Applied Data Science

Session 6,7,8,9,10: Data Preprocessing

Dr. Soharab Hossain Shaikh

1

Agenda

Data Preprocessing

>> Introduction to Preprocessing
>> Data Cleaning – Tidy Data Principle
>> Handling Outliers - Detection & Removal
>> Handling Missing Values - Detection and Imputation
>> Handling Duplicates
>> Feature Engineering
>>> Feature Transformation – Normalization, Standardization
>>> Feature Scaling
>>> Feature Encoding
>> Feature Selection

2

Data Preprocessing

Different steps are involved in Data Preprocessing.
These steps are described below –

- Data Cleaning: This is the first step which is implemented in Data Preprocessing. The primary focus is on handling missing data, noisy data, detection, and removal of outliers, minimizing duplication and computed biases within the data.
- Data Integration: This process is used when data is gathered from various data sources and data are combined to form consistent data. This consistent data after performing data cleaning is used for analysis.

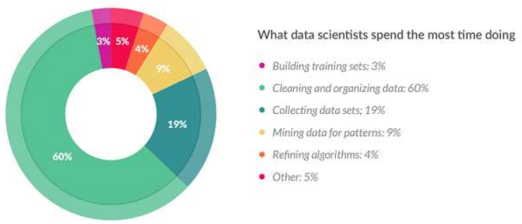
3

Data Preprocessing

- **Data Transformation**: This step is used to convert the raw data into a specified format according to the need of the model. The options used for transformation of data are given below -
- Normalization - In this method, numerical data is converted into the specified range, i.e., between 0 and one so that scaling of data can be performed.
- Aggregation – This method is used to combine the features into one. For example, combining two categories can be used to form a new group.
- Generalization - In this case, lower level attributes are converted to a higher standard.
- Data Reduction - After the transformation and scaling of data duplication, i.e., redundancy within the data is removed and efficiently organize the data.

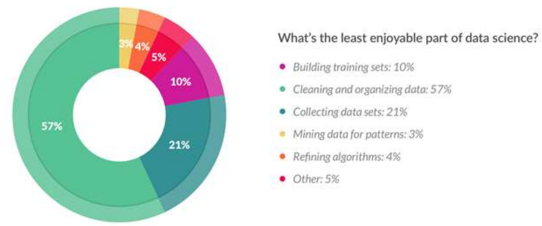
4

Data Cleaning



5

Data Cleaning



6

Data Cleaning

- ***Data Cleaning/cleansing*** is also referred to as ***Data Wrangling***, ***Data Munging***, ***Data Janitor Work*** and ***Data Preparation***.
- All of these refer to preparing data for **ingestion** into a data processing stream of some kind.
- Computers are very intolerant of format differences, so all of the data must be reformatted to conform to a standard (or "clean") format.
- Missing data and partial datasets can be problematic, so an initial goal is to identify data deficiencies before they lead to spurious results.
- It is generally not possible to carry out an **ETL** (Extract, Transform and Load) job without doing at least ***some*** data cleaning.
- If you are asked for a time estimate for an ETL job, remember to factor in time for data examination & data cleaning. [How to handle outliers (drop or not? if so, what is a good cutoff point? etc.).]

7

Data Cleaning

- Other requirements may include ***normalizing*** data sets, which generally means scaling the data to values between 0 and 1 (this enables certain types of numerical analysis).
- The end result may sometimes be referred to as ***tidy data***, however it is important to remember that data cleaning is not always a one-time task.
- The further use of any given dataset may well highlight details that need further cleaning.

8

Data Preprocessing for ML

- Step 1: Importing the required libraries These two are essential libraries which we will import everytime.
- Numpy is a Library which contains Mathematical functions . Pandas is the library used to import and manage the data sets.
- Step 2: Importing the Dataset Data sets generally available in .CSV format.A CSV files stores tabular data in plain text.Each line of the file is a data record. We use the read_csv method of the Pandas library to read a local CSV files as a dataframe.Then we make separate Matrix and Vector of independent and dependent variables from the dataframe.
- Step 3: Handling the Missing Data The data we get is rarely homogeneous Data can be missing due to various reasons and needs to be handled so that it does not reduce the performance of our machine learning model. We can replace the missing data by the Mean or Median of the entire column .We use Imputer class of sklearn.preprocessing for this task.
- Step 4: Encoding Categorical Data Categorical data are variables that contain label values rather than numeric values.The number of possible values is often limited to a fixed set.Examples values such as "Yes "and "No" cannot be used in mathematical equations of the model so we need to encode these variables into numbers .To achieve this we import LabelEncoder class from sklearn.preprocessing library.
- Step 5: Splitting the dataset into test set and training set We make two partitions of dataset one for training the model called training set and other for testing the performance of the trained model called test set . The split is generally 80/20. We import train_test_split() method of sklearn.crossvalidation library.
- Step 6: Feature Scaling Most of the machine learning algorithms use the Euclidean distance between two data points in their computations. features highly varying in magnitudes.units and range pose problems .high magnitude features will weigh more in the distance calculations than features with low magnitudes. Done by Feature standardization or Z-score normalization.standardscalar of sklearn.preprocessing is imported.

9

Data Cleaning

10

DATA CLEANING STEPS

Removing unwanted observations	• Duplicate/ redundant or irrelevant values deletion .
Missing Data handling	• Fixing issue of unknown missing values
Structural error solving	• Fixing problems with mislabeled classes, typesin names of features, same attribute with different name etc.
Outliers Management	• Unwanted values which are not fiting in datasets.

