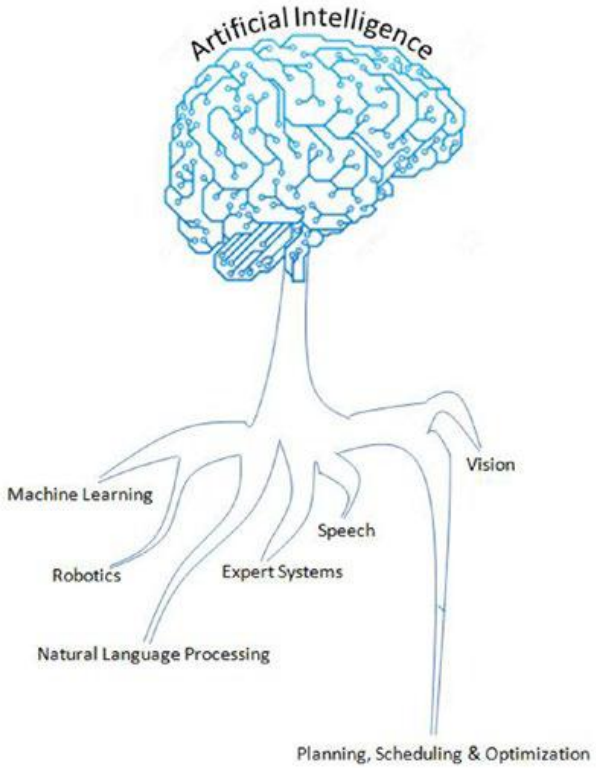
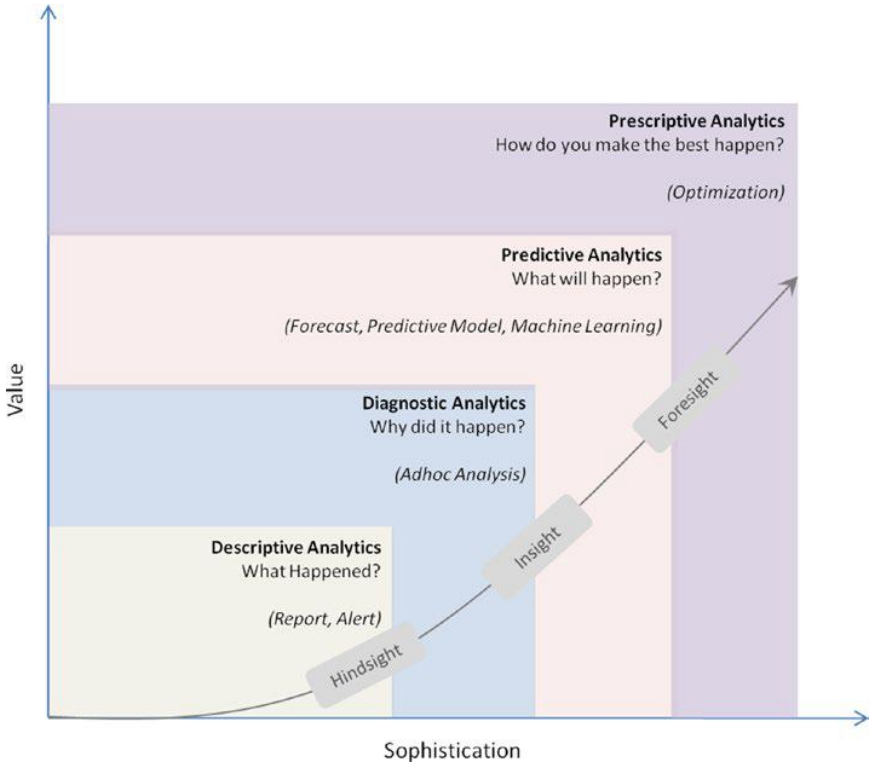
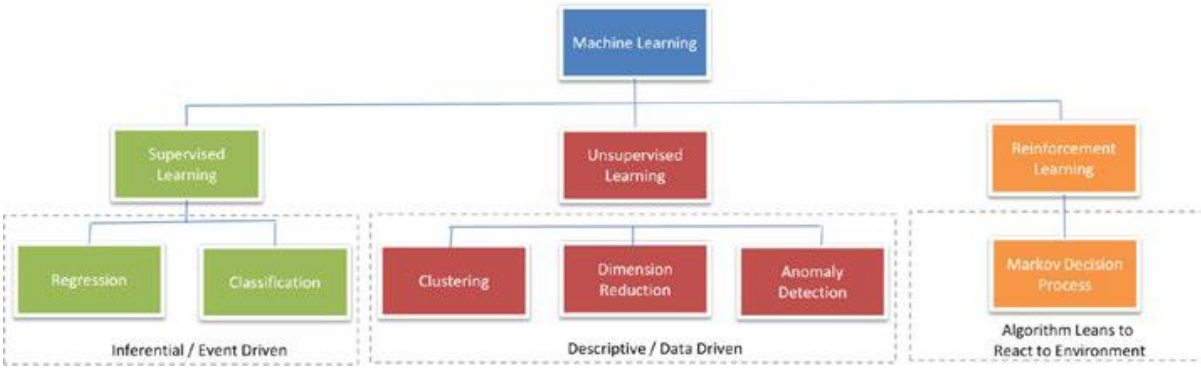
Python Machine Learning



