Session-11

Time Series

datetime Library

```
root@6879840ae648:/datascience/sessions/eleven# more 1.py
#!/usr/bin/python3
from datetime import datetime

now = datetime.now()
print('The dateformat: {}'.format(now))
print ('The year is = {}'.format(now.year))
print ('The month is = {}'.format(now.month))
print ('The day is = {} '.format(now.day))

#           datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
#                      Y   M   d    h   m    s      ms
that_day = datetime(2017, 3, 1, 17, 9,  21, 832092)
print('The dateformat: {}'.format(that_day))
print ('The year is = {}'.format(that_day.year))
print ('The month is = {}'.format( that_day.month))
print ('The day is = {} '.format(that_day.day))

root@6879840ae648:/datascience/sessions/eleven# ./1.py
The dateformat: 2017-03-31 14:52:04.730385
The year is = 2017
The month is = 3
The day is = 31
The dateformat: 2017-03-01 17:09:21.832092
The year is = 2017
The month is = 3
The day is = 1
root@6879840ae648:/datascience/sessions/eleven#
```

Difference - datetime

```
root@6879840ae648:/datascience/sessions/eleven# more delta.py
#!/usr/bin/python3
from datetime import datetime
#           datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
#                   Y  M   d   h   m  s     ms                   Y  M  d   h   m  s     ms
delta = datetime(2017, 3, 25, 18, 21, 1, 123000) - datetime (2016, 1, 1, 18, 51, 51, 123000)
print(delta)
print (delta.days)
print (delta.seconds)


root@6879840ae648:/datascience/sessions/eleven# ./delta.py
448 days, 23:29:10
448
84550
```

timedelta

```
root@6879840ae648:/datascience/sessions/eleven# more timedelta.py
#!/usr/bin/python3
from datetime import datetime, timedelta

start = datetime(2017, 4, 1)
print (start)


new_s = start + timedelta (12)
print (new_s)

prev_s = start - 2 * timedelta(12)

print (prev_s)


root@6879840ae648:/datascience/sessions/eleven# ./timedelta.py
2017-04-01 00:00:00
2017-04-13 00:00:00
2017-03-08 00:00:00
```

strings and datetime

```
root@6879840ae648:/datascience/sessions/eleven# more str_1.py
#!/usr/bin/python3
from datetime import datetime

d_stamp = datetime(2017, 3, 31)
print(d_stamp)
print (str(d_stamp))
print ("---------------------\n")

# Use of strftime  - datetime to string
print (d_stamp.strftime('%Y/%m/%d'))
print (d_stamp.strftime('%Y-%m-%d'))
print (d_stamp.strftime('%Y, %d %m'))
print ("---------------------\n")


# Use of strptime - strings to datetime
d = '2017-03-31'

print (datetime.strptime(d, '%Y-%m-%d'))
print ("---------------------\n")

datestrs = ['01/Jan/2017', '01/Feb/2017', '01/Mar/2017', '01/Apr/2017']
print ([datetime.strptime(x, '%d/%b/%Y') for x in datestrs])
print ("---------------------\n")
```

strings and datetime (cont...)

```
root@6879840ae648:/datascience/sessions/eleven# ./str_1.py
2017-03-31 00:00:00
2017-03-31 00:00:00
---------------------

2017/03/31
2017-03-31
2017, 31 03
---------------------

2017-03-31 00:00:00
---------------------

[datetime.datetime(2017, 1, 1, 0, 0), datetime.datetime(2017, 2, 1, 0, 0), datetime.datetime(2017, 3, 1, 0, 0), datetime.datetime(2017, 4, 1, 0, 0)]
---------------------
```

