18.2.4

Data Selection

It's not every day that you and Martha have a chance to convince an accounting firm to invest in cryptocurrency! So, you want to make sure you know how to select the data that will best help the model determine patterns or grouping.

To help us select the data, let's return to some of the questions on our list.

What data is available?

First, account for the data you have. After all, you can't extract knowledge without data. We can use the `columns` method and output the columns, as shown below:

```
# Columns
df_shopping.columns
```

Looking at the columns, we see there is data for CustomerID, Age, Annual Income, and Spending Score:

```
Index(['Customer ID', 'Card Member', 'Age', 'Annual Income',
       'Spending Score (1-100)'],
      dtype='object')
```

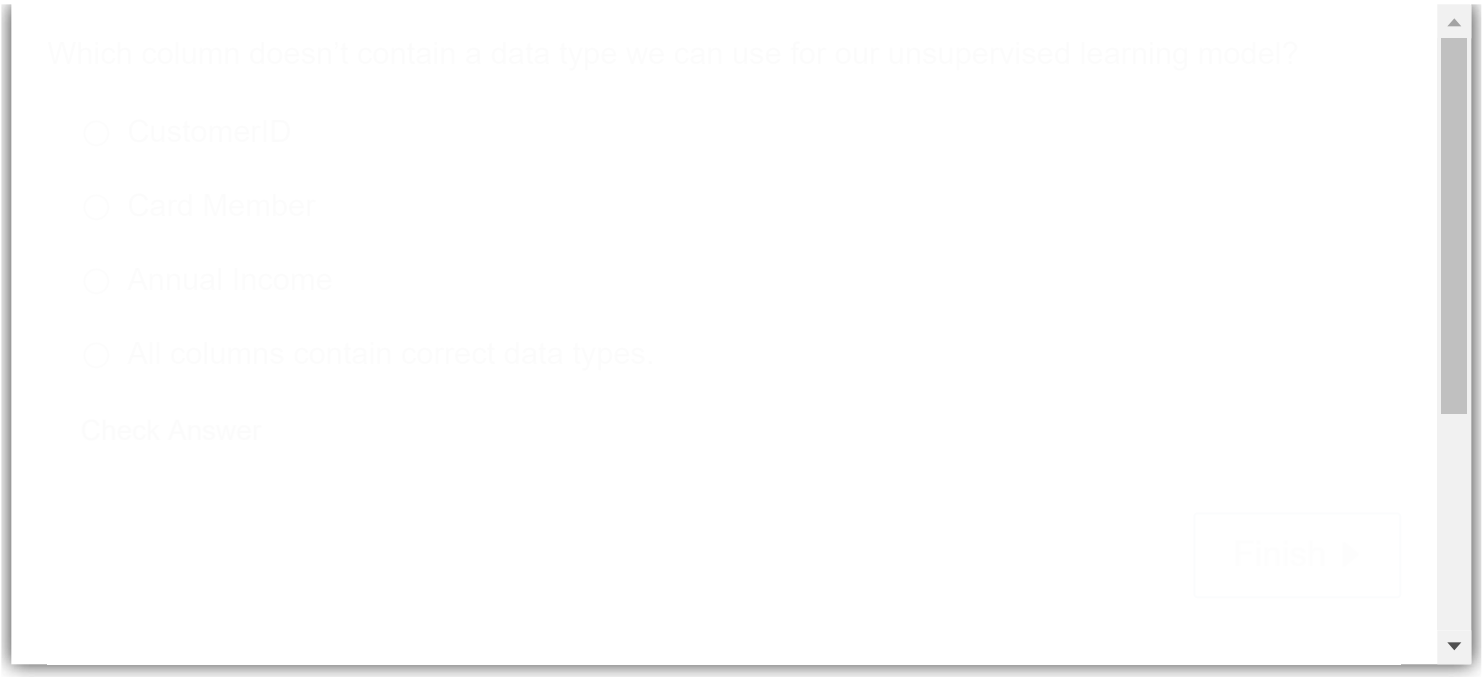
Now that we know what data we have, we can start thinking about possible analysis. For example, data points for features like Age and Annual Income might appear in our end result as groupings or clusters. However, there are no data points for items purchased, so our algorithms cannot discover related patterns.

