HyperionDev

Data Cleaning and Preprocessing

Task

Visit our website

Introduction

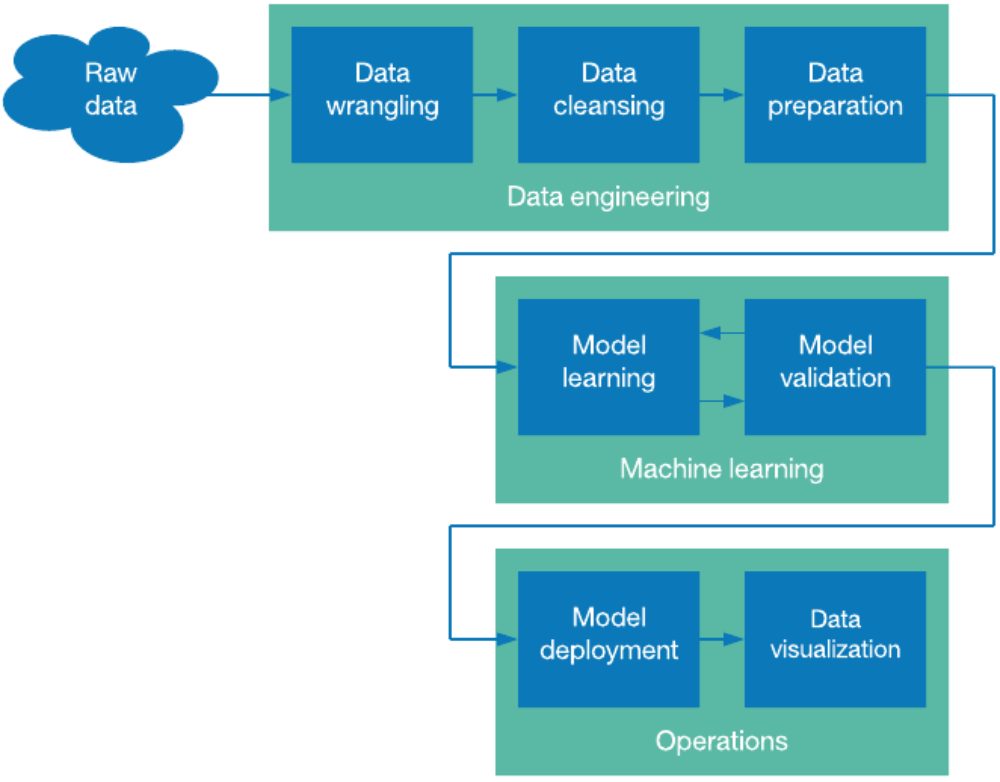
We live in a digital age where vast amounts of data are generated and stored daily. In this era, it is crucial to organise, manipulate, and interpret data effectively. Data cleaning stands as a fundamental pillar in the arsenal of any data scientist. Without proper data cleaning and preprocessing, creating accurate models or deriving meaningful insights becomes exceedingly challenging. Moreover, inaccuracies stemming from poorly cleaned or formatted data can lead to misguided assumptions and flawed decisions. Hence, data scientists devote a substantial portion of their time to cleaning and transforming data, ensuring it is appropriately formatted, free of nonsensical outliers, and devoid of erroneous entries.

In the upcoming lesson, we will delve into the realm of data cleaning, focusing on how to handle categorical and missing data, as well as scaling data.

Before we focus on data cleaning and preprocessing or data wrangling let's step back and have a look at the bigger picture.

The data science pipeline

The diagram below provides a high-level overview of the steps that are usually involved in the data science pipeline:



The data science pipeline (Jones, 2018)

There are several key steps that data scientists perform to analyse data in a meaningful way. While data scientists might use slightly different terminology and approaches, the core process of transforming raw data into actionable insights follows a general pipeline. This pipeline consists of several stages, though the order can be flexible depending on the specific project.

The first step in the pipeline is to collect or identify raw data. Sources for raw data are plentiful and diverse. Public repositories like Kaggle and UCI Machine Learning Repository offer a wealth of datasets for various tasks. Government agencies release open data, including census data and climate records. Web scraping can extract data from websites, while APIs provide structured data from services like Twitter and Google Analytics. Sensor data from IoT devices and wearable tech offers real-time insights. Finally, businesses often have internal databases of customer interactions, sales figures, and operational metrics, which can be a valuable source of proprietary data.

Code hack

A useful Python library is **Scrapy**; it can extract and collect data from web pages and web APIs!

