Python tools for Machine Learning

Dr. Amilcar Soares

Institute for Big Data Analytics

Summary

Institute for Big Data Analytics

- Pandas
- The animals tracking dataset
- Scikit-learn
- Assignment
- Extra challenge - Trajectory visualization

Pandas (1)

- **Pandas consists of:**
  - A set of labeled array **data structures**, the primary of which are **Series** and **DataFrames**
  - Index objects enabling both **simple axis indexing and multi-level / hierarch**
    indexing
  - An integrated **group by** engine **for aggregating** and **transforming** data sets
  - **Input/Output tools**: loading tabular data from flat files (CSV, delimited, Excel
    saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
  - **Memory-efficient** “sparse” versions of the **standard data structures** for stori
    mostly missing or mostly constant (some fixed value)
  - **Moving window statistics** (rolling mean, rolling standard deviation, etc.)

...

Pandas (2)

Institute for Big Data Analytics

- **Data structures**

Dimensions	Name
1	Series
2	DataFrame
3	Panel

The best way to think about the pandas data structures is as **flexible containers** for lower dimensional data. For example, DataFrame is a container for Series, and Panel is a container for DataFrame objects. We would like to **be able** to **insert and remove objects from these containers** in a dictionary-like fashion.

Pandas (3)

Institute for Big Data Analytics

- **DataFrames**
  - **DataFrame** is a 2-dimensional labeled data structure with columns of potentia
    types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

```
In [32]: d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
   .....:      'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
   .....:

In [33]: df = pd.DataFrame(d)

In [34]: df
Out[34]:
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
```

Pandas (4)

- **DataFrames:**

```
In [56]: df['one']
Out[56]:
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64

In [57]: df['three'] = df['one'] * df['two']

In [58]: df['flag'] = df['one'] > 2

In [59]: df
Out[59]:
   one  two  three   flag
a  1.0  1.0    1.0  False
b  2.0  2.0    4.0  False
c  3.0  3.0    9.0   True
d  NaN  4.0    NaN  False
```

```
In [72]: iris.assign(sepal_ratio = lambda x: (x['SepalWidth'] /
   ....:                                      x['SepalLength'])).head()
   ....:
Out[72]:
   SepalLength  SepalWidth  PetalLength  PetalWidth         Name  sepal_ratio
0          5.1         3.5          1.4         0.2  Iris-setosa       0.6863
1          4.9         3.0          1.4         0.2  Iris-setosa       0.6122
2          4.7         3.2          1.3         0.2  Iris-setosa       0.6809
3          4.6         3.1          1.5         0.2  Iris-setosa       0.6739
4          5.0         3.6          1.4         0.2  Iris-setosa       0.7200
```

Pandas (5)

Institute for Big Data Analytics

- **DataFrames (TimeSeries) - DateTimeIndex**

```
In [1]: dates = pd.date_range('1/1/2000', periods=8)

In [2]: df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C', 'D'])

In [3]: df
Out[3]:
                   A         B         C         D
2000-01-01  0.469112 -0.282863 -1.509059 -1.135632
2000-01-02  1.212112 -0.173215  0.119209 -1.044236
2000-01-03 -0.861849 -2.104569 -0.494929  1.071804
2000-01-04  0.721555 -0.706771 -1.039575  0.271860
2000-01-05 -0.424972  0.567020  0.276232 -1.087401
2000-01-06 -0.673690  0.113648 -1.478427  0.524988
2000-01-07  0.404705  0.577046 -1.715002 -1.039268
2000-01-08 -0.370647 -1.157892 -1.344312  0.844885
```

Property	Description
year	The year of the datetime
month	The month of the datetime
day	The days of the datetime
hour	The hour of the datetime
minute	The minutes of the datetime
second	The seconds of the datetime
microsecond	The microseconds of the datetime
nanosecond	The nanoseconds of the datetime
date	Returns datetime.date (does not contain timezone information)
time	Returns datetime.time (does not contain timezone information)
dayofyear	The ordinal day of year
weekofyear	The week ordinal of the year
week	The week ordinal of the year
dayofweek	The number of the day of the week with Monday=0, Sunday=6
weekday	The number of the day of the week with Monday=0, Sunday=6
weekday_name	The name of the day in a week (ex: Friday)
quarter	Quarter of the date: Jan-Mar = 1, Apr-Jun = 2, etc.
days_in_month	The number of days in the month of the datetime
is_month_start	Logical indicating if first day of month (defined by frequency)
is_month_end	Logical indicating if last day of month (defined by frequency)
is_quarter_start	Logical indicating if first day of quarter (defined by frequency)
is_quarter_end	Logical indicating if last day of quarter (defined by frequency)
is_year_start	Logical indicating if first day of year (defined by frequency)
is_year_end	Logical indicating if last day of year (defined by frequency)
is_leap_year	Logical indicating if the date belongs to a leap year

Pandas (6)