Statistics vs Data Mining vs Data Analytics vs Machine Learning



Types of Machine Learning

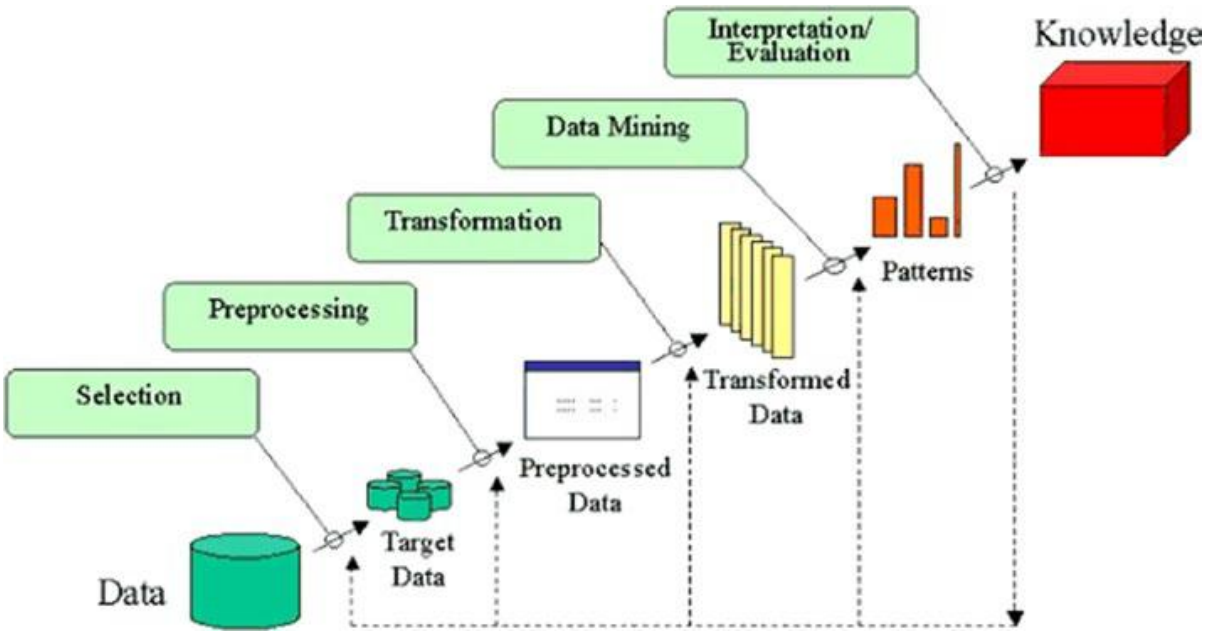


Building Blocks of Machine Learning

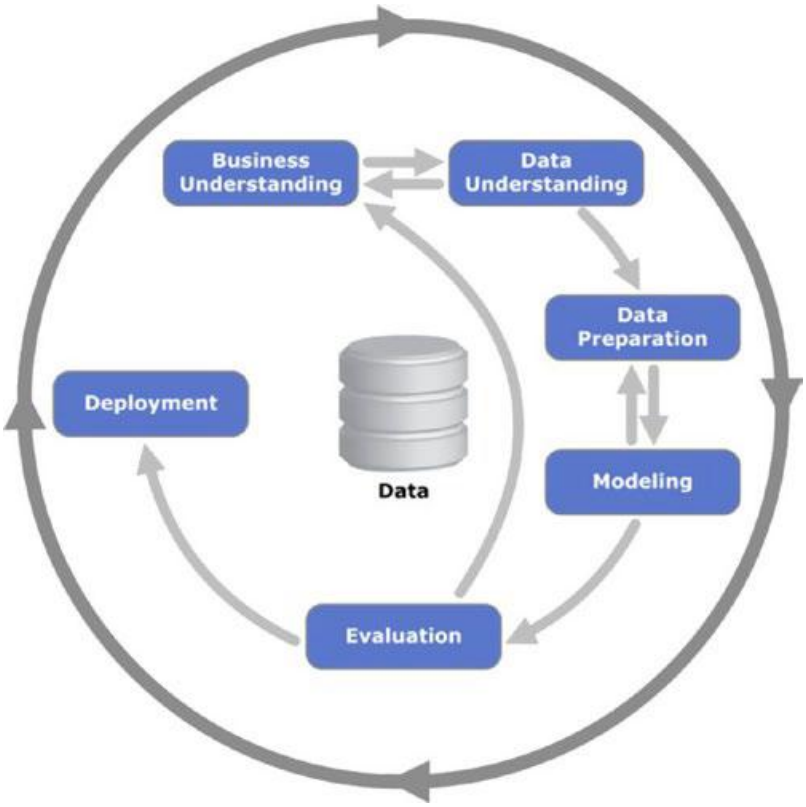
1. Knowledge Discovery Process

2. Cross Industrial Standard Process for Data Mining [CRISP - DM]

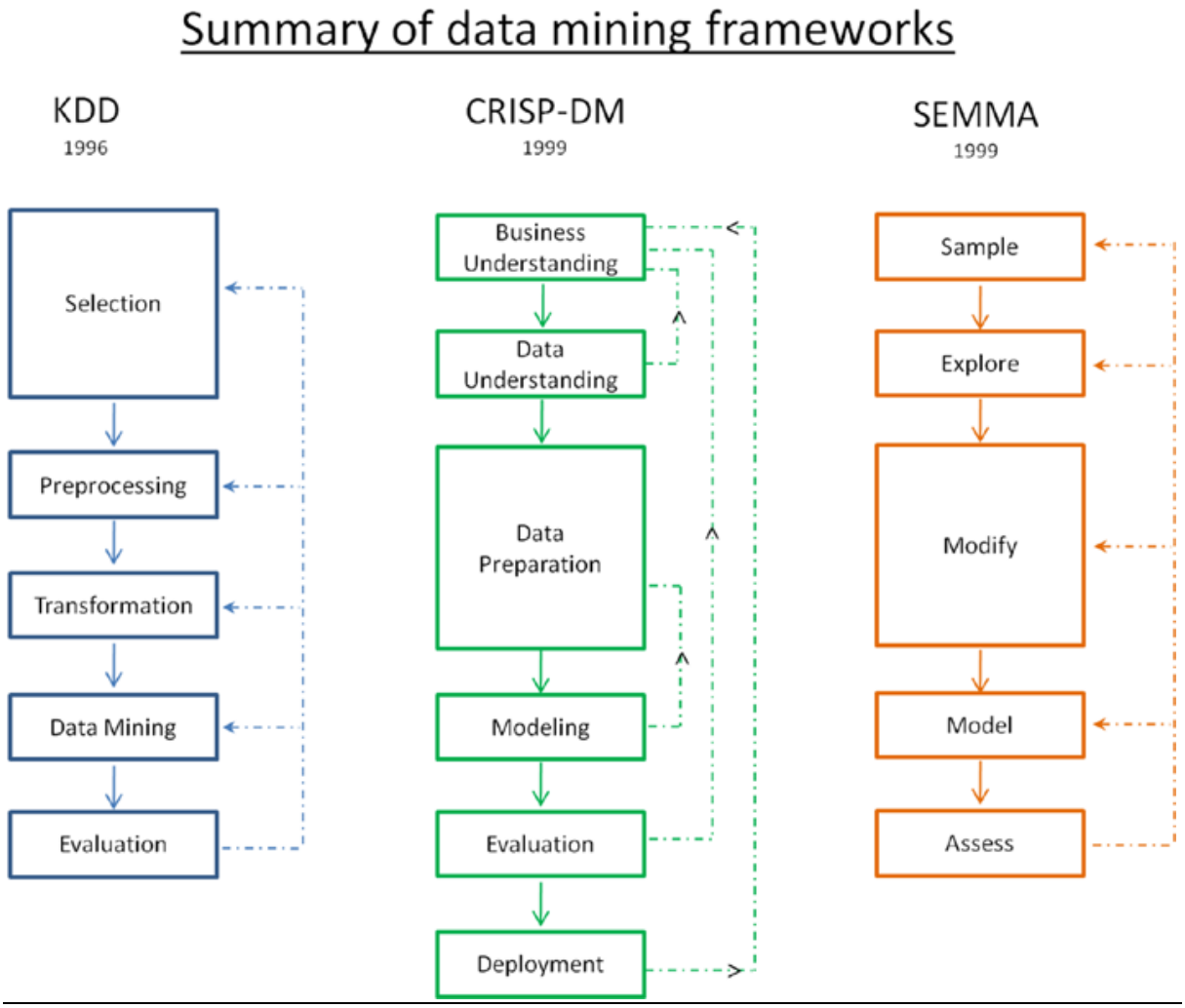
3. Sample, Explore, Modify, Model and Assess [SEMMA]