Once you have obtained your raw data the next step is data engineering. There are three main tasks in data engineering:

- Data wrangling
- Data cleaning
- Data preparation

Data wrangling is defined as “the process by which you identify, collect, merge, and preprocess one or more datasets in preparation for data cleansing” (Jones, 2018). Once you have wrangled the data into a format with which you can work, you clean the data by fixing any errors in the dataset. This could involve dealing with missing data, correcting formatting issues, and fixing, or removing, incorrect data. You can then prepare the data for machine learning. Various visualisations (such as graphs, tables, and charts) and descriptive statistics are also used to explore the data during this phase.

During the machine learning step, you will build and validate models to gain insight into the data.

Finally, the operations step involves deploying machine learning models and producing visualisations to present the findings of the models. Data visualisation is important at both ends of the data science pipeline to present the findings of a study and to help gain an understanding of data at the start. Deploying models is outside the scope of this bootcamp, but the visualisation techniques you will learn about will be relevant to this phase of the data science pipeline.

Python packages for data science

For each of the steps in the data science pipeline, there are third-party Python packages and libraries that you can leverage to perform the tasks in each step. As you know packages contain a collection of code files that have already been written so you do not need to reinvent the wheel to achieve certain functionality in your program. In this lesson, we will be using pandas and scikit-learn.

Recall pandas is a powerful Python library specifically designed for data analysis and manipulation. Explore the **pandas documentation** to leverage all the methods and properties it provides.

Scikit-learn (or sklearn) is a simple starting point for using machine learning. It comes bundled with many techniques, such as preprocessing, which is used during data preparation. However, it also comes bundled with many of the basic machine learning algorithms. It also has a simple and intuitive interface: simply instantiate your model, fit it to the data, and make a prediction. It also includes methods for calculating accuracy, loss, etc.!

FuzzyWuzzy is a Python library for fuzzy string matching. It calculates similarity between strings, making it ideal for tasks like data cleaning, record linkage, and search functionality.

Chardet is a Python library for detecting character encodings in text data. It is useful for handling text files with unknown encodings, ensuring correct interpretation and processing.

Take note

A package is a collection of related modules (Python files) that enable some particular functionality. A library is a more broad term meaning a bundle of code. These terms are often used interchangeably (Cepalia, n.d.).

Before we explore some data cleaning and preprocessing techniques let's check you have the packages you will need for this lesson installed. Open your terminal and run the following commands one step at a time:

```
pip3 show pandas
pip3 show scikit-learn
pip3 show fuzzywuzzy
pip3 show chardet
```

Use the following command to install pandas and scikit-learn if you do not have them installed.

```
pip install -U pandas
pip install -U scikit-learn
pip install -U fuzzywuzzy
pip install -U chardet
```

It may take some time for the packages to install.

Dropping columns in a dataframe

Often, you will find that not all the variables in a dataset are useful to you, or are not complete enough to be used in the analysis. For example, you might have a dataset containing student information (shown in the image below), but you want to focus on analysing student grades. In such a scenario, the 'address' or 'parents' names' categories are not important. Thus, it may be unnecessary (or even inefficient) to store all these columns.

first_name	last_name	student_id	gender	math_score	english_scor	science_scor	Mother	Father	Address
Theresa	Imore	1	F	89	78	50	Jane Claremint	Michael Scofield	90 East Buckingham
Brook	Gratrex	2	M	67	56	65	Joesephine Brow	Lincon Burrows	60 Fairway Ave.
Jerrold	Isenor	3	M	98	67	42	Claire Mint	Ballack Benet	749 West Kingston Street
Jo	Pretsel	4	M	56	72	87	Alexis Mahone	Duke Harry	Lawrence Township, NJ

When cleaning data, save the data that you need. What you need will depend on the goal of your data analysis. For example, if we wanted to do a study of the population characteristics of the students, then data such as the students' addresses would be important, but information about the students' scores may not be necessary. To drop columns in pandas, the syntax is:

```
school_data.drop(["math_score", "english_score", "science_score"], axis=1,
inplace=True)
```

Note the axis parameter is set to 1 (`axis=1`). This drops a specific column (and not a row). In addition, `inplace=True` tells the method to change the DataFrame (`school_data`) itself (and not return a copy).

Replacing values

Sometimes it is important to replace a value from your dataset with another value. For example, if you store data about a person's 'Nationality' but your data describes British nationals as either 'British', 'English', 'UK' or 'United Kingdom', you may want to replace all other values with one term e.g. 'British'. To do this, you could create a function that will look for all instances of the phrase (or phrases) and replace it (or them) with the desired output. You could choose to handle null values in a similar manner. Null values in a dataset are data points that are empty.

