

Exemplar__Validate and clean your data

April 2, 2025

1 Exemplar: Validate and clean your data

1.1 Introduction

In this activity, you will use input validation and label encoding to prepare a dataset for analysis. These are fundamental techniques used in all types of data analysis, from simple linear regression to complex neural networks.

In this activity, you are a data professional at an investment firm that is attempting to invest in private companies with a valuation of at least \$1 billion. These are often known as “unicorns.” Your client wants to develop a better understanding of unicorns, with the hope they can be early investors in future highly successful companies. They are particularly interested in the investment strategies of the three top unicorn investors: Sequoia Capital, Tiger Global Management, and Accel.

1.2 Step 1: Imports

Import relevant Python libraries and packages: `numpy`, `pandas`, `seaborn`, and `pyplot` from `matplotlib`.

```
[1]: # Import libraries and packages.  
  
### YOUR CODE HERE ###  
  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt
```

1.2.1 Load the dataset

The data contains details about unicorn companies, such as when they were founded, when they achieved unicorn status, and their current valuation. Load the dataset `Modified_Unicorn_Companies.csv` as `companies` and display the first five rows. The variables in the dataset have been adjusted to suit the objectives of this lab, so they may be different from similar data used in prior labs.

```
[2]: # Run this cell so pandas displays all columns
pd.set_option('display.max_columns', None)
```

```
[3]: companies = pd.read_csv('Modified_Unicorn_Companies.csv')

# Display the first five rows.

### YOUR CODE HERE ###
companies.head()
```

```
[3]:      Company  Valuation  Date Joined      Industry \
0  Bytedance      180  2017-04-07  Artificial intelligence
1   SpaceX      100  2012-12-01                Other
2   SHEIN      100  2018-07-03  E-commerce & direct-to-consumer
3   Stripe       95  2014-01-23                FinTech
4   Klarna       46  2011-12-12                Fintech
```

```
      City Country/Region  Continent  Year Founded  Funding \
0   Beijing      China      Asia      2012      $8B
1 Hawthorne  United States  North America      2002      $7B
2  Shenzhen      China      Asia      2008      $2B
3 San Francisco  United States  North America      2010      $2B
4  Stockholm      Sweden      Europe      2005      $4B
```

```
      Select Investors
0  Sequoia Capital China, SIG Asia Investments, S...
1  Founders Fund, Draper Fisher Jurvetson, Rothen...
2  Tiger Global Management, Sequoia Capital China...
3      Khosla Ventures, LowercaseCapital, capitalG
4  Institutional Venture Partners, Sequoia Capita...
```

1.3 Step 2: Data cleaning

Begin by displaying the data types of the columns in `companies`.

```
[4]: # Display the data types of the columns.

### YOUR CODE HERE ###
companies.dtypes
```

```
[4]: Company      object
Valuation      int64
Date Joined     object
Industry        object
City            object
Country/Region  object
```

```
Continent      object
Year Founded   int64
Funding        object
Select Investors object
dtype: object
```

Hint 1

Review what you have learned about exploratory data analysis in Python.

Hint 2

There is a `pandas` `DataFrame` attribute that displays the data types of the columns in the specified `DataFrame`.

Hint 3

The `pandas` `DataFrame` `dtypes` attribute will be helpful.

1.3.1 Modify the data types

Notice that the data type of the `Date Joined` column is an `object`—in this case, a string. Convert this column to `datetime` to make it more usable.

```
[5]: # Apply necessary datatype conversions.

    ### YOUR CODE HERE ###
    companies['Date Joined'] = pd.to_datetime(companies['Date Joined'])
```

1.3.2 Create a new column

Add a column called `Years To Unicorn`, which is the number of years between when the company was founded and when it became a unicorn.

```
[6]: # Create the column Years To Unicorn.

    ### YOUR CODE HERE ###
    companies['Years To Unicorn'] = companies['Date Joined'].dt.year -
    ↪companies['Year Founded']
```

Hint 1

Extract just the year from the `Date Joined` column.

Hint 2

Use `dt.year` to access the year of a `datetime` object.