Knowledge Discovery Process



Cross Industrial Standard Process for Data mining



Summary of Data Mining Frameworks

Data Analysis Packages

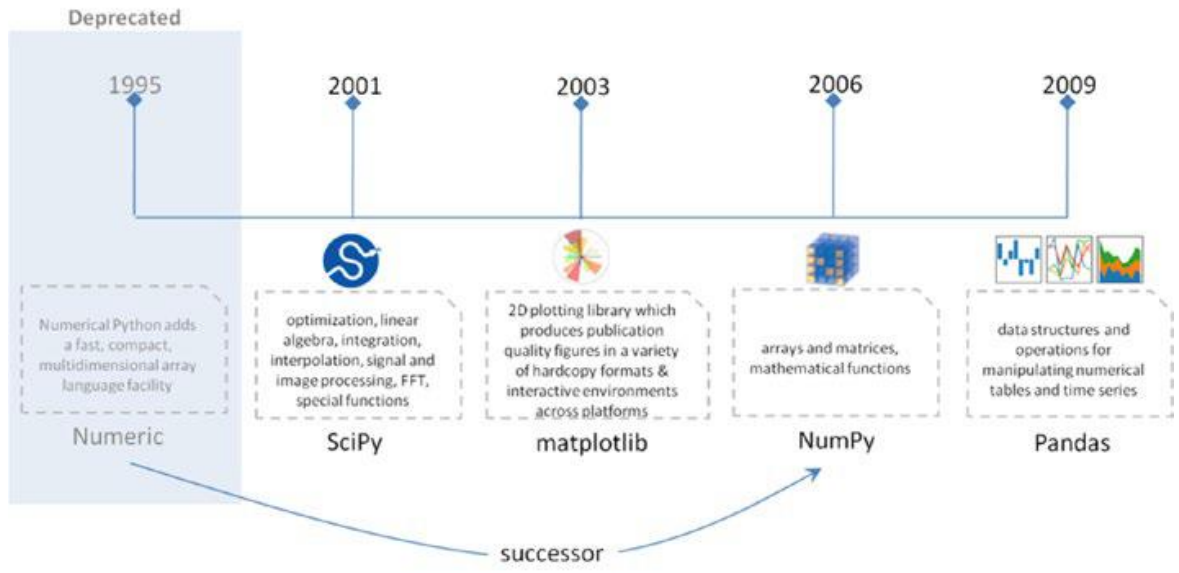


Table 1-2. *Python object types*

Type	Examples	Comments
None	None	# singleton null object
Boolean	True, False	
Integer	-1, 0, 1, sys.maxint	
Long	1L, 9787L	
Float	3.141592654	
	inf, float('inf')	# infinity
	-inf	# neg infinity
	nan, float('nan')	# not a number
Complex	2+8j	# note use of j
String	'this is a string', "also me"	# use single or double quote
	r'raw string', b'ASCII string'	
	u'unicode string'	
Tuple	empty = ()	# empty tuple
	(1, True, 'ML')	# immutable list or unalterable list
List	empty = []	empty list
	[1, True, 'ML']	# mutable list or alterable list
Set	empty = set()	# empty set
	set(1, True, 'ML')	# mutable or alterable
dictionary	empty = {} {'1':'A', '2':'AA', True = 1, False = 0}	# mutable object or alterable object
File	f = open('filename', 'rb'),	

When to Use List vs. Tuples vs. Set vs. Dictionary

- *List*: Use when you need an ordered sequence of homogenous collections, whose values can be changed later in the program.
- *Tuple*: Use when you need an ordered sequence of heterogeneous collections whose values need not be changed later in the program.
- *Set*: It is ideal for use when you don't have to store duplicates and you are not concerned about the order or the items. You just want to know whether a particular value already exists or not.
- *Dictionary*: It is ideal for use when you need to relate values with keys, in order to look them up efficiently using a key.

Arithmetic Operators

Operator	Description	Example
+	Addition	x + y = 30
-	Subtraction	x – y = -10
*	Multiplication	x * y = 200
/	Division	y / x = 2
%	Modulus	y % x = 0
** Exponent	Exponentiation	x**b =10 to the power 20
//	Floor Division – Integer division rounded toward minus infinity	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0/

Relational or Comparison Operators

Operator	Description	Example
==	The condition becomes True, if the values of two operands are equal.	(x == y) is not true.
!=	The condition becomes True, if the values of two operands are not equal.	
<>	The condition becomes True, if values of two operands are not equal.	(x<>y) is true. This is similar to != operator.
>	The condition becomes True, if the value of left operand is greater than the value of right operand.	(x>y) is not true.
<	The condition becomes True, if the value of left operand is less than the value of right operand.	(x<y) is true.
>=	The condition becomes True, if the value of left operand is greater than or equal to the value of right operand.	(x>= y) is not true.
<=	The condition becomes True, if the value of left operand is less than or equal to the value of right operand.	(x<= y) is true.

