Lesson A-1

Introduction to Data Mining

Topics

- Introduction to Data Mining/Big Data
- Data mining project development process

POPULAR SCIENCE

THE FUTURE NOW

THE CONTROL CENTERS

Using Data to Feed the World, Solve Cold Cases, Battle Malware, Predict Our Fate P.52

OFFICER ALGORITHM

Can a Crime Be Prevented Before It Begins? P.38

NEW WAYS OF SEEING

A Gallery of Extraordinary Infographics P.69

PLUS

Juan Enriquez Reprograms Life P.31

James Gleick Unsplits the Bit P.58

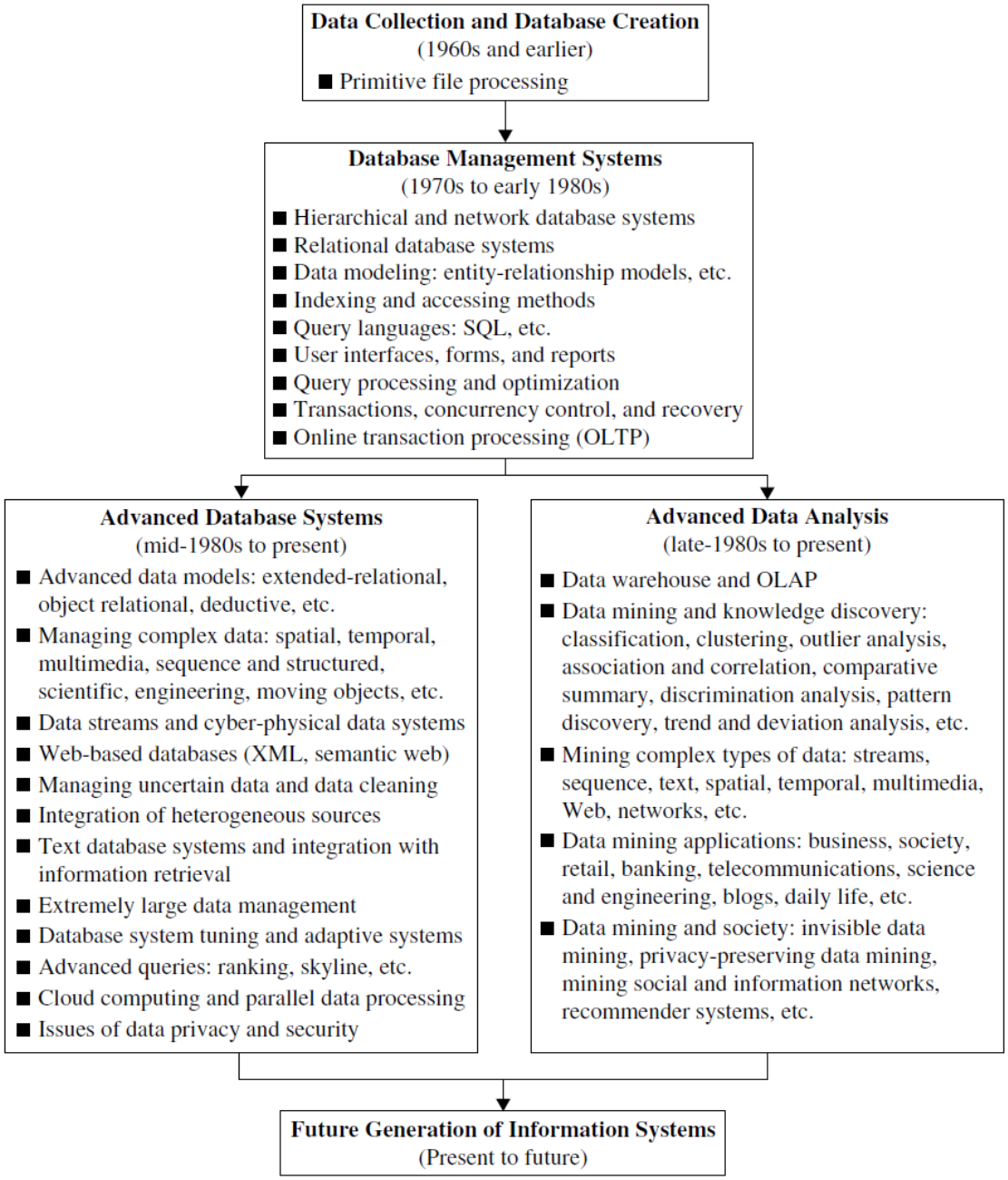
AND **Lawrence Weschler** Questions the Cloud P.76

SPECIAL ISSUE

DATA IS POWER

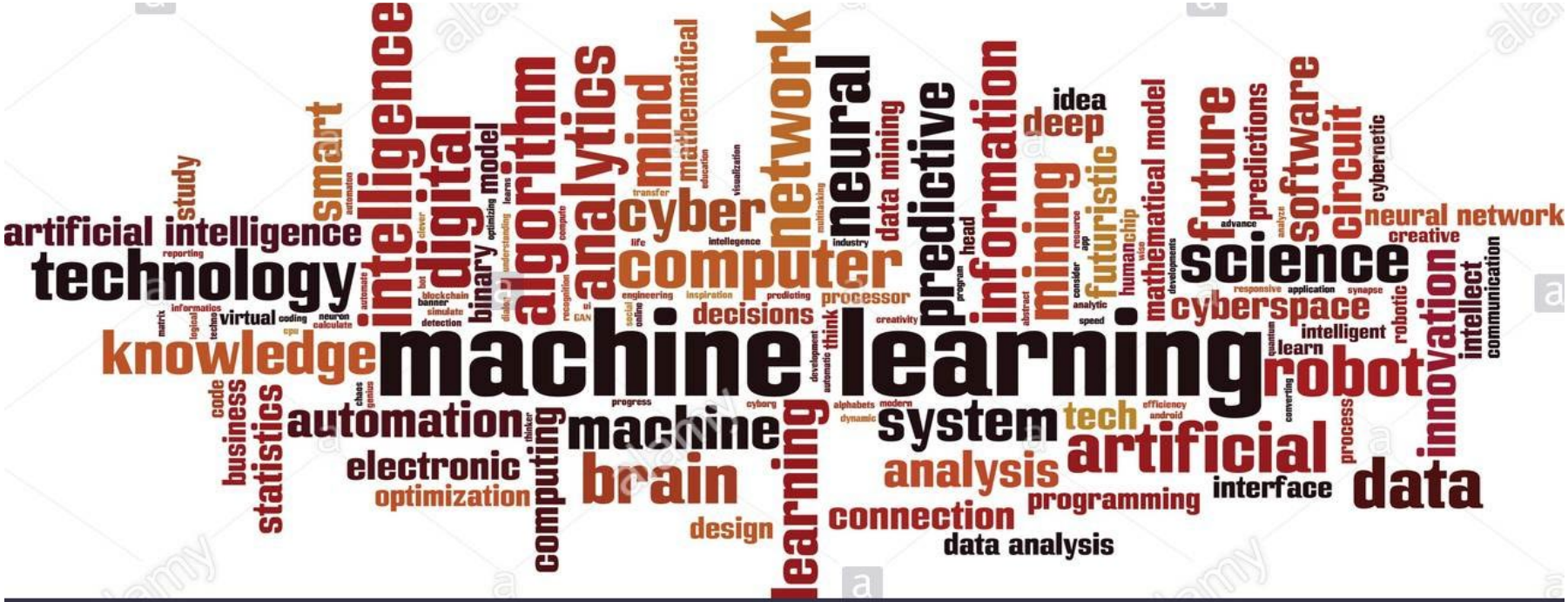
HOW INFORMATION IS DRIVING THE FUTURE

NOVEMBER 2011 US $4.99

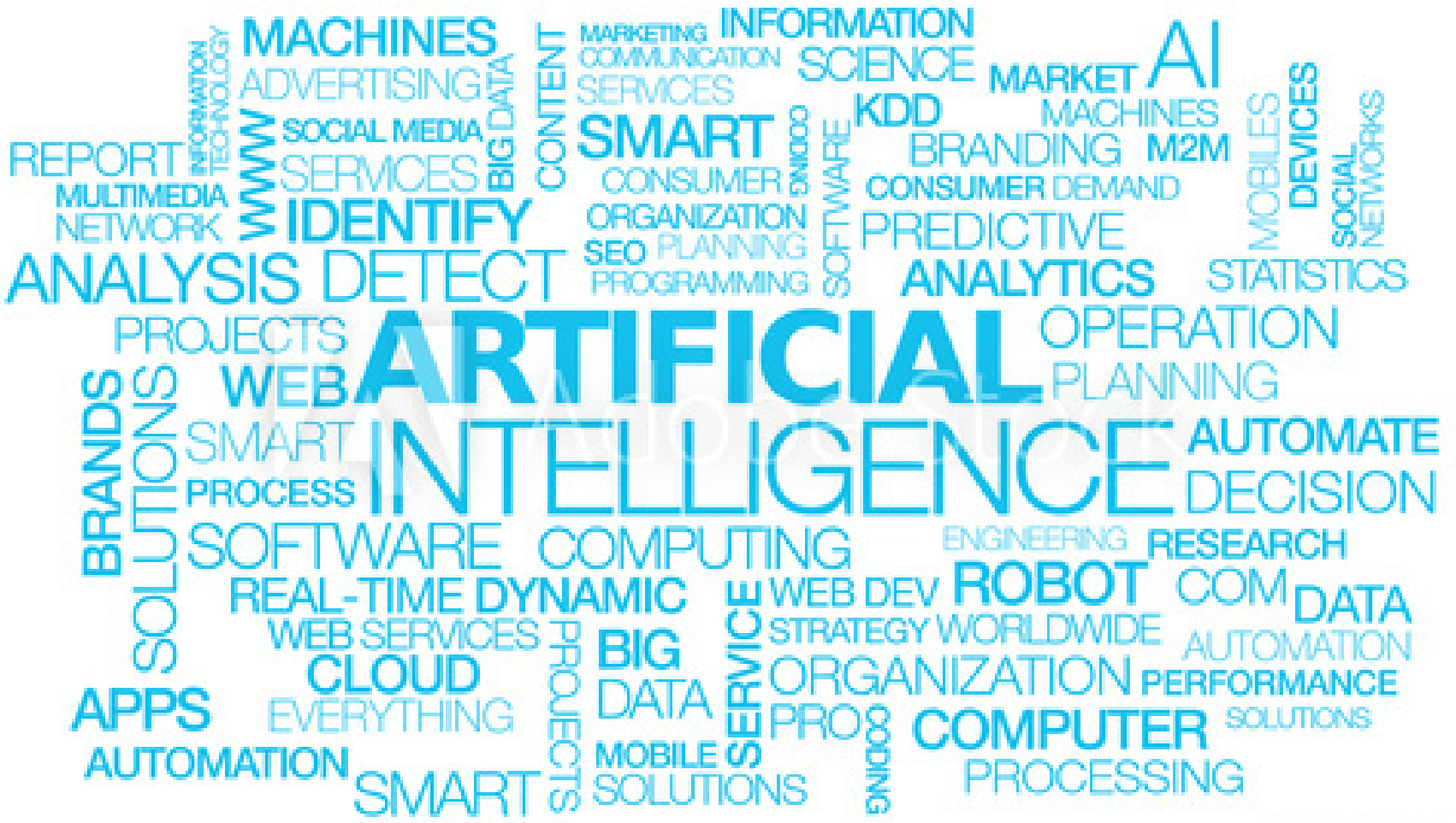


The evolution of database system technology.

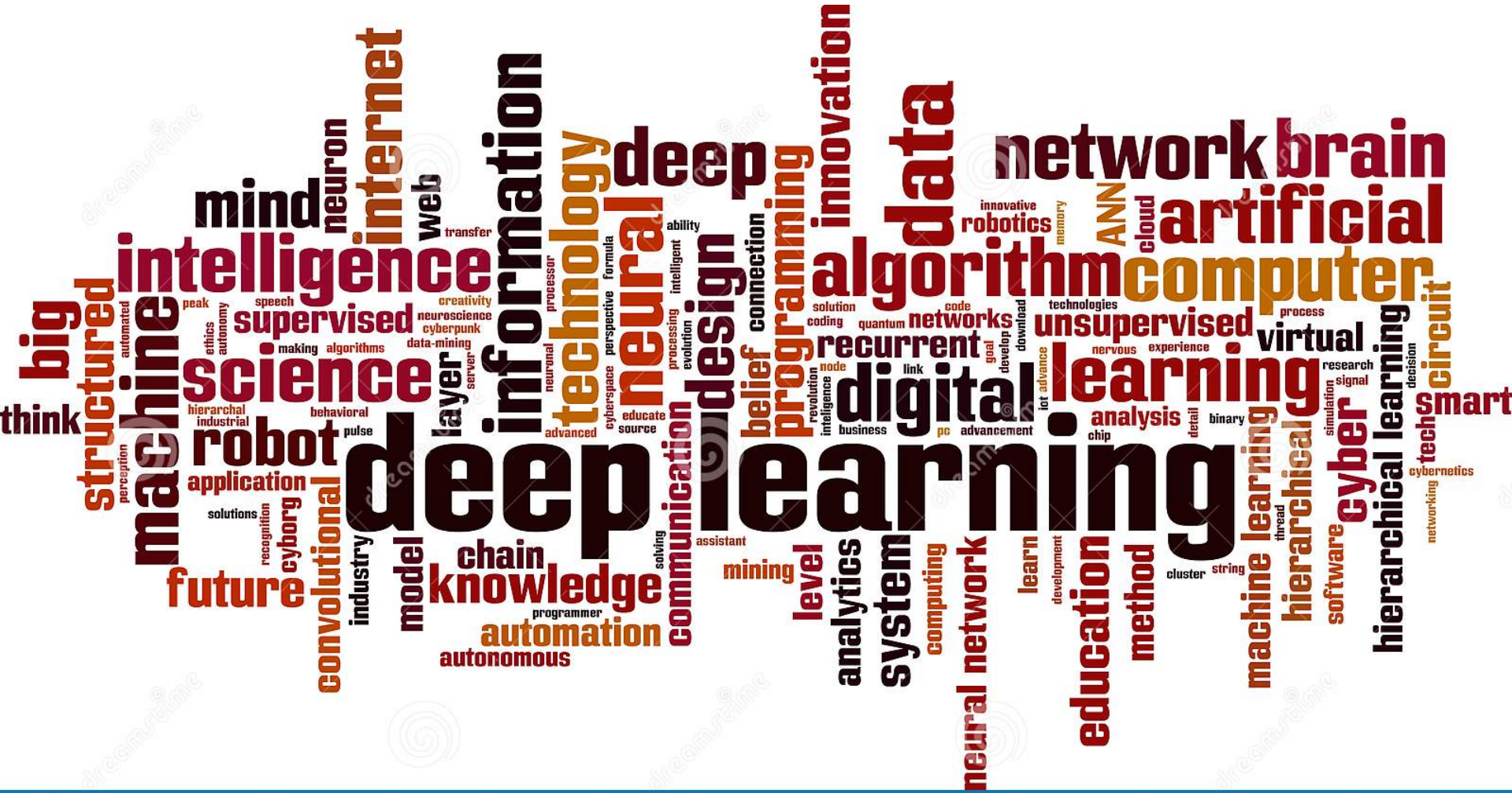

DATA
MINING
PATTERNS
LEARNING
ANALYSIS
SET
LIKELY
PRESENT
GROUPS
MARKETING
GENERAL
SENDING
BUSINESS
KNOWN
MANAGEMENT
METHODS
STRUCTURES
LEARNED
PROCESS
NEW
DATABASE
DECISION
PREDICT
MIGHT
PRODUCT
BASKET
APPLICATIONS
ALGORITHMS
VISUALIZATION
TEST
TERM
POTENTIAL
MUST
USING
WITHOUT
E-MAILS
WITHIN
MARKET
FINAL
DESIRED
OFFERS
DISCOVERY
ASSOCIATION
OFFER
SEND
USE
TASK
MACHINE
TARGET
CUSTOMERS
INVOLVES
EXAMPLE
STEP
ANOTHER
DETERMINE
PEOPLE
RULE
LARGER
DISCOVERED
INTELLIGENCE
CUSTOMER
SUPPORT
CITATION
KNOWLEDGE
MODELING
TRAINED
INFORMATION
ARTIFICIAL
NEEDED
NUMBER
PRE-PROCESSING
IDENTIFY
OFTEN
HELP
SYSTEM
SOPHISTICATED
RECORDS
DETECTION
ALGORITHM
MANY
E-MAIL
MODEL
MAKING
MODELS
COMMON
FINALLY
LARGE




DETECTION
BIG DATA
COMPUTING
INFORMATION TECHNOLOGY
WWW
SOCIAL MEDIA
PROMOTION
E-MARKETING
COMMUNICATION
MODELS
SERVICES
BIG
CONTENT
PROCESSING
BRANDING
COMPUTER
MULTIMEDIA
NETWORK
PROJECTS
CONSUMER
CONSUMER DEMAND MARKETS
SOFTWARE
ORGANIZATION
CODING
WEB MARKETING
PREDICTIVE
PLANNING
PROGRAM
ANALYTICS
EVENTS
PROGRAMMING
DATA MINING
DATA SCIENCE
INFORMATION
MACHINE LEARNING
VISION
ENGINEERING
RESEARCH
PROBABILITY
KDD
WEB DEV
COMPUTING
WEB SERVICES
BIG
SERVICE
STRATEGY
WORLDWIDE
MATHS PATTERN
DATA
VISUALIZATION
BIG DATA
ENGINEERING
PROJECTS
SOLUTIONS
PRO
PRICING
PLANNING
MEDIA
MOBILE
SEGMENTATION
SOCIAL NETWORK
INFORMATION
CODING
SOCIAL NETWORKS
STATISTICS
DIGITAL
TARGET


MACHINES
ADVERTISING
INFORMATION TECHNOLOGY
WWW
SOCIAL MEDIA
SERVICES
BIG DATA
CONTENT
MARKETING
COMMUNICATION
SERVICES
INFORMATION
SCIENCE
SMART
CONSUMER
ORGANIZATION
SEO
PLANNING
PROGRAMMING
SOFTWARE
KDD
MARKET
AI
MACHINES
BRANDING
M2M
CONSUMER DEMAND
PREDICTIVE
ANALYTICS
MOBILES
DEVICES
SOCIAL NETWORKS
STATISTICS
REPORT
MULTIMEDIA
NETWORK
IDENTIFY
ANALYSIS
DETECT
PROJECTS
ARTIFICIAL
INTELLIGENCE
OPERATION
PLANNING
AUTOMATE
DECISION
BRANDS
SOLUTIONS
WEB
SMART
PROCESS
SOFTWARE
COMPUTING
ENGINEERING
RESEARCH
ROBOT
COM
DATA
REAL-TIME
DYNAMIC
WEB SERVICES
CLOUD
APPS
EVERYTHING
AUTOMATION
PROJECTS
BIG
DATA
MOBILE
SOLUTIONS
SERVICE
WEB DEV
STRATEGY
WORLDWIDE
ORGANIZATION
PROG
CODING
COMPUTER
PROCESSING
AUTOMATION
PERFORMANCE
SOLUTIONS
SMART

#87886806

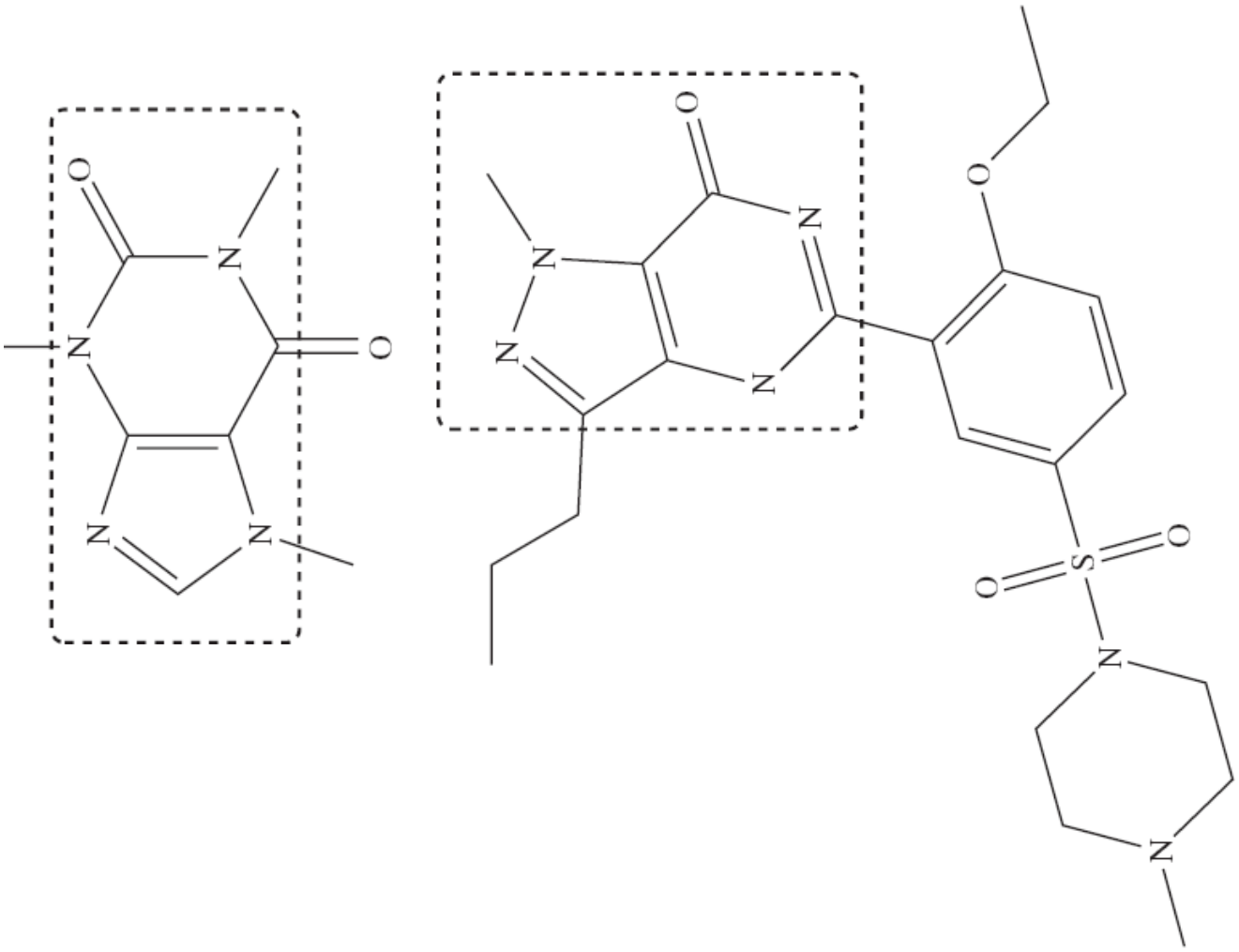



BIG
DATA
ALSO
TOOLS
DISK
TARGET
APPLIED
SENSOR
MAY
EXAMPLES
SHARED
DEFINITION
CURRENT
MOVING
WIRELESS
MANAGE
RECONSIDER
WITHIN
ARCHIVES
BIOGEOCHEMICAL
ZETTABYTES
PRACTITIONERS
CAPTURE
THOUGHT
TECHNOLOGIES
USING
INTERNET
BUSINESS
DISTRIBUTED
SETS
GENOMICS
DESCRIBING
RADIO-FREQUENCY
COMPLEXITY
NEEDED
REQUIRING
TIME
ONE
LARGE
MANAGEMENT
TERABYTES
CASE
ABILITY
SOFTWARE
PROCESS
PETABYTES
INCLUDE
ELAPSED
FG
FORMS
PERFORMANCE
STORE
SYSTEMS
TOLERABLE
SIZE
RESEARCH
INFORMATION
PRESENTATIONS
COMBAT
STORAGE
MPP
LOGS
COST
SIGNIFICANT
DEFINING
RELATED
COMPUTING
PARALLEL
SAN
QUALITIES
SINCE
MASSIVELY
GROW
SEARCH
DIFFICULTY
ANALYTICS
COMPLEX
TENS
LARGER
CONTINUES
SOCIAL
CITATION
INDEXING
SET
USE
RELATIONAL
CONNECTOMICS
ORGANIZATIONS
USED
CREATED
NOW
BIOLOGICAL
PROCESSING
UBIQUITOUS
CAPACITY
BURIED
WORLD'S
HUNDREDS
DESKTOP
RECORDS
NETWORKS
DATABASES
SOLID
TYPES
CURRENTLY
GARTNER
WORKING
AMOUNT
OPPORTUNITIES

What is Data Mining?

- *Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems*.
- *Extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) patterns or knowledge from huge amount of data.*

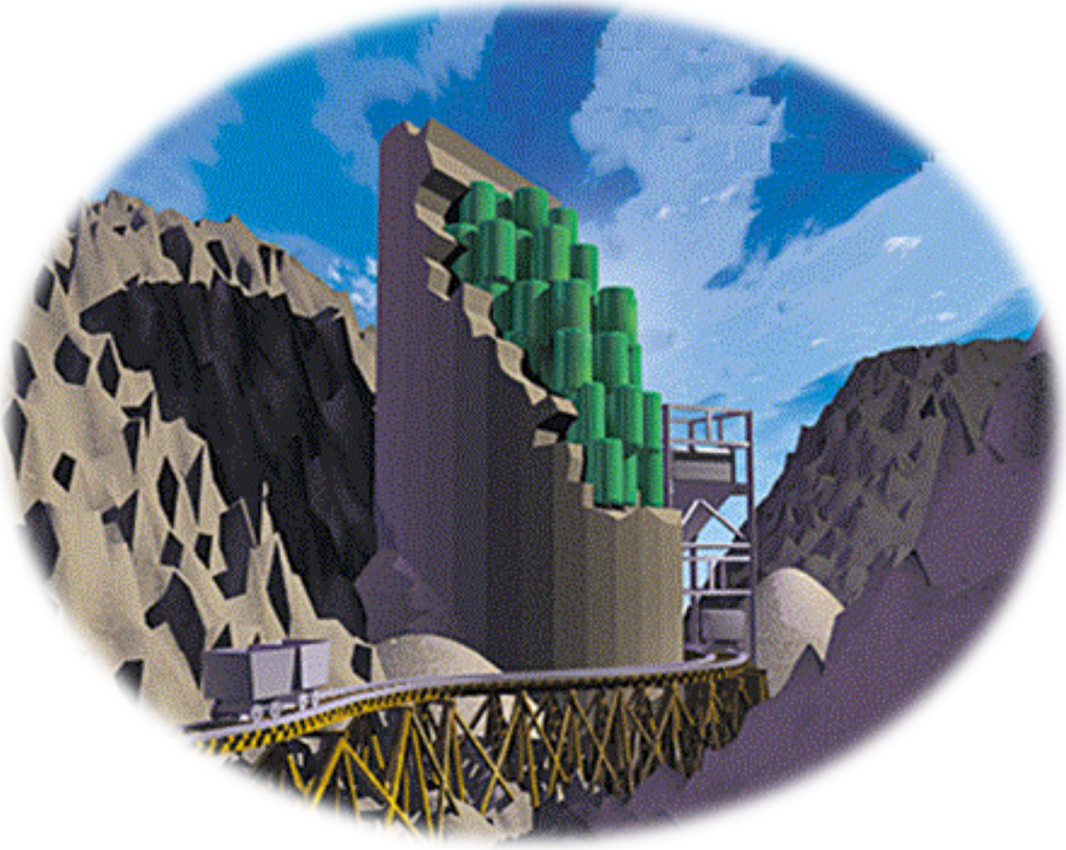
Market Basket Analysis



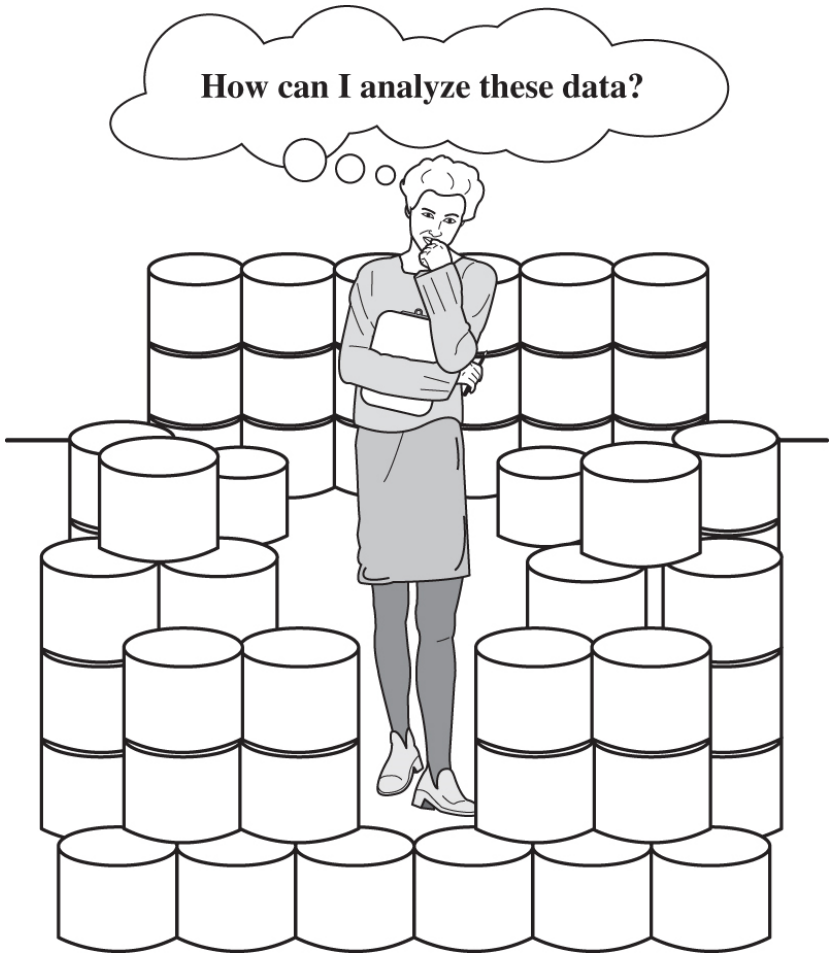
A Chemical database.

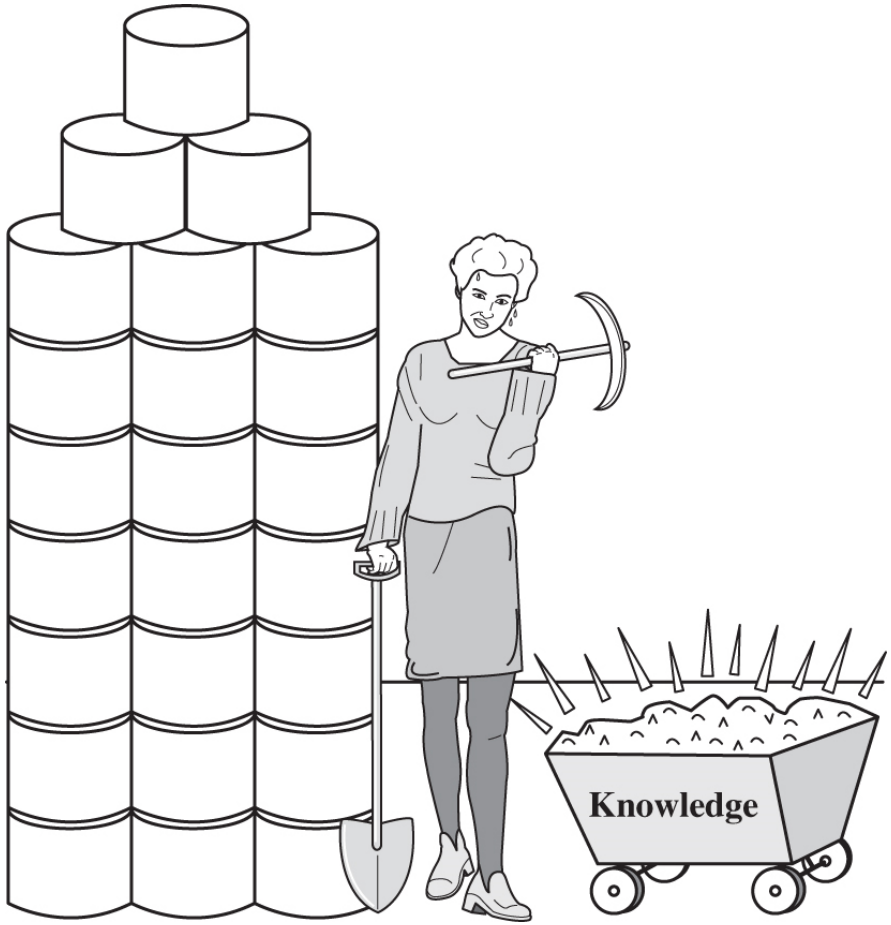
Chemistry Informatics

What is Data Mining?



Source: Cover page of *Advanced in Knowledge Discovery and Data Mining ,* edited by U. Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusamy, MIT Press

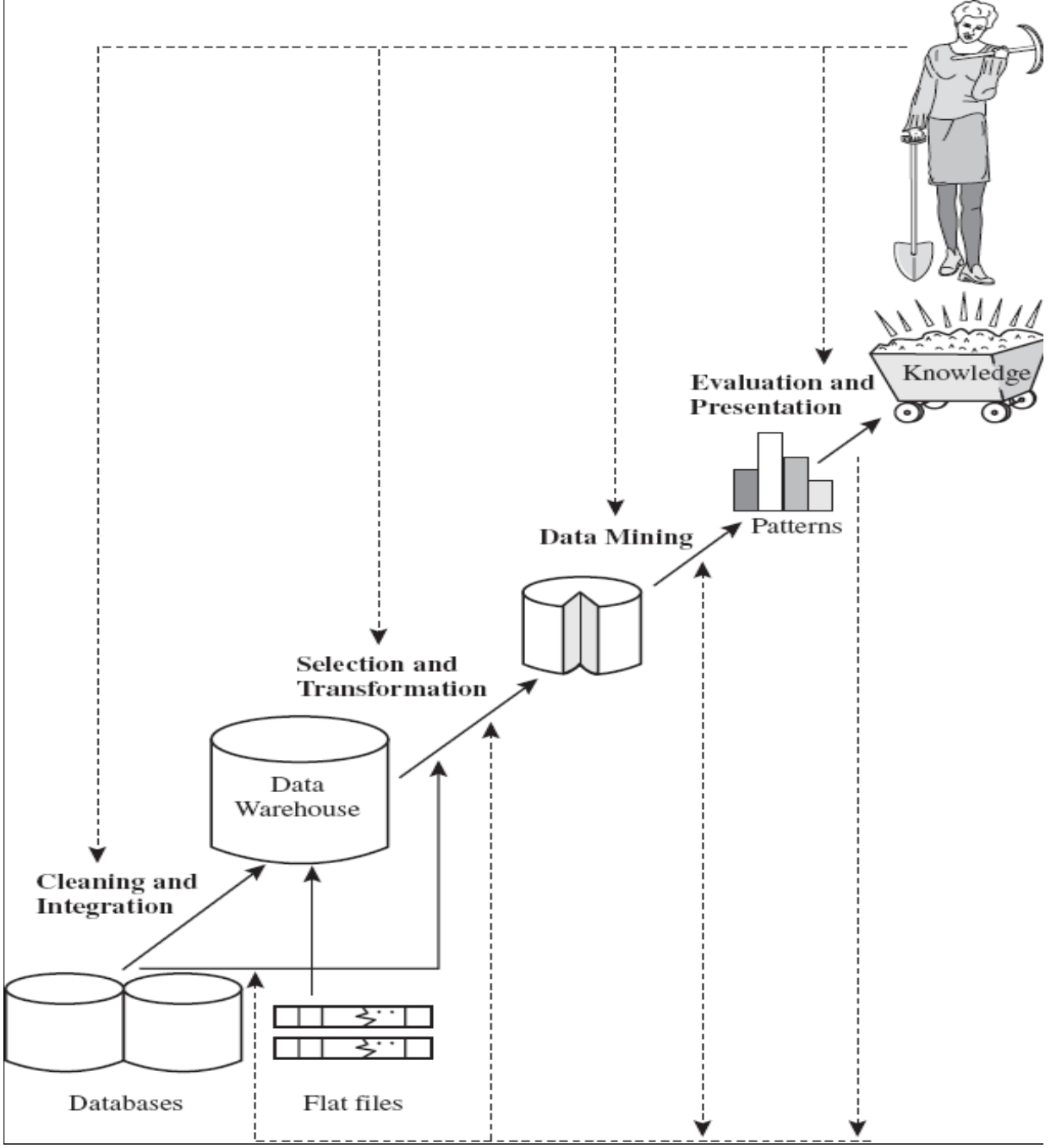

How can I analyze these data?


Knowledge

What Is Data Mining?

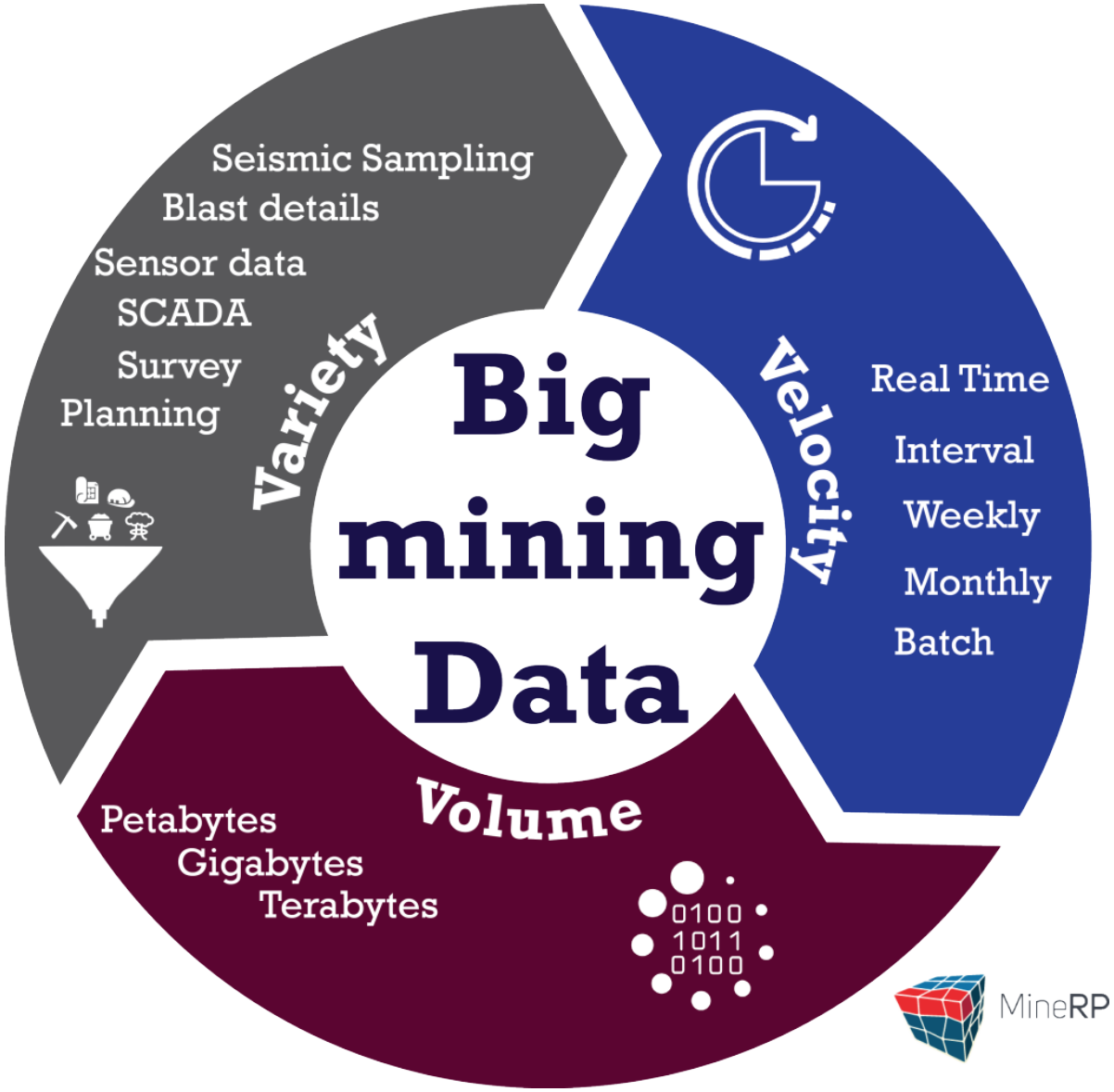
- Data mining (knowledge discovery from data)
  - Extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) patterns or knowledge from huge amount of data
  - Data mining: a misnomer?
- Alternative names
  - Knowledge discovery (mining) in databases (KDD), knowledge extraction, data/pattern analysis, data archeology, data dredging, information harvesting, business intelligence, etc.
- Watch out: Is everything "data mining"?
  - Simple search and query processing
  - (Deductive) expert systems

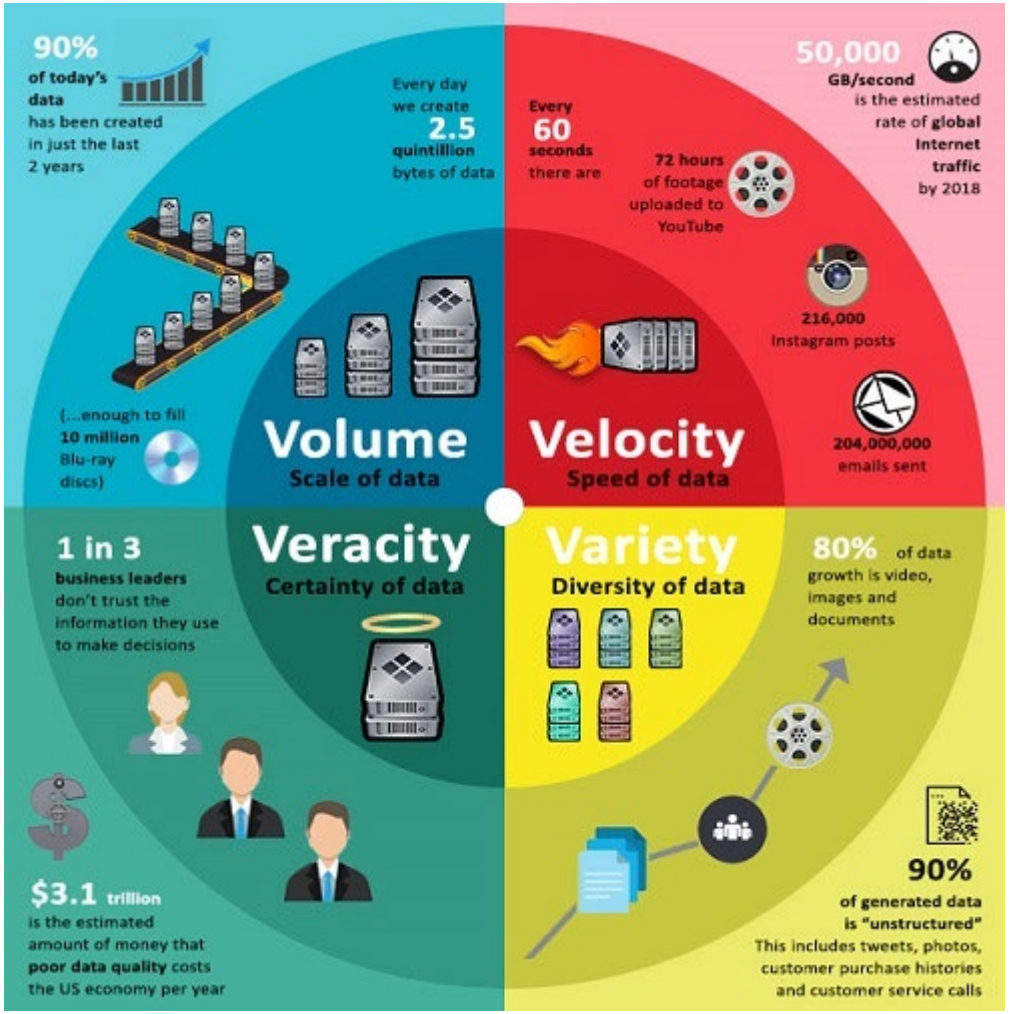
Data Mining is a Process of knowledge discovery



Data mining as a step in the process of knowledge discovery


Google
https://youtu.be/zDAYZU4A3w0


Big mining Data
Variety
Seismic Sampling
Blast details
Sensor data
SCADA
Survey
Planning
Velocity
Real Time
Interval
Weekly
Monthly
Batch
Volume
Petabytes
Gigabytes
Terabytes
0100
1011
0100
MineRP


90%
of today's data
has been created
in just the last
2 years
Every day
we create
2.5
quintillion
bytes of data
(...enough to fill
10 million
Blu-ray
discs)
Volume
Scale of data
Every
60
seconds
there are
72 hours
of footage
uploaded to
YouTube
50,000
GB/second
is the estimated
rate of global
Internet
traffic
by 2018
216,000
Instagram posts
204,000,000
emails sent
Velocity
Speed of data
1 in 3
business leaders
don't trust the
information they use
to make decisions
Veracity
Certainty of data
$3.1 trillion
is the estimated
amount of money that
poor data quality costs
the US economy per year
Variety
Diversity of data
80% of data
growth is video,
images and
documents
90%
of generated data
is "unstructured"
This includes tweets, photos,
customer purchase histories
and customer service calls

The Five V's of Big Data

Scale of Data

This refers to the sheer volume of data being generated every second.