dateutil

```
root@6879840ae648:/datascience/sessions/eleven# more pars_util.py
#!/usr/bin/python3
from dateutil.parser import parse

d1 = parse('2017-01-25')
print (type(d1))
print (d1)
print ("------------\n")

d2 = parse('Jan 01, 2017 10:30 AM')
print (type(d2))
print (d2)
print ("------------\n")

d3 = parse('01/01/2017', dayfirst=True)
print (type(d3))
print (d3)
print ("------------\n")

root@6879840ae648:/datascience/sessions/eleven# ./pars_util.py
<class 'datetime.datetime'>
2017-01-25 00:00:00
------------


<class 'datetime.datetime'>
2017-01-01 10:30:00
------------


<class 'datetime.datetime'>
2017-01-01 00:00:00
```

pandas & datetime

```
root@6879840ae648:/datascience/sessions/eleven# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> datestrs = ['01/01/2017', '02/01/2017']
>>> pd.to_datetime(datestrs)
DatetimeIndex(['2017-01-01', '2017-02-01'], dtype='datetime64[ns]', freq=None)
>>> idx = pd.to_datetime(datestrs + [None, ''])
>>> idx
DatetimeIndex(['2017-01-01', '2017-02-01', 'NaT', 'NaT'], dtype='datetime64[ns]', freq=None)
>>> idx[2]
NaT
>>> idx[3]
NaT
>>> pd.isnull(idx)
array([False, False,  True,  True], dtype=bool)
>>>
```

Series

```
root@6879840ae648:/datascience/sessions/eleven# more time_basics_pandas.py
#!/usr/bin/python3
import pandas as pd
import numpy as np

from pandas import Series, DataFrame
from datetime import datetime

dates = [
          datetime(2017, 1, 1),
          datetime(2017, 1, 2),
          datetime(2017, 1, 3),
          datetime(2017, 1, 4),
          datetime(2017, 1, 5),
        ]

ts =  Series(np.random.randn(5), index=dates)
print(type(ts))
print(ts)
print ("------------\n")

print(type(ts.index))
print(ts.index)
print ("------------\n")

print (ts.index[0])
print (type(ts.index[0]))
print (ts.index.dtype)
print ("------------\n")
```

Series (Cont...)

```
root@6879840ae648:/datascience/sessions/eleven# ./time_basics_pandas.py
<class 'pandas.core.series.Series'>
2017-01-01    0.949770
2017-01-02   -0.165319
2017-01-03   -1.112256
2017-01-04   -0.393262
2017-01-05   -0.828009
dtype: float64
------------

<class 'pandas.tseries.index.DatetimeIndex'>
DatetimeIndex(['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04',
               '2017-01-05'],
              dtype='datetime64[ns]', freq=None)
------------

2017-01-01 00:00:00
<class 'pandas.tslib.Timestamp'>
datetime64[ns]
------------
```

Handling Missing Values

Data Sets Missing - Why?