11

Tidy Data

- "Tidy Data" paper by Hadley Wickham, PhD
- Formalize the way we describe the shape of data
- Gives us a goal when formatting our data
- "Standard way to organize data values within a dataset"

12

Principles of Tidy Data

- Columns represent separate variables
- Rows represent individual observations
- Observational units form tables

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

13

Converting to Tidy Data

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

→

	name	treatment	value
0	Daniel	treatment a	-
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27

- Better for reporting vs. better for analysis
- Tidy data makes it easier to fix common data problems

14

Melting

- The data problem we are trying to fix:
  - Columns containing values, instead of variables
- Solution: `pd.melt()`

```
pd.melt(frame=df, id_vars='name',
        value_vars=['treatment a', 'treatment b'])
```

```
     name     variable  value
0  Daniel  treatment a      _
1    John  treatment a     12
2    Jane  treatment a     24
3  Daniel  treatment b     42
4    John  treatment b     31
5    Jane  treatment b     27
```

15

Melting

```
pd.melt(frame=df, id_vars='name',
value_vars=['treatment a', 'treatment b'],
var_name='treatment', value_name='result')
```

```
     name    treatment  result
0  Daniel  treatment a       _
1    John  treatment a      12
2    Jane  treatment a      24
3  Daniel  treatment b      42
4    John  treatment b      31
5    Jane  treatment b      27
```

16

Pivot – Un-Melting Data

- Opposite of melting
- In melting, we turned columns into rows
- Pivoting: turn unique values into separate columns
- Analysis-friendly shape to reporting-friendly shape
- Violates tidy data principle: rows contain observations
  - Multiple variables stored in the same column

17

Pivot – Un-Melting Data

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4

element	tmax	tmin
date		
2010-01-30	27.8	14.5
2010-02-02	27.3	14.4

18

pivot()

```
weather_tidy = weather.pivot(index='date',
                             columns='element',
                             values='value')
print(weather_tidy)
```

```
element     tmax tmin
date
2010-01-30  27.8 14.5
2010-02-02  27.3 14.4
```

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4
4	2010-02-02	tmin	16.4

19

Using pivot() when you have duplicate Entries

```
import numpy as np
weather2_tidy = weather.pivot(values='value',
                              index='date',
                              columns='element')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-2962bb23f5a3> in <module>()
      1 weather2_tidy = weather2.pivot(values='value',
      2                                index='date',
----> 3                                columns='element')
ValueError: Index contains duplicate entries, cannot reshape
```

20

pivot_table()

- Has a parameter that specifies how to deal with duplicate values
- Example: Can aggregate the duplicate values by taking their average

```
weather2_tidy = weather.pivot_table(values='value',
                                    index='date',
                                    columns='element',
                                    aggfunc=np.mean)
```

```
element     tmax tmin
date
2010-01-30  27.8 14.5
2010-02-02  27.3 15.4
```

21

Combining Data

- Data may not always come in 1 huge file
  - 5 million row dataset may be broken into 5 separate datasets
  - Easier to store and share
  - May have new data for each day
- Important to be able to combine then clean, or vice versa

22

Concatenation

```
concatenated = pd.concat([weather_p1, weather_p2])
print(concatenated)
```

```
        date    element  value
0  2010-01-30  tmax      27.8
1  2010-01-30  tmin      14.5
0  2010-02-02  tmax      27.3
1  2010-02-02  tmin      14.4
```

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5

	date	element	value
0	2010-02-02	tmax	27.3
1	2010-02-02	tmin	14.4

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
0	2010-02-02	tmax	27.3
1	2010-02-02	tmin	14.4

23

Concatenation

```
concatenated = concatenated.loc[0, :]
```

```
        date    element  value
0  2010-01-30  tmax      27.8
0  2010-02-02  tmax      27.3
```

```
pd.concat([weather_p1, weather_p2], ignore_index=True)
```

```
        date    element  value
0  2010-01-30  tmax      27.8
1  2010-01-30  tmin      14.5
2  2010-02-02  tmax      27.3
3  2010-02-02  tmin      14.4
```

24

Combining Data

- Concatenation is not the only way data can be combined

	state	population_2016
0	California	39250017
1	Texas	27862596
2	Florida	20612439
3	New York	19745289

	name	ANSI
0	California	CA
1	Florida	FL
2	New York	NY
3	Texas	TX

25

Merging Data

- Similar to joining tables in SQL
- Combine disparate datasets based on common columns

	state	population_2016
0	California	39250017
1	Texas	27862596
2	Florida	20612439
3	New York	19745289

	name	ANSI
0	California	CA
1	Florida	FL
2	New York	NY
3	Texas	TX

26

Merging

```
pd.merge(left=state_populations, right=state_codes,
         on=None, left_on='state', right_on='name')
```

```
      state   population_2016       name  ANSI
0  California         39250017  California    CA
1      Texas          27862596       Texas    TX
2    Florida          20612439     Florida    FL
3   New York          19745289    New York    NY
```

	name	ANSI	state	City
0	California	CA	California	San Diego
1	California	CA	California	Sacramento
2	New York	NY	New York	New York City
3	New York	NY	New York	Albany

We need to drop the duplicate columns.