For example, for the variable 'Nationality' if we have an empty entry in a record, how would you suggest we handle that? We can replace all null values with a default value such as 'Other' under the assumption that some people did not want to disclose their nationality. This would look something like this:

```
my_data.Nationality.fillna('Other', inplace=True)
```

A way to specify the column name is by using `.Nationality`. The `fillna` method fills all blank values with the specified value. To do more general replacements, such as replacing 'UK' with 'United Kingdom", use `replace()`:

```
my_data.Nationality.replace('UK', 'United Kingdom', inplace=True)
```

Working with categorical data

As discussed previously, categorical data deals with discrete (individually separate and distinct) data that fits neatly into a number of categories. Due to the nature of categorical data, we need some special tools to analyse and use it in machine learning models. Data tables and visualisations are often used to analyse this type of data and encoding transforms categorical data for use in machine learning models.

Data tables

Data tables are often used to count the number of variables within a particular category. For example, you may want to analyse a dataset that stores data about HyperionDev students. The dataset could contain categorical data such as which Bootcamp a student is currently registered for. Therefore, it could be useful to create a data table that displays the total number of students registered for each bootcamp to help analyse this dataset.

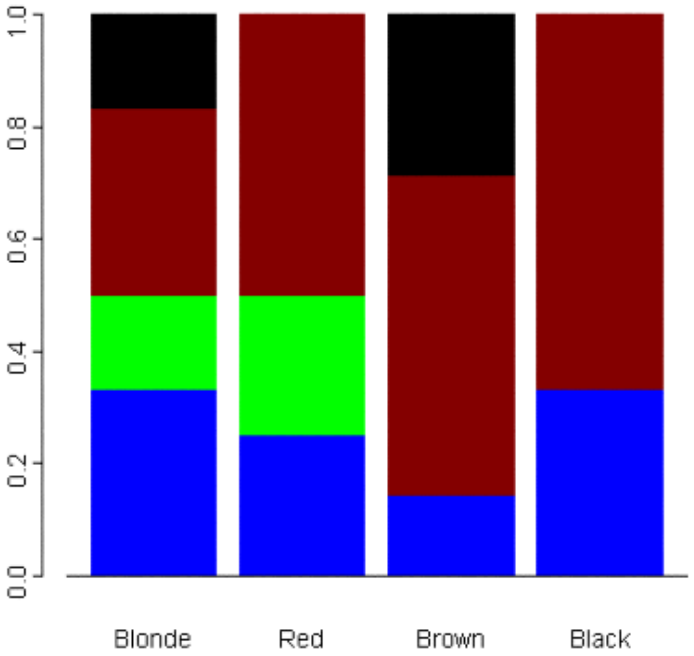
Two-way tables are useful when analysing how data in two categorical variables relate. Consider this example by Lacey: "suppose a survey was conducted of a group of 20 individuals, who were asked to identify their hair and eye colour.

A two-way table presenting the results might appear as follows"(Lacey, n.d.):

Hair colour	**Eye colour**				
	Blue	Green	Brown	Black	**Total**
Blonde	2	1	2	1	**6**
Red	1	1	2	0	**4**
Brown	1	0	4	2	**7**
Black	1	0	2	0	**3**
Total	**5**	**2**	**10**	**3**	**20**

The totals for each category are known as marginal distributions. Two-way tables are often converted to percentages to make this data easier to analyse. For example, it may be more helpful to know what percentage of people surveyed have blue eyes or what percentage of people with red hair have green eyes.

Segmented bar graphs are also often useful for analysing categorical data. For example, notice how the segmented bar graph below can be used to represent the information about people's eye colour (colour-coded) and hair colour:



Bar graph representing eye colour and hair colour (Lacey, n.d.)

Sometimes it is helpful to introduce a categorical variable into a dataset to make continuous variables easier to analyse. For example, suppose you had a dataset that stores the weight for several boxers (continuous variable). In that case, it may be useful to introduce a weight range variable (e.g. minimum_weight, light_fly_weight, fly_weight, etc.).

Encoding categorical variables

Machine learning models are unable to handle string values directly, so categorical data must be converted into a numerical form. There are two primary methods for doing this.

Label encoding

This technique assigns a unique integer to each category, such as 0 for 'male' and 1 for 'female'. However, this approach introduces a problem – by using numbers, it suggests a ranking or order between the categories, which can confuse some machine learning models into thinking one category is inherently 'greater' than the other.

