



NODE.JS AND R IN PYTHON

ANAND S, PYCONF HYDERABAD, 8 DEC 2019

@sanand0 

OUR ANALYTICS TEAM LIKES USING R

LINK

Commodity Price Forecasting

Gramener
A DATA SERVICE COMPANY



INVENTORY RECD.
Hold for next 3 weeks



FORECAST
Price to increase after 2 weeks

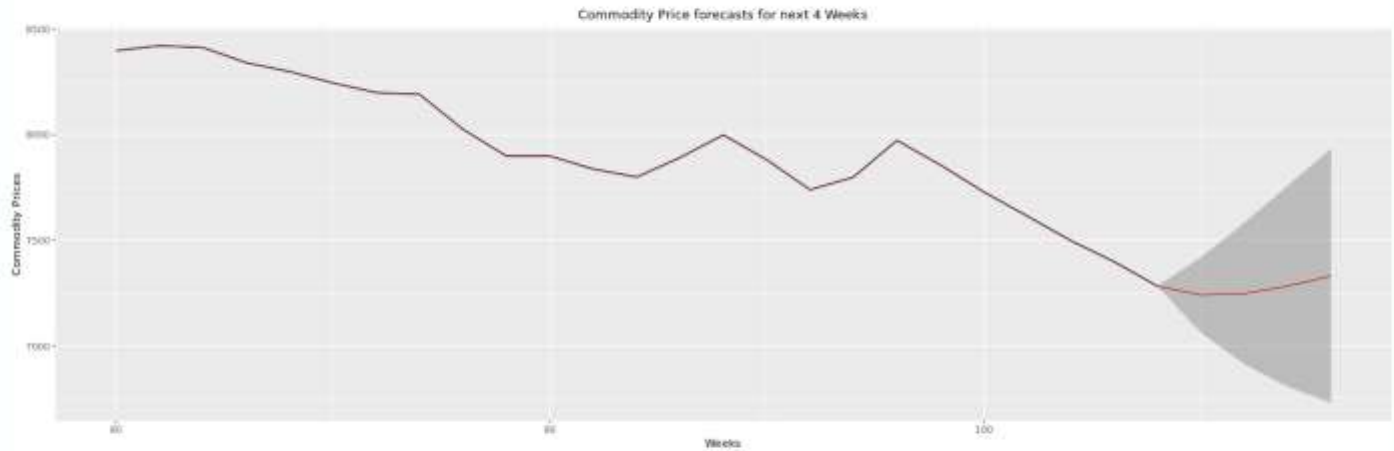


PRICE CHANGE
-0.5% from last week



ESTIMATED SAVINGS
\$ 341,080

Forecast plot (* Constructed using fictitious data)



Select the model
VECM

VECM is a multi-variate time series model

Average Weekly Sales(In tonnes)

Slider: 5,000 (selected), 12,000

Forecast Horizon

Slider: 1 (selected), 25

Confidence Interval(Percentage)

Slider: 80 (selected), 95

Extra total Profits made from not selling this week: 341,080 \$

Model Performance in last 5 weeks (* Constructed using fictitious data)

| Week | Actual Price | Predicted Price | Predicted Low | Predicted High | Prediction within limits |
|------------------------------|--------------|-----------------|---------------|----------------|--------------------------|
| 1 Week Ending 04th Feb, 2018 | 7284 | 7284 | 7284 | 7284 | YES |
| 2 Week Ending 20th Jan, 2018 | 7420 | 7446 | 7304 | 7785 | YES |
| 3 Week Ending 21st Jan, 2018 | 7576 | 7576 | 7576 | 7576 | YES |
| 4 Week Ending 14th Jan, 2018 | 7616 | 7667 | 7480 | 7855 | YES |
| 5 Week Ending 7th Jan, 2018 | 7730 | 7255 | 6809 | 7605 | NO |

Select the model
VECM

VECM, stands for Vector Error Correction Model, is a multi-variate model that uses predictor variables to forecast the future values of the time series. The current model uses 4 variables to predict the prices of present commodities. The 4 variables are downstream prices, exchange rates, and other factors.

BUT PRODUCTIONISING R IS HARD
SSO, MICRO-SERVICES, DISTRIBUTED COMPUTING

Customize and download

Customize Bootstrap's components, Less variables, and jQuery plugins to get your very own version. Requires IE9+ or latest Safari, Chrome, or Firefox.



Less variables

[Reset to defaults](#)

Customize Less variables to define colors, sizes and more inside your custom CSS stylesheets.

