Time Series Forecast Study with Python: Monthly Sales of French Champagne

by **Jason Brownlee** on February 20, 2017 in **Time Series**

Last Updated on February 11, 2020

Time series forecasting is a process, and the only way to get good forecasts is to practice this process.

In this tutorial, you will discover how to forecast the monthly sales of French champagne with Python.

Working through this tutorial will provide you with a framework for the steps and the tools for working through your own time series forecasting problems.

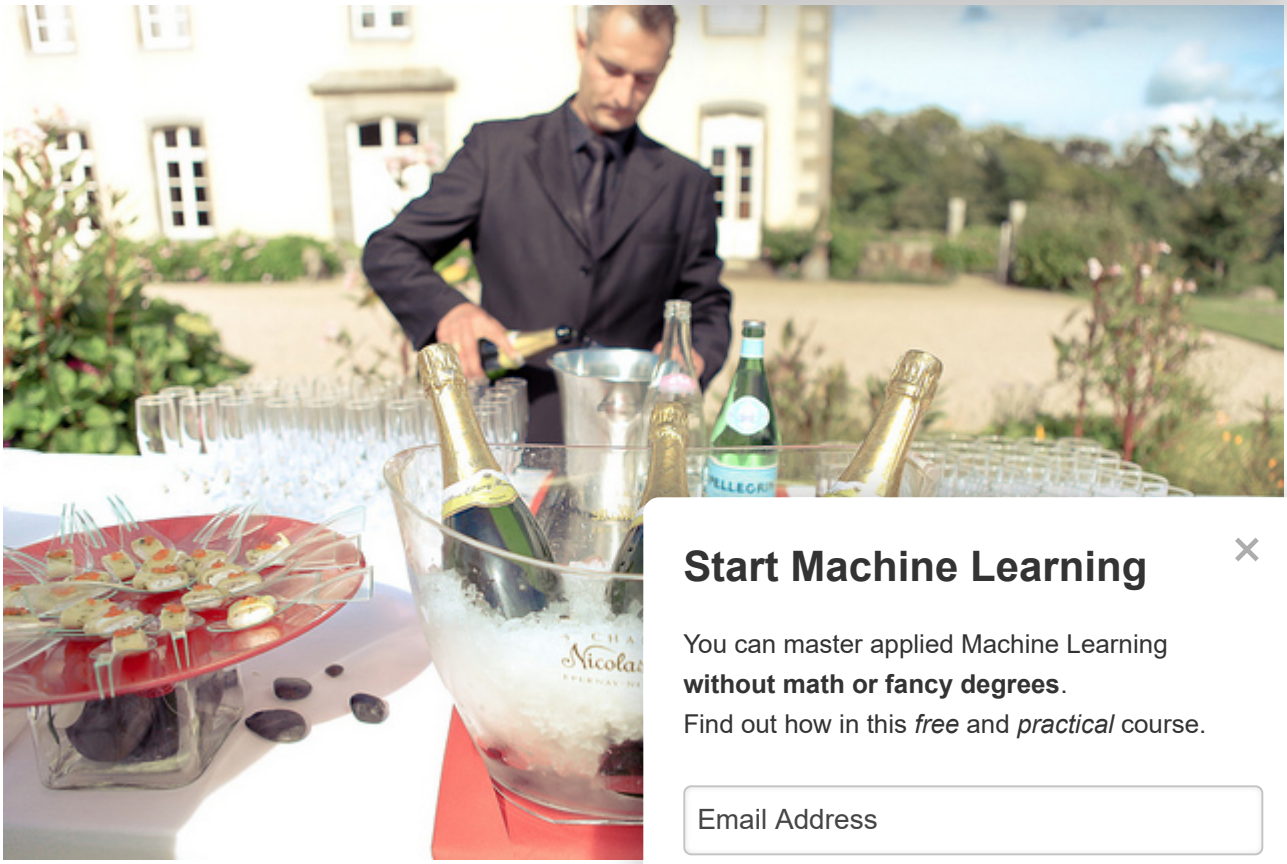
After completing this tutorial, you will know:

- How to confirm your Python environment and carefully define a time series forecasting problem.
- How to create a test harness for evaluating models, develop a baseline forecast, and better understand your problem with the tools of time series analysis.
- How to develop an autoregressive integrated moving average model, save it to file, and later load it to make predictions for new time steps.

Discover how to prepare and visualize time series data and develop autoregressive forecasting models in my new book, with 28 step-by-step tutorials, and full python code.

Let's get started.

- **Update Mar/2017**: Fixed a typo in the code example in the "Review Residual Errors" section where the wrong model was run.
- **Updated Apr/2019**: Updated the link to dataset.
- **Updated Aug/2019**: Updated data loading and date grouping to use new API.
- **Updated Feb/2020**: Updated to_csv() to remove
- **Updated Feb/2020**: Fixed data preparation and l



Time Series Forecast Study with Python
Photo by Basheer Tome

Overview

In this tutorial, we will work through a time series forecasting project from end-to-end, from downloading the dataset and defining the problem to training a final model and making predictions.

This project is not exhaustive, but shows how you can get good results quickly by working through a time series forecasting problem systematically.

The steps of this project that we will through are as follows.

1. Environment.
2. Problem Description.
3. Test Harness.
4. Persistence.
5. Data Analysis.
6. ARIMA Models.
7. Model Validation.

This will provide a template for working through a time series prediction problem that you can use on your own dataset.

1. Environment

This tutorial assumes an installed and working SciPy e

- SciPy
- NumPy
- Matplotlib
- Pandas
- scikit-learn
- statsmodels

If you need help installing Python and the SciPy enviro da distribution that manages much of it for you.

This script will help you check your installed versions of these libraries.

```
# scipy
import scipy
print('scipy: %s' % scipy.__version__)
# numpy
import numpy
print('numpy: %s' % numpy.__version__)
# matplotlib
import matplotlib
print('matplotlib: %s' % matplotlib.__version__)
# pandas
import pandas
print('pandas: %s' % pandas.__version__)
# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)
# statsmodels
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)
```

The results on my workstation used to write this tutorial are as follows:

```
scipy: 0.18.1
numpy: 1.11.2
matplotlib: 1.5.3
pandas: 0.19.1
sklearn: 0.18.1
statsmodels: 0.6.1
```

2. Problem Description

The problem is to predict the number of monthly sales of champagne for the Perrin Freres label (named for a region in France).

The dataset provides the number of monthly sales of champagne from January 1964 to September 1972, or just under 10 years of data.

The values are a count of millions of sales and there are 105 observations.

The dataset is credited to Makridakis and Wheelwright, 1989.

- Download the dataset.

Download the dataset as a CSV file and place it in you
"*champagne.csv*".

3. Test Harness

We must develop a test harness to investigate the dat

This involves two steps:

1. Defining a Validation Dataset.
2. Developing a Method for Model Evaluation.

3.1 Validation Dataset

The dataset is not current. This means that we cannot easily collect updated data to validate the model.

Therefore we will pretend that it is September 1971 and withhold the last one year of data from analysis and model selection.

This final year of data will be used to validate the final model.

The code below will load the dataset as a Pandas Series and split into two, one for model development (*dataset.csv*) and the other for validation (*validation.csv*).

```
# separate out a validation dataset
from pandas import read_csv
series = read_csv('champagne.csv', header=0, index_col=0, parse_dates=True, squeeze=True)
split_point = len(series) - 12
dataset, validation = series[0:split_point], series[split_point:]
print('Dataset %d, Validation %d' % (len(dataset), len(validation)))
dataset.to_csv('dataset.csv', header=False)
validation.to_csv('validation.csv', header=False)
```

Running the example creates two files and prints the number of observations in each.

```
Dataset 93, Validation 12
```

The specific contents of these files are:

- *dataset.csv*: Observations from January 1964 to September 1971 (93 observations)
- *validation.csv*: Observations from October 1971 to September 1972 (12 observations)

The validation dataset is about 11% of the original dataset.

Note that the saved datasets do not have a header line, therefore we do not need to cater for this when working with these files later.

3.2. Model Evaluation

Model evaluation will only be performed on the data in *dataset.csv* prepared in the previous section.

Model evaluation involves two elements:

1. Performance Measure.
2. Test Strategy.

3.2.1 Performance Measure

The observations are a count of champagne sales in r

We will evaluate the performance of predictions using ve more weight to predictions that are grossly wrong and

Any transforms to the data must be reversed before the RMSE is calculated and reported to make the performance between different methods directly comparable.

We can calculate the RMSE using the helper function from the scikit-learn library *mean_squared_error()* that calculates the mean squared error between a list of expected values (the test set) and the list of predictions. We can then take the square root of this value to give us an RMSE score.

For example:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
...
test = ...
predictions = ...
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
```

3.2.2 Test Strategy

Candidate models will be evaluated using walk-forward validation.

This is because a rolling-forecast type model is required from the problem definition. This is where one-step forecasts are needed given all available data.

The walk-forward validation will work as follows:

- The first 50% of the dataset will be held back to train the model.
- The remaining 50% of the dataset will be iterated and test the model.
- For each step in the test dataset:
  - A model will be trained.
  - A one-step prediction made and the prediction stored for later evaluation.
  - The actual observation from the test dataset will be added to the training dataset for the next iteration.
- The predictions made during the iteration of the test dataset will be evaluated and an RMSE score reported.

Given the small size of the data, we will allow a model[illegible]ach prediction.

We can write the code for the test harness using simp[illegible]

Firstly, we can split the dataset into train and test sets[illegible] dataset to *float32* in case the loaded data still has som[illegible]

```
# prepare data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
```


Next, we can iterate over the time steps in the test dataset. The train dataset is stored in a Python list as we need to easily append a new observation each iteration and NumPy array concatenation feels like overkill.

The prediction made by the model is called *yhat* for convention, as the outcome or observation is referred to as *y* and *yhat* (a '*y*' with a mark above) is the mathematical notation for the prediction of the *y* variable.

The prediction and observation are printed each observation for a sanity check prediction in case there are issues with the model.

```
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # predict
    yhat = ...
    predictions.append(yhat)
    # observation
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
```

4. Persistence

The first step before getting bogged down in data analysis and modeling is to establish a baseline of performance.

This will provide both a template for evaluating models using the proposed test harness and a performance measure by which all more elaborate predictive models can be compared.

The baseline prediction for time series forecasting is called the naive forecast, or persistence.

This is where the observation from the previous time step is used as the prediction for the observation at the next time step.

We can plug this directly into the test harness defined in the previous section.

The complete code listing is provided below.


```
from pandas import read_csv
from sklearn.metrics import mean_squared_erro
from math import sqrt
# load data
series = read_csv('dataset.csv', header=None,
# prepare data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # predict
    yhat = history[-1]
    predictions.append(yhat)
    # observation
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
```

Running the test harness prints the prediction and observation for each iteration of the test dataset.

The example ends by printing the RMSE for the model.

In this case, we can see that the persistence model achieved an RMSE of 3186.501. This means that on average, the model was wrong by about 3,186 million sales for each prediction made.

```
...
>Predicted=4676.000, Expected=5010
>Predicted=5010.000, Expected=4874
>Predicted=4874.000, Expected=4633
>Predicted=4633.000, Expected=1659
>Predicted=1659.000, Expected=5951
RMSE: 3186.501
```

We now have a baseline prediction method and perfo

5. Data Analysis

We can use summary statistics and plots of the data to quickly learn more about the structure of the prediction problem.

In this section, we will look at the data from five perspectives:

1. Summary Statistics.
2. Line Plot.
3. Seasonal Line Plots
4. Density Plots.
5. Box and Whisker Plot.

5.1 Summary Statistics

Summary statistics provide a quick look at the limits o f what we are working with.

The example below calculates and prints summary sta

```
1 from pandas import read_csv
2 series = read_csv('dataset.csv', header=None,
3 print(series.describe())
```

Running the example provides a number of summary

Some observations from these statistics include:

- The number of observations (count) matches our expectation, meaning we are handling the data correctly.
- The mean is about 4,641, which we might consider our level in this series.
- The standard deviation (average spread from the mean) is relatively large at 2,486 sales.
- The percentiles along with the standard deviation do suggest a large spread to the data.