Hint 3

Subtract the `Year Founded` from the `Date Joined` year, and save it to a new column called `Years To Unicorn`.

Ensure you're properly extracting just the year (as an integer) from `Date Joined`.

QUESTION: Why might your client be interested in how quickly a company achieved unicorn status?

Learning how quickly a company achieves unicorn status may reveal certain trends or commonalities. Your client could leverage this information to find future companies to invest in.

1.3.3 Input validation

The data has some issues with bad data, duplicate rows, and inconsistent `Industry` labels.

Follow the steps below to identify and correct each of these issues.

Correcting bad data Get descriptive statistics for the `Years To Unicorn` column.

```
[7]: companies['Years To Unicorn'].describe()
```

```
[7]: count    1074.000000
     mean       7.013035
     std       5.331842
     min      -3.000000
     25%       4.000000
     50%       6.000000
     75%       9.000000
     max      98.000000
     Name: Years To Unicorn, dtype: float64
```

Hint 1

Use the `describe()` method on the series. Considering the results, does anything seem problematic?

Hint 2

A company cannot reach unicorn status before it is founded. In other words, `Years to Unicorn` cannot be less than 0.

Hint 3

Using the `describe()` method on the `Years To Unicorn` series shows that the minimum value is -3. Since there cannot be negative time, this value and possibly others are problematic.

Isolate all rows where the `Years To Unicorn` column contains a negative value.

```
[8]: # Isolate any rows where `Years To Unicorn` is negative

     ### YOUR CODE HERE ###
     companies[companies['Years To Unicorn'] < 0]
```

```
[8]:      Company  Valuation Date Joined      Industry      City \
     527  InVision           2  2017-11-01  Internet software & services  New York
```

	Country/Region	Continent	Year Founded	Funding	\
527	United States	North America	2020	\$349M	

	Select Investors	Years To Unicorn
527	FirstMark Capital, Tiger Global Management, IC...	-3

Question: How many rows have negative values in the Years To Unicorn column, and what companies are they for?

- There is a single row that has a negative value in the Years To Unicorn column. The company represented in this row is InVision.

An internet search reveals that InVision was founded in 2011. Replace the value at Year Founded with 2011 for InVision's row.

```
[9]: # Replace InVision's `Year Founded` value with 2011

### YOUR CODE HERE ###
companies.loc[companies['Company']=='InVision', 'Year Founded'] = 2011

# Verify the change was made properly

### YOUR CODE HERE ###
companies[companies['Company']=='InVision']
```

```
[9]:      Company  Valuation Date Joined      Industry      City \
527  InVision          2   2017-11-01  Internet software & services  New York
```

	Country/Region	Continent	Year Founded	Funding	\
527	United States	North America	2011	\$349M	

	Select Investors	Years To Unicorn
527	FirstMark Capital, Tiger Global Management, IC...	-3

Hint 1

To overwrite data in a dataframe in a situation like this, you should use `loc[]` or `iloc[]` selection statements. Otherwise, you might overwrite to a view of the dataframe, which means that you're not overwriting the data in the dataframe itself, and the change will not take permanent effect.

Hint 2

The following code will **not** work:

```
companies[companies['Company']=='InVision']['Year Founded'] = 2011
```

You must use either `loc[]` or `iloc[]`.

Now, recalculate all the values in the Years To Unicorn column to remove the negative value for InVision. Verify that there are no more negative values afterwards.

```
[10]: # Recalculate all values in the `Years To Unicorn` column

### YOUR CODE HERE ###
companies['Years To Unicorn'] = companies['Date Joined'].dt.year - \
    companies['Year Founded']

# Verify that there are no more negative values in the column

### YOUR CODE HERE ###
companies['Years To Unicorn'].describe()
```

```
[10]: count    1074.000000
      mean      7.021415
      std       5.323155
      min       0.000000
      25%       4.000000
      50%       6.000000
      75%       9.000000
      max      98.000000
      Name: Years To Unicorn, dtype: float64
```

Issues with Industry labels The company provided you with the following list of industry labels to identify in the data for Industry.