6 Billion People have cell phones

40 Zettabytes of data will be created by 2020 and increase of 300 times from 2005

Most companies in the U.S. have at least **100 Terabytes** of data stored.

The New York Stock Exchange capture **1 TB of Trade Information**

Analysis of Streaming Data

Denotes the speed at which data is emanating and changes are occurring between the diverse data sets.

By 2016 it is projected there will be **18.9 Billion** network connections

Modern cars have close to **100 Sensors**

4 Billion+ hours of video are watched on You Tube each month

30 Billion pieces of content are shared on facebook every month

400 Million tweets are sent per day by about 200 million monthly active users

Diffrent forms of data

As more and more data is being digitized.

1 in 3 Business leaders don't trust the information they use to make decisions

Uncertainty Of Data

This refers to the discrepancies found in the data.

Poor data quality costa the US economy around **$ 3.1 Trillion a year**

Value Of Data

Having access to big data is all well and good but that's only useful if we can turn it into a value.

Exhibit 1

Big data can generate significant financial value across sectors

US health care
- $300 billion value per year
- ~0.7 percent annual productivity growth

Europe public sector administration
- €250 billion value per year
- ~0.5 percent annual productivity growth

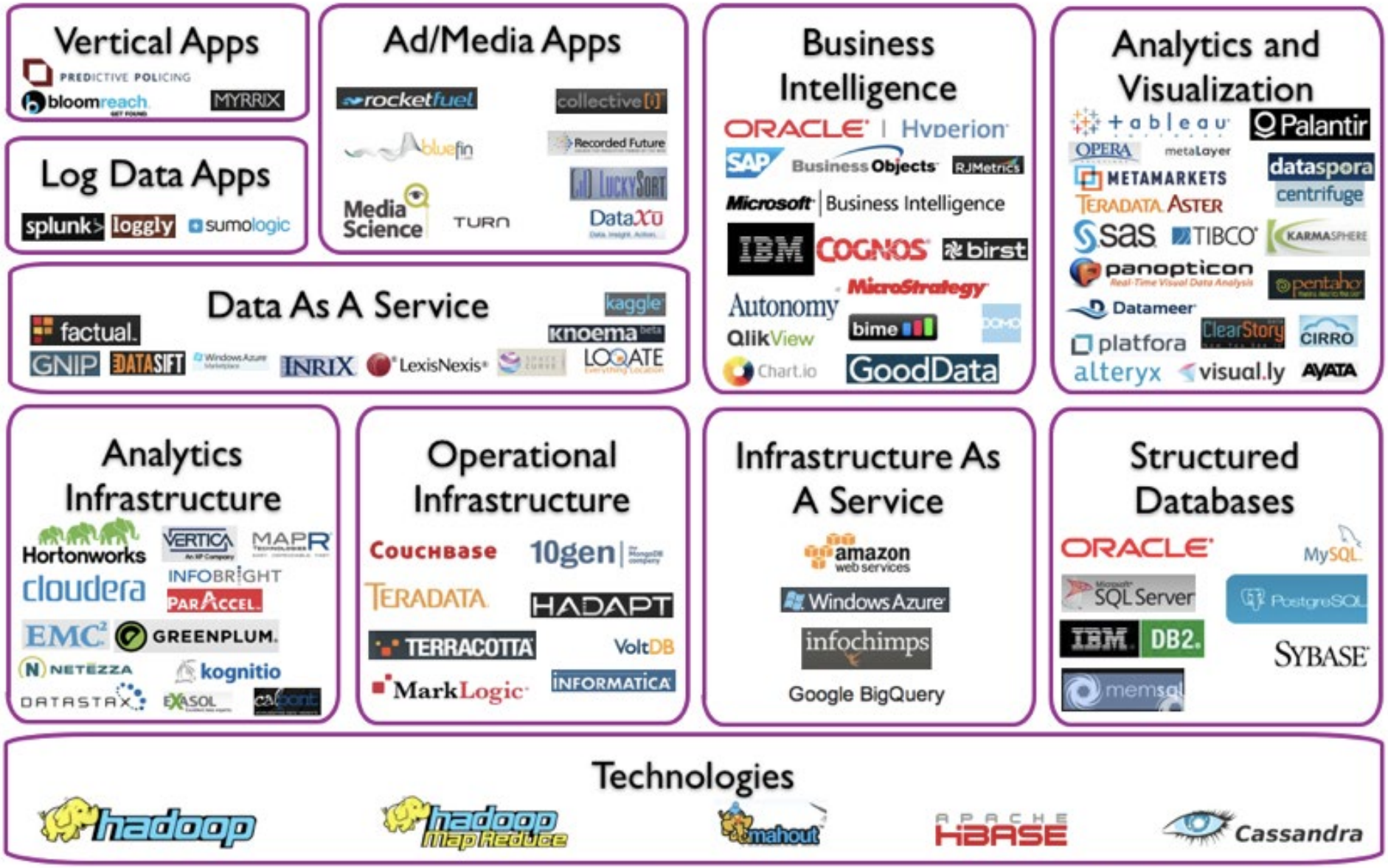
Global personal location data
- $100 billion+ revenue for service providers
- Up to $700 billion value to end users

US retail
- 60+% increase in net margin possible
- 0.5–1.0 percent annual productivity growth

Manufacturing
- Up to 50 percent decrease in product development, assembly costs
- Up to 7 percent reduction in working capital

SOURCE: McKinsey Global Institute analysis

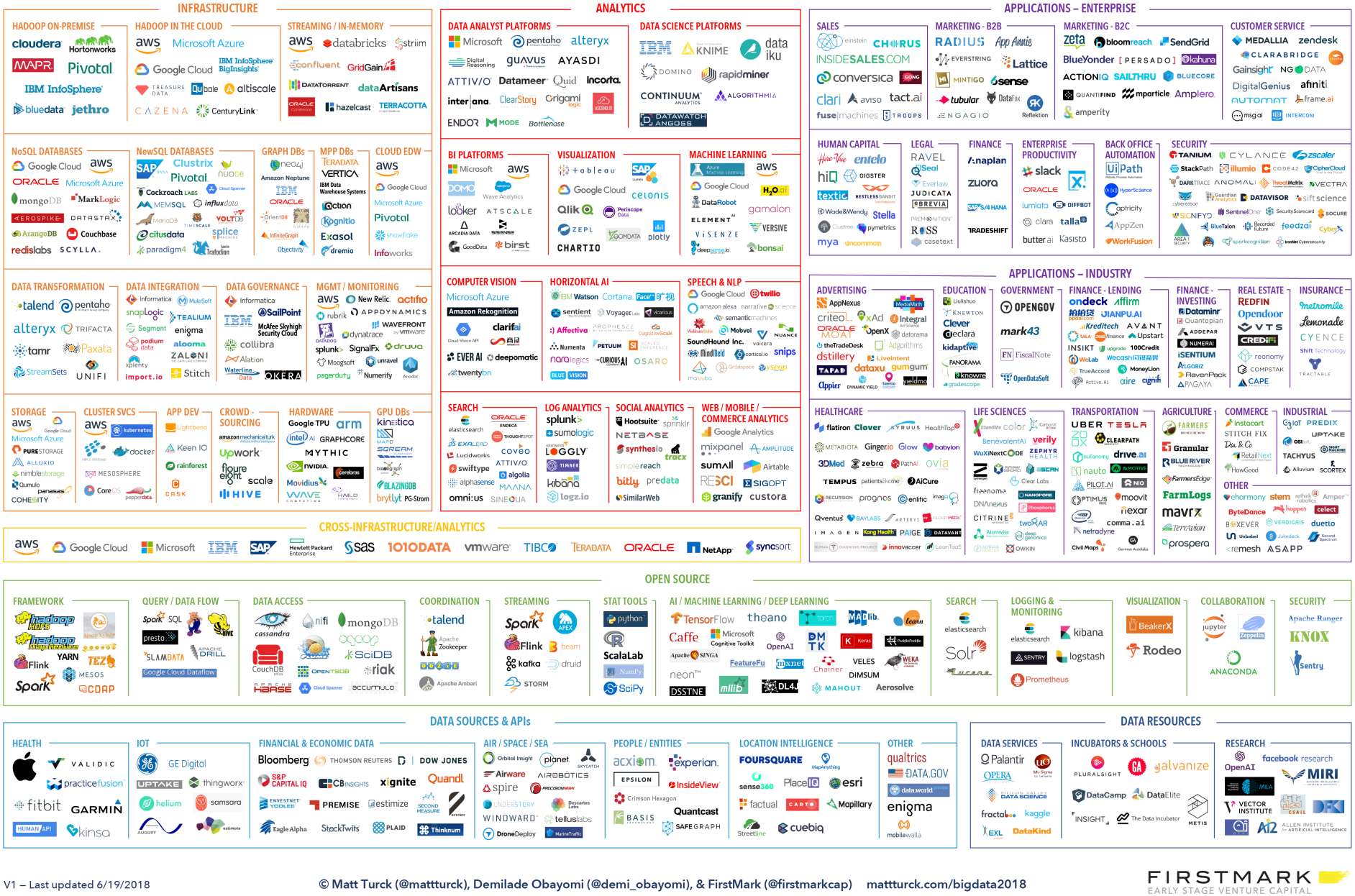
Big Data Landscape



dave@vcdave.com

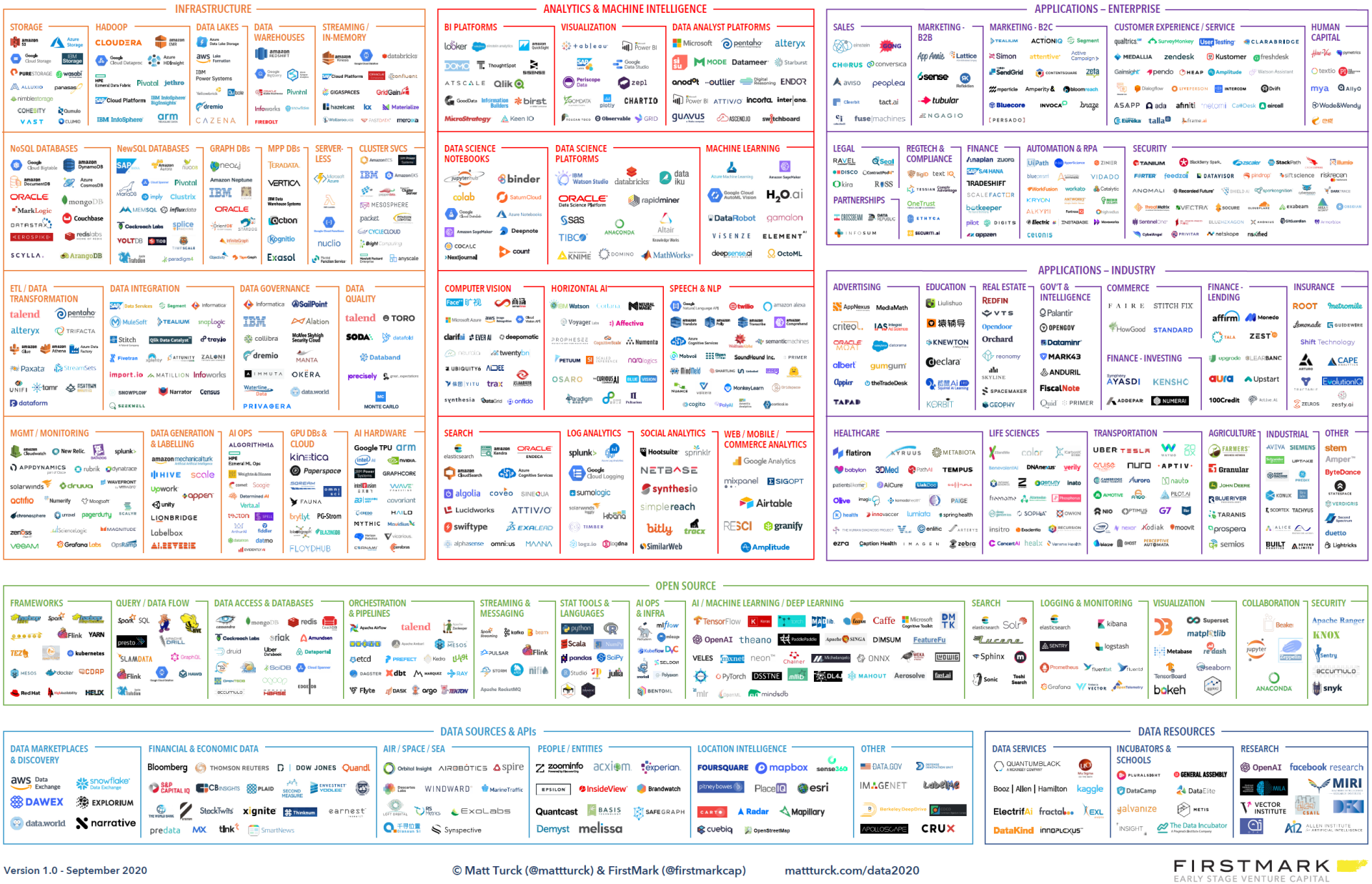
blogs.forbes.com/davefeinleib

BIG DATA & AI LANDSCAPE 2018



FIRSTMARK
EARLY STAGE VENTURE CAPITAL

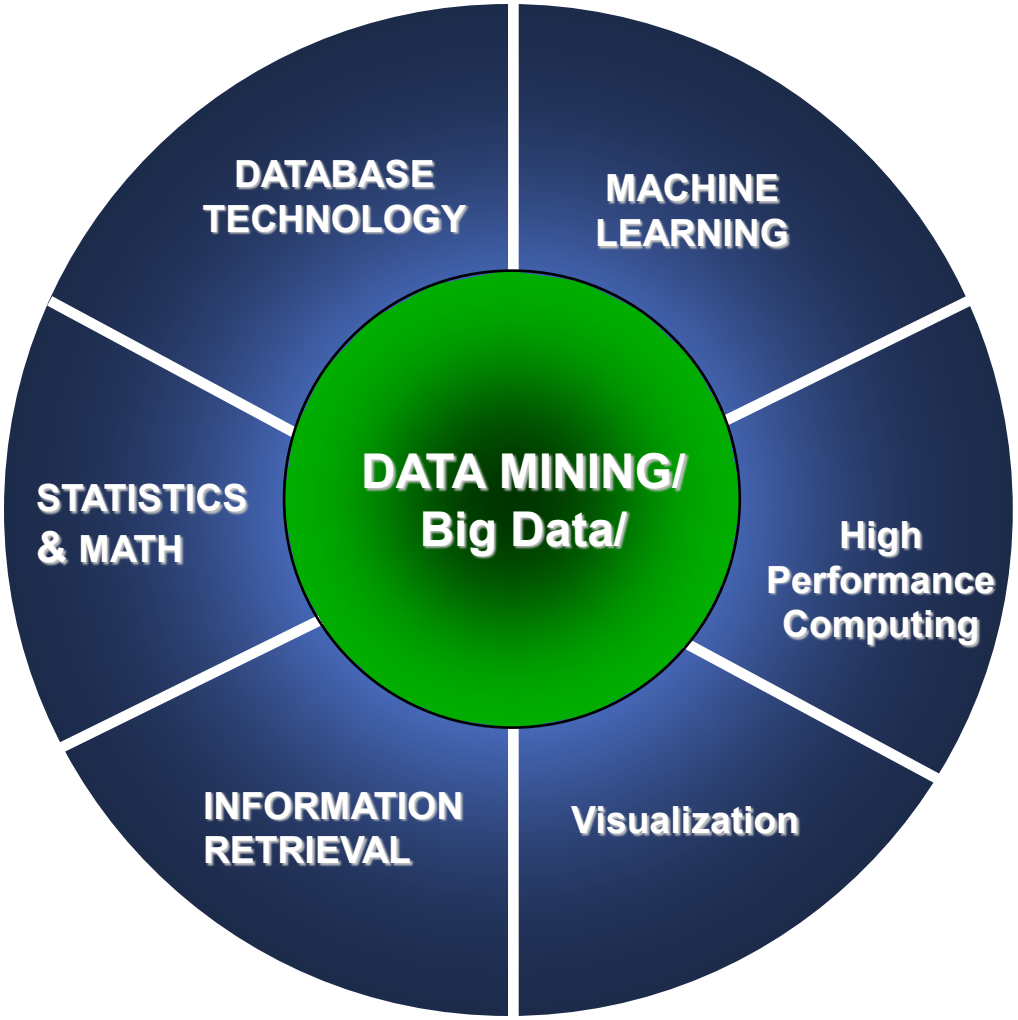
DATA & AI LANDSCAPE 2020



mattturck.com/data2020

FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Data Mining/Big Data/Data Science is an Interdisciplinary and Multidisciplinary Field



Why Data Mining/Big Data is Important?

- *McKinsey predicts that data--driven technologies will bring an additional $300 billion of value to the U.S. health care sector alone, by 2020, 1.5 million more "data--savvy managers" will be needed to capitalize on the potential of data*

  – *May 1, 2011*

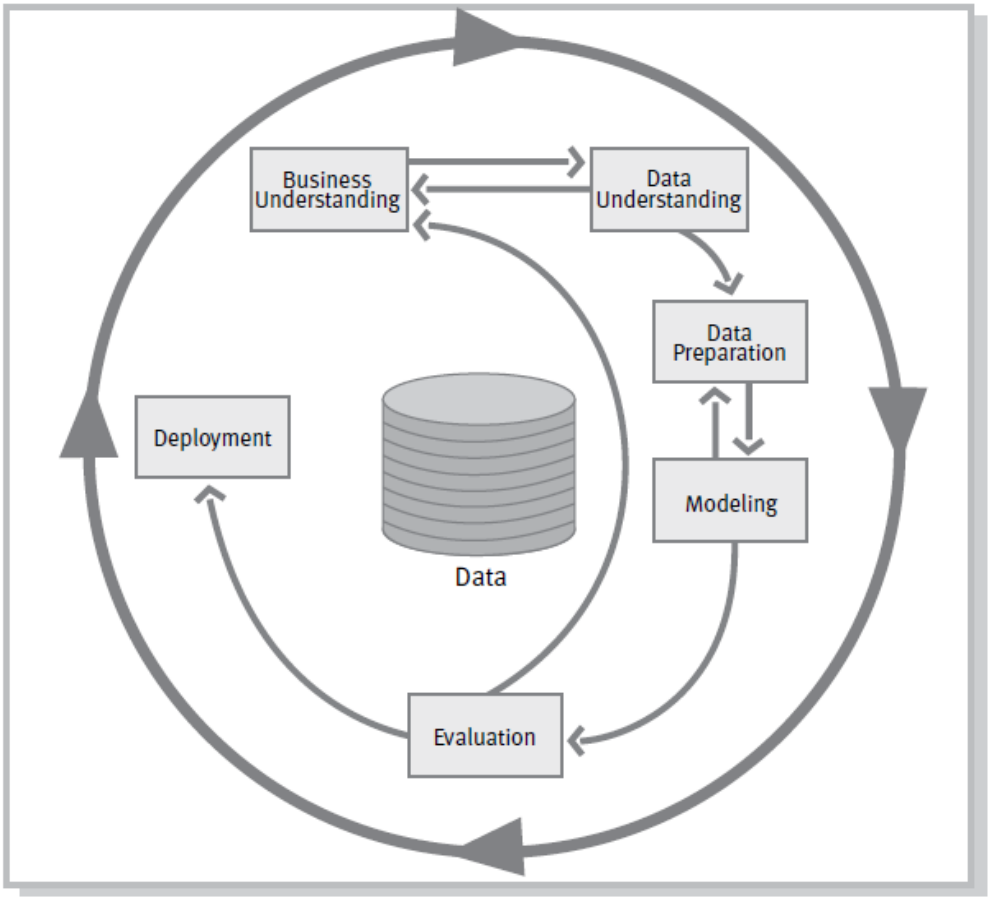
Multi-Dimensional View of Data Mining/Big Data

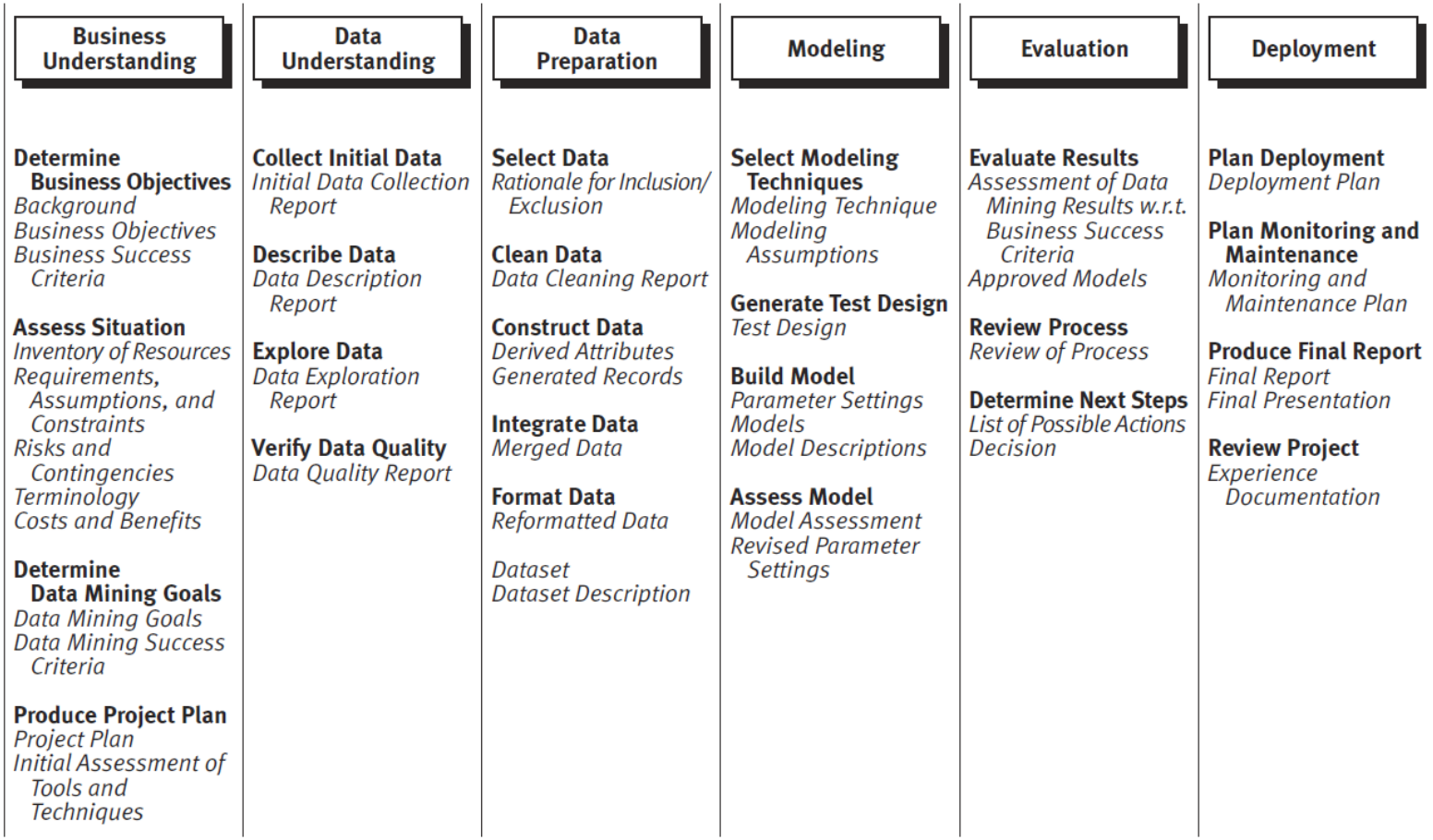
- **Data to be mined**
  - Structured data, semi-structured data, unstructured data, relational data, data warehouse, transactional data, stream, spatiotemporal, time-series, sequence, text and web, multi-media data, graphs data, social network, etc.
- **Data preprocessing and feature engineering**
  - Cleaning, integration, reduction, transformation, generalization, feature engineering, etc.
- **Knowledge to be mined (Data mining functions and models)**
  - generalization, association, classification, clustering, etc.
- **Techniques utilized**
  - data warehouse (OLAP), frequent pattern mining, machine learning, statistics, pattern recognition, visualization, high-performance computing, parallel and distributed computing, cloud computing, etc.
- **Applications adapted**
  - Market basket analysis, healthcare, manufacturing engineering, education, CRM, fraud detection, customer segmentation, intrusion detection, financial banking, criminal investigation, bio Informatics, direct marketing, classifying stars, medical diagnosis, computer vision, drug discovery, speech recognition, handwriting recognition, credit scoring, human genetic clustering, medical imaging, market segmentation, product positioning, new product development, social network analysis, image segmentation, recommender systems, anomaly detection, crime analysis, climatology, market research, customer trend analysis, market movement tracking, customer buying patterns, analyze tax records, decision making, on line operations and report, etc.

CRISP-DM

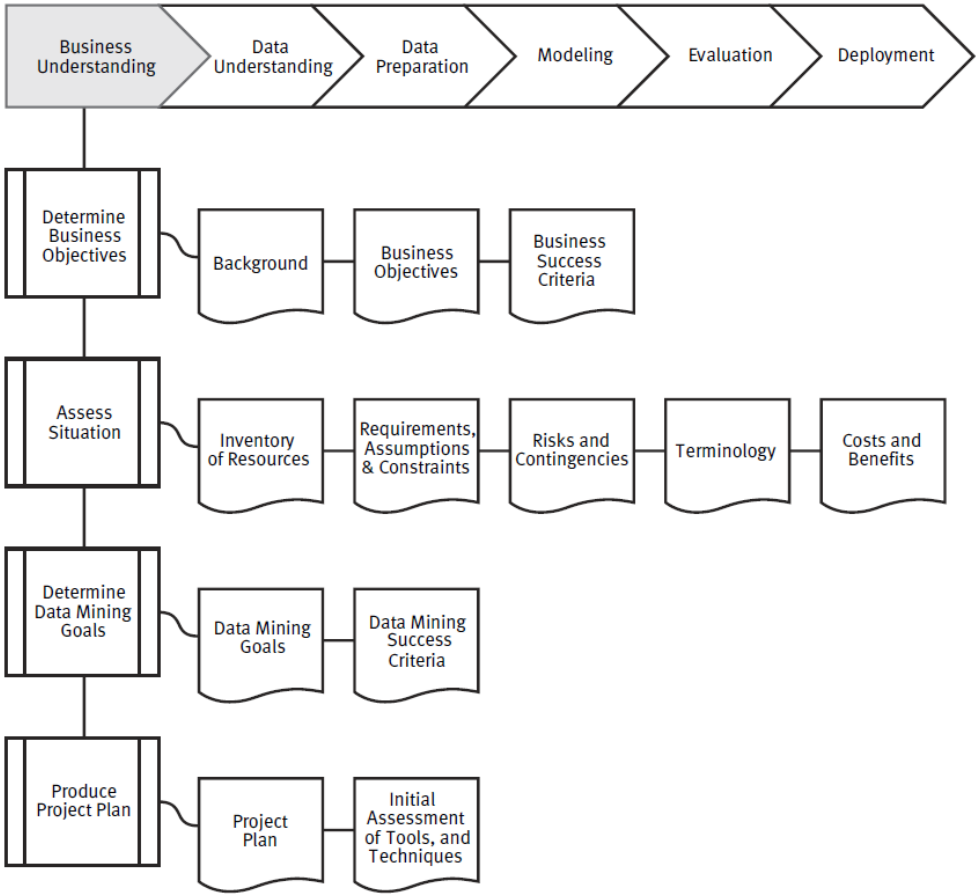
- CRoss-Industry Standard Process for Data Mining
  - is described in terms of a hierarchical process model, consisting of sets of tasks described at four levels of abstraction (from general to specific): phase, generic task, specialized task, and process instance

The life cycle of a data mining project

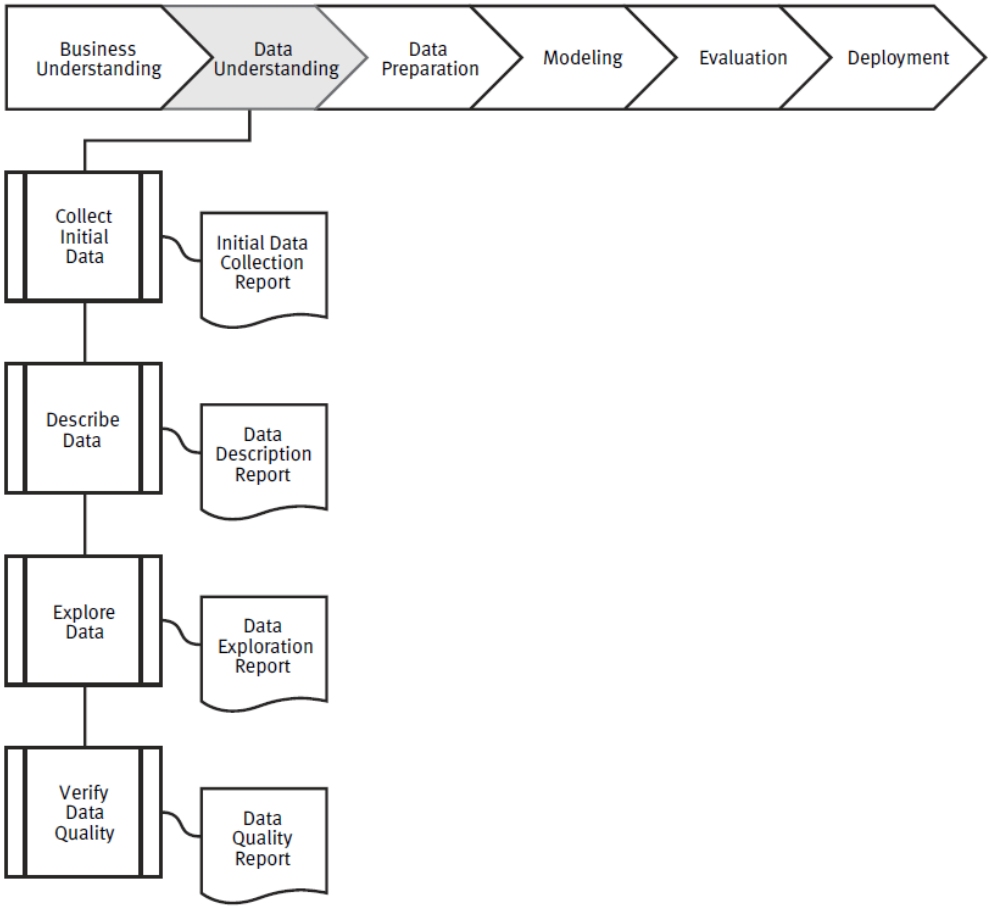



Business Understanding
Determine Business Objectives
Background
Business Objectives
Business Success Criteria
Assess Situation
Inventory of Resources
Requirements, Assumptions, and Constraints
Risks and Contingencies
Terminology
Costs and Benefits
Determine Data Mining Goals
Data Mining Goals
Data Mining Success Criteria
Produce Project Plan
Project Plan
Initial Assessment of Tools and Techniques
Data Understanding
Collect Initial Data
Initial Data Collection Report
Describe Data
Data Description Report
Explore Data
Data Exploration Report
Verify Data Quality
Data Quality Report
Data Preparation
Select Data
Rationale for Inclusion/ Exclusion
Clean Data
Data Cleaning Report
Construct Data
Derived Attributes
Generated Records
Integrate Data
Merged Data
Format Data
Reformatted Data
Dataset
Dataset Description
Modeling
Select Modeling Techniques
Modeling Technique
Modeling Assumptions
Generate Test Design
Test Design
Build Model
Parameter Settings
Models
Model Descriptions
Assess Model
Model Assessment
Revised Parameter Settings
Evaluation
Evaluate Results
Assessment of Data Mining Results w.r.t. Business Success Criteria
Approved Models
Review Process
Review of Process
Determine Next Steps
List of Possible Actions
Decision
Deployment
Plan Deployment
Deployment Plan
Plan Monitoring and Maintenance
Monitoring and Maintenance Plan
Produce Final Report
Final Report
Final Presentation
Review Project
Experience Documentation

Business understanding



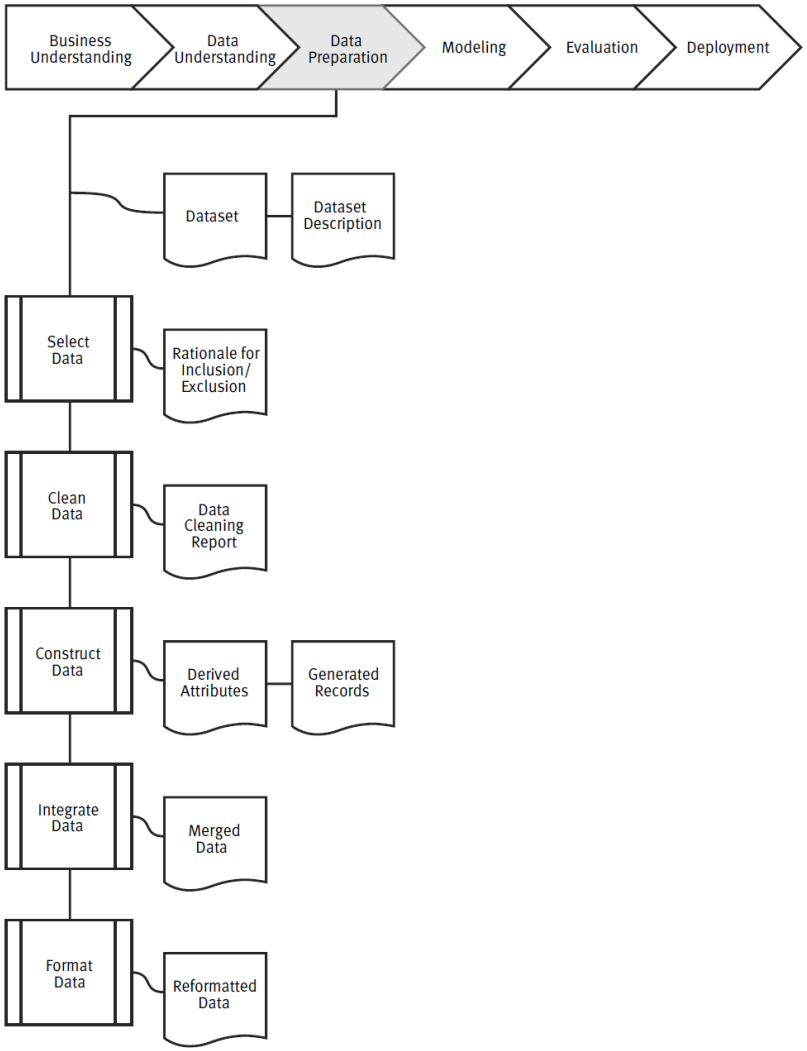
Data understanding



- Structured
- Semi-structured
- Unstructured
- Categorical
  - Nominal
    - Binary
  - Ordinal
- Numeric
  - Interval
  - Ratio
- Discrete
- Continuous

- Descriptive statistics
  - Count, mean, mode, standard deviation, quantiles, etc.
- Class distribution
- Feature skewness
- Correlations between features
  - Chi-square
  - Correlation
- Data visualization

Data preparation



Data preprocessing

Data cleaning

- **Missing values**
  - Use the most probable value to fill in the missing value (and five other methods)
- **Noisy data**
  - Binning; regression; clustering

Data integration

- **Entity ID problem**
  - Metadata
- **Redundancy**
  - Correlation analysis (Correlation coefficient, chi-square test)

Data trasnformation

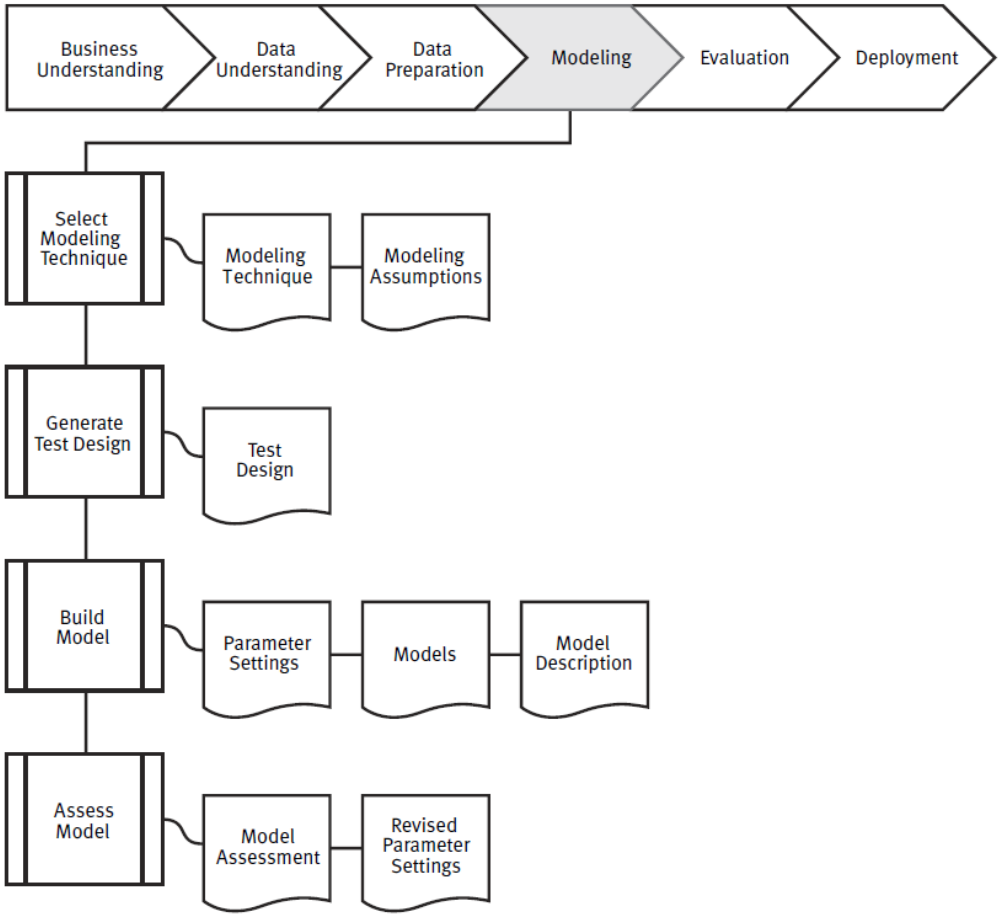
- **Smoothing**
  - **Data cleaning**
- **Aggregation**
  - **Data reduction**
- **Generailization**
  - **Data reduction**
- **Normalization**
  - Min-max; z-score; decimal scaling
- **Attribute construction**
  - New attributes are constructed and added from the given set of attributes to help mining process

Data reduction

- **Data cube aggregation**
  - Data cube store multidimensional aggregated information
- **Attribute selection**
  - Stepwise forward selection; stepwise backward selection; combination; decision tree induction
- **Dimensionality reduction**
  - Discrete wavelet transforms (DWT); Principle components analysis (PCA)
- **Numerosity reduction**
  - Parametric data reduction; clustering; data cube aggregation;
- **Data discretization**
  - Binning; historgram analysis; entropy-based discretization;
    Interval merging by chi-square analysis; cluster analysis; intuitive partitioning
- **Concept hierarchy**
  - Concept hierarchy generation

- Feature Engineering
  - Curse of dimensionality
  - Feature selection
    - Filter methods
    - Wrapper methods
    - Embedded methods
    - Hybrid methods
  - Feature construction
  - Feature creation

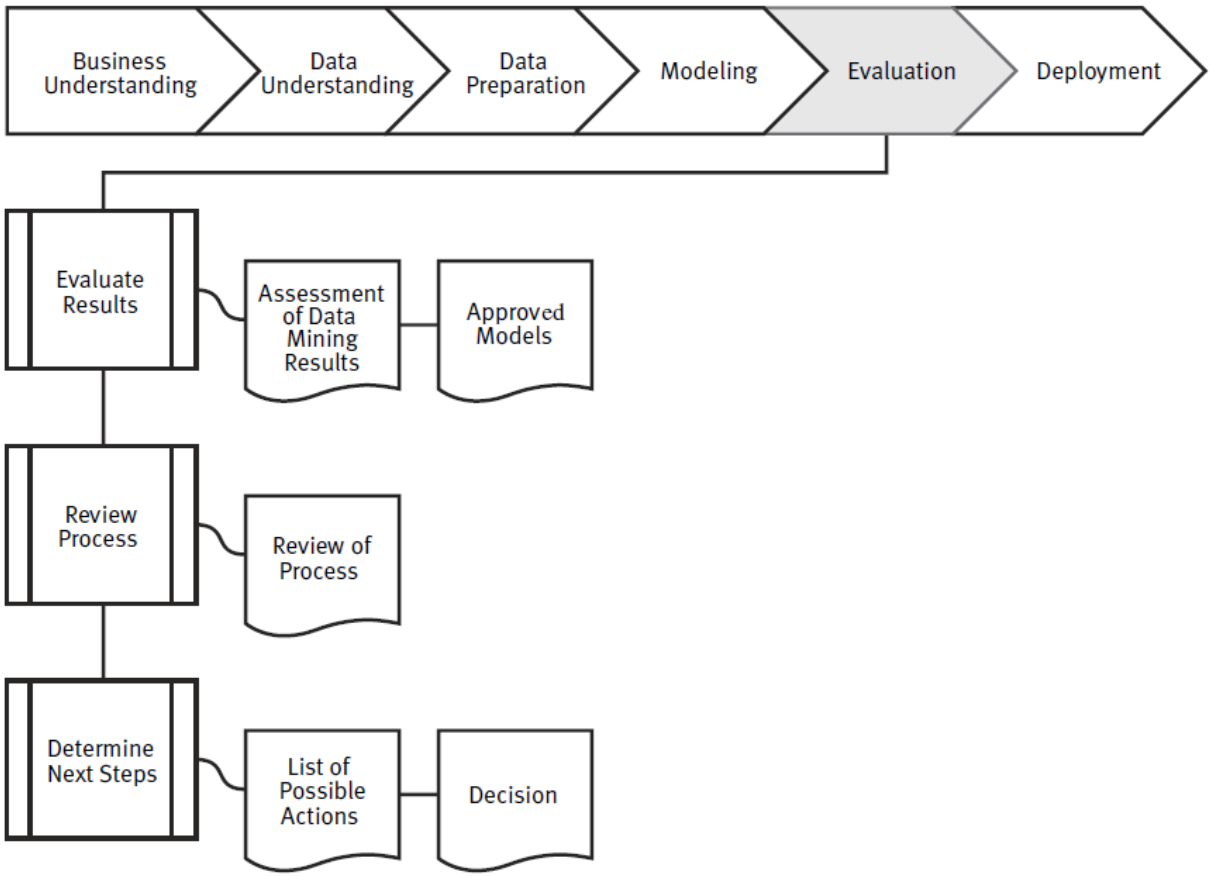
Modeling


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Select Modeling Technique
Modeling Technique
Modeling Assumptions
Generate Test Design
Test Design
Build Model
Parameter Settings
Models
Model Description
Assess Model
Model Assessment
Revised Parameter Settings

- Decision tree
- Bayesian classifier
- Regression
  - Simple linear regression
  - Multiple linear regression
  - Polynomial regression
  - Logistic regression
- k-Nearest Neighbor classifier (kNN)
- Perception
- Gradient Descent
- Neural Network
  - Feed-Forward Neural Network (FNN)
  - Backpropagation
  - Recurrent Neural Network (RNN)
  - Convolutional Neural Network (CNN)
  - Probabilistic Neural Network (PNN)
- Support Vector Machine (SVM)
- Binary Classification
- Multiclass Classification
- Rule-Based Classification
- Bayesian Belief Networks
- Genetic Algorithms
- Rough Set Approach
- Fuzzy Set Approaches
- Etc.