One-hot encoding

This method avoids the issue of implied order by creating separate binary columns for each category. For example, instead of assigning numbers to male and female, one-hot encoding creates two new columns – `Sex_male` and `Sex_female`. If a person is male, there will be a 1 in the `Sex_male` column and a 0 in the `Sex_female` column, and vice versa for females. This method ensures each category is treated independently, without any implied hierarchy.

In pandas, a straightforward way to perform one-hot encoding is by using the `get_dummies()` function, which automatically generates these binary columns for categorical variables in your dataset.

Take note

One caveat with using one-hot encoding is that you can run into the dummy variable trap. Explore what the **dummy variable trap** is and how to solve it

Missing variables

As you have already learnt, before you can do any useful analysis, it is important to clean your dataset. One problem that you will often encounter when cleaning data is missing data. Data can have missing values for many reasons. For example, an observation (a piece of data) may not have been collected or recorded, or data corruption may have occurred. Data corruption is when data becomes unusable, unreadable or in some other way inaccessible to a user or application. There is no perfect way of dealing with missing data. The approach you use will depend on a good understanding of the data and the effect that the method you use will have on the dataset. The common approaches to handling missing data are trying to find missing data, removing observations with missing data, or using an imputation method to replace missing data with a substitute value. However, before deciding how to deal with missing data, it is essential to understand why it is missing.

Missing data can be categorised into three types:

1. **Missing completely at random (MCAR):** This means that the missing data is not in any way related to any other data in the dataset.

   For example, data may be missing because a test paper was lost or a blood sample was damaged. This means there is no way to predict what was missing (and therefore no reasonable way to guess what those values are). This is entirely random.

2. **Missing at random (MAR):** This is when the missing data is somehow related to other data in the dataset. If data is MAR, it can sometimes be predicted based on other data in the dataset. In other words, MAR data can be explained by other data measured in the dataset. This is due to implicit biases in the data itself. These biases may be used partially to gain a more realistic statistic.

   For example, consider a survey on depression. Studies have shown that males are less likely to fill in surveys about depression severity. In other words, this missingness can be determined by their gender (which was noted in the dataset).

3. **Missing not at random (MNAR):** This occurs when there is a direct relation to some missing data that the researchers haven't measured. Like with MAR, certain biases affect the values in the dataset. However, these biases are from factors not measured in the dataset.

   An example of this is in COVID reporting. Since restrictions were lifted and it had a lower impact on our lives, we saw a drop in the reported COVID cases. This is not because lifting restrictions makes the disease go away – it is just that people are less likely to get tested. This is something that scientists cannot measure and, therefore, cannot be imputed.

Imputating missing values

In some cases, missing data can be replaced with substitute values instead of removing entries with missing values. Whether we remove observations with missing data or substitute the missing value with another value, we must be careful not to create bias! Rumsey defines **statistical bias** as the "systematic favouritism that is present in the data collection process, resulting in lopsided, misleading results." When we remove or add substituted data, we could cause bias by creating a dataset favouring a certain idea. For example, if you remove mainly data about people with low income or assume that all those with a missing income value are high income, your findings could be distorted. (Rumsey, 2020)

Here are some methods that can be used to calculate substitute values.

Mean, median, and mode imputation

Using the measures of central tendency involves substituting the missing values with the mean or median for numerical variables and the mode for categorical variables. This imputation technique works well when the values are missing completely at random (MCAR). One disadvantage is that mean imputation reduces variance in the dataset.

Imputation with linear regression

This imputation technique utilises variables from the observed data to replace the missing values with predicted values from a regression model. Complete observations are used to generate the regression equation; the equation is then used to predict missing values for incomplete observations. In an iterative process, values for the missing variable are inserted, and then all cases are used to predict the dependent variable. These steps are repeated until there is little difference between the predicted values from one step to the next; that is, they converge. The major drawback of using this method is that it reduces variability. Though we have yet to introduce regression, it is important to keep this in mind.

K-nearest neighbour (KNN) imputation

For K-nearest neighbour imputation, the values are obtained by using a similarity-based method that relies on distance metrics (Euclidean distance, Jaccard similarity, Minkowski norm etc.). This method can predict both discrete and continuous attributes. KNN works by finding other observations that are most similar to the observation with the missing value. For example, if the observation is 'Female' and 'Asian', we can find other users similar to her and get the mean or mode of the missing value from the other observations. The main disadvantage of using KNN imputation is that it becomes time-consuming when analysing large datasets because it searches for similar instances in the entire dataset.

