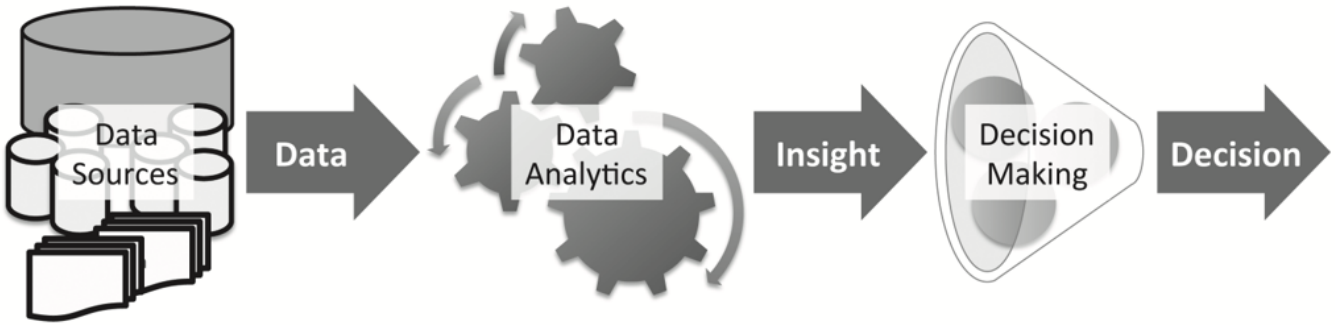
Data Science

Yogesh Kulkarni

Introduction to Data Analytics

What is Data Analysis?

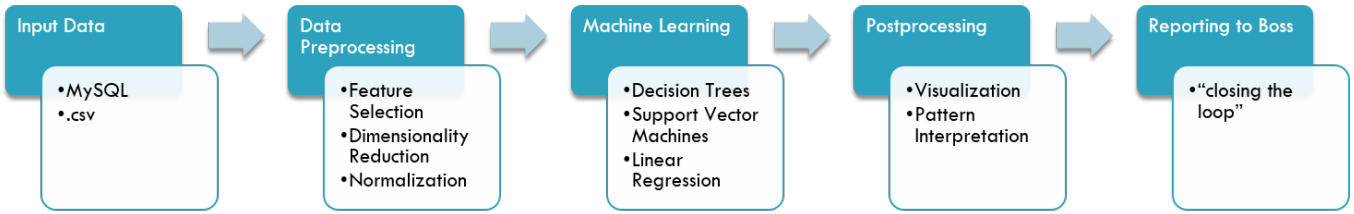
- Analyze data to extract meaning from it
- Data Mining and Business Analytics deal with collecting and analyzing data for better decision making
- Machine Learning is one of the techniques, mainly for prediction, classification, etc.
- Moving from data to insights to decisions.



Applications

- Businesses collect lots of data
  - Purchase information
  - Web site browsing habits
  - Social network data
- Goals
  - Customer profiling,
  - Targeted marketing,
  - Fraud detection

Process


Input Data
•MySQL
•.csv
Data Preprocessing
•Feature Selection
•Dimensionality Reduction
•Normalization
Machine Learning
•Decision Trees
•Support Vector Machines
•Linear Regression
Postprocessing
•Visualization
•Pattern Interpretation
Reporting to Boss
•"closing the loop"

Input Data

- Available in data in variety of formats:
  - Flat files (.csv or .txt)
  - Spreadsheets (Excel .xls tougher to deal with)
  - Relational tables (MySQL)
  - Text, data on web page (scraping necessary)
- Big Data / Data Warehouse
- Data spread out over multiple locations

Preprocessing

- To transform raw input data into an appropriate format for subsequent analysis:
  - Fusing data from multiple sources
  - Cleaning data to remove noise
  - Duplicate observations
- Selecting records and features that are relevant to the data mining task at hand

Machine Learning

- Linear Regression
- Support Vector Machines
- Decision Trees
- Clustering

Postprocessing

- Hypothesis testing to eliminate spurious data mining results
- Statistical significant tests, confidence intervals,
- Visualization

Challenges

- Scalability
  - Gigabytes, terabytes, petabytes, exabytes of data
  - Storage capacity, streaming?
  - Limits of python libraries
- Heterogeneous and Complex Data
- High Dimensionality
  - Datasets with hundreds or thousands of attributes
  - Many variables are collected; few turn out to be useful
- Need to reduce data
  - To avoid redundancy
  - For effieciency: time, resources

What is Data Reduction?

- Virtually all data analysis focuses on data reduction
- Data reduction lets us see critical features or patterns in the data
- Data reduction comes in the form of:
  - Descriptive statistics
  - Measures of association
  - Graphical visualizations

Summary

Traditional Data Analysis

- Laborious process
- Generation and evaluation of thousands of hypotheses
- Usually on relatively smaller datasets

Current Data Analysis

- Datasets analyzed typically not result of a carefully designed experiment
- Datasets of size TB
- Opportunistic data reduction

Introduction to Data Science

What is Data Science ?

*M*ulti-disciplinary field that brings together concepts from computer science, statistics/machine learning, and data analysis to understand and extract insights from the ever-increasing amounts of data.

Paradigms of Data Research

- Hypothesis-Driven: Given a problem, what kind of data do we need to help solve it?
- Data-Driven: Given some data, what interesting problems can be solved with it?

What is core of Data Science ?

- What can we learn from this data?
- What actions can we take once we find whatever it is we are looking for?

What is Statistics?

- Statistics is the science of learning from data.
- The goal of statistics is to summarize data in a way that allows for easy descriptions or inferences of the data.

Data

- Data is by Measurement
- Measurement: Process of assigning numbers to types
- Each Data has Type and Value

Data

id	Home Owner	Marital Status	Annual Income	Defaulted Barrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Vocabulary

- Column-Type: “attribute”, “feature”, “field”, “dimension”, “variable”
- Row-Value: “instance”, “record”, “observation”

Data

- Data Type
- Data Value
- Distinctions:
  - Same type - different values. Example: height can be measured in feet or meters
  - Different types same values. Example: Attribute values for ID and age

Types of Data

- **Categorical types (Qualitative):** Nominal and Ordinal
  - **Nominal** (numbers do not give sense of order/rank): eye color, zip codes
  - **Ordinal** (numbers give sense of order/rank): rankings, size in small-medium-large
- **Numeric types (Quantitative):** Interval and Ratio
  - **Discrete:** A discrete attribute has a finite or countably infinite set of values. **Binary attributes** are a special case of discrete attributes.
  - **Continuous:** A continuous attribute is one whose values are real numbers.
  - **Interval**: calendar dates
  - **Ratio**: counts, time

NOIR: **N**o **O**il **I**n **R**ivers

Ordinal

- To show relative rankings
- Order matters, but not diff
- $Diff\,(7,5) \neq Diff\,(5,3)$
- "First is first, however close the second is!!"
- Examples: class rank, levels of wellness.

Interval

- Diff equal if measured between two equivalent variables
- $Diff(100, 90) == Diff(90, 80)$
- Same amount of heat needed to take from 90 to 100 or 80 to 90
- Examples: test scores and temperature

Ratio

- Weight of 8 grams is twice the weight of 4 grams
- Clear definition of 0.0; none of a variable at 0.0
- Example: height, weight, pulse and BP

Ordered Data

- Temporal: time based, e.g. retail transaction
- Sequential: e.g. DNA sequence (ATGC possible letters)
- Time Series: Series of measurements taken over time. e.g.: financial stock price data
- Spatial Data: e.g. geographical locations

Measures of Similarity and Dissimilarity

- **Proximity:** either similarity or dissimilarity
  - **Similarity:** a numerical measure of likeliness.
  - **Dissimilarity:** a numerical measure of unlikeliness
- **Transformations:** re-parametrize proximity to, say, [0, 1].

