The Pandas Library

Introduction to Pandas

- an open source Python library providing high performance data structures and analysis tools.

```
>>>  import pandas as pd
```

- Often used in conjunction with numpy and matplotlib

```
>>>  import numpy as np
```

```
>>>  import matplotlib.pyplot as plt
```

Reading Files in Pandas

- Use **read_csv(file_path)** function

- Example:

```
import pandas as pd

url="http://analytics.romanko.ca/data/
       customer_dbase_sel.csv"

df = pd.read_csv(url)
```

Data Exploration in Pandas -1

- Display a number of rows from the dataset

```
>>> df.head(n = 5)
```

	custid	gender	age	age_cat	debtinc	card	carditems	cardspent	cardtype	creddebt	...	carown	region	ed_cat	ed_years
0	3964-QJWTRG-NPN	Female	20	18-24	11.1	Mastercard	5	81.66	None	1.20	...	Own	Zone 1	Some college	15
1	0648-AIPJSP-UVM	Male	22	18-24	18.6	Visa	5	42.60	Other	1.22	...	Own	Zone 5	College degree	17
2	5195-TLUDJE-HVO	Female	67	>65	9.9	Visa	9	184.22	None	0.93	...	Own	Zone 3	High school degree	14
3	4459-VLPQUH-3OL	Male	23	18-24	5.7	Visa	17	340.99	None	0.02	...	Own	Zone 4	Some college	16
4	8158-SMTQFB-CNO	Male	26	25-34	1.7	Discover	8	255.10	Gold	0.21	...	Lease	Zone 2	Some college	16

Data Exploration in Pandas -2

- Get summary stats for the data

```
>>> df.describe()
```

	age	debtinc	carditems	cardspent	creddebt	commutetime	card2items	card2spent	cars
count	5000.000000	5000.000000	5000.00000	5000.000000	5000.000000	4998.000000	5000.000000	5000.000000	5000.000000
mean	46.939800	9.957800	10.19920	339.635878	1.874982	25.346739	4.666000	161.331270	2.134200
std	17.703312	6.423173	3.39279	248.382982	3.441425	5.890674	2.482434	146.798035	1.306037
min	18.000000	0.000000	0.00000	0.000000	0.000000	7.000000	0.000000	0.000000	0.000000
25%	32.000000	5.175000	8.00000	184.860000	0.390000	21.000000	3.000000	67.682500	1.000000
50%	46.000000	8.800000	10.00000	278.655000	0.930000	25.000000	5.000000	125.455000	2.000000
75%	62.000000	13.500000	12.00000	422.402500	2.080000	29.000000	6.000000	208.612500	3.000000
max	79.000000	43.100000	23.00000	3926.410000	109.070000	48.000000	15.000000	2069.250000	8.000000

Data Exploration in Pandas -3

- Get further information about the data.

  > Get the number of rows and columns in the data

  ```
  >>> df.shape
      (5000, 30)
  ```

  > Display column names

  ```
  >>> df.columns
  Index(['custid', 'gender', 'age', 'age_cat', 'debtinc', 'card',
  'carditems', 'cardspent', 'cardtype', 'creddebt', 'commute',
  'commutetime', 'card2', 'card2items', 'card2spent',
  'card2type', 'marital', 'homeown','hometype', 'cars',
  'carown', 'region', 'ed_cat', 'ed_years', 'job_cat',
  'employ_years', 'emp_cat', 'retire', 'annual_income',
  'inc_cat'],dtype='object')
  ```

Pandas - “Labeling” Data

- Example: label customers in the data into high-income and low-income

```
df [ 'income_category' ] = df [  'annual_income' ].
            map( lambda x : 1 if x>30000 else 0)

df [ [ 'annual_income' , 'income_category' ] ]. head(n = 5)
```

	annual_income	income_category
0	31000	1
1	15000	0
2	35000	1
3	20000	0
4	23000	0

- Lambda functions:

lambda [arguments] : expression

```
lambda x ,y :  x * y

lambda x : 1 if x>30000 else 0
```

Pandas - Data Cleaning

- Drop NaN (Not-a-Number) observations:
  >>> df [['commutetime']].dropna().count()
  commutetime 4998

- Print observations with NaN commutetime:
  >>> print(df [np.isnan(df ["commutetime"])])

```
              custid  gender  age age_cat  debtinc      card  carditems  \
965   3622-JHDLVP-V1E  Female   48   35-49      6.5  Discover         12
2734  0860-BRGALK-LLR  Female   68     >65     17.3     Other          8

      cardspent  cardtype  creddebt       ...          region          ed_cat  \
965      261.91  Platinum      2.25       ...          Zone 1  College degree
2734     178.75  Platinum      1.08       ...          Zone 5    Some college

     ed_years                                job_cat  employ_years  \
965        19                                Service            12
2734       15  Operation, Fabrication, General Labor            20

           emp_cat retire annual_income     inc_cat  income_category
965       11 to 15     No        121000  $75 - $124                1
2734  More than 15    Yes         23000   Under $25                0

[2 rows x 31 columns]
```

