD DEVIATION

The **standard deviation** is a measure of the spread of a data set from the mean

Values at least 3 standard deviations away from the mean are considered outliers

- This is meant for normally distributed, or bell shaped, data
- The threshold of 3 standard deviations can be changed to 2 or 4+ depending on the data

```
import numpy as np

mean = np.mean(df.Grade)
sd = np.std(df.Grade)

mean, sd
```

```
(84.08987341772152, 8.723725033779411)
```

```
[grade for grade in df.Grade if (grade < mean - 3*sd) or (grade > mean + 3*sd)]
```

```
[50, 45]
```

This returns a list of values, or outliers, at least 3 standard deviations away from the mean

*Copyright Maven Analytics, LLC

HANDLING OUTLIERS

Like with missing data, there are multiple ways to **handle outliers**:

- Keep outliers
- Remove an entire row or column with outliers
- Impute outliers with NaN or a substitute like the average, mode, max, etc.
- Resolve outliers based on your domain expertise

```
df.head()
```

	Name	City	State	Income	Age
0	Susan Rodriguez	Maplewood	NJ	45000.0	52.0
1	Joseph Martinez	Portland	OR	120000.0	62.0
2	Wayne Nielson	Dahlgren	VA	75000.0	54.0
3	Beverly Nixon	Long Island	NY	45000000.0	61.0
4	Veronica Comerford	Tuskegee	AL	50000.0	29.0

How would you handle this?

*Copyright Maven Analytics, LLC

ASSIGNMENT: OUTLIERS & REVIEW CLEANED DATA

NEW MESSAGE

June 16, 2023

From: **Alan Alarm** *(Researcher)*

Subject: **Outlier Check**

Hi again,

I have one last request for you and then I think our data is clean enough for now.

Can you check the file for outliers and resolve any issues?

Thanks for all your help this week – you rock!

Best,
Alan

Reply Forward

Key Objectives

1. Find any outliers
2. Deal with the outliers
3. Quickly explore the updated DataFrame. How do things look now after handling the data issues compared to the original DataFrame?

*Copyright Maven Analytics, LLC

CREATING NEW COLUMNS

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

After cleaning data types & issues, you may still not have the exact data that you need, so you can **create new columns** from existing data to aid your analysis

- Numeric columns – calculating percentages, applying conditional calculations, etc.
- Datetime columns – extracting datetime components, applying datetime calculations, etc.
- Text columns – extracting text, splitting into multiple columns, finding patterns, etc.

Name	Cost	Date	Notes
Alexis	$0	4/15/23	Coach: great job!
Alexis	$0	4/22/23	Coach: keep it up
Alexis	$25	5/10/23	PT: add strength training
David	$20	5/1/23	Trainer: longer warm up
David	$20	5/10/23	Trainer: pace yourself

Add 8% tax *Extract the month* *Split into two columns*

Name	Cost + Tax	Month	Person	Note
Alexis	$0	April	Coach	great job!
Alexis	$0	April	Coach	keep it up
Alexis	$27.00	May	PT	add strength training
David	$21.60	May	Trainer	longer warm up
David	$21.60	May	Trainer	pace yourself

Data is ready for further analysis!

*Copyright Maven Analytics, LLC

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

	Category	Item	Price
0	Fruit	Apple	1.29
1	Fruit	Banana	0.40
2	Fruit	Grapes	3.99
3	Vegetable	Carrots	1.50
4	Vegetable	Celery	1.99

*This will be the **numerator***

*Copyright Maven Analytics, LLC

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

```
# calculate the total amount spent
shopping_list['Total Spend'] = shopping_list['Price'].sum()
shopping_list
```

	Category	Item	Price	Total Spend
0	Fruit	Apple	1.29	9.17
1	Fruit	Banana	0.40	9.17
2	Fruit	Grapes	3.99	9.17
3	Vegetable	Carrots	1.50	9.17
4	Vegetable	Celery	1.99	9.17

*This will be the **denominator***

***sum** = 9.17*

*Copyright Maven Analytics, LLC

CALCULATING PERCENTAGES

To **calculate a percentage**, you can set up two columns with the numerator and denominator values and then divide them *(you can also multiply by 100 if desired)*

EXAMPLE *Finding the percentage of total spend for each item*

```
shopping_list
```

```
# calculate the total amount spent
shopping_list['Total Spend'] = shopping_list['Price'].sum()
shopping_list
```

```
# calculate the percent spent
shopping_list['Percent Spend'] = shopping_list['Price'] / total_spend * 100
shopping_list
```

	Category	Item	Price	Total Spend	Percent Spend
0	Fruit	Apple	1.29	9.17	14.067612
1	Fruit	Banana	0.40	9.17	4.362050
2	Fruit	Grapes	3.99	9.17	43.511450
3	Vegetable	Carrots	1.50	9.17	16.357688
4	Vegetable	Celery	1.99	9.17	21.701200