- Multidimensional Data Modeling
  - Data Cube: A Multidimensional Data Model
  - Stars, Snowflakes, and Fact Constellations
  - Dimensions: The Role of Concept Hierarchies
  - OLAP Operations
- Mining Frequent Patterns, Associations, and Correlations
  - Apriori Algorithm
  - Frequent Itemset
  - Generating Association Rules from Frequent Itemsets
- Clustering
  - Partitional methods
    - k-means
    - k-medoids
  - Hierarchical methods
    - Agglomerative hierarchical clustering method
    - Divisive hierarchical clustering method
  - Density-Based Methods
    - DBSCAN: Density-Based Clustering Based on Connected Regions with High Density
    - OPTICS: Ordering Points to Identify the Clustering Structure
    - DENCLUE: Clustering Based on Density Distribution Functions
  - Grid-Based Methods
    - STING: STatistical INformation Grid
    - CLIQUE: An Apriori-like Subspace Clustering Method

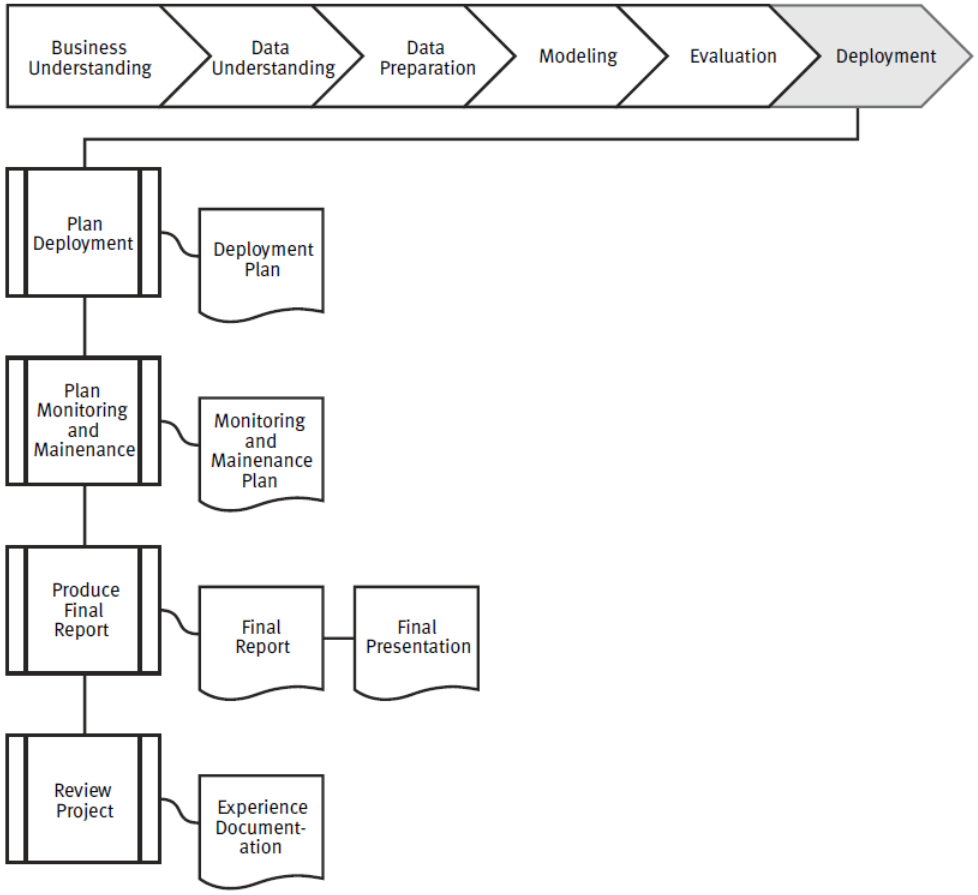
Evaluation


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Evaluate Results
Assessment of Data Mining Results
Approved Models
Review Process
Review of Process
Determine Next Steps
List of Possible Actions
Decision

confusion matrix
accuracy, recognition rate
error rate, misclassification rate
sensitivity, true positive rate, recall
specificity, true negative rate, precision
F1-score, harmonic mean of precision and recall
F-beta , where is a non-negative real number
k-fold cross-validation
bootstrap
.632 bootstrap
model selection using statistical tests of significance
comparing classifiers based on cost–benefit and ROC curves

Ensemble Methods
- Random Forests
- Bagging
- Boosting
- AdaBoost

Deployment


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Plan Deployment
Deployment Plan
Plan Monitoring and Mainenance
Monitoring and Mainenance Plan
Produce Final Report
Final Report
Final Presentation
Review Project
Experience Document-ation

Lesson A-2

Getting to Know Your Data

- Structure, Types, Attributes

CRISP-DM

- **Cross-industry standard process for data mining**
  - an open standard process model that describes common approaches used by data mining experts.
  - It is the most widely-used analytics model

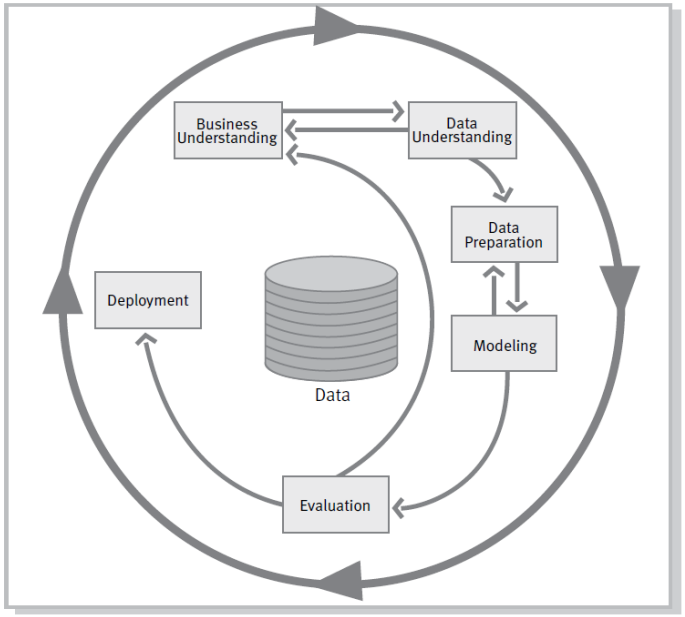
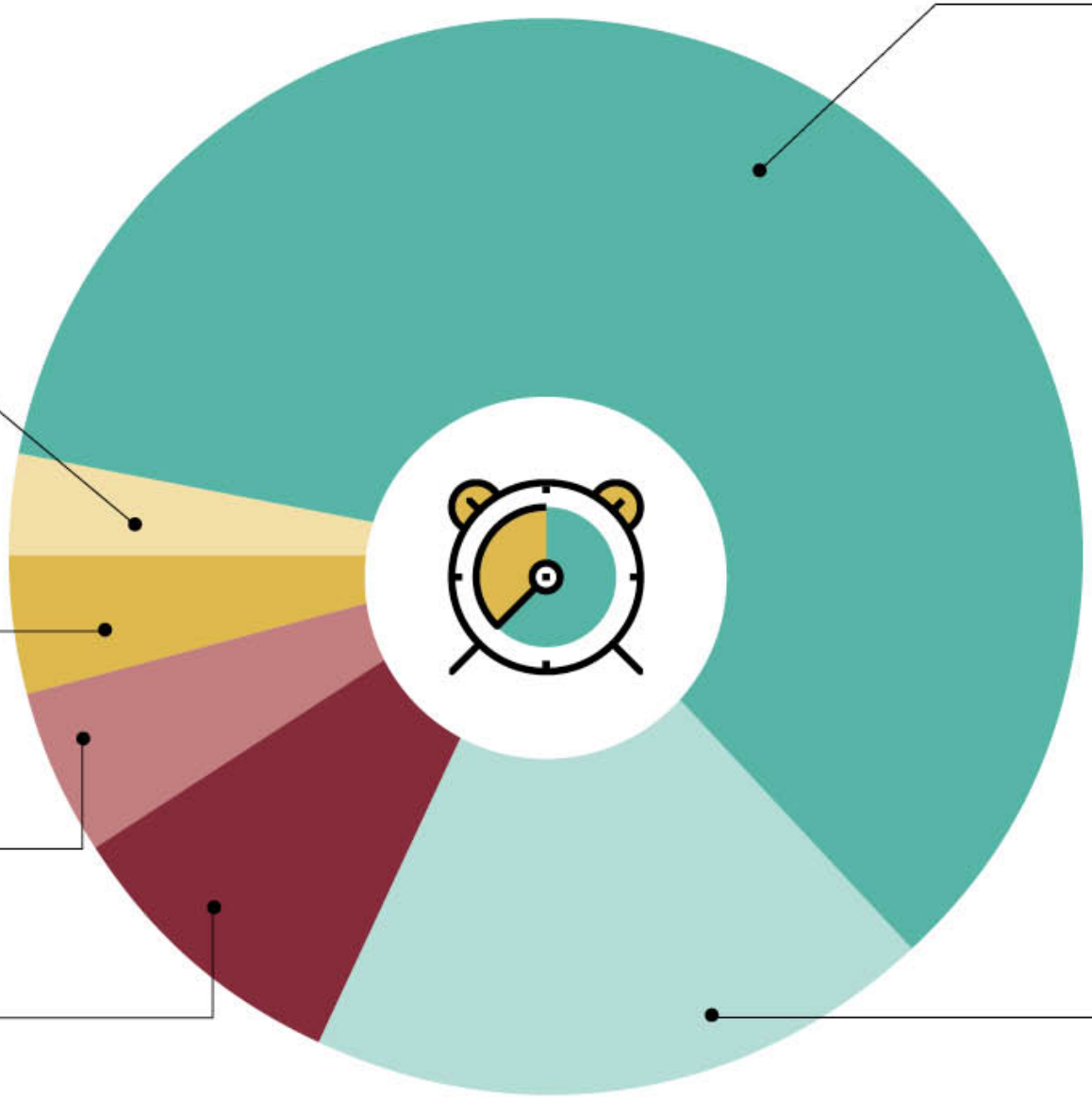
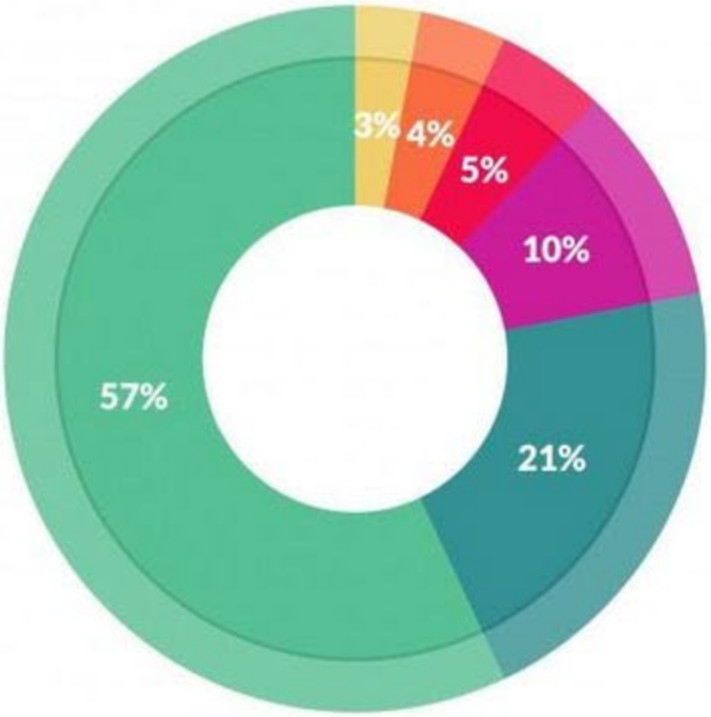


Figure 2: Phases of the CRISP-DM reference model

WHAT DATA SCIENTISTS SPEND THE MOST TIME DOING



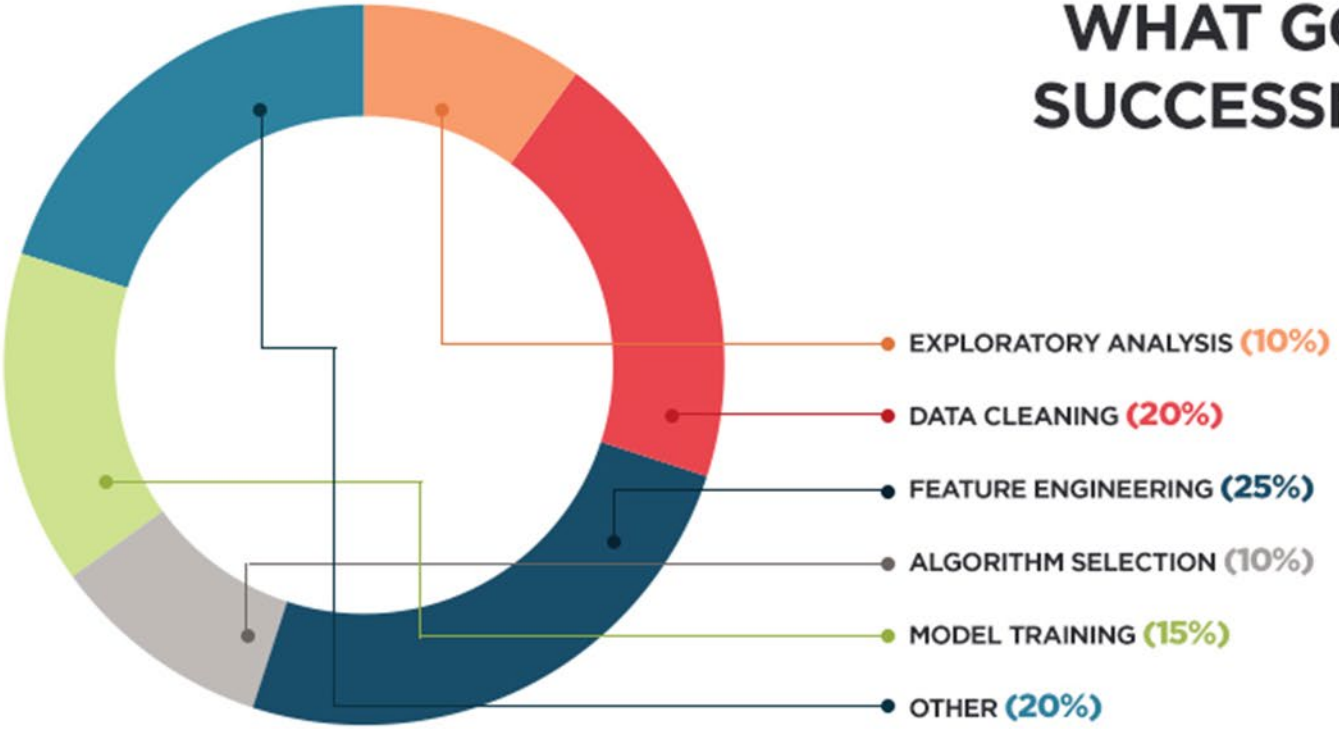
Source: CrowdFlower 2016



What's the least enjoyable part of data science?

- *Building training sets: 10%*
- *Cleaning and organizing data: 57%*
- *Collecting data sets: 21%*
- *Mining data for patterns: 3%*
- *Refining algorithms: 4%*
- *Other: 5%*

WHAT GOES INTO A SUCCESSFUL MODEL



DATA SCIENCE PRIMER
by EliteDataScience

Data and Attribute

- Data
  - is a set of fact or values (each value is known as a datum) of an entity (object, subject)
  - Structured data is described by a set of attributes
  - Unstructured data is not described by attributes
  - Semi-structured data is a form of structured data that does not defined by the formal form
    - It contains tags and other markers to separate elements
- Attribute
  - Data is described by a number of attribute
  - A set of Attributes used to describe a given data is called a attribute vector
    - Customer (customer_ID, names, address, age, gender, race, income, etc.)

Data and Attribute

- Data represents an entity
  - Samples, example, instances, data points, tuples, data tuples, record, event, case, vector, observation, entity, row, data objects, or objects are often used interchangeably
- A attribute represents a characteristic or attribute of an instance
  - feature, dimension, attribute, characteristic, field, column, and variable are often used interchangeably

Data Object/Attribute

- A data object represents an entity
  - Datasets made up of data object
  - Samples, examples, instances, data points, tuples, data tuples, record, event, case, vector, observation, entity, row, or objects are often used interchangeably
  - Record-based data sets are common, either in flat files ore relational database, there are other types of data sets for storing data.
- An **attribute** is a data field, representing a characteristic or attribute of a data object
  - feature, dimension, attribute, characteristic, field, column, and *variable* are often used interchangeably
  - The rows of a database (dataset) correspond to the data objects, and the columns correspond to the attributes.

Dataset/Database/DBMS

- Dataset
  - is a collection of data (instance)
- Database
  - is an organized collection of data, generally stored in tables and accessed from a database management system
  - The rows of a database (dataset) correspond to data objects, and the columns correspond to attributes
- Database management system
  - is a software system that enables users to define, create, maintain and control access to the database

Types of Datasets

- Record
  - Record data
  - Transaction data
  - Data Matrix
  - Document data
- Ordered
  - Sequential data
  - Sequence data
  - Spatial data
  - Temporal data
  - Time-series data
- Graph/Structure
  - World Wide Web
  - Social Network
  - Molecular Structures

Record Data

- Data that consists of a collection of records, each of which consists of a fixed set of attributes

Tid	Refund	Marital Status	Taxable Income	Cheat
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

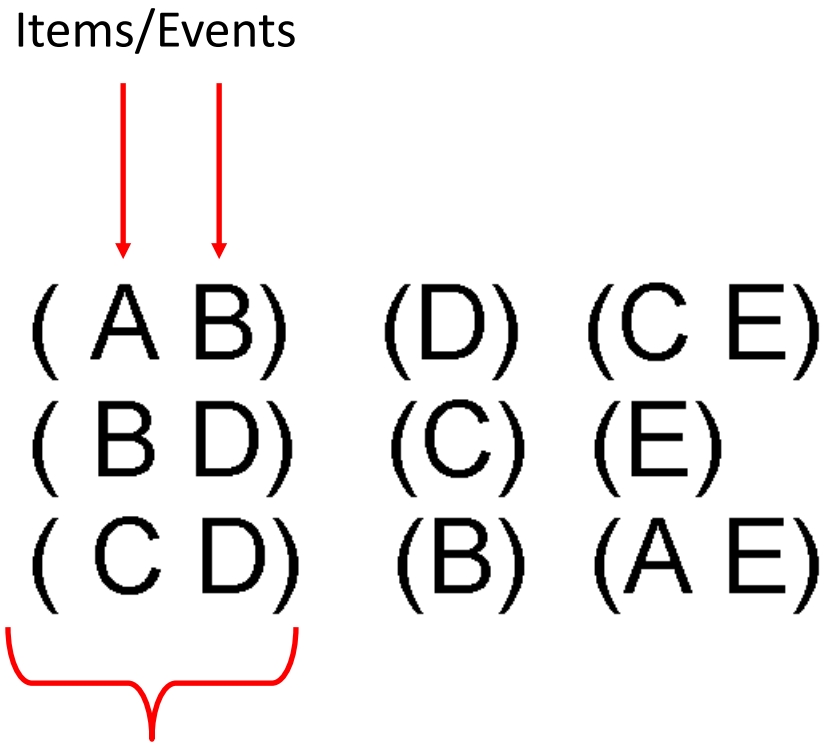
Transaction Data

- A special type of record data, where
  - each record (transaction) involves a set of items.
  - For example, consider a grocery store. The set of products purchased by a customer during one shopping trip constitute a transaction, while the individual products that were purchased are the items.

TID	*Items*
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Ordered Data

- Sequence



Ordered Data

- Genomic sequence data

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

Sequential Data

Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

Load and view the dataset

- pandas.DataFrame
  - Two-dimensional, size-mutable, potentially heterogeneous tabular data, labeled axes (rows and columns)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
import pandas as pd
data = pd.read_csv("diabetes.csv") # returns a
                                   # pandas.DataFrame
data.head()
data.head(10)
```

View the first 5 rows

```
# https://www.kaggle.com/san-francisco/sf-salary-ranges-by-job-classification

salary_ranges = pd.read_csv('salary-ranges-by-job-classification.csv')
```

```
salary_ranges.head()
```

:

	SetID	Job Code	Eff Date	Sal End Date	Salary SetID	Sal Plan	Grade	Step	Biweekly High Rate	Biweekly Low Rate	Union Code	Extended Step	Pay Type
0	COMMN	109	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	0.0	0.0	330	0	C
1	COMMN	110	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	15.0	15.0	323	0	D
2	COMMN	111	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	25.0	25.0	323	0	D
3	COMMN	112	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	50.0	50.0	323	0	D
4	COMMN	114	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	100.0	100.0	323	0	M

2.3-2.4 Shape of data set and attribute type

```
data.shape # shape of dataframe (number of rows and columns)
```

```
(768, 9)
```

```
data.info() # summary of dataframe
            # number rows,  name of feature,  number of non-null items, type of a feature
            # in each column
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
data.dtypes
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

Type of attribute (Data)

- Categorical (Qualitative) data
  - Are categorical in nature. They describe the quality of something (someone)
    - Nominal data
    - Ordinal data
- Numeric (Quantitative) data
  - Are numerical in nature. They measure the quantity of something (someone)
    - Interval data
    - Ratio data

```
salary_ranges.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1356 entries, 0 to 1355
Data columns (total 13 columns):
SetID                 1356 non-null object
Job Code              1356 non-null object
Eff Date              1356 non-null object
Sal End Date          1356 non-null object
Salary SetID          1356 non-null object
Sal Plan              1356 non-null object
Grade                 1356 non-null object
Step                  1356 non-null int64
Biweekly High Rate    1356 non-null float64
Biweekly Low Rate     1356 non-null float64
Union Code            1356 non-null int64
Extended Step         1356 non-null int64
Pay Type              1356 non-null object
dtypes: float64(2), int64(3), object(8)
memory usage: 137.8+ KB
```

```
salary_ranges.describe() # the describe method checks out some descriptive statistics of
                         # quantitative columns
```

:

	Step	Biweekly High Rate	Biweekly Low Rate	Union Code	Extended Step
count	1356.000000	1356.000000	1356.000000	1356.000000	1356.000000
mean	1.294985	3161.727021	3754.652006	392.676991	0.150442
std	1.045816	1481.002904	1605.157054	338.100562	1.006734
min	1.000000	0.000000	0.000000	1.000000	0.000000
25%	1.000000	2145.000000	2607.000000	21.000000	0.000000
50%	1.000000	2856.500000	3465.000000	351.000000	0.000000
75%	1.000000	3703.000000	4484.000000	790.000000	0.000000
max	5.000000	12120.770000	12120.770000	990.000000	11.000000

Nominal attribute

- Characteristics
  - Purely described by name
  - Nominal data is still categorical in nature, even if numbers are used to represent the categories
  - The categories do not have ordering or ranking
  - Numbers are used to represent the categories
- Mathematical operators
  - Distinctness: $=$ $\neq$
- Descriptive statistics
  - count (frequency), mode, percentage
- Visualization
  - bar charts, pie chart

Ordinal attribute

- Inherits all of the properties of nominal data, and has additional properties
  - Data can be ordered
  - This implies that data can be considered better than or greater than other
  - Like nominal data, ordinal data is still categorical in nature, even if numbers are used to represent the categories
- Mathematical operators
  - Distinctness: $=$ $\neq$
  - Order: $<$ $>$
- Descriptive statistics
  - count (frequency), mode, percentage, median
- Visualization
  - bar charts, pie chart, box plots

Interval Data

- Categorical data cannot describe a “true” quantity
  - We can work with interval data that not only has ordering but also meaningful differences between values
- Mathematical operators
  - Distinctness: = ≠
  - Order: < >
  - Differences: - +
- Descriptive statistics
  - count (frequency), mode, percentage, median, mean, standard deviation
- Visualization
  - Histogram, plotting (line, box, or scatter)

Ratio Data

- Like interval data we can add and subtract ratio data
- We can not only add and subtract values but also multiply and divide values because ration data has a true zero.
- Mathematical operators
  - Distinctness: = ≠
  - Order: < >
  - Differences: - +
  - Ratio: * /
- Descriptive statistics
  - count (frequency), mode, percentage, median, mean, standard deviation, harmonic mean, geometric mean
- Visualization
  - Histogram, Box plot

Types of attribute

- Nominal
  - Examples: ID numbers, eye color, zip codes
- Ordinal
  - Examples: rankings (e.g., taste of potato chips on a scale from 1-10), grades, height in {tall, medium, short}
- Interval
  - Examples: temperatures in Celsius or Fahrenheit, calendar date
- Ratio
  - Examples: temperature in Kelvin, length, time, counts

Properties of attribute Values

- The type of a attribute depends on which of the following properties it possesses:
  - Distinctness: = ≠
  - Order: < >
  - Addition: + -
  - Multiplication: * /

  - Nominal attribute : distinctness
  - Ordinal attribute : distinctness & order
  - Interval attribute : distinctness, order & addition
  - Ratio attribute : all 4 properties

```
# https://www.kaggle.com/san-francisco/sf-salary-ranges-by-job-classification

salary_ranges = pd.read_csv('salary-ranges-by-job-classification.csv')
```

```
salary_ranges.head()
```

:

	SetID	Job Code	Eff Date	Sal End Date	Salary SetID	Sal Plan	Grade	Step	Biweekly High Rate	Biweekly Low Rate	Union Code	Extended Step	Pay Type
0	COMMN	109	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	0.0	0.0	330	0	C
1	COMMN	110	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	15.0	15.0	323	0	D
2	COMMN	111	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	25.0	25.0	323	0	D
3	COMMN	112	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	50.0	50.0	323	0	D
4	COMMN	114	2009-07-01T00:00:00	2010-06-30T00:00:00	COMMN	SFM	0	1	100.0	100.0	323	0	M

```
salary_ranges.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1356 entries, 0 to 1355
Data columns (total 13 columns):
SetID                 1356 non-null object
Job Code              1356 non-null object
Eff Date              1356 non-null object
Sal End Date          1356 non-null object
Salary SetID          1356 non-null object
Sal Plan              1356 non-null object
Grade                 1356 non-null object
Step                  1356 non-null int64
Biweekly High Rate    1356 non-null float64
Biweekly Low Rate     1356 non-null float64
Union Code            1356 non-null int64
Extended Step         1356 non-null int64
Pay Type              1356 non-null object
dtypes: float64(2), int64(3), object(8)
memory usage: 137.8+ KB
```

```
salary_ranges.describe() # the describe method checks out some descriptive statistics of
                         # quantitative columns
```

	Step	Biweekly High Rate	Biweekly Low Rate	Union Code	Extended Step
count	1356.000000	1356.000000	1356.000000	1356.000000	1356.000000
mean	1.294985	3161.727021	3754.652006	392.676991	0.150442
std	1.045816	1481.002904	1605.157054	338.100562	1.006734
min	1.000000	0.000000	0.000000	1.000000	0.000000
25%	1.000000	2145.000000	2607.000000	21.000000	0.000000
50%	1.000000	2856.500000	3465.000000	351.000000	0.000000
75%	1.000000	3703.000000	4484.000000	790.000000	0.000000
max	5.000000	12120.770000	12120.770000	990.000000	11.000000

```
salary_ranges['Grade'].value_counts()
```

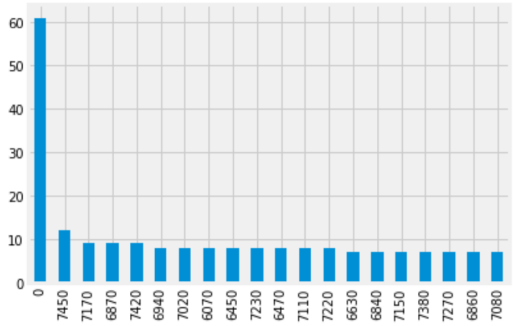
```
0        61
7450     12
7170      9
6870      9
7420      9
         ..
7685      1
2454F     1
1958F     1
Q3H00     1
H10H0     1
Name: Grade, Length: 688, dtype: int64
```

```
salary_ranges['Grade'].describe() # Grade is a categorical data
```

```
count     1356
unique     688
top          0
freq        61
Name: Grade, dtype: object
```

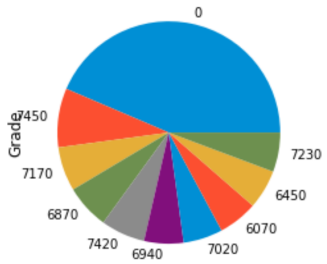
```
# Bar Chart of the Grade colomn
salary_ranges['Grade'].value_counts().sort_values(ascending = False).head(20).plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa7064cb48>
```




```
# Pie Chart of Grade column
salary_ranges['Grade'].value_counts().sort_values(ascending = False).head(10).plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa709a57c8>
```



```
# Load in the SFO dataset
customer = pd.read_csv('2013-sfo-customer-survey.csv')
```

```
customer.head()
```

	RESPNUM	CCGID	RUN	INTDATE	GATE	STRATA	PEAK	METHOD	AIRLINE	FLIGHT	...	Q17_COUNTRY	HOME	Q18_AGE	Q19_SEX	Q20_INCOME
0	1	1.0	1215	2	12	1	1	1	21	1437	...	US	1	2	1	1
1	2	2.0	1215	2	12	1	1	1	21	1437	...	US	5	6	1	0
2	3	3.0	1215	2	12	1	1	1	21	1437	...	US	1	4	2	2
3	4	4.0	1215	2	12	1	1	1	21	1437	...	US	90	4	1	2
4	5	5.0	1215	2	12	1	1	1	21	1437	...	US	10	3	1	3

5 rows × 95 columns

```
customer.shape
```

```
(3535, 95)
```

```
# focus on Q7A_ART
# Field Name: Q7A_ART
# Definition: Artwork and exhibitions
# Data Type: Ordinal
# Possible Field Values (if Fixed): 0,1,2,3,4,5,6
# Data Business Rules / Comments: 1=Unacceptable, 2=Below Average, 3=Average,
# 4=Good, 5=Outstanding, 6=Have Never Used or Visited,0=Blank

art_ratings = customer['Q7A_ART']
```

```
art_ratings.describe()
```

```
count    3535.000000
mean        4.300707
std         1.341445
min         0.000000
25%         3.000000
50%         4.000000
75%         5.000000
max         6.000000
Name: Q7A_ART, dtype: float64
```

```
]:  art_ratings = art_ratings.astype(str) # cast the values as strings
```

```
]:  art_ratings.describe()
```

```
:[54]: count     2656
       unique       5
       top          4
       freq      1066
       Name: Q7A_ART, dtype: object
```

```
]:  art_ratings.count()
```

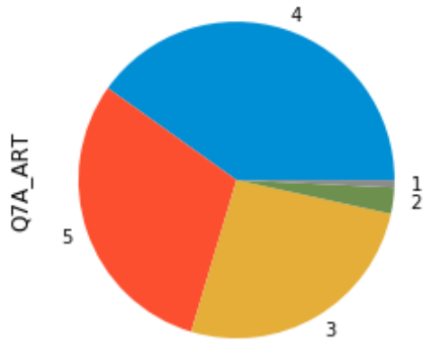
```
:[59]: 2656
```

```
]:  art_ratings.value_counts()
```

```
:[63]: 4    1066
       5     803
       3     696
       2      71
       1      20
       Name: Q7A_ART, dtype: int64
```

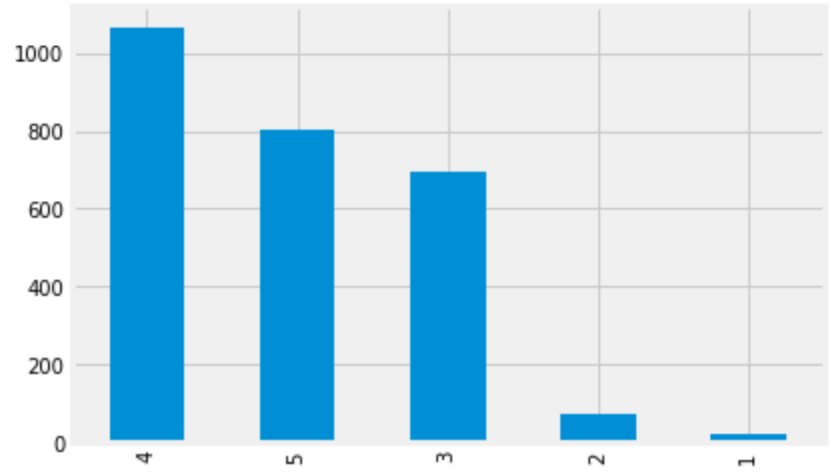
```
]:  art_ratings.value_counts().plot(kind='pie')
```

```
:[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa70aa5408>
```



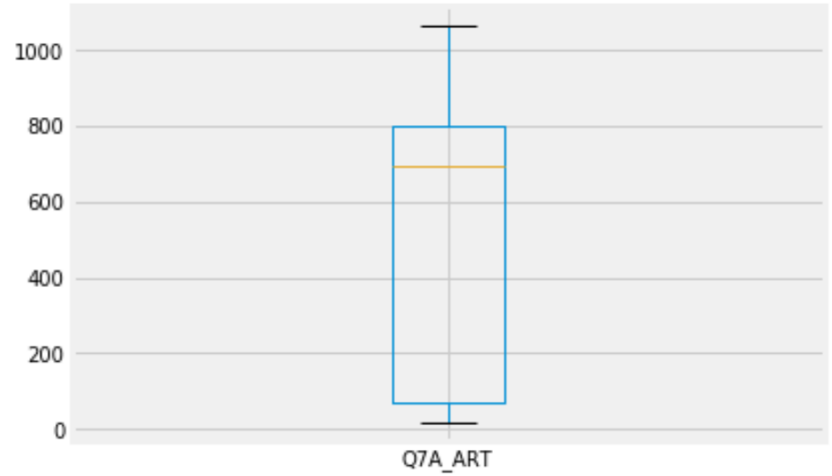
```
art_ratings.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa70b29208>
```




```
art_ratings.value_counts().plot(kind='box')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa70be0288>
```



```
climate.head()
```

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
0	1743-11-01	6.068	1.737	Århus	Denmark	57.05N	10.33E
1	1743-12-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
2	1744-01-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
3	1744-02-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
4	1744-03-01	NaN	NaN	Århus	Denmark	57.05N	10.33E

```
climate.shape
```

```
(8599212, 7)
```

```
# 21st century average temp in US minus 18th century average temp in US
century_changes[21] - century_changes[18]
```

```
3.124449115460754
```

```
salary_ranges.groupby('Grade')[['Biweekly High Rate']].mean().sort_values('Biweekly High Rate', asc
```

:

Grade	Biweekly High Rate
9186F	12120.77
0390F	11255.00
0140H	10843.00
0140F	10630.00
0395F	10376.00

```
salary_ranges.groupby('Grade')[['Biweekly High Rate']].mean().sort_values('Biweekly High Rate', asc
```

:

Grade	Biweekly High Rate
5160	1141.0
5030	1073.0
9916F	920.0
4660	899.0
4590	870.0

```
sorted_df = salary_ranges.groupby('Grade')[['Biweekly High Rate']].mean().sort_values('Biweekly High Rate', ascending=False)
```

```
# the ratio of the highest-paid employee (Grade) to # the lowest-paid employee (Grade)
sorted_df.iloc[0][0] / sorted_df.iloc[-1][0]
```

]: 13.931919540229886

	attribute type	properties	examples	descriptive statistics	graph
categorical (qualitative)	**nominal**	Names, information to distinguish (compare) one object from another (= , $\neq$)	Names of employee, zip codes, employee ID, eye color, gender	frequency, percentage, mode	Bar Pie
	ordinal	The value of an ordinal attribute provide enough information to order objects ($<$, $>$)	GPA, professor rank, Likert scales,	frequency, percentage, mode, median	Bar Pie Boxplot

	attribute type	properties	examples	descriptive statistics	graph
numeric (quantitative)	**interval**	Differences between values are meaningful. (+, -)	calendar date, temperature in Celsius or Fahrenheit	frequency, percentage, mode mean median standard deviation	Bar Pie Box plot Density plot Histogram
	ratio	True 0 allows ratio statements (*, /)	temperature in Kelvin, time, money, counts, age, weight, length, electrical current,	frequency, percentage, mode mean median standard deviation	Bar Pie Boxplot Histogram Density plot

Discrete vs Continuous Data

- Discrete data
  - finite or countable infinite set of values
    - *hair color*, *smoker*, *medical test*
    - *Customer_ID, zip codes*
- Continuous data
  - If a data is not discrete, it is continuous.
    - Temperature, height, speed
    - The terms *numeric data* and *continuous data* are often used interchangeably in the literature.

