9 - Preprocessing

Data Science Practicum 2021/22, Lesson 9

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Pre-processing

- The typical machine learning work-flow has the following steps:
  1. Acquire data
  2. Pre-process data
  3. Train/learn model
  4. Evaluate model
  5. Deploy model

Pre-processing

- The typical machine learning work-flow has the following steps:
  1. Acquire data
  2. Pre-process data
  3. Train/learn model
  4. Evaluate model
  5. Deploy model

- The **pre-processing** step can do many things:
  - Data cleaning
    - Missing values management
    - Duplicate values
    - Inconsistent data (e.g. Gender: M, Pregnant: True)
  - Feature scaling:
    - Standardization
    - Normalization
    - Binning
  - Dimensionality reduction

Table of Contents

Missing Values

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	0.0	11.0	12.0	NaN

Missing Values

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	0.0	11.0	12.0	NaN

```
df.isnull()
```

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

Missing values

- What can we do?
  - Remove data with missing values (rows, columns)
  - Replace missing values (impute) with some data (mean, median, constant, random . . .)
    - Imputed values may by systematically above or below their actual values
    - Rows with missing values may be unique in some other way

Drop missing values

- Eliminates a sample (row)

```
df.dropna(axis=0)
```

- Eliminates a feature (column)

```
df.dropna(axis=1)
```

- Remove only rows where NaN appears in a specific column

```
df.dropna(subset=['D'])
```

Replace missing values

```
data_frame.replace(missing_value,new_value)
```

Replace missing values

```
data_frame.replace(missing_value,new_value)
```

```
mean = data_frame["engine-size"].mean()
data_frame=data_frame.replace("?",mean)
```

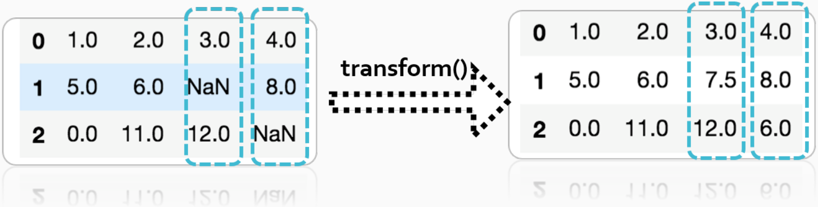
Scikit-learn

- Simple and efficient tools for data analytics
- Accessible and reusable in various applications
- Built on NumPy, and Matplotlib
- Open source, commercially usable - BSD license
- Pre-processing module: Imputer

```
from sklearn.preprocessing import Imputer
```

Imputer

```
from sklearn.preprocessing import Imputer

missing_values='NaN'
strategy='mean'

imputer =  Imputer(missing_values, strategy, axis=0)

imputer = imputer.fit(df)
imputed_data = imputer.transform(df.values)
```



Imputer Strategy

- The strategy parameter can:
  - If “mean”, then replace missing values using the mean along each column. Can only be used with numeric data.
  - If “median”, then replace missing values using the median along each column. Can only be used with numeric data.
  - If “most_frequent”, then replace missing using the most frequent value along each column. Can be used with strings or numeric data.
  - If “constant”, then replace missing values with fill_value. Can be used with strings or numeric data.

Exercise

Write a Pandas program to select the rows where the score is missing, i.e. is NaN.
Sample Python dictionary data and list labels:

```
exam_data = {
'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']
}
```

Exercise

Write a Pandas program to select the rows where the score is missing, i.e. is NaN.
Sample Python dictionary data and list labels:

```
exam_data = {
'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']
}
```

```
import pandas as pd
import numpy as np
exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}

df = pd.DataFrame(exam_data)
print("Rows where score is missing:")
print(df[df['score'].isnull()])
```

```
Rows where score is missing:
    name  score  attempts qualify
3  James    NaN         3      no
7  Laura    NaN         1      no
```