Similarity and Dissimilarity between Simple Attributes

- **Nominal:** simple equality test
- **Ordinal:** order should be taken into account.
- **Ratio:** simple absolute difference of their values.
- Distance is the measure of similarity-dissimilarity.
- Distance between two data items can either be 0 or 1 for congruence.

Euclidean distance

$$d(a,b) = \sqrt{\sum_{k=1}^{n}(a_k - b_k)^2}$$

Minkowski distances

$d(a,b) = \left(\sum_{k=1}^{n} |a_k - b_k|^r\right)^{1/r}$

- **r = 1** City block (Manhattan, taxicab, L_1 norm) distance.
- **r = 2** Euclidean distance (L_2 norm).
- **r = ∞** Supreme (L_{max} or L_∞) distance. Here r is not put equal to ∞ but tends to or limit to ∞

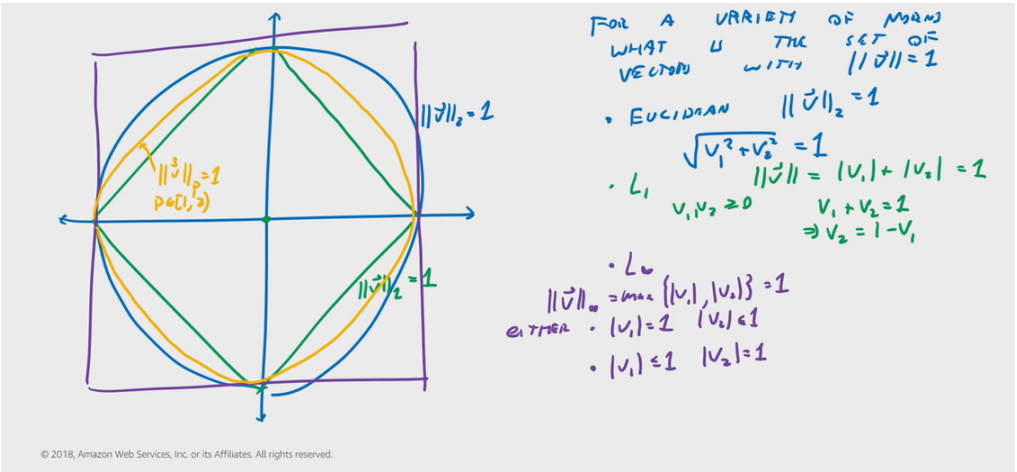
Considering this formula for finding norm (ie length, or tcan be same as length of diff between a and b):

- Putting a very large number, say, 1000, for $v = [0110]$, the norm would be $(0^{1000} + 1^{1000} + 10^{1000})^{1/1000}$.
- Here the last number becomes very big, but its 1000th root gives the same number as answer ie 10.
- So, this method biases towards large numbers.

Visualizing Norms

Vectors having:

- Euclidean distance Norm as 1, when plotted look like circle.
- Manhattan distance as Norm as 1, when plotted makes a inner Square.
- Minkowski distance with $r = \infty$ norm as 1, when plotted looks outer Square.
- Minkowski distance with smaller r norm as 1, when plotted looks somewhere between.



(Ref: Math for Machine Learning - Brent Werness, AWS)

Properties Euclidean distances

- **Positivity**
  - d(a,b) $\geq$ for all a and b,
  - d(a,b) $=$ 0 only if a $=$ b
- **Symmetry**
  d(a,b) $=$ d(b,a) for all a and b
- **Triangle Inequality**
  d(a,c) $\leq$ d(a,b) $+$ d(b,c) for all points a, b, and c

Metrics

Measures that satisfy all three properties are known as metrics.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

TABLE: (a) x and y coordinates, (b) Euclidean distance matrix

Metrics

Measures that satisfy all three properties are known as metrics.

L_1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

L_∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0
p4	5.0	3.0	2.0	0.0

TABLE: (a) L_1 distance matrix, (b) L_∞ distance matrix

Simple Matching Coefficient (SMC)

Have values between 0 and 1.
Let a and b be two objects that consist of n binary attributes
i.e., two binary vectors
Possibilities:

- f_{00} = the number of attributes where a is 0 and b is 0
- f_{01} = the number of attributes where a is 0 and b is 1
- f_{10} = the number of attributes where a is 1 and b is 0
- f_{11} = the number of attributes where a is 1 and b is 1

$$SMC = \frac{number\ of\ matching\ attribute\ values}{number\ of\ attributes} = \frac{f_{11}+f_{00}}{f_{01}+f_{10}+f_{11}+f_{00}}$$

Cosine Similarity

$$cos(a,b) = \frac{a.b}{\|a\|\|b\|} = \frac{\sum a_i.b_i}{\sqrt{\sum (a_i)^2}.\sqrt{\sum (b_i)^2}}$$

Introduction to Pandas

Pandas

pandas introduces two new data structures to Python - **Series** and **DataFrame**, both of which are built on top of NumPy.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.set_option('max_columns', 50)
```

Series

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call:
Here, data can be many different things:

- a Python `dict`
- an `ndarray`
- a scalar value (like 5)

```
s = Series(data, index=index)
```

- A Series is a one-dimensional object similar to an array, list, or column in a table.
- It will assign a labeled index to each item in the Series.
- By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.

```
# create a Series with an arbitrary list
s = pd.Series([7, 'Heisenberg', 3.14, -1789710578,     'Happy Eating!'])
s
```

Series

Output from Previous Slide

```
0                7
1       Heisenberg
2             3.14
3      -1789710578
4    Happy Eating!
dtype: object
```

Alternatively, you can specify an index to use when creating the Series.

```
s = pd.Series([7, 'Heisenberg', 3.14, -1789710578,
   'Happy Eating!'],
index=['A', 'Z', 'C', 'Y', 'E'])
s

A                 7
Z       Heisenberg
C             3.14
Y      -1789710578
E    Happy Eating!
dtype: object
```

Series

The Series constructor can convert a dictonary as well, using the keys of the dictionary as its index.

```
d = {'Chicago': 1000, 'New York': 1300, 'Portland': 900, 'San Francisco':
     1100,'Austin': 450, 'Boston': None}
cities = pd.Series(d)
cities
Out[4]:
Austin            450
Boston            NaN
Chicago          1000
New York         1300
Portland          900
San Francisco    1100
dtype: float64
```

Series

You can use the index to select specific items from the Series

```
cities['Chicago']
Out[5]:
1000.0
```

Series

```
cities[['Chicago', 'Portland', 'San Francisco']]
Out[6]:
Chicago          1000
Portland          900
San Francisco    1100
dtype: float64
```

Series

You can use ***boolean indexing*** for selection.

```
cities[cities < 1000]
Out[7]:
Austin      450
Portland    900
dtype: float64
```

That last one might be a little strange, so let's make it more clear - `cities < 1000` returns a Series of `True/False` values, which we then pass to our Series cities, returning the corresponding `True` items.

```
less_than_1000 = cities < 1000
print less_than_1000
print '\n'
print cities[less_than_1000]
Austin            True
Boston           False
Chicago          False
New York         False
Portland          True
San Francisco    False
dtype: bool

Austin      450
Portland    900
dtype: float64
```

You can also change the values in a Series on the fly.

```
# changing based on the index
print 'Old value:', cities['Chicago']
cities['Chicago'] = 1400
print 'New value:', cities['Chicago']
Old value: 1000.0
New value: 1400.0
```

Changing values using boolean logic

```
print cities[cities < 1000]
print '\n'
cities[cities < 1000] = 750
print cities[cities < 1000]
Austin      450
Portland    900
dtype: float64

Austin      750
Portland    750
dtype: float64
```

Working with Series

What if you aren't sure whether an item is in the Series? You can check using idiomatic Python.

```
print 'Seattle' in cities
print 'San Francisco' in cities
False
True
```

Mathematical operations can be done using scalars and functions.