Lesson A-3

Getting to Know Your Data

- Data Exploration

CRISP-DM

- **Cross-industry standard process for data mining**
  - an open standard process model that describes common approaches used by data mining experts.
  - It is the most widely-used analytics model

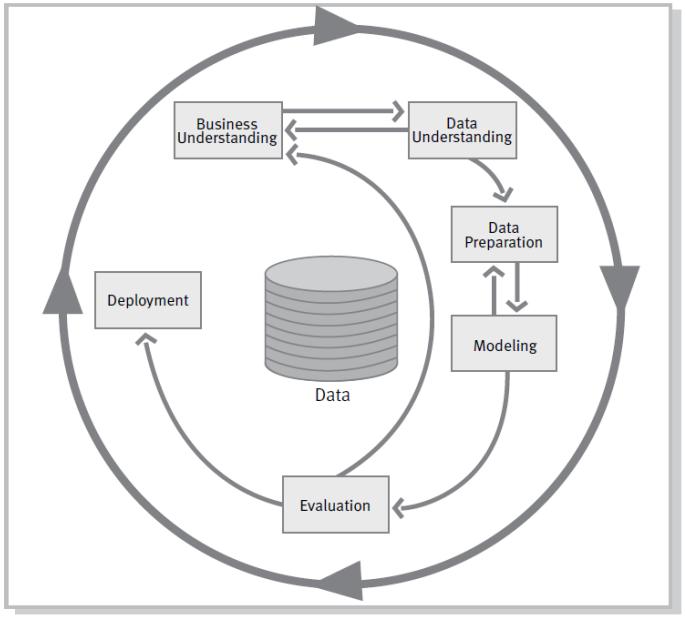
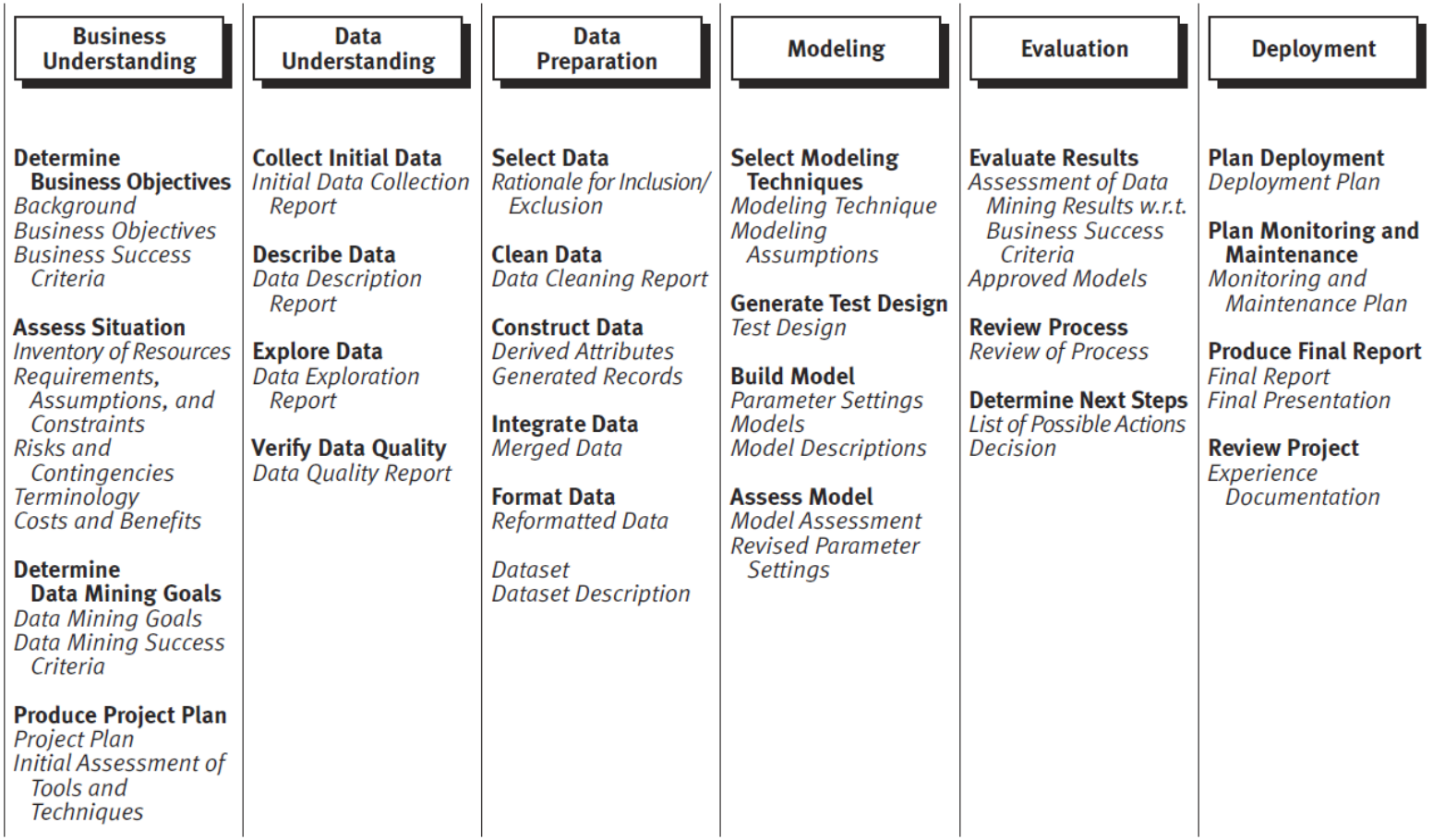


Figure 2: Phases of the CRISP-DM reference model

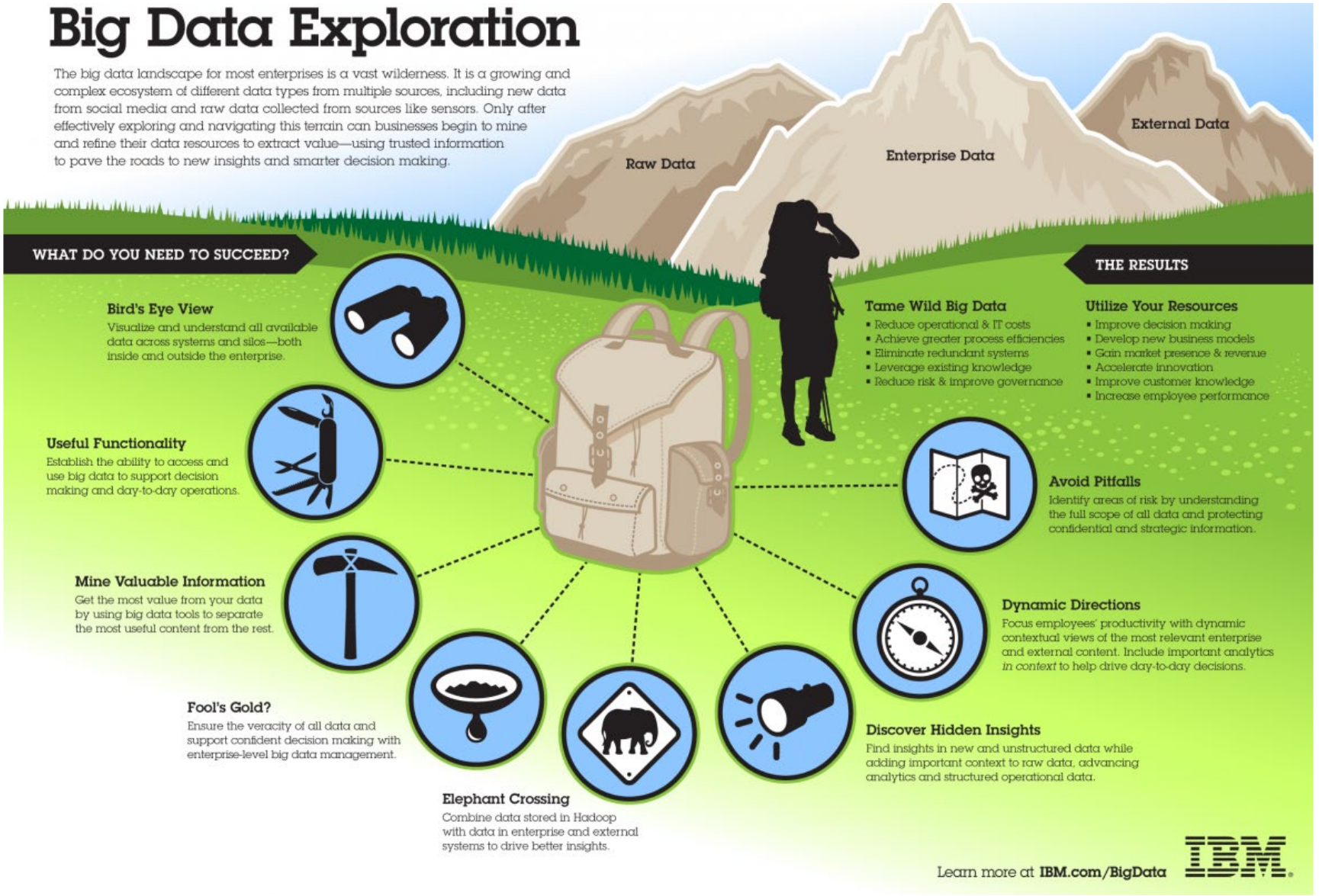

Business Understanding
Determine Business Objectives
Background
Business Objectives
Business Success Criteria
Assess Situation
Inventory of Resources
Requirements, Assumptions, and Constraints
Risks and Contingencies
Terminology
Costs and Benefits
Determine Data Mining Goals
Data Mining Goals
Data Mining Success Criteria
Produce Project Plan
Project Plan
Initial Assessment of Tools and Techniques
Data Understanding
Collect Initial Data
Initial Data Collection Report
Describe Data
Data Description Report
Explore Data
Data Exploration Report
Verify Data Quality
Data Quality Report
Data Preparation
Select Data
Rationale for Inclusion/ Exclusion
Clean Data
Data Cleaning Report
Construct Data
Derived Attributes
Generated Records
Integrate Data
Merged Data
Format Data
Reformatted Data
Dataset
Dataset Description
Modeling
Select Modeling Techniques
Modeling Technique
Modeling Assumptions
Generate Test Design
Test Design
Build Model
Parameter Settings
Models
Model Descriptions
Assess Model
Model Assessment
Revised Parameter Settings
Evaluation
Evaluate Results
Assessment of Data Mining Results w.r.t. Business Success Criteria
Approved Models
Review Process
Review of Process
Determine Next Steps
List of Possible Actions
Decision
Deployment
Plan Deployment
Deployment Plan
Plan Monitoring and Maintenance
Monitoring and Maintenance Plan
Produce Final Report
Final Report
Final Presentation
Review Project
Experience Documentation

What is Data Exploration

- *The purpose of exploratory analysis is to **"get to know"** the dataset.*
- Data exploration addresses data mining projects using statistics and visualization to investigate each type of attributes of a dataset

Big Data Exploration

The big data landscape for most enterprises is a vast wilderness. It is a growing and complex ecosystem of different data types from multiple sources, including new data from social media and raw data collected from sources like sensors. Only after effectively exploring and navigating this terrain can businesses begin to mine and refine their data resources to extract value—using trusted information to pave the roads to new insights and smarter decision making.



Learn more at **IBM.com/BigData**

IBM

Understand data

- Load dataset
- View dataset
- Shape of dataset
- Attribute types
- Statistical Techniques
  - Data summary based on statistical description
  - attribute skewness
  - Class distribution
  - Correlations between attributes
- Data visualization
  - bar charts, histograms, frequency polygons, pie charts, scatter plot , heatmap

Load and view the dataset

- pandas.DataFrame
  - Two-dimensional, size-mutable, potentially heterogeneous tabular data, labeled axes (rows and columns)

]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Shape of data set and attribute type

```
data.shape # shape of dataframe (number of rows and columns)
```

```
: (768, 9)
```

```
data.dtypes
```

```
]: Pregnancies                 int64
   Glucose                     int64
   BloodPressure               int64
   SkinThickness               int64
   Insulin                     int64
   BMI                       float64
   DiabetesPedigreeFunction  float64
   Age                         int64
   Outcome                     int64
   dtype: object
```

Shape of data set and attribute type

```
data.info() # summary of dataframe
            # number rows,  name of feature,  number of non-null items, type of a feature
            # in each column
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Type of Attribute

- Categorical (Qualitative) data
  - Are categorical in nature. They describe the quality of something (someone)
    - Nominal data
    - Ordinal data
- Numeric (Quantitative) data
  - Are numerical in nature. They measure the quantity of something (someone)
    - Interval data
    - Ratio data

	attribute type	properties	examples	descriptive statistics	graph
categorical (qualitative)	**nominal**	Names, information to distinguish (compare) one object from another (= , $\neq$)	Names of employee, zip codes, employee ID, eye color, gender	frequency, percentage, mode	Bar Pie
	ordinal	The value of an ordinal attribute provide enough information to order objects ($<$, $>$)	GPA, professor rank, Likert scales,	frequency, percentage, mode, median	Bar Pie Boxplot

	attribute type	properties	examples	descriptive statistics	graph
numeric (quantitative)	**interval**	Differences between values are meaningful. (+, -)	calendar date, temperature in Celsius or Fahrenheit	frequency, percentage, mode mean median standard deviation	Bar Pie Box plot Density plot Histogram
	ratio	True 0 allows ratio statements (*, /)	temperature in Kelvin, time, money, counts, age, weight, length, electrical current,	frequency, percentage, mode mean median standard deviation	Bar Pie Boxplot Histogram Density plot

Data summary based on statistical description

`count`	Number of non-NA values
`describe`	Compute set of summary statistics for Series or each DataFrame column
`min, max`	Compute minimum and maximum values
`argmin, argmax`	Compute index locations (integers) at which minimum or maximum value obtained, respectively
`idxmin, idxmax`	Compute index labels at which minimum or maximum value obtained, respectively
`quantile`	Compute sample quantile ranging from 0 to 1
`sum`	Sum of values
`mean`	Mean of values
`median`	Arithmetic median (50% quantile) of values
`mad`	Mean absolute deviation from mean value
`prod`	Product of all values
`var`	Sample variance of values
`std`	Sample standard deviation of values
`skew`	Sample skewness (third moment) of values
`kurt`	Sample kurtosis (fourth moment) of values
`cumsum`	Cumulative sum of values
`cummin, cummax`	Cumulative minimum or maximum of values, respectively
`cumprod`	Cumulative product of values
`diff`	Compute first arithmetic difference (useful for time series)
`pct_change`	Compute percent changes

Pandas `describe()` function

- The Pandas `describe()` function lists 8 statistical properties of each “numeric” attribute
  - Count
  - Mean
  - Standard deviation
  - Minimum value
  - 25th percentile
  - 50th percentile (Median)
  - 75th percentile
  - Maximum value

```
data.describe()
```

]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
from pandas import set_option
set_option('precision', 3) # Displays precision for decimal numbers
data.describe()
```

]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000
mean	3.845	120.895	69.105	20.536	79.799	31.993	0.472	33.241	0.349
std	3.370	31.973	19.356	15.952	115.244	7.884	0.331	11.760	0.477
min	0.000	0.000	0.000	0.000	0.000	0.000	0.078	21.000	0.000
25%	1.000	99.000	62.000	0.000	0.000	27.300	0.244	24.000	0.000
50%	3.000	117.000	72.000	23.000	30.500	32.000	0.372	29.000	0.000
75%	6.000	140.250	80.000	32.000	127.250	36.600	0.626	41.000	1.000
max	17.000	199.000	122.000	99.000	846.000	67.100	2.420	81.000	1.000

	attribute type	properties	examples	descriptive statistics	graph
categorical (qualitative)	**nominal**	Names, information to distinguish (compare) one object from another (= , $\neq$)	Names of employee, zip codes, employee ID, eye color, gender	frequency, percentage, mode	Bar Pie
	ordinal	The value of an ordinal attribute provide enough information to order objects ($<$, $>$)	GPA, professor rank, Likert scales,	frequency, percentage, mode, median	Bar Pie Boxplot

	attribute type	**properties**	**examples**	**descriptive statistics**	**graph**
numeric (quantitative)	**interval**	Differences between values are meaningful. (+, -)	calendar date, temperature in Celsius or Fahrenheit	frequency, percentage, mode mean median standard deviation	Bar Pie Box plot Density plot Histogram
	ratio	True 0 allows ratio statements (*, /)	temperature in Kelvin, time, money, counts, age, weight, length, electrical current,	frequency, percentage, mode mean median standard deviation	Bar Pie Boxplot Histogram Density plot

count

- Count
  - Count non-NA cells for each column or row.

```
# count
data.count()
```

```
4]: Pregnancies                768
    Glucose                    768
    BloodPressure              768
    SkinThickness              768
    Insulin                    768
    BMI                        768
    DiabetesPedigreeFunction   768
    Age                        768
    Outcome                    768
    dtype: int64
```

Measuring the Central Tendency - Arithmetic Mean

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

$x_1, x_2, \ldots, x_N$ is a set of N values

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

the mean salary is _____

weighted arithmetic mean or the **weighted average**
Each value *xi* in a set may be associated with a weight w_i for $i = 1, \ldots, N$.

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}.$$

Measuring the Dispersion of Data - Range

- *Let* $x_1, x_2, \ldots, x_N$ be a set of observations for some numeric attribute, **X**
- The range of the set is the difference between the largest value (max) and smallest (min) values

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

the range of the salary is _____

Measuring the Central Tendency – Midrange

- *Let* $x_1, x_2, \ldots, x_N$ be a set of observations for some numeric attribute, X
- It is the average of the largest and smallest values in the dataset.
- This measure is easy to computing using aggregating functions, `max()` and `min()`

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

the midrange of the salary is _____

Measuring the Central Tendency - Median

Median = the middle value of a set of ordered data.

Median = $\{(n + 1) \div 2\}^{th}$ value

n is the number of values

If *n* is an even number, the median is calculated by averaging the two middle values.

Median = (value below median + value above median) ÷ 2

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

the median salary is _____

```
# mean
data.mean()
```

```
2]: Pregnancies                 3.845
    Glucose                   120.895
    BloodPressure              69.105
    SkinThickness              20.536
    Insulin                    79.799
    BMI                        31.993
    DiabetesPedigreeFunction    0.472
    Age                        33.241
    Outcome                     0.349
    dtype: float64
```

```
# median
data.median()
```

```
1]: Pregnancies                 3.000
    Glucose                   117.000
    BloodPressure              72.000
    SkinThickness              23.000
    Insulin                    30.500
    BMI                        32.000
    DiabetesPedigreeFunction    0.372
    Age                        29.000
    Outcome                     0.000
    dtype: float64
```

```
# max
data.max()
```

```
Pregnancies                 17.00
Glucose                    199.00
BloodPressure              122.00
SkinThickness               99.00
Insulin                    846.00
BMI                         67.10
DiabetesPedigreeFunction     2.42
Age                         81.00
Outcome                      1.00
dtype: float64
```

```
# min
data.min()
```

```
Pregnancies                 0.000
Glucose                     0.000
BloodPressure               0.000
SkinThickness               0.000
Insulin                     0.000
BMI                         0.000
DiabetesPedigreeFunction    0.078
Age                        21.000
Outcome                     0.000
dtype: float64
```

```
# range
data.max() - data.min()
```

```
: Pregnancies                 17.000
  Glucose                    199.000
  BloodPressure              122.000
  SkinThickness               99.000
  Insulin                    846.000
  BMI                         67.100
  DiabetesPedigreeFunction     2.342
  Age                         60.000
  Outcome                      1.000
  dtype: float64
```

```
# mid-range
(data.max() - data.min())/2
```

```
: Pregnancies                 8.500
  Glucose                    99.500
  BloodPressure              61.000
  SkinThickness              49.500
  Insulin                   423.000
  BMI                        33.550
  DiabetesPedigreeFunction    1.171
  Age                        30.000
  Outcome                     0.500
  dtype: float64
```

Measuring the Central Tendency – Mode

- The mode is the value that occurs most frequently in the dataset.
- It can be determined for qualitative and quantitative attributes.
- Data sets with one, two, or three modes are respectively called unimodal, bimodal, and trimodal.
- In general, a data set with two or more modes is multimodal.

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

the mode(s) of the salary is (are) _____

```
# mode
data.mode()
```

:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

Measuring the Dispersion of Data - Quantiles

- Suppose that the data are sorted ascendingly.
- q-**quantiles** are cutting points that partition a finite set of values into q subsets of (nearly) equal sizes. There are $q - 1$ cutting points for the q-subsets
- There is one fewer quantile than the number of subset created.
- The 2-quantiles is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median.

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: {30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110}.

What is (are) quantile(s) of the 2-quantiles of this dataset?

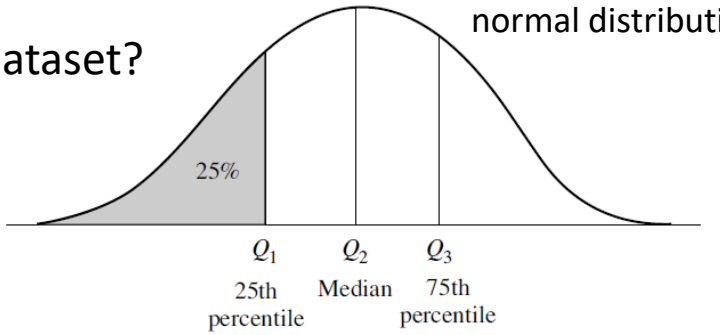
Measuring the Dispersion of Data - Quartiles

- The 4-quantiles are the three data points that split the data distribution into four (nearly) equal parts; each part represents one-fourth of the data distribution.
- The 4-quantiles are referred to as **quartiles**.

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: {30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110}.

What is (are) quantile(s) of the 4-quantiles of this dataset?

Quartiles of normal distribution



- The 100-quantiles are more commonly referred to as **percentiles**; they divide the data dataset into 100 equal-sized consecutive subsets.

- The median, quartiles, and percentiles are the most widely used forms of quantiles.

Measuring the Dispersion of Data - Interquartile Range (IQR)

- IQR is a measure of statistical dispersion and is the distance between the first and third quartiles. It is defined as

$$IQR = Q3 - Q1$$

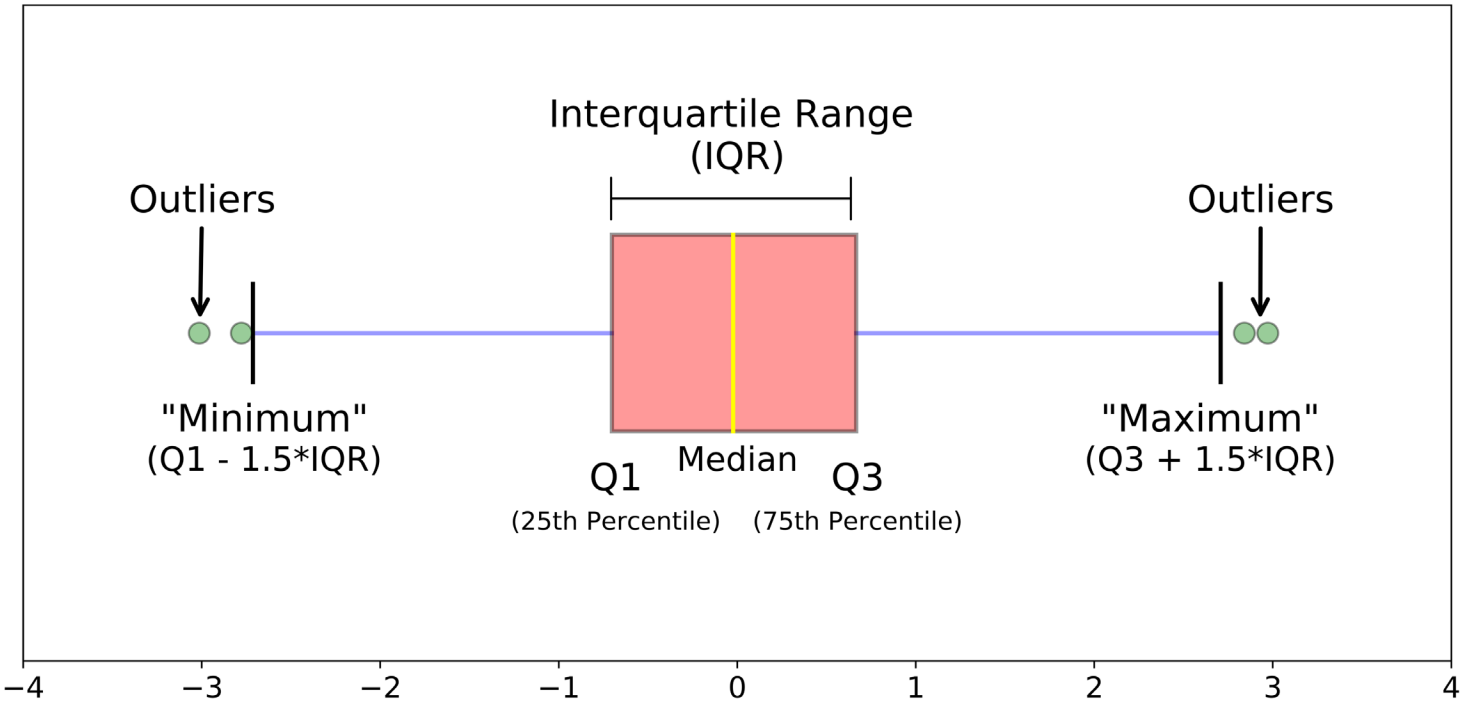
- Using IQR, we can identify suspected outliers that are values falling in at least 1.5 x IQR above Q3 and below Q1.

- Boxplot is a popular way to visualize a distribution based on minimum, Q1, median (Q2), Q3 and maximum

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: {30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110}.

What are the IQR of this dataset?

Boxplot


Interquartile Range
(IQR)
Outliers
Outliers
"Minimum"
(Q1 - 1.5*IQR)
Q1
Median
Q3
(25th Percentile)
(75th Percentile)
"Maximum"
(Q3 + 1.5*IQR)
−4
−3
−2
−1
0
1
2
3
4

```
# IQR
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
IQR
```

```
Pregnancies                   5.000
Glucose                      41.250
BloodPressure                18.000
SkinThickness                32.000
Insulin                     127.250
BMI                           9.300
DiabetesPedigreeFunction      0.382
Age                          17.000
Outcome                       1.000
dtype: float64
```

Measuring the Dispersion of Data – variance and standard deviation

- **Variance and standard deviation** are measures of data dispersion and they indicate how spread out a data distribution is.

- A low standard deviation means that the data tend to be very close to the mean, while a high standard deviation indicates that the data are spread out over a large range of values.

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2$$

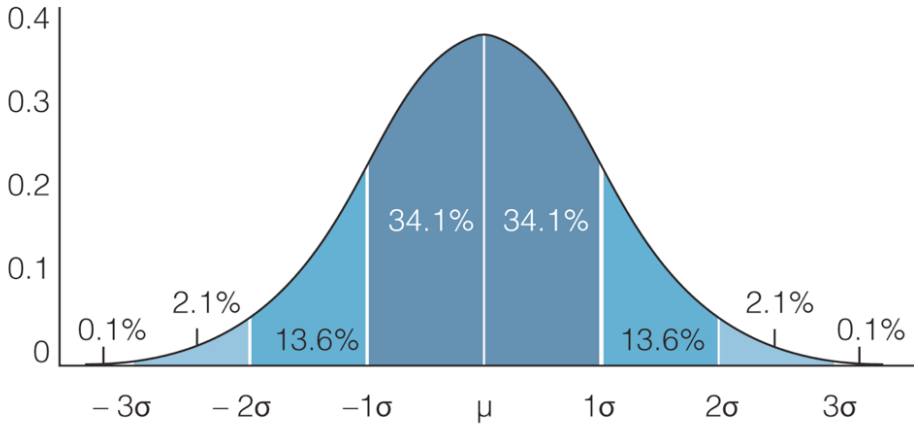
  - σ^2 is the **variance** of N values, $x_1, x_2, \ldots, x_N$,
  - $\overline{x}$ *is the mean of values*
  - The **standard deviation**, σ, is the square root of the variance, σ^2

Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

What are the variance and standard deviation of this dataset?

Measuring the Dispersion of Data – standard deviation

- The basic properties of the standard deviation, as a measure of spread are as follows:
- σ measures spread about the mean and should be considered only when the mean is chosen as the measure of center.
- $\sigma = 0$ only when there is no spread, that is, when all observations have the same value. Otherwise, $\sigma > 0$ or $\sigma < 0$.



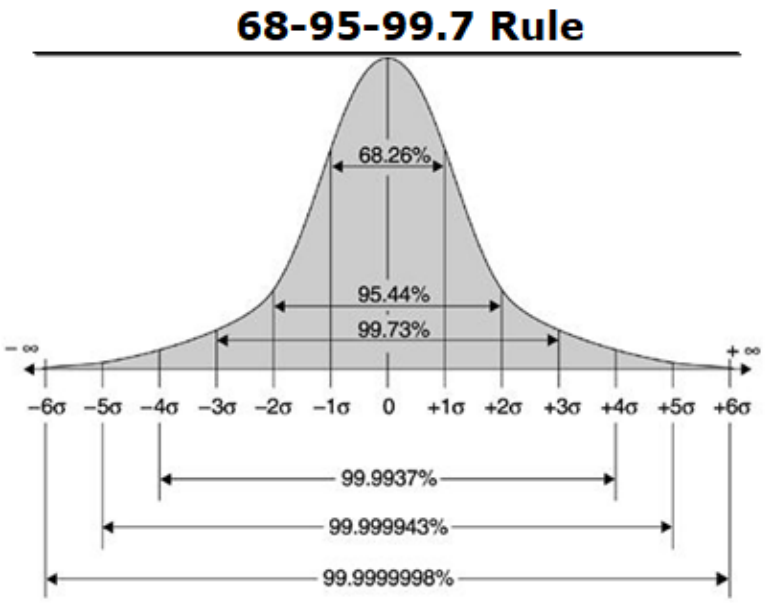
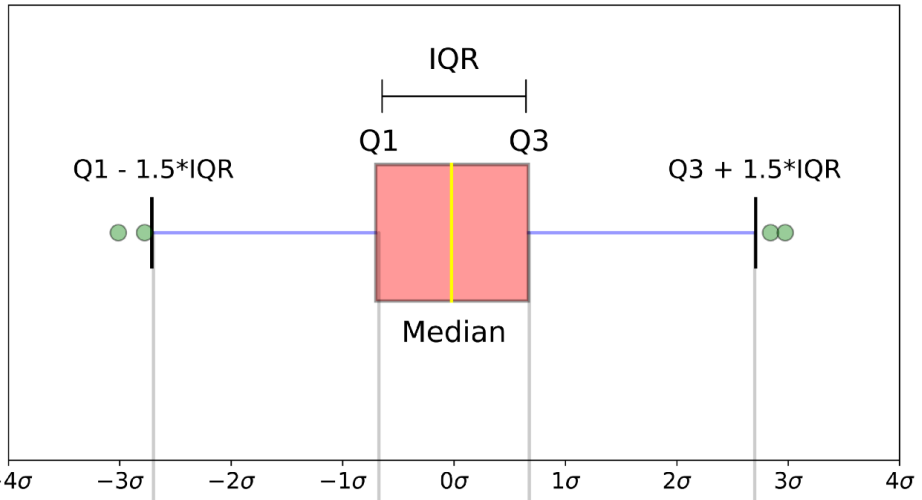
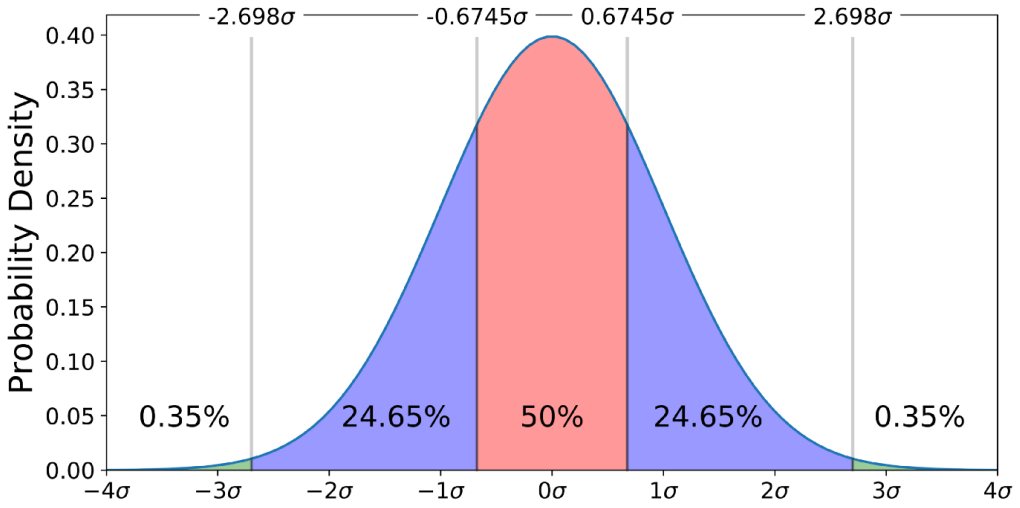
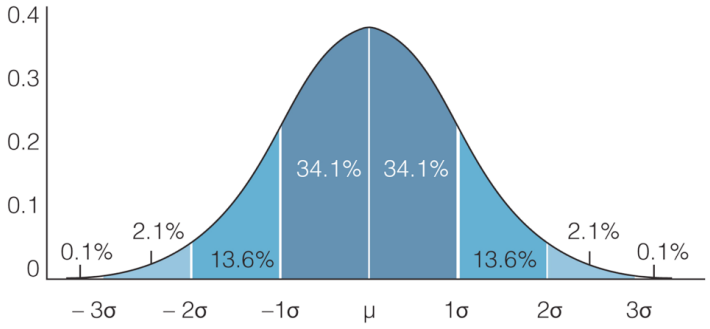


Image: Google

Boxplot on Standard Normal Distribution

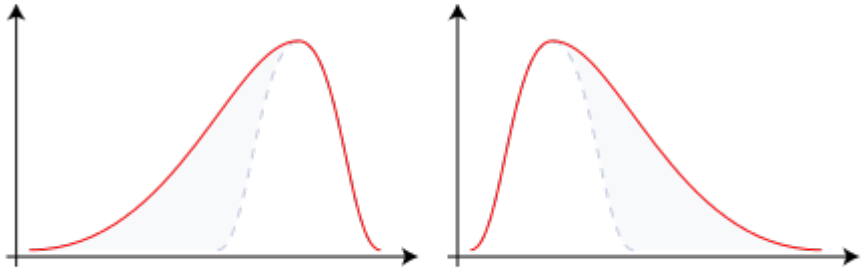


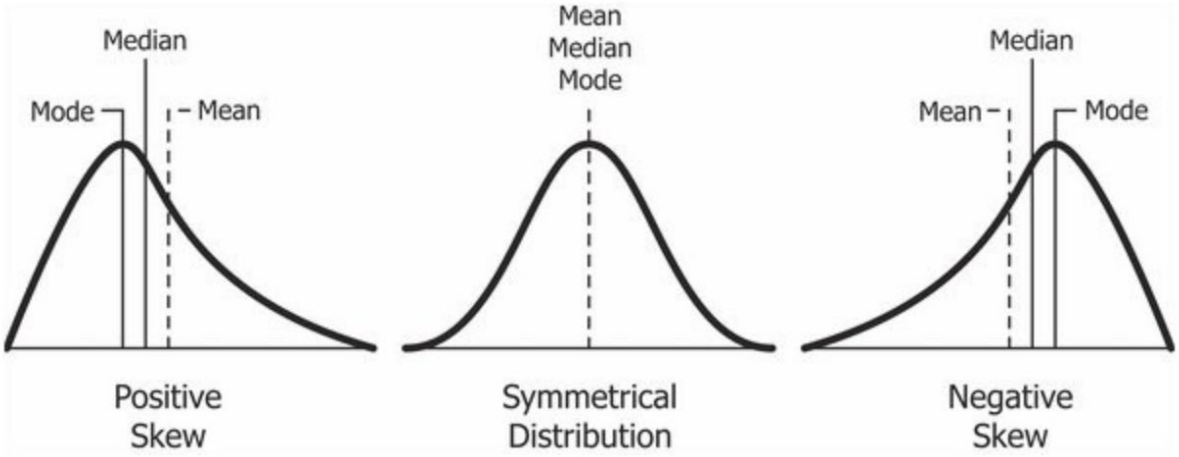




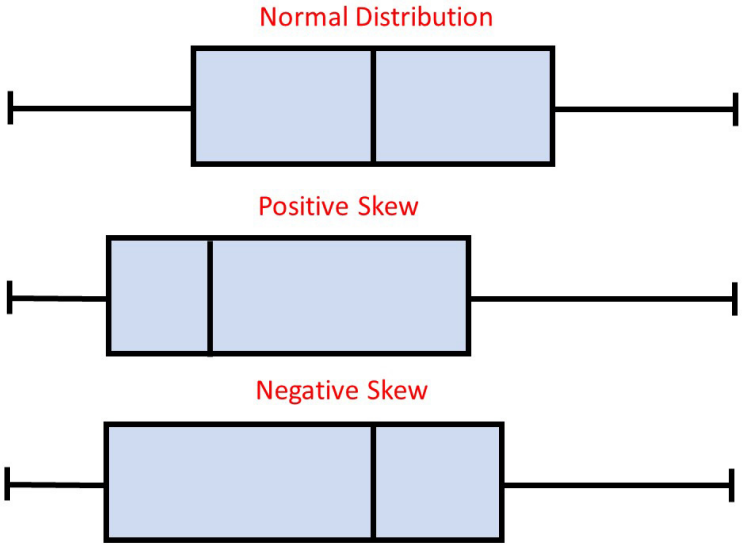
Skewness

- Skewness is a measure of asymmetry of a distribution
  - In a normal (Gaussian or bell curve) distribution, the mean divides the curve symmetrically into two equal parts at the median and the value of skewness is zero.
- When a distribution is asymmetrical the tail of the distribution is skewed to one side to the right or to the left.
  - When the value of the skewness is negative, the tail of the distribution is longer towards the left-hand side of the curve.
  - When the value of the skewness is positive, the tail of the distribution is longer towards the right-hand side of the curve





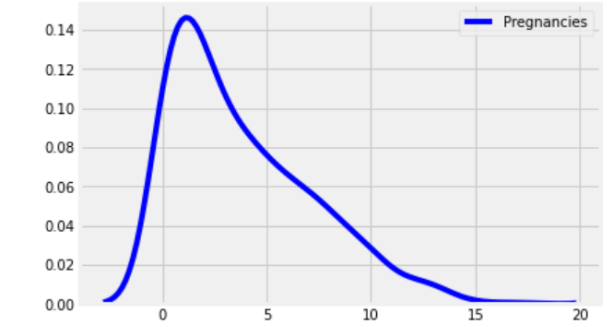
source: Wikipedia



Pearson Mode Skewness: skew = 3 * (Mean – Median) / Standard Deviation

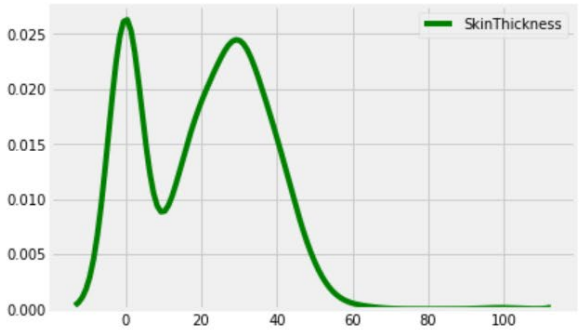
```
sns.kdeplot(data['Pregnancies'], color='b')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18becc82f08>
```



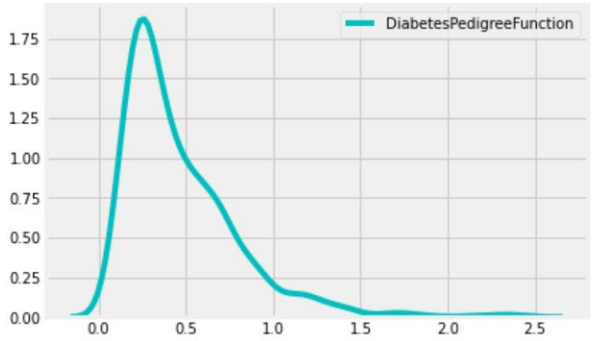

```
sns.kdeplot(data['SkinThickness'], color="g")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedd12ac8>
```



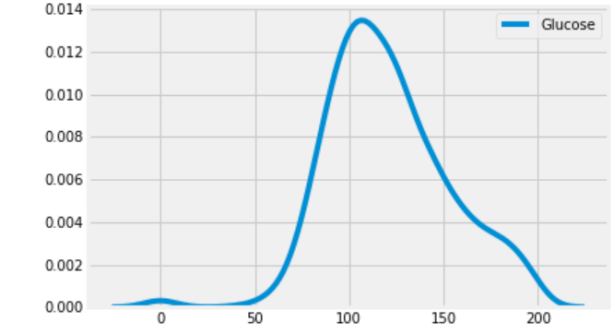

```
sns.kdeplot(data['DiabetesPedigreeFunction'], color="c")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedeebec8>
```



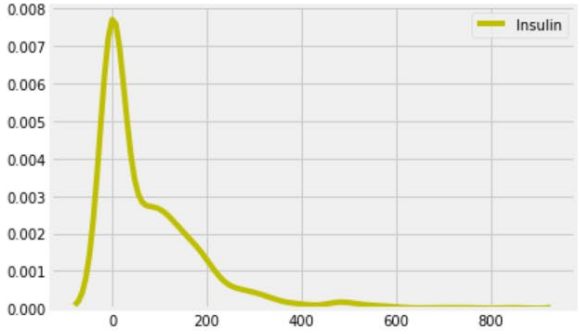

```
sns.kdeplot(data['Glucose'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bec872f48>
```



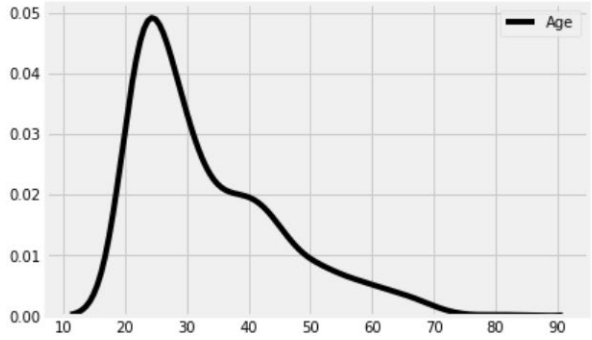

```
sns.kdeplot(data['Insulin'], color="y")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedd97f88>
```



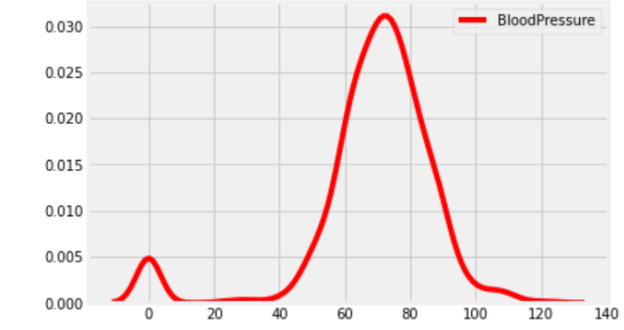

```
sns.kdeplot(data['Age'], color="k")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bee21b688>
```



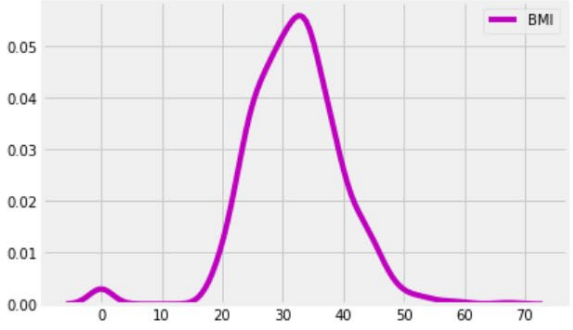

```
sns.kdeplot(data['BloodPressure'], color="r")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedcc5488>
```




```
sns.kdeplot(data['BMI'], color="m")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bec59b888>
```



Pregnancies	0.902
Glucose	0.174
BloodPressure	-1.844
SkinThickness	0.109
Insulin	2.272
BMI	-0.429
DiabetesPedigreeFunction	1.920
Age	1.130

t[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000
mean	3.845	120.895	69.105	20.536	79.799	31.993	0.472	33.241	0.349
std	3.370	31.973	19.356	15.952	115.244	7.884	0.331	11.760	0.477
min	0.000	0.000	0.000	0.000	0.000	0.000	0.078	21.000	0.000
25%	1.000	99.000	62.000	0.000	0.000	27.300	0.244	24.000	0.000
50%	3.000	117.000	72.000	23.000	30.500	32.000	0.372	29.000	0.000
75%	6.000	140.250	80.000	32.000	127.250	36.600	0.626	41.000	1.000
max	17.000	199.000	122.000	99.000	846.000	67.100	2.420	81.000	1.000

```
# mode
data.mode()
```

:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

```
# skew
skew = data.skew() # calculate the skew of each feature using the skew() function on the DataFrame
skew
```

:

```
Pregnancies                0.902
Glucose                    0.174
BloodPressure             -1.844
SkinThickness              0.109
Insulin                    2.272
BMI                       -0.429
DiabetesPedigreeFunction   1.920
Age                        1.130
Outcome                    0.635
dtype: float64
```

Skewness

- If skewness is less than -1 or greater than 1, the distribution is highly skewed.
- If skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed.
- If skewness is between -0.5 and 0.5, the distribution is approximately symmetric.

Class Distribution

- We need to know how balanced the class values are.
- Class imbalance problems are common
  - This results in models that have poor predictive performance, specifically for the minority class.
    - typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than the majority class.
    - Medical diagnosis, fraud detection, claim prediction, spam detection, anomaly detection, outlier detection, intrusion detection
- We need special handling in data preprocessing and model evaluation as well

Metrics for Evaluating Classifier

Measure	*Formula*
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$
F, F_1, F-score, harmonic mean of precision and recall	$\frac{2 \times precision \times recall}{precision + recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	P'	N'	$P + N$

Confusion matrix, shown with totals for positive and negative tuples.

Classes	*buys_computer = yes*	*buys_computer = no*	*Total*	*Recognition (%)*
buys_computer = yes	**6954**	**46**	7000	99.34
buys_computer = no	**412**	**2588**	3000	86.27
Total	7366	2634	10,000	95.42

Confusion matrix for the classes buys computer = yes and buys computer = no

Classes	*yes*	*no*	*Total*	*Recognition (%)*
yes	**90**	**210**	300	30.00
no	**140**	**9560**	9700	98.56
Total	230	9770	10,000	96.40

Confusion matrix for medical data where the class values are *yes* and *no* for a class label attribute, *cancer*.

Class Distribution

- **Slight Imbalance**. An imbalanced classification problem where the distribution of instances is uneven by a small amount in the training dataset (e.g. 1:4 or less)
- **Severe Imbalance**. An imbalanced classification problem where the distribution of instances is uneven by a large amount in the training dataset (e.g. 1:100 or more)

- *"Most of the contemporary works in class imbalance concentrate on imbalance ratios ranging from 1:4 up to 1:100. …. In real-life applications such as fraud detection or cheminformatics we may deal with problems with imbalance ratio ranging from 1:1000 up to 1:5000."*

Learning from imbalanced data: open challenges and future directions, *Progress in AI*, Bartosz Krawczyk

```
# load in the data set
data = pd.read_csv("diabetes.csv") # load the CSV file using pandas.read_csv function
                                   # The function returns a pandas.DataFrame
```

```
# class distribution
class_counts = data.groupby('Outcome').size()
class_counts
```

```
Outcome
0    500
1    268
dtype: int64
```

```
class_counts[0]/data['Outcome'].size
```

```
0.6510416666666666
```

```
class_counts[1]/data['Outcome'].size
```

```
0.3489583333333333
```

Correlations between attributes

- Correlation refers to the relationship between two attributes and how they related in terms of change.
  - Correlation analysis is used to detect attribute redundancy and improve performance of model
  - The performance of some machine learning models like linear and logistic regression may not be good if there are highly correlated attributes in the dataset.
  - We need to review all of the pairwise correlations of the attribute in the dataset.