```
1 count       93.000000
2 mean      4641.118280
3 std       2486.403841
4 min       1573.000000
5 25%       3036.000000
6 50%       4016.000000
7 75%       5048.000000
8 max      13916.000000
```

5.2 Line Plot

A line plot of a time series can provide a lot of insight into the problem.

The example below creates and shows a line plot of the dataset.

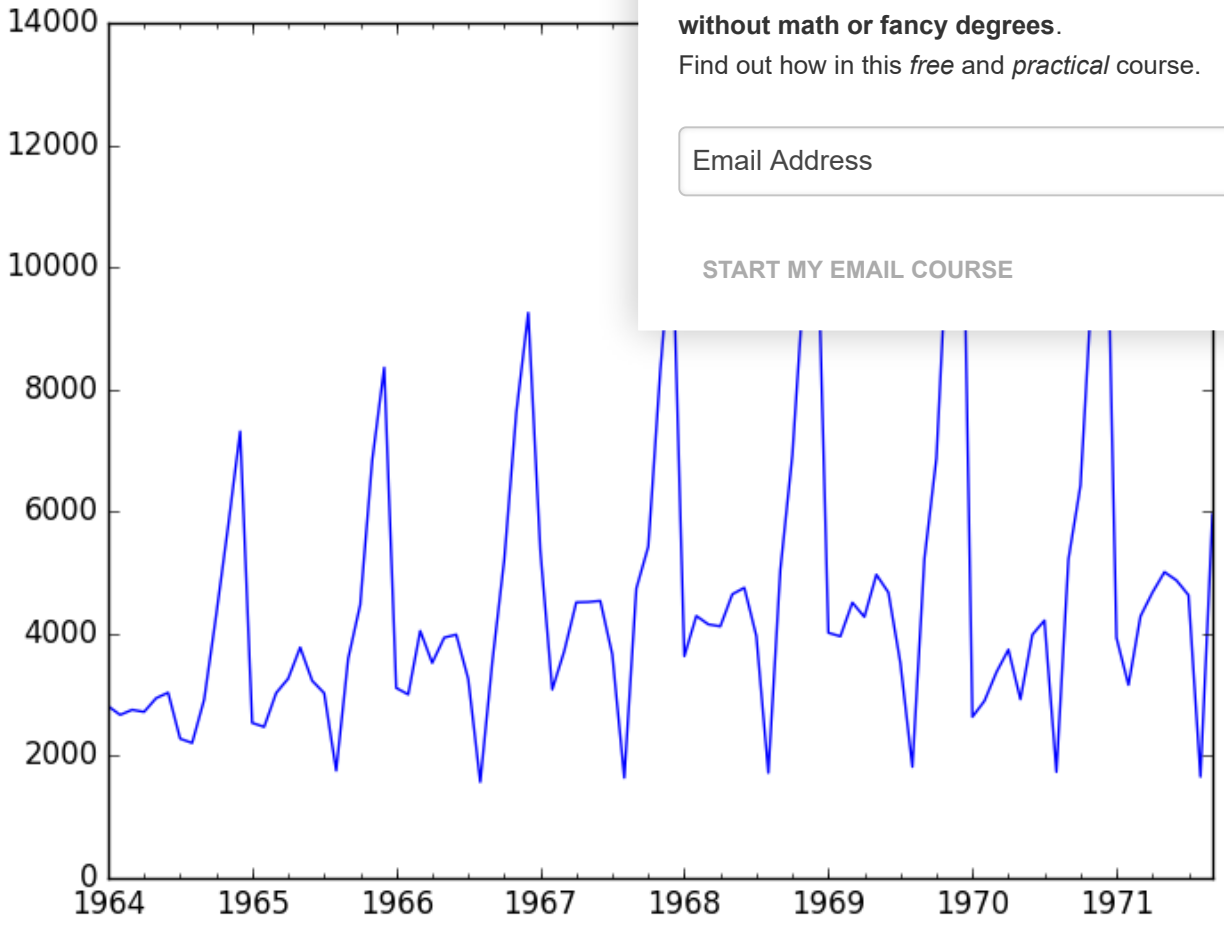
```
1 from pandas import read_csv
2 from matplotlib import pyplot
```

```
3 series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
4 series.plot()
5 pyplot.show()
```

Run the example and review the plot. Note any obvious temporal structures in the series.

Some observations from the plot include:

- There may be an increasing trend of sales over time.
- There appears to be systematic seasonality to the sales for each year.
- The seasonal signal appears to be growing over time, suggesting a multiplicative relationship (increasing change).
- There do not appear to be any obvious outliers.
- The seasonality suggests that the series is almos



Champagne Sales Line Plot

There may be benefit in explicitly modeling the seasonal component and removing it. You may also explore using differencing with one or two levels in order to make the series stationary.

The increasing trend or growth in the seasonal compo
transform.

5.3 Seasonal Line Plots

We can confirm the assumption that the seasonality is a yearly cycle by eyeballing line plots of the dataset by year.

The example below takes the 7 full years of data as separate groups and creates one line plot for each. The line plots are aligned vertically to help spot any year-to-year pattern.

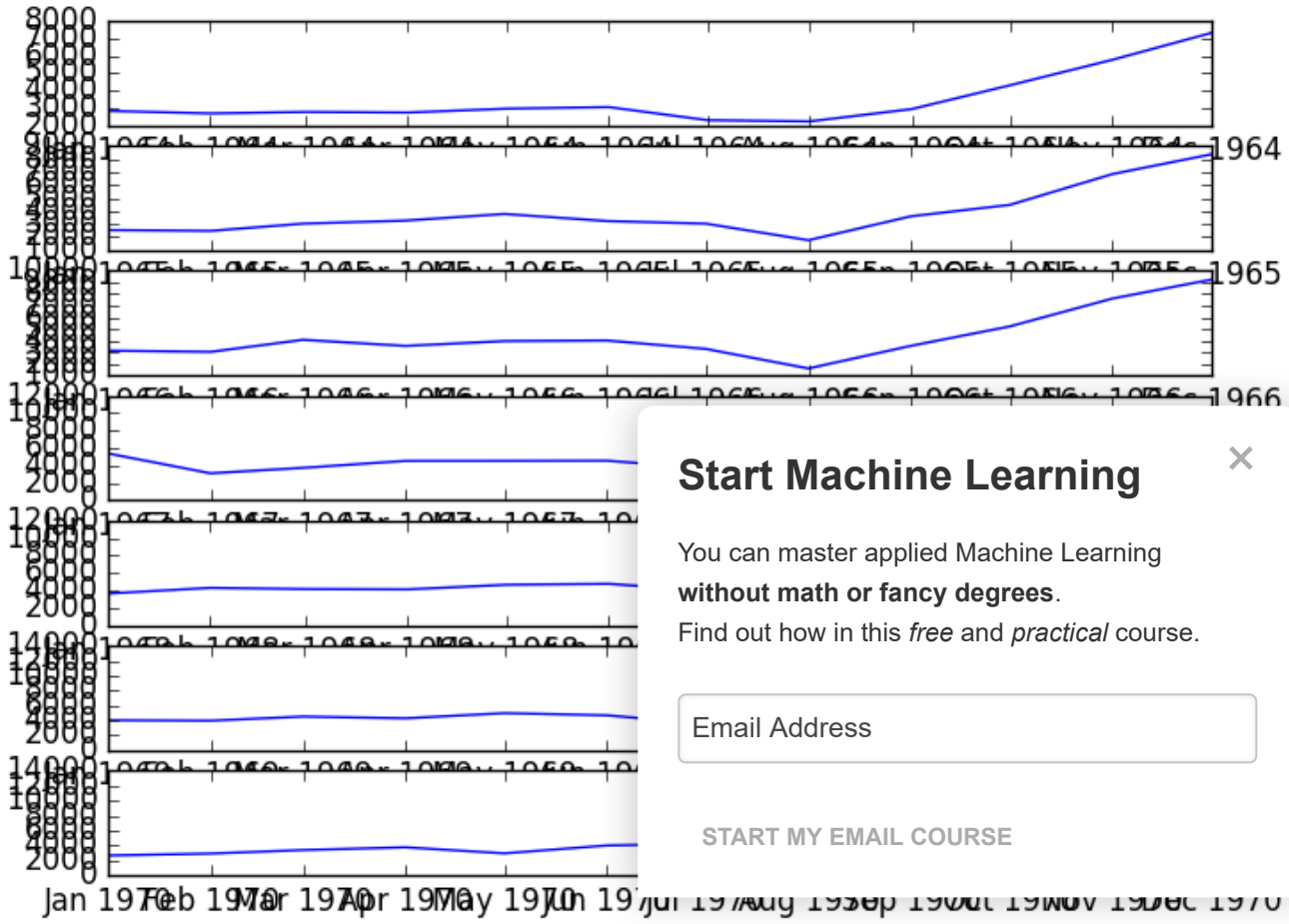
```
from pandas import read_csv
from pandas import DataFrame
from pandas import Grouper
from matplotlib import pyplot
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
groups = series['1964':'1970'].groupby(Grouper(freq='A'))
years = DataFrame()
pyplot.figure()
i = 1
n_groups = len(groups)
for name, group in groups:
    pyplot.subplot((n_groups*100) + 10 + i)
    i += 1
    pyplot.plot(group)
pyplot.show()
```


Running the example creates the stack of 7 line plots.

We can clearly see a dip each August and a rise from [illegible] he same each year, although at different levels.

This will help with any explicitly season-based modeling later.

Start Machine Learning



Seasonal Per Year Line Plots

It might have been easier if all season line plots were added to the one graph to help contrast the data for each year.

5.4 Density Plot

Reviewing plots of the density of observations can provide further insight into the structure of the data.

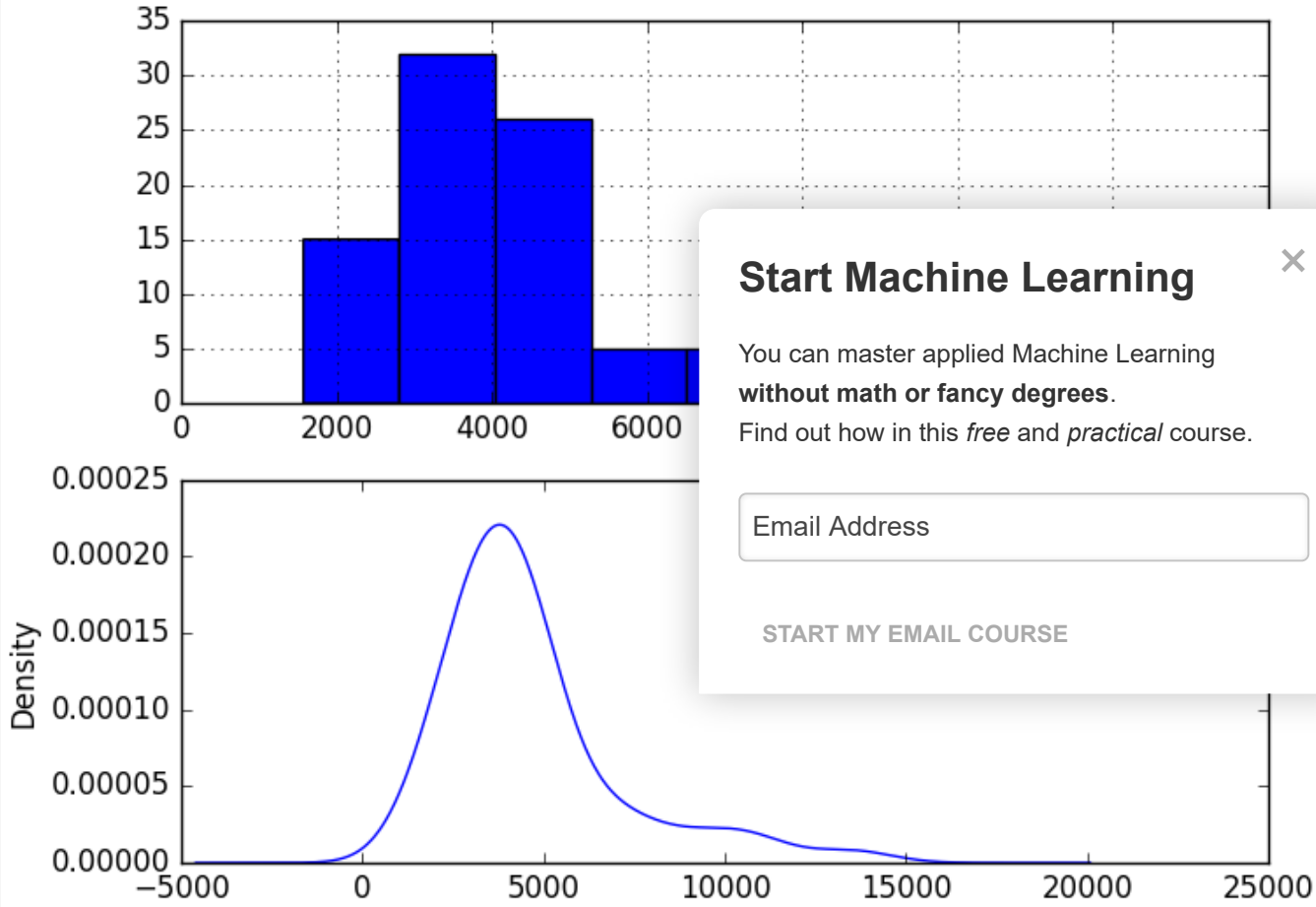
The example below creates a histogram and density plot of the observations without any temporal structure.