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand.	z = x + y assigns value of x + y into z
+= Add AND	It adds right operand to the left operand and assigns the result to left operand.	z += x is equivalent to z = z + x
-= Subtract AND	It subtracts right operand from the left operand and assigns the result to left operand.	z -= x is equivalent to z = z - x
*= Multiply AND	It multiplies right operand with the left operand and assigns the result to left operand.	z *= x is equivalent to z = z * x
/= Divide AND	It divides left operand with the right operand and assigns the result to left operand.	z /= x is equivalent to z = z/ xz /= x is equivalent to z = z / x
%= Modulus AND	It takes modulus using two operands and assigns the result to left operand.	z %= x is equivalent to z = z % x
**= Exponent AND	Performs exponential (power) calculation on operators and assigns value to the left operand.	z **= x is equivalent to z = z ** x
//= Floor Division	It performs floor division on operators and assigns value to the left operand.	z //= x is equivalent to z = z// x

Logical Operators

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(var1 and var2) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(var1 or var2) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not (var1 and var2) is false.

Python List Operations

Description	Python Expression	Example	Results
Creating a list of items	[item1, item2, ...]	list = ['a','b','c','d']	['a','b','c','d']
Accessing items in list	list[index]	list = ['a','b','c','d'] list[2]	c
Length	len(list)	len([1, 2, 3])	3
Concatenation	list_1 + list_2	[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]
Repetition	list * int	['Hello'] * 3	['Hello', 'Hello', 'Hello']
Membership	item in list	3 in [1, 2, 3]	TRUE
Iteration	for x in list: print x	for x in [1, 2, 3]: print x,	1 2 3
Count from the right	list[-index]	list = [1,2,3]; list[-2]	2
Slicing fetches sections	list[index:]	list = [1,2,3]; list[1:]	[2,3]
Comparing lists	cmp(list_1, list_2)	print cmp([1,2,3,4], [5,6,7]); print cmp([1,2,3], [5,6,7,8])	1 -1
Return max item	max(list)	max([1,2,3,4,5])	5
Return min item	min(list)	max([1,2,3,4,5])	1
Append object to list	list.append(obj)	[1,2,3,4].append(5)	[1,2,3,4,5]
Count item occurrence	list.count(obj)	[1,1,2,3,4].count(1)	2
Append content of sequence to list	list.extend(seq)	['a', 1].extend(['b', 2])	['a', 1, 'b', 2]
Return the first index position of item	list.index(obj)	['a', 'b','c',1,2,3]. index('c')	2
Insert object to list at a desired index	list.insert(index, obj)	['a', 'b','c',1,2,3]. insert(4, 'd')	['a', 'b','c','d', 1,2,3]
Remove and return last object from list	list.pop(obj=list[-1])	['a', 'b','c',1,2,3].pop() ['a', 'b','c',1,2,3].pop(2)	3 c
Remove object from list	list.remove(obj)	['a', 'b','c',1,2,3]. remove('c')	['a', 'b', 1,2,3]
Reverse objects of list in place	list.reverse()	['a', 'b','c',1,2,3]. reverse()	[3,2,1,'c','b','a']
Sort objects of list	list.sort()	['a', 'b','c',1,2,3].sort() ['a', 'b','c',1,2,3]. sort(reverse = True)	[1,2,3,'a', 'b','c'] ['c','b','a',3,2,1]

Tuple Operations

Description	Python Expression	Example	Results
Creating a tuple	(item1, item2, ...) () # empty tuple (item1,) # Tuple with one item, note comma is required	tuple = ('a','b','c', 'd',1,2,3) tuple = () tuple = (1,)	('a','b','c','d',1,2,3) () 1
Accessing items in tuple	tuple[index] tuple[start_index: end_index]	tuple = ('a','b','c', 'd',1,2,3) tuple[2] tuple[0:2]	c a, b, c
Deleting a tuple	del tuple_name	del tuple	
Length	len(tuple)	len((1, 2, 3))	3
Concatenation	tuple_1 + tuple_2	(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)
Repetition	tuple * int	('Hello',) * 4	('Hello', 'Hello', 'Hello', 'Hello')
Membership	item in tuple	3 in (1, 2, 3)	TRUE
Iteration	for x in tuple: print x	for x in (1, 2, 3): print x	1 2 3
Count from the right	tuple[-index]	tuple = (1,2,3); list[-2]	2
Slicing fetches sections	tuple[index:]	tuple = (1,2,3); list[1:]	(2,3)
Comparing lists	cmp(tuple_1, tuple_2)	print cmp((1,2,3,4), (5,6,7)); print cmp((1,2,3), (5,6,7,8))	1 -1
Return max item	max(tuple)	max((1,2,3,4,5))	5
Return min item	min(tuple)	max((1,2,3,4,5))	1
Convert a list to tuple	tuple(seq)	tuple([1,2,3,4])	(1,2,3,4,5)

Set Operations

Description	Python Expression	Example	Results
Creating a set.	set{item1, item2, ...} set() # empty set	languages = set(['Python', 'R', 'SAS', 'Julia'])	set(['SAS', 'Python', 'R', 'Julia'])
Add an item/ element to a set.	add()	languages. add('SPSS')	set(['SAS', 'SPSS', 'Python', 'R', 'Julia'])
Remove all items/ elements from a set.	clear()	languages.clear()	set([])
Return a copy of a set.	copy()	lang = languages. copy() print lang	set(['SAS', 'SPSS', 'Python', 'R', 'Julia'])
Remove an item/ element from set if it is a member. (Do nothing if the element is not in set).	discard()	languages = set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R', 'SAS', 'Machine Learning']) languages. discard('AI')	set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Learning'])
Remove an item/ element from a set. If the element is not a member, raise a KeyError.	remove()	languages = set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R', 'SAS', 'Machine Learning']) languages. remove('AI')	set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Learning'])
Remove and return an arbitrary set element. RaiseKeyError if the set is empty.	pop()	languages = set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R', 'SAS', 'Machine Learning']) print "Removed:", (languages.pop()) print(languages)	Removed: C set(['Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Learning'])