Correlation analysis

- Correlation analysis
  - Measure how strongly one attribute is related (dependent) with the other
  - Numeric data
    - Covariance
    - Pearson's Correlation Coefficient
    - `corr()` Pandas function
  - Categorical data
    - Chi-square test
    - `chi2_contingency()` SciPy function
    - `from sklearn.attribute_selection import chi2`

Covariance between attributes

Covariance measures the strength and the direction of the relationship between the observations of two attributes and the correlation is derived from the covariance.

The sample covariance between two variables, X and Y, is

$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \,, \quad \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

	X	Y
1	**1**	**3**
2	**-2**	**2**
3	**3**	**4**
4	**0**	**6**
5	**3**	**0**

Cov(X, Y) =

Covariance between attributes

- **Positive covariance**: If $cov(X, Y) > 0$, then X and Y both tend to be larger than their expected values.
- **Negative covariance**: If $cov(X, Y) < 0$ then if X is larger than its expected value, Y is likely to be smaller than its expected value.
- **Independence**: $cov(X, Y) = 0$

```
# Covariance between features
set_option('precision', 3) # Displays precision for decimal numbers
covariance = data.cov()
covariance
```

1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	11.354	13.947	9.215	-4.390	-28.555	0.470	-0.037	21.571	0.357
Glucose	13.947	1022.248	94.431	29.239	1220.936	55.727	1.455	99.083	7.115
BloodPressure	9.215	94.431	374.647	64.029	198.378	43.005	0.265	54.523	0.601
SkinThickness	-4.390	29.239	64.029	254.473	802.980	49.374	0.972	-21.381	0.569
Insulin	-28.555	1220.936	198.378	802.980	13281.180	179.775	7.067	-57.143	7.176
BMI	0.470	55.727	43.005	49.374	179.775	62.160	0.367	3.360	1.101
DiabetesPedigreeFunction	-0.037	1.455	0.265	0.972	7.067	0.367	0.110	0.131	0.027
Age	21.571	99.083	54.523	-21.381	-57.143	3.360	0.131	138.303	1.337
Outcome	0.357	7.115	0.601	0.569	7.176	1.101	0.027	1.337	0.227

Pearson Correlation Coefficient between attributes

The correlation is derived from the covariance.

$$corr(X,Y) = r_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

- $r_{X,Y}$ can take a range of values from +1 to -1
- If $r_{X,Y} > 0$: X and Y are positively correlated (X's values increase as Y's).
- The higher, the stronger correlation.
- $r_{X,Y} = 0$: independent;
- $r_{X,Y} < 0$: negatively correlated

corr(X, Y) =

	X	Y
1	**1**	**3**
2	**-2**	**2**
3	**3**	**4**
4	**0**	**6**
5	**3**	**0**

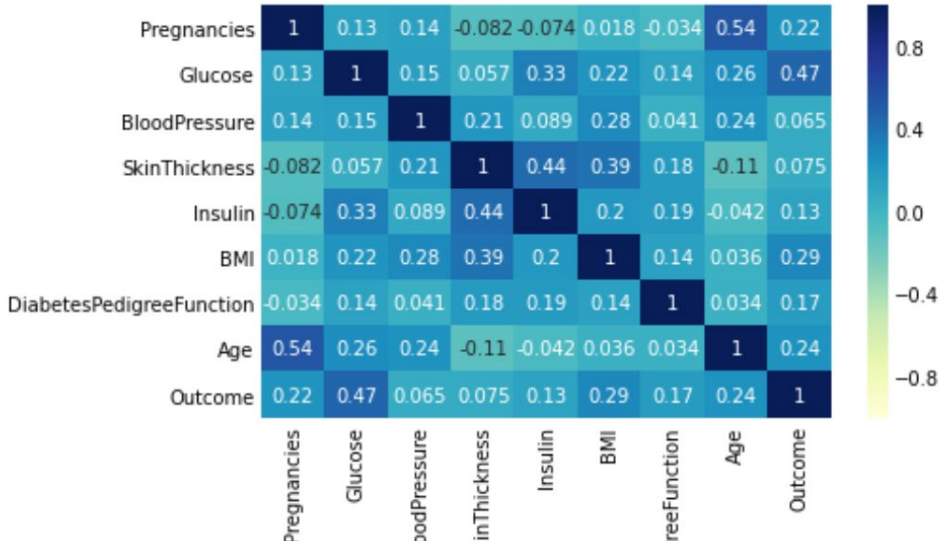
```
# Pairwise Pearson correlations
set_option('precision', 3) # Displays precision for decimal numbers
correlations = data.corr(method='pearson')
correlations
```

|:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000	0.129	0.141	-0.082	-0.074	0.018	-0.034	0.544	0.222
Glucose	0.129	1.000	0.153	0.057	0.331	0.221	0.137	0.264	0.467
BloodPressure	0.141	0.153	1.000	0.207	0.089	0.282	0.041	0.240	0.065
SkinThickness	-0.082	0.057	0.207	1.000	0.437	0.393	0.184	-0.114	0.075
Insulin	-0.074	0.331	0.089	0.437	1.000	0.198	0.185	-0.042	0.131
BMI	0.018	0.221	0.282	0.393	0.198	1.000	0.141	0.036	0.293
DiabetesPedigreeFunction	-0.034	0.137	0.041	0.184	0.185	0.141	1.000	0.034	0.174
Age	0.544	0.264	0.240	-0.114	-0.042	0.036	0.034	1.000	0.238
Outcome	0.222	0.467	0.065	0.075	0.131	0.293	0.174	0.238	1.000

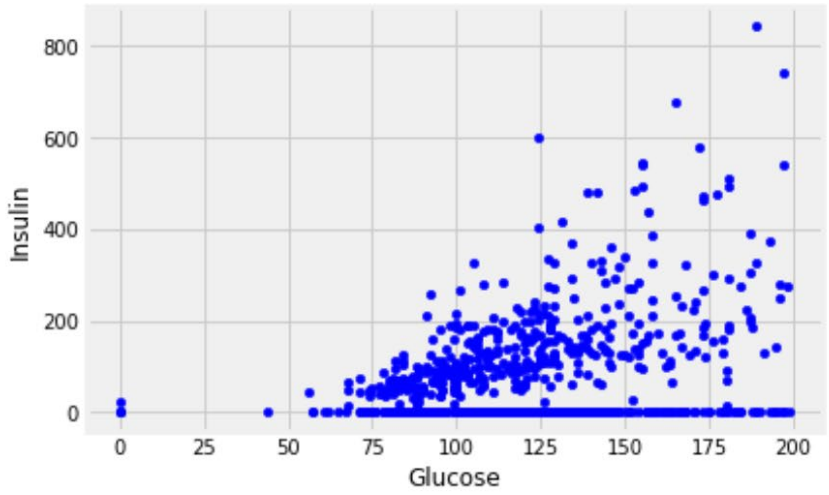
```
1 sns.heatmap(data.corr(), vmin=-1, vmax=1, cmap="YlGnBu", annot = True)
```

|: <matplotlib.axes._subplots.AxesSubplot at 0x1acaa3e9a48>



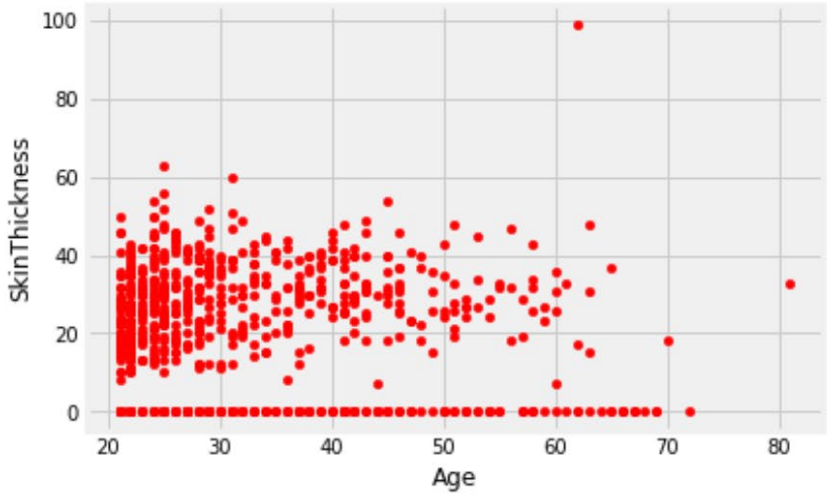
```
# Pandas Scatter Plot
data.plot(x='Glucose',y='Insulin',kind='scatter',color='B')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f08d8ee548>
```

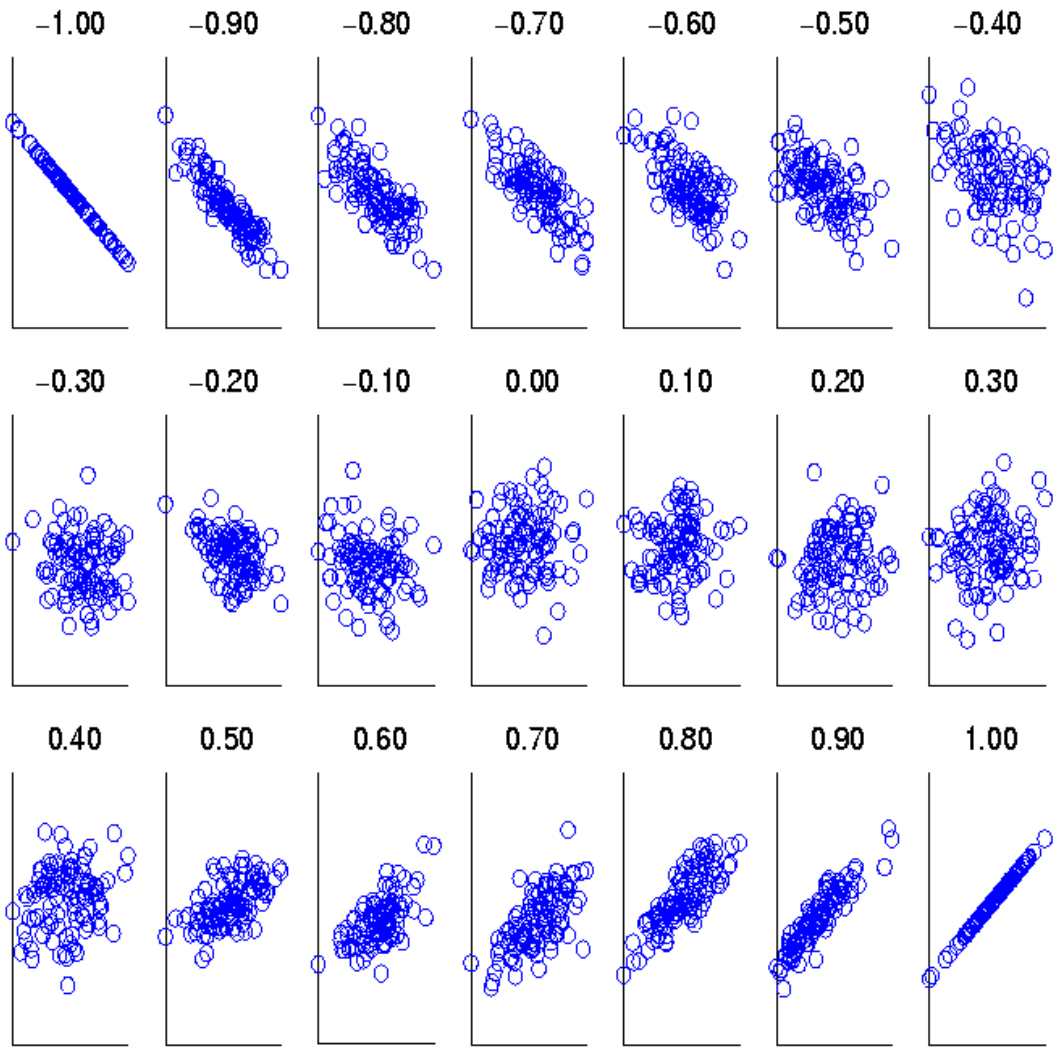



```
# Pandas Scatter Plot
data.plot(x='Age',y='SkinThickness',kind='scatter',color='R')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f08db397c8>
```



Evaluating Correlation



Scatter plots showing the similarity from –1 to 1.

Chi-Square Test of Independent

- Test the independence (or dependence) of two categorical attributes using the chi-square test
- The null hypothesis assumes true until we have evidence to go for it or go against it
  - H_0 (null hypothesis): the two attributes are independent
  - H_a (alternative hypothesis): the two attributes are dependent

Chi-Square Test

- Summarize the data in the two-way contingency table, the observed table
  - This table represents the observed counts of each cell
- Calculate the expected count for each cell in the table to find the expected table from observed counts
  - This table displays what the counts for each cell would be for sample data if there were not relations between two attributes
  - To find the expected count for each cell in the expected table we multiply the marginal row and column totals for that cell and divide by the overall total in the observed table
  - i.e. for each cell this is
  - $E = (row\ total \times column\ total) \,/\, total\ number\ of\ data$

Chi-Square Test

- Compute a Chi-Square test statistic as follows:
  - $\chi^2 = \Sigma(O_i - E_i)^2/E_i$
    - where O_i is an observed count of each cell of the contingency table, and E_i is an expected count of each cell of the contingency table
- With Chi-Square test statistic, a significance level (α) chosen, and the degree of freedom for the Chi-Square distribution, we can make a decision.
  - Degree of Freedom (df) = (number of rows – 1) x (numbers of columns – 1)
  - a significance level = 0.001

Chi-Square Test

- The decision is made by
  - Either comparing the value of test Chi-Square statistic to a critical chi-square value at a chosen significance level, α (rejection region approach)
    - Test statistic >= Critical value: reject null hypothesis, attributes are dependent (H_a)
    - Test statistic < Critical value: fail to reject null hypothesis, attributes are independent (H_0)
  - or finding the probability of getting of test Chi-Square statistic (p-value approach)
    - p-value <= α (a chosen significance level): reject null hypothesis, attributes are dependent (H_a)
    - p-value > α: fail to reject null hypothesis, attributes are independent (H_0)
  - http://courses.atlas.illinois.edu/spring2016/STAT/STAT200/pchisq.html

the hypothesis that *gender* and *preferred reading* are independent

	male	*female*	*Total*
fiction	250 (90)	200 (360)	450
non_fiction	50 (210)	1000 (840)	1050
Total	300	1200	1500

Note: Are *gender* and *preferred_reading* correlated?

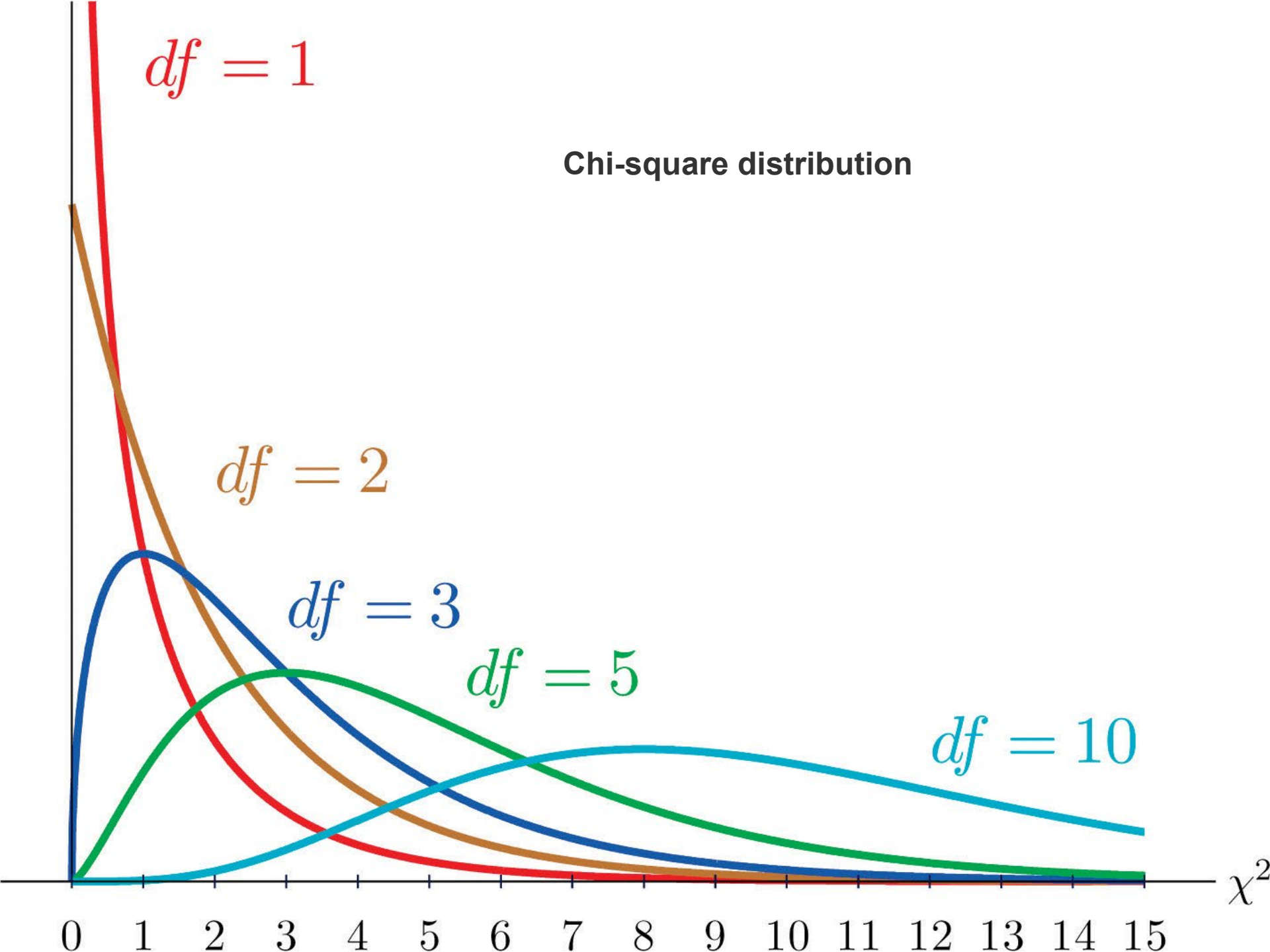
$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840}$$
$$= 284.44 + 121.90 + 71.11 + 30.48 = 507.93.$$

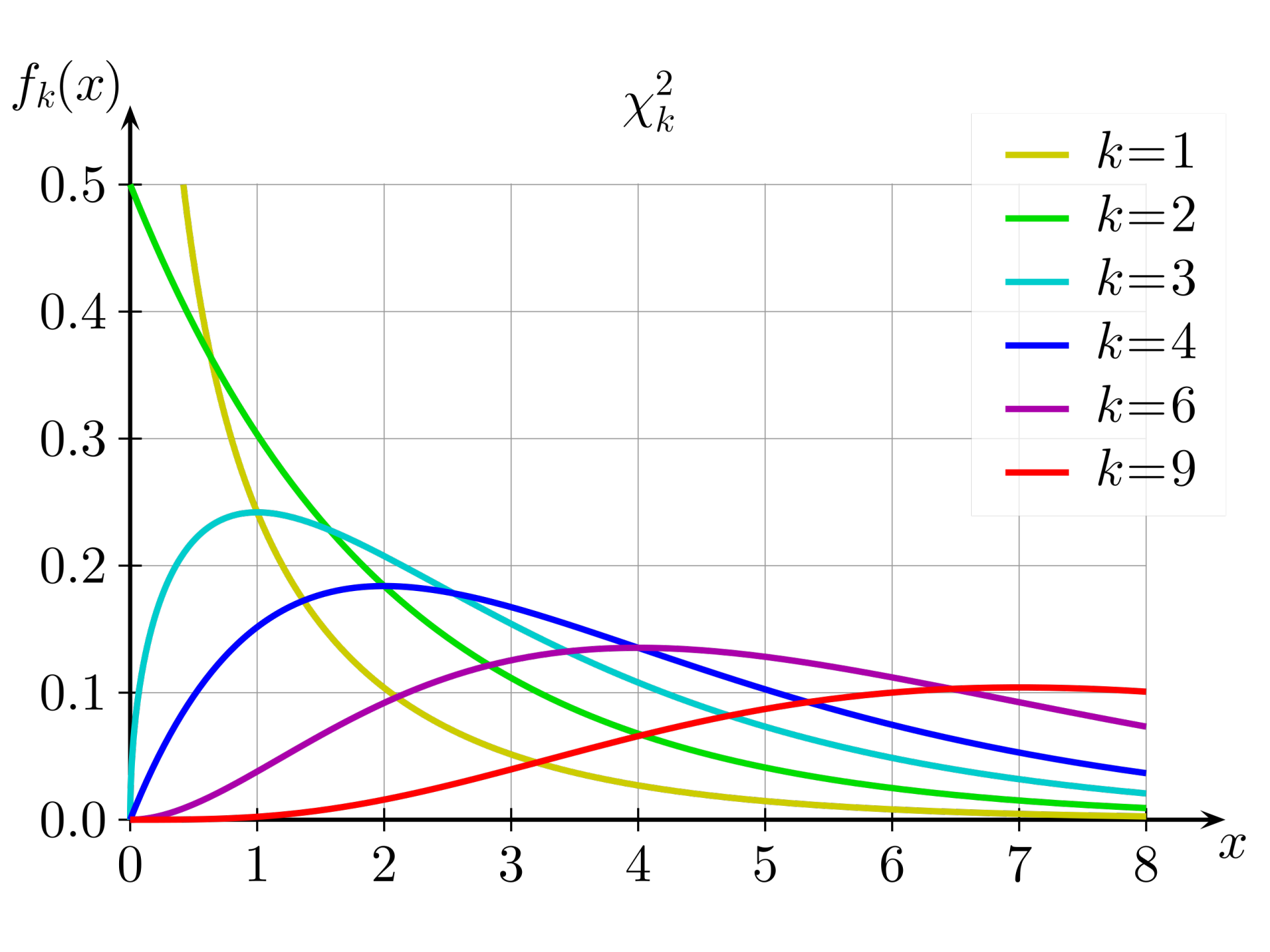
Data Mining, Jiawei

For the degree of freedom is 1 and the significance level, α = 0.001

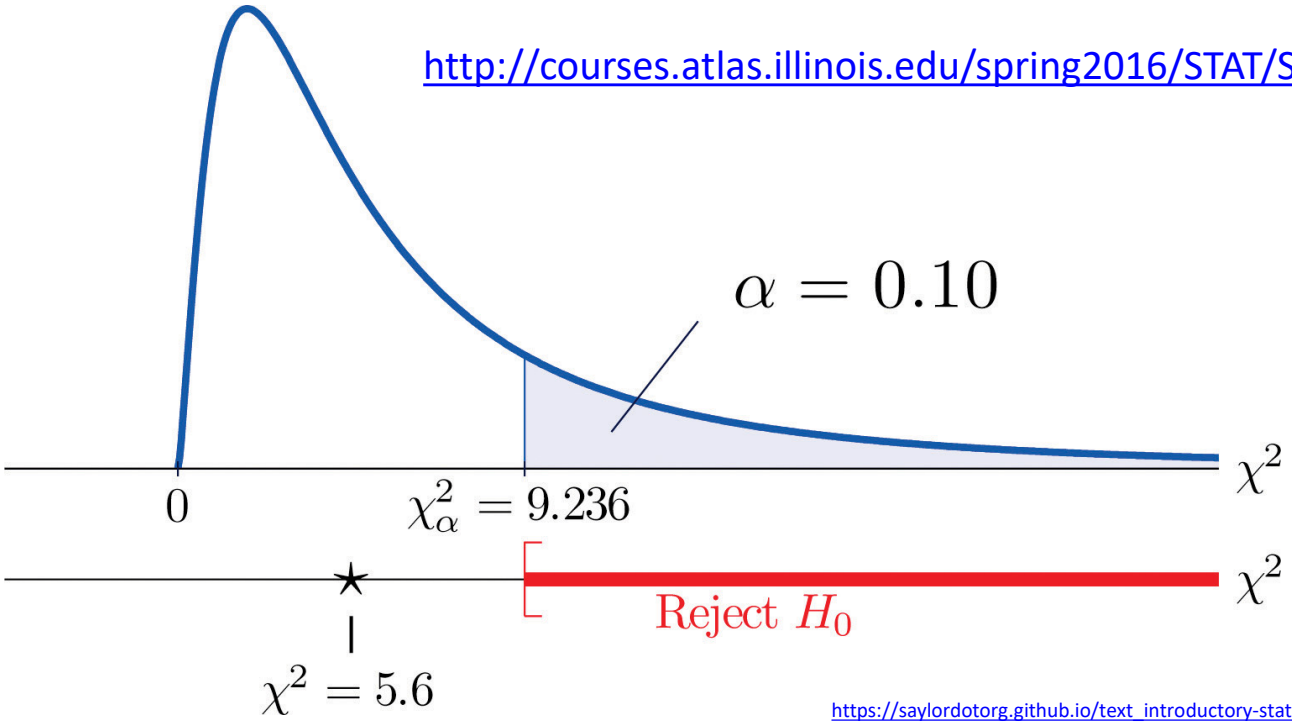
- the test Chi-Square statistic (χ^2) needed to reject the hypothesis at the significance level is equal to or greater than Chi-Square critical value (x_α^2), 10.828.
- the p-value of the test Chi-Square statistic needed to reject the hypothesis is equal to or less than the significance level, α, 0.001.

Since the computed Chi-Square value (χ^2) > x_α^2, (since p-value < α) we can reject the hypothesis that gender and preferred reading are independent and conclude that the two attributes are (strongly) correlated for the given group of people.


Chi-square distribution
df = 1
df = 2
df = 3
df = 5
df = 10
χ²
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


$f_k(x)$
χ^2_k
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
6
7
8
x
$k=1$
$k=2$
$k=3$
$k=4$
$k=6$
$k=9$

http://courses.atlas.illinois.edu/spring2016/STAT/STAT200/pchisq.html



https://saylordotorg.github.io/text_introductory-statistics/s15-chi-square-tests-and-f-tests.html

Critical Value for Chi-Square

Select your significance level, input your degrees of freedom, and then hit "Calculate for Chi-Square".

Significance Level: 0.10 ▼

Degrees of Freedom: 5

Critical value = 9.236.

Calculate for Chi-Square

https://www.socscistatistics.com/tests/criticalvalues/default.aspx

Chi-square score: 5.6

DF: 5

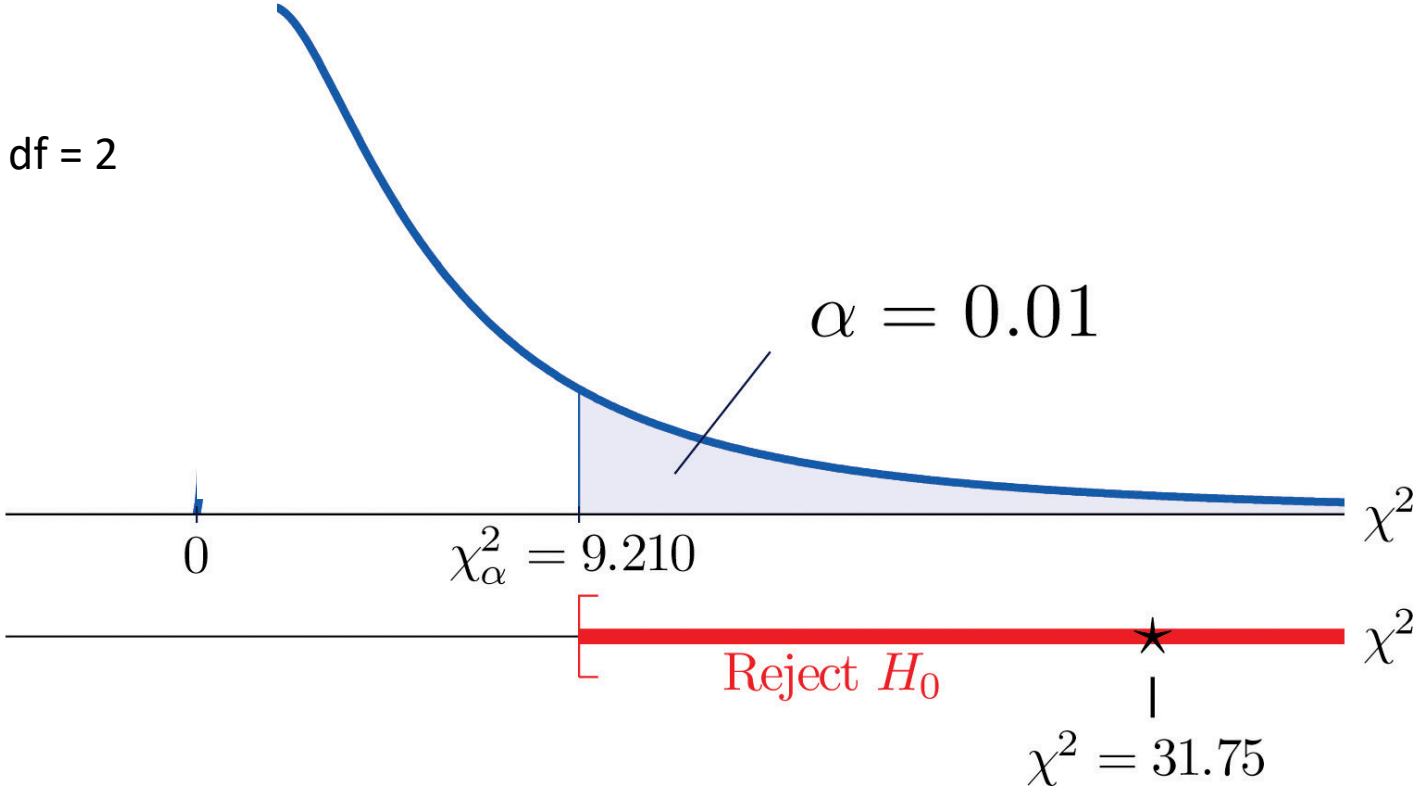
Significance Level:

- 0.01
- 0.05
- 0.10

The P-Value is .347105. The result is *not* significant at p < .10.

Calculate

https://www.socscistatistics.com/pvalues/chidistribution.aspx



Critical Value for Chi-Square

Select your significance level, input your degrees of freedom, and then hit "Calculate for Chi-Square"

Significance Level: 0.01

Degrees of Freedom: 2

Critical value = 9.21.

Calculate for Chi-Square

Chi-square score: 31.75

DF: 2

Significance Level:

- (•) 0.01
- ( ) 0.05
- ( ) 0.10

The P-Value is < .00001. The result is significant at p < .01.

Calculate

Critical values of the Chi-square distribution with *d* degrees of freedom

	Probability of exceeding the critical value						
d	0.05	0.01	0.001	*d*	0.05	0.01	0.001
1	3.841	6.635	10.828	11	19.675	24.725	31.264
2	5.991	9.210	13.816	12	21.026	26.217	32.910
3	7.815	11.345	16.266	13	22.362	27.688	34.528
4	9.488	13.277	18.467	14	23.685	29.141	36.123
5	11.070	15.086	20.515	15	24.996	30.578	37.697
6	12.592	16.812	22.458	16	26.296	32.000	39.252
7	14.067	18.475	24.322	17	27.587	33.409	40.790
8	15.507	20.090	26.125	18	28.869	34.805	42.312
9	16.919	21.666	27.877	19	30.144	36.191	43.820
10	18.307	23.209	29.588	20	31.410	37.566	45.315

***INTRODUCTION TO POPULATION GENETICS,* Table D.1**

Degree of Freedom	Probability of Exceeding the Critical Value								
	0.99	0.95	0.90	0.75	0.50	0.25	0.10	0.05	0.01
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84	6.63
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99	9.21
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81	11.34
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49	13.28
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07	15.09
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59	16.81
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07	18.48
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51	20.09
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92	21.67
10	2.558	3.940	4.865	6.737	9.342	12.55	15.99	18.31	23.21
11	3.053	4.575	5.578	7.584	10.341	13.70	17.28	19.68	24.72
12	3.571	5.226	6.304	8.438	11.340	14.85	18.55	21.03	26.22
13	4.107	5.892	7.042	9.299	12.340	15.98	19.81	22.36	27.69
14	4.660	6.571	7.790	10.165	13.339	17.12	21.06	23.68	29.14
15	5.229	7.261	8.547	11.037	14.339	18.25	22.31	25.00	30.58
16	5.812	7.962	9.312	11.912	15.338	19.37	23.54	26.30	32.00
17	6.408	8.672	10.085	12.792	16.338	20.49	24.77	27.59	33.41
18	7.015	9.390	10.865	13.675	17.338	21.60	25.99	28.87	34.80
19	7.633	10.117	11.651	14.562	18.338	22.72	27.20	30.14	36.19
20	8.260	10.851	12.443	15.452	19.337	23.83	28.41	31.41	37.57
22	9.542	12.338	14.041	17.240	21.337	26.04	30.81	33.92	40.29
24	10.856	13.848	15.659	19.037	23.337	28.24	33.20	36.42	42.98
26	12.198	15.379	17.292	20.843	25.336	30.43	35.56	38.89	45.64
28	13.565	16.928	18.939	22.657	27.336	32.62	37.92	41.34	48.28
30	14.953	18.493	20.599	24.478	29.336	34.80	40.26	43.77	50.89
40	22.164	26.509	29.051	33.660	39.335	45.62	51.80	55.76	63.69
50	27.707	34.764	37.689	42.942	49.335	56.33	63.17	67.50	76.15
60	37.485	43.188	46.459	52.294	59.335	66.98	74.40	79.08	88.38
	Not Significant							***Significant***	

```
# calculate Chi-Squared test
# Returns of scipy.stats.chi2_contingency()
# chi2: The test statistic.
# p: The p-value of the test
# dof: Degrees of freedom
# expected: The expected frequencies, based on the marginal sums of the table.
```

```
data = pd.read_csv('courses.csv')
```

```
data
```

:

	Gender	Interest
0	Male	Science
1	Male	Science
2	Male	Science
3	Male	Science
4	Male	Science
...	...	...
87	Female	Art
88	Female	Art
89	Female	Art
90	Female	Art
91	Female	Art

92 rows × 2 columns

]:
```
#Contingency Table
table = pd.crosstab(data["Gender"],data["Interest"])
```

]:
```
table
```

:[95]:

Interest	Art	Math	Science
Sex			
Female	30	20	10
Male	17	9	6

]:
```
# Observed Values
Observed_Values = table.values
Observed_Values
```

:109]:
```
array([[30, 20, 10],
       [17,  9,  6]], dtype=int64)
```

```
# calculate Chi-Squared test
# Returns of scipy.stats.chi2_contingency()
# chi2: The test statistic.
# p: The p-value of the test
# dof: Degrees of freedom
# expected: The expected frequencies, based on the marginal sums of the table.

import scipy.stats as sp

from scipy.stats import chi2_contingency
from scipy.stats import chi2

chi2_test_statistic, p, dof, expected = sp.chi2_contingency(table)

print('dof=%d' % dof)
```

```
dof=2
```

```
chi2_test_statistic, p, dof, expected
```

```
10]: (0.271574651504035,
      0.8730282833800731,
      2,
      array([[30.65217391, 18.91304348, 10.43478261],
             [16.34782609, 10.08695652,  5.56521739]]))
```

```
# interpret test-statistic

# Test Statistic >= Critical Value: reject null hypothesis, dependent (Ha)
# Test Statistic < Critical Value:  fail to reject null hypothesis,independent (Ho).
# chi2.ppf(q, df, loc=0, scale=1) inverset CDF.

from scipy.stats import chi2

prob = 0.95 # significant value = 1 - 0.95 = 0.05
critical = chi2.ppf(prob, dof)
```

```
round(critical, 3)
```

: 5.991

```
print('critical=%.3f, chi2_test_statistic=%.3f' % (critical, chi2_test_statistic))
```

```
critical=5.991, chi2_test_statistic=0.272
```

```
if chi2_test_statistic >= critical:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
```

```
Independent (fail to reject H0)
```

```
# interpret p-value
# p-value <= alpha: reject null hypothesis, dependent (Ha)
# p-value > alpha:  fail to reject null hypothesis,independent (Ho).
alpha = 1.0 - prob
print('significance=%.3f, p=%.3f' % (alpha, p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
```

```
significance=0.050, p=0.873
Independent (fail to reject H0)
```

Lesson A-4

Getting to Know Your Data

- Data Exploration (Visualization)

Data Visualization

- *"Data visualization is the graphic representation of data"*
- *"The main goal of data visualization is to communicate information clearly and effectively through graphical means"*

  Vitaly Friedman
- Data visualization is nowadays one of the most important tasks in data science
  - It is a part of the data exploratory phase of the data science process or part of modeling as presentation of the output

Data Visualization

- Univariate plots
  - Visualize each feature of dataset independently
  - Bar, histogram, density, box, line, area
- Multivariate plots
  - Visualize the correlations (dependencies, interactions) between multiple variables.
  - Scatter, matrix (heatmap), qq-plot

Types of plots

- line plot
- vertical bar plot
- horizontal bar plot
- histogram
- pie plot
- density plot (Kernel Density Estimation plot)
- area plot
- box (and whisker) plot
- scatter plot
- hexbin plot
- correlation matrix plot (heat map)
- KDE plot

Data Visualization in Python

- Matplotlib library
  - Python's the most popular visualization library
  - is more easily customizable
- Seaborn library
  - Python’s popular visualization library
  - is a library based on Matplotlib and closely integrated with pandas data
    - uses fewer syntax
  - has default themes
    - Provides visualization patterns
- Pandas plotting module
  - built-in plot functions available on its Series and DataFrame objects.

Matplotlib vs. Seaborn

- Functionality
  - Matplotlib is mainly deployed for basic plotting.
  - Seaborn provides a variety of visualization patterns.
- Handling Multiple Figures
  - Matplotlib has multiple figures can be opened, but need to be closed explicitly.
  - Seaborn automates the creation of multiple figures.
- Visualization
  - Matplotlib is a graphics package for data visualization in Python.
  - Seaborn is more integrated for working with Pandas data frames

Matplotlib vs. Seaborn

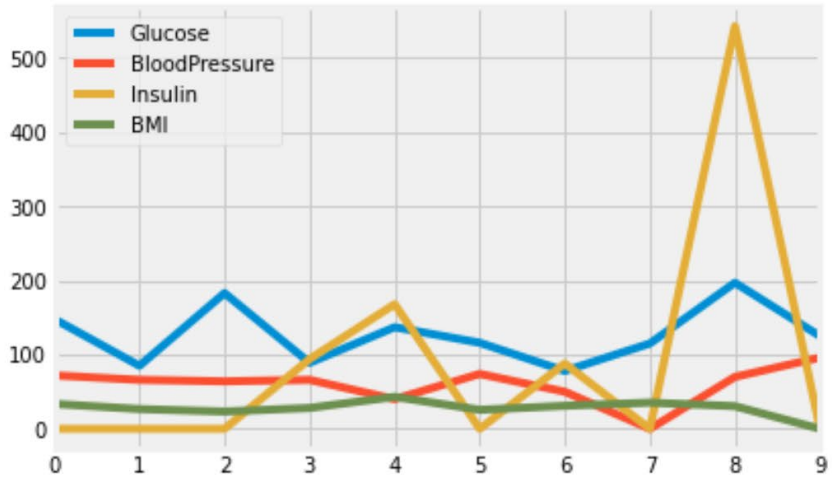
- Data frames and Arrays
  - Matplotlib works with data frames and arrays. It has different stateful APIs for plotting.
  - Seaborn works with the dataset as a whole and is much more intuitive than Matplotlib.
- Flexibility
  - Matplotlib is highly customizable and powerful.
  - Seaborn avoids a lot of boilerplate by providing default themes which are commonly used.

Pandas plotting module

- dataframe `plot()` functions create decent looking plots with the dataframe
- The plot function on Series and DataFrame is a simple wrapper around Matplotlib `plt.plot()`
  - you don’t need to write those long matplotlib codes for plotting.
  - if you want some advanced plots which cannot be done using the plot function then you can switch to matplotlib or seaborn

```
# Pandas Line Chart
# data.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'line')

df1=data[:10]
df1.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'line')
```

0]: <matplotlib.axes._subplots.AxesSubplot at 0x1d379e72508>

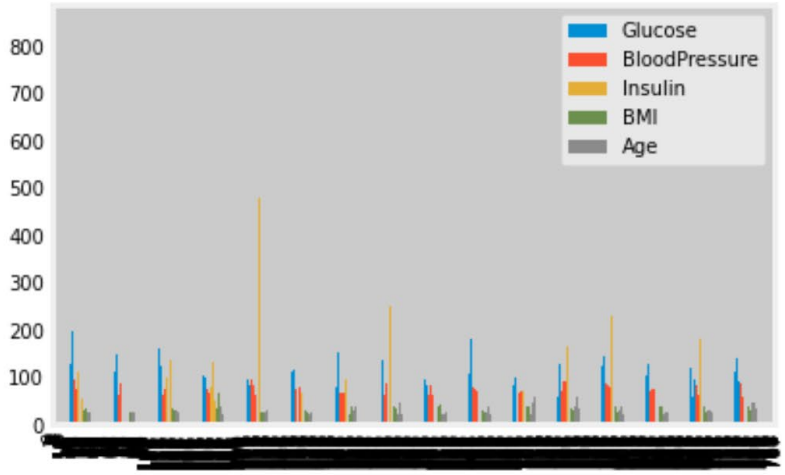


Line Graph

- A line graph, also known as a line chart, is a type of chart used to visualize the value of something over the other one (which is usually time)
  - The line graph is a efficient visual tool for time-series data such as finance, marketing, weather, laboratory data.
  - Data points are plotted and connected by a line

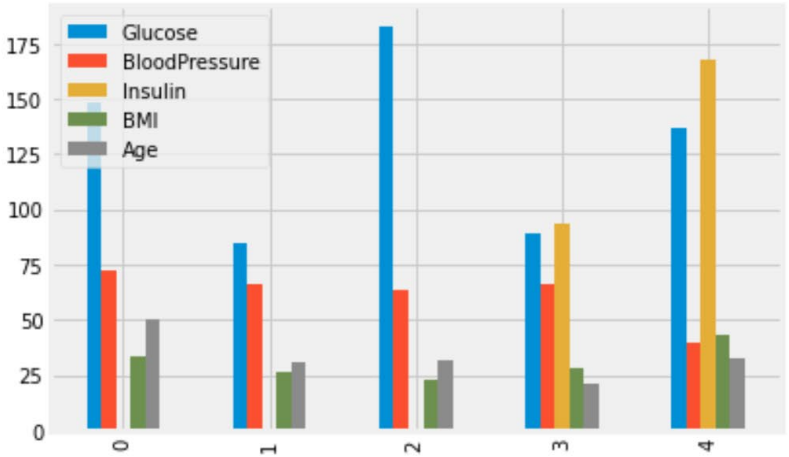
```
# Pandas bar plot
data.plot(data.index.name,['Glucose','BloodPressure','Insulin','BMI', 'Age'],kind = 'bar')
```

6]: <matplotlib.axes._subplots.AxesSubplot at 0x1d30afaf348>




```
# Pandas bar plot
df1=data[:5]
df1.plot(data.index.name,['Glucose','BloodPressure','Insulin','BMI', 'Age'],kind = 'bar')
```

6]: <matplotlib.axes._subplots.AxesSubplot at 0x1d37a2f2588>

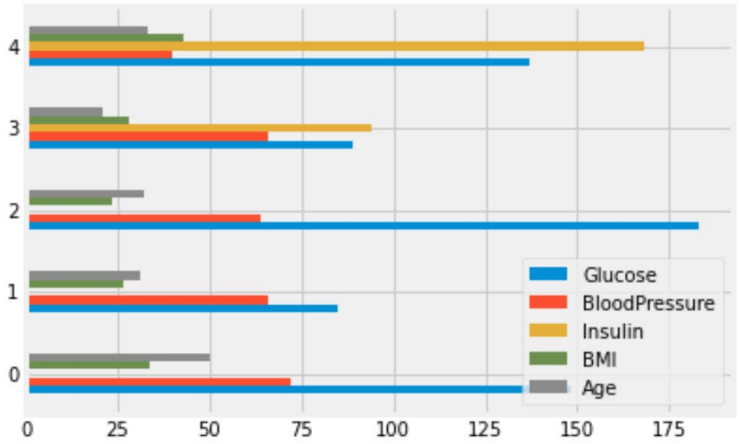


Bar Graph (Chart)

- A bar graph uses bars to compare data among categories.
- Bar graph presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent.
- The bars can be plotted vertically or horizontally.
  - A vertical bar chart is sometimes called a column chart.
  - A bar graph is not good for a large number of categorical data

```
# Horizontal Bar with positions
df1.plot(data.index.name,[1, 2, 4, 5, 7],kind = 'barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d37a4606c8>
```

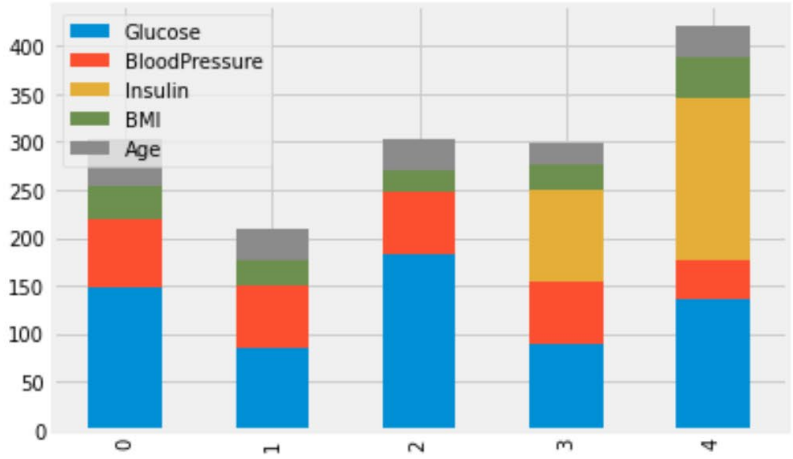


Horizontal Bar Chart

- represent the data horizontally.
  - bars are drawn horizontally.
  - data categories are shown on the vertical axis
  - data values are shown on the horizontal axis.
  - the length of each bar is equal to the value corresponding the data category and bars go across from left to right.

```
# Pandas stacked bar plot
df1.plot(data.index.name,['Glucose','BloodPressure','Insulin','BMI', 'Age'],kind = 'bar', stacked=True)
```

: <matplotlib.axes._subplots.AxesSubplot at 0x1d306c838c8>

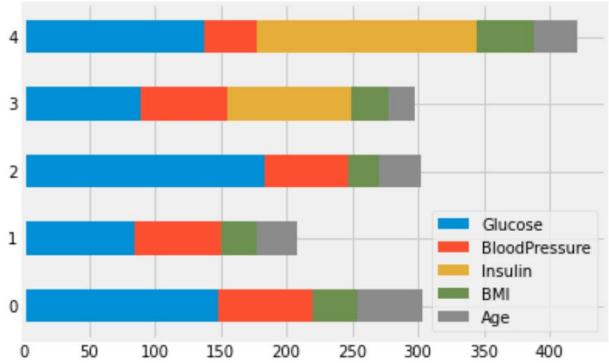


Stacked bar chart

- The stacked bar chart extends the standard bar chart from looking at numeric values across one categorical feature to multiple.
- Each bar in a standard bar chart is divided into a number of sub-bars stacked end to end, each one corresponding to a level of the corresponding categorical feature.