```
# divide city values by 3
cities / 3
Out[12]:
Austin           250.000000
Boston                  NaN
Chicago          466.666667
New York         433.333333
Portland         250.000000
San Francisco    366.666667
dtype: float64
```

```
# square city values
np.square(cities)
Out[13]:
Austin            562500
Boston               NaN
Chicago          1960000
New York         1690000
Portland          562500
San Francisco    1210000
dtype: float64
```

You can add two Series together, which returns a union of the two Series with the addition occurring on the shared index values. Values on either Series that did not have a shared index will produce a NULL/NaN (not a number).

```
print cities[['Chicago', 'New York', 'Portland']]
print'\n'
print cities[['Austin', 'New York']]
print'\n'
print cities[['Chicago', 'New York', 'Portland']] + cities[['Austin', 'New
    York']]
```

```
Chicago     1400
New York    1300
Portland     750
dtype: float64

Austin       750
New York    1300
dtype: float64

Austin       NaN
Chicago      NaN
New York    2600
Portland     NaN
dtype: float64
```

Working with Series

NULL Checking

- Notice that because Austin, Chicago, and Portland were not found in both Series, they were returned with NULL/NaN values.
- NULL checking can be performed with `isnull()` and `notnull()`.

Return a boolean series indicating which values aren't NULL

```
cities.notnull()

Austin            True
Boston           False
Chicago           True
New York          True
Portland          True
San Francisco     True
dtype: bool
```

Using boolean logic to grab the NULL cities

```
print cities.isnull()
print '\n'
print cities[cities.isnull()]
Austin           False
Boston            True
Chicago          False
New York         False
Portland         False
San Francisco    False
dtype: bool

Boston   NaN
dtype: float64
```

Creating Dataframes

Creating a DataFrame by passing a numpy array, with a datetime index and labeled columns:

```
In [6]: dates = pd.date_range('20130101', periods=6)

In [7]: dates
Out[7]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [8]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

In [9]: df
Out[9]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Creating Dataframes

Creating a DataFrame by passing a dict of objects that can be converted to series-like.

```
In [10]: df2 = pd.DataFrame({ 'A' : 1.,
    ....:                     'B' : pd.Timestamp('20130102'),
    ....:                     'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
    ....:                     'D' : np.array([3] * 4,dtype='int32'),
    ....:                     'E' : pd.Categorical(["test","train","test","train"]),
    ....:                     'F' : 'foo' })
    ....:

In [11]: df2
Out[11]:
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

Viewing Data

See the top & bottom row s of the frame

```
In [14]: df.head()
Out[14]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

In [15]: df.tail(3)
Out[15]:
                   A         B         C         D
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Viewing Data

Display the index, columns, and the underlying numpy data

```
In [16]: df.index
Out[16]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [17]: df.columns
Out[17]: Index([u'A', u'B', u'C', u'D'], dtype='object')

In [18]: df.values
Out[18]:
array([[ 0.4691, -0.2829, -1.5091, -1.1356],
       [ 1.2121, -0.1732,  0.1192, -1.0442],
       [-0.8618, -2.1046, -0.4949,  1.0718],
       [ 0.7216, -0.7068, -1.0396,  0.2719],
       [-0.425 ,  0.567 ,  0.2762, -1.0874],
       [-0.6737,  0.1136, -1.4784,  0.525 ]])
```

Viewing Data

Describe show s a quick statistic summary of your data

```
In [19]: df.describe()
Out[19]:
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.073711 -0.431125 -0.687758 -0.233103
std    0.843157  0.922818  0.779887  0.973118
min   -0.861849 -2.104569 -1.509059 -1.135632
25%   -0.611510 -0.600794 -1.368714 -1.076610
50%    0.022070 -0.228039 -0.767252 -0.386188
75%    0.658444  0.041933 -0.034326  0.461706
max    1.212112  0.567020  0.276232  1.071804
```

Viewing Data

Sorting by an axis

```
In [21]: df.sort_index(axis=1, ascending=False)
Out[21]:
                   D         C         B         A
2013-01-01 -1.135632 -1.509059 -0.282863  0.469112
2013-01-02 -1.044236  0.119209 -0.173215  1.212112
2013-01-03  1.071804 -0.494929 -2.104569 -0.861849
2013-01-04  0.271860 -1.039575 -0.706771  0.721555
2013-01-05 -1.087401  0.276232  0.567020 -0.424972
2013-01-06  0.524988 -1.478427  0.113648 -0.673690
```

Sorting by values

```
In [22]: df.sort_values(by='B')
Out[22]:
                   A         B         C         D
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
```

Selection of Data

Selecting a single column, which yields a Series, equivalent to df.A

```
In [23]: df['A']
Out[23]:
2013-01-01    0.469112
2013-01-02    1.212112
2013-01-03   -0.861849
2013-01-04    0.721555
2013-01-05   -0.424972
2013-01-06   -0.673690
Freq: D, Name: A, dtype: float64
```

Selection of Data

Selecting via [], which slices the row s.

```
In [24]: df[0:3]
Out[24]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804

In [25]: df['20130102':'20130104']
Out[25]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

Selection of Data

Selecting on a multi-axis by label

```
In [27]: df.loc[:,['A','B']]
Out[27]:
                   A         B
2013-01-01  0.469112 -0.282863
2013-01-02  1.212112 -0.173215
2013-01-03 -0.861849 -2.104569
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
2013-01-06 -0.673690  0.113648
```

Selection of Data

By integer slices, acting similar to numpy/python

```
In [33]: df.iloc[3:5,0:2]
Out[33]:
                   A         B
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
```

By lists of integer position locations, similar to the numpy/python style

```
In [34]: df.iloc[[1,2,4],[0,2]]
Out[34]:
                   A         C
2013-01-02  1.212112  0.119209
2013-01-03 -0.861849 -0.494929
2013-01-05 -0.424972  0.276232
```

Boolean Indexing

Using a single column's values to select data.

```
In [39]: df[df.A > 0]
Out[39]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

Missing Data

To drop any row s that have missing data.

```
In [55]: df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E'])

In [56]: df1.loc[dates[0]:dates[1],'E'] = 1

In [57]: df1
Out[57]:
                   A         B         C  D    F    E
2013-01-01  0.000000  0.000000 -1.509059  5  NaN  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  NaN
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  NaN
```

Filling missing data

```
In [59]: df1.fillna(value=5)
Out[59]:
                   A         B         C  D    F    E
2013-01-01  0.000000  0.000000 -1.509059  5  5.0  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  5.0
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  5.0
```

Stats

Operations in general exclude missing data.

Performing a descriptive statistic

```
In [61]: df.mean()
Out[61]:
A   -0.004474
B   -0.383981
C   -0.687758
D    5.000000
F    3.000000
dtype: float64
```

Same operation on the other axis

```
In [62]: df.mean(1)
Out[62]:
2013-01-01    0.872735
2013-01-02    1.431621
2013-01-03    0.707731
2013-01-04    1.395042
2013-01-05    1.883656
2013-01-06    1.592306
Freq: D, dtype: float64
```

Apply

Applying functions to the data

```
In [66]: df.apply(np.cumsum)
Out[66]:
                   A         B         C   D     F
2013-01-01  0.000000  0.000000 -1.509059   5   NaN
2013-01-02  1.212112 -0.173215 -1.389850  10   1.0
2013-01-03  0.350263 -2.277784 -1.884779  15   3.0
2013-01-04  1.071818 -2.984555 -2.924354  20   6.0
2013-01-05  0.646846 -2.417535 -2.648122  25  10.0
2013-01-06 -0.026844 -2.303886 -4.126549  30  15.0

In [67]: df.apply(lambda x: x.max() - x.min())
Out[67]:
A    2.073961
B    2.671590
C    1.785291
D    0.000000
F    4.000000
dtype: float64
```

Exploratory Data Analysis

Data Exploration

Data Exploration: a preliminary investigation of the data in order to better understand its specific characteristics

Data Exploration

- Machine Learning won't accept ANY data that you supply.
- Garbage In - Garbage Out
- If you are struggling with model accuracy, then data exploration techniques will come to your rescue.

