Hands-on Lab: Generative AI for Data Preparation

Estimated time: 30 minutes

In practice, the initial part of a data science workflow involves cleaning and preparing data for better analysis. This part usually requires the removal of blank entries, normalization of numerical attributes, numerical interpretation of categorical variables, and so on. In this lab, you will use a generative AI model to create a Python code to perform all the required tasks on a real-world data set.

Learning objectives

In this lab, you will learn how to use generative AI for creating a Python code to:

- Handle missing values in the data set
- Correct the data type for the required data set attributes
- Perform standardization and normalization on required parameters
- Convert categorical data into numerical indicator variables

Code execution environment

To test the prompt-generated code, keep the Jupyter Notebook (in the link below) open in a separate tab in your web browser. The notebook has some setup instructions that you should complete now.

Jupyter-Lite Test Environment

The data set for this lab is available in the following URL:

```
1
```

```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/laptop_pricing_dataset_mod1.csv
```

Copied!

The dataset is a filtered and modified version of the Laptop Price Prediction using specifications dataset, available under the Database Contents License (DbCL) v1.0 on the Kaggle website.

Complete the setup in the Jupyter Notebook and then proceed to the following actions.

Building the prompt: Importing the data set

Now, you will start building your prompt. First, you must ask the Gen AI model to generate a code to import the provided data set to a Pandas' data frame. You must specify if you are importing the data. Assuming the first row is the header, the data set headers should be available as the first row of the CSV file.

The prompt to create the code can be structured as follows.

```
1
2
```

```
Write a Python code that can perform the following tasks.
Read the CSV file, located on a given file path, into a Pandas data frame, assuming that the first rows of the file are the headers for the data.
```

Copied!

Running this prompt using the GPT 3.5 engine in the GenAI classroom should result in the following code:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

```
import pandas as pd

# Specify the file path
file_path = "path/to/your/file.csv"

# Read the CSV file into a Pandas data frame
df = pd.read_csv(file_path)

# Assuming the first rows of the file are the headers, you don't need to specify any additional parameters

# Additional details:
# - The `pd.read_csv()` function is used to read a CSV file into a Pandas data frame.
```

```
# - By default, it assumes that the first row of the file contains the headers for the data.
# - If your file doesn't have headers, you can specify `header=None` as an additional parameter.
# - You can also specify other parameters, such as `sep` to specify the delimiter used in the file.
# - Make sure you have the Pandas library installed in your Python environment before running this code.
```

Copied!

You may copy the code from the response and try it on the Jupyter Notebook with appropriate modifications in the file path.

Building the prompt: Handle missing data

You can now ask the Generative AI model to generate a script to handle the missing data.

First, use the model to identify the attributes with missing data.

For this, you may run the following query.

```
1
```

```
Write a Python code that identifies the columns with missing values in a pandas data frame.
```

Copied!

You can use the relevant part of the code generated for your task.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
import pandas as pd

# Assuming you already have a Pandas data frame named 'df'

# Identify columns with missing values
columns_with_missing_values = df.columns[df.isnull().any()]

# Additional details:
# - The `df.isnull()` function returns a Boolean data frame where each cell is True if it contains a missing value (NaN), and False otherwise.
# - The `df.columns` attribute returns the column labels of the data frame.
# - The `.any()` method returns a Boolean Series indicating whether any value in the given axis (in this case, columns) is True.
# - Finally, the `.columns` attribute is used to retrieve the column labels where the condition is True.

# You can now use the 'columns_with_missing_values' variable to further analyze or handle the columns with missing values.
```

Copied!

Once run, you will see that the two attributes with missing values are "Screen_Size_cm", a categorical variable, and "Weight_kg", a continuous variable.

Now, you need to replace the missing values with appropriate values. The following are the rules for this:

1. Missing entries in columns containing categorical values need to be replaced with the most frequent entries
2. Missing entries in columns with continuous data need to be replaced with the mean value of the column
3. If a value is missing in the target column, you may need to drop that row

You can build a prompt for this, as shown below.