27

Datatypes

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

```
print(df.dtypes)
```

```
name           object
sex            object
treatment a    object
treatment b     int64
dtype: object
```

- There may be times we want to convert from one type to another
  - Numeric columns can be strings, or vice versa

28

Converting Data Types

```
df['treatment b'] = df['treatment b'].astype(str)
df['sex'] = df['sex'].astype('category')
df.dtypes
```

```
name              object
sex             category
treatment a       object
treatment b       object
dtype: object
```

29

Checking and Converting Data Types

```
df.dtypes
```

```
df['column'] = pd.to_numeric(df['column'])
```

```
df['column'] = df['column'].astype(str)
```

30

Categorical Data

- Converting categorical data to 'category' dtype:
  - Can make the DataFrame smaller in memory
  - Can make them be utilized by other Python libraries for analysis

31

Cleaning Data

- Numeric data loaded as a string

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

32

Cleaning Bad Data

```
df['treatment a'] = pd.to_numeric(df['treatment a'],
                                   errors='coerce')
df.dtypes
```

```
name             object
sex            category
treatment a     float64
treatment b      object
dtype: object
```

Coerce sets invalid parsing to NaN.

33

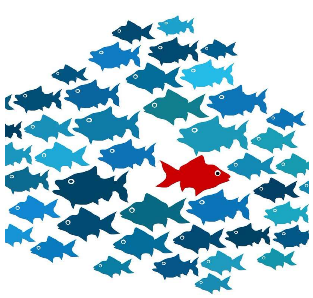
Data Quality Check

```
def cleaning_function(row_data):
        # data cleaning steps
        return ...

df.apply(cleaning_function, axis=1)

assert (df.column_data > 0).all()
```

34

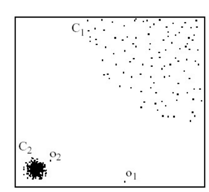


Outlier Detection

35

What is an outlier?

- Observations inconsistent with rest of the dataset – Global Outlier
- Special outliers – Local Outlier
  - Observations inconsistent with their neighborhoods
  - A local instability or discontinuity



36

Motivation for Outlier Analysis

- Fraud Detection (Credit card, telecommunications, criminal activity in e-Commerce)
- Customized Marketing (high/low income buying habits)
- Medical Treatments (unusual responses to various drugs)
- Analysis of performance statistics (professional athletes)
- Weather Prediction
- Financial Applications (loan approval, stock tracking)

"One persons noise could be another person's signal."

37

Causes of Outliers

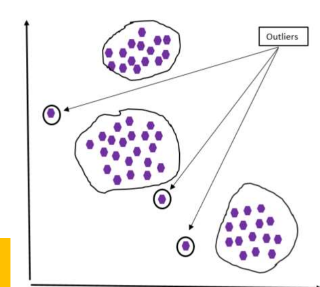
- Poor data quality / contamination
- Low quality measurements, malfunctioning equipment, manual error
- Correct but exceptional data

38

Outlier Detection Approaches

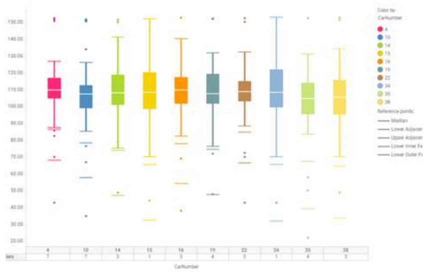
- Objective:
  - Define what data can be considered as inconsistent in a given data set
    - Statistical-Based Outlier Detection
    - Deviation-Based Outlier Detection
    - Distance-Based Outlier Detection
  - Find an efficient method to mine the outliers

39



Based on position in the Feature Space (Distance-based)

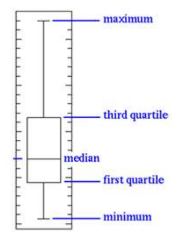
40



Box Plot

41

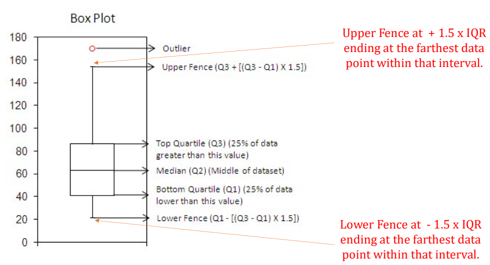
Box Plots and Outlier Detection



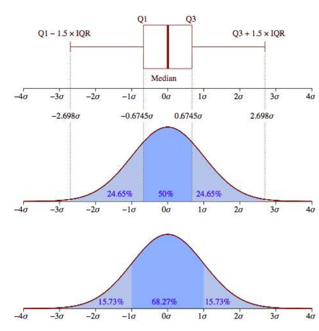
- Box plots have box from LQ to UQ, with median marked.
- They portray a five-number graphical summary of the data Minimum, LQ, Median, UQ, Maximum.
- Helps us to get an idea on the data distribution
- Helps us to identify the outliers easily
- 25% of the population is below first quartile,
- 75% of the population is below third quartile
- If the box is pushed to one side and some values are far away from the box then it's a clear indication of outliers.

42

Box Plots Drawn By Most Software Packages (by-default, Pandas, Seaborn etc.)



43



Based on Deviation from the Mean of the Distribution

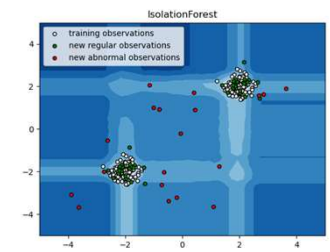
44

Isolation Forest – Sklearn Outlier Detection

- The ensemble.IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.
- Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.
- This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.
- Random partitioning **produces noticeably shorter paths for anomalies.**
- Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.
- The implementation of ensemble.IsolationForest is based on an ensemble of tree.ExtraTreeRegressor.

45

Isolation Forest



One efficient way of performing outlier detection in high-dimensional datasets is to use random forests.