Exploring Data Analysis - EDA

- Patterns can sometime be found simply by visualizing the data
- Summary statistics also used
- Most summary statistics can be calculated in a single pass through the data

Exploring Data Analysis - EDA

Steps to understand, clean and prepare your data for building your predictive model:

- Variable Identification
- Univariate Analysis
- Bi-variate Analysis
- Missing values treatment
- Outlier treatment
- Variable transformation
- Variable creation

Variable Identification

- Identify Predictor (Input) and Target (output) variables.
- Identify the data type and category of the variables.

Variable Identification

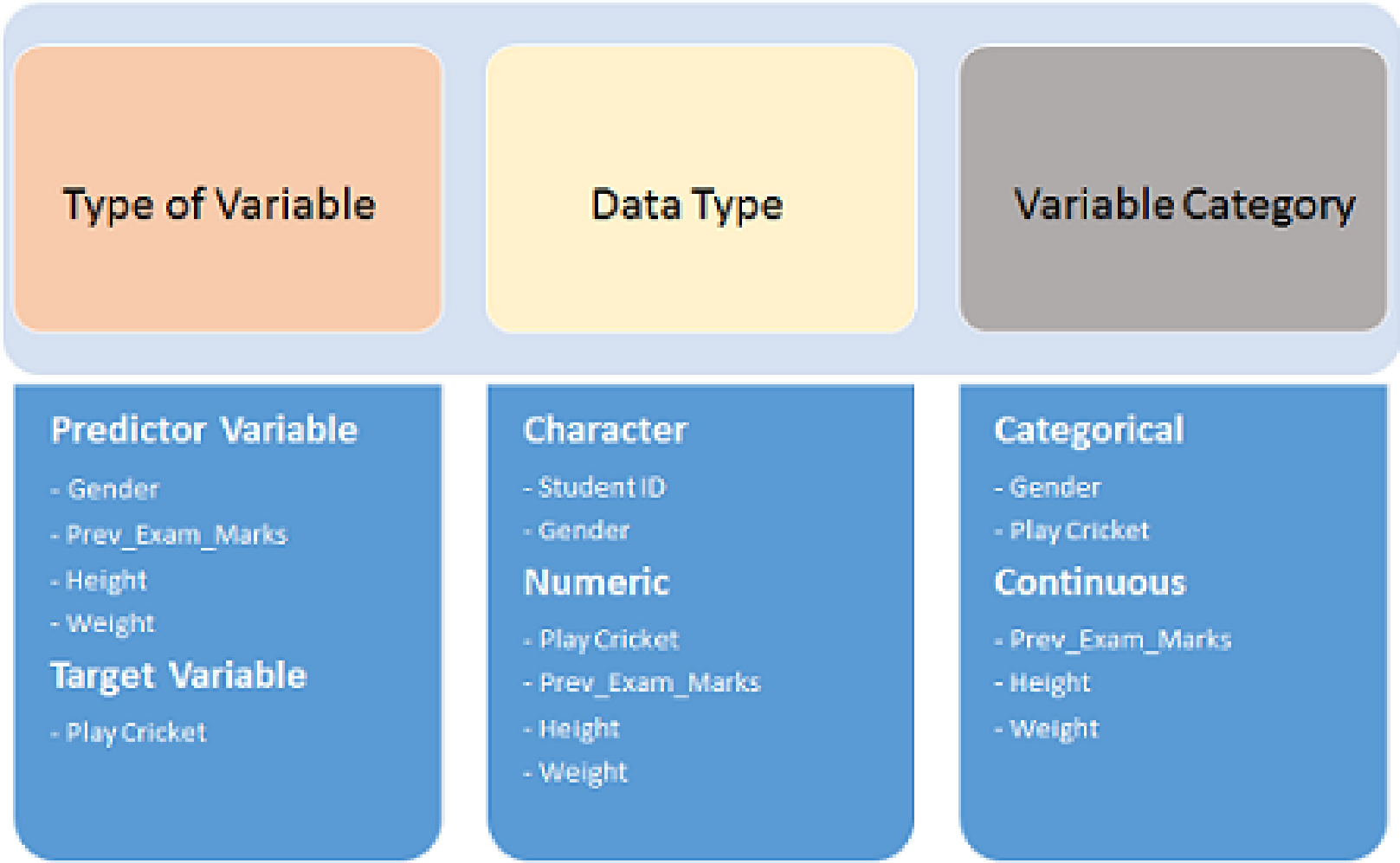
Example:- Suppose, we want to predict, whether the students will play cricket or not (refer below data set).

Student_ID	Gender	Prev_Exam_Marks	Height (cm)	Weight Caregory (kgs)	Play Cricket
S001	M	65	178	61	1
S002	F	75	174	56	0
S003	M	45	163	62	1
S004	M	57	175	70	0
S005	F	59	162	67	0

Here you need to identify:

- Predictor variables
- Target variable
- Data type of variables
- Category of variables.

Variable Identification



Uni-variate Analysis

- Explore variables one by one
- Method of analysis depends on type: categorical or continuous
- Also used to highlight missing and outlier values

Univariate Analysis

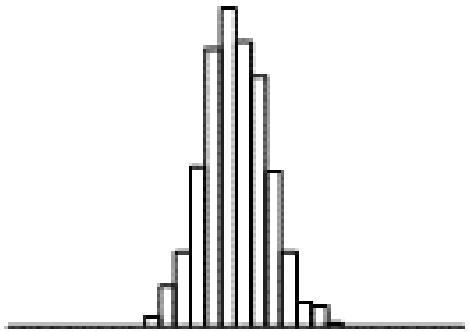
Categorical: use frequency table to understand distribution of each category.

Uni-variate Analysis

Continuous: understand the central tendency and spread of the variable.

- Examine the mean and standard deviation of each feature
- Get a sense of the central tendency and variation of the values
- Examine the minimum and maximum values to understand the range that is possible for each feature
- Histograms of continuous features will resemble the following well understood shapes (probability distributions)
- Recognizing the distribution of values for a feature will be useful when applying machine learning models