```
1
2
3
```

```
Write a Python code to replace the missing values in a pandas data frame, per the following guidelines.
1. For a categorical attribute "Screen_Size_cm", replace the missing values with the most frequent value in the column.
2. For a continuous value attribute "Weight_kg", replace the missing values with the mean value of the entries in the column.
```

Copied!

The prompt will generate a code. You can extract the required statements, as shown below.

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
```

```
 1. import pandas as pd
 2.
 3. # Assuming you already have a Pandas data frame named 'df'
 4.
 5. # Replace missing values in the 'Screen_Size_cm' column with the most frequent value
 6. most_frequent_value = df['Screen_Size_cm'].mode()[0]
 7. df['Screen_Size_cm'].fillna(most_frequent_value, inplace=True)
 8.
 9. # Replace missing values in the 'Weight_kg' column with the mean value
10. mean_value = df['Weight_kg'].mean()
11. df['Weight_kg'].fillna(mean_value, inplace=True)
12.
13. # Additional details:
14. # - The `.mode()` method is used to calculate the most frequent value in a column.
15. # - The `[0]` indexing is used to retrieve the most frequent value from the resulting Series.
16. # - The `.fillna()` method is used to replace missing values with a specified value.
17. # - The `inplace=True` parameter is used to modify the original data frame instead of creating a new one.
18.
19. # You can now use the modified 'df' data frame, which has the missing values replaced according to the guidelines.
```

Copied!

Building the prompt: Modify data type

Further, you should update both attributes'data type to floating values.

You should see a similar response to the following prompt:

```
1. 1
```

```
1. Write a Python code snippet to change the data type of the attributes "Screen_Size_cm" and "Weight_kg" of a data frame to float.
```

Copied!

You should see a code as shwon below:

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
```

```
 1. import pandas as pd
 2.
 3. # Assuming you already have a Pandas data frame named 'df'
 4.
 5. # Change the data type of 'Screen_Size_cm' and 'Weight_kg' to float
 6. df['Screen_Size_cm'] = df['Screen_Size_cm'].astype(float)
 7. df['Weight_kg'] = df['Weight_kg'].astype(float)
 8.
 9. # Additional details:
10. # - The `.astype()` method is used to change the data type of a column.
11. # - In this case, we're specifying `float` as the desired data type.
12. # - Make sure the columns contain numeric values that can be converted to float.
13. # - If there are any non-numeric values in the columns, the conversion will raise an error.
14.
15. # You can now use the modified 'df' data frame, which has the data types of 'Screen_Size_cm' and 'Weight_kg' changed to float.
```

Copied!

You can use the relevant part of the code in your script.

Once run, the code data attributes will have the correct data type.

Building the prompt: Standardization and Normalization

You may notice that the parameter `Screen_Size_cm` represents the screen size in centimeters instead of the standard unit, which is inches. Also notice that the parameter `Weight_kg` would be better expressed in pounds.

You can ask generative AI to create a code to convert these parameters without having to mention the conversion rates.

You can build a prompt for this, as shown below.

```
1
2
3
```

```
Write a Python code to modify the contents under the following attributes of the data frame as required.
1. Data under 'Screen_Size_cm' is assumed to be in centimeters. Convert this data into inches. Modify the name of the attribute to 'Screen_Size_inch'
2. Data under 'Weight_kg' is assumed to be in kilograms. Convert this data into pounds. Modify the name of the attribute to 'Weight_pounds'.
```

Copied!

The expected output should look as shown below.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

```
import pandas as pd

# Assuming you already have a Pandas data frame named 'df'

# Convert 'Screen_Size_cm' from centimeters to inches and modify the attribute name
df['Screen_Size_inch'] = df['Screen_Size_cm'] * 0.393701
df.drop('Screen_Size_cm', axis=1, inplace=True)

# Convert 'Weight_kg' from kilograms to pounds and modify the attribute name
df['Weight_pounds'] = df['Weight_kg'] * 2.20462
df.drop('Weight_kg', axis=1, inplace=True)

# Additional details:
# - The code multiplies the values under 'Screen_Size_cm' by 0.393701 to convert centimeters to inches.
# - The resulting values are stored in a new attribute named 'Screen_Size_inch'.
# - The original 'Screen_Size_cm' attribute is dropped from the data frame using the `.drop()` method.
# - Similarly, the code multiplies the values under 'Weight_kg' by 2.20462 to convert kilograms to pounds.
# - The resulting values are stored in a new attribute named 'Weight_pounds'.
# - The original 'Weight_kg' attribute is dropped from the data frame.