```
1 from pandas import read_csv
2 from matplotlib import pyplot
3 series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
4 pyplot.figure(1)
5 pyplot.subplot(211)
6 series.hist()
7 pyplot.subplot(212)
8 series.plot(kind='kde')
9 pyplot.show()
```

Run the example and review the plots.

Some observations from the plots include:

- The distribution is not Gaussian.
- The shape has a long right tail and may suggest an exponential distribution



This lends more support to exploring some power transforms of the data prior to modeling.

5.5 Box and Whisker Plots

We can group the monthly data by year and get an idea of the spread of observations for each year and how this may be changing.

We do expect to see some trend (increasing mean or median), but it may be interesting to see how the rest of the distribution may be changing.

The example below groups the observations by year and creates one box and whisker plot for each year of observations. The last year (1971) only contains 9 months and may not be a useful comparison with the 12 months of observations for other years. Therefore, only data between 1964 and 1970 was plotted.

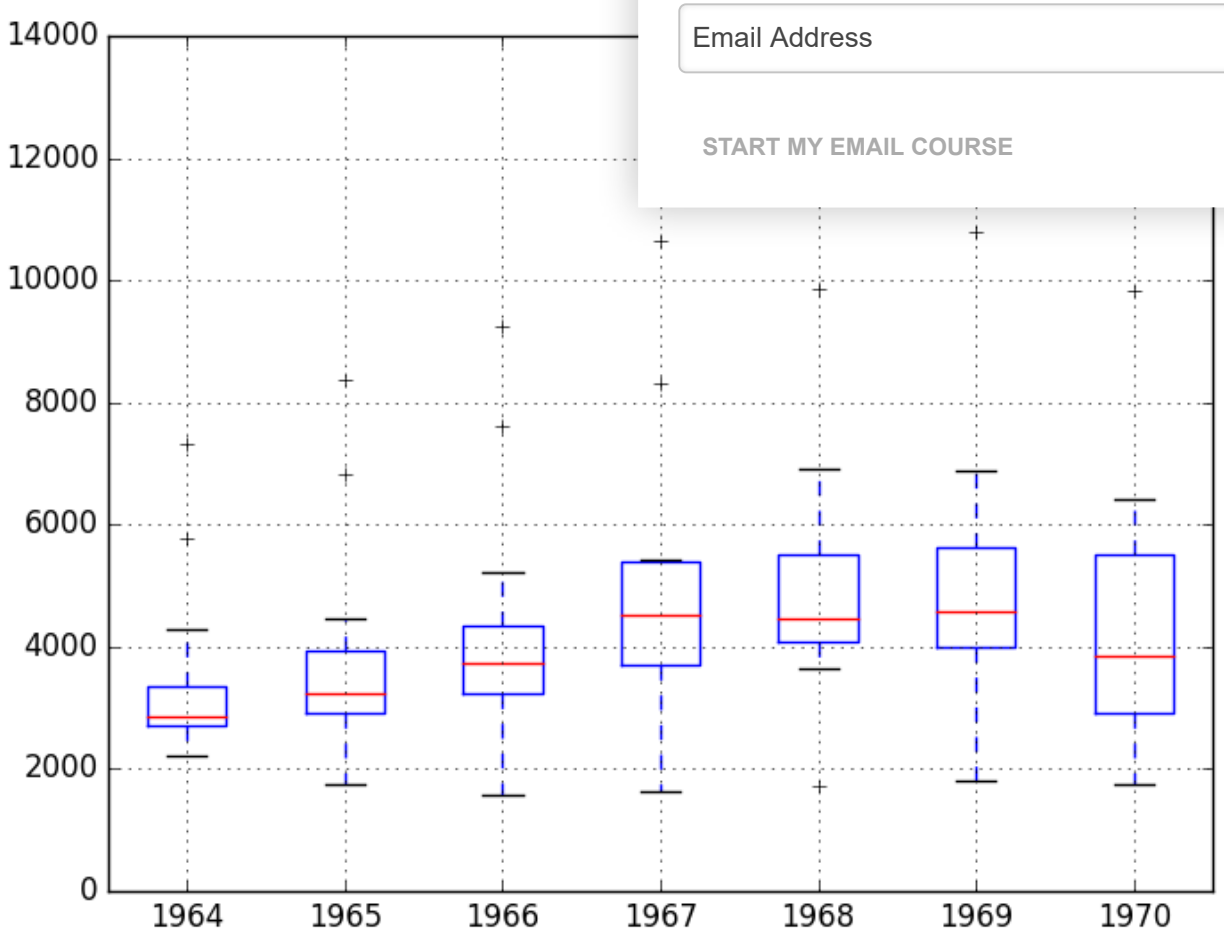
```
1  from pandas import read_csv
2  from pandas import DataFrame
```

```
from pandas import Grouper
from matplotlib import pyplot
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
groups = series['1964':'1970'].groupby(Grouper(freq='A'))
years = DataFrame()
for name, group in groups:
    years[name.year] = group.values
years.boxplot()
pyplot.show()
```

Running the example creates 7 box and whisker plots side-by-side, one for each of the 7 years of selected data.

Some observations from reviewing the plots include:

- The median values for each year (red line) may s
- The spread or middle 50% of the data (blue boxes
- There are outliers each year (black crosses); thes
- The last year, 1970, does look different from the t



Champagne Sales Box and Whisker Plots

The observations suggest perhaps some growth trend the seasonal cycle.

This yearly view of the data is an interesting avenue and could be pursued further by looking at summary statistics from year-to-year and changes in summary stats from year-to-year.

6. ARIMA Models

In this section, we will develop Autoregressive Integrated Moving Average, or ARIMA, models for the problem.

We will approach modeling by both manual and automatic configuration of the ARIMA model. This will be followed by a third step of investigating the residual errors of the chosen model.

As such, this section is broken down into 3 steps:

1. Manually Configure the ARIMA.
2. Automatically Configure the ARIMA.
3. Review Residual Errors.

6.1 Manually Configured ARIMA

The ARIMA(*p,d,q*) model requires three parameters a

Analysis of the time series data assumes that we are

The time series is almost certainly non-stationary. We series and using a statistical test to confirm that the result is stationary.

The seasonality in the series is seemingly year-to-year. Seasonal data can be differenced by subtracting the observation from the same time in the previous cycle, in this case the same month in the previous year. This does mean that we will lose the first year of observations as there is no prior year to difference with.

The example below creates a deseasonalized version of the series and saves it to file *stationary.csv*.

```
from pandas import read_csv
from pandas import Series
from statsmodels.tsa.stattools import adfuller
from matplotlib import pyplot

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
X = series.values
X = X.astype('float32')
# difference data
months_in_year = 12
stationary = difference(X, months_in_year)
stationary.index = series.index[months_in_yea
# check if stationary
```

```
result = adfuller(stationary)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
# save
stationary.to_csv('stationary.csv', header=False)
# plot
stationary.plot()
pyplot.show()
```

Running the example outputs the result of a statistical significance test of whether the differenced series is stationary. Specifically, the augmented Dickey-Fuller test.

The results show that the test statistic value -7.134898 is smaller than the critical value at 1% of -3.515. This suggests that we can reject the null hypothesis w
probability that the result is a statistical fluke).

Rejecting the null hypothesis means that the process
stationary or does not have time-dependent structure.

```
ADF Statistic: -7.134898
p-value: 0.000000
Critical Values:
    5%: -2.898
    1%: -3.515
    10%: -2.586
```

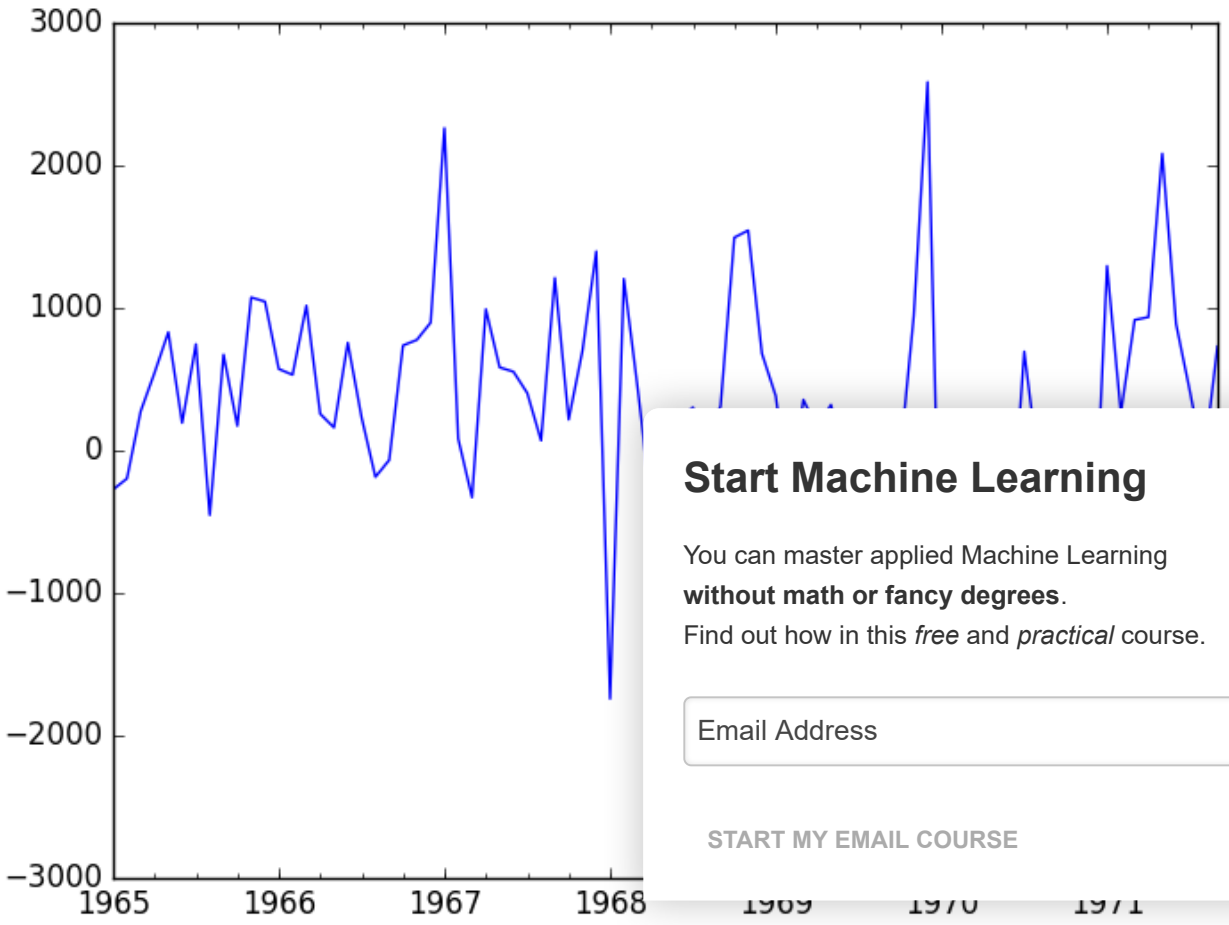

For reference, the seasonal difference operation can be inverted by adding the observation for the same month the year before. This is needed in the case that predictions are made by a model fit on seasonally differenced data. The function to invert the seasonal difference operation is listed below for completeness.

```
# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

A plot of the differenced dataset is also created.

The plot does not show any obvious seasonality or trend, suggesting the seasonally differenced dataset is a good starting point for modeling.

We will use this dataset as an input to the ARIMA model. It also suggests that no further differencing may be required, and that the d parameter may be set to 0.



Seasonal Differenced Champagne Sales Line Plot

The next first step is to select the lag values for the Autoregression (AR) and Moving Average (MA) parameters, *p* and *q* respectively.

We can do this by reviewing Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots.

Note, we are now using the seasonally differenced stationary.csv as our dataset.

The example below creates ACF and PACF plots for the series.

```
from pandas import read_csv
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib import pyplot
series = read_csv('stationary.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
pyplot.figure()
pyplot.subplot(211)
plot_acf(series, ax=pyplot.gca())
pyplot.subplot(212)
plot_pacf(series, ax=pyplot.gca())
pyplot.show()
```

Run the example and review the plots for insights into how to set the *p* and *q* variables for the ARIMA model.