Feature scaling: normalisation and standardisation

Another common problem we encounter when trying to analyse data is having different units of measurement for a particular variable. For example, if you wanted to compare housing prices in different countries, you would have different datasets that store price values with different currencies. Clearly, it is not easy to compare these values. To get rid of the unit of measurement, we scale data by either normalising or standardising the data. In this section, we briefly consider some methods of doing this.

Normalisation

Normalisation involves scaling a variable to have a value between 0 and 1. Normalisation is often done using the formula below:

$$z = \frac{x \text{ - } \min(x)}{\max(x) \text{ - } \min(x)}$$

This helps to eliminate units of measurement for data. For example, suppose you are working with a dataset containing variables containing different measurement units, such as kilometres, miles, etc. In that case, using scaling reduces all the variables to a scale between 0 and 1, removing the need for units of measurement. This helps by allowing you to compare data from different sources. For example, if you wanted to compare housing prices in America vs. Germany, you would have to change your data to measure the prices using the same scale.

Standardisation

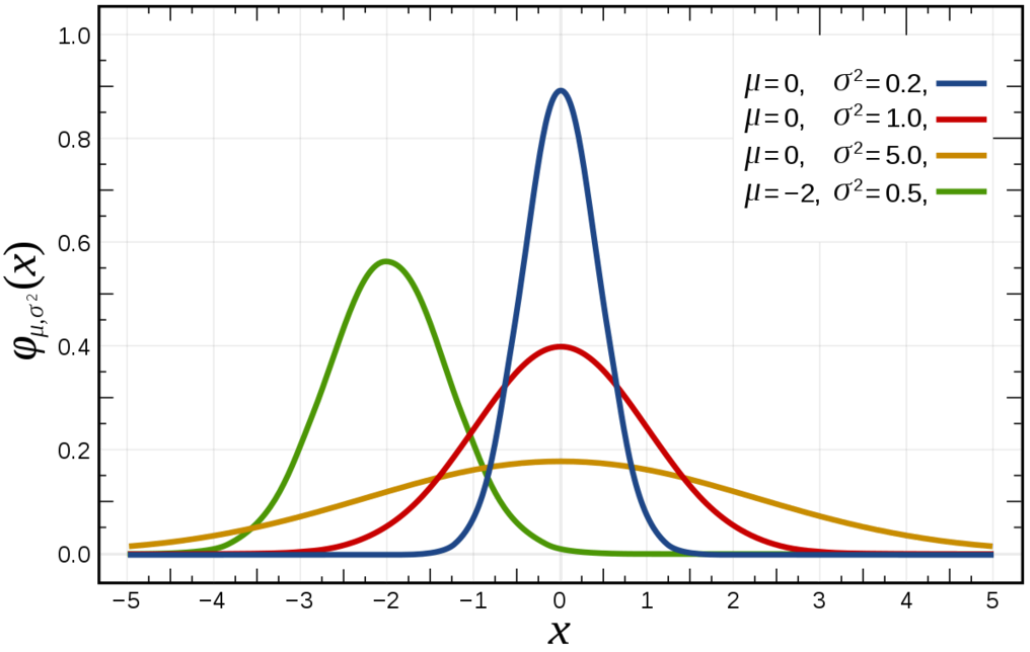
Standardisation involves scaling the data to have a mean (average) of 0 and a standard deviation (the average deviation from the mean in distribution) of 1. Standardisation uses this formula:

$$z = \frac{x_i - \mu}{\sigma}$$

This is a better form of feature scaling than normalisation, as the data does not restrict the scale as much as for normalisation. Consequently, outliers will not affect the data

range as much as normalisation. However, in order to standardise data, you must ensure that your data follows a **Gaussian distribution**.

Note a Gaussian distribution or normal distribution has a bell shape, as shown in the image below.



*Examples of normal distribution (**Wikimedia**, n.d.)*

Normalisation vs. standardisation

Now we know the difference between normalisation and standardisation. Normalisation restricts all values between 0 and 1, and standardisation causes the data to have a mean of 0 and a standard deviation of 1. But how do we know when to use which one? Quite simply, you must **standardise** the data when it follows a **Gaussian** distribution. If not, **normalise** the data.

We can only suggest some methods to clean and preprocess your data. Every dataset will come with a different set of unique problems and you will have to get creative to work it out.

Extra resource

Read more about normalisation and standardisation and the **effect of standardisation on machine learning** algorithms.