Pandas - Visualizing Data

- Selecting/reducing the number of columns

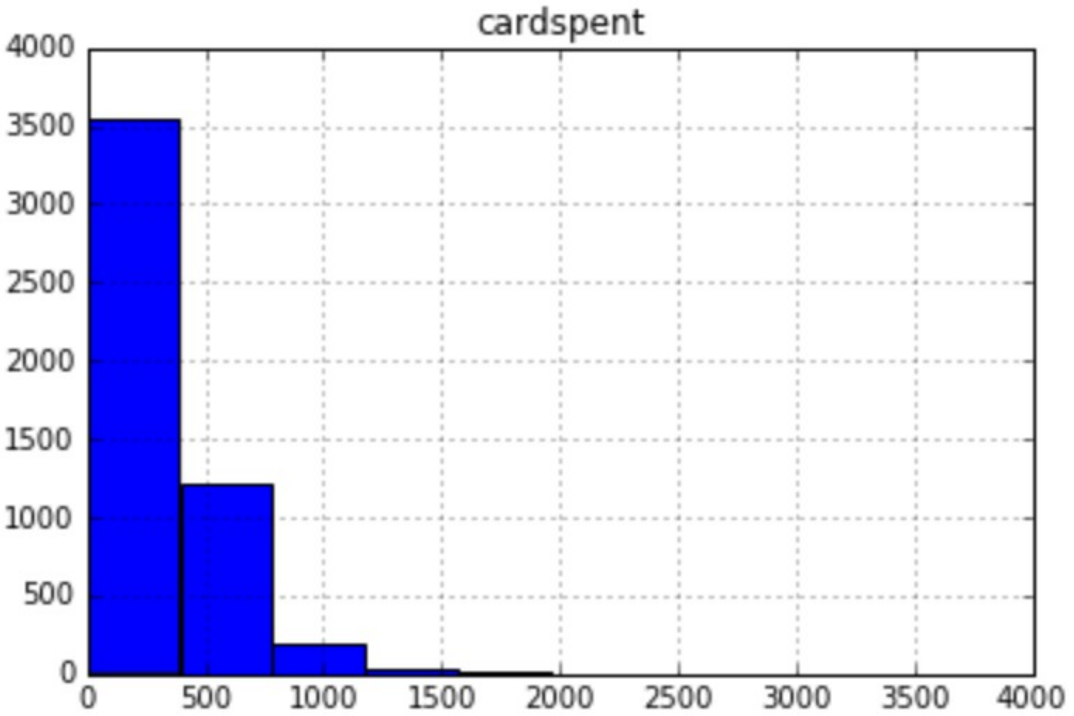
```
viz = df [ ['cardspent' , 'debtinc' , 'carditems' , 'commutetime']]
viz.head()
```

	cardspent	debtinc	carditems	commutetime
0	81.66	11.1	5	22
1	42.60	18.6	5	29
2	184.22	9.9	9	24
3	340.99	5.7	17	38
4	255.10	1.7	8	32