Description	Python Expression	Example	Results

Return the difference of two or more sets as a new set.	difference()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.difference(B)
			{1, 2, 3}

Remove all item/ elements of another set from this set.	difference_update()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.difference_ update(B) print A
			set([1, 2, 3])

Return the intersection of two sets as a new set.	intersection()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.intersection(B)
			{4, 5}

Update the set with the intersection of itself and another.	intersection_ update()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.intersection_ update B print A
			set([4, 5])

Return True if two sets have a null intersection.	isdisjoint()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.isdisjoint(B)
			FALSE

Return True if another set contains this set.	issubset()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8} print
		A.issubset(B)
			FALSE

Return True if this set contains another set.	issuperset()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8} print
		A.issuperset(B)
			FALSE

Return the symmetric difference of two sets as a new set.	symmetric_ difference()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A. symmetric_ difference(B)
			{1, 2, 3, 6, 7, 8}

Description	Python Expression	Example	Results

Update a set with the symmetric difference of itself and another.	symmetric_ difference_update()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.symmetric_ difference(B) print A
		A.symmetric_ difference_update(B) print A
			set([1, 2, 3, 6, 7, 8])

Return the union of sets in a new set.	union()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.union(B) print A
			set([1, 2, 3, 4, 5])

Update a set with the union of itself and others.	update()	# initialize A and B
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		A.update(B) print A
			set([1, 2, 3, 4, 5, 6, 7, 8])

Return the length (the number of items) in the set.	len()	A = {1, 2, 3, 4, 5} len(A)	5
Return the largest item in the set.	max()	A = {1, 2, 3, 4, 5} max(A)	1
Return the smallest item in the set.	min()	A = {1, 2, 3, 4, 5} min(A)	5
Return a new sorted list from elements in the set. Does not sort the set.	sorted()	A = {1, 2, 3, 4, 5} sorted(A)	[4, 5, 6, 7, 8]
Return the sum of all items/elements in the set.	sum()	A = {1, 2, 3, 4, 5} sum(A)	15

Dictionary Operations

Description	Python Expression	Example	Results
Creating a dictionary	dict = {'key1':'value1', 'key2':'value2'.....}	dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}	{'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Accessing items in dictionary	dict ['key']	dict['Name']	dict['Name']: Jivin
Deleting a dictionary	del dict['key']; dict.clear(); del dict;	del dict['Name']; dict.clear(); del dict;	{'Age':6, 'Class':'First'}; {};
Updating a dictionary	dict['key'] = new_value	dict['Age'] = 6.5	dict['Age']: 6.5
Length	len(dict)	len({'Name': 'Jivin', 'Age': 6, 'Class': 'First'})	3
Comparing elements of dicts	cmp(dict_1, dict_2)	dict1 = {'Name': 'Jivin', 'Age': 6}; dict2 = {'Name': 'Pratham', 'Age': 7}; dict3 = {'Name': 'Pranuth', 'Age': 7}; dict4 = {'Name': 'Jivin', 'Age': 6}; print "Return Value: ", cmp (dict1, dict2) print "Return Value: " , cmp (dict2, dict3) print "Return Value: ", cmp (dict1, dict4)	Return Value : -1 Return Value : 1 Return Value : 0

Description	Python Expression	Example	Results
String representation of dict	str(dict)	dict = {'Name': 'Jivin', 'Age': 6}; print "Equivalent String: ", str (dict)	Equivalent String : {'Age': 6, 'Name': 'Jivin'}
Return the shallow copy of dict	dict.copy()	dict = {'Name': 'Jivin', 'Age': 6}; dict1 = dict.copy() print dict1	{'Age': 6, 'Name': 'Jivin'}

Create a new dictionary with keys from seq and values set to value	dict.fromkeys()	seq = ('name', 'age', 'sex') dict = dict. fromkeys(seq) print "New Dictionary: ", str(dict) dict = dict. fromkeys(seq, 10) print "New Dictionary: ", str(dict)	New Dictionary : {'age': None, 'name': None, 'sex': None} New Dictionary : {'age': 10, 'name': 10, 'sex': 10}
For key key, returns value or default if key not in dictionary	dict.get(key, default=None)	dict = {'Name': 'Jivin', 'Age': 6} print "Value for Age: ", dict.get('Age') print "Value for Education: ", dict.get('Education', "First Grade")	Value : 6 Value : First Grade
Returns true if key in dictionary dict, false otherwise	dict.has_key(key)	dict = {'Name': 'Jivin', 'Age': 6} print "Age exists? ", dict.has_key('Age') print "Sex exists? ", dict.has_key('Sex')	Value : True Value : False
Returns a list of dict's (key, value) tuple pairs	dict.items()	dict = {'Name': 'Jivin', 'Age': 6} print "dict items: ", dict.items()	Value : [('Age', 6), ('Name', 'Jivin')]
Returns list of dictionary dict's keys	dict.keys()	dict = {'Name': 'Jivin', 'Age': 6} print "dict keys: ", dict.keys()	Value : ['Age', 'Name']

Description	Python Expression	Example	Results
Similar to get(), but will set dict[key]=default if key is not already in dict	dict.setdefault(key, default=None)	dict = {'Name': 'Jivin,' 'Age': 6} print "Value for Age: ", dict.setdefault ('Age,' None) print "Value for Sex: ", dict. setdefault('Sex,' None)	Value : 6 Value : None
Adds dictionary dict2's key-values pairs to dict	dict.update(dict2)	dict = {'Name': 'Jivin', 'Age': 6} dict2 = {'Sex': 'male' } dict.update(dict2) print "dict. update(dict2) = ", dict	Value : {'Age': 6, 'Name': 'Jivin', 'Sex': 'male'}
Returns list of dictionary dict's values	dict.values()	dict = {'Name': 'Jivin', 'Age': 6} print "Value: ", dict. values()	Value : [6, 'Jivin']