Normal Distribution



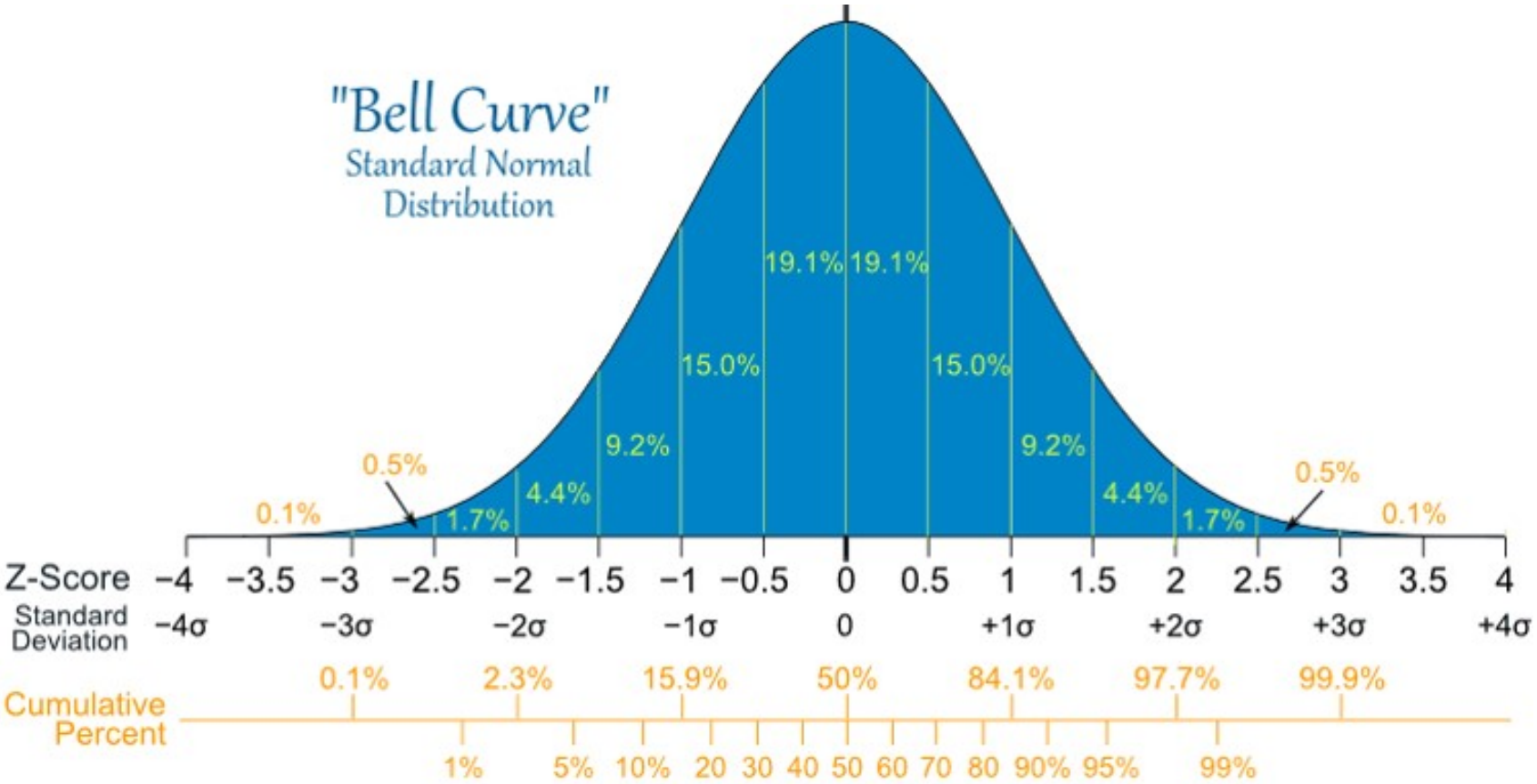
Normal (Unimodal)

- Naturally occurring phenomena (heights, weights of a randomly selected group of men, women) tend to follow a normal distribution.
- Features following a normal distribution are characterized by a strong tendency towards a central value and symmetrical variation to either side of this.

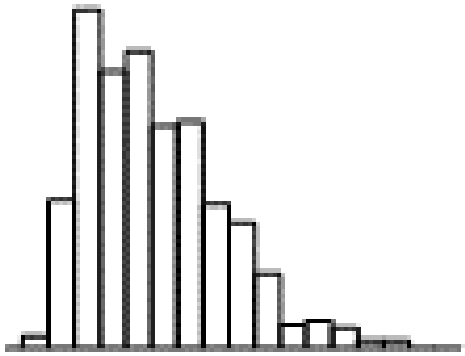
68-95-99.7 Rule

- The 68 - 95 - 99.7 rule is a useful characteristic of the normal distribution.
- The rule states that approximately:
  - 68% of the observations will be within one *sigma* of μ
  - 95% of observations will be within two *sigma* of μ
  - 99.7% of observations will be within three *sigma* of μ
- Very low probability of observations occurring that differ from the mean by more than two standard deviations.

Bell Curve


"Bell Curve"
Standard Normal
Distribution
19.1%
19.1%
15.0%
15.0%
9.2%
9.2%
4.4%
4.4%
1.7%
1.7%
0.5%
0.5%
0.1%
0.1%
Z-Score
−4 −3.5 −3 −2.5 −2 −1.5 −1 −0.5 0 0.5 1 1.5 2 2.5 3 3.5 4
Standard Deviation
−4σ −3σ −2σ −1σ 0 +1σ +2σ +3σ +4σ
Cumulative Percent
0.1% 2.3% 15.9% 50% 84.1% 97.7% 99.9%
1% 5% 10% 20 30 40 50 60 70 80 90% 95% 99%

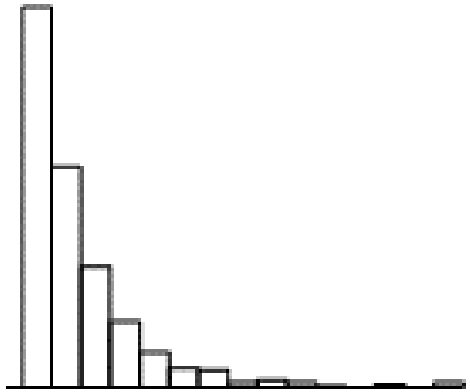
Skewed Distribution



Unimodal (skewed right)

- Skew is simply a tendency towards very high (right skew) or very low (left skew) values.

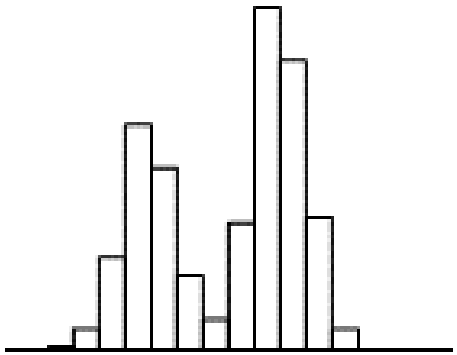
Exponential Distribution



Exponential

- In a feature following an exponential distribution the likelihood of occurrence of a small number of low values is very high, but sharply diminishes as values increase.
- Examples: number of times a person has been married; number of times a person has made an insurance claim

Multimodal Distribution



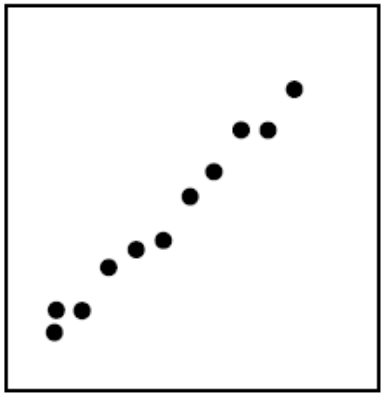
Multimodal

- A feature characterized by a multimodal distribution has two or more very commonly occurring ranges of values that are clearly separated.
- Bi-modal distribution: two clear peaks
- "two normal distributions pushed together"
- Tends to occur when a feature contains a measurement made across two distinct groups

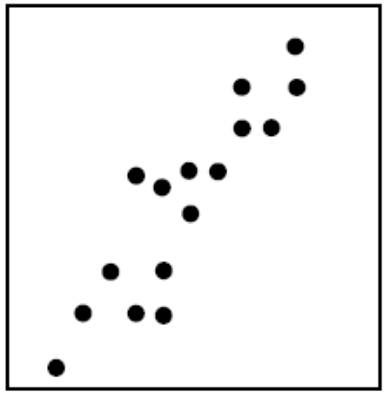
Bi-variate Analysis

- Finds out the relationship between two variables
- Looks for association and disassociation between variables at a pre-defined significance level.
- Most common: between Continuous & Continuous: Correlation plots
- Correlation varies between -1 and +1.