Institute for Big Data Analytics

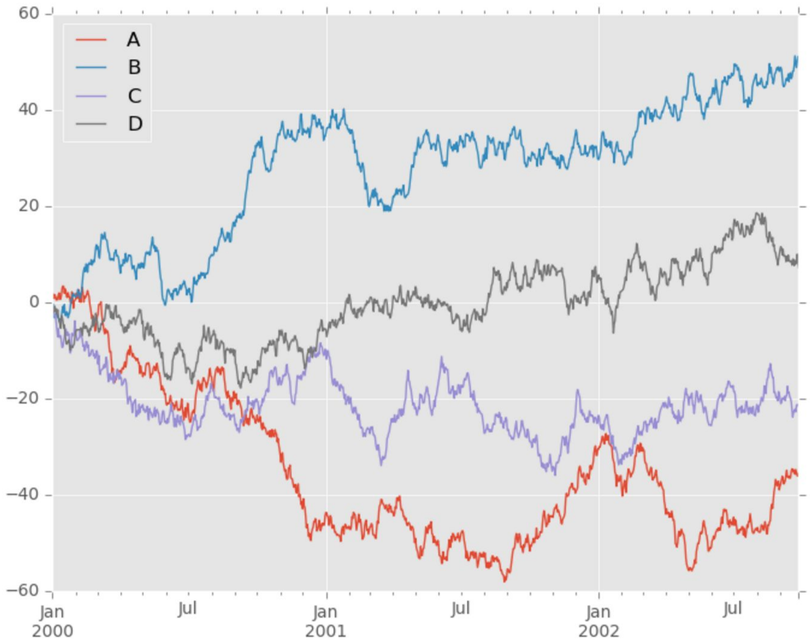
- **DataFrames (Multi-Indexing)**

```
In [10]: arrays = [np.array(['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux']),
   ....:           np.array(['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two'])]
   ....:

In [11]: s = pd.Series(np.random.randn(8), index=arrays)

In [12]: s
Out[12]:
bar  one   -0.861849
     two   -2.104569
baz  one   -0.494929
     two    1.071804
foo  one    0.721555
     two   -0.706771
qux  one   -1.039575
     two    0.271860
dtype: float64

In [13]: df = pd.DataFrame(np.random.randn(8, 4), index=arrays)

In [14]: df
Out[14]:
                0         1         2         3
bar one -0.424972  0.567020  0.276232 -1.087401
    two -0.673690  0.113648 -1.478427  0.524988
baz one  0.404705  0.577046 -1.715002 -1.039268
    two -0.370647 -1.157892 -1.344312  0.844885
foo one  1.075770 -0.109050  1.643563 -1.469388
    two  0.357021 -0.674600 -1.776904 -0.968914
qux one -1.294524  0.413738  0.276662 -0.472035
    two -0.013960 -0.362543 -0.006154 -0.923061
```

Pandas (7)

- **Data Visualization**
  - **TimeSeries**

```
In [5]: df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=list('ABCD'))

In [6]: df = df.cumsum()

In [7]: plt.figure(); df.plot();
```

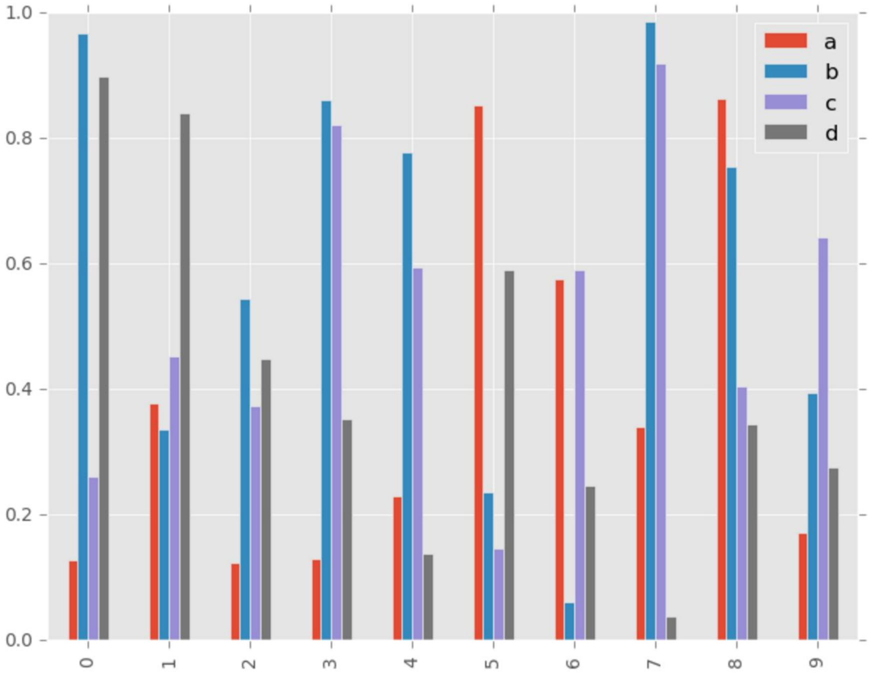


Pandas (8)