# You can now use the modified 'df' data frame, which has the contents and attribute names modified as required.
```

Copied!

You can use the relevant part of the code in your script.

It may also be required to normalize the data under some attributes. Since there are many normalization forms, mentioning the exact needs and tasks is important. Also, you can save the normalized data as a new attribute or change the original attribute. You need to ensure that all the details of the prompt are clear.

For example, let us assume that the data under “CPU_frequency” needs to be normalized w.r.t. the max value under the attribute. You need the changes to be reflected directly under the attribute instead of creating a new attribute.

You can ask Generative AI to generate a script that does this for you.

You can build a prompt for this, as shown below.

```
1
```

```
Write a Python code to normalize the content under the attribute "CPU_frequency" in a data frame df concerning its maximum value. Make changes to the
```

Copied!

This should generate the response as shown below.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
import pandas as pd

```

```
# Assuming you already have a Pandas data frame named 'df'

# Normalize the content under 'CPU_frequency' with respect to its maximum value
max_value = df['CPU_frequency'].max()
df['CPU_frequency'] = df['CPU_frequency'] / max_value

# Additional details:
# - The code calculates the maximum value of the 'CPU_frequency' attribute using the `.max()` method.
# - It then divides the values under 'CPU_frequency' by the maximum value to normalize them.
# - The resulting normalized values overwrite the original values in the 'CPU_frequency' attribute.

# You can now use the modified 'df' data frame, which has the content under the 'CPU_frequency' attribute normalized.
```

Copied!

Use the relevant part of the code in your script.

Building the prompt: Categorical to numerical

For predictive modeling, the categorical variables are not usable currently. So, you must convert the important categorical variables into indicator numerical variables. Indicator variables are typically new attributes, with content being 1 for the indicated category and 0 for all others. Once you create the indicator variables, you may drop the original attribute.

For example, assume that attribute `Screen` needs to be converted into individual indicator variables for each entry. Once done, the attribute `Screen` needs to be dropped.

You can build a prompt for this, as shown below.

```
1
2
3
4
```

```
Write a Python code to perform the following tasks.
1. Convert a data frame df attribute "Screen", into indicator variables, saved as df1, with the naming convention "Screen_<unique value of the attrib
2. Append df1 into the original data frame df.
3. Drop the original attribute from the data frame df.
```

Copied!

The expected response is as shown below.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

```
import pandas as pd

# Assuming you already have a Pandas data frame named 'df'

# Convert the 'Screen' attribute into indicator variables
df1 = pd.get_dummies(df['Screen'], prefix='Screen')

# Append df1 into the original data frame df
df = pd.concat([df, df1], axis=1)

# Drop the original 'Screen' attribute from the data frame
df.drop('Screen', axis=1, inplace=True)

# Additional details:
# - The `pd.get_dummies()` function is used to convert a categorical attribute into indicator variables.
# - The resulting indicator variables are stored in a new data frame named 'df1'.
# - The `prefix` parameter is used to specify the naming convention for the indicator variables.
# - The `pd.concat()` function is used to concatenate the original data frame 'df' and 'df1' along the column axis (axis=1).
# - The resulting concatenated data frame is assigned back to 'df'.
# - Finally, the `.drop()` method is used to drop the original 'Screen' attribute from 'df'.

# You can now use the modified 'df' data frame, which has the 'Screen' attribute converted into indicator variables, appended, and the original attri
```

Copied!

You can use the relevant part in your code.

Practice problems

1. Create a prompt to generate a Python code that converts the values under `Price` from USD to Euros.
2. Modify the normalization prompt to perform min-max normalization on the `CPU_frequency` parameter.

Conclusion

Congratulations! You have completed the lab on data preparation.

With this, you have learned the process of using generative AI to create Python codes that can:

1. Handle missing values in the data frame
2. Modify the attribute data types
3. Perform attribute transformations like standardization and normalization
4. Convert categorical attributes into indicator variable attributes.

Author(s)

Abhishek Gagneja