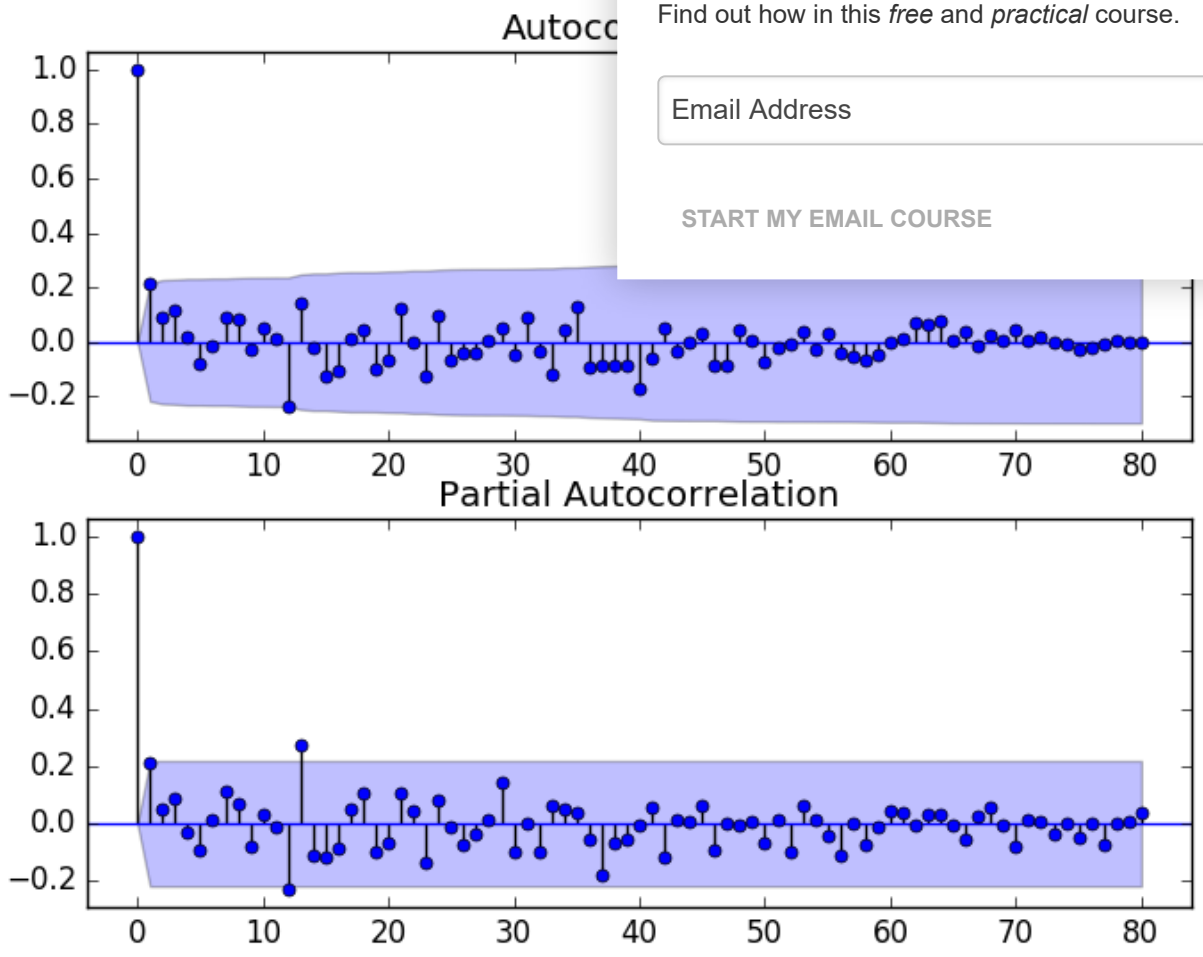
Below are some observations from the plots.

- The ACF shows a significant lag for 1 month.
- The PACF shows a significant lag for 1 month, with perhaps some significant lag at 12 and 13 months.
- Both the ACF and PACF show a drop-off at the same point, perhaps suggesting a mix of AR and MA.

A good starting point for the p and q values is also 1.

The PACF plot also suggests that there is still some seasonality present in the differenced data.

We may consider a better model of seasonality, such a om the model rather than seasonal differencing.



ACF and PACF Plots of Seasonally Differenced Champagne Sales

This quick analysis suggests an ARIMA(1,0,1) on the stationary data may be a good starting point.

The historic observations will be seasonally differenced prior to the fitting of each ARIMA model. The differencing will be inverted for all predictions made to make them directly comparable to the expected observation in the original sale count units.

Experimentation shows that this configuration of ARIMA does not converge and results in errors by the underlying library. Further experimentation showed that adding one level of differencing to the stationary data made the model more stable. The model can be extended to ARIMA(1,1,1).

We will also disable the automatic addition of a trend constant from the model by setting the '*trend*' argument to '*nc*' for no constant in the call to fit(). From experimentation, I find that this can result in better forecast performance on some problems.

The example below demonstrates the performance of


```
from pandas import read_csv
from sklearn.metrics import mean_squared_erro
from statsmodels.tsa.arima_model import ARIMA
from math import sqrt

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - inte
        diff.append(value)
    return diff

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load data
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# prepare data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # difference data
    months_in_year = 12
    diff = difference(history, months_in_year)
    # predict
    model = ARIMA(diff, order=(1,1,1))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = inverse_difference(history, yhat, months_in_year)
    predictions.append(yhat)
    # observation
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
```

Note, you may see a warning message from the underlying linear algebra library; this can be ignored for now.

Running this example results in an RMSE of 956.942, which is dramatically better than the persistence RMSE of 3186.501.

```
...
>Predicted=3157.018, Expected=5010
>Predicted=4615.082, Expected=4874
>Predicted=4624.998, Expected=4633
>Predicted=2044.097, Expected=1659
>Predicted=5404.428, Expected=5951
RMSE: 956.942
```

This is a great start, but we may be able to get improved results with a better configured ARIMA model.

6.2 Grid Search ARIMA Hyperparamet

The ACF and PACF plots suggest that an ARIMA(1,0,

To confirm this analysis, we can grid search a suite of ls result in better out of sample RMSE performance.

In this section, we will search values of *p*, *d*, and *q* for), and find the combination that results in the best perfor o explore all combinations in a subset of integer values.

Specifically, we will search all combinations of the following parameters:

- *p*: 0 to 6.
- *d*: 0 to 2.
- *q*: 0 to 6.

This is (*7* * *3* * *7*), or 147, potential runs of the test harness and will take some time to execute.

It may be interesting to evaluate MA models with a lag of 12 or 13 as were noticed as potentially interesting from reviewing the ACF and PACF plots. Experimentation suggested that these models may not be stable, resulting in errors in the underlying mathematical libraries.

The complete worked example with the grid search version of the test harness is listed below.

```
import warnings
from pandas import read_csv
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - inte
```

```
        diff.append(value)
    return numpy.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# evaluate an ARIMA model for a given order (p,d,q) and return RMSE
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    X = X.astype('float32')
    train_size = int(len(X) * 0.50)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        # difference data
        months_in_year = 12
        diff = difference(history, months_in_
        model = ARIMA(diff, order=arima_order
        model_fit = model.fit(trend='nc', dis
        yhat = model_fit.forecast()[0]
        yhat = inverse_difference(history, yh
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    mse = mean_squared_error(test, prediction
    rmse = sqrt(mse)
    return rmse

# evaluate combinations of p, d and q values
def evaluate_models(dataset, p_values, d_valu
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s RMSE=%.3f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

# load dataset
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# evaluate parameters
p_values = range(0, 7)
d_values = range(0, 3)
q_values = range(0, 7)
warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)
```


Running the example runs through all combinations and reports the results on those that converge without error. The example takes a little over 2 minutes to run on modern hardware.

The results show that the best configuration discovered was ARIMA(0, 0, 1) with an RMSE of 939.464, slightly lower than the manually configured ARIMA fro

not be statistically significant.

```
...
ARIMA(5, 1, 2) RMSE=1003.200
ARIMA(5, 2, 1) RMSE=1053.728
ARIMA(6, 0, 0) RMSE=996.466
ARIMA(6, 1, 0) RMSE=1018.211
ARIMA(6, 1, 1) RMSE=1023.762
Best ARIMA(0, 0, 1) RMSE=939.464
```

We will select this ARIMA(0, 0, 1) model going forward.

6.3 Review Residual Errors

A good final check of a model is to review residual forecast errors.

Ideally, the distribution of residual errors should be a G

We can check this by using summary statistics and pl
ARIMA(0, 0, 1) model. The example below calculates


```
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load data
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# prepare data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # difference data
    months_in_year = 12
    diff = difference(history, months_in_year)
    # predict
    model = ARIMA(diff, order=(0,0,1))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = inverse_difference(history, yhat, months_in_year)
    predictions.append(yhat)
    # observation
    obs = test[i]
```

```
    history.append(obs)
# errors
residuals = [test[i]-predictions[i] for i in range(len(test))]
residuals = DataFrame(residuals)
print(residuals.describe())
# plot
pyplot.figure()
pyplot.subplot(211)
residuals.hist(ax=pyplot.gca())
pyplot.subplot(212)
residuals.plot(kind='kde', ax=pyplot.gca())
pyplot.show()
```

Running the example first describes the distribution of the residuals.

We can see that the distribution has a right shift and that the mean is non-zero at 165.904728.

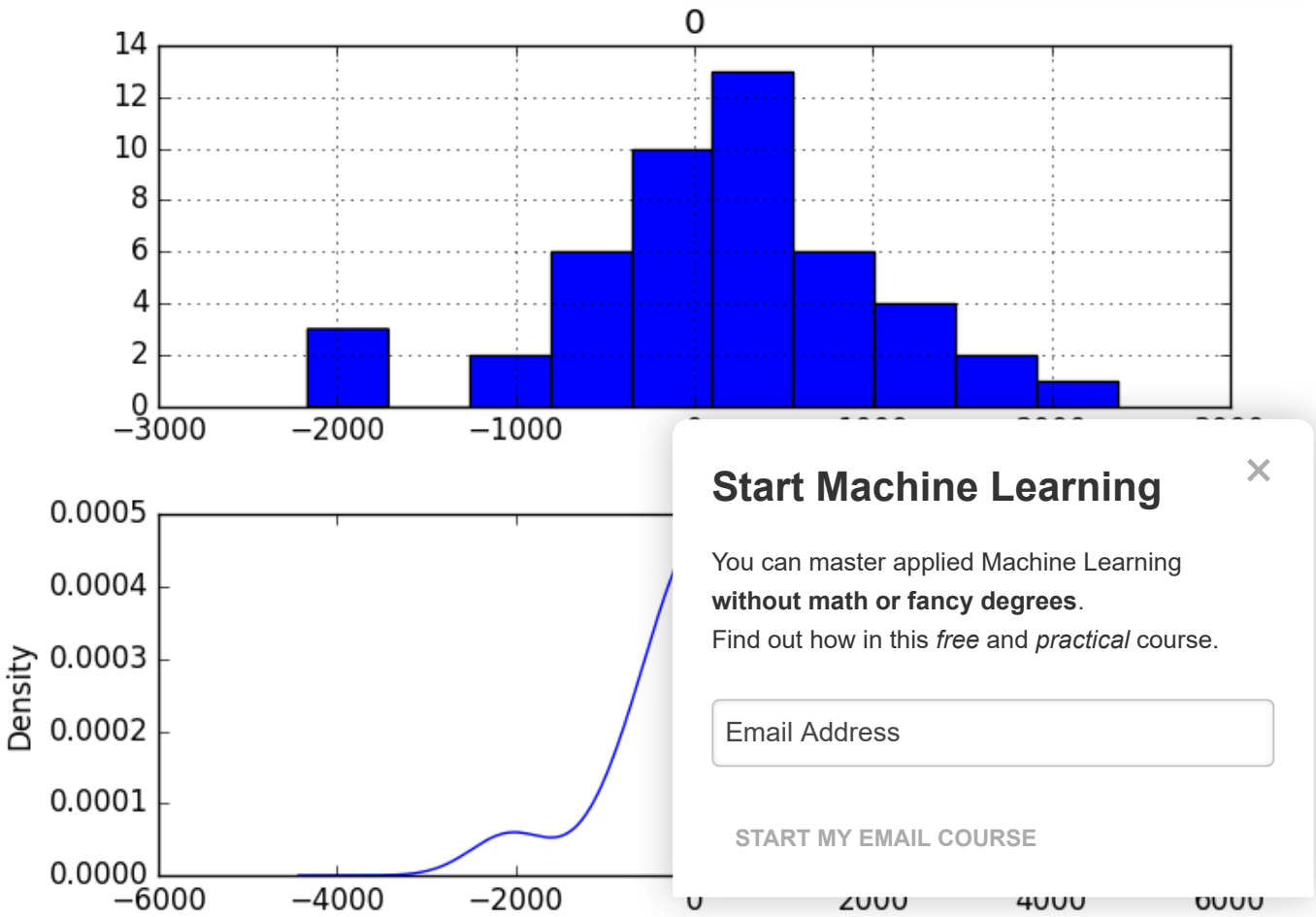
This is perhaps a sign that the predictions are biased.

```
count    47.000000
mean    165.904728
std     934.696199
min   -2164.247449
25%    -289.651596
50%     191.759548
75%     732.992187
max    2367.304748
```

The distribution of residual errors is also plotted.

The graphs suggest a Gaussian-like distribution with a bumpy left tail, providing further evidence that perhaps a power transform might be worth exploring.

Start Machine Learning



Residual Forecast Error Density Plots

We could use this information to bias-correct predictions by adding the mean residual error of 165.904728 to each forecast made.