Note: Any labels in the Industry column that are not in `industry_list` are misspellings.

```
[11]: # List provided by the company of the expected industry labels in the data
industry_list = ['Artificial intelligence', 'Other', 'E-commerce & \
    direct-to-consumer', 'Fintech', \
        'Internet software & services', 'Supply chain, logistics, & delivery', \
    'Consumer & retail', \
        'Data management & analytics', 'Edtech', 'Health', 'Hardware', 'Auto & \
    transportation', \
        'Travel', 'Cybersecurity', 'Mobile & telecommunications']
```

First, check if there are values in the Industry column that are not in `industry_list`. If so, what are they?

```
[12]: # Check which values are in `Industry` but not in `industry_list`

### YOUR CODE HERE ###
set(companies['Industry']) - set(industry_list)
```

```
[12]: {'Artificial Intelligence', 'Data management and analytics', 'FinTech'}
```

HINT 1

There are many ways to do this. One approach is to consider what data type reduces iterable sequences to their unique elements and allows you to compare membership.

HINT 2

A set is a data type that consists of unique elements and supports membership testing with other sets.

HINT 3

Set A – Set B will result in all the elements that are in Set A but not in Set B. Convert `industry_list` to a set and subtract it from the set of the values in the `Industry` series.

Question: Which values currently exist in the `Industry` column that are not in `industry_list`?

- 'Artificial Intelligence', 'Data management and analytics', and 'FinTech' are misspellings that are currently in the `Industry` column.

Now, correct the bad entries in the `Industry` column by replacing them with an approved string from `industry_list`. To do this, use the `replace()` Series method on the `Industry` series. When you pass a dictionary to the method, it will replace the data in the series where that data matches the dictionary's keys. The values that get imputed are the values of the dictionary. If a value is not specified in the dictionary, the series' original value is retained.

Example:

```
[IN]: column_a = pd.Series(['A', 'B', 'C', 'D'])
      column_a
```

```
[OUT]: 0    A
       1    B
       2    C
       3    D
      dtype: object
```

```
[IN]: replacement_dict = {'A':'z', 'B':'y', 'C':'x'}
      column_a = column_a.replace(replacement_dict)
      column_a
```

```
[OUT]: 0    z
       1    y
       2    x
       3    D
      dtype: object
```

1. Create a dictionary called `replacement_dict` whose keys are the incorrect spellings in the `Industry` series and whose values are the correct spelling, as indicated in `industry_list`.
2. Call the `replace()` method on the `Industry` series and pass to it `replacement_dict` as its argument. Reassign the result back to the `Industry` column.
3. Verify that there are no longer any elements in `Industry` that are not in `industry_list`.

```
[13]: # 1. Create `replacement_dict`

### YOUR CODE HERE ###
replacement_dict = {'Artificial Intelligence': 'Artificial intelligence',
                    'Data management and analytics': 'Data management & analytics',
                    'FinTech': 'Fintech'
                    }

# 2. Replace the incorrect values in the `Industry` column

### YOUR CODE HERE ###
companies['Industry'] = companies['Industry'].replace(replacement_dict)

# 3. Verify that there are no longer any elements in `Industry` that are not in
    ↳ `industry_list`

### YOUR CODE HERE ###
set(companies['Industry']) - set(industry_list)
```

[13]: set()

Hint 1

Refer to the example provided for how to use the `replace()` Series method.

Hint 2

When you define the `replacement_dict` dictionary, the misspellings should be the keys and the correct spellings should be the values.

Hint 3

When you call `replace()` on the `Industry` series and pass to it the `replacement_dict` dictionary as an argument, you must reassign the result back to `companies['Industry']` for the change to take effect.

Handling duplicate rows The business mentioned that no company should appear in the data more than once.

Verify that this is indeed the case, and if not, clean the data so each company appears only once.