Institute for Big Data Analytics

- **Data Visualization**
  - **Bar chart**

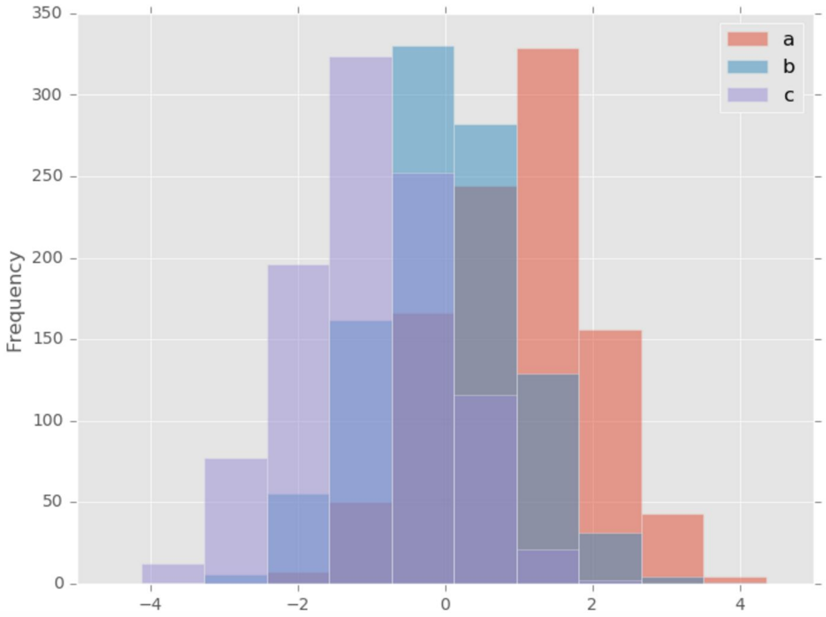
```
In [17]: df2 = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])

In [18]: df2.plot.bar();
```



Pandas (9)

- **Data Visualization**
  - **Histograms**

```
In [21]: df4 = pd.DataFrame({'a': np.random.randn(1000) + 1, 'b': np.random.randn(1000),
   ....:                     'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
   ....:

In [22]: plt.figure();

In [23]: df4.plot.hist(alpha=0.5)
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x130e51c50>
```

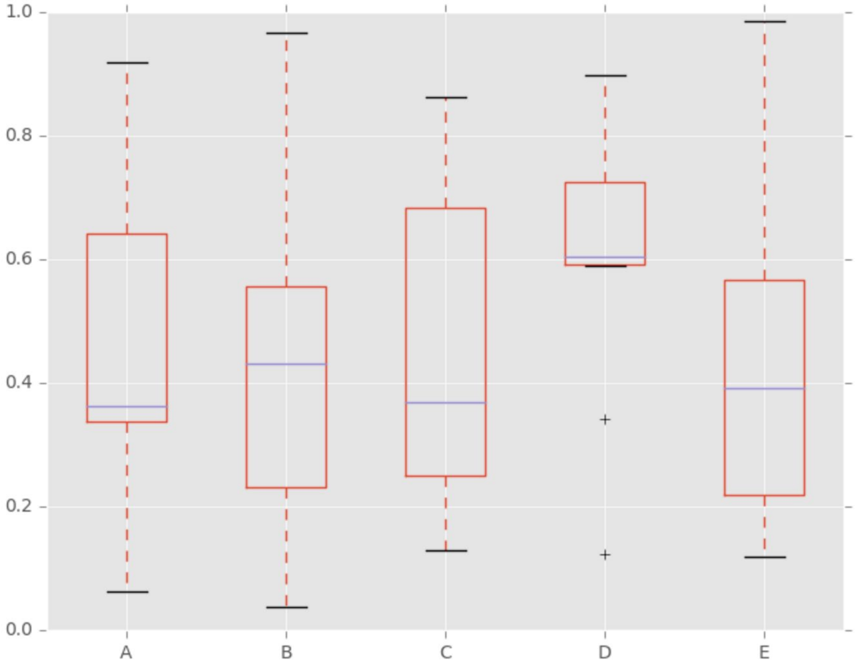


Pandas (10)

Institute for Big Data Analytics

- **Data Visualization**
  - **Boxplots**

```
In [34]: df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])

In [35]: df.plot.box()
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x13534fcf8>
```