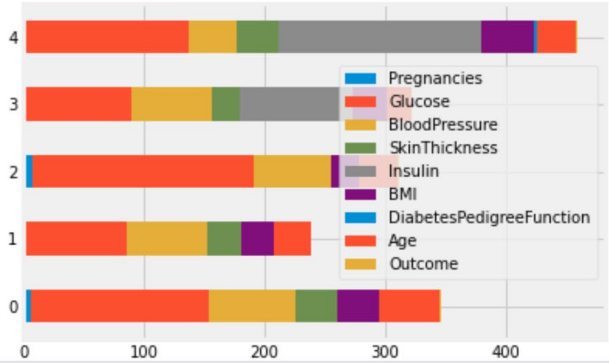
```
# Horizontal Stacked Bar with positions
df1.plot.barh(data.index.name,['Glucose','BloodPressure','Insulin','BMI', 'Age'], stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d306d65f08>
```



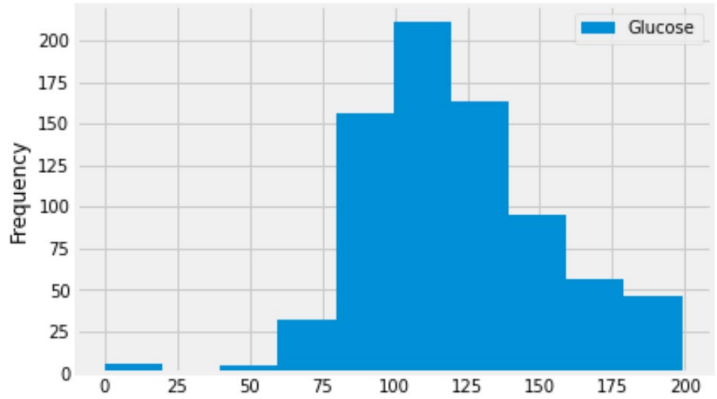

```
# Horizontal Stacked Bar with positions
df1.plot.barh(stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d306dc0c88>
```



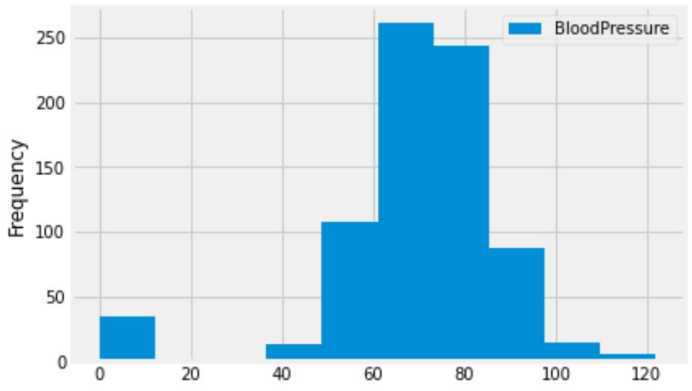
```
# Histogram
data.plot(data.index.name,[1], kind = 'hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d37a711588>
```




```
# Histogram
data.plot(data.index.name,[2],kind = 'hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d37a76ec48>
```

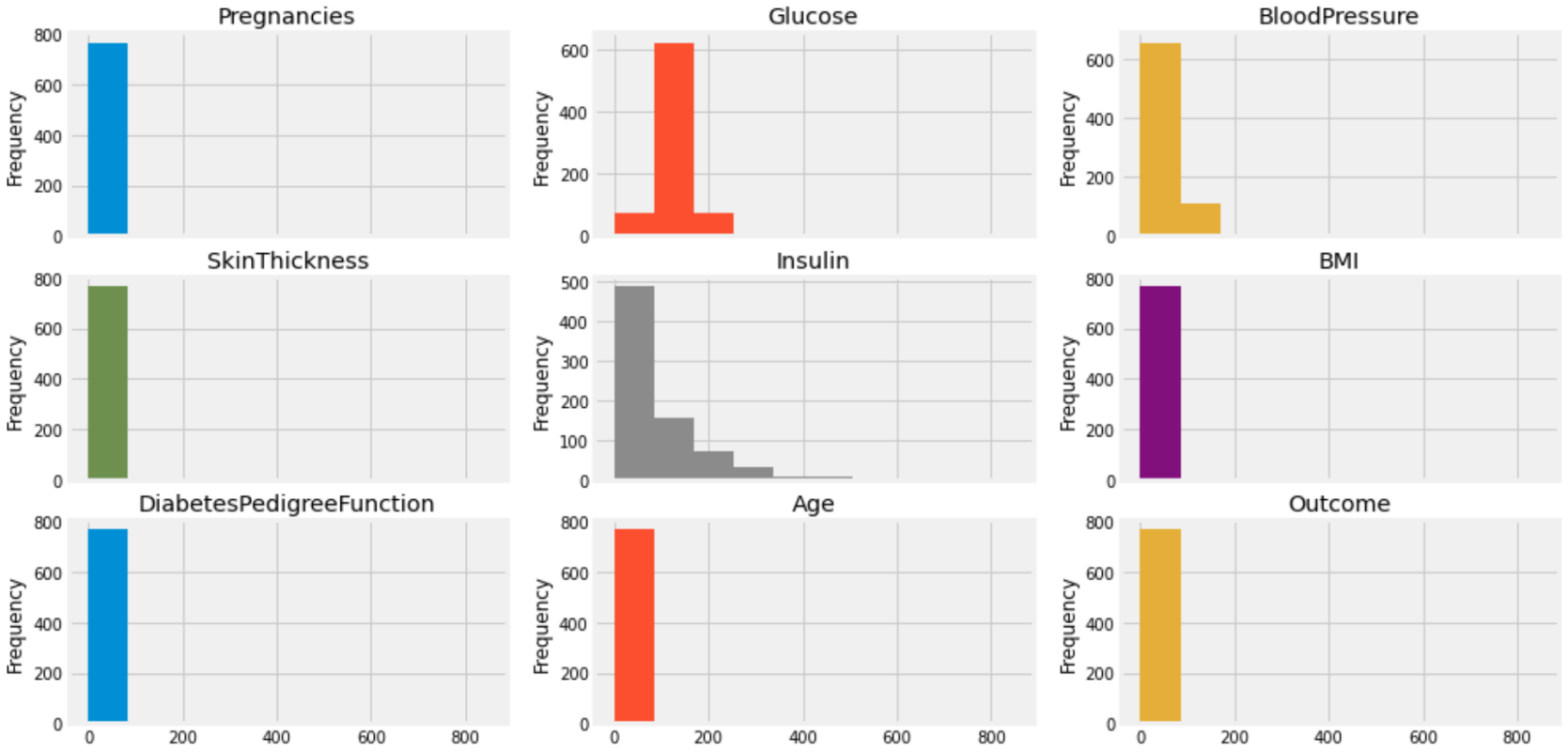


Histogram

- A graphical display of data using bars of different heights.
  - Display the shape and spread of continuous data.
  - Display the frequency of discrete data.
  - Efficient to visualize a large amount of data
- Each bar groups numbers into ranges.
  - Taller bars show that more data falls in that range.
  - visualizes buckets (bins) of quantities and shows frequencies of these buckets.

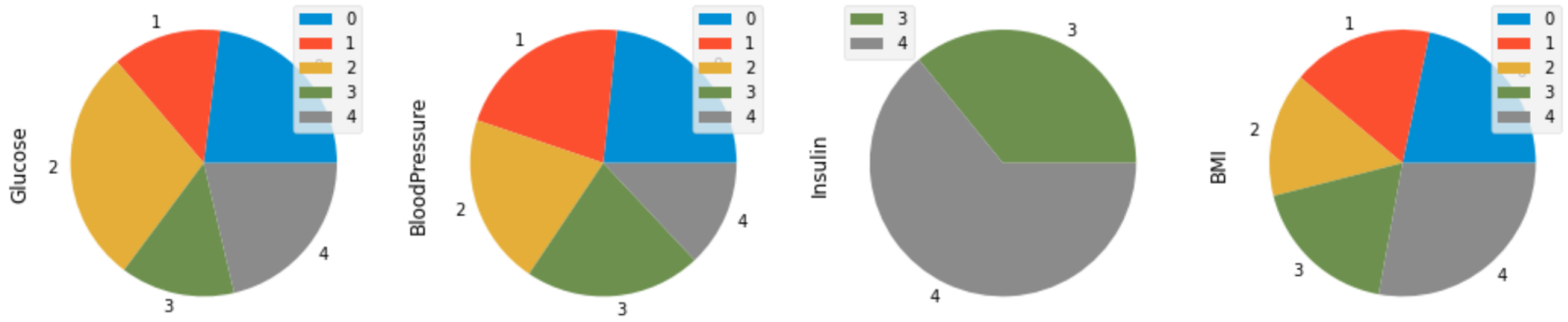
```
# Title above all subplots
data.plot(kind = 'hist',subplots=True, layout = (3,3) ,figsize=(15,8), legend=False,title = ['Pregnancies', 'Glucose','B
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A592F88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30AC291C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A8E03C8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A9125C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A9467C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A97B9C8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A9AEE88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A9E8688>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D30A9E8788>]],
      dtype=object)
```



```
:
1  # Pandas Pie Chart
2  df1.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'pie', subplots=True, figsize=(15,8))
```

[45]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001D302824B48>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001D3022739C8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001D3019008C8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001D3023401C8>],
      dtype=object)
```

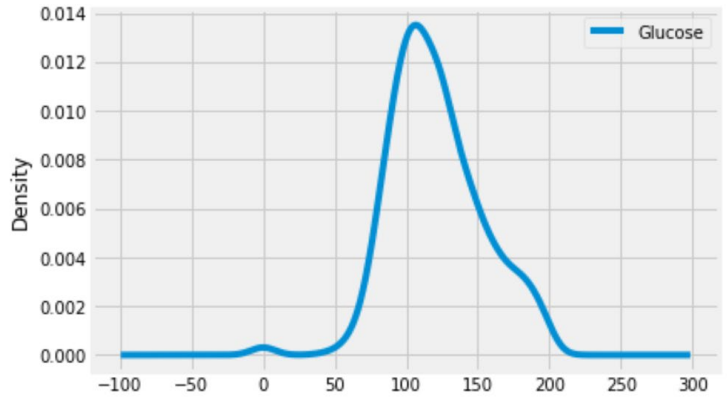


Pie Chart

- A pie chart is typically used with categorical features that have a relatively small number of values.
  - Pie charts are common in articles, but they are used less frequently in the technical publications because the size of relative areas can be hard to judge.

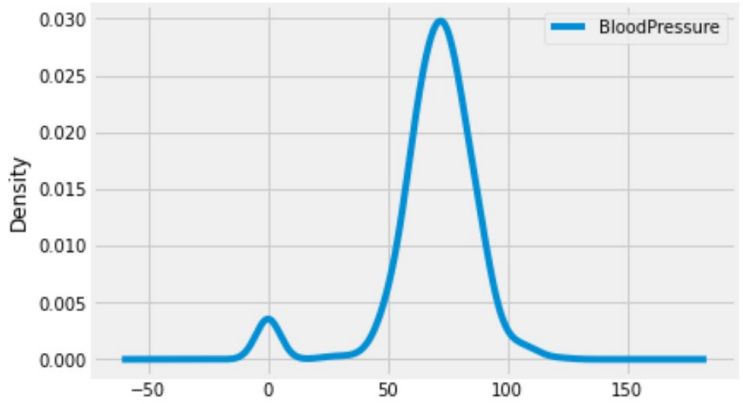
```
: 1 # Pandas density plot
  2 data.plot(data.index.name, ['Glucose'],kind = 'density')
```

```
[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1d30273ba08>
```




```
  1 # Pandas density plot
  2 data.plot(data.index.name, ['BloodPressure'],kind = 'density')
```

```
3]: <matplotlib.axes._subplots.AxesSubplot at 0x1d302710d08>
```

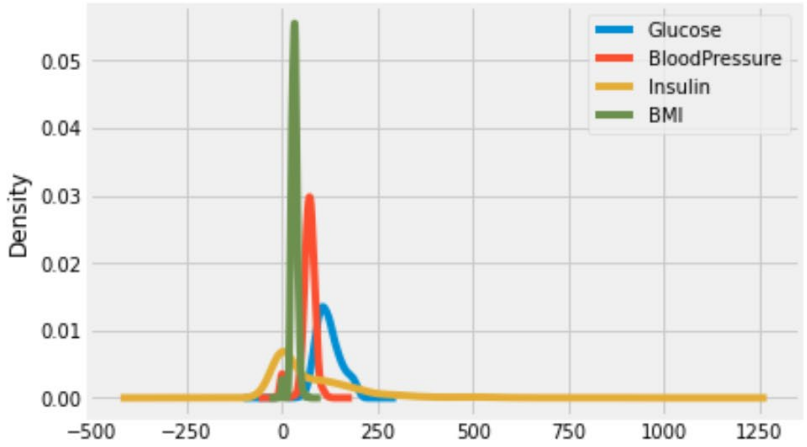


Density Plot

- A density plot is a smoothed, continuous version of a histogram estimated from the data.
- The most common form of estimation is known as kernel density estimation.
  - In this method, a continuous curve (the kernel) is drawn at every individual data point and all of these curves are then added together to make a single smooth probability density estimation.
  - The kernel most often used is a Gaussian
    - produces a Gaussian bell curve at each data point.

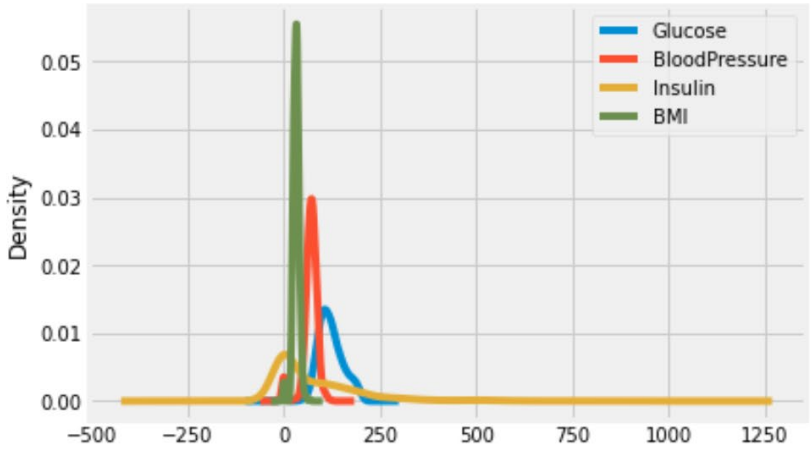
```
# Pandas density plot
data.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'density')
```

6]: <matplotlib.axes._subplots.AxesSubplot at 0x1d3026c5548>



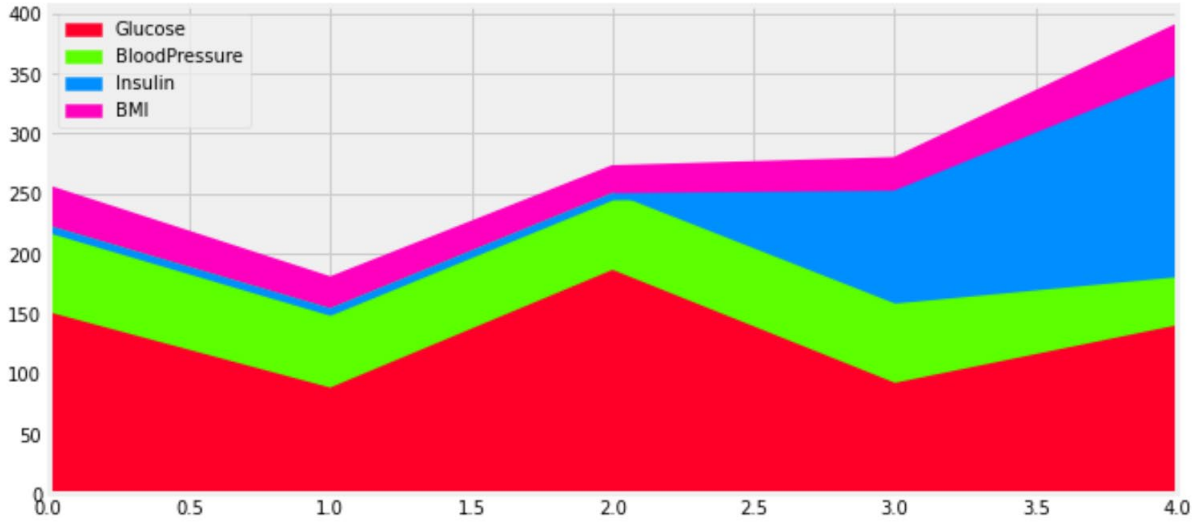

```
#  KDE plot
# Pandas density plot
data.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'kde')
```

0]: <matplotlib.axes._subplots.AxesSubplot at 0x1d302438c48>



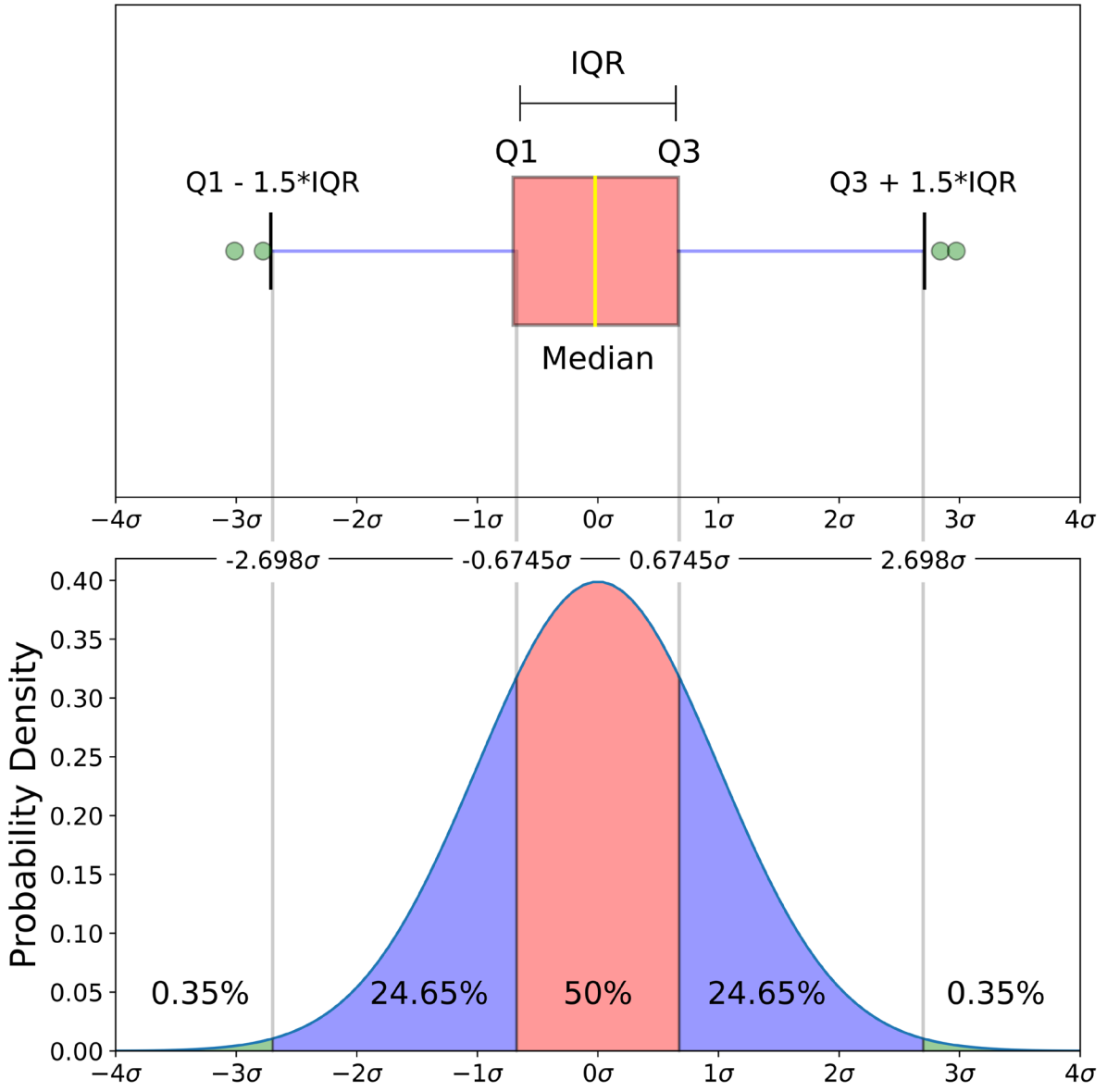
```
# Pandas colormap
df1.plot(data.index.name, ['Glucose','BloodPressure','Insulin','BMI'],kind = 'area', figsize=(10,5),
        colormap='gist_rainbow')
```

3]: <matplotlib.axes._subplots.AxesSubplot at 0x1d30b1da088>



Area Chart

- An area chart or area graph displays graphically quantitative data.
  - It is based on the line chart.
- The area between axis and line are commonly emphasized with colors and textures.
- Commonly one compares two or more quantities with an area chart.



https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

Box Plot

- Box plots are a popular way of visualizing a distribution.
- A boxplot include the five number summary:
  - The ends of the box are at the quartiles so that the box length is the interquartile range.
  - The median is marked by a line within the box.
  - Two lines (called whiskers) outside the box extend to the smallest (Minimum) and largest (Maximum) values.
  - IQR = Q3 – Q1

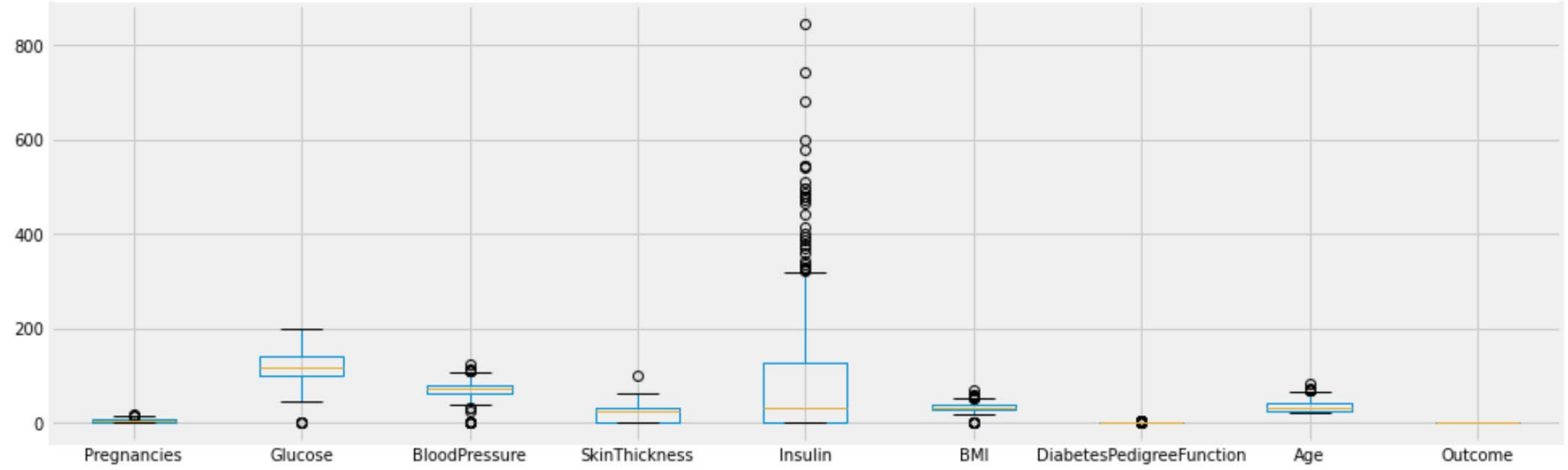
```
1 data.describe()
```

7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
1 data.plot(x=data.index.name,kind='box',figsize=(15,5))
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x1d3017c4588>



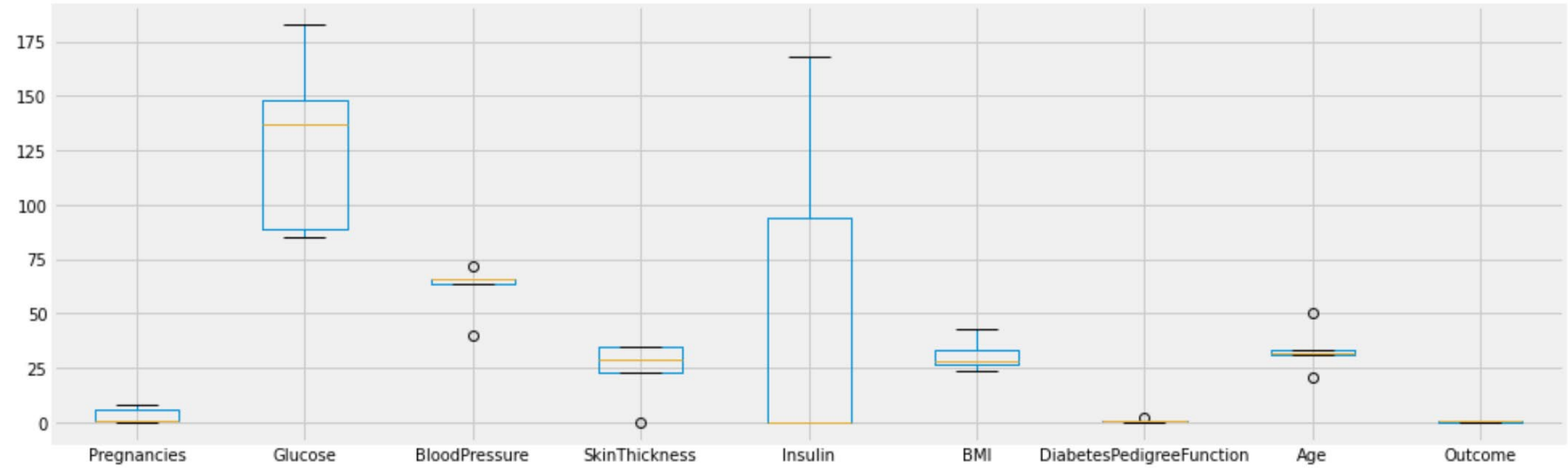
```
1 data.head()
```

8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
1 data[:5].plot(x=data.index.name,kind='box',figsize=(15,5))
```

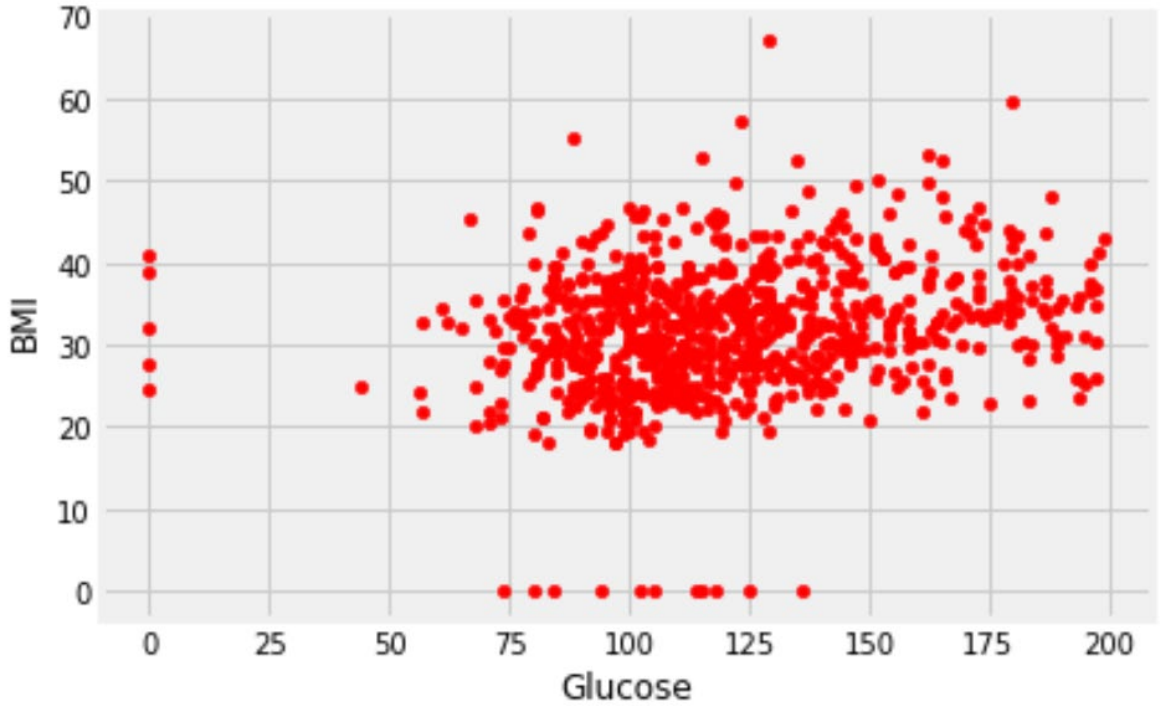
: <matplotlib.axes._subplots.AxesSubplot at 0x1d3017dd5c8>



```
1  # Pandas Scatter Plot
2  data.plot(x='Glucose',y='BMI',kind='scatter',color='R')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplo
r uppercase single-letter colors is deprecated since Matplotlib 3.
d.
  **self.kwds
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x1d306692b48>
```



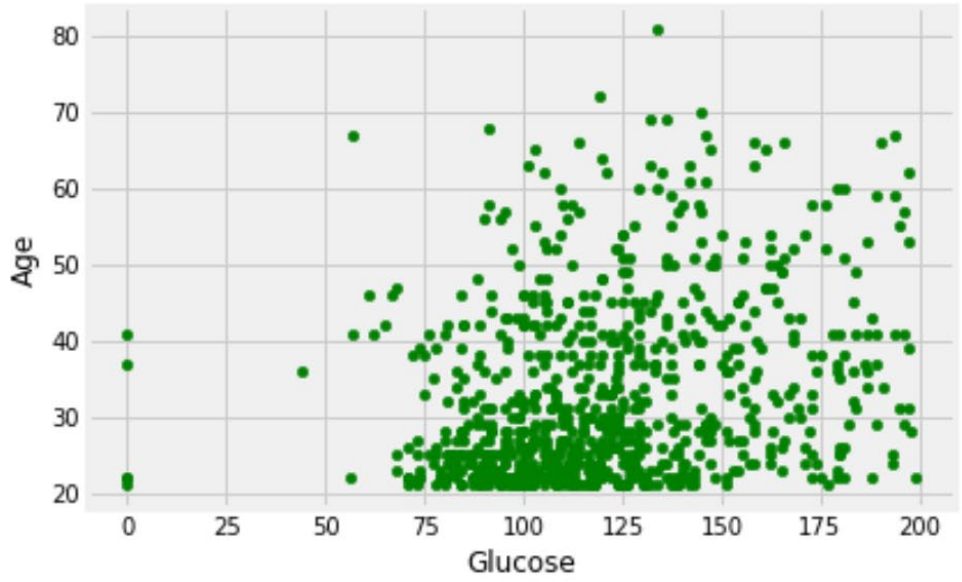
Scatter Plot

- Multivariate Plot
- An effective graphical method for determining if there is a relationship, pattern, or trend between two numeric features.
  - A scatter plot can suggest various kinds of correlations between features
    - Positive Correlation
    - Negative Correlation
    - No Correlation
- To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane.

In [74]:

```
1  # Pandas Scatter Plot
2  data.plot(x='Glucose',y='Age',kind='scatter',color='G')
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1d306a94948>

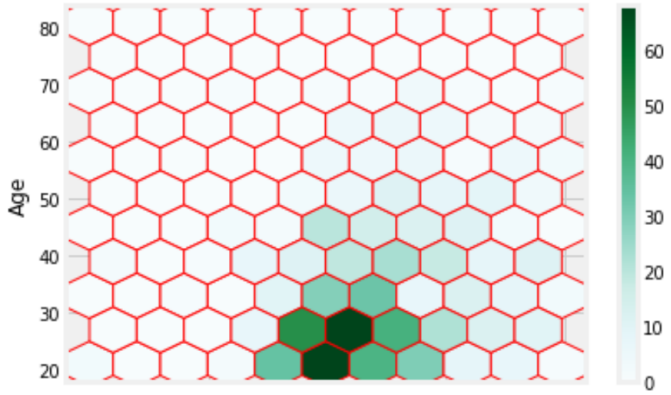


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
egnancies	1.000	0.129	0.141	-0.082	-0.074	0.018	-0.034	0.544	0.222
Glucose	0.129	1.000	0.153	0.057	0.331	0.221	0.137	0.264	0.467
dPressure	0.141	0.153	1.000	0.207	0.089	0.282	0.041	0.240	0.065
Thickness	-0.082	0.057	0.207	1.000	0.437	0.393	0.184	-0.114	0.075
Insulin	-0.074	0.331	0.089	0.437	1.000	0.198	0.185	-0.042	0.131
BMI	0.018	0.221	0.282	0.393	0.198	1.000	0.141	0.036	0.293
eeFunction	-0.034	0.137	0.041	0.184	0.185	0.141	1.000	0.034	0.174
Age	0.544	0.264	0.240	-0.114	-0.042	0.036	0.034	1.000	0.238
Outcome	0.222	0.467	0.065	0.075	0.131	0.293	0.174	0.238	1.000

```
# hexbin plot
data.plot(x='Glucose',y='Age',kind='hexbin',color='R', gridsize =10)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\tool
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed t
().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\tool
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed t
().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\tool
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed t
().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\tool
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed t
().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```

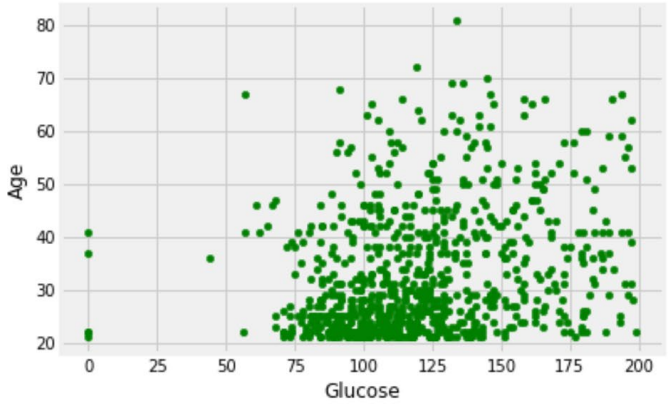
'6]: <matplotlib.axes._subplots.AxesSubplot at 0x1d306b1c608>



In [74]:

```
# Pandas Scatter Plot
data.plot(x='Glucose',y='Age',kind='scatter',color='G')
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1d306a94948>

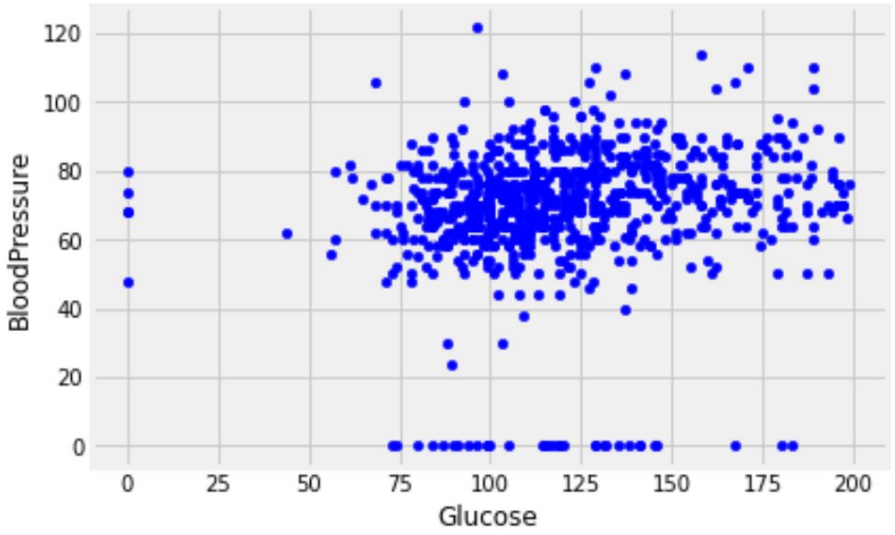


Hexbin Plot

- A Hexbin plot is useful to represent the relationship of 2 numerical variables when you have a lot of data point.
  - Instead of overlapping, the plotting window is split in several hexbins, and the number of points per hexbin is counted.
  - The color denotes this number of points.
  - you can change the size of the bins using the `gridsize` argument.

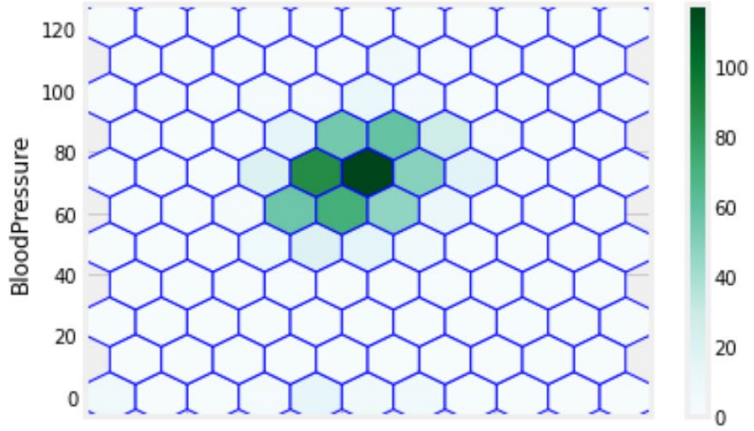
```
# Pandas Scatter Plot
data.plot(x='Glucose',y='BloodPressure',kind='scatter',color='B')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d306a18b48>
```




```
# hexbin plot
data.plot(x='BMI',y='BloodPressure',kind='hexbin',color='B', gridsize =10)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d306eb6cc8>
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
egnancies	1.000	0.129	0.141	-0.082	-0.074	0.018	-0.034	0.544	0.222
Glucose	0.129	1.000	0.153	0.057	0.331	0.221	0.137	0.264	0.467
dPressure	0.141	0.153	1.000	0.207	0.089	0.282	0.041	0.240	0.065
Thickness	-0.082	0.057	0.207	1.000	0.437	0.393	0.184	-0.114	0.075
Insulin	-0.074	0.331	0.089	0.437	1.000	0.198	0.185	-0.042	0.131
BMI	0.018	0.221	0.282	0.393	0.198	1.000	0.141	0.036	0.293
eeFunction	-0.034	0.137	0.041	0.184	0.185	0.141	1.000	0.034	0.174
Age	0.544	0.264	0.240	-0.114	-0.042	0.036	0.034	1.000	0.238
Outcome	0.222	0.467	0.065	0.075	0.131	0.293	0.174	0.238	1.000

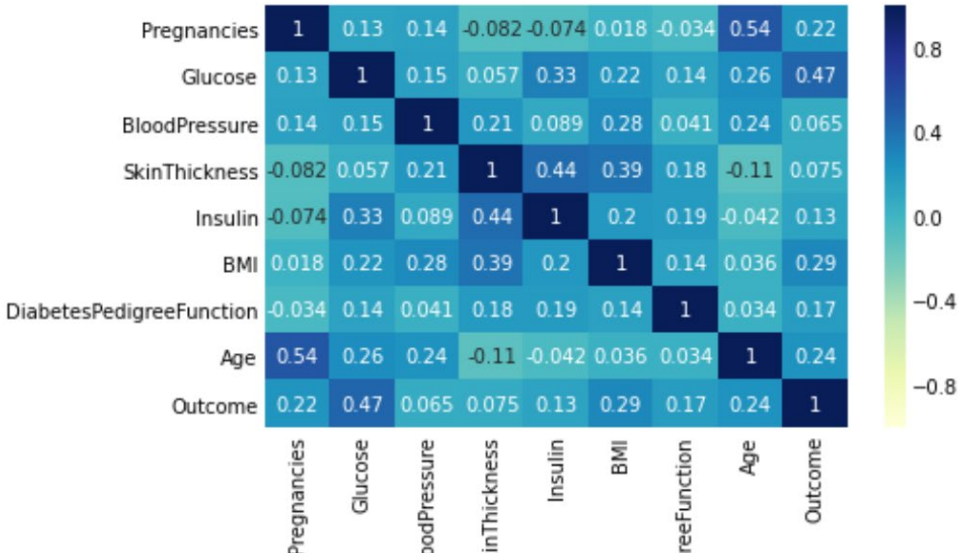
```
# pairwise Pearson correlations

correlations = data.corr(method='pearson')
correlations
```

46]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
sns.heatmap(data.corr(), vmin=-1, vmax=1, cmap="YlGnBu", annot = True)
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x1acaa3e9a48>

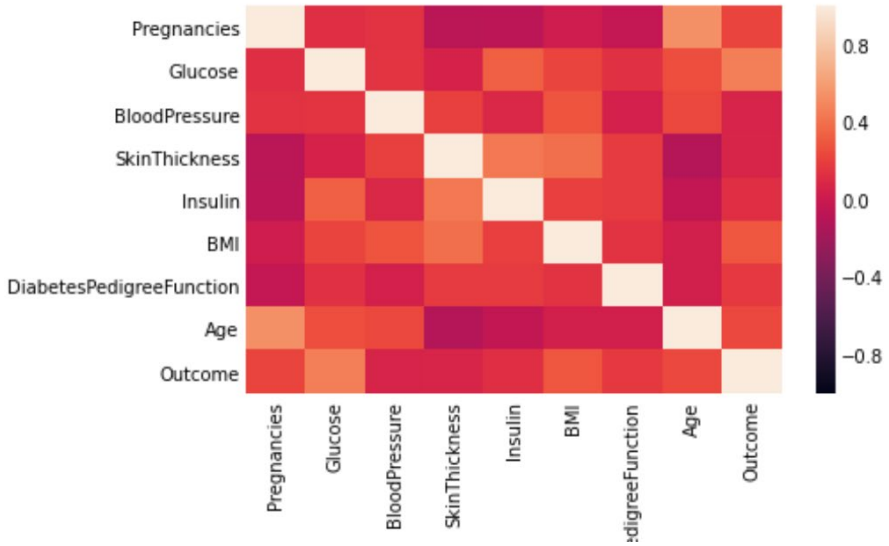


Correlation Matrix Plot (Heatmap)

- Calculate the correlation between each pair of features
  - Correlation gives an indication of how related the changes are between two variables.
    - Positive correlation
    - Negative correlation
    - None
- Plot the correlation matrix and get an idea of which variables have a high correlation with each other
  - This is useful to know, because some machine learning algorithms can have poor performance if there are highly correlated input variables in data

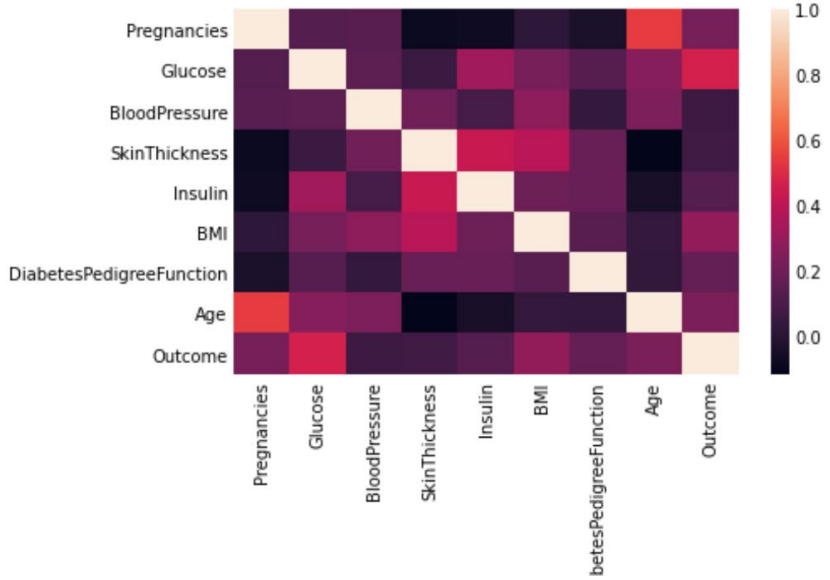
```
1 sns.heatmap(data.corr(), vmin=-1, vmax=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aca87e7548>
```




```
1 import seaborn as sns # another popular data visualization tool
2 sns.heatmap(data.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aca8f0df08>
```

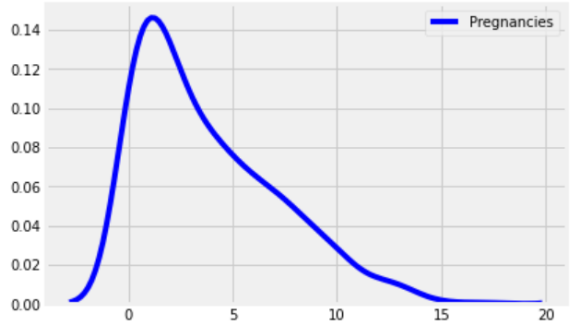


KDE Plot

- KDE (Kernel Density Estimate) Plot is used for visualizing the Probability Density of a continuous variable.
- It depicts the probability density at different values in a continuous variable.