What type of data is available?

Using the `dtypes` method, confirm the data type, which also will alert us if anything should be changed in the next step (e.g., converting text to numerical data). All the columns we plan to use in our model must contain a numerical data type:

```
# List dataframe data types
df_shopping.dtypes
```

```
CustomerID                int64
Card Member              object
Age                     float64
Annual Income             Int64
Spending Score (1-100)  float64
dtype: object
```



What data is missing?

Next, let's see if any data is missing. Unsupervised learning models can't handle missing data. If you try to run a model on a dataset with missing data, you'll get an error such as the one below:

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

If you initially had hoped to produce an outcome using a type of data, but it turned out more than 80% of those rows are empty, then the results won't be very accurate!

For example, return to our Age and Income groups: If it turns out there are 1,200 rows without any Age data points, then we clearly can't use that column in our model. There is no set cutoff for missing data—that decision is left up to you, the analyst, and must be made based on your understanding of the business needs.

NOTE

Handling missing data is a complex topic that is out of scope for this unit. However, if you're interested, read this **article (https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4)** on the possible approaches to handling missing data.

Pandas has the `isnull()` method to check for missing values. We'll loop through each column, check if there are null values, sum them up, and print out a readable total:

```
# Find null values
for column in df_shopping.columns:
    print(f"Column {column} has {df_shopping[column].isnull().sum()}
null values")
```

```
Column CustomerID had 0 null values
Column Card Member has 2 null values
Column Age has 2 null values
Column Annual Income has 0 null values
Column Spending Score (1-100) has 1 null values
```

There will be a few rows with missing values that we'll need to handle. The judgement call will be to either remove these rows or decide that the dataset is not suitable for our model. In this case, we'll proceed with handling these values because they are a small percentage of the overall data.

IMPORTANT

When deciding to proceed, the percentage of data missing isn't always the only determining factor. See the Note callout above for a resource on handling missing data.

What data can be removed?

You have begun to explore the data and have taken a look at null values. Next, determine if the data can be removed. Consider: Are there string columns that we can't use? Are there columns with excessive null data points? Was our decision to handle missing values to just remove them?

In our example, there are no string type columns, and we made the decision that only a few rows have null data points, but not enough to remove a whole column's worth.

Rows of data with null values can be removed with the `dropna()` method, as shown below:

```
# Drop null rows
df_shopping = df_shopping.dropna()
```

Duplicates can also be removed.

Why should we remove duplicate rows from our dataset? Select all applicable reasons.

- [ ] Duplicates add too much data to be processed.
- [ ] Duplicates aren't telling us anything new.
- [ ] Duplicates aren't really part of the dataset.
- [ ] Duplicates could skew our results.

Check Answer

Finish ▶

Use the `duplicated().sum()` method to check for duplicates, as shown below:

```
# Find duplicate entries
print(f"Duplicate entries: {df_shopping.duplicated().sum()}")
```

```
Duplicate entries: 0
```

Looks good with no duplicates!

We also can remove data that doesn't tell us anything interesting. Knowing this, is there anything we can remove from this DataFrame?

- ○ Card Member
- ○ CustomerID
- ○ Annual Income
- ○ None of the above

Check Answer

Finish ▶

To remove the column, just enter the code below:

```
# Remove the CustomerID Column
df_shopping.drop(columns=["CustomerID"], inplace=True)
df_shopping.head()
```

	Card Member	Age	Annual Income	Spending Score (1-100)
0	Yes	19.0	15000	39.0
1	Yes	21.0	15000	81.0
2	No	20.0	16000	6.0
3	No	23.0	16000	77.0
4	No	31.0	17000	40.0