The example below performs this bias correlation.

```
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot
from sklearn.metrics import mean_squared_error
from math import sqrt

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

```

```
# load data
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# prepare data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
# walk-forward validation
history = [x for x in train]
predictions = list()
bias = 165.904728
for i in range(len(test)):
    # difference data
    months_in_year = 12
    diff = difference(history, months_in_year)
    # predict
    model = ARIMA(diff, order=(0,0,1))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = bias + inverse_difference(history,
    predictions.append(yhat)
    # observation
    obs = test[i]
    history.append(obs)
# report performance
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
# errors
residuals = [test[i]-predictions[i] for i in
residuals = DataFrame(residuals)
print(residuals.describe())
# plot
pyplot.figure()
pyplot.subplot(211)
residuals.hist(ax=pyplot.gca())
pyplot.subplot(212)
residuals.plot(kind='kde', ax=pyplot.gca())
pyplot.show()
```

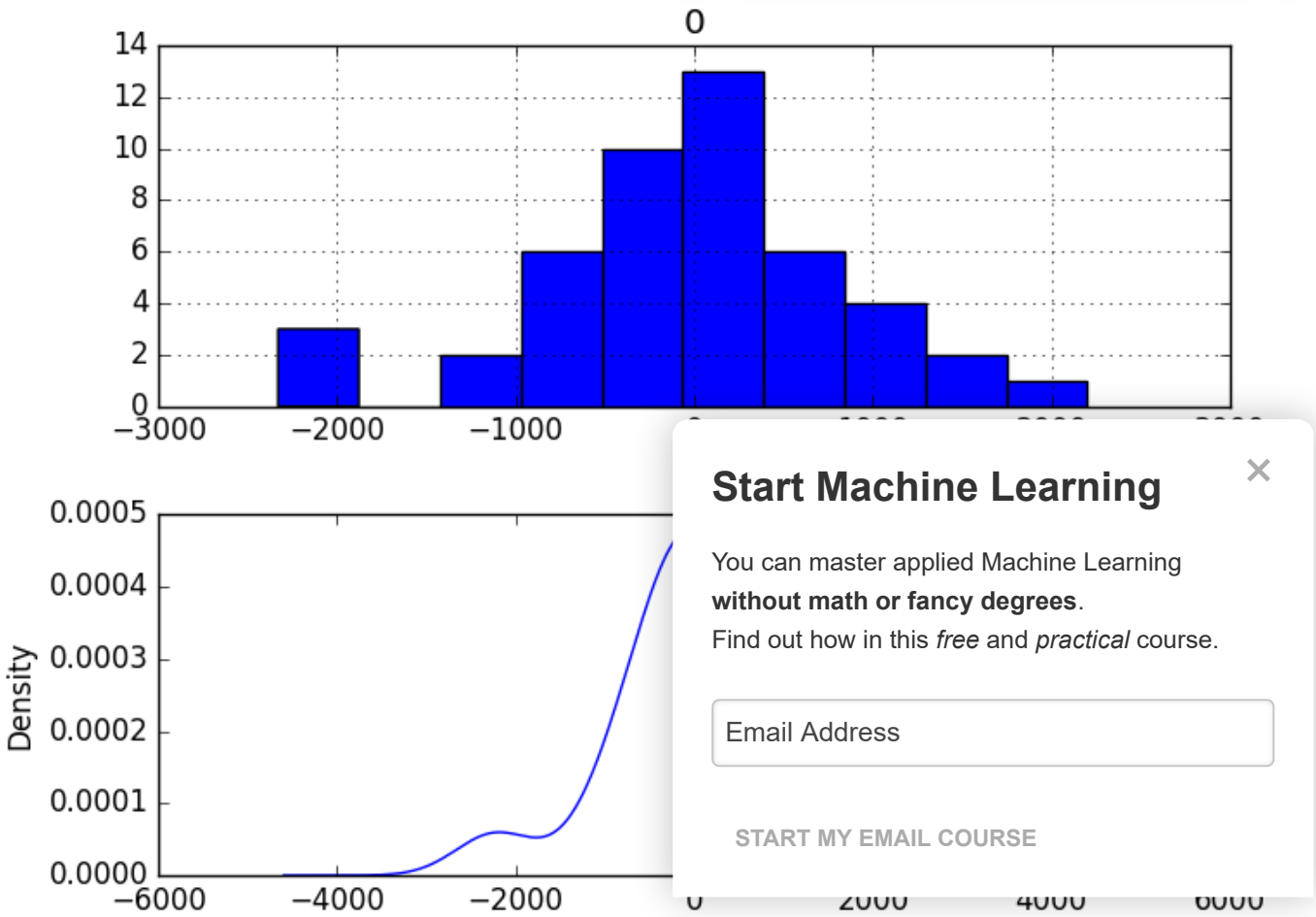

The performance of the predictions is improved very slightly from 939.464 to 924.699, which may or may not be significant.

The summary of the forecast residual errors shows that the mean was indeed moved to a value very close to zero.

```
RMSE: 924.699

count  4.700000e+01
mean   4.965016e-07
std    9.346962e+02
min   -2.330152e+03
25%   -4.555563e+02
50%    2.585482e+01
75%    5.670875e+02
max    2.201400e+03
```

Finally, density plots of the residual error do show a small shift towards zero.

It is debatable whether this bias correction is worth it, but we will use it for now.



Bias-Corrected Residual Forecast Error Density Plots

It is also a good idea to check the time series of the residual errors for any type of autocorrelation. If present, it would suggest that the model has more opportunity to model the temporal structure in the data.

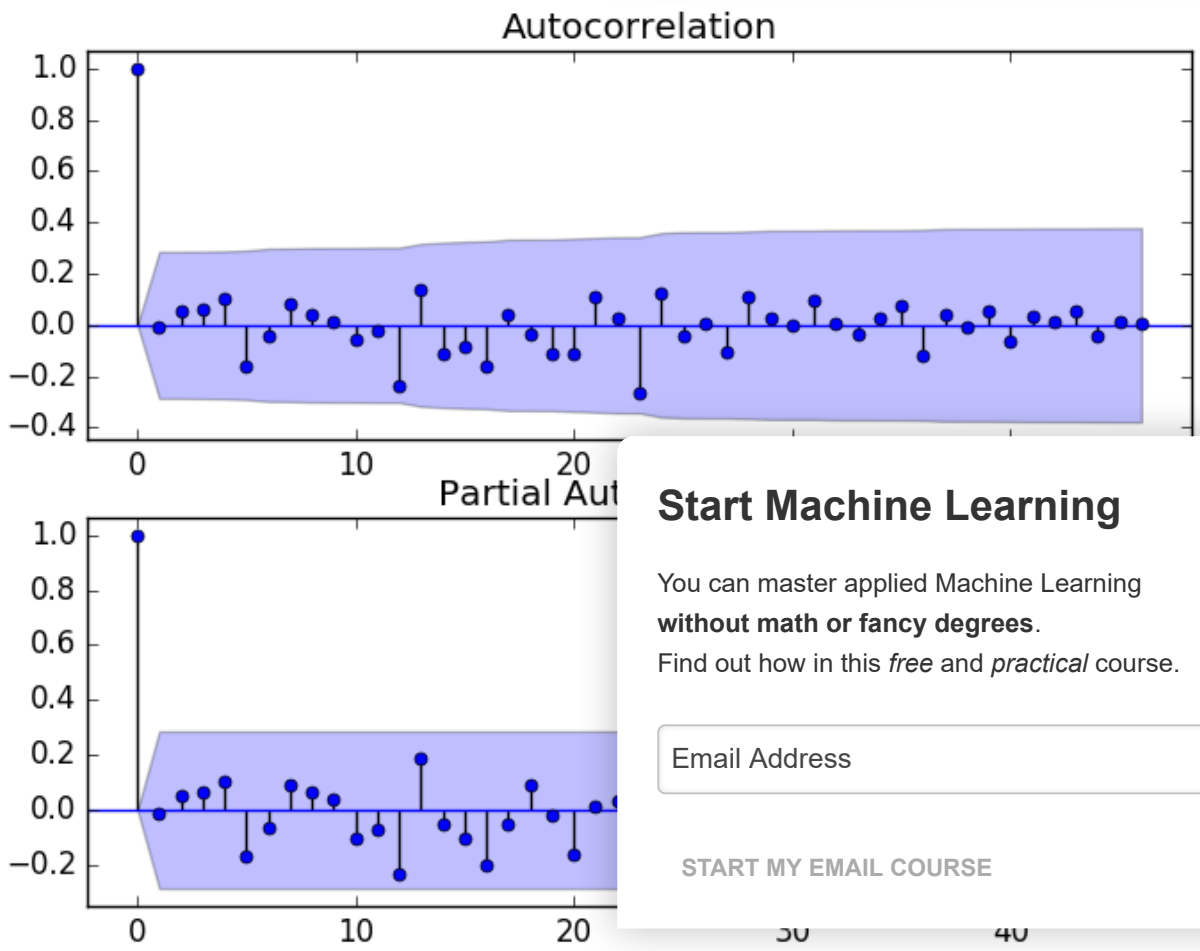
The example below re-calculates the residual errors and creates ACF and PACF plots to check for any significant autocorrelation.

```
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

```
19
20 # load data
21 series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
22 # prepare data
23 X = series.values
24 X = X.astype('float32')
25 train_size = int(len(X) * 0.50)
26 train, test = X[0:train_size], X[train_size:]
27 # walk-forward validation
28 history = [x for x in train]
29 predictions = list()
30 for i in range(len(test)):
31     # difference data
32     months_in_year = 12
33     diff = difference(history, months_in_year)
34     # predict
35     model = ARIMA(diff, order=(0,0,1))
36     model_fit = model.fit(trend='nc', disp=0)
37     yhat = model_fit.forecast()[0]
38     yhat = inverse_difference(history, yhat,
39     predictions.append(yhat)
40     # observation
41     obs = test[i]
42     history.append(obs)
43 # errors
44 residuals = [test[i]-predictions[i] for i in
45 residuals = DataFrame(residuals)
46 print(residuals.describe())
47 # plot
48 pyplot.figure()
49 pyplot.subplot(211)
50 plot_acf(residuals, ax=pyplot.gca())
51 pyplot.subplot(212)
52 plot_pacf(residuals, ax=pyplot.gca())
53 pyplot.show()
```


The results suggest that what little autocorrelation is present in the time series has been captured by the model.

Start Machine Learning



Residual Forecast Error ACF and PACF Plots

7. Model Validation

After models have been developed and a final model selected, it must be validated and finalized.

Validation is an optional part of the process, but one that provides a 'last check' to ensure we have not fooled or misled ourselves.

This section includes the following steps:

1. **Finalize Model**: Train and save the final model.
2. **Make Prediction**: Load the finalized model and make a prediction.
3. **Validate Model**: Load and validate the final model.

7.1 Finalize Model

Finalizing the model involves fitting an ARIMA model on the entire dataset, in this case on a transformed version of the entire dataset.

Once fit, the model can be saved to file for later use.

The example below trains an ARIMA(0,0,1) model on the dataset and saves the whole fit object and the bias to file.

There is a bug in the current stable version of the statsmodels library (v0.6.1) that results in an error when you try to load a saved ARIMA model from file. The error reported is:

```
TypeError: __new__() takes at least 3 arguments (1 given)
```

This bug also seems present in the 0.8 release candidate 1 of statsmodels when I tested it. For more details, see Zae Myung Kim's discussion and fix in this GitHub issue.

We can work around this with a monkey patch that adds a *__getnewargs__()* instance function to the ARIMA class before saving.

The example below saves the fit model to file in the co ter.

```
from pandas import read_csv
from statsmodels.tsa.arima_model import ARIMA
from scipy.stats import boxcox
import numpy

# monkey patch around bug in ARIMA class
def __getnewargs__(self):
    return ((self.endog),(self.k_lags, self.k

ARIMA.__getnewargs__ = __getnewargs__

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# load data
series = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# prepare data
X = series.values
X = X.astype('float32')
# difference data
months_in_year = 12
diff = difference(X, months_in_year)
# fit model
model = ARIMA(diff, order=(0,0,1))
model_fit = model.fit(trend='nc', disp=0)
# bias constant, could be calculated from in-sample mean residual
bias = 165.904728
# save model
model_fit.save('model.pkl')
numpy.save('model_bias.npy', [bias])
```


Running the example creates two local files:

- *model.pkl* This is the ARIMAResult object from the call to *ARIMA.fit()*. This includes the coefficients and all other internal data returned when fitting the model.
- *model_bias.npy* This is the bias value stored as a

7.2 Make Prediction

A natural case may be to load the model and make a single forecast.

This is relatively straightforward and involves restoring the saved model and the bias and calling the *forecast()* method. To invert the seasonal differencing, the historical data must also be loaded.

The example below loads the model, makes a prediction for the next time step, and prints the prediction.

