csvParser

December 15, 2015

In [1]:
```
import pandas
from sqlalchemy import create_engine
import json
```

In [2]:
```
class csvParser:

    data = pandas.DataFrame()
    config = {}
    type_map = {
            'float':float,
            'int':int,
            'str':str
    }

    def __init__(self, confPath, auto=True):
        self.op_map = {
            "remove":{"func":self.removeCols,"args":1},
            "hash":{"func":self.hashCols,"args":1},
            "cast":{"func":self.castCols,"args":1},
            "sample":{"func":self.sampleRows, "args":1}
        }

        self.loadConfig(confPath)
        if (auto==True):
            self.loadData()
            self.processData()

    def loadConfig(self,path):
        try:
            with open(path, 'r') as config:
                data=config.read().replace('\n', '')
            self.config = json.loads(data)
            return True
        except:
            return False

    def loadData(self):
        #TODO: dtypes and select cols
        try:
            filepath = self.config['filepath']
            print("loading data from "+filepath)
            col_dtypes = self.config['load_cols']
            cols = list(col_dtypes.keys())
            self.data = pandas.read_csv(filepath, usecols = cols, dtype = col_dtypes)
```

```
            self.data.columns = list(map(lambda x: x.strip(), self.data.columns))
            return True
        except:
            return False

    def processData(self):
        for step in self.config['processing_steps']:
            op = list(step.keys())[0]
            params = list(step.values())[0]
            self.processDataStep(op,params)

    def processDataStep(self,op,params):
        if (type(params)!=list):
            params = [params]
        f = self.op_map[op]['func']
        arg_num = self.op_map[op]['args']
        if (arg_num==1):
            f(params)
        elif (arg_num==2):
            ps = params.items()
            f(ps[0], ps[1])

    def removeCols(self, names):
        for name in names:
            try:
                print("removing col="+name)
                del self.data[name]
            except:
                print("failed to del" + name)
                return False
        return True

    def castCols(self, type_castings):
        for cast in type_castings:
            for col in cast.keys():
                print("casting col="+col+" to type="+cast[col])
                self.data[col] = self.data[col].astype(self.type_map[cast[col]])
        return True

    def hashCols(self, names):
        #converts col contents to string before hashing
        for name in names:
            print("hashing col="+name)
            self.data[name] = self.data[name].apply(lambda x: hash(str(x)))
        return True

    def sampleRows(self,perc):
        print("sampling "+str(perc[0]*100)+"% of data")
        self.data = self.data.sample(frac=perc[0])
        return True

    def storeData(self):
        dbname = self.config['sql']['db']
        tablename = self.config['sql']['table']
```

```
            print("storing data in table="+tablename+" in db="+dbname)
            cols = list(self.data.columns)
            engine = create_engine(dbname)
            self.data.to_sql(tablename, engine)
            return True
```

```
In [3]: testParser = csvParser("./csvParse.config", auto=False)

In [4]: testParser.config

Out[4]: {'filepath': './testData.csv',
         'load_cols': {'col1': 'float', 'col2': 'int', 'col3': 'int', 'col4': 'str'},
         'processing_steps': [{'remove': ['col1']},
          {'hash': ['col2', 'col4']},
          {'cast': {'col3': 'float'}},
          {'sample': 0.5}],
         'sql': {'db': 'sqlite:////data/quartet/testParse.db', 'table': 'testCsv'}}

In [5]: testParser.loadData()
        testParser.data

loading data from ./testData.csv

Out[5]:    col1  col2  col3 col4
        0     1     2     3    a
        1     4     5     6    b

In [6]: testParser.processData()
        testParser.data

removing col=col1
hashing col=col2
hashing col=col4
casting col=col3 to type=float
sampling 50.0% of data

Out[6]:                   col2  col3                 col4
        1 -2480217190145936623     6  6106447415896647459

In [7]: testParser.data['col3'].dtype

Out[7]: dtype('float64')

In [8]: testParser.storeData()

storing data in table=testCsv in db=sqlite:////data/quartet/testParse.db

Out[8]: True
```

1 FUTURE:

- Currently headers aren't stripped of whitespace before being compared to config, it would be nice to fix this
- More datatypes (ex datetime)
- Seperate out loading, processing, and storing into seperate classes for better modularity

- More input types (ex load from db)
- More output types (ex csv)
- More transform types (ex round)
- Save transformed cols as new cols instead of overwriting (ex col3_float)
- Better error handling