Instructions

In the following task, you will need to import the following libraries into your notebook. Click on the links for installation guidance if you have not already installed them.

- **Scikit-learn**
- **fuzzywuzzy**
- **chardet**

Take note

Within this task folder, you will find an example Jupyter notebooks. Read and work through these notebooks before moving on to the task.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task 1

In this task, you will clean the country column and parse the “date_measured” column in the **store_income_data_task.csv** file. Use the examples given in **data_cleaning_example.ipynb** to guide you.

Complete the following in **store_income_task.ipynb**:

- Load **store_income_data_task.csv**.
- Display the first five observations. (The first five rows of the dataset.)
- Take a look at all the unique values in the "country" column. Then, convert the column entries to lowercase and remove any trailing white spaces.
- Different names have been used to describe the same country in the “country” column. Clean up the “country” column so that there are three distinct countries.
- Create a new column called `days_ago` in the DataFrame that is a copy of the “date_measured” column but instead shows a number that represents the number of days ago that the income was measured. Note that the current date can be obtained using datetime.date.today().

Auto-graded task 2

Open the **data_preprocessing.ipynb** file and explore the examples, then follow these steps in the notebook for the **missing data** task:

1. Read in the **store_income_data_task.csv** file.
2. Display the first five observations. (The first five rows of the dataset.)
3. Get the number of missing values per column and print the results.
4. Write a note on why you think we have missing data on the following three columns: **store_email**, **department**, and **country**.
    - Remember to classify them according to the three categories of missingness we have considered.

Auto-graded task 3

Follow these steps:

This task handles **feature scaling** of variables. Continue to task 2 in **data_preprocessing.ipynb**, in which you do the following:

1. For the following examples, decide whether normalisation or standardisation makes more sense:

    a. You want to build a linear regression model to predict someone's grades, given how much time they have spent on various activities during a typical school week. You notice that your measurements for how much time students spend studying aren't normally distributed: some students spend almost no time studying, while others study for four or more hours daily. Should you normalise or standardise this variable?

    b. You are still working with your student's grades, but you also want to include information on how students perform on several fitness tests. You have information on how many jumping jacks and push-ups each student can complete in a minute. However, you notice that students perform far more jumping jacks than push-ups: the average for the former is 40, and for the latter, only 10. Should you normalise or standardise this variable?

2. Visualise the "EG.ELC.ACCS.ZS" column from the 'countries' dataset using a histogram. Then, scale the column using the appropriate scaling method (normalisation or standardisation). Finally, visualise the original and scaled data alongside each other. Note EG.ELC.ACCS.ZS is the percentage of the population with access to electricity.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Agarwal, M. (n.d.). Pythonic Data Cleaning With NumPy and Pandas. Retrieved April 23, 2019, from Real Python:
https://realpython.com/python-data-cleaning-numpy-pandas/

Boiko, S. (2018, October 3). Python for Machine Learning: Indexing and Slicing for Lists, Tuples, Strings, and other Sequential Types. Retrieved from Railsware.com:
https://railsware.com/blog/python-for-machine-learning-indexing-and-slicing-for-lists-tuples-strings-and-other-sequential-types/

Cepalia, A. (n.d.). Scripts, modules, packages, and libraries [Video]. Real Python.
https://realpython.com/lessons/scripts-modules-packages-and-libraries/

DeFilippi, R. R. (2018, April 29). Standardize or Normalize? Examples in Python. Retrieved April 13, 2019, from Medium.com:
https://medium.com/@rrfd/standardize-or-normalize-examples-in-python-e3f174b65dfc

Kwak, S. K., & Kim, J. H. (2017). Statistical data preparation: management of missing values and outliers. Korean journal of anesthesiology, 70(4), 407–411. doi:10.4097/kjae.2017.70.4.407

Lacey, M. (n.d.). Categorical data. Retrieved April 30, 2019, from stats.yale.edu:
http://www.stat.yale.edu/Courses/1997-98/101/catdat.htm

Laerd statistics. (n.d.). Types of variables. Retrieved April 30, 2019, from statistics.laerd.com:
https://statistics.laerd.com/statistical-guides/types-of-variable.php

Rumsey, D. (2020). How to Identify Statistical Bias - dummies. Retrieved 25 August 2020, from
https://www.dummies.com/education/math/statistics/how-to-identify-statistical-bias/

Swalin, A. (2018, January 31). How to Handle Missing Data. Retrieved May 13, 2019, from Towards Data Science:
https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4