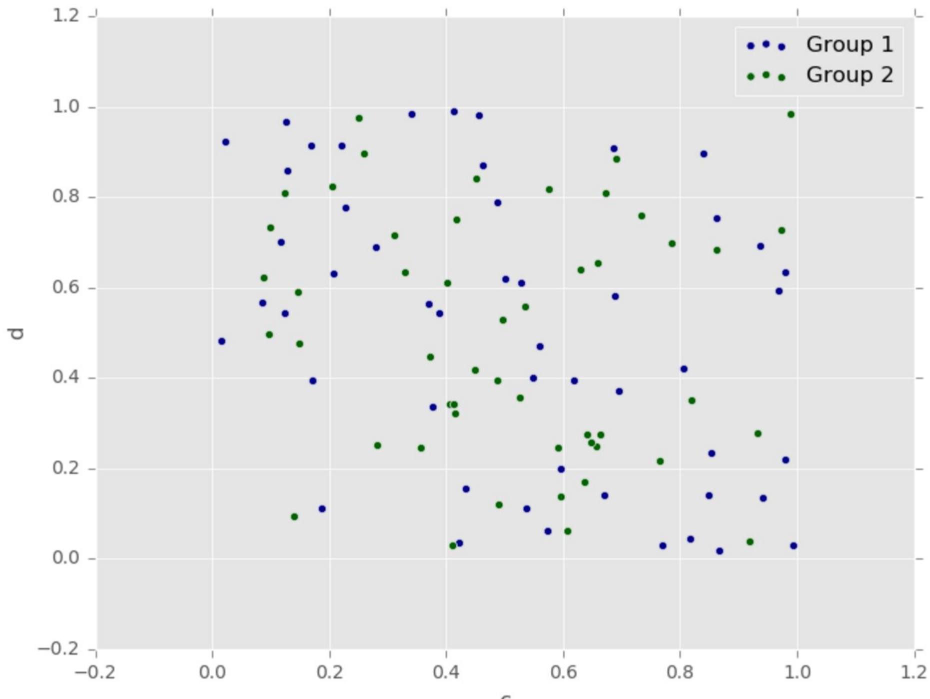


Pandas (11)

Institute for Big Data Analytics

- **Data Visualization**
  - **Scatter plots**

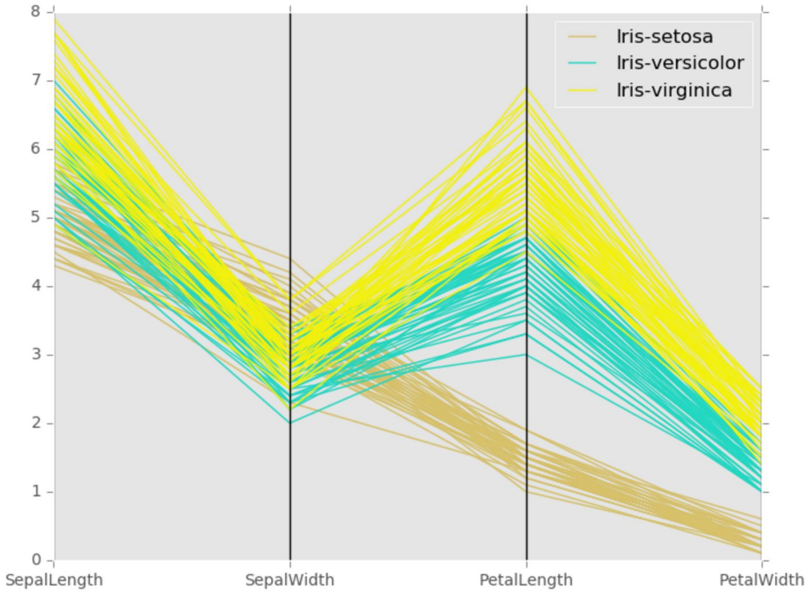
```
In [62]: ax = df.plot.scatter(x='a', y='b', color='DarkBlue', label='Group 1');

In [63]: df.plot.scatter(x='c', y='d', color='DarkGreen', label='Group 2', ax=ax);
```



Pandas (11)

- **Data Visualization**
  - **Parallel coordinates**

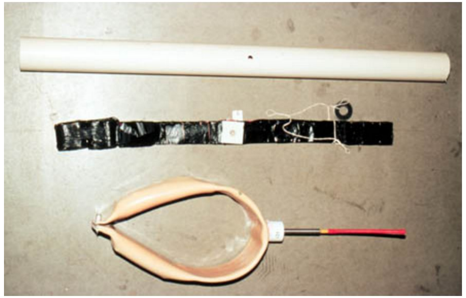
```
In [89]: from pandas.plotting import parallel_coordinates

In [90]: data = pd.read_csv('data/iris.data')

In [91]: plt.figure()
Out[91]: <matplotlib.figure.Figure at 0x13212da90>

In [92]: parallel_coordinates(data, 'Name')
Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x12da44da0>
```



The animals tracking dataset (1)