Pandas View Functions

Describe	Syntax
Looking at the top n records default n value is 5 if not specified	df.head(n=2)
Looking at the bottom n records	df.tail()
Get column names	df.columns
Get column datatypes	df.dtypes
Get dataframe index	df.index
Get unique values	df[column_name].unique()
Get values	df.values
Sort DataFrame	df.sort_values(by =['Column1', 'Column2'], ascending=[True,True'])
select/view by column name	df[column_name]
select/view by row number	df[0:3]
selection by index	df.loc[0:3] # index 0 to 3 df.loc[0:3,['column1', 'column2']] # index 0 to 3 for specific columns

selection by position	df.iloc[0:2] # using range, first 2 rows df.iloc[2,3,6] # specific position df.iloc[0:2,0:2] # first 2 rows and first 2 columns
selection without it being in the index	print df.ix[1,1] # value from fist row and first column print df.ix[:,2] # all rows of column at 2nd position
Faster alternative to iloc to get scalar values	print df.iat[1,1]
Transpose DataFrame	df.T
Filter DataFrame based on value condition for one column	df[df['column_name'] > 7.5]
Filter DataFrame based on a value condition on one column	df[df['column_name'].isin(['condition_value1', 'condition_value2'])]
Filter based on multiple conditions on multiple columns using AND operator	df[(df['column1']>7.5) & (df['column2']>3)]
Filter based on multiple conditions on multiple columns using OR operator	df[(df['column1']>7.5) | (df['column2']>3)]

Pandas Basic Operations

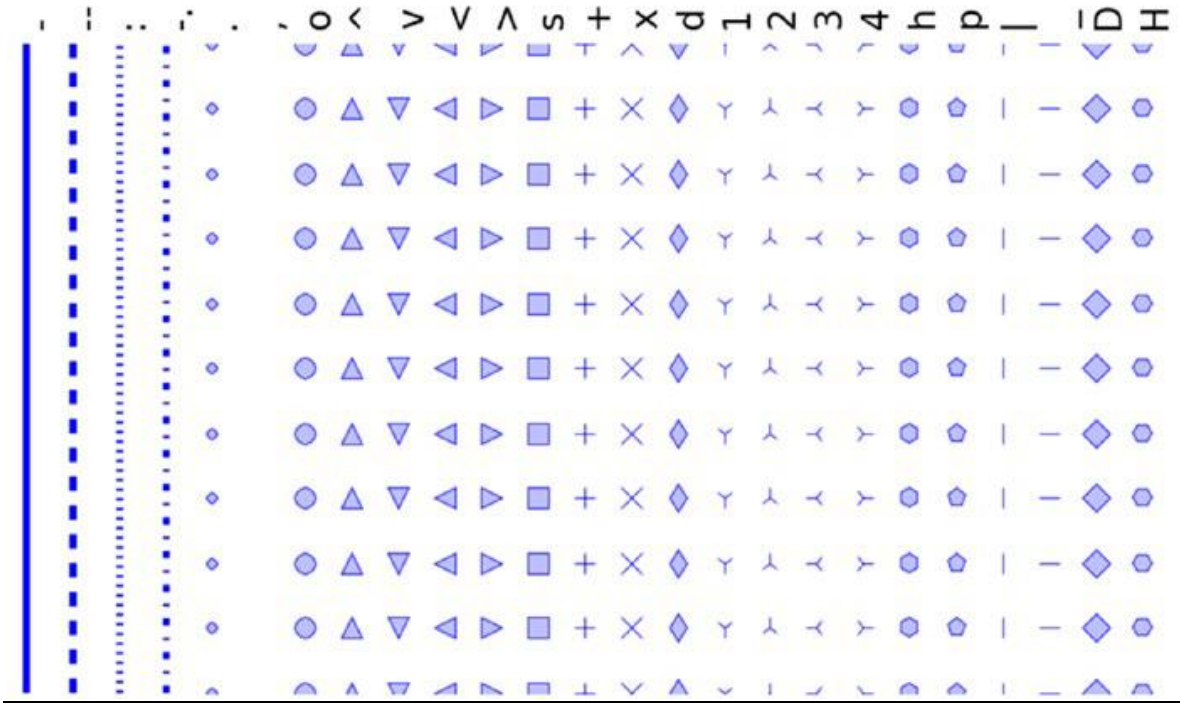
Description	Syntax
Convert string to date series	pd.to_datetime(pd. Series(['2017-04-01','2017-04-02','2017-04-03']))
Rename a specific column name	df.rename(columns={'old_columnname':'new_ columnname'}, inplace=True)
Rename all column names of DataFrame	df.columns = ['col1_new_name','col2_new_ name'....]
Flag duplicates	df.duplicated()
Drop duplicates	df = df.drop_duplicates()
Drop duplicates in specific column	df.drop_duplicates(['column_name'])
Drop duplicates in specific column, but retain the first or last observation in duplicate set	df.drop_duplicates(['column_name'], keep = 'first') # change to last for retaining last obs of duplicate
Creating new column from existing column	df['new_column_name'] = df['existing_ column_name'] + 5
Creating new column from elements of two columns	df['new_column_name'] = df['existing_ column1'] + '_' + df['existing_column2']

Adding a list or a new column to DataFrame	df['new_column_name'] = pd.Series(mylist)
Drop missing rows and columns having missing values	df.dropna()
Replaces all missing values with 0 (or you can use any int or str)	df.fillna(value=0)
Replace missing values with last valid observation (useful in time series data). For example, temperature does not change drastically compared to previous observation. So better approach is to fill NA is to forward or backward fill than mean. There are mainly two methods available 1) 'pad' / 'ffill' - forward fill 2) 'bfill' / 'backfill' - backward fill Limit: If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill	df.fillna(method='ffill', inplace=True, limit = 1)