Pandas – Plotting Data -1

- Get a histogram of the data and plot it

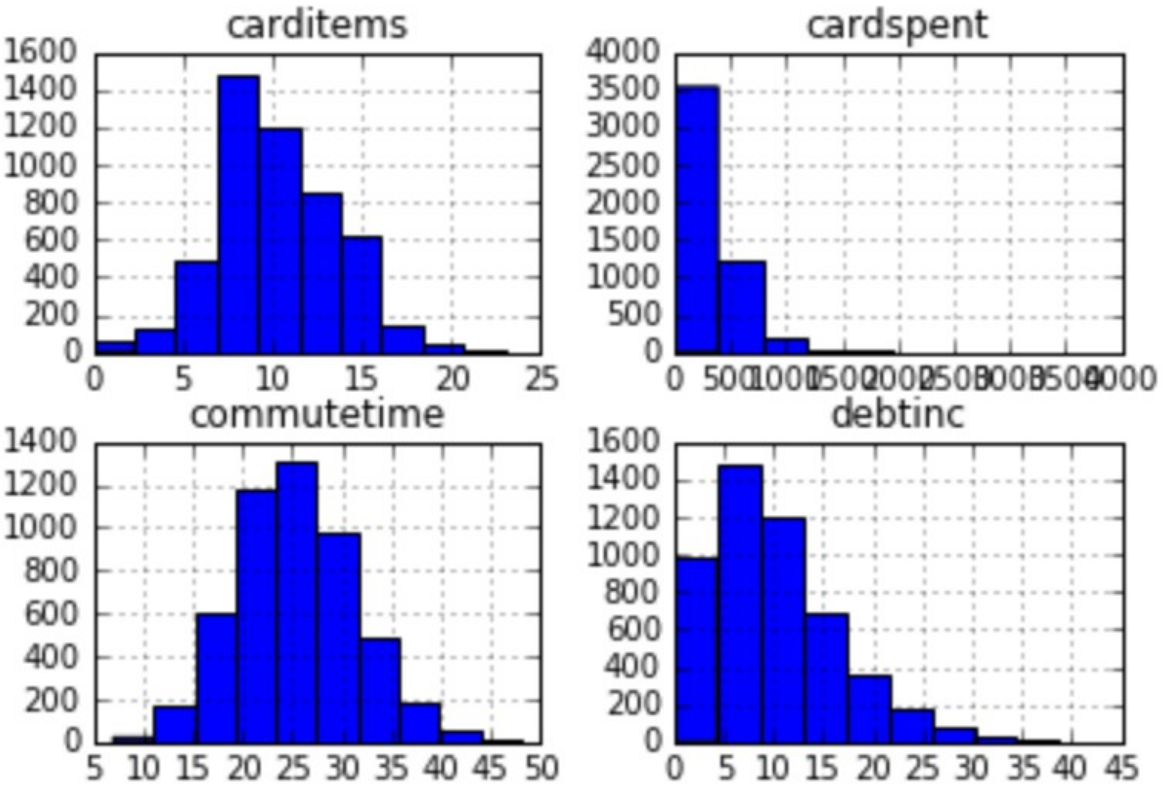
```
df [ [ 'cardspent' ] ].hist()
plt.show()
```



Pandas – Plotting Data -2

- Get a histogram of the data and plot it

```
>>> viz.hist()
>>> plt.show()
```



Pandas Data Structures -Series

- One-dimensional labeled array
- Holds any data type (integers, strings, floating point numbers, Python objects, etc.)
- The axis labels are collectively referred to as the index.

```
>>> s = pd.Series(data, index=index)
```

- **data**: a dictionary, an ndarray, a scalar value (e.g., 11)
- **index:** is a list of axis labels.

Series from from an ndarray

```
>>> s = pd.Series(np.random.randn(5), index=['a',
'b', 'c', 'd', 'e'])
>>> s
a    0.2735
b    0.6052
c   -0.1692
d    1.8298
e    0.5432
dtype: float64

>>> s.index
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

>>> pd.Series(np.random.randn(5))
0    0.3674
1   -0.8230
2   -1.0295
3   -1.0523
4   -0.8502
dtype: float64
```

Series from from a dictionary

- If an **index** is passed, the values in data corresponding to the labels in the index will be pulled out.
- If no **index** is passed, an index will be constructed from the sorted keys of the dict, if possible.

```
>>> d = {'a' : 0., 'b' : 1., 'c' : 2.}
>>> pd.Series(d)
a    0.0
b    1.0
c    2.0
dtype: float64

>>> pd.Series(d, index=['b', 'c', 'd', 'a'])
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```


- **NOTE**: NaN is the standard missing data marker used in pandas

Series from from a scalar value

- If **data** is a scalar value, an **index** must be provided. The value will be repeated to match the length of index

```
>>> pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

Series Behaviour

- Series acts very similarly to a ndarray, and is a valid argument to most NumPy functions.

```
>>> s[0]
>>> 0.27348116325673794
>>> s[:3]
>>> a    0.2735
    b    0.6052
    c   -0.1692
    dtype: float64
```

- A Series is like a fixed-size dict in that you can get and set values by index label:

```
>>> s['a']
>>> 0.27348116325673794
>>> s['e'] = 12.

>>> s.get('a')
>>> 0.27348116325673794
```

DataFrame Objects

- class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)[source]

- Two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled rows and columns.

- Dictionar-like container for Series objects.

- Arithmetic operations align on both row and column labels.

DataFrame - Parameters

- **data** : numpy ndarray, dictionary (Series, arrays, constants, or list-like objects), or DataFrame
- **index** : index or array-like to use for resulting frame.
- **columns** : Index or array-like, labels to use for resulting frame.
- **dtype** : data_type (default None), to force, otherwise infer
- **copy** : boolean (default False), to copy data from inputs.

```
>>> d = {'col1': ts1, 'col2': ts2}
>>> df1 = DataFrame(data = d, index = index)
>>> df2 = DataFrame(numpy.random.randn(10, 5))
>>> df3 = DataFrame(numpy.random.randn(10, 5),
columns=['a', 'b', 'c', 'd', 'e'])
```

- `numpy.random.randn` returns a sample(s) from the "standard normal" distribution.