Exercise

- replace the missing values with -1

Exercise

- replace the missing values with -1

```
df = df.replace(np.nan,-1)
print(df[df['score'].isnull()])
```

```
Empty DataFrame
Columns: [name, score, attempts, qualify]
Index: []
```

Exercise

- Load the car_data.csv
- Run

```
df.hist(figsize = (10,10))
```

- Compute the number of missing values in each column (NaN values)?
  - Which column has the largest number of missing values?
  - What is mean and std of this column?
- Replace the missing values (NaN) of this column with the mean value.
- Now compute the mean and std of this column again. Has it changed?
  - Interpret the changes
- Repeat the exercise by replacing missing values with the median

Exercise

```
df = pd.read_csv("car_data.csv")
df.hist(figsize = (10,10))
print(df.isna().sum())
mean_before=df.loc[:,"normalized-losses"].mean()
std_before=df.loc[:,"normalized-losses"].std()

df=df.replace(np.NAN,mean_before)
mean_after=df.loc[:,"normalized-losses"].mean()
std_after=df.loc[:,"normalized-losses"].std()

print('mean before:',mean_before,'--->', \
      'mean_after:',mean_after)
print('std before:',std_before,'--->', \
      'std after:',std_after)
```

```
mean before: 122.0 ---> mean_after: 122.0
std before: 35.442167530553256 ---> std after: 31.75894428144489
```

Exercise

```
df = pd.read_csv("car_data.csv")
df.hist(figsize = (10,10))
print(df.isna().sum())
mean_before=df.loc[:,"normalized-losses"].mean()
std_before=df.loc[:,"normalized-losses"].std()

df=df.replace(np.NAN,mean_before)
mean_after=df.loc[:,"normalized-losses"].mean()
std_after=df.loc[:,"normalized-losses"].std()

print('mean before:',mean_before,'--->', \
      'mean_after:',mean_after)
print('std before:',std_before,'--->', \
      'std after:',std_after)
```

```
mean before: 122.0 ---> mean_after: 122.0
std before: 35.442167530553256 ---> std after: 31.75894428144489
```

```
mean before: 122.0 ---> mean_after: 120.62745098039215
std before: 35.442167530553256 ---> std after: 31.88091189448927
```

Table of Contents

Standardization and Normalization

- It is desirable that the variables are roughly in the same range

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$$

- If $x \in (-\infty, \infty)$ we use standardization (z-normalization)

$$\hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}$$

- If $x \in [a, b]$ then we can use min-max scaling (normalization)

$$\hat{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- If distribution is skewed we may want to apply some nonlinear function

Binning

- Binning is assigning labels to ranges of values
  - e.g. when assigning student scores
- pandas qcut function: Discretize variable into equal-sized buckets based on rank or based on sample quantiles. For example 1000 values for 10 quantiles would produce a Categorical object indicating quantile membership for each data point.

```
pandas.qcut(x, q, labels=None, retbins=False, precision=3, duplicates='raise')
```

```
pd.qcut(df['some_column'], q=4)
```

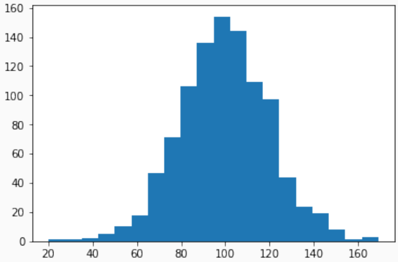
Exercise

- generate a sample with the normal distribution
  - mean = 100
  - std = 20
  - N=1000
- plot the histogram
  - 20 bins

Exercise

- generate a sample with the normal distribution
  - mean = 100
  - std = 20
  - N=1000
- plot the histogram
  - 20 bins

```
n = np.random.normal(100,20,1000)
plt.hist(n,bins=20)
plt.show
```



Exercise

- for the sample above, calculate:
  - mean
  - median
  - standard deviation

Exercise

- for the sample above, calculate:
  - mean
  - median
  - standard deviation

```
print(np.mean(n))
print(np.median(n))
print(np.std(n))
```

```
100.90677841590438
100.41003261542633
20.477742779833427
```

