django-pandas Documentation

Release 0.0.2

Christopher Clarke

Jan 08, 2019

Contents

build passing coverage 90% Tools for working with pandas in your Django projects

CHAPTER 1

Contributors

- Christopher Clarke
- Bertrand Bordage
- Guillaume Thomas
- Parbhat Puri
- Fredrik Burman (coachHIPPO)
- Safe Hammad
- Jeff Sternber
- @MiddleFork
- Daniel Andrlik
- Kevin Abbot
- Yousuf Jawwad
- @henhuy
- Hélio Meira Lins
- @utpyngo
- Anthony Monthe

CHAPTER 2

What's New

- Improved corece_float option
- While we still support legacy versions (Python 2.7 and Django < 1.8). But you need to install your preferred version of Django. Recall that Django 2 does not support Python 2.
- Test now pass on with Django 2+ and python 3.7

CHAPTER 3

Dependencies

`django-pandas` supports Django (>=1.4.5) or later and requires django-model-utils (>= 1.4.0) and Pandas (>= 0.12.0). **Note** because of problems with the `requires` directive of setuptools you probably need to install `numpy` in your virtualenv before you install this package or if you want to run the test suite

```
pip install numpy
python setup.py test
```

Some `pandas` functionality requires parts of the Scipy stack. You may wish to consult http://www.scipy.org/install.html for more information on installing the `Scipy` stack.

You need to install your preferred version of Django. as that Django 2 does not support Python 2.

CHAPTER 4

Contributing

Please file bugs and send pull requests to the GitHub repository and issue tracker.

CHAPTER 5

Installation

Start by creating a new `virtualenv` for your project

```
mkvirtualenv myproject
```

Next install `numpy` and `pandas` and optionally `scipy`

```
pip install numpy
pip install pandas
```

You may want to consult the scipy documentation for more information on installing the `Scipy` stack.

Finally, install `django-pandas` using `pip`:

```
pip install django-pandas
```

or install the development version from `github`

```
pip install https://github.com/chrisdev/django-pandas/tarball/master
```

CHAPTER 6

Usage

6.1 IO Module

The `django-pandas.io` module provides some convenience methods to facilitate the creation of DataFrames from Django QuerySets.

6.1.1 read_frame

Parameters

- qs: A Django QuerySet.
- **fieldnames: A list of model field names to use in creating the `DataFrame`.** You can span a relationship in the usual Django way by using double underscores to specify a related field in another model
- **index_col: Use specify the field name to use for the `DataFrame` index.** If the index field is not in the field list it will be appended
- **coerce_float** [Boolean, defaults to True] Attempt to convert values to non-string, non-numeric objects (like decimal.Decimal) to floating point.
- **verbose: If this is `True` then populate the DataFrame with the** human readable versions of any foreign key or choice fields else use the actual values set in the model.

6.1.2 Examples

Assume that this is your model:

```
class MyModel(models.Model):

    full_name = models.CharField(max_length=25)
    age = models.IntegerField()
```

(continues on next page)

(continued from previous page)

```
    department = models.CharField(max_length=3)
    wage = models.FloatField()
```

First create a query set:

```
from django_pandas.io import read_frame
qs = MyModel.objects.all()
```

To create a dataframe using all the fields in the underlying model

```
df = read_frame(qs)
```

The *df* will contain human readable column values for foreign key and choice fields. The *DataFrame* will include all the fields in the underlying model including the primary key. To create a DataFrame using specified field names:

```
df = read_frame(qs, fieldnames=['age', 'wage', 'full_name'])
```

To set `full_name` as the `DataFrame` index

```
qs.to_dataframe(['age', 'wage'], index='full_name'])
```

You can use filters and excludes

```
qs.filter(age__gt=20, department='IT').to_dataframe(index='full_name')
```

6.2 DataFrameManager

`django-pandas` provides a custom manager to use with models that you want to render as Pandas Dataframes. The `DataFrameManager` manager provides the `to_dataframe` method that returns your models queryset as a Pandas DataFrame. To use the DataFrameManager, first override the default manager (*objects*) in your model's definition as shown in the example below

```
#models.py

from django_pandas.managers import DataFrameManager

class MyModel(models.Model):

    full_name = models.CharField(max_length=25)
    age = models.IntegerField()
    department = models.CharField(max_length=3)
    wage = models.FloatField()

    objects = DataFrameManager()
```

This will give you access to the following QuerySet methods:

- `to_dataframe`
- `to_timeseries`
- `to_pivot_table`

6.2.1 to_dataframe

Returns a DataFrame from the QuerySet

Parameters

- **fieldnames: The model field names to utilise in creating the frame.** to span a relationship, use the field name of related fields across models, separated by double underscores,
- **index: specify the field to use for the index. If the index** field is not in the field list it will be appended
- **coerce_float: Attempt to convert the numeric non-string data** like object, decimal etc. to float if possible
- **verbose: If this is `True` then populate the DataFrame with the** human readable versions of any foreign key or choice fields else use the actual value set in the model.