*These add up to **100%***

*Copyright Maven Analytics, LLC

CALCULATING BASED ON A CONDITION

Data Cleaning Overview

Data Types

Data Issues

Creating New Columns

Use **np.where()** to create a new column based on a logical condition

```
run_times
```

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	school	2023-04-15	2023-06-01	False	0.0
1	Alexis	10.3842	3.5	school	2023-04-22	2023-06-01	True	0.0
2	Alexis	8.1209	3	gym	2023-05-10	2023-06-01	False	2.5
3	David	7.2123	2.2	school	2023-05-01	2023-06-15	False	0.0
4	David	6.8342	2	gym	2023-05-10	2023-06-15	False	2.5

```
# update all gym fees to include tax
run_times['Fee with Tax'] = np.where(run_times.Location == 'gym', run_times.Fee * 1.08, run_times.Fee)
run_times
```

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee	Fee with Tax
0	Alexis	9.2343	3.5	school	2023-04-15	2023-06-01	False	0.0	0.0
1	Alexis	10.3842	3.5	school	2023-04-22	2023-06-01	True	0.0	0.0
2	Alexis	8.1209	3	gym	2023-05-10	2023-06-01	False	2.5	2.7
3	David	7.2123	2.2	school	2023-05-01	2023-06-15	False	0.0	0.0
4	David	6.8342	2	gym	2023-05-10	2023-06-15	False	2.5	2.7

If Location is equal to 'gym', then increase the Fee by 8% in the Fee with Tax column

Otherwise, set it to the existing Fee

*Copyright Maven Analytics, LLC

ASSIGNMENT: CREATE COLUMNS FROM NUMERIC DATA

NEW MESSAGE

July 5, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Pen Sales Data**

Hello,

I've attached the data on our June pen sales.

Can you create two new columns?

- A "Total Spend" column that includes both the pen cost and shipping cost for each sale
- A "Free Shipping" column that says yes if the sale included free shipping, and no otherwise

Thanks!
Peter

Pen Sales Data.xlsx

Reply Forward

Key Objectives

1. Read data into Python
2. Check the data type of each column
3. Create a numeric column using arithmetic
4. Create a numeric column using conditional logic

*Copyright Maven Analytics, LLC

EXTRACTING DATETIME COMPONENTS

Use **dt.component** to extract a component from a datetime value *(day, month, etc.)*


```
run_times
```

	Run Date	Race Date
0	2023-04-15 12:00:00	2023-06-01
1	2023-04-22 12:30:00	2023-06-01
2	2023-05-10 15:00:00	2023-06-01
3	2023-05-01 15:15:00	2023-06-15
4	2023-05-10 16:30:00	2023-06-15

```
# extract the day from the date
run_times['Run Date'].dt.day
```

```
0    15
1    22
2    10
3     1
4    10
Name: Run Date, dtype: int64
```

```
# extract the day of the week
run_times['Run Date'].dt.dayofweek
```

```
0    5
1    5
2    2
3    0
4    2
Name: Run Date, dtype: int64
```

```
# extract the time
run_times['Run Date'].dt.time
```

```
0    12:00:00
1    12:30:00
2    15:00:00
3    15:15:00
4    16:30:00
Name: Run Date, dtype: object
```

Component	Output
dt.date	Date *(without time component)*
dt.year	Year
dt.month	Numeric month *(1-12)*
dt.day	Day of the month
dt.dayofweek	Numeric weekday *(Mon=0, Sun=6)*
dt.time	Time *(without date component)*
dt.hour	Hour *(0-23)*
dt.minute	Minute *(0-59)*
dt.second	Second *(0-59)*

*Copyright Maven Analytics, LLC

DATETIME CALCULATIONS

Datetime calculations between columns can be done using basic arithmetic

- Use **pd.to_timedelta()** to add or subtract a particular timeframe

```
# calculate the difference between two dates
run_times['Race Date'] - run_times['Run Date']
```

```
0   46 days 12:00:00
1   39 days 11:30:00
2   21 days 09:00:00
3   44 days 08:45:00
4   35 days 07:30:00
dtype: timedelta64[ns]
```

Note that the data type changed

```
# add two weeks to the race date
run_times['Race Date'] + pd.to_timedelta(2, unit='W')
```

```
0   2023-06-15
1   2023-06-15
2   2023-06-15
3   2023-06-29
4   2023-06-29
Name: Race Date, dtype: datetime64[ns]
```

Time delta units:

- *D = day*
- *W = week*
- *H = hour*
- *T = minute*
- *S = second*

*Copyright Maven Analytics, LLC

ASSIGNMENT: CREATE COLUMNS FROM DATETIME DATA

NEW MESSAGE

July 12, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Delivery Time of Pens?**

Hello again,

Using the data I sent over last week, can you calculate the number of days between the purchase and delivery date for each sale and save it as a new column called “Delivery Time”?

What were the average days from purchase to delivery?