```
from pandas import read_csv
from statsmodels.tsa.arima_model import ARIMAResults
import numpy

# invert differenced value
def inverse_difference(history, yhat, interval
    return yhat + history[-interval]

series = read_csv('dataset.csv', header=None,
months_in_year = 12
model_fit = ARIMAResults.load('model.pkl')
bias = numpy.load('model_bias.npy')
yhat = float(model_fit.forecast()[0])
yhat = bias + inverse_difference(series.value
print('Predicted: %.3f' % yhat)
```


Running the example prints the prediction of about 67

```
Predicted: 6794.773
```

If we peek inside *validation.csv*, we can see that the value on the first row for the next time period is 6981.

The prediction is in the right ballpark.

7.3 Validate Model

We can load the model and use it in a pretend operational manner.

In the test harness section, we saved the final 12 months of the original dataset in a separate file to validate the final model.

We can load this *validation.csv* file now and use it see how well our model really is on “unseen” data.

There are two ways we might proceed:

- Load the model and use it to forecast the next 12 months. The forecast beyond the first one or two months will quickly start to degrade in skill.
- Load the model and use it in a rolling-forecast manner, updating the transform and model for each time step. This is the preferred method as it is how one would use this model in practice as it would achieve the best performance.

As with model evaluation in previous sections, we will make predictions in a rolling-forecast manner. This means that we will step over lead times in the validati to

the history.

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARIMAResults
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# invert differenced value
def inverse_difference(history, yhat, interva
    return yhat + history[-interval]

# load and prepare datasets
series = read_csv('dataset.csv', header=None,
X = dataset.values.astype('float32')
history = [x for x in X]
months_in_year = 12
validation = read_csv('validation.csv', heade                                    =Tru
y = validation.values.astype('float32')
# load model
model_fit = ARIMAResults.load('model.pkl')
bias = numpy.load('model_bias.npy')
# make first prediction
predictions = list()
yhat = float(model_fit.forecast()[0])
yhat = bias + inverse_difference(history, yhat, months_in_year)
predictions.append(yhat)
history.append(y[0])
print('>Predicted=%.3f, Expected=%3.f' % (yhat, y[0]))
# rolling forecasts
for i in range(1, len(y)):
    # difference data
    months_in_year = 12
    diff = difference(history, months_in_year)
    # predict
    model = ARIMA(diff, order=(0,0,1))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = bias + inverse_difference(history, yhat, months_in_year)
    predictions.append(yhat)
    # observation
    obs = y[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
mse = mean_squared_error(y, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
pyplot.plot(y)
pyplot.plot(predictions, color='red')
pyplot.show()
```


Running the example prints each prediction and expe   set.

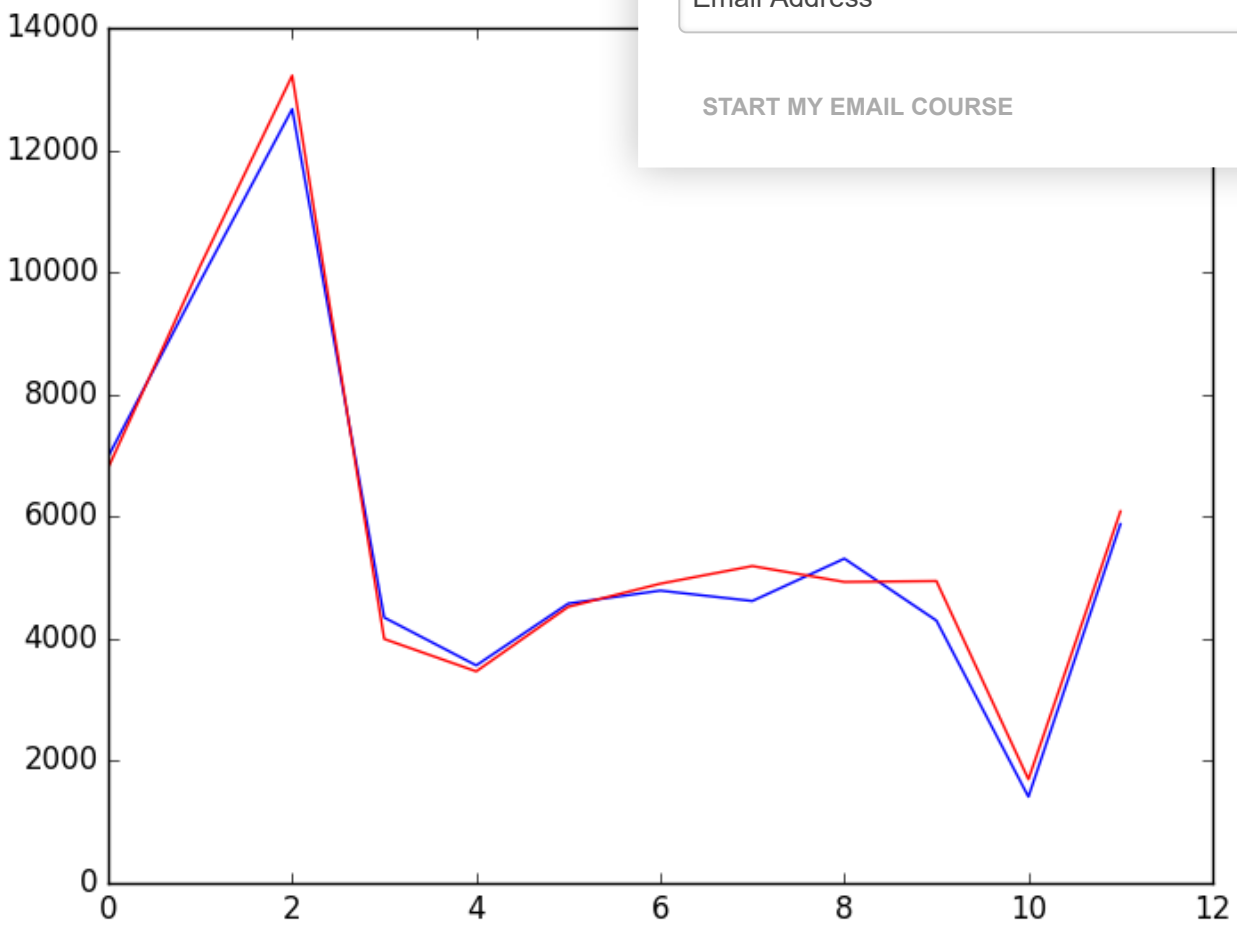
The final RMSE for the validation period is predicted at 361.110 million sales.

This is much better than the expectation of an error of a little more than 924 million sales per month.

```
>Predicted=6794.773, Expected=6981
>Predicted=10101.763, Expected=9851
>Predicted=13219.067, Expected=12670
>Predicted=3996.535, Expected=4348
>Predicted=3465.934, Expected=3564
>Predicted=4522.683, Expected=4577
>Predicted=4901.336, Expected=4788
>Predicted=5190.094, Expected=4618
>Predicted=4930.190, Expected=5312
>Predicted=4944.785, Expected=4298
>Predicted=1699.409, Expected=1413
>Predicted=6085.324, Expected=5877
RMSE: 361.110
```

A plot of the predictions compared to the validation da

At this scale on the plot, the 12 months of forecast sal



Forecast for Champagne Sales Validation Dataset

Summary

In this tutorial, you discovered the steps and the tools for a time series forecasting project with Python.

We have covered a lot of ground in this tutorial; specifically:

- How to develop a test harness with a performance measure and evaluation method and how to quickly develop a baseline forecast and skill.
- How to use time series analysis to raise ideas for how to best model the forecast problem.
- How to develop an ARIMA model, save it, and later load it to make predictions on new data.

How did you do? Do you have any questions about this tutorial?
Ask your questions in the comments below and I will c

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

‹ How Álvaro Lemos got a Machine Learning Internship on a Da

Start Machine Learning

How to Difference a Time Series Dataset with Python ›

169 Responses to *Time Series Forecast Study with Python: Monthly Sales of French Champagne*

Benson Dube February 20, 2017 at 8:46 pm #

REPLY

Thank You Jason.

Would you recommend this example to a starter. I am
and I am currently learning Python sytnax and underst

Regards,

Benson

Jason Brownlee February 21, 2017 at 9:34 a

Hi Benson, this tutorial is an example for
assume you know your way around Python first (o

Benson March 1, 2017 at 6:45 am #

REPLY

A surely steep learning curve taking you out of your comfort zone, but that's the way to learn. ?

Jason Brownlee March 1, 2017 at 8:48 am #

REPLY

For sure. Dive in!

laurenciatitan November 14, 2018 at 1:57 am #

REPLY

helo jason will i be able to contact you and discuss, need explaination..

Jason Brownlee November 14, 20

You can use the contact page to contact me any time via email:
https://machinelearningmastery.com/contact/

Viral Mehta February 24, 2017 at 4:56 am #

REPLY

Jason, thanks for very detailed instructions on how to construct the model. When I add a few additional periods in the validation set (manually) for a short-term forecast beyond the dataset, the model won't run until I provide some 'fake' targets (expected y). However, when I provide some fake targets, I see that model quickly adjusts to those targets. I tried different levels of y values and I see model predicted accordingly, shouldn't the predictions be independent of what targets it sees?

Jason Brownlee February 24, 2017 at 10:14 a

You should not need fake targets Viral.

You can forecast a new out of sample data point b
predicting one-step. E.g.:

```
1 yhat = model_fit.forecast()[0]
```


Viral Mehta February 25, 2017 at 2:14 am #

REPLY

Jason, much appreciate your response. What I am trying to convey is that the model should predict same outputs regardless of the observations. For example, when I change y values in my validation set (last 12 months that the model hasn't seen), my predictions change and the model gives me a very nice RMSE every time. If the model was trained properly, it should have same output for next twelve months regardless of my y values in the validation set. I think you will see this effect very quickly if you also change your y values from validation set. Unless the model is only good for one period forward and needs to continuously adjust based on observed values of last period.

Jason Brownlee February 25, 2017 at 6:00 am #

REPLY

In this specific case we are using walk-forward validation with a model trained for one-step forecasts.

This means as we walk over the validation dataset, observations are moved from validation into training and the model is refit. This is to simulate a new observation being available after a prediction is made for a time step and us being able to update our model in response.

I think what you are talking about is a multi-step forecast, e.g. fitting the model on all of the training data, then forecasting the next n time steps. This is not the experimental setup here, but you are welcome to try it.

Benson Dube March 15, 2017 at 9:11 am #

REPLY

Hello Jason,

If you don't mind me asking, where exactly should the line below fit?:

yhat = model_fit.forecast()[0]

Jason Brownlee March 16, 2017 at

Hi Benson,

Fit the ARIMA model on all of your data.

After that, make a forecast for the next tim

Juanlu February 27, 2017 at 10:15 am #

Time to upgrade to matplotlib 2.0, the colors are nicer 🙂

Jason Brownlee February 28, 2017 at 8:09 am #

REPLY

I agree Juanlu! I have recently upgraded myself.

Hugo March 24, 2017 at 2:38 am #

REPLY

Dear Jason,
Thanks for the post. Clear, concise and working.

Have you considered to forecast the TS using a SARIMA model instead of substracting the seasonality and adding it latter? As a matter of fact, statsmodel has it integrated in its dev version. (http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html)

I am wondering if it is possible to take into account hierarchy, like the forecast of monthly sales divided in sub-forecasts of french champagne destination markets, using python.

Thanks and keep posting!

Jason Brownlee March 24, 2017 at 7:59 am #

REPLY

Hi Hugo, yes I am familiar with SARIMA, but I decided to keep the example limited to ARIMA.

Yes, as long as you have the data to train/verify the model. I would consider starting with independent models for each case, using all available data and tease out how to optimize it further after that.

Luis Ashurei May 2, 2017 at 7:25 pm #

REPLY

Thanks for the wonderful post Jason,

Two quick question:

1.
When I'm doing Grid Search ARIMA Hyperparameters spent about 1 minutes. However the parameter range end? I'm concerning the model is too slow to use.

2.
Does ARIMA support multi-step forecast? E.g. What if overfitting?

Thanks!

Jason Brownlee May 3, 2017 at 7:34 am #

REPLY

Your timing sounds fine. If it is too slow for you, consider working with a sub-sample of your data.

You can do multi-step forecasting directly (forecast() function) or by using the ARIMA model recursively: http://machinelearningmastery.com/multi-step-time-series-forecasting/

Luis May 3, 2017 at 6:50 pm #

REPLY

Thanks in advice.

Pramod Gupta July 3, 2017 at 7:58 pm #

REPLY

Hello Jason

Thanks for this awesome hands-on on Time series. Ho

I extracted year and month from date column as my features for model. Then I built a Linear Regression model and a Random Forest model. My final prediction was a weighted average of both these models. I got an rmse of 366 (similar to yours i.e. 361 on validation data).

Can this approach be the alternative for an ARIMA model? What can be the possible drawbacks of this approach?

Appreciate your comments
Thanks

Jason Brownlee July 6, 2017 at 10:01 am #

REPLY

Nice work. ARIMA may be a simpler mod

Roberto July 5, 2017 at 7:26 am #

Hi just a silly note have you consider using ite
https://pastebin.com/w0apb6Tg

in python nested loops are not so readable.

Jason Brownlee July 6, 2017 at 10:21 am #

REPLY

Thanks for the suggestion.

Rishi Patil July 8, 2017 at 9:15 am #

REPLY

Jason, I just finished reading “Introduction to Time Series Forecasting in Python” in detail and had two questions:
* Are you planning to come out with a “Intermediate course in Time Series Forecasting in Python” soon?
* You spend the first few chapters discussing how to reframe a time-series problem into a typical Machine Learning problem using Lag features. However, in later chapters, you only use ARIMA models, that obviate the necessity of using explicitly generated lag features. What’s the best way to use explicitly generated timeseries features with other ML algorithms (such as SVM or XGBoost)?
(I tried out a few AR lags with XGB and got decent results using but couldn’t figure out how to incorporate the MA parts).
Would appreciate any insights.

Jason Brownlee July 9, 2017 at 10:50 am #

I may have a more advanced TS book in the future.