So many Reasons -

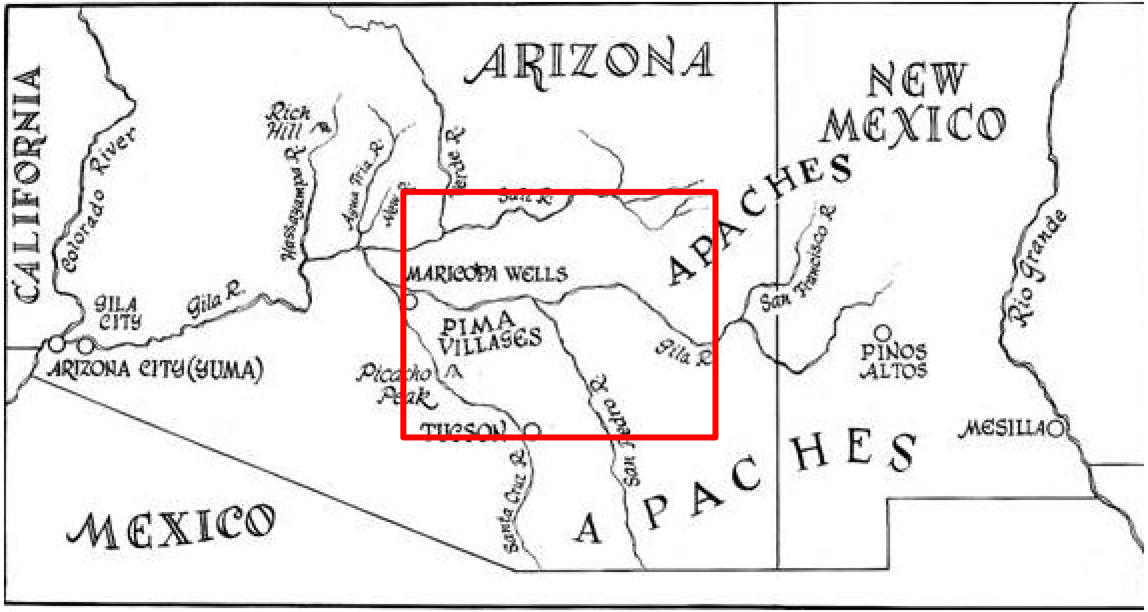
- Data could not get recorded - Manual, Automation (both can fail - its magical)
- Data got corrupted - world is not perfect

What do we do then?

- We need to handle it sensibly - off course, knowledge about data plays important Role

Example - Pima Indians Diabetes Dataset

- Info on Pima Indians - https://en.wikipedia.org/wiki/Pima_people
- Data Source - https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes



Pima

O'odham portraits

Pima Indians Diabetes Dataset (Cont...)

The number of observations for each class is not balanced. There are 768 observations with 8 input variables. The variable names are as follows:

0	Number of times pregnant.
1	Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2	Diastolic blood pressure (mm Hg).
3	Triceps skinfold thickness (mm).
4	2-Hour serum insulin (mu U/ml).
5	Body mass index (weight in kg/(height in m)^2).
6	Diabetes pedigree function.
7	Age (years).
8	Class variable (0 or 1).

Pima Indians Diabetes Dataset (Cont...)

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

zero for body mass index or blood pressure is invalid.

Pima Indians Diabetes Dataset (Cont...)

You can not have “Zero” to a few columns

1, 2, 3, 4, 5

1	Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2	Diastolic blood pressure (mm Hg).
3	Triceps skinfold thickness (mm).
4	2-Hour serum insulin (mu U/ml).
5	Body mass index (weight in kg/(height in m)^2).

In [15]:
```python
import pandas as pd
dataset = pd.read_csv('pima_diab.csv', header=None)
dataset.describe()
```

Zero is affecting here

Out[15]:

	0	1	2	3	4	5	6	7	8
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

The columns

2, 3, 4, 5 - clearly has

“0” as values.

That’s confirmation :)

In [9]:
```
dataset.head(20)
```

Out[9]:

	0	1	2	3	4	5	6	7	8
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1
11	4	110	92	0	0	37.6	0.191	30	0
12	10	168	74	0	0	38	0.537	34	1
13	10	139	80	0	0	27.1	1.441	57	0
14	1	189	60	23	846	30.1	0.398	59	1
15	5	166	72	19	175	25.8	0.587	51	1
16	7	100	0	0	0	30	0.484	32	1
17	0	118	84	47	230	45.8	0.551	31	1
18	7	107	74	0	0	29.6	0.254	31	1
19	1	103	30	38	83	43.3	0.183	33	0

How many invalid records?

```
In [16]: print((dataset[[1,2,3,4,5]] == 0).sum())

1      5
2     35
3    227
4    374
5     11
dtype: int64
```

Replace invalid with meaningful values

In [18]:
```
import numpy as np
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, np.NaN)
```

In [19]:
```
dataset
```

Out[19]:

	0	1	2	3	4	5	6	7	8
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
5	5	116.0	74.0	NaN	NaN	25.6	0.201	30	0
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
7	10	115.0	NaN	NaN	NaN	35.3	0.134	29	0
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
9	8	125.0	96.0	NaN	NaN	NaN	0.232	54	1
10	4	110.0	92.0	NaN	NaN	37.6	0.191	30	0
11	10	168.0	74.0	NaN	NaN	38.0	0.537	34	1
12	10	139.0	80.0	NaN	NaN	27.1	1.441	57	0

Let us mark them with "NaN" using pandas "replace" method

In [20]:
```
print(dataset.isnull().sum())
```

```
0      0
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8      0
dtype: int64
```

Missing Values are trouble

Missing Values can be troublesome to some of the Machine Learning Algorithms

Let us evaluate Linear Discriminant Analysis (LDA) here.

Missing Values are trouble (Cont…)

```
In [13]: from pandas import read_csv
         import numpy
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score

         dataset = read_csv('pima_diab.csv', header=None)
         # mark zero values as missing or NaN
         dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

         # split dataset into inputs and outputs
         values = dataset.values
         X = values[:,0:8]
         y = values[:,8]

         # evaluate an LDA model on the dataset using k-fold cross validation
         model = LinearDiscriminantAnalysis()
         kfold = KFold(n_splits=3, random_state=7)
         result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')

         print(result.mean())
```

Missing Values are trouble (Cont...)

We are prevented from evaluating an LDA algorithm (and other algorithms) on the dataset with missing values.

```
    522     if multi_output:
    523         y = check_array(y, 'csr', force_all_finite=True, ensure_2d=False,

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, dtype, order,
 copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    405                              % (array.ndim, estimator_name))
    406         if force_all_finite:
--> 407             _assert_all_finite(array)
    408
    409     shape_repr = _shape_repr(array.shape)

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py in _assert_all_finite(X)
     56             and not np.isfinite(X).all()):
     57         raise ValueError("Input contains NaN, infinity"
---> 58                          " or a value too large for %r." % X.dtype)
     59
     60

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

Drop the Rows with Missing Values

Pandas provides the “dropna()” function that can be used to drop either columns or rows with missing data

```
In [14]: from pandas import read_csv
         import numpy
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score

         dataset = read_csv('pima_diab.csv', header=None)
         # mark zero values as missing or NaN
         dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

         # drop rows with missing values
         dataset.dropna(inplace=True)
         # summarize the number of rows and columns in the dataset
         print(dataset.shape)

         (392, 9)
```

Finally

```
In [15]: from pandas import read_csv
         import numpy
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score

         dataset = read_csv('pima_diab.csv', header=None)
         # mark zero values as missing or NaN
         dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

         # drop rows with missing values
         dataset.dropna(inplace=True)
         # summarize the number of rows and columns in the dataset
         print(dataset.shape)

         # split dataset into inputs and outputs
         values = dataset.values
         X = values[:,0:8]
         y = values[:,8]

         # evaluate an LDA model on the dataset using k-fold cross validation
         model = LinearDiscriminantAnalysis()
         kfold = KFold(n_splits=3, random_state=7)
         result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
         print(result.mean())

(392, 9)
0.78582892934
```

Impute those Values

Removing rows with missing values can be too limiting on some predictive modeling problems, an alternative is to impute missing values.

A few Options to replace a missing value

- A constant value that has meaning within the domain, such as 0, distinct from all other values.
- A value from another randomly selected record.
- A mean, median or mode value for the column.
- A value estimated by another predictive model.

So new Data Derived - and new models applied on the data

Let us go with mean column values.

These mean column values will need to be stored to file for later use.

Pandas provides the “fillna()” function for replacing missing values with a specific value.

Replace with Mean

In [16]:
```
from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
dtype: int64
```

Finally

In [17]:

```
from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
#print(dataset.isnull().sum())

values = dataset.values
X = values[:,0:8]
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

```
0.766927083333
```

Thank you