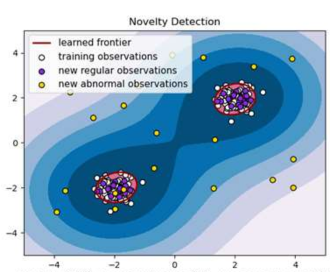
46

Novelty Detection – Sklearn Outlier Detection

- Consider a data set of **n** observations from the same distribution described by **p** features.
- Consider now that we add one more observation to that data set. Is the new observation so different from the others that we can doubt it is regular? (i.e. does it come from the same distribution?) Or on the contrary, is it so similar to the other that we cannot distinguish it from the original observations?
- This is the question addressed by the novelty detection tools and methods.
- In general, it is about to learn a rough, close frontier delimiting the contour of the initial observations distribution, plotted in embedding p-dimensional space.
- Then, if further observations lay within the frontier-delimited subspace, they are considered as coming from the same population than the initial observations.
- Otherwise, if they lay outside the frontier, we can say that they are abnormal with a given confidence in our assessment.
- The One-Class SVM has been introduced by Schölkopf et al. for that purpose and implemented in the Support Vector Machines module in the svm.OneClassSVM object.

47

Novelty Detection with OneClassSVM



48

Handling Missing Values

49

Records with Missing Entries

- **Complete-Case Analysis**
- Drops any observation with any missing value.
- Pros: Results will be well-behaved, simplest, usually software default.
- Cons: Drops some collected data, loses "information" and precision.

- **Available-Case Analysis**
- Drops no observations and calculates results based on available data.
- Pros: Uses all data available.
- Cons: Can get "not well-behaved results," i.e. invalid covariance matrices

50

Missing Values

- There are two types of missing values in every dataset:

1. **Visible Errors**: blank cells, special symbols like **-**, **?**, **NA** (Not Available), **NaN** (Not a Number), **None** type etc.
2. **Obscure Errors**: non-corrupt but **invalid values**. For example, a negative salary or a number for a name.

51

Operating on Null Values

- Pandas treats None and NaN as essentially interchangeable for indicating missing or null values.
- To facilitate this convention, there are several useful methods for detecting, removing, and replacing null values in Pandas data structures.

- `isnull()` : Generate a boolean mask indicating missing values
- `notnull()` : Opposite of `isnull()`
- `dropna()` : Return a filtered version of the data
- `fillna()` : Return a copy of the data with missing values filled or imputed

52

Detecting Null Values

- Pandas data structures have two useful methods for detecting null data: isnull() and notnull().
- Either one will return a Boolean mask over the data. For example:

```
data = pd.Series([1, np.nan, 'hello', None])
```

```
data.isnull()
```

```
0    False
1     True
2    False
3     True
dtype: bool
```

53

Detecting Null Values

- Boolean masks can be used directly as a Series or DataFrame index:

```
data[data.notnull()]
```

```
0        1
2    hello
dtype: object
```

- The isnull() and notnull() methods produce similar Boolean results for DataFrames.

54

Dropping Null Values

- In addition to the masking used before, there are the convenience methods, **dropna()** (which removes NA values) and **fillna()** (which fills in NA values).
- For a Series, the result is straightforward:

```
data.dropna()
```

```
0        1
2    hello
dtype: object
```

55

Dropping Null Values

- For a DataFrame, there are more options.
  Consider the following DataFrame:

```
df = pd.DataFrame([[1,      np.nan, 2],
                   [2,      3,      5],
                   [np.nan, 4,      6]])
df
```

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

56

Dropping Null Values

- We cannot drop single values from a DataFrame; we can only drop full rows or full columns.
- Depending on the application, you might want one or the other, so dropna() gives a number of options for a DataFrame.
- By default, dropna() will drop all rows in which any null value is present:

```
df.dropna()
```

	0	1	2
1	2.0	3.0	5

57

Dropping Null Values

- Alternatively, you can drop NA values along a different axis; axis=1 drops all columns containing a null value:

```
df.dropna(axis='columns')
```

	2
0	2
1	5
2	6

58

Dropping Null Values

- But this drops some good data as well; you might rather be interested in dropping rows or columns with all NA values, or a majority of NA values. This can be specified through the how or thresh parameters, which allow fine control of the number of nulls to allow through.
- The default is how='any', such that any row or column (depending on the axis keyword) containing a null value will be dropped.
- You can also specify how='all', which will only drop rows/columns that are all null values:

```
df[3] = np.nan
df
```

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

```
df.dropna(axis='columns', how='all')
```

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

59

Dropping Null Values

- For finer-grained control, the thresh parameter lets you specify a minimum number of non-null values for the row/column to be kept:
- Here the first and last row have been dropped, because they contain only two non-null values.

```
df.dropna(axis='rows', thresh=3)
```

	0	1	2	3
1	2.0	3.0	5	NaN

60

Missing Data Imputation

61

Deductive Imputation

Uses logical relations to fill in missing values.

- Respondent mentions they were not the victim of a crime, so the column for "victim of a crime" contains a 0. However, an "NA" exists in the column for "victim of a violent crime." Because the respondent mentioned they were not the victim of a crime, we know that the respondent was not the victim of a violent crime.
- If someone has 2 children in year 1, NA children in year 2, and 2 children in year 3, we can probably impute that they have 2 children in year 2.

- Pros: Requires no "inference," true value can be assessed, valid method.
- Cons: Can be time consuming or requires specific coding

62

Mean/Median/Mode Imputation

For any "NA" value in a given column, mean imputation replaces "NA" with the mean of that column.

> Same for median and mode imputation.

