Introduction to Data Science

Lecture 8

Data Wrangling, preprocessing, and Visualization

Part I

Data Preprocessing (cleaning)

1. Reading the file in a structural format (excel type format)
2. Dealing with missing values
3. Finding outliers and dealing with them
4. normalization
5. possible visualization

Reading and Handling Data files using Pandas

Function	Description
read_csv	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
read_table	Load delimited data from a file, URL, or file-like object; use tab ('\t') as default delimiter
read_fwf	Read data in fixed-width column format (i.e., no delimiters)
read_clipboard	Version of read_table that reads data from the clipboard; useful for converting tables from web pages
read_excel	Read tabular data from an Excel XLS or XLSX file
read_hdf	Read HDF5 files written by pandas
read_html	Read all tables found in the given HTML document
read_json	Read data from a JSON (JavaScript Object Notation) string representation
read_msgpack	Read pandas data encoded using the MessagePack binary format
read_pickle	Read an arbitrary object stored in Python pickle format
read_sas	Read a SAS dataset stored in one of the SAS system's custom storage formats
read_sql	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
read_stata	Read a dataset from Stata file format
read_feather	Read the Feather binary file format

Working on AirPassengers data set:

- The classic Box & Jenkins airline data. Monthly totals of international airline passengers, 1949 to 1960.
- source AARSHAY JAIN

In [2]:
```
import pandas as pd
import numpy as np
import datetime as dtime
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
```

Now, we can load the data set and look at some initial rows and data types of the columns:

In [16]:
```
filepath = '/Users/martin/Documents/MyLecturesSBU/Fall2018/CSE391/data/A
data = pd.read_csv(filepath)
data.head(10)
#print('\n Data Types:')
#print(data.dtypes)
```

Out[16]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
5	1949-06	135
6	1949-07	148
7	1949-08	148
8	1949-09	136
9	1949-10	119

In [17]:
```
?pd.read_csv
```

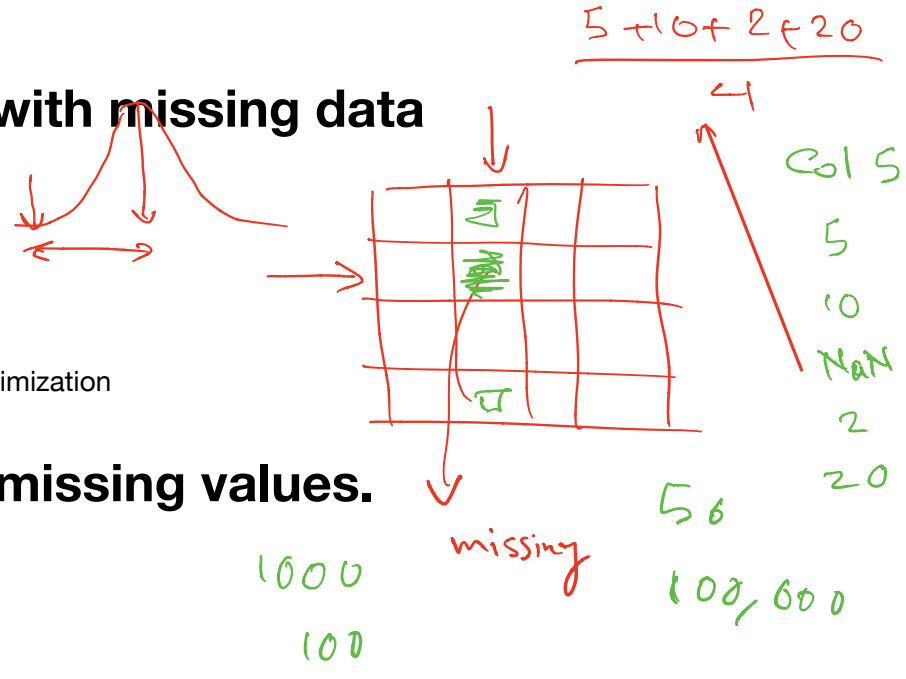
Dealing with missing values

Two type of missing data

- missing completly at random
- mising not at random

How to deal with missing data

- drop out
- imputation
  - expert gussing
  - averaging
  - regression
  - Expectation maximization