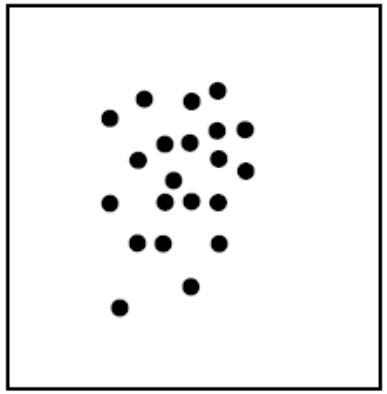
Bi-variate Analysis



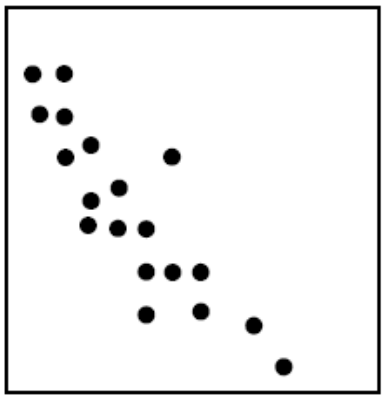
Strong positive correlation



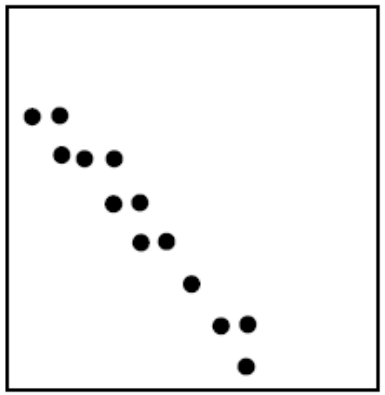
Moderate positive correlation



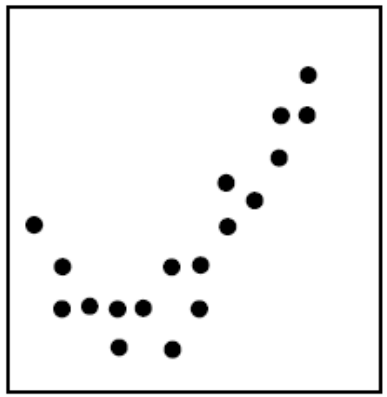
No correlation



Moderate negative correlation



Strong negative correlation



Curvilinear relationship

Bi-variate Analysis

Correlation = Covariance(X,Y) / SQRT(Var(X)* Var(Y))

X	65	72	78	65	72	70	65	68
Y	72	69	79	69	84	75	60	73

Metrics	Formula	Value
Co-Variance (X,Y)	=COVAR(E6:L6,E7:L7)	18.77
Variance (X)	=VAR.P(E6:L6)	18.48
Variance (Y)	=VAR.P(E7:L7)	45.23
Correlation	=G10/SQRT(G11*G12)	0.65

Bi-variate Analysis

- Chi-Square Test: Used for finding relationship between categorical variables.
- Z-Test/ T-Test: Assess whether mean of two groups are statistically different from each other or not.
- Anova: Assesses whether the average of more than two groups is statistically different.

Data collection problems

- Missing values
- Outliers
- Inconsistent values: it often detected inconsistent data with manual typing or handwriting. It is important to correct the data as soon as possible.
- Duplicate data: Often detected when there exists two objects that actually represent a single object.

Missing Value Treatment

Why missing values treatment is required?

- Can reduce the power / fit of a model
- Can lead to a biased model
- Can lead to wrong prediction or classification.

Missing Value Treatment

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	2	1	50%
M	4	2	50%
Missing	2	2	100%

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	4	3	75%
M	4	2	50%

Missing Value Treatment

Why my data has missing values?

- Data Extraction: e.g text extraction failures
- Data collection: eg Unfilled forms, transmission problems

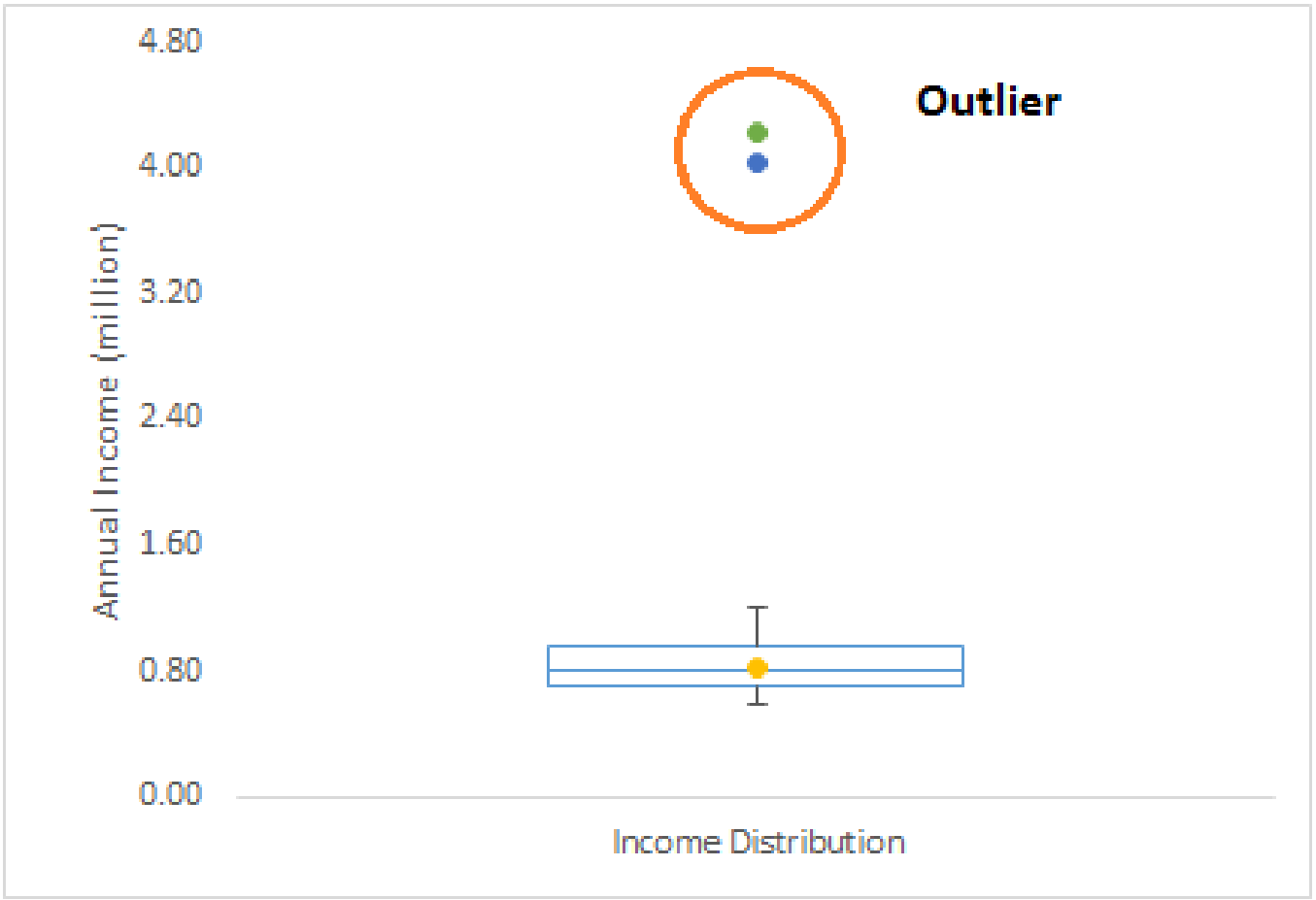
Outliers

- Values of an attribute that are unusual with respect to the typical values for that attribute.
- Important to distinguish between noise and outliers.
- Outliers can be legitimate data, objects or values, unlike noise, outliers may sometimes be of interest.

Outliers

- Data objects that have characteristics that differ from most other data objects. In fraud detection, the goal is identifying these outliers
- Value of an attribute is very unusual with respect to the typical value. Do we have a "data error?" or is some individual really eight foot tall?
- Various statistical definitions for what an outliers is.
- Outliers can be legitimate data objects or values (and may be of interest).

Outliers


4.80
4.00
3.20
2.40
1.60
0.80
0.00
Annual Income (million)
Outlier
Income Distribution

Outliers

Without Outlier	With Outlier
4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7	4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300
Mean = 5.45	Mean = 30.00
Median = 5.00	Median = 5.50
Mode = 5.00	Mode = 5.00
Standard Deviation = 1.04	Standard Deviation = 85.03

- Data set with outliers has significantly different mean and standard deviation.
- In the first scenario, we will say that average is 5.45.
- But with the outliers, average soars to 30.
- This would change the estimate completely.