Pros: Easy to implement and comprehend. Seems reasonable.

Cons: Significantly distorts histogram, underestimates variance, mean and median imputation will give very different results for asymmetric data.

63

Regression Imputation

- For any "NA" value in a given column, regression imputation replaces
- "NA" with a predicted value based on a regression line.

- i.e. Given observed demographic data, estimated income = $\boldsymbol{\beta_0 + \beta_1 * age + \beta_2 * sex}$
- Then use observed age and sex as predictors to impute missing income data.

- Pros: Easy to comprehend, seems logical, better than mean/median/mode imputation.

- Cons: Still distorts histogram and underestimates variance

64

Hot-Deck Imputation

- Divide sample units into classes (i.e. based on age and sex).
- For any "NA" value in a given class, randomly select the value of one of the observed values in that class and impute that value for the missing value.
- i.e. Among 18-34 year old women, there are 20 observed values and 3 missing values. For each missing value, pick one observed value at random and fill in the missing value with that observed value. You will select three observed values with replacement.
- Pros: You're using existing data.
- Cons: If columns are imputed separately, multivariate relationships are not preserved.

65

Filling Null Values

- Sometimes rather than dropping NA values, you'd rather replace them with a valid value.
- This value might be a single number like zero, or it might be some sort of imputation or interpolation from the good values.
- You could do this in-place using the isnull() method as a *mask*, but because it is such a common operation Pandas provides the fillna() method, which returns a copy of the array with the null values replaced.

66

Filling Null Values

- Consider the following Series:

```
data = pd.Series([1, np.nan, 2, None, 3], index=list('abcde'))
data
```

```
a    1.0
b    NaN
c    2.0
d    NaN
e    3.0
dtype: float64
```

67

Filling Null Values

- We can fill NA entries with a single value, such as zero:

```
data.fillna(0)
```

```
a    1.0
b    0.0
c    2.0
d    0.0
e    3.0
dtype: float64
```

68

Filling Null Values

- We can specify a forward-fill to propagate the previous value forward:

```
# forward-fill
data.fillna(method='ffill')
```

```
a    1.0
b    1.0
c    2.0
d    2.0
e    3.0
dtype: float64
```

69

Filling Null Values

- Or we can specify a back-fill to propagate the next values backward:

```
# back-fill
data.fillna(method='bfill')
```

```
a    1.0
b    2.0
c    2.0
d    3.0
e    3.0
dtype: float64
```

70

Filling Null Values

- For DataFrames, the options are similar, but we can also specify an axis along which the fills take place.
- Notice that if a previous value is not available during a forward fill, the NA value remains.

```
df
```

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

```
df.fillna(method='ffill', axis=1)
```

	0	1	2	3
0	1.0	1.0	2.0	2.0
1	2.0	3.0	5.0	5.0
2	NaN	4.0	6.0	6.0

71

Read Employee Dataset

```
# Importing libraries
import pandas as pd
import numpy as np

# Read csv file into a pandas dataframe
df = pd.read_csv("employees.csv")
```

```
df.head()
```

	First Name	Gender	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	97308.0	6.945	True	Marketing
1	Thomas	Male	61933.0	NaN	True	NaN
2	Maria	Female	130590.0	11.858	False	Finance
3	Jerry	Male	NaN	9.340	True	Finance
4	Larry	Male	101004.0	1.389	True	Client Services

72

Replace method

The replace() method is a more generic form of the fillna method.

```
# Will replace NaN value in Salary with value 0
df['Salary'].replace(to_replace = np.nan, value = 0)

0       97308.0
1       61933.0
2      130590.0
3           0.0
4      101004.0
         ...
995    132483.0
996     42392.0
997     96914.0
998     60500.0
999    129949.0
Name: Salary, Length: 1000, dtype: float64
```

73

Interpolate method

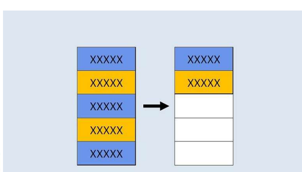
interpolate() function is used to fill NaN values using various interpolation techniques.

```
df['Salary'].interpolate(method='linear', direction = 'forward')

0       97308.0
1       61933.0
2      130590.0
3      115797.0
4      101004.0
         ...
995    132483.0
996     42392.0
997     96914.0
998     60500.0
999    129949.0
Name: Salary, Length: 1000, dtype: float64
```

74

Remove Duplicates



75

Removing Duplicates in Pandas

Remove Duplicate Records

Return DataFrame with duplicate rows removed.

DataFrame.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)

> Return DataFrame with duplicate rows removed.
>
> Considering certain columns is optional. Indexes, including time indexes are ignored.
>
> Parameters subsetcolumn label or sequence of labels, optional

- Generate Boolean Mask of the Duplicate Entries

pandas.DataFrame.duplicated

DataFrame.duplicated(subset=None, keep='first')

Returns boolean Series denoting duplicate rows.

76

Feature Engineering

77

Feature Engineering

Missing Data Imputation

Feature-engine includes widely used techniques for missing data imputation, such as mean and median imputation, frequent category imputation, random sample imputation, and adding a missing indicator. Feature-engine also includes alternative techniques, like end of tail imputation.

Categorical Variable Encoding

Feature-engine comprises the most extensive library for categorical variable encoding to date, including one hot encoding, ordinal numbering, count or frequency encoding, as well as, more powerful techniques like target encoding, and weight of evidence. Feature-engine also handles rare labels automatically.