Thanks!
Peter

Reply Forward

Key Objectives

1. Calculate the difference between two datetime columns and save it as a new column
2. Take the average of the new column

*Copyright Maven Analytics, LLC

EXTRACTING TEXT

You can use **.str[start:end]** to extract characters from a text field

- Note that the position of each character in a string is 0-indexed, and the "end" is non-inclusive

```
run_notes
```

	notes
0	Day 1 - Starting off: Congrats on starting you...
1	Day 2 - Progressing: On your second day, you m...
2	Day 3 - Finding a rhythm: By day 3, you may ha...
3	Day 4 - Pushing yourself: On day 4, you may ha...
4	Day 5 - Consistency: On your final day, you sh...

```
# first 6 characters
run_notes.notes.str[:6]
```

```
0    Day 1
1    Day 2
2    Day 3
3    Day 4
4    Day 5
Name: notes, dtype: object
```

The blank starts from 0 by default

```
# last 11 characters
run_notes.notes.str[-11:]
```

```
0     endurance.
1    Keep it up!
2     your runs.
3     endurance.
4    Keep it up!
Name: notes, dtype: object
```

The negative grabs the "start" from the end of the list, and the blank "end" goes to the end of the text string

*Copyright Maven Analytics, LLC

SPLITTING INTO MULTIPLE COLUMNS

Use **str.split()** to split a column by a delimiter into multiple columns


```
run_notes
```

	notes
0	Day 1 - Starting off: Congrats on starting you...
1	Day 2 - Progressing: On your second day, you m...
2	Day 3 - Finding a rhythm: By day 3, you may ha...
3	Day 4 - Pushing yourself: On day 4, you may ha...
4	Day 5 - Consistency: On your final day, you sh...

```
# split on the - character
two_fields = run_notes.notes.str.split('-')
two_fields
```

```
0    [Day 1 ,  Starting off: Congrats on starting y...
1    [Day 2 ,  Progressing: On your second day, you...
2    [Day 3 ,  Finding a rhythm: By day 3, you may ...
3    [Day 4 ,  Pushing yourself: On day 4, you may ...
4    [Day 5 ,  Consistency: On your final day, you ...
Name: notes, dtype: object
```

Splitting text returns a list

```
# turn list series into dataframe
pd.DataFrame(two_fields.to_list(),
             columns=['Day', 'Notes'])
```

	Day	Notes
0	Day 1	Starting off: Congrats on starting your runni...
1	Day 2	Progressing: On your second day, you may have...
2	Day 3	Finding a rhythm: By day 3, you may have foun...
3	Day 4	Pushing yourself: On day 4, you may have felt...
4	Day 5	Consistency: On your final day, you should fe...

This is now a DataFrame with two columns

*Copyright Maven Analytics, LLC

FINDING PATTERNS

Use **str.contains()** to find words or patterns within a text field


```
# create a new column that says whether or not
# the note contains the term 'final'
run_notes['Contains final'] = run_notes.notes.str.contains('final')
run_notes
```

	notes	Contains final
0	Day 1 - Starting off: Congrats on starting you...	False
1	Day 2 - Progressing: On your second day, you m...	False
2	Day 3 - Finding a rhythm: By day 3, you may ha...	False
3	Day 4 - Pushing yourself: On day 4, you may ha...	False
4	Day 5 - Consistency: On your final day, you sh...	True

```
# contains great or congrats
run_notes.notes.str.contains('great|congrats', regex=True)
```

```
0    False
1    False
2     True
3     True
4    False
Name: notes, dtype: bool
```

Regex stands for **regular expression**, which is a way of finding patterns within text (more on this topic will be covered in the Natural Language Processing course)

*Copyright Maven Analytics, LLC

ASSIGNMENT: CREATE COLUMNS FROM TEXT DATA

NEW MESSAGE

July 19, 2023

From: **Peter Penn** *(Sales Rep)*

Subject: **Pen Reviews**

Hello again,

You may have noticed that the data I sent over a few weeks ago also includes pen reviews.

Can you split the reviews on the "|" character to create two new columns: "User Name" and "Review Text"?

Can you also create a "Leak or Spill" column that flags the reviews that mention either "leak" or "spill"?

Thanks!
Peter

Reply Forward

Key Objectives

1. Split one column into multiple columns
2. Create a Boolean column (True/False) to show whether a text field contains particular words

*Copyright Maven Analytics, LLC

KEY TAKEAWAYS

Always check that the **data type** for each column matches their intended use

- *Sometimes all columns are read in as objects into a DataFrame, so they may need to be converted into numeric or datetime columns to be able to do any appropriate calculations down the line*

It's important to resolve any **messy data issues** that could impact your analysis

- *When you're looking over a data set for the first time, check for missing data, inconsistent text & typos, duplicate data, and outliers, then be thoughtful and deliberate about how you handle each*

You can **create new columns** based on existing columns in your DataFrame

- *Depending on the data type, you can apply numeric, datetime, or string calculations on existing columns to create new columns that can be useful for your analysis*

You goal should be to make the data **clean enough** for your analysis

- *It's difficult to identify all the issues in your data in order to make it 100% clean (especially if working with text data), so spending extra time trying to do so can be counterproductive – remember the MVP approach!*

*Copyright Maven Analytics, LLC