Inconsistent Values

- Some inconsistencies are easy to detect (and fix) automatically; others are not.
- Example:
  - Data object with address, city, zip code in three separate fields
  - But address / city is in a different zip code

Duplicate Values

Example: many people receive duplicate mailings because they are in a database multiple times under slightly different names

Case Study: Data Quality Report

- A data quality report includes tabular reports that describe the characteristics of each feature in a dataset using standard statistical measures of central tendency and variation.
- The tabular reports are accompanied by data visualizations
  - histogram for each continuous feature
  - bar plot for each categorical feature, also generally used for continuous features with *cardinality* < 10

Data Quality Report

(a) Continuous Features

Feature	Count	% Miss.	Card.	Min.	1st Qrt.	Mean	Median	3rd Qrt.	Max.	Std. Dev.
—	—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—	—

(b) Categorical Features

Feature	Count	% Miss.	Card.	Mode	Mode Freq.	Mode %	2nd Mode	2nd Mode Freq.	2nd Mode %
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—

Card = Cardinality, Measures the number of distinct values present for a feature

Case Study: Data Quality Report

ID	Type	Inc.	Marital Status	Num Claimts.	Injury Type	Hospital Stay	Claim Amnt.	Total Claimed	Num Claims	Num Soft Tiss.	% Soft Tiss.	Claim Amt Rcvd.	Fraud Flag
1	CI	0		2	Soft Tissue	No	1,625	3250	2	2	1.0	0	1
2	CI	0		2	Back	Yes	15,028	66,112	1		0	15,028	0
3	CI	54,613	Married	1	Broken Limb	No	-89,999	0	0	0	0	572	0
4	CI	0		4	Broken Limb	Yes	5,097	11,661	1	1	1.0	7,864	0
5	CI	0		4	Soft Tissue	No	8869	0	0	0	0	0	1
6	CI	0		1	Broken Limb	Yes	17,460	0	0	0	0	17,460	0
7	CI	52,567	Single	3	Broken Limb	No	3,017	18,102	2	1	0.5	0	1
8	CI	0		2	Back	Yes	7463	0	0	0	0	7,463	0
9	CI	0		1	Soft Tissue	No	2,067	0	0	0	0	2,067	0
10	CI	42,300	Married	4	Back	No	2,260	0	0	0	0	2,260	0
300	CI	0		2	Broken Limb	No	2,244	0	0	0	0	2,244	0
301	CI	0		1	Broken Limb	No	1,627	92,283	3	0	0	1,627	0
302	CI	0		3	Serious	Yes	270,200	0	0	0	0	270,200	0
303	CI	0		1	Soft Tissue	No	7,668	92,806	3	0	0	7,668	0
304	CI	46,365	Married	1	Back	No	3,217	0	0		0	1,653	0
458	CI	48,176	Married	3	Soft Tissue	Yes	4,653	8,203	1	0	0	4,653	0
459	CI	0		1	Soft Tissue	Yes	881	51,245	3	0	0	0	1
460	CI	0		3	Back	No	8,688	729,792	56	5	0.09	8,688	0
461	CI	47,371	Divorced	1	Broken Limb	Yes	3,194	11,668	1	0	0	3,194	0
462	CI	0		1	Soft Tissue	No	6,821	0	0	0	0	0	1
491	CI	40,204	Single	1	Back	No	75,748	11,116	1	0	0	0	1
492	CI	0		1	Broken Limb	No	6,172	6,041	1		0	6,172	0
493	CI	0		1	Soft Tissue	Yes	2,569	22,055	1	0	0	2,569	0
494	CI	31,951	Married	1	Broken Limb	No	5,227	22,095	1	0	0	5,227	0
495	CI	0		2	Back	No	5,813	9,892	3	0	0	0	1
496	CI	0		1	Soft Tissue	No	2,118	0	0	0	0	0	1
497	CI	29,280	Married	4	Broken Limb	Yes	5,199	0	0	0	0	0	1
498	CI	0		1	Broken Limb	Yes	32,469	0	0	0	0	16,763	0
499	CI	49,683	Married	1	Broken Limb	No	179,448	0	0		0	179,448	0
500	CI	0		1	Broken Limb	No	8,252	0	0	0	0	0	1

Case Study: Data Quality Report

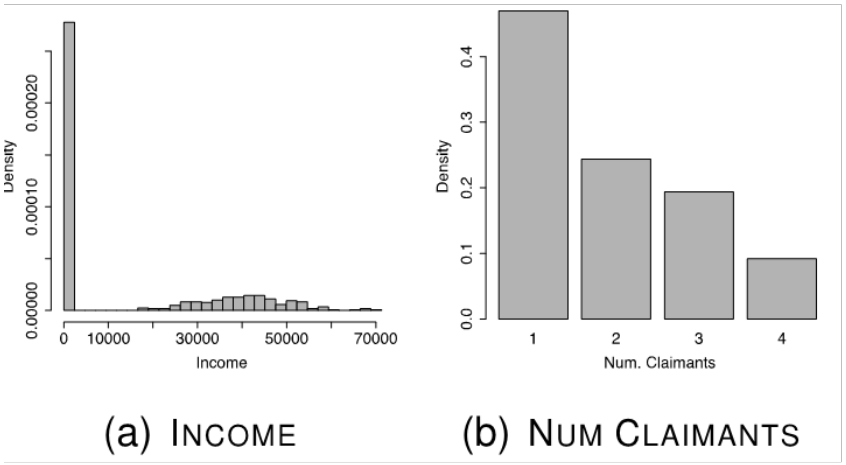
(a) Continuous Features

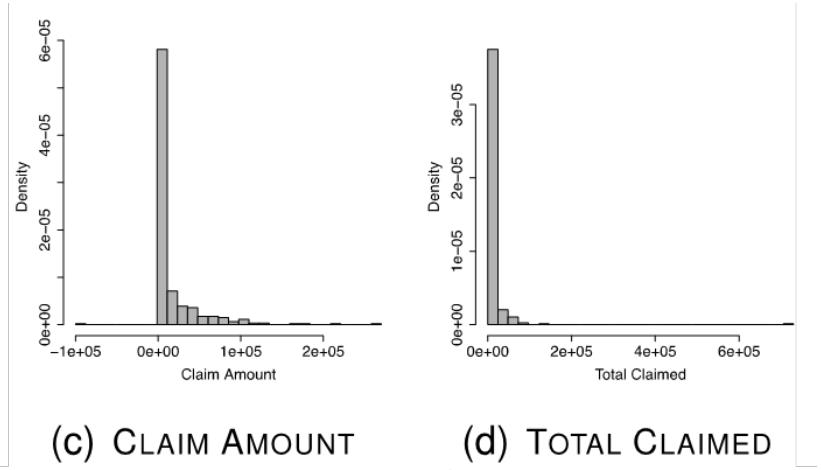
Feature	Count	% Miss.	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt.	Max	Std. Dev.
Income	500	0.0	171	0.0	0.0	13,740.0	0.0	33,918.5	71,284.0	20,081.5
Num Claimants	500	0.0	4	1.0	1.0	1.9	2	3.0	4.0	1.0
Claim Amount	500	0.0	493	-99,999	3,322.3	16,373.2	5,663.0	12,245.5	270,200.0	29,426.3
Total Claimed	500	0.0	235	0.0	0.0	9,597.2	0.0	11,282.8	729,792.0	35,655.7
Num Claims	500	0.0	7	0.0	0.0	0.8	0.0	1.0	56.0	2.7
Num Soft Tissue	500	2.0	6	0.0	0.0	0.2	0.0	0.0	5.0	0.6
% Soft Tissue	500	0.0	9	0.0	0.0	0.2	0.0	0.0	2.0	0.4
Amount Received	500	0.0	329	0.0	0.0	13,051.9	3,253.5	8,191.8	295,303.0	30,547.2
Fraud Flag	500	0.0	2	0.0	0.0	0.3	0.0	1.0	1.0	0.5