Discretisation

Feature-engine comes with the most popular methods of variable discretisation: equal width and equal frequency discretisation. Feature-engine also includes a method which uses decision trees to automatically find the buckets for each variable.

78

Feature Engineering

Outlier Handling

Feature-engine allows you to cap variables at specific arbitrary values, or it automatically determines the capping values for you, using the inter-quantal range proximity rule.

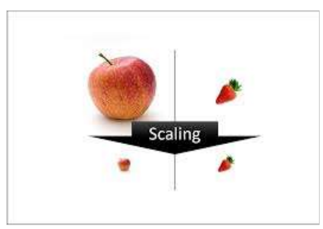
Variable Transformation

Feature-engine brings along the most used variable transformation mathematical functions, including logarithmic, exponential, reciprocal and Box-Cox transformations.

Engineer Individual Feature Groups

Feature-engine allows you to select a subset of variables for each engineering step. All engineering steps can be integrated into a machine learning pipeline to smooth model deployment.

79



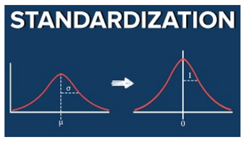
Feature Scaling

80

Standardize

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

$\mu =$ Mean
$\sigma =$ Standard Deviation

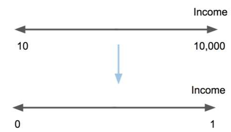


Standardize

81

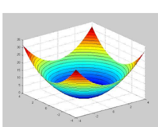
Normalize

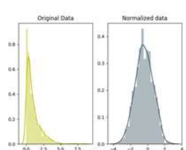
$$X_{normalized} = \frac{(X - X_{minimum})}{(X_{maximum} - X_{minimum})}$$

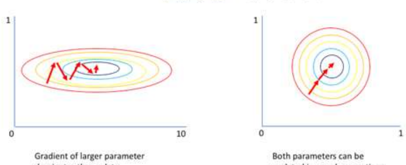


82

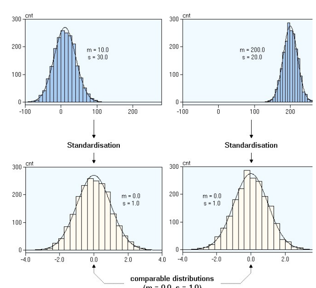
Normalization







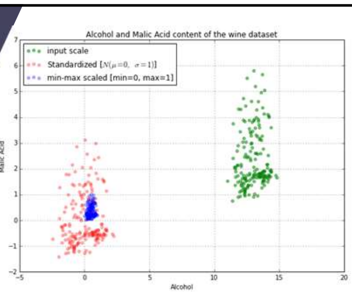
83



Comparable Distributions after Standardization

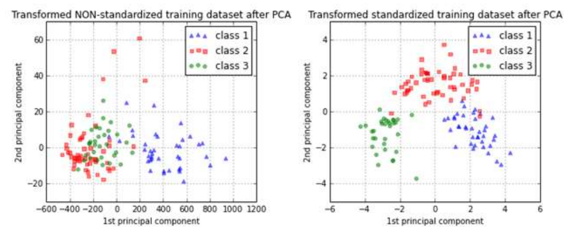
84

How does the data points move in the feature space due to such transformations?



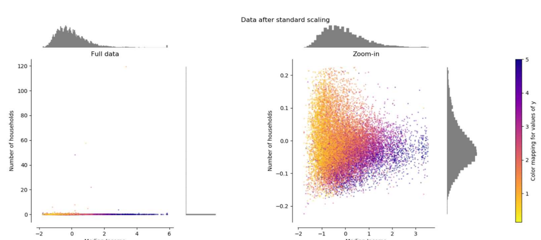
85

Standardization – Effect on ML Algorithms



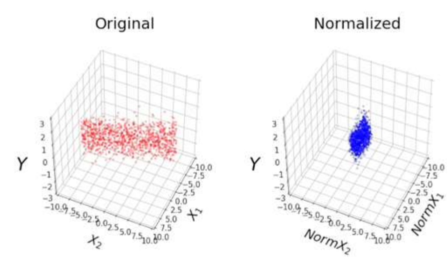
86

Standardization – Effect on ML Algorithms



87

Feature Normalization



88

Feature Scaling in Sklearn

- StandardScalar
  - Standarlize the data by: (x - mean(x)) / std(x)
  - StandardScalar assumes that the dataset is normal distributed. So if you're dataset is skewed, you may want to handle skewness first.(We talked about how to handle skewed data in the previous part.)
- MinMaxScalar
  - MinMaxScalar scales the data to (0, 1) range by: (x - min(x)) / (max(x) - min(x))
  - Pretty useful in, for example, image processing or handling non-normal distributed data.
  - However, MinMaxScalar is sensitive to outliers. So if your model is also sensitive to outliers, use RobustScalar instead.

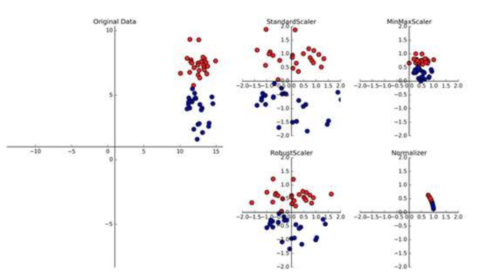
89

Feature Scaling in Sklearn

- RobustScaler
  - (x - Q1(x)) / (Q3(x) - Q1(x))
  - RobustScaler use quartiles instead of using minimum and maximum value. And because of this, RobustScaler is robust to outliers.
  - Some models are sensitive to outliers. For instance,linear model like Lasso (L1 regularization). In this case, use RobustScaler instead.
- Normalizer
  - Normalizer scales individual samples and make samples have unit norm.
  - (In other words, Normalizer do normalization on each row, while the other scalers introduced above scale on each column.)
  - It is commonly used when doing text classification or some other vector space tasks.