1.1 Drop the missing values.

In [7]:
```
filepath = '/Users/martin/Documents/MyLecturesSBU/Fall2018/CSE391/data/
titanic_train.csv'
passengers = pd.read_csv(filepath)
passengers.dropna(axis=0, how='any', thresh=None, subset=None,
                  inplace=False)
passengers.describe()
passengers.head()
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

- This will drop any rows with missing values. Clearly this isn't a good idea
- What instead if we wanted to remove any columns with missing values? How?

In [19]:
```
passengers.dropna(axis = 1).head(5)
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	0	0	373450	8.0500

In [ ]:
```
1
```

In [ ]:
```
# Keep only the rows with at least 11 non-na values:
passengers.dropna(thresh = 11)
```

1.2 imputation (a way to replace the missing values)

- Filling the missing values
  - mean
  - your suggestion? median

median

In [7]:
```
passengers["Age"].fillna(value=passengers["Age"].mean()).head()
```

Out[7]:
```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

Summarizing and Computing Descriptive Statistics

Pandas objects are equipped with a set of common mathematical and statistical methods. Most of these fall into the category of reductions or summary statistics, methods that extract a single value (like the sum or mean) from a Series or a Series of values from the rows or columns of a DataFrame.

In [9]:
```
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5], [np.nan, np.nan], [0.75,
.....: columns=['one', 'two'])
print(df)
```

```
   one  two
a  1.40  NaN
b  7.10 -4.5
c   NaN  NaN
d  0.75 -1.3
```

- Calling DataFrame's sum method returns a Series containing column sums

In [11]:
```
df.sum()
```

Out[11]:
```
one    9.25
two   -5.80
dtype: float64
```

- computing mean

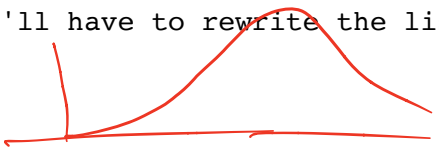
In [ ]:
```
df.mean(axis='columns', skipna=False)
```

m

Method	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

Note on mean, median, mode and their differences:

```
Find the mean, median, mode, and range for the following list of values:
13, 18, 13, 14, 13, 16, 14, 21, 13

The mean is the usual average, so I'll add and then divide:

(13 + 18 + 13 + 14 + 13 + 16 + 14 + 21 + 13) ÷ 9 = 15

Note that the mean, in this case, isn't a value from the original list. This is a common result. You should not assume that your mean will be one of your original numbers.

The median is the middle value, so first I'll have to rewrite the list in numerical order:

13, 13, 13, 13, 14, 14, 16, 18, 21

There are nine numbers in the list, so the middle one will be the (9 + 1) ÷ 2 = 10 ÷ 2 = 5th number:

```



```
13, 13, 13, 13, 14, 14, 16, 18, 21

So the median is 14.

The mode is the number that is repeated more often than any other, so 13 is the mode.

The largest value in the list is 21, and the smallest is 13, so the range is 21 – 13 = 8.

mean: 15
median: 14
mode: 13
range: 8

Note: The formula for the place to find the median is "([the number of data points] + 1) ÷ 2", but you don't have to use this formula. You can just count in from both ends of the list until you meet in the middle, if you prefer, especially if your list is short. Either way will work.
```

Reading Text Files in Pieces

- When processing very large files or figuring out the right set of arguments to correctly process a large file, you may only want to read in a small piece of a file or iterate through smaller chunks of the file.

In [ ]:
```
# If you want to only read a small number of rows (avoiding reading the
pd.read_csv('examples/ex6.csv', nrows=5)
# To read a file in pieces, specify a chunksize as a number of rows:
chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