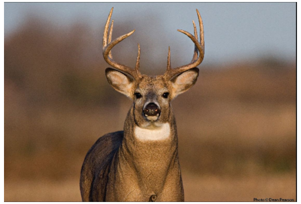
Institute for Big Data Analytics

- **The Starkey Project**



Pictures from starkey project

Cattle



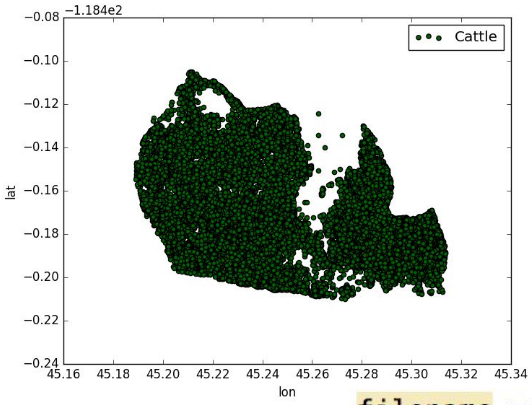
Deer

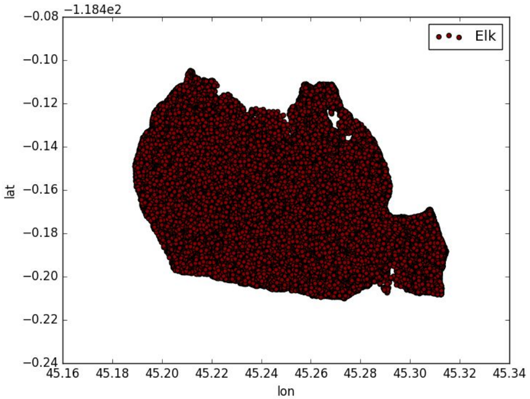
Elk

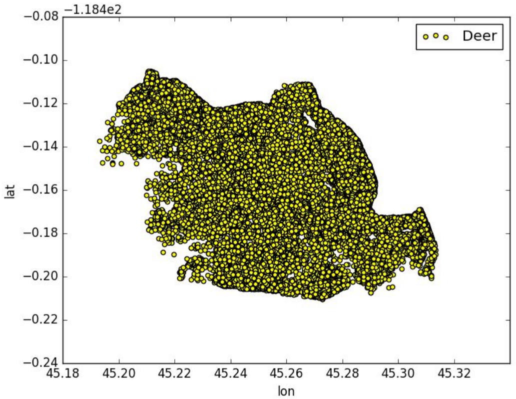
Pictures from web: Cattle - https://fm.cnbc.com/applications/cnbc.com/resources/img/editorial/2014/01/29/101374915-186501746.530x298.jpg?v=1485533269
Deer - http://gfp.sd.gov/hunting/big-game/images/deer1-01.jpg
Elk - https://elknetwork.com/wp-content/uploads/2016/09/bull_elk.jpg

The animals tracking dataset (2)

- **Plot geographical data points with pandas**
  - **Warning: This is not the proper way. The points are not projected.**



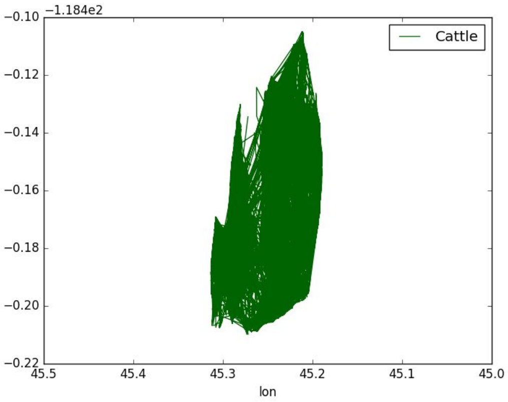


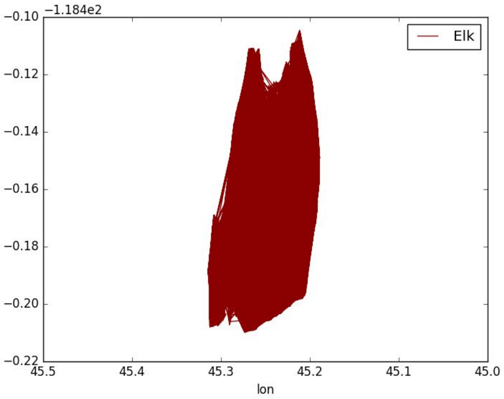


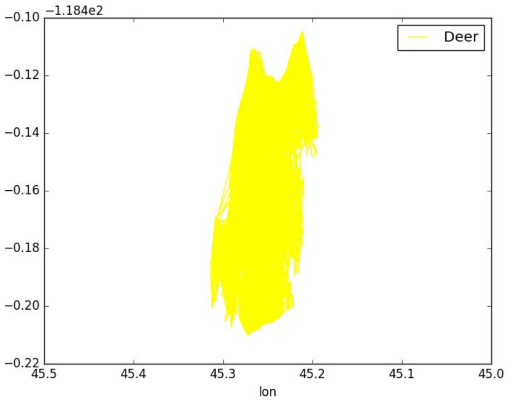

```
filename = './input/cattle_visualization.txt'
col_names = ['id', 'time', 'lon', 'lat', 'spd']
df = pd.read_csv(filename, names=col_names, sep=',', index_col=False)

df.plot.scatter(x='lon', y='lat', color='DarkGreen', label='Cattle')
plt.savefig('./images/cattle_points.png')
```

The animals tracking dataset (3)

```
x_ticks = [45.0, 45.1, 45.2, 45.3, 45.4, 45.5]

filename = './input/cattle_visualization.txt'
df = pd.read_csv(filename, names=['id', 'time', 'lon', 'lat', 'spd'],sep=',',parse_dates=[1])
df.plot.line(x='lon',y='lat', color='DarkGreen', label='Cattle', xticks=x_ticks)
plt.savefig('./images/cattle-linestring.png')

filename = './input/elk_visualization.txt'
df = pd.read_csv(filename, names=['id', 'time', 'lon', 'lat', 'spd'],sep=',',parse_dates=[1])
df.plot.line(x='lon',y='lat', color='DarkRed', label='Elk', xticks=x_ticks)
plt.savefig('./images/elk-linestring.png')

filename = './input/deer_visualization.txt'
df = pd.read_csv(filename, names=['id', 'time', 'lon', 'lat', 'spd'],sep=',',parse_dates=[1])
df.plot.line(x='lon',y='lat', color='Yellow', label='Deer', xticks=x_ticks)
plt.savefig('./images/deer-linestring.png')
```