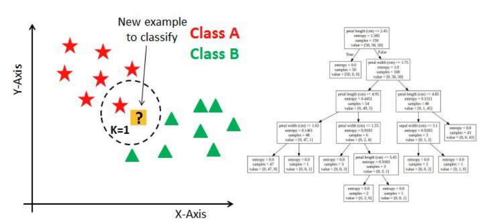
90

Feature Scaling - Comparison



91

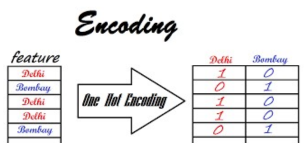
Feature Scaling – Model Performance



K-nearest Neighbour (KNN) vs. Decision Tree Classifier

92

Feature Encoding



93

Categorical Variables

- One-Hot-Encoding
- Dummy Variables
- Encoding Categorical Variables with continuous Dense Vectors

94

One-Hot Encoding

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

95

One-Hot Encoding

Scikit-Learn OneHotEncoder

```
encoder = OneHotEncoder()
df_fruit_encoded = pd.DataFrame(encoder.fit_transform(df[['fruit']]).todense(),
                                columns=encoder.get_feature_names())
df_fruit_encoded
```

	x0_apple	x0_banana	x0_orange
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	1.0	0.0
3	0.0	0.0	1.0
4	0.0	1.0	0.0
5	1.0	0.0	0.0

Original DataFrame has three categories for *'fruit'* – apple, banana & orange.

Pandas get_dummies

```
pd.get_dummies(df['size'])
```

	large	medium	small
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	0	1

Original DataFrame has three categories for *'size'* – large, medium & small.

96

One-hot Encoding V/s Dummy Encoding

1. One-hot encoding and dummy encoding are different ways of encoding the categorical data.

2. If we have a categorical variable that has **k** different categories then one-hot encoding will convert it into **k** variables whereas dummy encoding will convert it into **k-1** variables.

Multicollinearity occurs when two or more independent variables (features) in the dataset are correlated with each other.

97

One-Hot vs. Dummy Encoding

id	X
1	a
2	c
3	a
4	b
5	a
6	c
7	c
8	b

One-Hot Encoding

id	X = a	X = b	X = c
1	1	0	0
2	0	0	1
3	1	0	0
4	0	1	0
5	1	0	0
6	0	0	1
7	0	0	1
8	0	1	0

Dummy Encoding

id	X = a	X = b
1	1	0
2	0	0
3	1	0
4	0	1
5	1	0
6	0	0
7	0	0
8	0	1

98

sklearn.preprocessing.OneHotEncoder

class `sklearn.preprocessing.OneHotEncoder`(**, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error'*) [source]

drop : *{'first', 'if_binary'} or a array-like of shape (n_features,), default=None*

Specifies a methodology to use to drop one of the categories per feature. This is useful in situations where perfectly collinear features cause problems, such as when feeding the resulting data into a neural network or an unregularized regression.

However, dropping one category breaks the symmetry of the original representation and can therefore induce a bias in downstream models, for instance for penalized linear classification or regression models.

- None : retain all features (the default).
- 'first' : drop the first category in each feature. If only one category is present, the feature will be dropped entirely.
- 'if_binary' : drop the first category in each feature with two categories. Features with 1 or more than 2 categories are left intact.
- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

99

pandas.get_dummies

`pandas.get_dummies`(*data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None*)

Convert categorical variable into dummy/indicator variables.

Parameters:

data : *array-like, Series, or DataFrame*
Data of which to get dummy indicators.

prefix : *str, list of str, or dict of str, default None*
String to append DataFrame column names. Pass a list with length equal to the number of columns when calling get_dummies on a DataFrame. Alternatively, *prefix* can be a dictionary mapping column names to prefixes.

prefix_sep : *str, default '_'*
If appending prefix, separator/delimiter to use. Or pass a list or dictionary as with *prefix*.

dummy_na : *bool, default False*
Add a column to indicate NaNs, if False NaNs are ignored.

columns : *list-like, default None*
Column names in the DataFrame to be encoded. If *columns* is None then all the columns with *object* or *category* dtype will be converted.

sparse : *bool, default False*
Whether the dummy-encoded columns should be backed by a `SparseArray` (True) or a regular NumPy array (False).

drop_first : *bool, default False*
Whether to get k-1 dummies out of k categorical levels by removing the first level.

dtype : *dtype, default np.uint8*
Data type for new columns. Only a single dtype is allowed.
New in version 0.23.0.

100

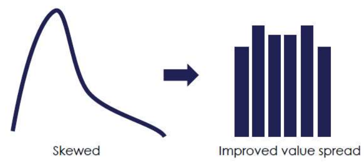
Feature-engine: A Feature Engineering for Machine Learning Library in Python

- **Feature-engine** is a Python library with multiple transformers to engineer features for use in machine learning models. Feature-engine preserves Scikit-learn functionality with fit() and transform() methods to learn parameters from and then transform data.
- Feature-engine includes transformers for:
- Missing value imputation
- Categorical variable encoding
- Outlier capping
- Discretisation
- Numerical variable transformation
- Feature-engine allows you to select the variables to engineer within each transformer. This way, different engineering procedures can be easily applied to different feature subsets.
- Feature-engine's transformers can be assembled within the Scikit-learn pipeline, therefore making it possible to save and deploy one single object (.pkl) with the entire machine learning pipeline.