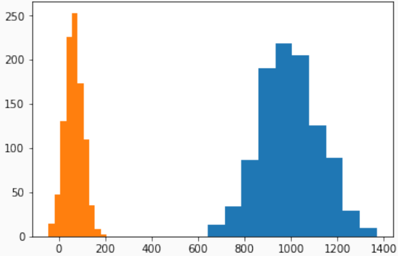
Exercise

- create a new sample s1: M=1000, SD=120
- create a new sample s2: M=64, SD=38
- plot both histograms

Exercise

- create a new sample s1: M=1000, SD=120
- create a new sample s2: M=64, SD=38
- plot both histograms

```
s1 = np.random.normal(1000,120,1000)
s2 = np.random.normal(64,38,1000)
plt.hist(s1)
plt.hist(s2)
plt.show()
```



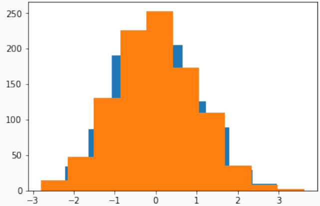
Exercise

- standardize the samples from the previous example (z normalization)
- plot both histograms on the same plot

Exercise

- standardize the samples from the previous example (z normalization)
- plot both histograms on the same plot

```
s1_s = (s1 -np.mean(s1))/np.std(s1)
s2_s = (s2 -np.mean(s2))/np.std(s2)
plt.hist(s1_s)
plt.hist(s2_s)
plt.show()
```

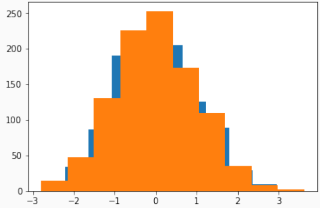


- how to make the histogram more informative (transparent -> google it)?

Exercise

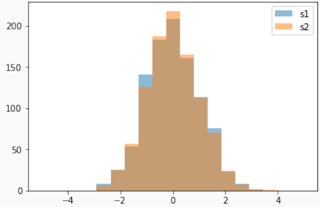
- standardize the samples from the previous example (z normalization)
- plot both histograms on the same plot

```
s1_s = (s1 -np.mean(s1))/np.std(s1)
s2_s = (s2 -np.mean(s2))/np.std(s2)
plt.hist(s1_s)
plt.hist(s2_s)
plt.show()
```



- how to make the histogram more informative (transparent -> google it)?

```
b = np.linspace(-5, 5, 20)
plt.hist(s1_s, bins=b, alpha=0.5, label="s1")
plt.hist(s2_s, bins=b, alpha=0.5, label="s2")
plt.legend(loc='upper right')
plt.show()
```



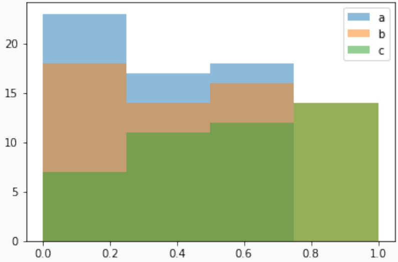
Exercsie

- you want to compare school grades from different countries:
  - A: min:0, max:5, pass:2
  - B: max:1, min:7, pass:4
  - C: min:1, max:10, pass:6
- how would you normalize the data to make it comparable?
- create such samples and normalize them!
- visualize properly

Exercsie

- you want to compare school grades from different countries:
  - A: min:0, max:5, pass:2
  - B: max:1, min:7, pass:4
  - C: min:1, max:10, pass:6
- how would you normalize the data to make it comparable?
- create such samples and normalize them!
- visualize properly