Great question. You would have to calculate the MA features manually.

Jan August 2, 2017 at 8:34 pm #

REPLY

Hi Jason,

can it be that TimeGrouper() does not work if there are months missing in a year? Also there are no docs available for TimeGrouper(). May you can use pd.Grouper in your future examples?

Regards

Jason Brownlee August 3, 2017 at 6:49 am #

I think it does work with missing data, allo

TimeGrouper does not have docs, but Grouper do
https://pandas.pydata.org/pandas-docs/stable/gen

Sambit August 17, 2017 at 5:09 am #

Hi,
Thanks for this great link on time series.

I am not able to acces the dataset stored at:- https://datamarket.com/data/set/22r5/perrin-freres-monthly-champagne-sales-millions-64-72#!ds=22r5&display=line

Regards,
Sambit

Jason Brownlee August 17, 2017 at 6:49 am #

REPLY

I'm sorry to hear that, here is the full dataset:

```
"Month","Sales"
"1964-01",2815
"1964-02",2672
"1964-03",2755
"1964-04",2721
"1964-05",2946
"1964-06",3036
"1964-07",2282
"1964-08",2212
"1964-09",2922
"1964-10",4301
```

```
"1964-11",5764
"1964-12",7312
"1965-01",2541
"1965-02",2475
"1965-03",3031
"1965-04",3266
"1965-05",3776
"1965-06",3230
"1965-07",3028
"1965-08",1759
"1965-09",3595
"1965-10",4474
"1965-11",6838
"1965-12",8357
"1966-01",3113
"1966-02",3006
"1966-03",4047
"1966-04",3523
"1966-05",3937
"1966-06",3986
"1966-07",3260
"1966-08",1573
"1966-09",3528
"1966-10",5211
"1966-11",7614
"1966-12",9254
"1967-01",5375
"1967-02",3088
"1967-03",3718
"1967-04",4514
"1967-05",4520
"1967-06",4539
"1967-07",3663
"1967-08",1643
"1967-09",4739
"1967-10",5428
"1967-11",8314
"1967-12",10651
"1968-01",3633
"1968-02",4292
"1968-03",4154
"1968-04",4121
"1968-05",4647
"1968-06",4753
"1968-07",3965
"1968-08",1723
"1968-09",5048
"1968-10",6922
"1968-11",9858
"1968-12",11331
"1969-01",4016
"1969-02",3957
"1969-03",4510
"1969-04",4276
"1969-05",4968
"1969-06",4677
"1969-07",3523
"1969-08",1821
"1969-09",5222
"1969-10",6872
"1969-11",10803
"1969-12",13916
"1970-01",2639
"1970-02",2899
"1970-03",3370
```

```
"1970-04",3740
"1970-05",2927
"1970-06",3986
"1970-07",4217
"1970-08",1738
"1970-09",5221
"1970-10",6424
"1970-11",9842
"1970-12",13076
"1971-01",3934
"1971-02",3162
"1971-03",4286
"1971-04",4676
"1971-05",5010
"1971-06",4874
"1971-07",4633
"1971-08",1659
"1971-09",5951
"1971-10",6981
"1971-11",9851
"1971-12",12670
"1972-01",4348
"1972-02",3564
"1972-03",4577
"1972-04",4788
"1972-05",4618
"1972-06",5312
"1972-07",4298
"1972-08",1413
"1972-09",5877
```


sarrauste August 24, 2017 at 1:15 am #

REPLY

hi,
i am having the following error:

X = X.astype('float32')
ValueError: could not convert string to float: '1972-09'

can you please help me
ty

Jason Brownlee August 24, 2017 at 6:44 am #

REPLY

Looks like you may have missed some lines of code. Confirm you have them all.

Nico October 29, 2018 at 5:02 am #

REPLY

Hi Jason, I am having the same error. I checked the lines and I have them all. How can python convert 'yyyy-mm' to float32?
I dowload the csv from the link of datamarket y

What am i missing?
Thnanx

REPLY

Jason Brownlee October 29, 2018 at 6:04 am #

I have some suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

sandip August 28, 2017 at 8:09 pm #

Hi Jason , Nice approach for time series forec

Just had a one small doubt,Here we are tuning our AR
measure but generally we tune our parameter on the t
whether those parameters are well generalized or not
Because my concern is that if we choose our paramet
will generalized to coming new test data or not? I feel t
because we are specifically choosing parameter those
checking whether our model is working/fitted well for o
If i am wrong just correct me.Let me know the logic bel

REPLY

Jason Brownlee August 29, 2017 at 5:03 pm #

Yes, that is a less biased way to prepare a model.

More here:
https://machinelearningmastery.com/difference-test-validation-datasets/

REPLY

sandip August 29, 2017 at 5:58 pm #

Thank you so much Jason.

REPLY

Jason Brownlee August 30, 2017 at 6:13 am #

You're welcome sandip.

Anthy August 31, 2017 at 2:24 am #

REPLY

Hi Jason,
Thank you for this detailed and clear tutorial.
I have a question concerning fiding ARIMA's parameters. I didn't understand why we have an imbricated "for" loop and we do consider all the p values and not all the q values.

Please ask if the question is not that clear.

Thank you in advance.

Jason Brownlee August 31, 2017 at 6:23 am

Please review the example again, we gric

Ella Zhao September 2, 2017 at 4:26 am #

Hi Jason,

Thanks for this awesome example! It helped me a lot!
prediction loop.

Why did you use history.append(obs) instead of history.append(yhat) when doing the loop for prediction? Sees like you add the observation (real data in test) to history. And yhat = bias + inverse_difference(history, yhat, months_in_year) is based on the history data. but actually we don't have those observation data when solving practical problem.

I've tried to use history.append(yhat) in my model, but the result is worse than using history.append(obs).

Appreciate your comments

Thanks,

Ella

Jason Brownlee September 2, 2017 at 6:16 am #

REPLY

Good question, because in this scenario, we are assuming that each new real observation is available after we make a prediction.

Pretend we are running this loop one iteration per day, and we get yesterdays observation this morning so we can accurately predict tomorrow.

Ella Zhao September 2, 2017 at 11:46 pm #

REPLY

Thank you so much for your explanation!!

Jason Brownlee September 3, 2017 at 5:45 am #

REPLY

You're welcome.

Alaoui September 3, 2017 at 12:27 am #

Hello,

Thanks a lot for this work.
I have a point to discuss on concerning the walk forwa
I saw that you fill the history part by a value from the te
…..
# observation
obs = test[i]
history.append(obs)
….

Do we really have to do this or we have to use the new yhat computed for the next predictions?
Indeed when we have to do a future prediction on some range date we don't have the historical value….
Regards and thanks in advance for your feedback

Simo

Jason Brownlee September 3, 2017 at 5:48 am #

REPLY

It depends on your model and project goals.

In this case, we are assuming that the true observations are available after each modeling step and can be used in the formulation of the next prediction, a reasonable assumption, but not required in general.

If you use outputs as inputs (e.g. recursive multi-step forecasting), error will compound and things will go more crazy, sooner (e.g. model skill will drop). See this post:
https://machinelearningmastery.com/multi-step-time-series-forecasting/

Alaoui September 3, 2017 at 12:49 am #

REPLY

Start Machine Learning

Sorry, I just read the Ella question…So for long range future prediction we have ti use the yhat…
Thanks

REPLY

Jason Brownlee September 3, 2017 at 5:48 am #

You can if you choose.

REPLY

Sujeet September 16, 2017 at 12:22 pm #

Hi,

I am getting NAN after these steps:

```
from pandas import Series
series = Series.from_csv('champagne.csv', header=0)
print(len(series))
split_point = len(series) – 12
print(split_point)
dataset, validation = series[0:split_point], series[split_p
print(dataset)
print('Dataset %d, Validation %d' % (len(dataset), len(v
dataset.to_csv('dataset.csv')
validation.to_csv('validation.csv')
```


REPLY

Jason Brownlee September 17, 2017 at 5:22 am #

I'm sorry to hear that.

Ensure that you have copied all of the code and that you have removed the footer from the data file.
Also confirm that your environment is up to date.

REPLY

Jaryn September 20, 2017 at 3:24 am #

Hi Jason,

This is extremely useful. Thank you very much!

I was wondering if you could provide the full code to extend the forecast for 1 year ahead. I know you mentioned above that there is a forecast function but when I run:

```
yhat = model_fit.forecast()[0]
pyplot.plot(y)
pyplot.plot(yhat, color='green')
```

pyplot.plot(predictions, color='red')
pyplot.show()

I don't get any green lines in my code. I'm sure I'm missing a lot of things here, but I don't know what.

Thank you again, much appreciated!

REPLY

Jason Brownlee September 20, 2017 at 6:01 am #

This post will help you with making longer-term forecasts:
https://machinelearningmastery.com/make-sample-forecasts-arima-python/

Kunal Shrivastava September 20, 2017 at 4:54 pm

Hi Jason,
I am using this modelling steps to model my problem. V
from Grid Search for doing prediction, I am getting Line
Due to this I am not able to predict the values.

My second concern is if this is happening for Grid sear
environment. Lets say i have to forecast some values f
achieve this goal ?

REPLY

Jason Brownlee September 21, 2017 at 5:36 am #

Sorry to hear that, perhaps you have not set all of the parameters the same as you used in the grid search?

I have some notes about models in production here that may help:
http://machinelearningmastery.com/deploy-machine-learning-model-to-production/

REPLY

Tri Bien September 24, 2017 at 7:57 pm #

Hi Jason,

Thank for your post, it is really helpful for me. It is great!
I have a small question:
In the post, you use 2 data set: dataset.csv and validation.csv
+ In dataset.csv, you split it and use Walk-forward validation -> I totally agree.
+ In validation.csv, you still re-train ARIMA model with walk-forward validation (model_fit = model.fit(trend='nc', disp=0) – line 45 in section 7.3) -> I think it is unseen data for testing, so we don't train model here and only test the model's performance. Is i

Jason Brownlee September 25, 2017 at 5:38 am #

REPLY

In the case of walk forward validation, we are assuming that the real observations are available after the time of prediction.

You can make different assumptions when evaluating your model.

Navid September 30, 2017 at 1:50 am #

REPLY

Hi, Jason
First of all, thank you for your wonderful tutorial.
Until now (part 5.4) I just have one doubt: why you plot
If the goal is to see how the distribution shape is simila
the distribution of data?

Thanks
Navid

Jason Brownlee September 30, 2017 at 7:44

Great question, at that point we are early in the analysis. Generally, I'd recommend looking at density plots before and after making the series stationary.

Trend removal generally won't impact the form of the distribution if we are doing simple linear operations.

Nick October 19, 2017 at 12:46 am #

REPLY

Hi Jason,
Thanks for your awesome post. Could you explain how to set the 'interval' in the function difference if I only have 1-year data? My dataset is recorded in half an hour, from June 2016 to June 2017. These data number are large in summer and small in winter, i.e. in winter it is between 0-2000, but in summer it is between 5000-14000.

Shashank Hegde October 22, 2017 at 3:46 pm #

REPLY

Hi Jason,
Thank you for the post.I want to implement 'ARIMA' fur

Do you know where I can find the algorithm to implement ‘ARIMA’ function as well understanding that in detail manner?

REPLY

Jason Brownlee October 23, 2017 at 5:41 am #

A good textbook on the math is this one: http://amzn.to/2zvfIcB

REPLY

Hara October 28, 2017 at 3:38 am #

Hi Jason,

Thanks for the brief tutorial on Time Series forecasting
the index does not contain dates” when running the AR

Jason Brownlee October 28, 2017 at 5:16 am

Be sure you copied all of the sample code
have prepared the data, including removing the foo

REPLY

Makis December 18, 2017 at 8:27 pm #

Dear Jason,

Thanks for your amazing tut!

I have one problem only when i run the rolling forecasts.

I use a different dataset which has temperature values for every minute of the day. I train the model with the first four days and i use the last day as a test dataset. My problem might be because i add the test observation to the history and my arima returns the following error: raise ValueError(“The computed initial AR coefficients are not ”
ValueError: The computed initial AR coefficients are not stationary
You should induce stationarity, choose a different model order, or you can
pass your own start_params.

for i in range(0, len(x_test)):

# difference data
diff = difference(history, minutes)
# predict
model = importer.ARIMA(diff, order=(5, 0, 1))
model_fit = model.fit(trend=’nc’, disp=0)
yhat = model_fit.forecast()[0]

```
yhat = inverse_difference(history, yhat, minutes)
predictions.append(yhat)
# observation
obs = x_test[i]
history.append(obs)
print('Predicted=%.6f, Expected=%.6f' % (yhat, obs))
```

My x_test dataset contains 1440 rows and i get the error on the 1423 iteration of the loop. Until iteration 1423 the each arima model does not have issues.

Your help is precious to me.

Thanks again!

Kindest Regards,
Makis

Jason Brownlee December 19, 2017 at 5:17 #

Have you confirmed that your data set is s
seasonal structure?

I have many posts on detrending and seasonal adj

Makis December 20, 2017 at 12:54 am #

REPLY

Dear Jason,

Thanks for your response!