Web-Link: https://feature-engine.readthedocs.io/en/latest/index.html

101

Discretization

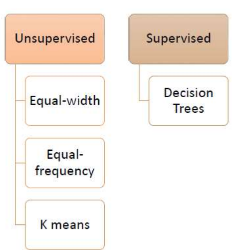


Discretisation is the process of transforming continuous variables into discrete variables by creating a set of contiguous intervals that span the range of the variable's values.

Discretisation is also called **binning**, where bin is an alternative name for interval.

102

Discretization Approaches



Feature-Engine – Python Library for Feature Engineering and ML

https://feature-engine.readthedocs.io/en/latest/discretisers/EqualFrequencyDiscretiser.html

https://feature-engine.readthedocs.io/en/latest/discretisers/EqualWidthDiscretiser.html

https://feature-engine.readthedocs.io/en/latest/discretisers/DecisionTreeDiscretiser.html

103

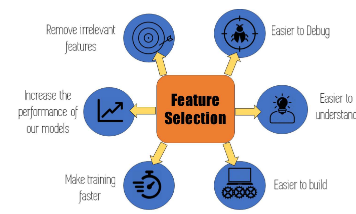
Feature Selection

104

Feature Selection

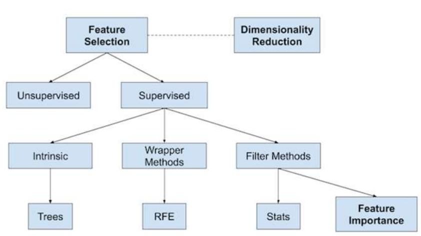
105

Feature Selection



106

Overview of Feature Selection Techniques



107

Intrinsic

- There are some machine learning algorithms that perform feature selection automatically as part of learning the model. We might refer to these techniques as intrinsic feature selection methods.
- Some models contain built-in feature selection, meaning that the model will only include predictors that help maximize accuracy. In these cases, the model can pick and choose which representation of the data is best.
- This includes algorithms such as penalized regression models like Lasso and decision trees, including ensembles of decision trees like random forest.
- Some models are naturally resistant to non-informative predictors. Tree- and rule-based models, MARS and the lasso, for example, intrinsically conduct feature selection.

108

Wrapper Methods

- Wrapper feature selection methods create many models with different subsets of input features and select those features that result in the best performing model according to a performance metric.
- These methods are unconcerned with the variable types, although they can be computationally expensive. RFE is a good example of a wrapper feature selection method.
- Wrapper methods evaluate multiple models using procedures that add and/or remove predictors to find the optimal combination that maximizes model performance.

109

Filter Methods

- Filter feature selection methods use statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to choose (filter) those input variables that will be used in the model.
- Filter methods evaluate the relevance of the predictors outside of the predictive models and subsequently model only the predictors that pass some criterion.
- **Numerical Output**: Regression predictive modeling problem.
- **Categorical Output**: Classification predictive modeling problem.

110

Filter Methods

- **Numerical Input, Numerical Output**
- This is a regression predictive modeling problem with numerical input variables.
- The most common techniques are to use a correlation coefficient, such as Pearson's for a linear correlation, or rank-based methods for a nonlinear correlation.
- Pearson's correlation coefficient (linear).
- Spearman's rank coefficient (nonlinear)

111

Filter Methods

- **Numerical Input, Categorical Output**
- This is a classification predictive modeling problem with numerical input variables.
- This might be the most common example of a classification problem,
- Again, the most common techniques are correlation based, although in this case, they must take the categorical target into account.
- ANOVA correlation coefficient (linear).
- Kendall's rank coefficient (nonlinear).
- Kendall does assume that the categorical variable is ordinal.

112

Filter Methods

- **Categorical Input, Categorical Output**
- This is a classification predictive modeling problem with categorical input variables.
- The most common correlation measure for categorical data is the chi-squared test. You can also use mutual information (information gain) from the field of information theory.
- Chi-Squared test (contingency tables).
- Mutual Information.
- In fact, mutual information is a powerful method that may prove useful for both categorical and numerical data, e.g. it is agnostic to the data types.

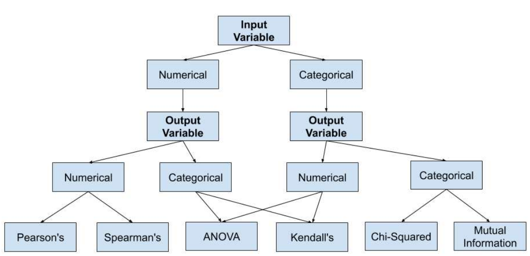
113

Feature Selection Techniques

- **Feature Selection**: Select a subset of input features from the dataset.
  - **Unsupervised**: Do not use the target variable (e.g. remove redundant variables).
    - Correlation
  - **Supervised**: Use the target variable (e.g. remove irrelevant variables).
    - **Wrapper**: Search for well-performing subsets of features. (RFE)
    - **Filter**: Select subsets of features based on their relationship with the target.
      - Statistical Methods; Feature Importance Methods
    - **Intrinsic**: Algorithms that perform automatic feature selection during training.
      - Decision Trees
- **Dimensionality Reduction**: Project input data into a lower-dimensional feature space.

114

How to choose a Feature Selection Method?



115

Feature Selection in Sklearn

- The classes in the sklearn.feature_selection module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.
- https://scikit-learn.org/stable/modules/feature_selection.html

116

Go to the Coding Demos…

117

118

To be continued in the next session…..

119