BUT PYTHON CAN'T HANDLE HTML/CSS/JS AS WELL AS NODE.JS

Colors

Gray and brand colors for use across Bootstrap.

@gray-base

```
#000
```

@gray-darker

```
lighten(@gray-base, 13.5%)
```

@gray-dark

```
lighten(@gray-base, 20%)
```

@gray

```
lighten(@gray-base, 33.5%)
```

@gray-light

```
lighten(@gray-base, 46.7%)
```

@gray-lighter

```
lighten(@gray-base, 93.5%)
```



SHOULD PEOPLE SWITCH LANGUAGES?

I DON'T THINK SO

[Index of topics](#)[About](#)[Gramex](#)[Quickstart](#)[Tutorials](#)[FAQ](#)[API Reference](#)[Release notes](#)[License](#)[Contributing](#)[Usage](#)[Install Gramex](#)[Gramex init](#)[Configurations](#)[Run apps](#)[Deployment patterns](#)[Debugging Gramex](#)[Testing Gramex](#)[Snippets](#)[Handlers](#)[FileHandler](#)

R

- [R commands](#)
- [R scripts](#)
- [R arguments](#)
- [R packages](#)
- [R plots](#)
- [R async](#)
- [RMarkdown](#)

To run R in Gramex, install `rpy2` first:

```
conda install rpy2 # Do not use pip. Gramex assumes you use conda
```

This installs a new R to the Anaconda PATH, ignoring the system R.

Caution: You'll have 2 `Rs` in your system – the Anaconda R and the system R. Running `R` from the command line will run whichever is first in your PATH. Installing a package in one **does not** install a package in the other.

Here is an example of a prime number calculation in R:

- `prime`: calculate prime numbers up to 100
- `prime?n=1000`: calculate prime numbers up to 1,000

See the [Python source](#) and the [R script](#).

R commands

Call `gramex.ml.r('R expression')` to run an R command and return its result.

```
import gramex.ml
```

AS A SINGLE INSTANCE WITH CODE, SCRIPT AND PACKAGE SUPPORT

```
129 def r(code=None, path=None, rel=True, conda=True, convert=True,
130        repo='https://cran.microsoft.com/', **kwargs):
131     '''
132     Runs the R script and returns the result.
133
134     :arg str code: R code to execute.
135     :arg str path: R script path. Cannot be used if code is specified
136     :arg bool rel: True treats path as relative to the caller function's file
137     :arg bool conda: True overrides R_HOME to use the Conda R
138     :arg bool convert: True converts R objects to Pandas and vice versa
139     :arg str repo: CRAN repo URL
140
141     All other keyword arguments as passed as parameters
142     '''
143     # Use Conda R if possible
144     if conda:
145         r_home = _conda_r_home()
146         if r_home:
147             os.environ['R_HOME'] = r_home
```

THIS HAS THE BENEFITS OF BOTH R & PYTHON

R for Statistical Analysis

R for exploratory analysis

R for reproducing scientific papers

Python for Deep Learning

Python for deploying analytics

Python for solving problems roughly

But the biggest benefit is...

If someone has created an R script, integrate it.

Don't argue with them about changing languages.

Other libraries
rpy2, Pyper, pyRserve



[Index of topics](#)[About](#)[Gramex](#)[Quickstart](#)[Tutorials](#)[FAQ](#)[API Reference](#)[Release notes](#)[License](#)[Contributing](#)[Usage](#)[Install Gramex](#)[Gramex init](#)[Configurations](#)[Run apps](#)[Deployment patterns](#)[Debugging Gramex](#)[Testing Gramex](#)[Snippets](#)[Handlers](#)[FileHandler](#)

Run JavaScript

Gramex can run JavaScript code via `node.js` using `gramex.pynode.node`.

- [Run JavaScript](#)
- [JavaScript conversion](#)

Run JavaScript

Gramex FunctionHandlers can run JavaScript code in node.js. Here is a simple example:

```
from gramex.pynode import node
from tornado.gen import coroutine, Return

@coroutine
def total(handler):
    result = yield node.js('return Math.abs(x + y)', x=5, y=-10) # Run code in JS
    raise Return(result)
```

This returns the result:

```
{"error":null,"result":5}
```

[Run total in JS](#)[Source](#)

JavaScript conversion

Here is how `node.js()` works:

- **Run any JS code.** You can pass any JavaScript code to `node.js()`. Whatever the code returns is the result. For


```
37 @coroutine
38 def js(self, code=None, path=None, **kwargs):
39     if self.conn is None:
40         try:
41             self.conn = yield websocket_connect(self.url, connect_timeout=self.timeout)
42         except OSError as exc:
43             import errno
44             if exc.errno != errno.ECONNREFUSED:
45                 raise
46             # TODO: node_path
47             self.proc = yield daemon(
48                 [which('node'), self._path, '--port=%s' % self.port],
49                 first_line=re.compile(r'pynode: 1.\d+.\d+ port: %s' % self.port),
50                 cwd=self.cwd,
51             )
52             self.conn = yield websocket_connect(self.url, connect_timeout=self.timeout)
53
54     # code= takes preference over path=
55     if code is not None:
```

Gramex UI Components

Introduction

Quick start

[Custom Bootstrap](#)[Custom Fonts](#)[Custom SASS](#)[Libraries](#)

Utilities

[Colors](#)[Hover styles](#)[Focus styles](#)[Active styles](#)[Cursor](#)[Round](#)[Ripples](#)[Border](#)[Shadows](#)[Text size](#)[Text alignment](#)[Letter spacing](#)[Line height](#)[Text shadow](#)[Z Index](#)[Tours](#)[Underline](#)

Backgrounds

[Background](#)[Full height](#)[Gradient](#)

Custom Bootstrap


[ui/bootstraptheme.css](#) is a customized version of Bootstrap with additional features.

You can customize Bootstrap by modifying [variable](#) from the URL. For example: [ui/bootstraptheme.css?primary=maroon](#) creates a maroon primary color.

See the [list of Bootstrap variables](#) you can change. Here are some examples:

- [?body-bg=yellow](#)
- [?link-color=black](#)
- [?paragraph-margin-bottom=5rem](#)
- [Reset theme](#)

In addition, any variable beginning with [color](#) is added as a theme color. For example, initially **these buttons have no color:**

 `.btn-color1``.btn-color-abc`

But click on this link: [?color1=purple&color-abc=teal](#) to add a new [color1](#) and [color-abc](#) to the theme. All color methods like [?color1=teal](#) will work.

You can change some common variables from the [Change Theme](#) button on at the top of this page to colors, fonts and style. The [Change Theme](#) button is located at the top of the page.

- [?enable-rounded=false](#) (default: true)
- [?enable-shadows=true](#) (default: false)
- [?enable-gradients=true](#) (default: false)
- [?enable-transitions=false](#) (default: true)
- [?enable-responsive-font-sizes=true](#) (default: false)
- [?enable-print-styles=false](#) (default: true)

There are also additional components and utilities in the library. See the sidebar for a full list.

Index of topics

About

Gramex

Quickstart

Tutorials

FAQ

API Reference

Release notes

License

Contributing

Usage

Install Gramex

Gramex init

Configurations

Run apps

Deployment patterns

Debugging Gramex

CaptureHandler takes screenshots

CaptureHandler takes screenshots of pages using either [Chrome](#) or [PhantomJS](#).

- [Chrome](#)
- [PhantomJS](#)
- [Screenshot service](#)
- [Screenshot library](#)

Chrome

Chrome is the recommended engine from v1.23. To set it up:

- Install [Node 8.x](#) – earlier versions won't work. Ensure that `node` is in your PATH.
- Uninstall and re-install [Gramex](#) (or run `gramex install capture` instead.)

Add this to `gramex.yaml`:

```
url:
  capture:
    pattern: /$YAMLURL/capture
    handler: CaptureHandler
    kwargs:
      engine: chrome
```

When Gramex runs, it starts `node chromecapture.js --port 9900` running a node.js based web application (`chromecapture.js`) at port 9900.

THIS HAS THE BENEFITS OF BOTH JS & PYTHON

JS for UI

JS for front-end interactivity

Python for data

Python for back-end processing

Another strong benefit is...

**If someone has built it in JS,
you can integrate it.**

You don't need to worry about
re-creating it.

Other libraries

V8Py, Naked, PyExecJS



I BELIEVE WE SHOULD...

**USE AVAILABLE SKILLS & LIBRARIES
INTEGRATE, DON'T RE-WRITE**

EXPLORE AT [GITHUB.COM / GRAMENER/GRAMEX](https://github.com/gramener/gramex)

IDEATE AS A GROUP. TWEET: [#RPY](#) / [#JSPY](#)

TALK TO ANYONE FROM [GRAMENER](#)

[S.ANAND @ GRAMENER.COM](#)