Begin by checking which, if any, companies are duplicated. Filter the data to return all occurrences of those duplicated companies.

```
[14]: # Isolate rows of all companies that have duplicates

### YOUR CODE HERE ###
companies[companies.duplicated(subset=['Company'], keep=False)]
```



```
[14]:
```

	Company	Valuation	Date Joined	Industry	City \
385	BrewDog	2	2017-04-10	Consumer & retail	Aberdeen
386	BrewDog	2	2017-04-10	Consumer & retail	Aberdeen
510	ZocDoc	2	2015-08-20	Health	New York
511	ZocDoc	2	2015-08-20	Health	NaN
1031	SoundHound	1	2018-05-03	Artificial intelligence	Santa Clara
1032	SoundHound	1	2018-05-03	Other	Santa Clara

	Country/Region	Continent	Year Founded	Funding \
385	United Kingdom	Europe	2007	\$233M
386	United Kingdom	Europe	2007	\$233M
510	United States	North America	2007	\$374M
511	United States	North America	2007	\$374M
1031	United States	North America	2005	\$215M
1032	United States	North America	2005	\$215M

	Select Investors	Years To Unicorn
385	TSG Consumer Partners, Crowdcube	10
386	TSG Consumer Partners	10
510	Founders Fund, Khosla Ventures, Goldman Sachs	8
511	Founders Fund	8
1031	Tencent Holdings, Walden Venture Capital, Glob...	13
1032	Tencent Holdings	13

Hint 1

Check for duplicated values specifically in the `Company` column, not entire rows that are duplicated.

Hint 2

The pandas `duplicated()` `DataFrame` method can indentify duplicated rows. Apply it to the `Company` column in `companies` to find which companies appear more than once.

Hint 3

- To specify that you want to check for duplicates only in the `Company` column, indicate this with the `subset` parameter.
- To return *all* occurrences of duplicates, set the `keep` parameter to `False`.

Question: Do these duplicated companies seem like legitimate data points, or are they problematic data?

- The duplicated companies are not legitimate because they are clearly not different companies with the same name. They are the same company represented twice with minor variation.

Keep the first occurrence of each duplicate company and drop the subsequent rows that are copies.

```
[15]: # Drop rows of duplicate companies after their first occurrence

### YOUR CODE HERE ###
companies = companies.drop_duplicates(subset=['Company'], keep='first')
```

Hint 1

Use the `drop_duplicates()` DataFrame method.

Hint 2

Make sure to subset `Company` and reassign the results back to the `companies` dataframe for the changes to take effect.

Question: Why is it important to perform input validation?

Input validation is an essential practice for ensuring data is complete, error-free, and high quality. A low-quality dataset may lend itself to an analysis that is incorrect or misleading.

Question: What steps did you take to perform input validation for this dataset?

The input validation steps for this lab included:

- Fixing incorrect values
- Correcting inconsistencies in the data
- Removing duplicate data

1.3.4 Convert numerical data to categorical data

Sometimes, you'll want to simplify a numeric column by converting it to a categorical column. To do this, one common approach is to break the range of possible values into a defined number of equally sized bins and assign each bin a name. In the next step, you'll practice this process.

Create a High Valuation column The data in the `Valuation` column represents how much money (in billions, USD) each company is valued at. Use the `Valuation` column to create a new column called `High Valuation`. For each company, the value in this column should be `low` if the company is in the bottom 50% of company valuations and `high` if the company is in the top 50%.

```
[16]: # Create new `High Valuation` column

### YOUR CODE HERE ###

# Use qcut to divide Valuation into 'high' and 'low' Valuation groups
companies['High Valuation'] = pd.qcut(companies['Valuation'], 2, labels = ['low', 'high'])
```

Hint 1

There are multiple ways to complete this task. Review what you've learned about organizing data into equal quantiles.

Hint 2

Consider using the pandas `qcut()` function.

Hint 3

Use `pandas qcut()` to divide the data into two equal-sized quantile buckets. Use the `labels` parameter to define the output labels. The values you give for `labels` will be the values that are inserted into the new column.

1.3.5 Convert categorical data to numerical data

Three common methods for changing categorical data to numerical are:

1. Label encoding: order matters (ordinal numeric labels)
2. Label encoding: order doesn't matter (nominal numeric labels)
3. Dummy encoding: order doesn't matter (creation of binary columns for each possible category contained in the variable)

The decision on which method to use depends on the context and must be made on a case-to-case basis. However, a distinction is typically made between categorical variables with equal weight given to all possible categories vs. variables with a hierarchical structure of importance to their possible categories.