# The TextParser object returned by read_csv allows you to iterate over
#the value counts in the 'key' column like so:
chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)
tot = pd.Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)
    tot = tot.sort_values(ascending=False)
```

Pivot tables in Pandas

Panada pivot table example

- Adapted fromChris Moffitt (http://pbpython.com/pandas-pivot-table-explained.html)

In [2]:
```
import pandas as pd
```

In [5]:
```
# Read in the data; sale pipleline data
df = pd.read_excel("/Users/martin/Documents/MyLecturesSBU/past/
                   Spring2018/CSE391/data/sales-funnel.xlsx")
df
```

3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won
5	218895	Kulas Inc	Daniel Hilton	Debra Henley	CPU	2	40000	pending
6	218895	Kulas Inc	Daniel Hilton	Debra Henley	Software	1	10000	presented
7	412290	Jerde-Hilpert	John Smith	Debra Henley	Maintenance	2	5000	pending
8	740150	Barton LLC	John Smith	Debra Henley	CPU	1	35000	declined
9	141962	Herman LLC	Cedric Moss	Fred Anderson	CPU	2	65000	won
10	163416	Purdy-Kunde	Cedric Moss	Fred Anderson	CPU	1	30000	presented
11	239344	Stokes LLC	Cedric [illegible]	Fred [illegible]	Maintenance	1	5000	pending

In [7]:
```
# This isn't strictly required but helps us keep the order we want as
#we work through analyzing the data.

df["Status"] = df["Status"].astype("category")
df["Status"].cat.set_categories(["won","pending","presented","declined"]
                                inplace=True)

```

In [8]:
```
#The simplest pivot table must have a dataframe and an index .
#In this case, let's use the Name as our index.
pd.pivot_table(df,index=["Name"])
```

Barton LLC	740150.0	35000.0	1.000000
Fritsch, Russel and Anderson	737550.0	35000.0	1.000000
Herman LLC	141962.0	65000.0	2.000000
Jerde-Hilpert	412290.0	5000.0	2.000000
Kassulke, Ondricka and Metz	307599.0	7000.0	3.000000
Keeling LLC	688981.0	100000.0	5.000000
Kiehn-Spinka	146832.0	65000.0	2.000000
Koepp Ltd	729833.0	35000.0	2.000000
Kulas Inc	218895.0	25000.0	1.500000
Purdy-Kunde	163416.0	30000.0	1.000000
Stokes LLC	239344.0	7500.0	1.000000
Trantow-Barrows	714466.0	15000.0	1.333333

In [9]:
```
# You can have multiple indexes as well
pd.pivot_table(df,index=["Name","Rep","Manager"])
```

Out[9]:

			Account	Price	Quantity
Name	**Rep**	**Manager**			
Barton LLC	**John Smith**	**Debra Henley**	740150.0	35000.0	1.000000
Fritsch, Russel and Anderson	**Craig Booker**	**Debra Henley**	737550.0	35000.0	1.000000
Herman LLC	**Cedric Moss**	**Fred Anderson**	141962.0	65000.0	2.000000
Jerde-Hilpert	**John Smith**	**Debra Henley**	412290.0	5000.0	2.000000
Kassulke, Ondricka and Metz	**Wendy Yule**	**Fred Anderson**	307599.0	7000.0	3.000000
Keeling LLC	**Wendy Yule**	**Fred Anderson**	688981.0	100000.0	5.000000
Kiehn-Spinka	**Daniel Hilton**	**Debra Henley**	146832.0	65000.0	2.000000
Koepp Ltd	**Wendy Yule**	**Fred Anderson**	729833.0	35000.0	2.000000
Kulas Inc	**Daniel Hilton**	**Debra Henley**	218895.0	25000.0	1.500000
Purdy-Kunde	**Cedric Moss**	**Fred Anderson**	163416.0	30000.0	1.000000
Stokes LLC	**Cedric Moss**	**Fred Anderson**	239344.0	7500.0	1.000000
Trantow-Barrows	**Craig Booker**	**Debra Henley**	714466.0	15000.0	1.333333

In [10]:
```
pd.pivot_table(df,index=["Manager","Rep"])
```

Out[10]:

Manager	Rep	Account	Price	Quantity
Debra Henley	Craig Booker	720237.0	20000.000000	1.250000
	Daniel Hilton	194874.0	38333.333333	1.666667
	John Smith	576220.0	20000.000000	1.500000
Fred Anderson	Cedric Moss	196016.5	27500.000000	1.250000
	Wendy Yule	614061.5	44250.000000	3.000000

In [11]:
```
 # the Account and Quantity columns aren't really useful.
#Let's remove it by explicitly defining the columns
#we care about using the values field.