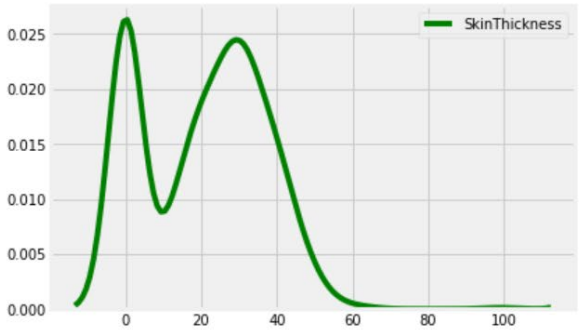
```
sns.kdeplot(data['Pregnancies'], color='b')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18becc82f08>
```



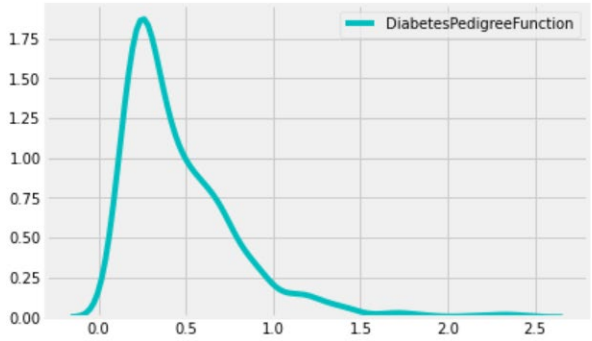

```
sns.kdeplot(data['SkinThickness'], color="g")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedd12ac8>
```



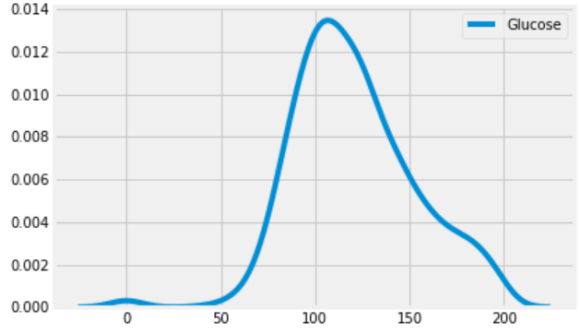

```
sns.kdeplot(data['DiabetesPedigreeFunction'], color="c")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedeebec8>
```



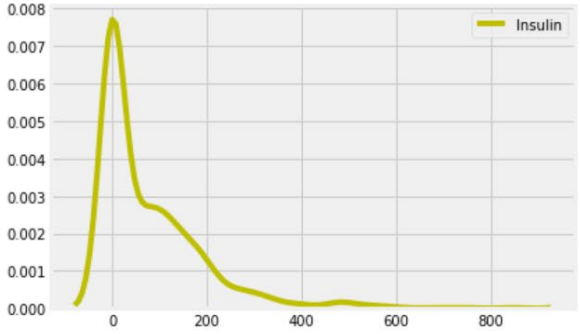

```
sns.kdeplot(data['Glucose'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bec872f48>
```



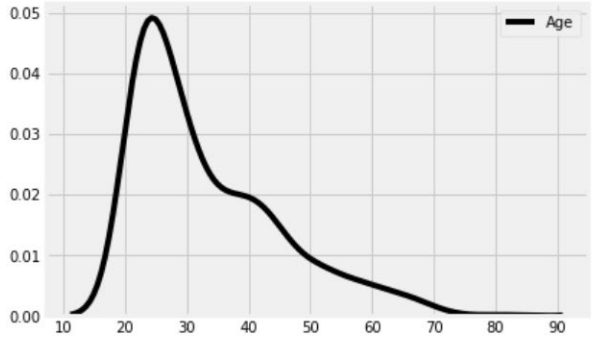

```
sns.kdeplot(data['Insulin'], color="y")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedd97f88>
```



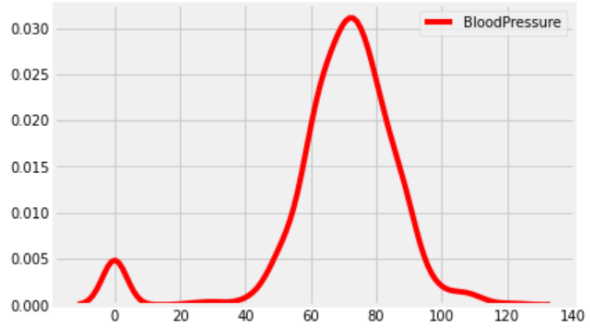

```
sns.kdeplot(data['Age'], color="k")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bee21b688>
```



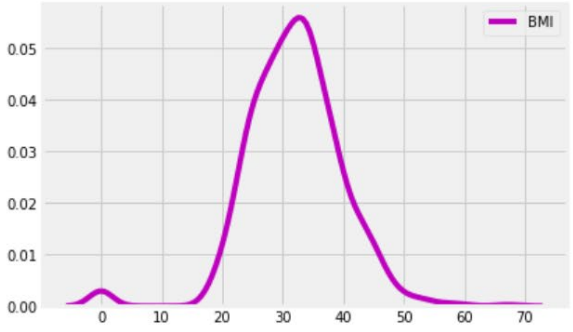

```
sns.kdeplot(data['BloodPressure'], color="r")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bedcc5488>
```




```
sns.kdeplot(data['BMI'], color="m")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bec59b888>
```

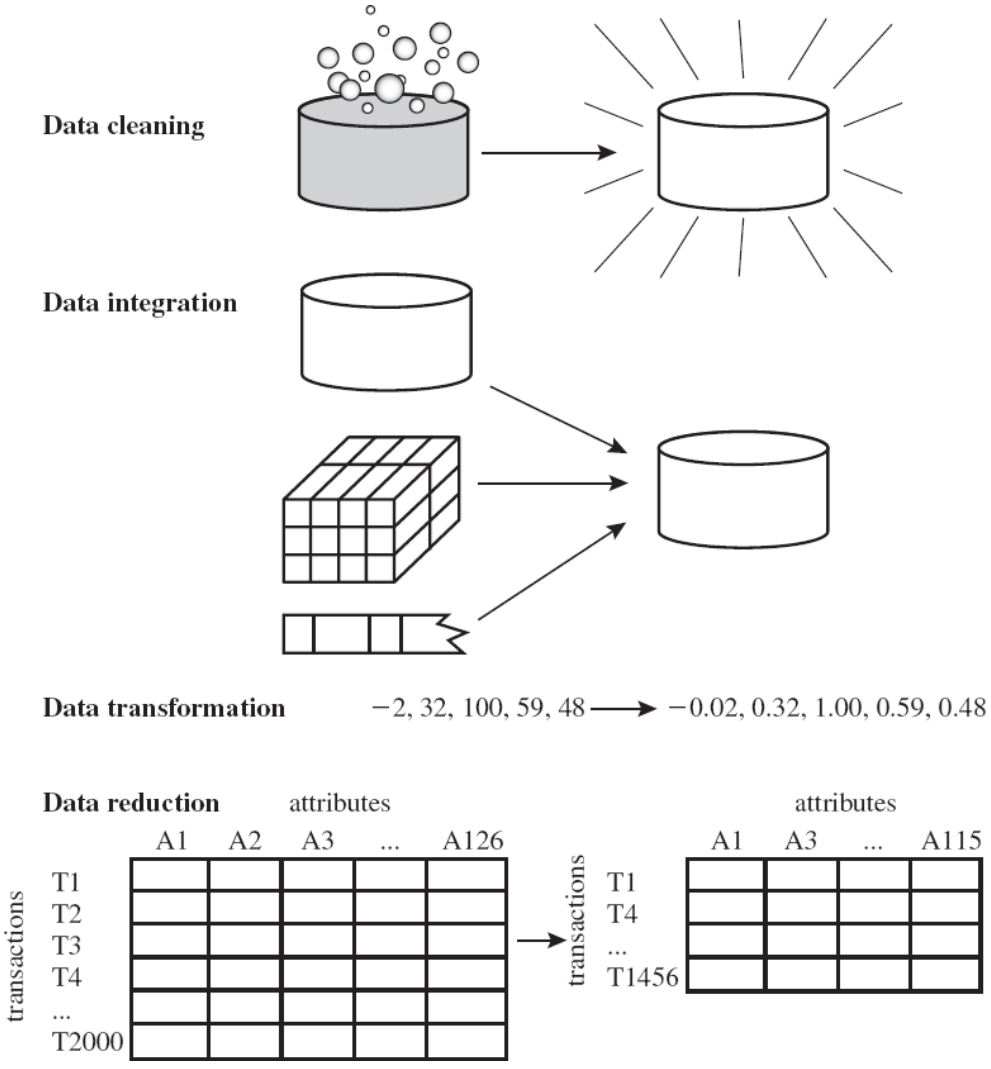


Lesson 6-A

Data Integration and Data Transformation

- Handling Redundancy, Summarizing, Aggregation, Binning, Normalization

Major Tasks in Data Preprocessing



Data Integration

- Combines data from multiple sources into a coherent store may cause *entity identification problem (schema conflict and value conflict)*
  - Schema conflict
    - *A.cust-id* ≡ *B.cust-number*
    - Integrate metadata from different sources
  - Data value conflict
    - “Bill Clinton” = “William Clinton”
    - Data codes for *pay_type* in one database may be “H” and “S” but “1” and “2” in another
    - Possible reasons: different representations, different scales, e.g., metric vs. British units

Handling Redundancy in Data Integration

- Redundant data occur often when integration of multiple databases
  - *Object identification*: The same attribute or object may have different names in different databases
  - *Derivable data:* One attribute may be a "derived" attribute in another table, e.g., age from dob
- Redundant attributes may be able to be detected by
  - Correlation coefficient for numeric data
  - Chi-square test for categorical data
- Careful integration of the data from multiple sources may help reduce/avoid redundancies and inconsistencies and improve mining speed and quality

4

Correlations between Attributes

- Correlation refers to the relationship between two attributes and how they related in terms of change.
  - Correlation analysis is used to detect attribute redundancy and improve performance of model
  - The performance of some machine learning models like linear and logistic regression may not be good if there are highly correlated attributes in the dataset.
  - We need to review all of the pairwise correlations of the attribute in the dataset.

Correlation Analysis

- Correlation analysis
  - Measure how strongly one attribute is related (dependent) with the other
  - Numeric data
    - Covariance
    - Pearson's Correlation Coefficient
    - `corr()` Pandas function
  - Categorical data
    - Chi-square test
    - `chi2_contingency()` SciPy function
    - `from sklearn.attribute_selection import chi2`

Pearson Correlation Coefficient between Attributes

$$corr(X,Y) = r_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

- $r_{X,Y}$ can take a range of values from +1 to -1
- If $r_{X,Y} > 0$: X and Y are positively correlated (X's values increase as Y's), The higher, the stronger correlation.
- $r_{X,Y} = 0$: independent;
- $r_{X,Y} < 0$: negatively correlated

Chi-Square Test

- Summarize the data in the two-way contingency table, the observed table
  - This table represents the observed counts of each cell
- Calculate the expected count for each cell in the table to find the expected table from observed counts
  - This table displays what the counts for each cell would be for sample data if there were not relations between two attributes
  - To find the expected count for each cell in the expected table we multiply the marginal row and column totals for that cell and divide by the overall total in the observed table
  - i.e. for each cell this is
  - $E = (row\ total \times column\ total) \,/\, total\ number\ of\ data$

Chi-Square Test

- Compute a Chi-Square test statistic as follows:
  - $\chi^2 = \Sigma(O_i - E_i)^2/E_i$
    - where O_i is an observed count of each cell of the contingency table, and E_i is an expected count of each cell of the contingency table
- With Chi-Square test statistic, a significance level (α) chosen, and the degree of freedom for the Chi-Square distribution, we can make a decision.
  - Degree of Freedom (df) = (number of rows – 1) x (numbers of columns – 1)
  - a significance level = 0.001

Chi-Square Test

- The decision is made by
  - Either comparing the value of test Chi-Square statistic to a critical chi-square value at a chosen significance level, α (rejection region approach)
    - Test statistic >= Critical value: reject null hypothesis, attributes are dependent (H_a)
    - Test statistic < Critical value: fail to reject null hypothesis, attributes are independent (H_0)
  - or finding the probability of getting of test Chi-Square statistic (p-value approach)
    - p-value <= α (a chosen significance level): reject null hypothesis, attributes are dependent (H_a)
    - p-value > α: fail to reject null hypothesis, attributes are independent (H_0)

Data Transformation

- We need to transform data in order that
  - resulting modeling process can be more efficient
  - modeling accuracy can be improved

How to Transform Data

- Summarizing and aggregation
  - Summarizing: summarizing (with group by features) using a statistical operation such as mean, max, min, standard deviation, etc. over dataset or specified feature(s)
  - Aggregation: aggregating (with group by features) using one or more statistical operations over the specified feature(s)
- Discretization
  - Raw values of a numeric feature are replaced by interval labels
  - Binning, histogram
- Normalization
  - Data are scaled so as to fall within a smaller range such as -1.0 to 1.0 or 0.0 to 1.0
  - z-score, min-max

Summarizing

- Summarizing the dataset or each feature
  - using different statistics such as `mean, max, min, sum, count, standard deviation`, etc.
- Summarizing groups in the dataset
  - `groupby` splits the dataset into different groups depending on a selected feature or more
  - Functions like `max, min, mean`, etc. can be applied to the `groupby` object

```
# import packages we need for loading the data set
import pandas as pd # to store tabular data
import numpy as np # to do some math
import matplotlib.pyplot as plt # a popular data visualization tool
import seaborn as sns # another popular data visualization tool

# allows the notebook to render graphics
%matplotlib inline

plt.style.use('fivethirtyeight') # a popular data visualization theme
```

```
# load in the data set. This data was scrapped from Basketball-reference
data = pd.read_csv("nba.csv") # load the CSV file using pandas.read_csv function
```

```
data.head()
```

|:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

```
data.shape # (# rows, # cols)
```

|: (458, 9)

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 458 entries, 0 to 457
Data columns (total 9 columns):
Name        457 non-null object
Team        457 non-null object
Number      457 non-null float64
Position    457 non-null object
Age         457 non-null float64
Height      457 non-null object
Weight      457 non-null float64
College     373 non-null object
Salary      446 non-null float64
dtypes: float64(4), object(5)
memory usage: 32.3+ KB
```

```
data.describe()
```

6]:

	Number	Age	Weight	Salary
count	457.000000	457.000000	457.000000	4.460000e+02
mean	17.678337	26.938731	221.522976	4.842684e+06
std	15.966090	4.404016	26.368343	5.229238e+06
min	0.000000	19.000000	161.000000	3.088800e+04
25%	5.000000	24.000000	200.000000	1.044792e+06
50%	13.000000	26.000000	220.000000	2.839073e+06
75%	25.000000	30.000000	240.000000	6.500000e+06
max	99.000000	40.000000	307.000000	2.500000e+07

```
1 data['Team'].value_counts()
```

```
New Orleans Pelicans      19
Memphis Grizzlies         18
New York Knicks           16
Milwaukee Bucks           16
Denver Nuggets            15
Portland Trail Blazers    15
San Antonio Spurs         15
Brooklyn Nets             15
Toronto Raptors           15
Miami Heat                15
Sacramento Kings          15
Phoenix Suns              15
Atlanta Hawks             15
Washington Wizards        15
Utah Jazz                 15
Los Angeles Lakers        15
Chicago Bulls             15
Houston Rockets           15
Los Angeles Clippers      15
Philadelphia 76ers        15
Charlotte Hornets         15
Dallas Mavericks          15
Indiana Pacers            15
Golden State Warriors     15
Boston Celtics            15
Detroit Pistons           15
Oklahoma City Thunder     15
Cleveland Cavaliers       15
Minnesota Timberwolves    14
Orlando Magic             14
Name: Team, dtype: int64
```

```
1 data.groupby('Team').groups.keys()
```

```
dict_keys(['Atlanta Hawks', 'Boston Celtics', 'Brooklyn Nets', 'Charlotte Hornets', 'Chicago Bulls', 'Cleveland Cavaliers',
'Dallas Mavericks', 'Denver Nuggets', 'Detroit Pistons', 'Golden State Warriors', 'Houston Rockets', 'Indiana Pacers', 'Los
Angeles Clippers', 'Los Angeles Lakers', 'Memphis Grizzlies', 'Miami Heat', 'Milwaukee Bucks', 'Minnesota Timberwolves', 'Ne
w Orleans Pelicans', 'New York Knicks', 'Oklahoma City Thunder', 'Orlando Magic', 'Philadelphia 76ers', 'Phoenix Suns', 'Por
tland Trail Blazers', 'Sacramento Kings', 'San Antonio Spurs', 'Toronto Raptors', 'Utah Jazz', 'Washington Wizards'])
```

```
1 len(data.groupby('Team').groups['Los Angeles Lakers'])
```

```
15
```

```
1 data.groupby('Team').mean()
```

]:

Team	Number	Age	Weight	Salary
Atlanta Hawks	19.000000	28.200000	221.266667	4.860197e+06
Boston Celtics	31.866667	24.733333	219.466667	4.181505e+06
Brooklyn Nets	18.266667	25.600000	215.600000	3.501898e+06
Charlotte Hornets	17.133333	26.133333	220.400000	5.222728e+06
Chicago Bulls	19.200000	27.400000	218.933333	5.785559e+06
Cleveland Cavaliers	14.466667	29.533333	227.866667	7.642049e+06
Dallas Mavericks	20.000000	29.733333	227.000000	4.746582e+06
Denver Nuggets	15.266667	25.733333	217.533333	4.294424e+06
Detroit Pistons	17.266667	26.200000	222.200000	4.477884e+06
Golden State Warriors	20.866667	27.666667	224.600000	5.924600e+06
Houston Rockets	14.666667	26.866667	220.333333	5.018868e+06
Indiana Pacers	18.933333	26.400000	222.266667	4.450122e+06
Los Angeles Clippers	19.533333	29.466667	219.733333	6.323643e+06
Los Angeles Lakers	16.066667	27.533333	227.066667	4.784695e+06
Memphis Grizzlies	15.555556	28.388889	218.000000	5.467920e+06
Miami Heat	10.466667	28.933333	218.400000	6.347359e+06
Milwaukee Bucks	20.000000	24.562500	224.062500	4.350220e+06
Minnesota Timberwolves	19.571429	26.357143	228.642857	4.593054e+06
New Orleans Pelicans	17.000000	26.894737	221.000000	4.355304e+06
New York Knicks	13.250000	27.000000	223.625000	4.581494e+06
Oklahoma City Thunder	14.000000	27.066667	229.400000	6.251020e+06
Orlando Magic	16.428571	25.071429	213.357143	4.297248e+06
Philadelphia 76ers	18.066667	24.600000	222.133333	2.213778e+06
Phoenix Suns	15.466667	25.866667	218.600000	4.229676e+06
Portland Trail Blazers	16.000000	25.066667	218.600000	3.220121e+06
Sacramento Kings	16.933333	26.800000	221.333333	4.778911e+06
San Antonio Spurs	17.933333	31.600000	223.933333	5.629516e+06
Toronto Raptors	22.466667	26.133333	221.800000	4.741174e+06
Utah Jazz	17.866667	24.466667	220.000000	4.204006e+06
Washington Wizards	17.600000	27.866667	219.000000	5.088576e+06

```
1 data.groupby('Team')['Age', 'Weight'].mean()
```

]:

Team	Age	Weight
Atlanta Hawks	28.200000	221.266667
Boston Celtics	24.733333	219.466667
Brooklyn Nets	25.600000	215.600000
Charlotte Hornets	26.133333	220.400000
Chicago Bulls	27.400000	218.933333
Cleveland Cavaliers	29.533333	227.866667
Dallas Mavericks	29.733333	227.000000
Denver Nuggets	25.733333	217.533333
Detroit Pistons	26.200000	222.200000
Golden State Warriors	27.666667	224.600000
Houston Rockets	26.866667	220.333333
Indiana Pacers	26.400000	222.266667
Los Angeles Clippers	29.466667	219.733333
Los Angeles Lakers	27.533333	227.066667
Memphis Grizzlies	28.388889	218.000000
Miami Heat	28.933333	218.400000
Milwaukee Bucks	24.562500	224.062500
Minnesota Timberwolves	26.357143	228.642857
New Orleans Pelicans	26.894737	221.000000
New York Knicks	27.000000	223.625000
Oklahoma City Thunder	27.066667	229.400000
Orlando Magic	25.071429	213.357143
Philadelphia 76ers	24.600000	222.133333
Phoenix Suns	25.866667	218.600000
Portland Trail Blazers	25.066667	218.600000
Sacramento Kings	26.800000	221.333333
San Antonio Spurs	31.600000	223.933333
Toronto Raptors	26.133333	221.800000
Utah Jazz	24.466667	220.000000
Washington Wizards	27.866667	219.000000

Aggregation

- Aggregate using one or more operations over the specified features per group
  - `pandas.DataFrame.aggregate`
  - *`agg`* is an alias for *`aggregate`*. Use the alias.

```
1 data.groupby(['Team', 'Position'])['Age', 'Weight'].mean()
```

Team	Position	Age	Weight
Atlanta Hawks	C	28.333333	250.00
	PF	28.250000	239.50
	PG	24.500000	179.00
	SF	29.000000	210.50
	SG	29.500000	208.00
...	...	...	...
Washington Wizards	C	30.666667	244.00
	PF	30.000000	247.50
	PG	27.500000	192.50
	SF	25.500000	208.25
	SG	27.250000	210.00

149 rows × 2 columns

```
1 data.groupby(['Team', 'Position']).agg({'Age': 'mean', 'Weight':'mean'})
```

Team	Position	Age	Weight
Atlanta Hawks	C	28.333333	250.00
	PF	28.250000	239.50
	PG	24.500000	179.00
	SF	29.000000	210.50
	SG	29.500000	208.00
...	...	...	...
Washington Wizards	C	30.666667	244.00
	PF	30.000000	247.50
	PG	27.500000	192.50
	SF	25.500000	208.25
	SG	27.250000	210.00

149 rows × 2 columns

```
1 data.groupby(['Team', 'Position']).agg({'Age': ['mean', 'max', 'min'], 'Weight':['mean', 'max', 'min']})
```

2]:

		Age			Weight		
		mean	max	min	mean	max	min
Team	Position						
Atlanta Hawks	C	28.333333	31.0	24.0	250.00	260.0	245.0
	PF	28.250000	31.0	24.0	239.50	246.0	235.0
	PG	24.500000	27.0	22.0	179.00	186.0	172.0
	SF	29.000000	32.0	26.0	210.50	220.0	201.0
	SG	29.500000	35.0	24.0	208.00	225.0	190.0
...	...	...	...	...	...	...	...
Washington Wizards	C	30.666667	33.0	27.0	244.00	250.0	240.0
	PF	30.000000	34.0	26.0	247.50	250.0	245.0
	PG	27.500000	30.0	25.0	192.50	195.0	190.0
	SF	25.500000	30.0	20.0	208.25	225.0	198.0
	SG	27.250000	33.0	22.0	210.00	220.0	195.0

149 rows × 6 columns

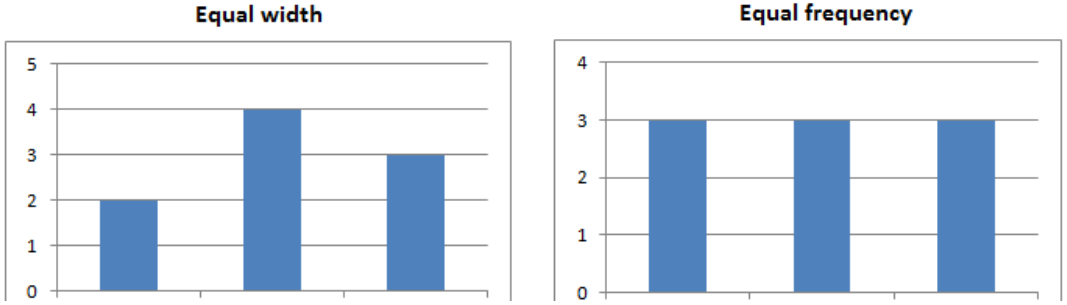
```
1 data.groupby('Team').agg({'Age': ['mean', 'max', 'min'], 'Weight':['mean', 'max', 'min']})
```

]:

	Age			Weight		
	mean	max	min	mean	max	min
Team						
Atlanta Hawks	28.200000	35.0	22.0	221.266667	260.0	172.0
Boston Celtics	24.733333	29.0	20.0	219.466667	260.0	180.0
Brooklyn Nets	25.600000	32.0	21.0	215.600000	275.0	175.0
Charlotte Hornets	26.133333	31.0	21.0	220.400000	289.0	184.0
Chicago Bulls	27.400000	35.0	21.0	218.933333	275.0	161.0
Cleveland Cavaliers	29.533333	35.0	24.0	227.866667	275.0	179.0
Dallas Mavericks	29.733333	37.0	22.0	227.000000	275.0	185.0
Denver Nuggets	25.733333	36.0	20.0	217.533333	280.0	175.0
Detroit Pistons	26.200000	36.0	20.0	222.200000	279.0	172.0
Golden State Warriors	27.666667	33.0	20.0	224.600000	273.0	175.0
Houston Rockets	26.866667	38.0	22.0	220.333333	265.0	185.0
Indiana Pacers	26.400000	30.0	20.0	222.266667	255.0	180.0
Los Angeles Clippers	29.466667	39.0	23.0	219.733333	265.0	175.0
Los Angeles Lakers	27.533333	37.0	20.0	227.066667	270.0	175.0
Memphis Grizzlies	28.388889	39.0	21.0	218.000000	270.0	165.0
Miami Heat	28.933333	36.0	20.0	218.400000	265.0	165.0
Milwaukee Bucks	24.562500	32.0	19.0	224.062500	265.0	190.0
Minnesota Timberwolves	26.357143	40.0	20.0	228.642857	307.0	189.0
New Orleans Pelicans	26.894737	31.0	23.0	221.000000	270.0	170.0
New York Knicks	27.000000	34.0	20.0	223.625000	278.0	195.0
Oklahoma City Thunder	27.066667	38.0	21.0	229.400000	255.0	185.0
Orlando Magic	25.071429	32.0	20.0	213.357143	260.0	169.0
Philadelphia 76ers	24.600000	37.0	20.0	222.133333	275.0	175.0
Phoenix Suns	25.866667	33.0	19.0	218.600000	260.0	175.0
Portland Trail Blazers	25.066667	34.0	20.0	218.600000	265.0	173.0
Sacramento Kings	26.800000	36.0	22.0	221.333333	270.0	175.0
San Antonio Spurs	31.600000	40.0	22.0	223.933333	290.0	185.0
Toronto Raptors	26.133333	36.0	20.0	221.800000	255.0	190.0
Utah Jazz	24.466667	28.0	20.0	220.000000	265.0	179.0
Washington Wizards	27.866667	34.0	20.0	219.000000	250.0	190.0

Binning

- Binning methods sort the data in ascending order and then partition them into a set of equal-frequency or equal-size bins.
  - Equal-width (distance) binning
  - Equal-frequency (depth) binning
- There are several different terms for binning
  - bucketing, discrete binning, discretization or quantization



Equal-frequency binning

- It divides the range into N intervals, each containing the same number of samples
  - Data scaling (smooth out the data to remove the noise)
    - Smoothing by bin means
    - Smoothing by bin medians
    - Smoothing by bin boundaries
- Given the sorted data:
  - 0, 4, 12, 16, 16, 18, 24, 26, 28
  - Equal-frequency bins of size 3 (i.e. each bin contains three values)

Equal-width (distance) binning

- It divides the range into N intervals of equal size:
  - If A and B are the lowest and highest values of the feature, the width of intervals with be: $W = (B - A)/ N$.
- Given the sorted data:
  - 0, 4, 12, 16, 16, 18, 24, 26, 28
- Equal-width bins of size 10 (i.e. each bin's interval range is 10)

pandas.qcut

- `qcut` is a quantile-based discretization function.
  - divides up the underlying data into equal size bins to make sure the **distribution** of data in the bins is (almost) equal
  - defines the bins using percentiles based on the distribution of the data
  - Using `qcut`, we can create 4 bins (aka quartiles), 5 bins (aka quintiles) and 10 bins (aka deciles)

```
1 data['Glucose'].describe()
```

```
count    768.000000
mean     120.894531
std       31.972618
min        0.000000
25%       99.000000
50%      117.000000
75%      140.250000
max      199.000000
Name: Glucose, dtype: float64
```

```
1 pd.qcut(data['Glucose'], q=4)
```

```
0      (140.25, 199.0]
1       (-0.001, 99.0]
2      (140.25, 199.0]
3       (-0.001, 99.0]
4      (117.0, 140.25]
             ...
763      (99.0, 117.0]
764    (117.0, 140.25]
765    (117.0, 140.25]
766    (117.0, 140.25]
767     (-0.001, 99.0]
Name: Glucose, Length: 768, dtype: category
Categories (4, interval[float64]): [(-0.001, 99.0] < (99.0, 117.0] < (117.0, 140.25] < (140.25, 199.0]]
```

```
1 pd.qcut(data['Glucose'], q=4).value_counts()
```

```
(-0.001, 99.0]     197
(99.0, 117.0]      194
(140.25, 199.0]    192
(117.0, 140.25]    185
Name: Glucose, dtype: int64
```

```
1 pd.qcut(data['Glucose'], q=10)
```

```
79]: 0      (147.0, 167.0]
     1      (43.999, 86.2]
     2      (167.0, 199.0]
     3        (86.2, 95.0]
     4      (135.0, 147.0]
                 ...
     763     (95.0, 102.0]
     764    (117.0, 125.0]
     765    (117.0, 125.0]
     766    (125.0, 135.0]
     767      (86.2, 95.0]
     Name: Glucose, Length: 768, dtype: category
     Categories (10, interval[float64]): [(43.999, 86.2] < (86.2, 95.0] < (95.0, 102.0] < (102.0, 109.0] ... (125.0, 135.0] < (13
     5.0, 147.0] < (147.0, 167.0] < (167.0, 199.0]]
```

```
1 pd.qcut(data['Glucose'], q=10).value_counts()
```

```
30]: (117.0, 125.0]    80
     (135.0, 147.0]    78
     (102.0, 109.0]    78
     (86.2, 95.0]      78
     (109.0, 117.0]    77
     (43.999, 86.2]    77
     (167.0, 199.0]    76
     (95.0, 102.0]     76
     (147.0, 167.0]    72
     (125.0, 135.0]    71
     Name: Glucose, dtype: int64
```

In [98]:
```
bin_labels_5 = ['very low', 'low', 'medium', 'high', 'very high']
data['quantile_glucose'] = pd.qcut(data['Glucose'], q = 5, labels=bin_labels_5)
data.head()
```

Out[98]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	quantile_glucose
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1	very high
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0	very low
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1	very high
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	very low
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	high

In [99]:
```
data['quantile_glucose'].value_counts()
```

Out[99]:
```
medium       157
very low     155
low          154
high         149
very high    148
Name: quantile_glucose, dtype: int64
```

pandas.cut

- Bin values into discrete intervals
  - Use *cut* when you need to segment and sort data values into equal width bins
    - No guarantee about the distribution of values in each bin
  - This function is also useful for going from a continuous variable to a categorical variable
    - convert ages to groups of age ranges

```
1 pd.cut(data['Glucose'], bins=4)
```

```
: 0       (99.5, 149.25]
  1        (49.75, 99.5]
  2      (149.25, 199.0]
  3        (49.75, 99.5]
  4       (99.5, 149.25]
             ...
  763     (99.5, 149.25]
  764     (99.5, 149.25]
  765     (99.5, 149.25]
  766     (99.5, 149.25]
  767      (49.75, 99.5]
  Name: Glucose, Length: 768, dtype: category
  Categories (4, interval[float64]): [(-0.199, 49.75] < (49.75, 99.5] < (99.5, 149.25] < (149.25, 199.0]]
```

```
1 pd.cut(data['Glucose'], bins=4).value_counts()
```

```
: (99.5, 149.25]     428
  (49.75, 99.5]      191
  (149.25, 199.0]    143
  (-0.199, 49.75]      6
  Name: Glucose, dtype: int64
```

qcut vs cut

- If you want equal distribution of the values in the bins, use `qcut`
- If you want to define your own numeric bin ranges (bin edges), then use `cut`

```
1 data['Glucose'].describe()
```

```
count    768.000000
mean     120.894531
std       31.972618
min        0.000000
25%       99.000000
50%      117.000000
75%      140.250000
max      199.000000
Name: Glucose, dtype: float64
```

```
1 pd.qcut(data['Glucose'], q=4)
```

```
0      (140.25, 199.0]
1       (-0.001, 99.0]
2      (140.25, 199.0]
3       (-0.001, 99.0]
4      (117.0, 140.25]
             ...
763     (99.0, 117.0]
764    (117.0, 140.25]
765    (117.0, 140.25]
766    (117.0, 140.25]
767     (-0.001, 99.0]
Name: Glucose, Length: 768, dtype: category
Categories (4, interval[float64]): [(-0.001, 99.0] < (99.0, 117.0] < (117.0, 140.25] < (140.25, 199.0]]
```

```
1 pd.qcut(data['Glucose'], q=4).value_counts()
```

```
(-0.001, 99.0]     197
(99.0, 117.0]      194
(140.25, 199.0]    192
(117.0, 140.25]    185
Name: Glucose, dtype: int64
```

```
1 pd.cut(data['Glucose'], bins=4)
```

```
0      (99.5, 149.25]
1       (49.75, 99.5]
2     (149.25, 199.0]
3       (49.75, 99.5]
4      (99.5, 149.25]
             ...
763    (99.5, 149.25]
764    (99.5, 149.25]
765    (99.5, 149.25]
766    (99.5, 149.25]
767     (49.75, 99.5]
Name: Glucose, Length: 768, dtype: category
Categories (4, interval[float64]): [(-0.199, 49.75] < (49.75, 99.5] < (99.5, 149.25] < (149.25, 199.0]]
```

```
1 pd.cut(data['Glucose'], bins=4).value_counts()
```

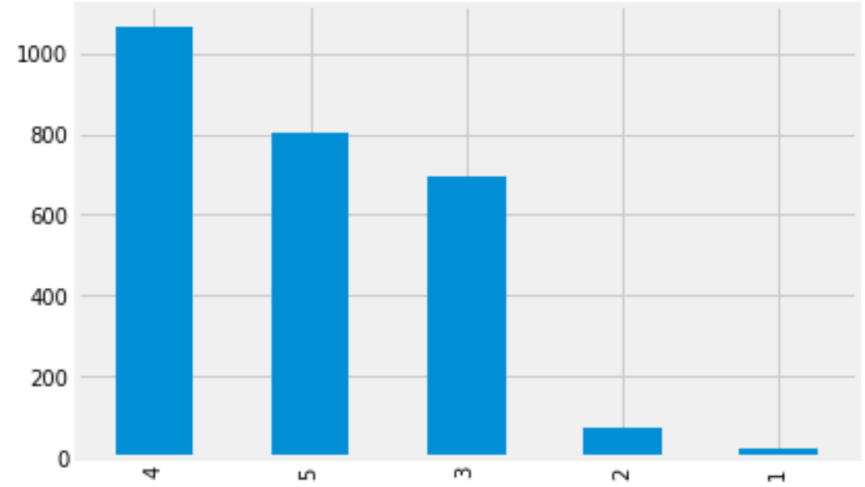
```
(99.5, 149.25]     428
(49.75, 99.5]      191
(149.25, 199.0]    143
(-0.199, 49.75]      6
Name: Glucose, dtype: int64
```

Bar Plot and Histogram

- If a feature is categorical, a bar plot is preferred
  - The height of the bar the indicates the frequency (count) of the feature.
  - This is called a bar graph
- If a feature is numeric, a histogram is preferred.
  - The range of values for a feature is partitioned into disjoin consecutive subranges.
  - The subrange referred as buckets or bins are disjoint subset of the data consecutive for a feature
  - The range of a bucket is known a width
  - This is called a histogram

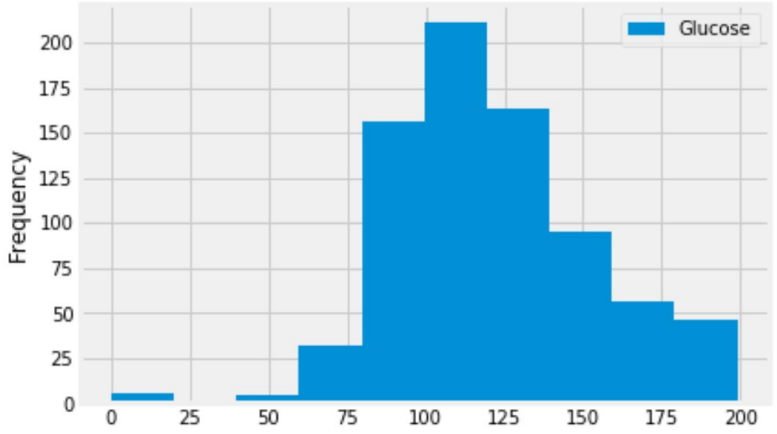
```
1 art_ratings.value_counts().plot(kind='bar')
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x1fa70b29208>
```




```
1 # get a histogram of the Glucose column for diabetes class
2 data.plot('Outcome', 'Glucose', kind = 'hist')
```

```
9]: <matplotlib.axes._subplots.AxesSubplot at 0x1da8b1f8c48>
```



Normalization

- Normalizing the data attempts to give all attributes an equal weight
  - The measurement unit used can affect the modeling and data analysis
  - To avoid dependence on the choice of measurement units, the data should be *normalized* or *standardized*
    - This involves transforming the data to fall within a smaller or common range such as [-1, 1] or [0.0, 1.0].
  - Normalization is particularly useful for classification

```
1 data_imputed_mean = pd.DataFrame(data_imputed, columns=data_column_names)
```