For example, a variable called `subject` might have possible values of `history`, `mathematics`, `literature`. In this case, each subject might be **nominal**—given the same level of importance. However, you might have another variable called `class`, whose possible values are `freshman`, `sophomore`, `junior`, `senior`. In this case, the class variable is **ordinal**—its values have an ordered, hierarchical structure of importance.

Machine learning models typically need all data to be numeric, and they generally use ordinal label encoding (method 1) and dummy encoding (method 3).

In the next steps, you'll convert the following variables: `Continent`, `Country/Region`, and `Industry`, each using a different approach.

1.3.6 Convert Continent to numeric

For the purposes of this exercise, suppose that the investment group has specified that they want to give more weight to continents with fewer unicorn companies because they believe this could indicate unrealized market potential.

Question: Which type of variable would this make the `Continent` variable in terms of how it would be converted to a numeric data type?

- This would make `Continent` an ordinal variable, since more importance is placed on continents with fewer unicorn companies. There is a hierarchy of importance.

Rank the continents in descending order from the greatest number of unicorn companies to the least.

```
[17]: # Rank the continents by number of unicorn companies

      ### YOUR CODE HERE ###
      companies['Continent'].value_counts()
```

```
[17]: North America    586
      Asia            310
      Europe          143
      South America    21
      Oceania          8
      Africa           3
      Name: Continent, dtype: int64
```

Hint

Use the `value_counts()` method on the `Continent` series.

Now, create a new column called `Continent Number` that represents the `Continent` column converted to numeric in the order of their number of unicorn companies, where North America is encoded as 1, through Africa, encoded as 6.

```
[18]: # Create numeric `Continent Number` column

### YOUR CODE HERE ###
continent_dict = {'North America': 1,
                  'Asia': 2,
                  'Europe': 3,
                  'South America': 4,
                  'Oceania': 5,
                  'Africa': 6
                  }

companies['Continent Number'] = companies['Continent'].replace(continent_dict)
companies.head()
```

```
[18]:
```

	Company	Valuation	Date Joined	Industry	\
0	Bytedance	180	2017-04-07	Artificial intelligence	
1	SpaceX	100	2012-12-01	Other	
2	SHEIN	100	2018-07-03	E-commerce & direct-to-consumer	
3	Stripe	95	2014-01-23	Fintech	
4	Klarna	46	2011-12-12	Fintech	

	City	Country/Region	Continent	Year Founded	Funding	\
0	Beijing	China	Asia	2012	\$8B	
1	Hawthorne	United States	North America	2002	\$7B	
2	Shenzhen	China	Asia	2008	\$2B	
3	San Francisco	United States	North America	2010	\$2B	
4	Stockholm	Sweden	Europe	2005	\$4B	

	Select Investors	Years To Unicorn	\
0	Sequoia Capital China, SIG Asia Investments, S...	5	
1	Founders Fund, Draper Fisher Jurvetson, Rothen...	10	
2	Tiger Global Management, Sequoia Capital China...	10	
3	Khosla Ventures, LowercaseCapital, capitalG	4	

	High Valuation	Continent Number
0	high	2
1	high	1
2	high	2
3	high	1
4	high	3

Hint

Create a mapping dictionary and use the `replace()` method on the `Category` column. Refer to the example provided above for more information about `replace()`.

1.3.7 Convert Country/Region to numeric

Now, suppose that within a given continent, each company's `Country/Region` is given equal importance. For analytical purposes, you want to convert the values in this column to numeric without creating a large number of dummy columns. Use label encoding of this nominal categorical variable to create a new column called `Country/Region Numeric`, wherein each unique `Country/Region` is assigned its own number.

```
[19]: # Create `Country/Region Numeric` column

      ### YOUR CODE HERE ###

      # Create numeric categories for Country/Region
      companies['Country/Region Numeric'] = companies['Country/Region'].
      ↪astype('category').cat.codes
```

Hint 1

Review what you have learned about converting a variable with a string/object data type to a category.

Hint 2

To use label encoding, apply `.astype('category').cat.codes` to the `Country/Region` in `companies`.