pd.pivot_table(df,index=["Manager","Rep"],values=["Price"])
```

Out[11]:

Manager	Rep	Price
Debra Henley	Craig Booker	20000.000000
	Daniel Hilton	38333.333333
	John Smith	20000.000000
Fred Anderson	Cedric Moss	27500.000000
	Wendy Yule	44250.000000

In [13]:
```
import numpy as np
# The price column automatically averages the data
# but we can do a count or a sum. Adding them is simple
#using aggfunc and np.sum .
pd.pivot_table(df,index=["Manager","Rep"],
               values=["Price"],aggfunc=np.sum)
```

Out[13]:

Manager	Rep	Price
Debra Henley	Craig Booker	80000
	Daniel Hilton	115000
	John Smith	40000
Fred Anderson	Cedric Moss	110000
	Wendy Yule	177000

- if we want to see sales broken down by the products, the columns variable allows us to define one or more columns.

In [14]:
```
table = pd.pivot_table(df,index=["Manager","Rep"],values=["Price"],
                       columns=["Product"],aggfunc=[np.sum])
table
```

Out[14]:

		sum			
		Price			
	Product	CPU	Maintenance	Monitor	Software
Manager	Rep				
Debra Henley	Craig Booker	65000.0	5000.0	NaN	10000.0
	Daniel Hilton	105000.0	NaN	NaN	10000.0
Fred Anderson	John Smith	35000.0	5000.0	NaN	NaN
	Cedric Moss	95000.0	5000.0	NaN	10000.0
	Wendy Yule	165000.0	7000.0	5000.0	NaN

- Remember, columns are optional - they provide an additional way to segment the actual values you care about. The aggregation functions are applied to the values you list.

Join and Merge Pandas Dataframe

- adapted from Chris Albon (https://chrisalbon.com/python/data_wrangling/pandas_join_merge_dataframe/)

In [ ]:
```
#Create a dataframe
raw_data = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'la
df_a
```

In [ ]:
```
# Create a second dataframe
raw_data = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'la
df_b
```

In [ ]:
```
# create a third dataframe
raw_data = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
df_n = pd.DataFrame(raw_data, columns = ['subject_id','test_id'])
df_n
```

- Join the two dataframes along rows

In [ ]:
```
df_new = pd.concat([df_a, df_b])

df_new
```

- Join the two dataframes along columns

In [ ]:
```
pd.concat([df_a, df_b], axis=1)
```

Reading big files using Pandas

In [ ]:
```
import pandas as pd
import numpy as np
#SQLAlchemy is the Python SQL toolkit and Object Relational Mapper
# that gives application developers the full power and
#flexibility of SQL.
from sqlalchemy import create_engine
```

In [ ]:
```
file = '/Users/martin/Documents/TCGA/tcga_mRNA/COAD_mRNA/COAD.uncv2.mRNA


#This command uses pandas' "read_csv" command to read
#in only 5 rows (nrows=5)
# and then print those rows to the screen.
pd.read_csv(file,sep='\t',nrows=1)
```

- Before we can actually work with the data, we need to do something with it so we can begin to filter it to work with subsets of the data. This is usually what I would use pandas' dataframe for but with large data files, we need to store the data somewhere else. In this case, we'll set up a local sqllite database, read the csv file in chunks and then write those chunks to sqllite.
- To do this, we'll first need to create the sqllite database using the following command.

In [ ]:
```
csv_database = create_engine('sqlite:///csv_database.db')
```

In [ ]:
```
#Next, we need to iterate through the CSV file in chunks and store the c

chunksize = 1000
i = 0
j = 1
for df in pd.read_csv(file, chunksize=chunksize, iterator=True):
      df = df.rename(columns={c: c.replace(' ', '') for c in df.columns}
      df.index += j
      i+=1
      df.to_sql('table', csv_database, if_exists='append')
      j = df.index[-1] + 1
```

In [ ]:
```
# To access the data now, you can run commands like the following:

df = pd.read_sql_query('SELECT * FROM "table"', csv_database)

```

In [ ]:
```

```