6.2.2 Examples

Create a dataframe using all the fields in your model as follows

```
qs = MyModel.objects.all()

df = qs.to_dataframe()
```

This will include your primary key. To create a DataFrame using specified field names:

```
df = qs.to_dataframe(fieldnames=['age', 'department', 'wage'])
```

To set `full_name` as the index

```
qs.to_dataframe(['age', 'department', 'wage'], index='full_name'])
```

You can use filters and excludes

```
qs.filter(age__gt=20, department='IT').to_dataframe(index='full_name')
```

6.3 to_timeseries

A convenience method for creating a time series i.e the DataFrame index is instance of a DateTime or PeriodIndex

Parameters

- **fieldnames: The model field names to utilise in creating the frame.** to span a relationship, just use the field name of related fields across models, separated by double underscores,
- **index: specify the field to use for the index. If the index** field is not in the field list it will be appended. This is mandatory.
- **storage: Specify if the queryset uses the *wide* or *long* format** for data.
- **pivot_columns: Required once the you specify *long* format** storage. This could either be a list or string identifying the field name or combination of field. If the pivot_column is a single column then the unique values in this column become a new columns in the DataFrame If the pivot column is a list the values in these columns are concatenated (using the '-' as a separator) and these values are used for the new timeseries columns
- **values: Also required if you utilize the *long* storage the** values column name is use for populating new frame values

- freq: the offset string or object representing a target conversion
- rs_kwargs: Arguments based on pandas.DataFrame.resample
- **verbose: If this is `True` then populate the DataFrame with the** human readable versions of any foreign key or choice fields else use the actual value set in the model.

6.3.1 Examples

Using a *long* storage format

```
#models.py

class LongTimeSeries(models.Model):
    date_ix = models.DateTimeField()
    series_name = models.CharField(max_length=100)
    value = models.FloatField()

    objects = DataFrameManager()
```

Some sample data::

```
========   =====       =====
date_ix    series_name value
========   =====       ======
2010-01-01  gdp        204699

2010-01-01  inflation  2.0

2010-01-01  wages      100.7

2010-02-01  gdp        204704

2010-02-01  inflation  2.4

2010-03-01  wages      100.4

2010-02-01  gdp        205966

2010-02-01  inflation  2.5

2010-03-01  wages      100.5
==========  ========== ======
```

Create a QuerySet

```
qs = LongTimeSeries.objects.filter(date_ix__year__gte=2010)
```

Create a timeseries dataframe

```
df = qs.to_timeseries(index='date_ix',
                      pivot_columns='series_name',
                      values='value',
                      storage='long')
df.head()

date_ix      gdp     inflation     wages
```

(continues on next page)

(continued from previous page)

```
2010-01-01   204966     2.0       100.7

2010-02-01   204704      2.4       100.4

2010-03-01   205966      2.5       100.5
```

Using a *wide* storage format

```
class WideTimeSeries(models.Model):
    date_ix = models.DateTimeField()
    col1 = models.FloatField()
    col2 = models.FloatField()
    col3 = models.FloatField()
    col4 = models.FloatField()

    objects = DataFrameManager()

qs = WideTimeSeries.objects.all()

rs_kwargs = {'how': 'sum', 'kind': 'period'}
df = qs.to_timeseries(index='date_ix', pivot_columns='series_name',
                      values='value', storage='long',
                      freq='M', rs_kwargs=rs_kwargs)
```

6.4 to_pivot_table

A convenience method for creating a pivot table from a QuerySet

Parameters

- **fieldnames: The model field names to utilise in creating the frame.** to span a relationship, just use the field name of related fields across models, separated by double underscores,
- values : column to aggregate, optional
- **rows** [list of column names or arrays to group on] Keys to group on the x-axis of the pivot table
- **cols** [list of column names or arrays to group on] Keys to group on the y-axis of the pivot table
- **aggfunc** [function, default numpy.mean, or list of functions] If list of functions passed, the resulting pivot table will have hierarchical columns whose top level are the function names (inferred from the function objects themselves)
- **fill_value** [scalar, default None] Value to replace missing values with
- **margins** [boolean, default False] Add all row / columns (e.g. for subtotal / grand totals)
- dropna : boolean, default True

Example

```
# models.py
class PivotData(models.Model):
    row_col_a = models.CharField(max_length=15)
    row_col_b = models.CharField(max_length=15)
    row_col_c = models.CharField(max_length=15)
```

(continues on next page)

(continued from previous page)

```
    value_col_d = models.FloatField()
    value_col_e = models.FloatField()
    value_col_f = models.FloatField()

    objects = DataFrameManager()
```

Usage

```
rows = ['row_col_a', 'row_col_b']
cols = ['row_col_c']

pt = qs.to_pivot_table(values='value_col_d', rows=rows, cols=cols)
```