```
1 data_imputed_mean.head()
```

:

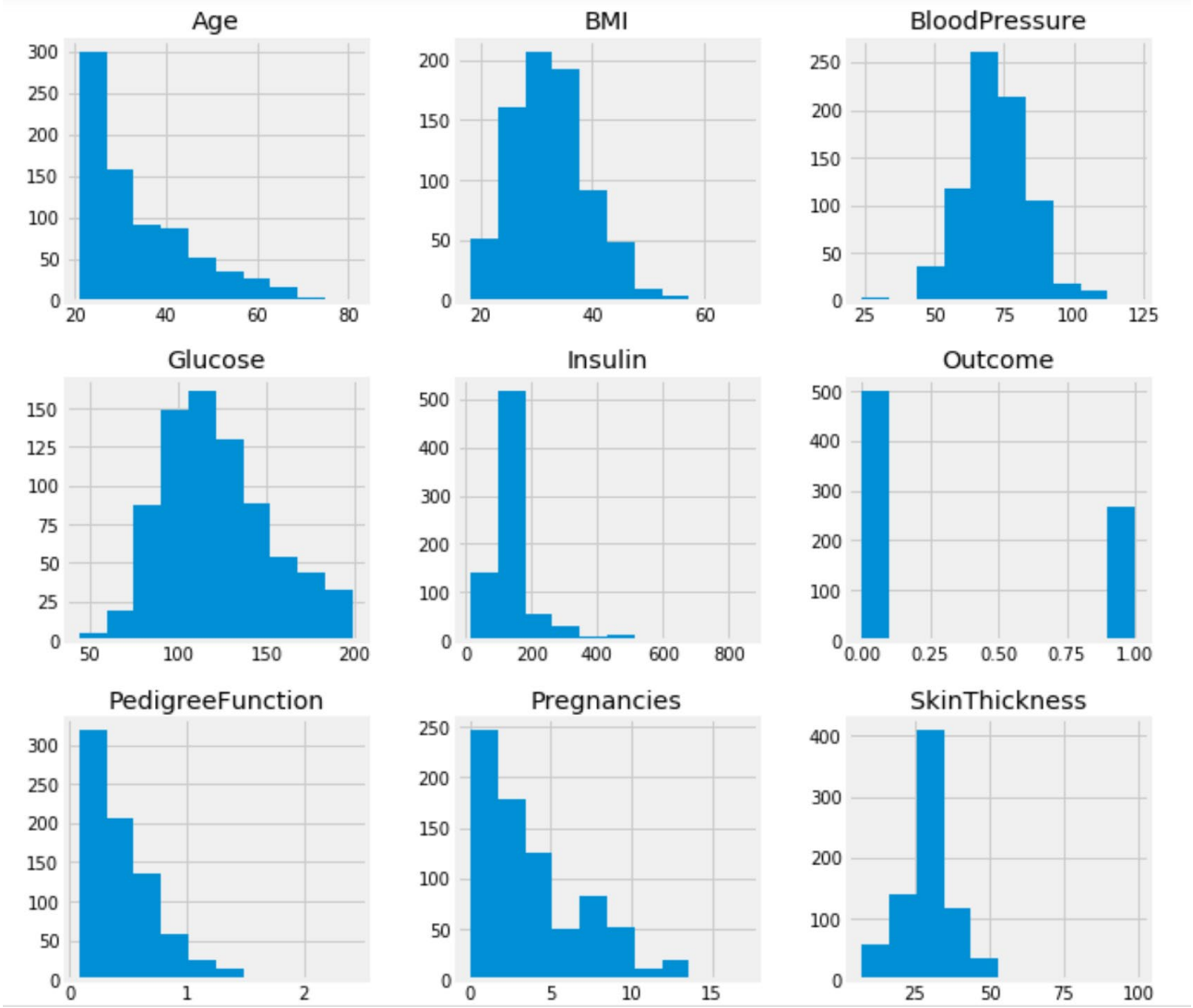
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
0	6.0	148.0	72.0	35.00000	155.548223	33.6	0.627	50.0	1.0
1	1.0	85.0	66.0	29.00000	155.548223	26.6	0.351	31.0	0.0
2	8.0	183.0	64.0	29.15342	155.548223	23.3	0.672	32.0	1.0
3	1.0	89.0	66.0	23.00000	94.000000	28.1	0.167	21.0	0.0
4	0.0	137.0	40.0	35.00000	168.000000	43.1	2.288	33.0	1.0

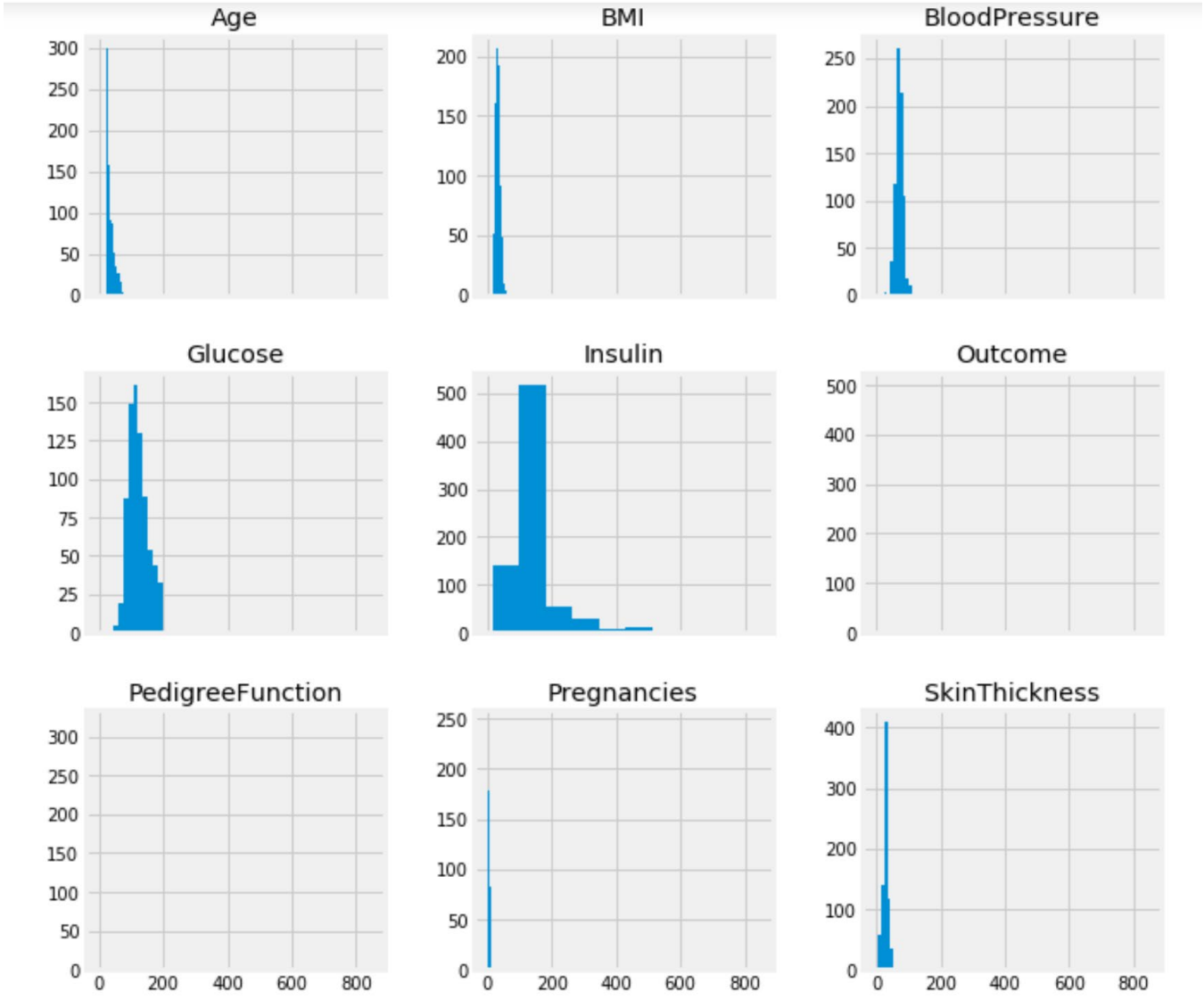
```
1 data_imputed_mean.describe()
```

:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
1 data_imputed_mean.hist(figsize=(10,10))
```


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness



It is clear that each column uses a different scale weight.
Normalization ensures that all values are treated equally for the model.

Z-score Normalization

- The process of converting a raw data into a z-score (standard) score is called z—score **standardizing** or **normalizing**

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - v: the value of a feature A,
  - v': the normalized value,
  - μ_A: the mean of the feature A,
  - σ_A: the standard deviation of the feature A

  - Ex. Let μ_A = 54,000, σ_A = 16,000. Then the normalized value of 73,600 is

Z-score

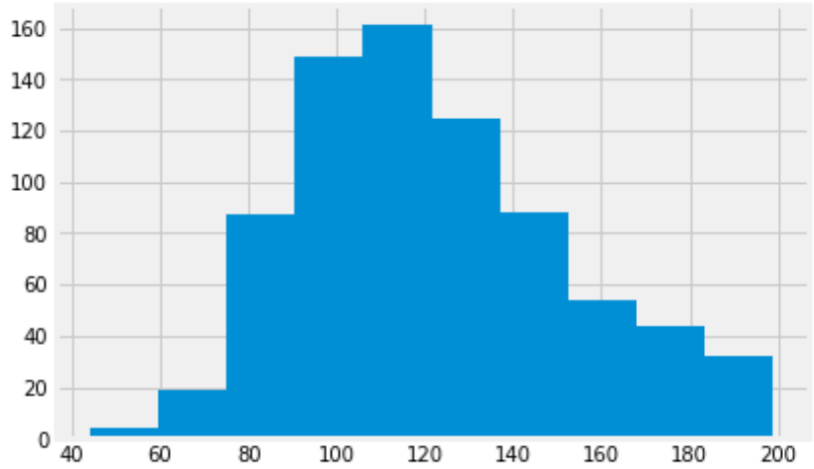
- A **Z-score** is a numerical measurement used in statistics of a **value's** relationship to the mean (average) of a group of values, measured in terms of standard deviations from the mean.
  - If a Z-score is 0, it indicates that the data point is identical to the mean
  - If a Z-scores is positive, it indicates that the score is above the mean
    - A Z-score of 1.0 would indicate a value that is one standard deviation from the mean
  - If a Z-scores is negative, it indicates that the score is below the mean
- $z = \frac{x-\mu}{\sigma}$

```
1  data['Glucose'].head()
```

```
35]: 0    148.0
     1     85.0
     2    183.0
     3     89.0
     4    137.0
     Name: Glucose, dtype: float64
```

```
1  data['Glucose'].hist()
```

```
34]: <matplotlib.axes._subplots.AxesSubplot at 0x176daf5ba88>
```




```
1  mean = data['Glucose'].mean()
```

```
1  std = data['Glucose'].std()
```

```
1  data['Glucose'].mean(), data['Glucose'].std()
```

```
38]: (121.6867627785059, 30.53564107280403)
```

```
1 z_score = (data['Glucose']-mean)/std
```

```
1 z_score.head()
```

```
0    0.861722
1   -1.201441
2    2.007924
3   -1.070446
4    0.501487
Name: Glucose, dtype: float64
```

```
1 z_score.mean(), z_score.std()
```

```
(1.0025473970730444e-16, 1.000000000000001)
```

```
1 print("%.3f" % z_score.mean())
```

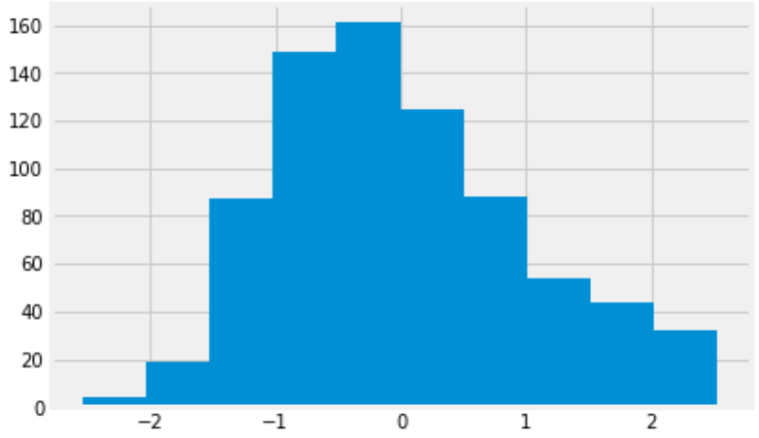
```
0.000
```

```
1 print("%.3f" % z_score.std())
```

```
1.000
```

```
1 z_score.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x176daffe1c8>
```



```
1  # built-in z-score normalizer
```

```
1  from sklearn.preprocessing import StandardScaler
```

```
1  scaler = StandardScaler ()
```

```
1  z_score_glucose = scaler.fit_transform(data[['Glucose']])
```

```
1  type (z_score_glucose)
```

8]: numpy.ndarray

```
1  z_score_glucose_df = pd.DataFrame(z_score_glucose, columns=['Glucose'])
```

```
1  z_score_glucose_df.head()
```

6]:

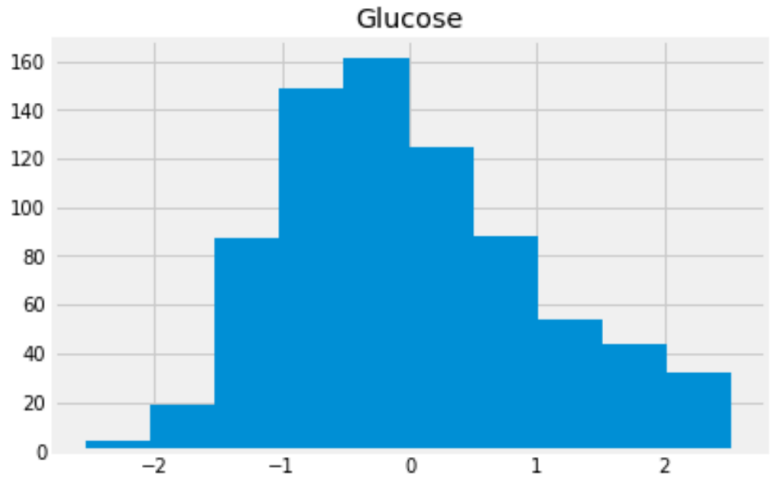
	Glucose
0	0.862287
1	-1.202229
2	2.009241
3	-1.071148
4	0.501816

```
1 z_score_glucose_df.mean(), z_score_glucose_df.std()
```

```
: (Glucose    1.316844e-16
   dtype: float64, Glucose    1.000656
   dtype: float64)
```

```
1 z_score_glucose_df.hist()
```

```
: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001C568558588>]],
      dtype=object)
```



```
1 data_imputed_mean.describe()
```

:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
column_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
                'DiabetesPedigreeFunction', 'Age', 'Outcome']
```

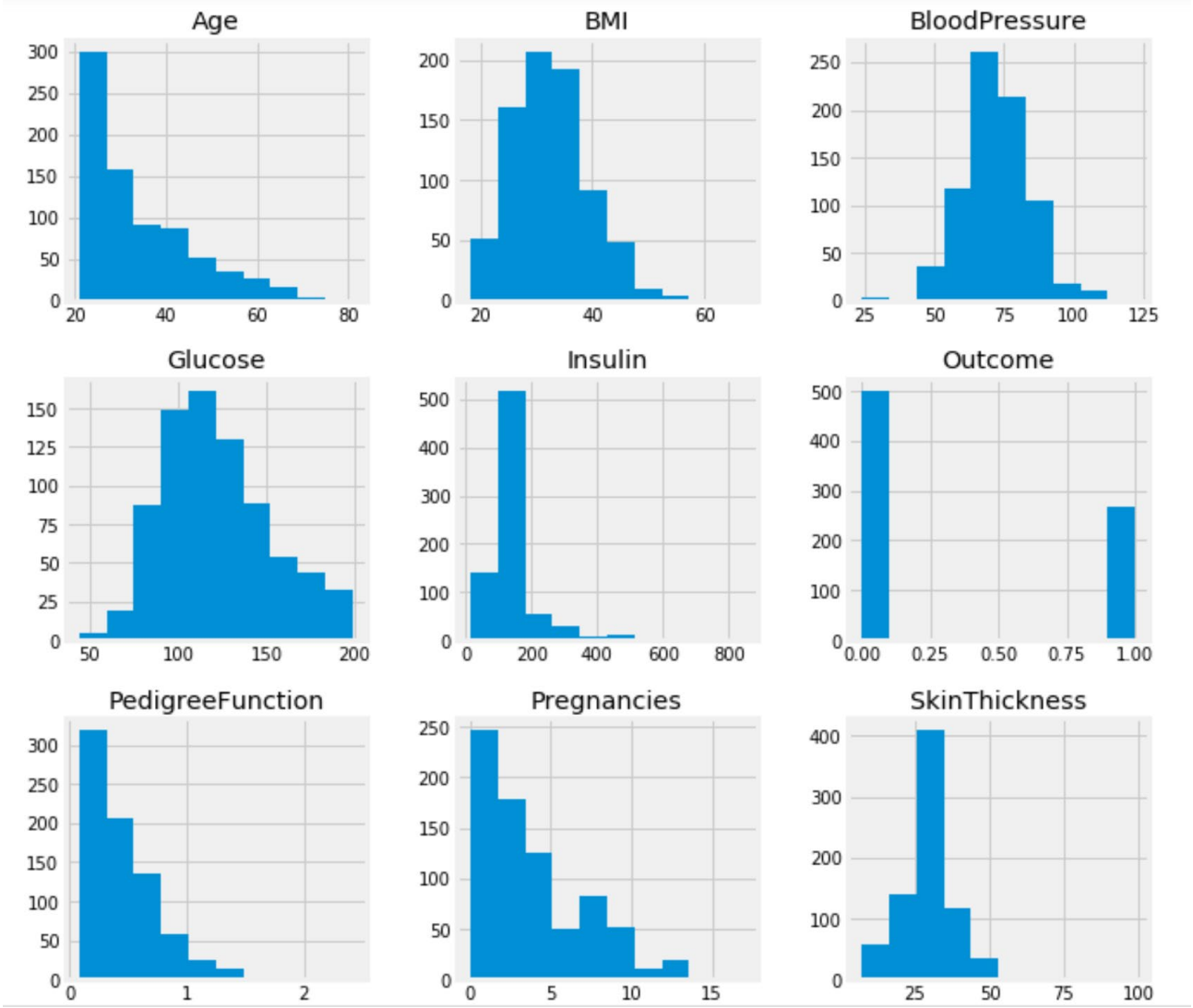
```
data_imputed_mean_z_score_scaled = pd.DataFrame(scaler.fit_transform(data_imputed_mean), columns = data_column_names)
```

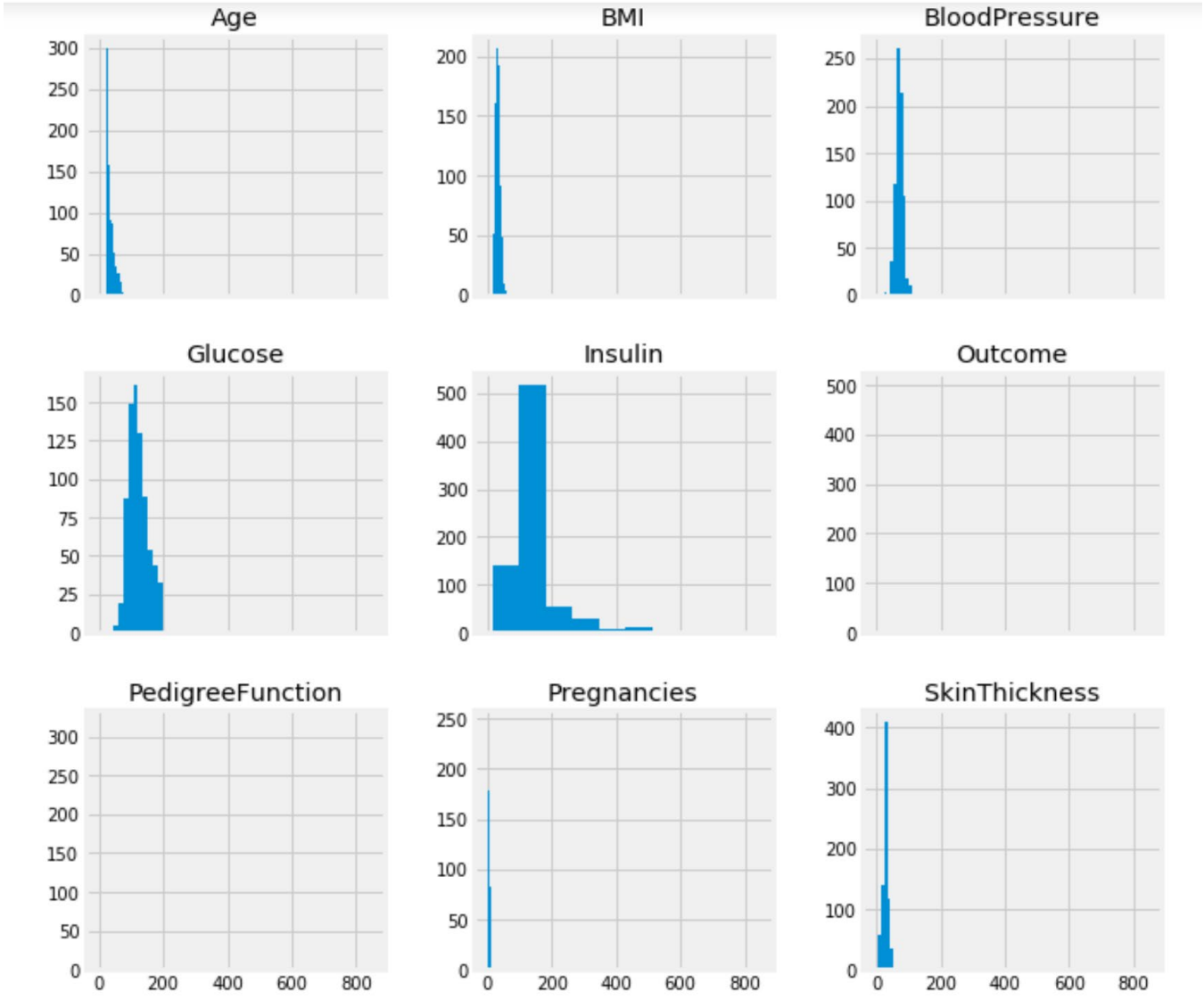
```
data_imputed_mean_z_score_scaled.describe()
```

```
1 data_imputed_mean_z_score_scaled.describe()
```

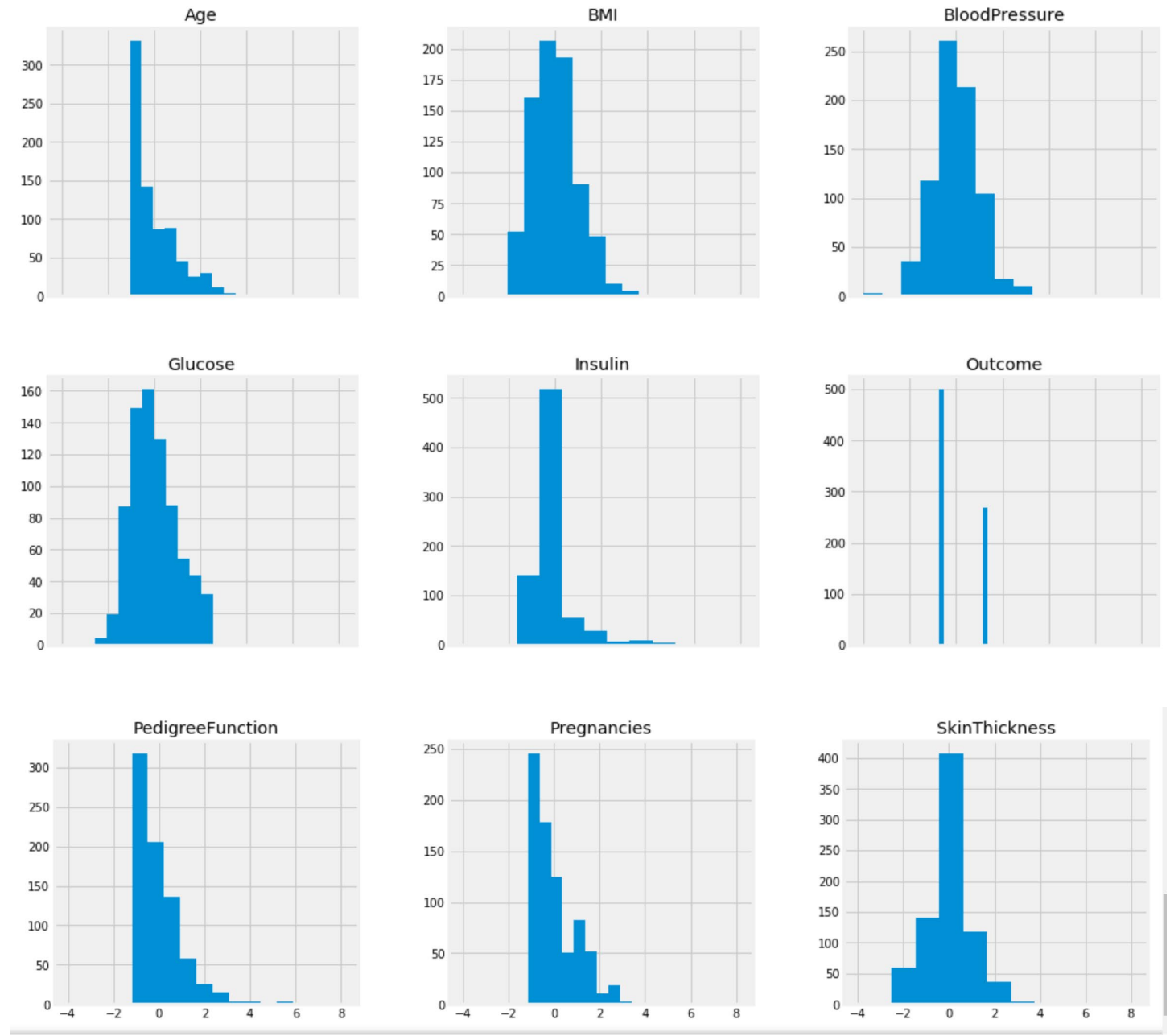
:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02
mean	2.544261e-17	-3.301757e-16	6.966722e-16	6.866252e-16	-2.352033e-16	3.553292e-16	2.398978e-16	1.857600e-16	2.408374e-16
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00
min	-1.141852e+00	-2.554131e+00	-4.004245e+00	-2.521670e+00	-1.665945e+00	-2.075119e+00	-1.189553e+00	-1.041549e+00	-7.321202e-01
25%	-8.448851e-01	-7.212214e-01	-6.953060e-01	-4.727737e-01	-4.007289e-01	-7.215397e-01	-6.889685e-01	-7.862862e-01	-7.321202e-01
50%	-2.509521e-01	-1.540881e-01	-1.675912e-02	8.087936e-16	-3.345079e-16	-8.363615e-03	-3.001282e-01	-3.608474e-01	-7.321202e-01
75%	6.399473e-01	6.103090e-01	6.282695e-01	3.240194e-01	-3.345079e-16	6.029301e-01	4.662269e-01	6.602056e-01	1.365896e+00
max	3.906578e+00	2.541850e+00	4.102655e+00	7.950467e+00	8.126238e+00	5.042087e+00	5.883565e+00	4.063716e+00	1.365896e+00


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness



It is clear that each column uses a different scale weight.
Normalization ensures that all values are treated equally for the model.


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness

Min-Max Normalization

- Performs a linear transformation on the raw data based on the minimum and maximum value.
  - Suppose that min_A and max_A are the minimum and maximum values of a feature, *A*.
  - Min-max normalization maps a value, v_i , of *A* to v_i' in the range [*new* min_A, *new* max_A] by computing

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to ?

```
# import the sklearn module
from sklearn.preprocessing import MinMaxScaler
```

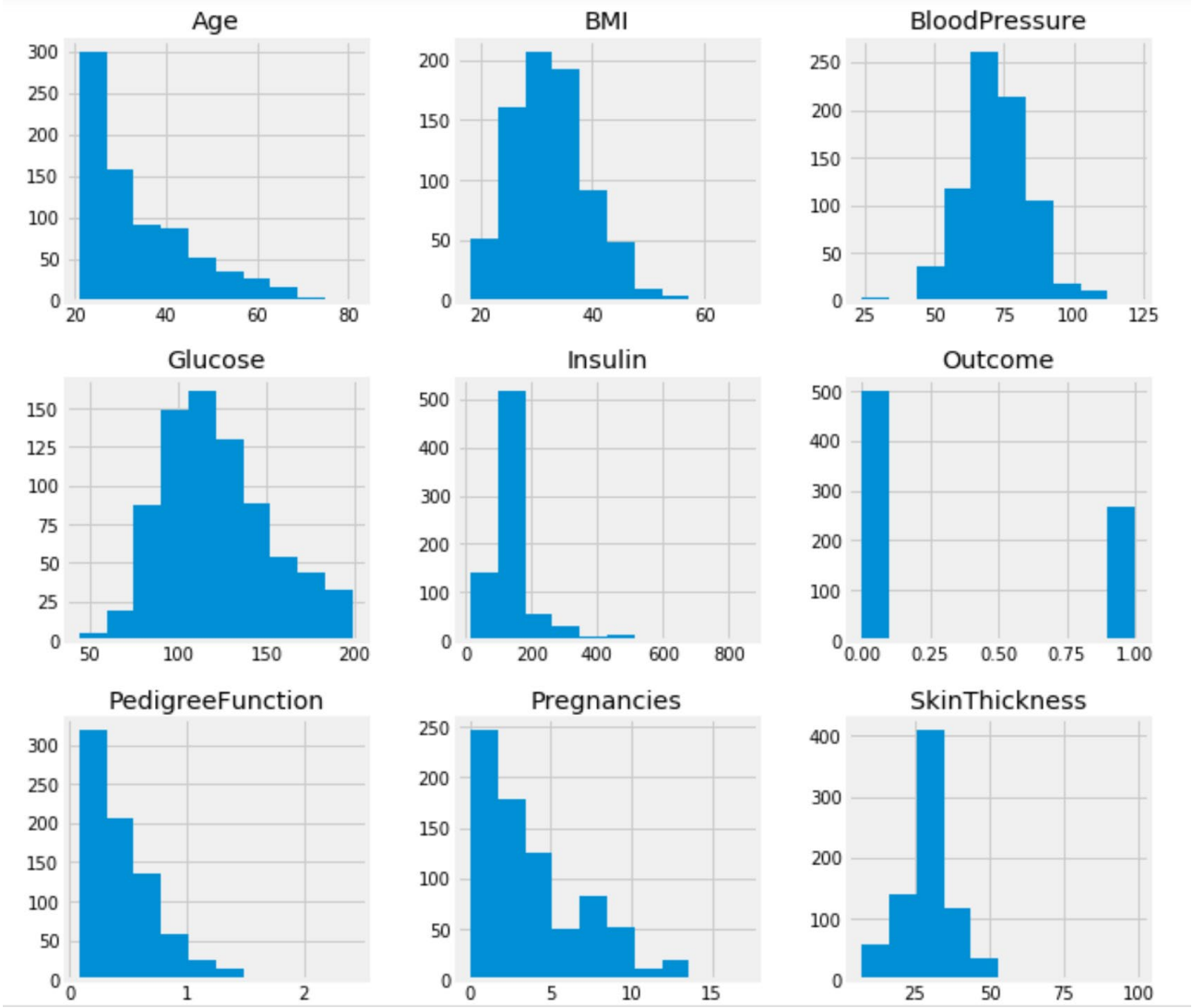
```
# instantiate the class
min_max = MinMaxScaler()

# apply the min_min normalization (scaling)
data_imputed_minmax_scaled = pd.DataFrame(min_max.fit_transform(data_imputed), columns=data_column_names)
```

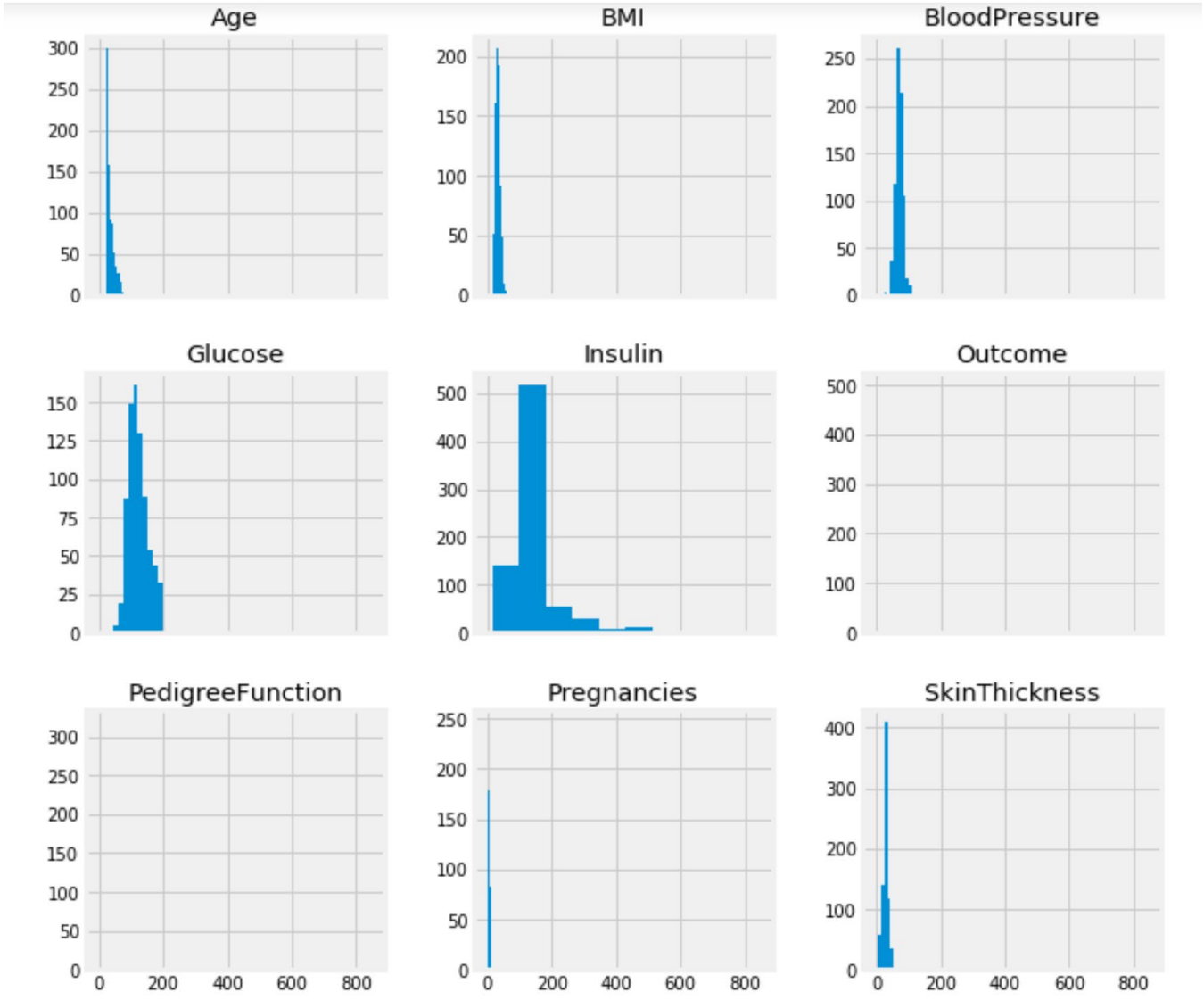
```
data_imputed_minmax_scaled.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	0.226180	0.501205	0.493930	0.240798	0.170130	0.291564	0.168179	0.204015	0.348958
std	0.198210	0.196361	0.123432	0.095554	0.102189	0.140596	0.141473	0.196004	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.058824	0.359677	0.408163	0.195652	0.129207	0.190184	0.070773	0.050000	0.000000
50%	0.176471	0.470968	0.491863	0.240798	0.170130	0.290389	0.125747	0.133333	0.000000
75%	0.352941	0.620968	0.571429	0.271739	0.170130	0.376278	0.234095	0.333333	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

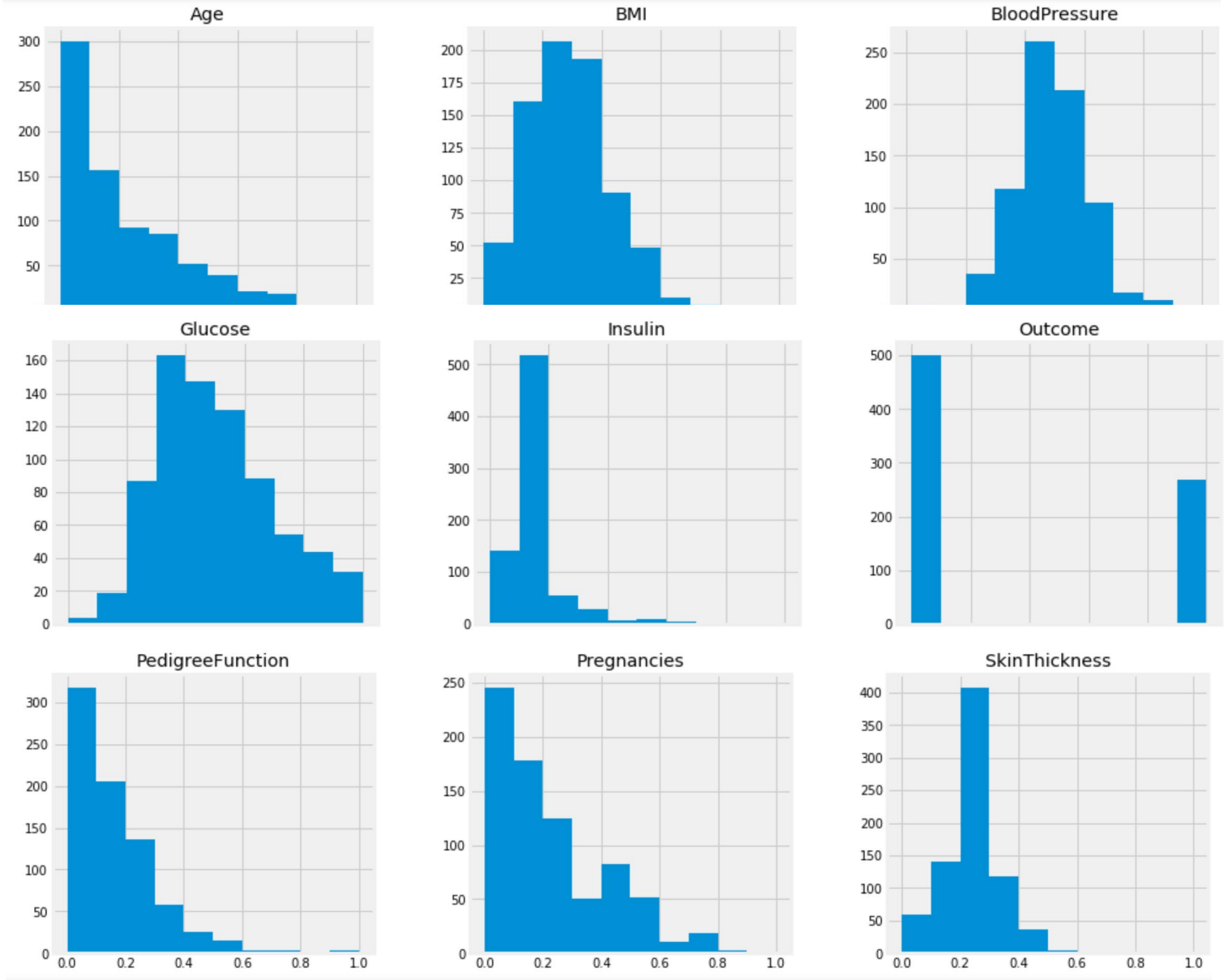
```
data_imputed_mean.describe()
```

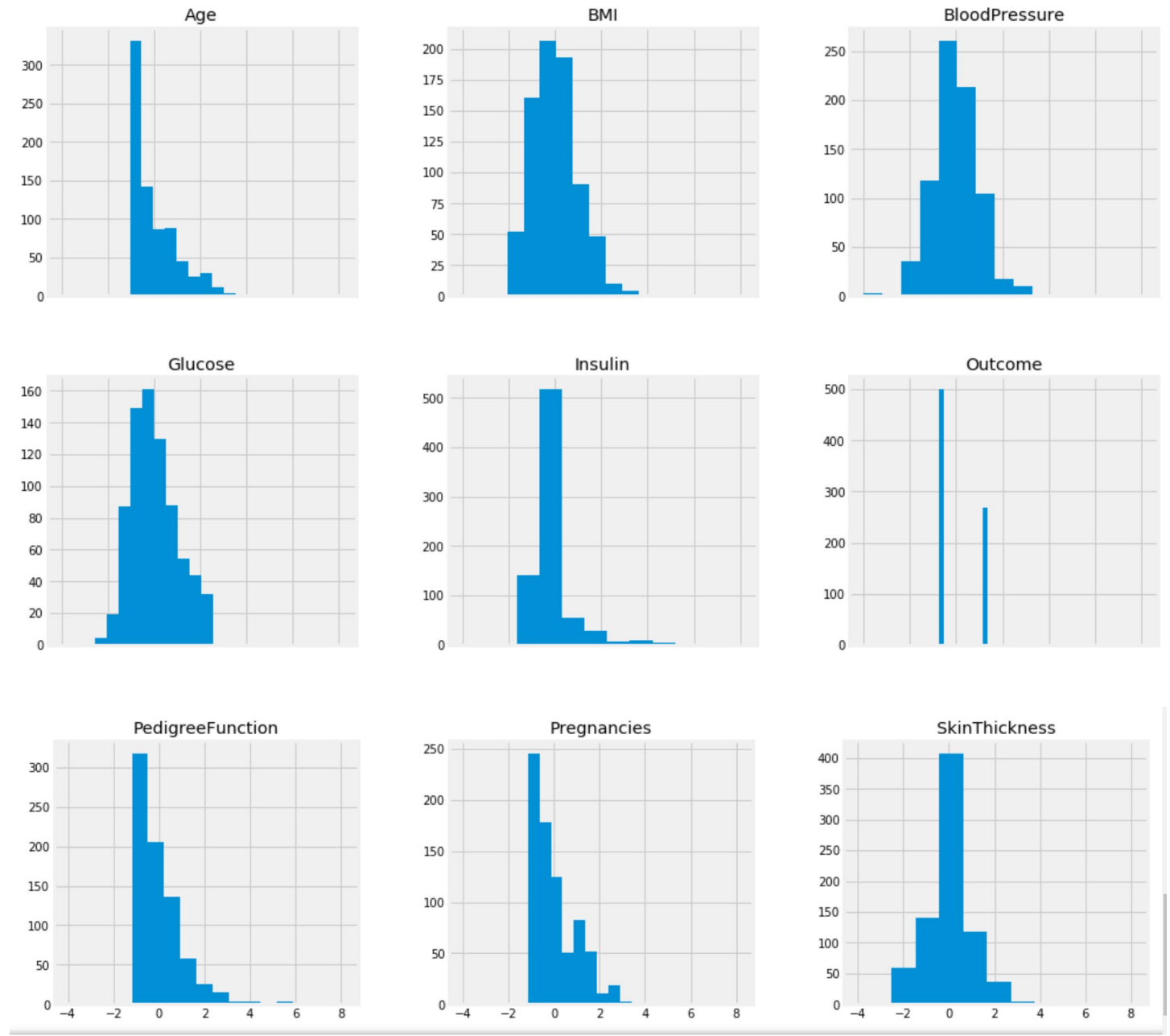
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness



It is clear that each column uses a different scale weight.
Normalization ensures that all values are treated equally for the model.


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness


Age
BMI
BloodPressure
Glucose
Insulin
Outcome
PedigreeFunction
Pregnancies
SkinThickness

Model Accuracy Before and After Dealing with Missing values and Normalizing the Dataset

	# of rows a model learned from	Cross-validation accuracy
Drop missing-valued rows	391	.7449
Impute values with 0	768	.7304
Impute values with mean of column	768	.7318
Impute values with median of column	768	.7357
Z-score normalization with mean imputing	768	.7422
Min-max normalization with mean imputing	**768**	**.7461**