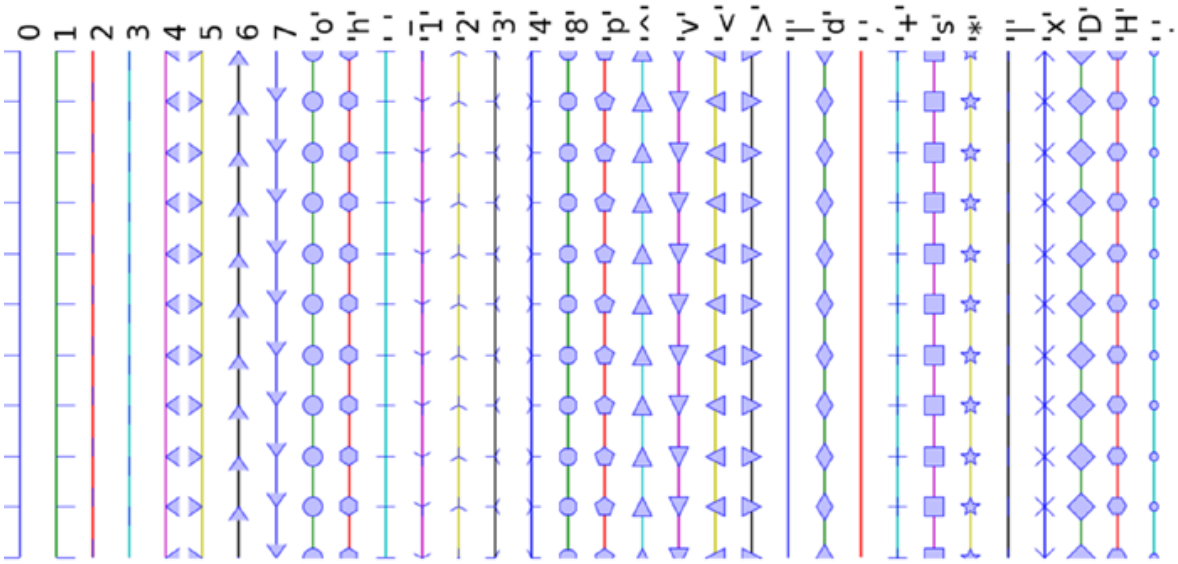
Description	Syntax
Check missing value condition and return Boolean value of true or false for each cell	pd.isnull(df)
Replace all missing values for a given column with its mean	mean=df['column_name'].mean(); df['column_name'].fillna(mean)
Return mean for each column	df.mean()
Return max for each column	df.max()
Return min for each column	df.min()
Return sum for each column	df.sum()
Return count for each column	df.count()
Return cumulative sum for each column	df.cumsum()
Applies a function along any axis of the DataFrame	df.apply(np.cumsum)
Iterate over each element of a series and perform desired action	df['column_name'].map(lambda x: 1+x) # this iterates over the column and adds value 1 to each element

Apply a function to each element of dataframe	func = lambda x: x + 1 # function to add a constant 1 to each element of dataframe
	df.applymap(func)

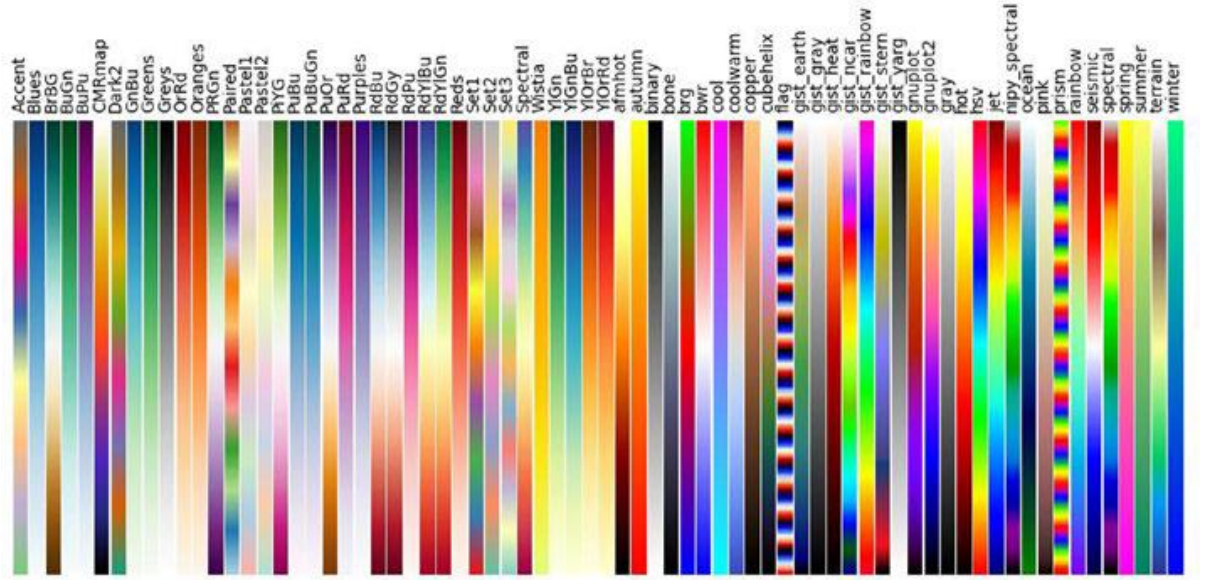
Line Style Reference



Marker reference



Colormaps reference



Nominal Scale of Measurement

Data are measured at the nominal level when each case is classified into one of a number of discrete categories. This is also called categorical, that is, used only for classification.

Variable Name	Example Measurement Values
Color	Red, Green, Yellow, etc.
Gender	Female, Male
Football Players Jersey Number	1, 2, 3, 4, 5, etc.

Ordinal Scale of Measurement

Data are measured on an ordinal scale if the categories imply order. The difference between ranks is consistent in direction and authority, but not magnitude.

Variable Name	Example Measurement Values
Military rank	Second Lieutenant, First Lieutenant, Captain, Major, Lieutenant Colonel, Colonel, etc.
Clothing size	Small, Medium, Large, Extra Large. Etc.
Class rank in an exam	1,2,3,4,5, etc.

Interval Scale of Measurement

If the differences between values have meanings, the data are measured at the interval scale.

Variable Name	Example Measurement Values
Temperature	10, 20, 30, 40, etc.
IQ rating	85 - 114, 115 - 129, 130 - 144, 145 – 159, etc.