1.3.8 Convert Industry to numeric

Finally, create dummy variables for the values in the `Industry` column.

```
[20]: # Convert `Industry` to numeric data

      ### YOUR CODE HERE ###
```

```
# Create dummy variables with Industry values
industry_encoded = pd.get_dummies(companies['Industry'])

# Combine `companies` DataFrame with new dummy Industry columns
companies = pd.concat([companies, industry_encoded], axis=1)
```

Display the first few rows of companies

```
[21]: companies.head()
```

```
[21]:
```

	Company	Valuation	Date Joined	Industry	\
0	Bytedance	180	2017-04-07	Artificial intelligence	
1	SpaceX	100	2012-12-01	Other	
2	SHEIN	100	2018-07-03	E-commerce & direct-to-consumer	
3	Stripe	95	2014-01-23	Fintech	
4	Klarna	46	2011-12-12	Fintech	

	City	Country/Region	Continent	Year Founded	Funding	\
0	Beijing	China	Asia	2012	\$8B	
1	Hawthorne	United States	North America	2002	\$7B	
2	Shenzhen	China	Asia	2008	\$2B	
3	San Francisco	United States	North America	2010	\$2B	
4	Stockholm	Sweden	Europe	2005	\$4B	

	Select Investors	Years To Unicorn	\
0	Sequoia Capital China, SIG Asia Investments, S...	5	
1	Founders Fund, Draper Fisher Jurvetson, Rothen...	10	
2	Tiger Global Management, Sequoia Capital China...	10	
3	Khosla Ventures, LowercaseCapital, capitalG	4	
4	Institutional Venture Partners, Sequoia Capita...	6	

	High Valuation	Continent Number	Country/Region Numeric	\
0	high	2	9	
1	high	1	44	
2	high	2	9	
3	high	1	44	
4	high	3	38	

	Artificial intelligence	Auto & transportation	Consumer & retail	\
0	1	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	Cybersecurity	Data management & analytics	\
0	0	0	

1	0	0
2	0	0
3	0	0
4	0	0

	E-commerce & direct-to-consumer	Edtech	Fintech	Hardware	Health	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	1	0	0	0	0	
3	0	0	1	0	0	
4	0	0	1	0	0	

	Internet software & services	Mobile & telecommunications	Other	\
0	0		0	0
1	0		0	1
2	0		0	0
3	0		0	0
4	0		0	0

	Supply chain, logistics, & delivery	Travel
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

Hint 1

Consider using `pd.get_dummies` on the specified column.

Hint 2

When you call `pd.get_dummies()` on a specified series, it will return a dataframe consisting of each possible category contained in the series represented as its own binary column. You'll then have to combine this new dataframe of binary columns with the existing `companies` dataframe.

Hint 3

You can use `pd.concat([col_a, col_b])` to combine the two dataframes. Remember to specify the correct axis of concatenation and to reassign the result back to the `companies` dataframe.

Question: Which categorical encoding approach did you use for each variable? Why?

- **Continent** - Ordinal label encoding was used because there was a hierarchical order to the categories.
- **Country/Region** - Nominal label encoding was used because there was not a hierarchical order the categories.
- **Industry** - Dummy encoding was used because there were not many different categories represented and they were all equally important.

Question: How does label encoding change the data?

Label encoding changes the data by assigning each category a unique number instead of a qualitative value.

Question: What are the benefits of label encoding?

Label encoding is useful in machine learning models, because many types of machine learning require all variables to be of a numeric data type.

Question: What are the disadvantages of label encoding?

Label encoding may make it more difficult to directly interpret what a column value represents. Further, it may introduce unintended relationships between the categorical data in a dataset.

1.4 Conclusion

What are some key takeaways that you learned during this lab?

- Input validation is essential for ensuring data is high quality and error-free.
- In practice, input validation requires trial and error to identify issues and determine the best way to fix them.
- There are benefits and disadvantages to both label encoding and dummy/one-hot encoding.
- The decision to use label encoding versus dummy/one-hot encoding needs to be made on a case-by-case basis.

Reference

Bhat, M.A. *Unicorn Companies*

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.