The animals tracking dataset (4)

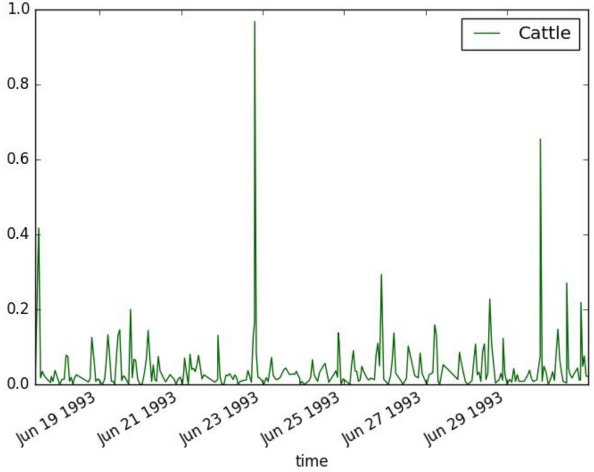
- **Computation of point features.**

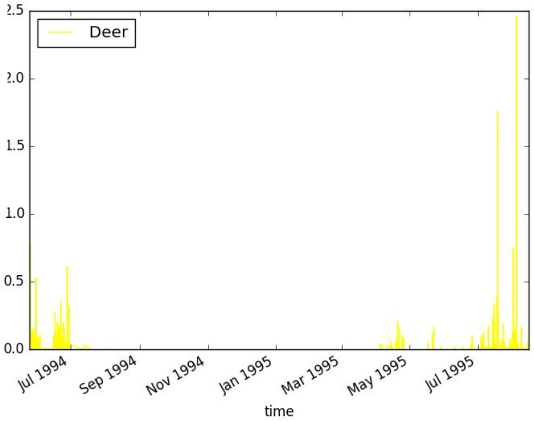
```python
# get the last index of an animal
#   this prevents calculating wrong distances across two different animals
def get_masks(df):
  ids = df.id.unique()
  print(str(len(ids))+" unique animals in total.")

  ids_list = np.array(df.id)
  mask = [0 if i==(len(ids_list)-1) or ids_list[i]!=ids_list[i+1] else 1 for i in range(len(ids_list))]
  return mask


def get_timediff(df):
  timediff = [(df.time[i+1]-df.time[i]).total_seconds() if i!=(len(df.time)-1) else 0 for i in range(len(df.time))]
  timediff = [timediff[i]*mask[i] for i in range(len(timediff))]
  df['timediff']=timediff
  return df


def get_distance(df):
  distance = [gpxpy.geo.haversine_distance(df.lat[i], df.lon[i], df.lat[i+1], df.lon[i+1]) if i!=(len(df.time)-1) else 0 for i in range(len(df.time))]
  distance = [distance[i]*mask[i] for i in range(len(distance))]
  df['distance']=distance
  return df


def get_speed(df):
  speed = [df.distance[i]/(df.timediff[i]+0.1**10) for i in range(len(df.time))]
  speed = [speed[i]*mask[i] for i in range(len(speed))]
  df['speed']= speed
  return df


def get_acc(df):
  acc = [(df.speed[i+1]-df.speed[i])/(df.timediff[i]+0.1**10) if i!=(len(df.time)-1) else 0 for i in range(len(df.time))]
  acc = [acc[i]*mask[i] for i in range(len(acc))]
  df['acc']= acc
  return df


def get_bearing(df):
  bearing = [calculate_initial_compass_bearing((df.lat[i], df.lon[i]), (df.lat[i+1], df.lon[i+1])) if i!=(len(df.time)-1) else 0 for i in range(len(df.time))]
  bearing = [bearing[i]*mask[i] for i in range(len(bearing))]
  df['bearing']=bearing
  return df
```

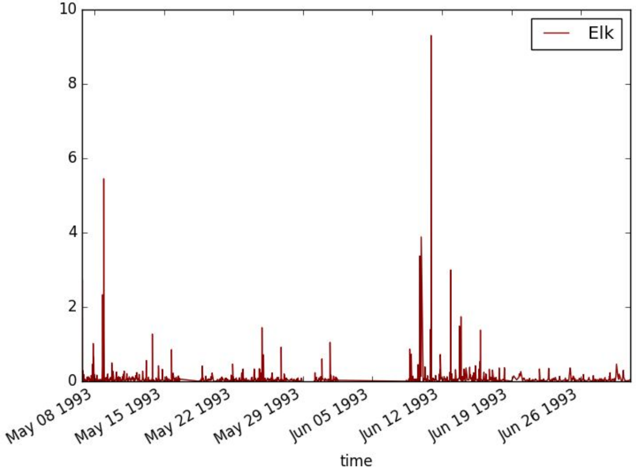
```python
data = read_file()
mask = get_masks(data)
data = get_timediff(data)
data = get_distance(data)
data = get_speed(data)
data = get_acc(data)
data = get_bearing(data)


data.drop('spd', axis=1, inplace=True)
data = data.round({'distance': 2, 'speed': 2, 'acc': 2, 'bearing': 2, 'timediff': 2})

data = data.query('speed <= 20. and acc <= 100. and acc >= -100.')

print data.head()
data.to_csv(out_filename)
```