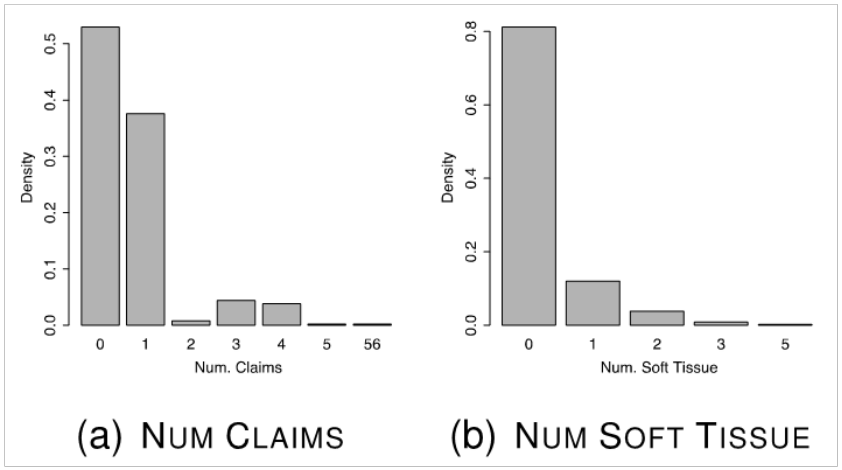
(a) Categorical Features

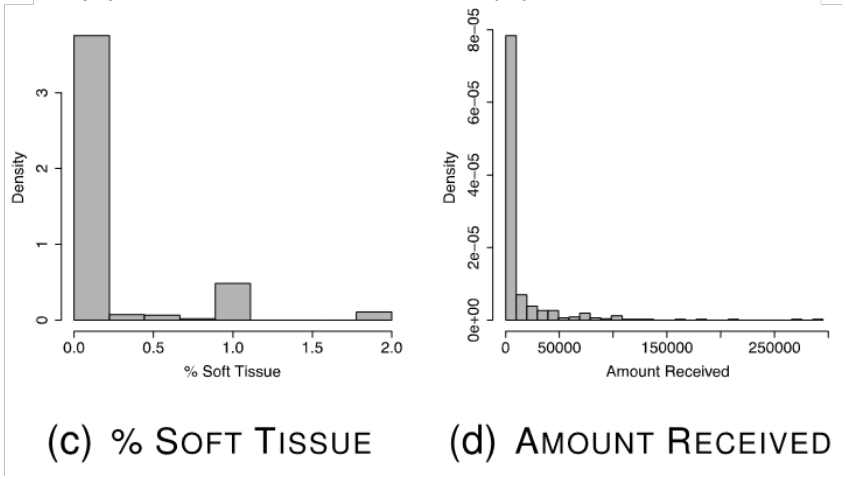
Feature	Count	% Miss.	Card.	Mode	Mode Freq.	Mode %	2nd Mode	2nd Mode Freq.	2nd Mode %
Insurance Type	500	0.0	1	CI	500	1.0	–	–	–
Marital Status	500	61.2	4	Married	99	51.0	Single	48	24.7
Injury Type	500	0.0	4	Broken Limb	177	35.4	Soft Tissue	172	34.4
Hospital Stay	500	0.0	2	No	354	70.8	Yes	146	29.2

Case Study: Data Quality Report

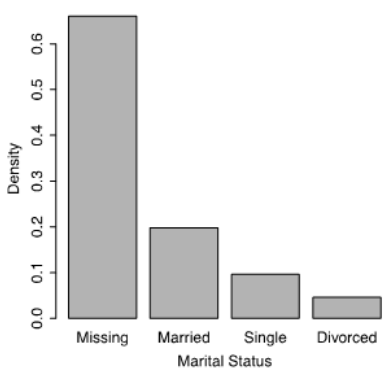




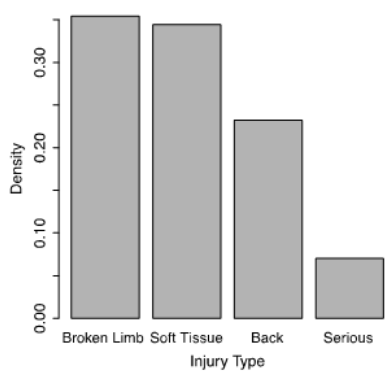




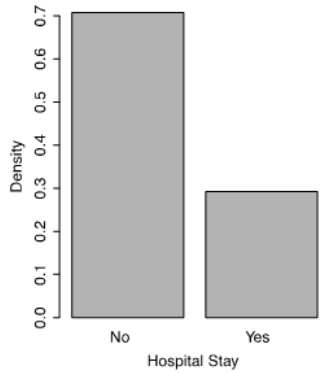
Case Study: Data Quality Report



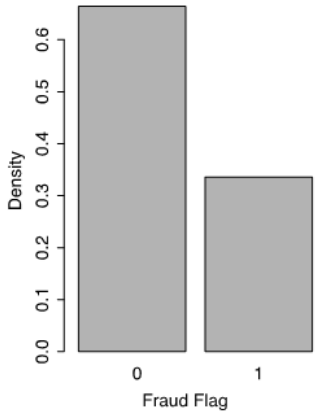
(a) MARITAL STATUS



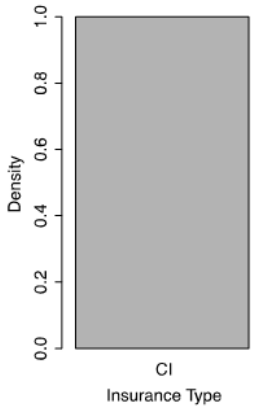
(b) INJURY TYPE



(c) HOSPITAL STAY



(a) FRAUD FLAG



(b) INSURANCE TYPE

Identifying Data Quality Issues

- A data quality issue is loosely defined as anything unusual about the data
- The most common data quality issues are:
  - missing values. Rule of thumb: remove feature is more than 60% of data is missing
  - irregular cardinality. Cardinality of 1: everything has the same value; no useful predictive information. Continuous features will usually have a cardinality value close to the number of instances. Investigate further if cardinality seems much lower or higher than expected
  - outliers (invalid vs. valid). Investigate using domain knowledge. Compare gap between 3rd quartile and max vs. median and 3rd quartile

Identifying Data Quality Issues

- Data quality issues possible due to invalid data. Need to be corrected! (e.g. calculation errors, data entry errors,)
- Data quality issues possible due to valid data, e.g. missing data
- Measurement error: any problem resulting from the measurement process; value recorded differs from true value to some extent
- Data collection error
  - data objects are omitted
  - attribute values are missing for some objects
  - inappropriately including a data object

Case Study

- Become familiar with the central tendency and variation of each feature using the data quality report.
- Note bar graphs and histograms (earlier slides).
- Note number of levels and frequency of Injury Type.
- What is the type of probability distribution for each histogram?
  - Exponential Distribution: all except Income and Fraud Flag
  - Normal Distribution: Income (except for the 0 bar)
  - Fraud Flag: not a typical continuous feature

Thanks . . .

- Feel free to follow me at:
  - Github (github.com/yogeshhk) for open-sourced Data Science training material, etc.
  - Kaggle (www.kaggle.com/yogeshkulkarni) for Data Science datasets and notebooks.
  - Medium (yogeshharibhaukulkarni.medium.com) and also my Publications:
    - Desi Stack https://medium.com/desi-stack
    - TL;DR,W,L https://medium.com/tl-dr-w-l
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com