DataFrames from Series or dictionaries

- The result index will be the union of the indexes of the various Series.

```
d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
     'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
>>> df = pd.DataFrame(d)
>>> df
one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0

>>> pd.DataFrame(d, index=['d', 'b', 'a'])
one  two
d  NaN  4.0
b  2.0  2.0
a  1.0  1.0
```

Accessing Rows and Columns

- The row and column labels can be accessed, respectively, by accessing the index and columns attributes:

```
>>> df.index
Index(['a', 'b', 'c', 'd'], dtype='object')
>>> df.columns
Index(['one', 'two'], dtype='object')
```

Index

- An immutable ndarray implementing an ordered, sliceable set.
- The basic object storing axis labels for all pandas objects
- Parameters:

  data : array-like (1-dimensional)

  dtype : NumPy dtype (default: object)

  copy : bool Make a copy of input ndarray

  name : objectName to be stored in the index

  tupleize_cols : bool (default: True)

  When True, attempt to create a MultiIndex if possible

Index Attributes

Attributes

`T`	return the transpose, which is by definition self
`asi8`	
`base`	return the base object if the memory of the underlying data is
`data`	return the data pointer of the underlying data
`dtype`	
`dtype_str`	
`flags`	
`has_duplicates`	
`hasnans`	
`inferred_type`	
`is_all_dates`	
`is_monotonic`	alias for is_monotonic_increasing (deprecated)
`is_monotonic_decreasing`	return if the index is monotonic decreasing (only equal or
`is_monotonic_increasing`	return if the index is monotonic increasing (only equal or
`is_unique`	
`itemsize`	return the size of the dtype of the item of the underlying data

Index Methods

Methods

all(*args, **kwargs)	Return whether all elements are True
any(*args, **kwargs)	Return whether any element is True
append(other)	Append a collection of Index options together
argmax([axis])	return a ndarray of the maximum argument indexer
argmin([axis])	return a ndarray of the minimum argument indexer
argsort(*args, **kwargs)	Returns the indices that would sort the index and its underlying data.
asof(label)	For a sorted index, return the most recent label up to and including the passed label.
asof_locs(where, mask)	where : array of timestamps
astype(dtype[, copy])	Create an Index with values cast to dtypes.
copy([name, deep, dtype])	Make a copy of this object.
delete(loc)	Make new Index with passed location(-s) deleted
difference(other)	Return a new Index with elements from the index that are not in *other*.
drop(labels[, errors])	Make new Index with passed list of labels deleted
drop_duplicates(*args, **kwargs)	Return Index with duplicate values removed

Reshaping DataFrame Objects by pivoting -1

- Reshaping by pivoting DataFrame objects
- Data is often stored in CSV files or databases in so-called "stacked" or "record" format:

```
>>> df
        date variable     value
0 2000-01-03        A  0.469112
1 2000-01-04        A -0.282863
2 2000-01-05        A -1.509059
3 2000-01-03        B -1.135632
4 2000-01-04        B  1.212112
5 2000-01-05        B -0.173215
6 2000-01-03        C  0.119209
7 2000-01-04        C -1.044236
8 2000-01-05        C -0.861849
9 2000-01-03        D -2.104569
10 2000-01-04       D -0.494929
11 2000-01-05       D  1.071804
```

Reshaping DataFrame Objects by pivoting -2

- To select out everything for variable A we could do:

```
>>>  df [ df [ 'variable' ]  == 'A' ]
>>>  date            variable    value
  0  2000-01-03       A           0.469112
  1  2000-01-04       A          -0.282863
  2  2000-01-05       A          -1.509059
```

- For time series operations a better representation would have the columns as unique variables and an index of dates identifying individual observations.
- To reshape the data use the **pivot** function:

```
>>> df.pivot( index = 'date', columns = 'variable', values = 'value')
>>> variable              A          B          C          D
    date
    2000-01-03    0.469112   -1.135632   0.119209   -2.104569
    2000-01-04   -0.282863    1.212112  -1.044236   -0.494929
    2000-01-05   -1.509059   -0.173215  -0.861849    1.071804
```

Computing Descriptive Statistics

- DataFrame.describe(percentiles=None, include=None, exclude=None)[source]
- Generate various summary statistics, excluding NaN values.
- **Parameters**:
- **percentiles** : array-like, optional. The percentiles to include in the output. Should all be in the interval [0, 1].
- **include**, **exclude** : list-like, 'all', or None (default) Specify the form of the returned result. Either:

  - None to both (default). The result will include only numeric-typed columns or, if none are, only categorical columns.

  - A list of dtypes or strings to be included/excluded.

  - If **include**= 'all', the output column-set will match the input one.

Returns: summary statistics