The animals tracking dataset (5)

- **Visualizing the speed along the trajectory**







The animals tracking dataset (6)

Institute for Big Data Analytics

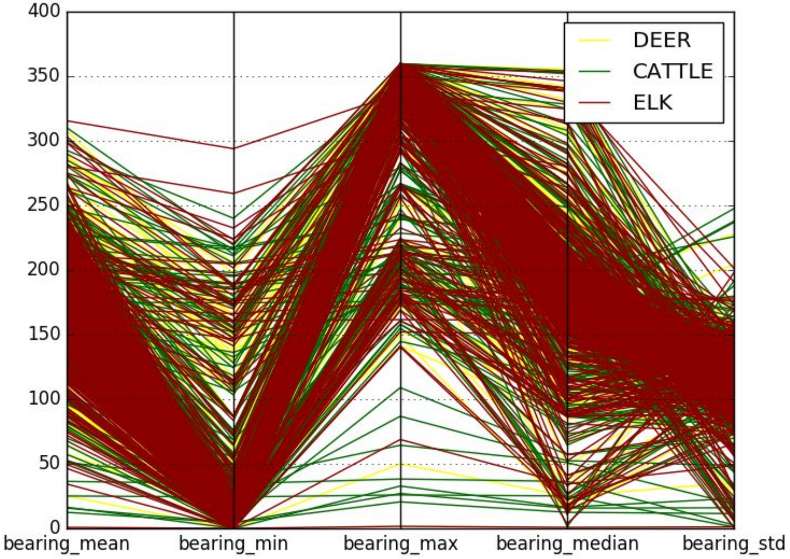
- **From points sequence to trajectories.**

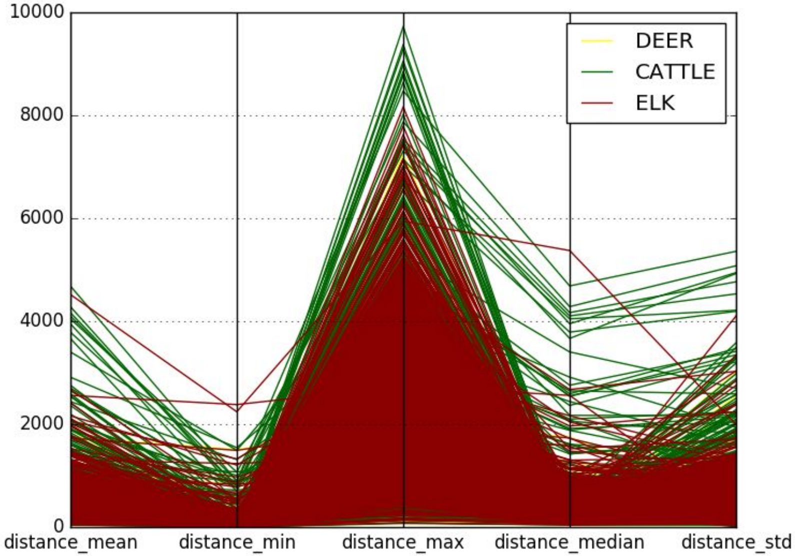
```
data = read_file(f['filename'])
all_trajs = pd.DataFrame()
out = open(f['out_filename'], 'w')

#process by trajectory id to avoid a high memory usage
for id in data.id.unique():
    t = data[data['id'] == id]
    t.time = pd.to_datetime(t.time)
    slice = t[['time', 'timediff','distance','speed','acc','bearing']].set_index('time').resample('W')

    means = slice.mean()
    means.columns = ['timediff_mean','distance_mean','speed_mean','acc_mean','bearing_mean']
    amin = slice.min()
    amin.columns = ['timediff_min', 'distance_min', 'speed_min', 'acc_min', 'bearing_min']
    amax = slice.max()
    amax.columns = ['timediff_max', 'distance_max', 'speed_max', 'acc_max', 'bearing_max']
    amedian = slice.median()
    amedian.columns = ['timediff_median', 'distance_median', 'speed_median', 'acc_median', 'bearing_median']
    astd = slice.std()
    astd.columns = ['timediff_std', 'distance_std', 'speed_std', 'acc_std', 'bearing_std']

    subtrajs = pd.concat([means, amin], axis=1)
    subtrajs = pd.concat([subtrajs, amax], axis=1)
    subtrajs = pd.concat([subtrajs, amedian], axis=1)
    subtrajs = pd.concat([subtrajs, astd], axis=1)
    subtrajs['class'] = f['cls']

    all_trajs = all_trajs.append(subtrajs)


all_trajs = all_trajs.dropna(axis=0, how='any')
print all_trajs.head()
all_trajs.to_csv(f['out_filename'], index=False)

all_trajs.to_csv('./output/animals.csv', header=False, index=False, mode='a')
```