Ratio Scale of Measurement

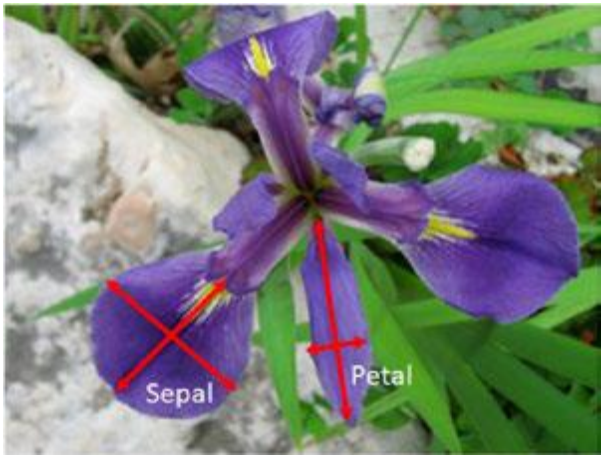
Data measured on a ratio scale have differences that are meaningful, and relate to some true zero point. This is the most common scale of measurement.

Variable Name	Example Measurement Values
Weight	10, 20, 30, 40, 50, 60, etc.
Height	5, 6, 7, 8, 9, etc.
Age	1, 2, 3, 4, 5, 6, 7, etc.

Comparison of Different scales of measurements

	Scales of measurement			
	Nominal	Ordinal	Interval	Ratio
Properties	Identity	Identity Magnitude	Identity Magnitude Equal intervals	Identity Magnitude Equal intervals True zero
Mathematical Operations	Count	Rank order	Addition Subtraction	Addition Subtraction Multiplication Division
Descriptive Statistics	Mode Proportion	Mode Median Range statistics	Mode Median Range statistics Variance Standard deviation	Mode Median Range statistics Variance Standard deviation

IRIS Dataset



Iris Versicolor

Iris Flowers

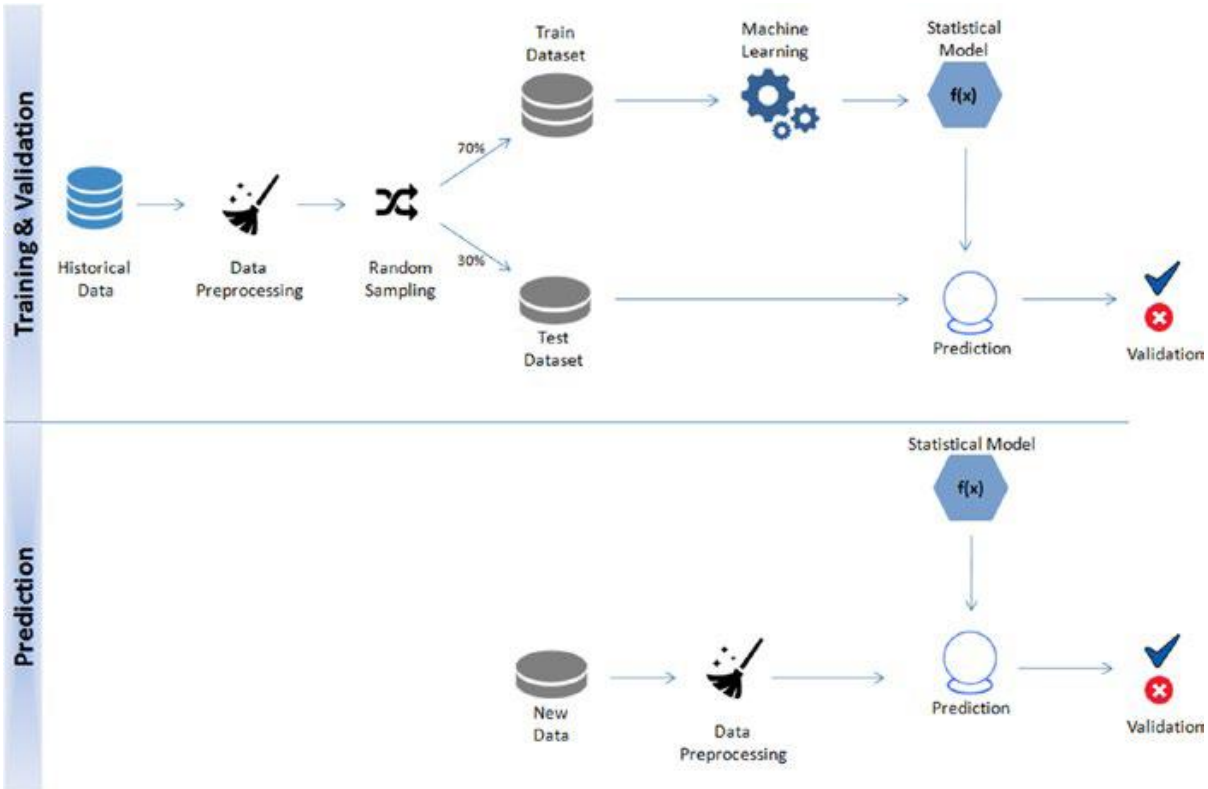
Supervised Learning Use Case Examples : Regression

Domain	Question
Retail	How much will be the daily, monthly, and yearly sales for a given store for the next three years?
Retail	How many car parking spaces should be allocated for a retail store?
Manufacturing	How much will be the product-wise manufacturing labor cost?
Manufacturing / Retail	How much will be my monthly electricity cost for the next three years?
Banking	What is the credit score of a customer?
Insurance	How many customers will claim the insurance this year?
Energy / Environmental	What will be the temperature for the next five days?

Supervised Learning Use case examples: Classification

Domain	Question
Telecom	Is a customer likely to leave the network? (churn prediction)
Retail	Is he a prospective customer?, that is, likelihood of purchase vs. non-purchase?
Insurance	To issue insurance, should a customer be sent for a medical checkup?
Insurance	Will the customer renew the insurance?
Banking	Will a customer default on the loan amount?
Banking	Should a customer be given a loan?
Manufacturing	Will the equipment fail?
Health Care	Is the patient infected with a disease?
Health Care	What type of disease does a patient have?
Entertainment	What is the genre of music?

Supervised Learning: Process Flow



Unsupervised Learning Process Flow