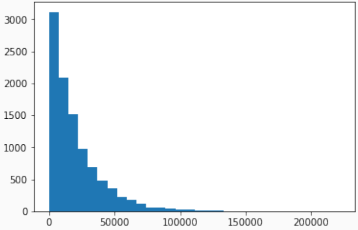
```
a = np.random.randint(0,5,100)
b = np.random.randint(1,7,100)
c = np.random.randint(1,10,100)

a_n = (a - 2)/(5-2)
b_n = (4-b)/(7-4)
c_n = (c-6)/(10-6)

b = np.linspace(0,1,5)

plt.hist(a_n, bins=b, alpha = 0.5, label = "a")
plt.hist(b_n, bins=b, alpha = 0.5, label = "b")
plt.hist(c_n, bins=b, alpha = 0.5, label = "c")
plt.legend(loc='upper right')
plt.show()
```



Exercise

- assume you have a skewed distribution of your feature
- how would you normalize/standardize it?

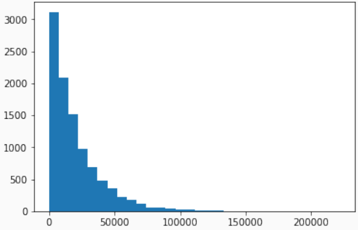
```
income = np.random.gamma(1,2,10000)*10000
plt.hist(income,bins=30)
plt.show()
```



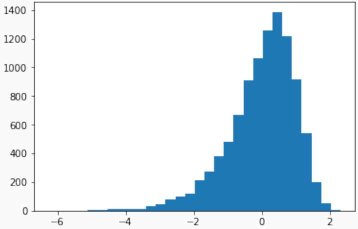
Exercise

- assume you have a skewed distribution of your feature
- how would you normalize/standardize it?

```
income = np.random.gamma(1,2,10000)*10000
plt.hist(income,bins=30)
plt.show()
```




```
income_norm = np.log(income)
i_n = (income_norm - np.mean(income_norm))/np.std(income_norm)
plt.hist(i_n,bins=30)
plt.show()
```



Exercise

Exercise - binning

- Load the dataset StudentsPerformance.csv
- plot the histogram of the `math score` column
- by binning create a new column in the dataframe:
  - Assign the scores "A", "A-", "B", "B-", and "C" based on the equal binning of the `math score` column

Exercise - binning

- Load the dataset StudentsPerformance.csv
- plot the histogram of the `math score` column
- by binning create a new column in the dataframe:
  - Assign the scores "A", "A-", "B", "B-", and "C" based on the equal binning of the `math score` column

```
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv("StudentsPerformance.csv")
df['math score'].plot(kind='hist')
plt.show()

labels_5 = ['C', "B-", 'B', 'A-', 'A']
df['math_score_7'] = pd.qcut(df['math score'], 5, labels=labels_5)
df
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	math_score_7
0	female	group B	bachelor's degree	standard	none	72	72	74	A-
1	female	group C	some college	standard	completed	69	90	88	B
2	female	group B	master's degree	standard	none	90	95	93	A
3	male	group A	associate's degree	free/reduced	none	47	57	44	C
4	male	group C	some college	standard	none	76	78	75	A-
...	...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95	A
996	male	group C	high school	free/reduced	none	62	55	55	B-
997	female	group C	high school	free/reduced	completed	59	71	65	B-
998	female	group D	some college	standard	completed	68	78	77	B
999	female	group D	some college	free/reduced	none	77	86	86	A-

1000 rows × 9 columns

Table of Contents

Assignment

- Load the file melb_data.csv
- The following instructions are valid for numeric data only
- Clean (deal with missing values) the data according to your judgment. Provide a short description (as comment in the code) why you used the approach you used
- Create a figure with many axeses, each representing the distribution of one of the numerical variables. Save it to a file
- Apply normalization/standardization/binning according to your judgment. Provide a short description (as comment in the code) why you used the approach you used
- Create a figure with many axeses, each representing the distribution of one of the variables of the corrected dataframe. Save it to a file
- Create a figure with many axes (one for each variable). In each axes display both histograms, before and after the correction

Table of Contents

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://towardsdatascience.com/sort-and-segment-your-data-into-bins-to-get-sorted-ranges-pandas-cut-and-qcut-7785931bbfde
- https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html