The animals tracking dataset (7)

Institute for Big Data Analytics

- **Plotting some parallels coordinates**





Scikit-learn (1)

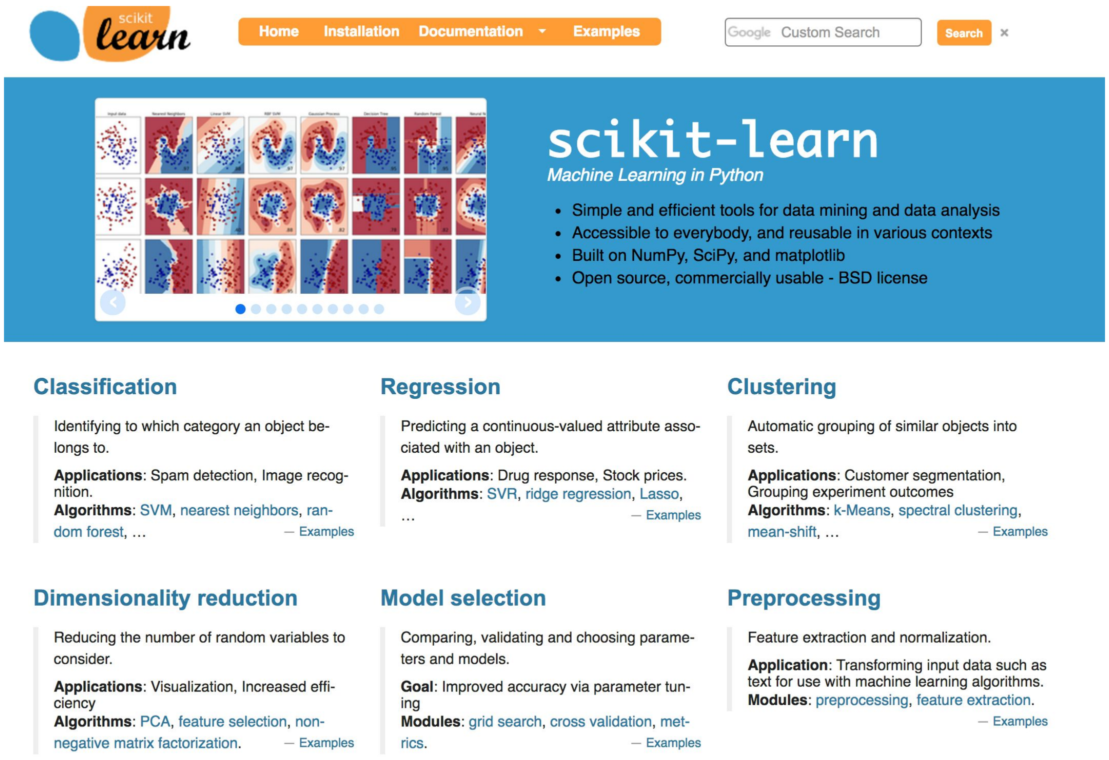
Institute for Big Data Analytics

- **Machine learning for applications**
  - Ease of use
  - Light and easy to install package
- **State-of-the-art algorithms**
- **High quality bindings: performance and fine control**
- **Open Source**
  - BSD license

Scikit-learn (2)

Institute for Big Data Analytics

- **Scikit-learn.org**



Scikit-learn (3)

Institute for Big Data Analytics

- **All objects**
  - estimator.fit(X_train, y_train)
- **Classification, regression, clustering**
  - y_pred = estimator.predict(X_test)
- **Filters, dimension reduction, latent variables**
  - X_new = estimator.transform(X_test)
- **Predictive models, density estimation**
  - test_score = estimator.score(X_test)
- **On-line learning**
  - estimator.refit(X_train, y_train)

Scikit-learn (4)

- **Pre-processing (reading csv)**

```
data = pd.read_csv(filename, names=col_names, sep=',', index_col=False)
data = data.values

train = np.array(data[:,:25])
# print 'Train', train

target = np.array(data[ :, -1:])
# print 'Target', target
```

- **Binarize the classes**

```
# transform targets

target_a = [0 if target[i] == 'A' else 1 for i in range(len(target))]
target_b = [0 if target[i] == 'B' else 1 for i in range(len(target))]
target_c = [0 if target[i] == 'C' else 1 for i in range(len(target))]
```

Scikit-learn (5)

- **Predict and score**

```
from sklearn.tree.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(train, true_labels, random_state=0)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print confusion_matrix(y_test, y_pred)
print accuracy_score(y_test, y_pred)
```

References

- Pandas
  - https://pandas.pydata.org/pandas-docs/stable/index.html
- Trajectory Dataset
  - https://www.fs.fed.us/pnw/starkey/
- Scikit-learn
  - http://scikit-learn.org/stable/