I have used a different hyperparameter (Arima 5,1,1) instead, and everything worked. I don't know if this is the right thing, since 5,0,1 and 5,1,1 had exactly the same RMSE. What is your opinion about that? And please, one final question, my results now are extremely good, with an RMSE less than 0,01. The prediction line in the plot is almost over the the real data line. Does this has to do with the history.append(obs)? I'm not sure i understand correctly why the test observation is added to the history. And what is the difference of doing a prediction in a for loop with a new model for every step compared by using the steps parameter in 1 model?

Sorry for the long questions!

Your tutorials are even better than the books i'm currently reading!

Cheers from Greece.

Jason Brownlee December 20, 2017 at 5:47 am #

REPLY

Well done. I'd recommend using the model that is skillful and stable.

Makis December 20, 2017 at 8:22 am #

Hey Jason! What i'm trying to say is, maybe the good predictions are because i append the y observation to the history. Why we do this? what is the purpose? If we do not append the y observation then the results are going to be still valid?

Jason Brownlee December 20

In the test setup we are assu
prediction that we can in turn use to m

Your specific framing of the problem m
harness to capture that.

Makis December 20, 2017 at 10:4

Hey Jason, for once more tha
I will follow your suggestion right now 🙂

Thanks for everything

Kind Regards,
Makis

Makis December 20, 2017 at 10:57 pm #

Hey Jason, one last question.

Since i do the rolling forecast manner and i introduce a new model for every iteration, why do i need to save a model and do a the first prediction based on that?

Can i simply do the rolling forecast loop to start from 0 instead of 1?

Jason Brownlee December 21, 2017 at 5:26 am #

Sure.

Amir December 19, 2017 at 6:51 pm #

REPLY

Hi Jason,

Thank you so much for this amazing tutorial. It really helps me to learn time series forecasting and prepare for my new job. I'm a total beginner in data analysis/science as I'm currently making transition from engineering to data analysis.

But the problem that I'm facing with this tutorial is as I'm running the code, I'm stuck with the final step, "Validate the Model". I've tried to re-copy and re-run the sample code many times but it didn't seem to work. Here is the error.

```
>Predicted=10101.763, Expected=9851
>Predicted=13219.067, Expected=12670
>Predicted=3996.535, Expected=4348
>Predicted=3465.934, Expected=3564
>Predicted=4522.683, Expected=4577
>Predicted=4901.336, Expected=4788
>Predicted=5190.094, Expected=4618
>Predicted=4930.190, Expected=5312
>Predicted=4944.785, Expected=4298
>Predicted=1699.409, Expected=1413
>Predicted=6085.324, Expected=5877
>Predicted=7135.720, Expected=nan

ValueError Traceback (most recent call last)
in ()
56
57 # Report performance
—> 58 mse = mean_squared_error(y, predictions)
59 rmse = sqrt(mse)
60 print('RMSE: %.3f' % rmse)

/Users/amir/Library/Enthought/Canopy/edm/envs/User/lib/python3.5/site-packages/sklearn/metrics/regression.py in mean_squared_error(y_true, y_pred, sample_weight, multioutput)
229 """
230 y_type, y_true, y_pred, multioutput = _check_reg_targets(
–> 231 y_true, y_pred, multioutput)
232 output_errors = np.average((y_true – y_pred) ** 2, axis=0,
233 weights=sample_weight)

/Users/amir/Library/Enthought/Canopy/edm/envs/User/lib/python3.5/site-packages/sklearn/metrics/regression.py in _check_reg_targets(y_true, y_pred, multioutput)
73 """
74 check_consistent_length(y_true, y_pred)
—> 75 y_true = check_array(y_true, ensure_2d=False)
76 y_pred = check_array(y_pred, ensure_2d=False)
77
```


Start Machine Learning

/Users/amir/Library/Enthought/Canopy/edm/envs/User/lib/python3.5/site-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)

/Users/amir/Library/Enthought/Canopy/edm/envs/User/lib/python3.5/site-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 l

Could you please help me with this?

Thank you so much

Amir

Jason Brownlee December 20, 2017 at 5:41 am #

REPLY

I'm not sure of the cause of your fault, sorry.

Ensure your libraries are up to date and that you have copied all of the code exactly?

Ankit Tripathi April 30, 2018 at 7:24 pm #

REPLY

The csv file contains some jargon text which should be deleted before reading the file as list.

Satyajit Pattnaik December 28, 2017 at 5:03 pm #

REPLY

Hi Jason,

In the prediction area, where you have added the observation to history and then running the loop to find the ARIMA results.
i.e.

model = ARIMA(history, order=(4,1,2))
model_fit = model.fit(disp=0)

```
output = model_fit.forecast()
yhat = output[0]
predictions.append(yhat)
obs = test[t]
history.append(obs)
```

Here, what if we append yhat to history, when i am appending yhat to history, my results are really bad, pls help..

Because as per my model, we need to predict the test data by using only the training data, so we cannot use obs to be appended in history, hope you get my point.

Jason Brownlee December 29, 2017 at 5:18

Yes, that would be called a recursive mult

more about it here:

https://machinelearningmastery.com/multi-step-tim

Satyajit Pattnaik December 29, 2017 at

Concept wise i understand what is Re

used in my earlier reply code, i have appended obs in history, so that means everytime my loop runs, it takes the existing training data + next observation, so that should work and should predict correctly, but my results are really bad..

Or do you mean to say, for this we need to plot the ACF, PACF graph in a a loop to determine the pdq values, and then run the ARIMA function on the pdq values, if that so, pls help us in finding a way to determine pdq values in loop

Jason Brownlee December 30, 2017 at 5:18 am #

REPLY

Generally using forecasts in a recursive model is challenging, you may need to get creative with your model/models.

Udi January 8, 2018 at 8:57 pm #

REPLY

Hello Jason,

Thank you for your clear and straightforward post!
I have the same question as Nick above – about the choice of differentiation interval.
In your example you set it to 12 according to the expec

However, in a more problematic case the data does not seem to imply a clear cycle (ACF and PACF graphs notwithstanding).
In my case, I've found out that setting the interval to 12 yielded better results than my default, which was 1. I can understand why choosing a small interval would be generally bad – random noise is too dominant. I have more difficulty understanding how to calibrate the ideal interval value for my data (except brute force, that is. Maybe I shouldn't calibrate? That could induce overfit. Anyway, I still should find a generally decent value).

Jason Brownlee January 9, 2018 at 5:29 am #

REPLY

If the amount of data is relatively small, co... see what works.

Udi January 8, 2018 at 9:09 pm #

And another question. I applied a grid search ... model. I fear that bias correction could be counter proc...

Do you have any insights on this subject? Would you p... Is the answer data dependent?

Jason Brownlee January 9, 2018 at 5:29 am #

REPLY

I would only perform bias correction if it lifted model skill (e.g. if the model was biased).

Nirmal Kanti Sahoo January 29, 2018 at 11:33 am #

REPLY

Hi Jason,

I have around more than 1000 of products. I have sales yearly history for last 7 years for each 1000 product. I want to develop a model by which I can predict sales amount for next 3 years.
Could you please help me how do I proceed with evaluate, prediction, validation and interpret the same.

Thanking in advance.

Jason Brownlee January 30, 2018 at 9:45 am #

REPLY

Perhaps start here:
https://machinelearningmastery.com/start-here/#tir...

Udi February 18, 2018 at 9:32 pm #

REPLY

(Noob's parallelization question, probably)

Does anybody know whether statmodels ARIMA uses multi-threading or any other kind of parallelization?

I'm trying to run an analysis based on multiple ARIMA fits on my laptop and thread parallelization increases total runtime rather than decrease it. It seems a single ARIMA fit (part of a single thread) uses several processors at once….

Thanks,
Udi

Jason Brownlee February 19, 2018 at 9:05 a

I think it is single threaded.

Raul February 28, 2018 at 3:49 am #

Hi Jason,

This is an excellent article, and it is super helpful. I had two questions for extension that may have been asked but after reading through the comments, I'm not sure if the same advice applies to me. My question is most similar to Nirmals.

1. I have a dataset with multiple "wines", each with their own historical sales data

2. This dataset has other variables that aren't just time related.

For example,

Month 1 Sales Month 2 Sales Month 3 Sales online Reviews,
Wine A
Wine B
Wine C
Wine D

I'm wondering if there's an extension of this Time Series model that can take into account other variables and other instances of historical data. Let me know if I should clarify.

Thank you for taking the time to clarify.

Jason Brownlee February 28, 2018 at 6:10 am #

REPLY

Yes, perhaps you could model each series separately, or groups of series or all together, or all 3 approaches and ensemble the results.

Also, I'd recommend moving on to other models, such as ML or even MLP models.

I hope to cover this topic in more detail soon.

Raul February 28, 2018 at 7:01 am #

REPLY

Thanks for the quick reply.

I'll try out your first method.
But do you have any good ML/MLP models/tut
have a nice article on multivariate time series h
https://machinelearningmastery.com/multivaria

I haven't read through it yet, but I think it only t
multiple variables.

I think it would be interesting to find out how to
#1. one instance of historical data (Wine A's la
#2. one instance of non-historical variables (W

To me, # 1's output is a "variable" to be used ir
think that's the right way of going about things'

Thanks for being so responsive!

Jason Brownlee March 1, 2018 at 6:01 am #

REPLY

I hope to cover this topic in more detail soon – maybe write a book on the topic working one dataset from many different directions.

Anchal April 6, 2018 at 4:20 am #

REPLY

Thank you Jason! for really making ML practitioners like me being awesome in ML.

In this blog you have mentioned that the results suggest that what little autocorrelation is present in the time series has been captured by the model.

What will be the next steps if there is high autocorrelation?

Thanks in advance.

Jason Brownlee April 6, 2018 at 6:35 am #

REPLY

Great question!

Some ideas:
– Perhaps try alternate models.
– Perhaps improve the data preparation.
– Perhaps try downstream models to further correct the forecast.

Ankit Tripathi May 10, 2018 at 5:55 pm #

REPLY

Hey Jason,

That was a very well informed article! I am trying to for
model since weekly data is hard to forecast. Also shou
changed to “weeks in month=4” to accommodate for w

Thanks

Jason Brownlee May 11, 2018 at 6:34 am #

My best tips are to try lots of data preparation techniques and tune the ARIMA using a grid search.

Maria Dossin May 26, 2018 at 9:47 am #

REPLY

Hi Jason,
Thank you for an amazing tutorial!
Just one quick question: can you please provide solution for the first scenario (not rolling forward forecast):
– Load the model and use it to forecast the next 12 months. The forecast beyond the first one or two months will quickly start to degrade in skill.
Thank you~

Jason Brownlee May 27, 2018 at 6:43 am #

REPLY

Yes, this will help:
https://machinelearningmastery.com/make-sample-forecasts-arima-python/

Piyasi Choudhury June 2, 2018 at 11:38 am #

REPLY

Hi Jason

My time series data is non stationary as well and I tried upto 3rd degree of differentiation after which I cant proceed any more as the data is exhausted. It has 3 calculated points and on doing Fuller test for checking stationarity, it errors out “maxlag should be < nobs". What can I do here?

Jason Brownlee June 3, 2018 at 6:20 am #

REPLY

Sounds like you might be running out of d
observations?

KACEM June 3, 2018 at 11:54 am #

Hello, thank you.

i wonder if we can say that python give good forecasts

To forecast sales we have to base ourselves in historic
factors that complexify the calculation of our forecast li
… How can we consider these perturbation in our forecasts to ensure good value of accuracy.

Thank you for answering.

Jason Brownlee June 4, 2018 at 6:21 am #

REPLY

What is Aperia?

Additional elements could be used as exogenous variables in the model.

cathy June 23, 2018 at 10:00 am #

REPLY

Jason,

The dataset I have requires to double difference in order to make it stationary, so i used:

diff1 = difference(history, 12)
diff2 = difference(diff1, 12),

it worked and made it stationary with ADF test, however, how do I reverse it back please?

Thank you

Jason Brownlee June 24, 2018 at 7:27 am #

REPLY

Add the values back. It requires that you keep the original and the first diffed data.

Cathy June 24, 2018 at 10:28 am #

REPLY

Thanks, so would the function look like this?

yhat = inverse_difference(diff1, yhat, interval)
yhat = inverse_difference(history, yhat, interva

Thank you!

Jason Brownlee June 25, 2018 at 6

Correct.

Rui July 27, 2018 at 4:11 am #

First of all, this is a great article, well written and detailed. I have a question regarding on the use of the test set on all of the analysis. What about putting this model in production? You wouldn't have the 'test[i]' (or 'y[i]') for each iteration to add to the list 'history' to have a truly generalized prediction.
My point is that that instead of adding the values from the test set ('y[i]', 'test[i]') you'd rather add the predictions being made to the training set in order to do a true random walk.
Thanks again for the resource and all the help putting out this content.

Jason Brownlee July 27, 2018 at 5:57 am #

REPLY

A final model will be created by training it on all available data and then using it to make predictions in the future.

You can learn more about final models here:
https://machinelearningmastery.com/train-final-machine-learning-model/

You can learn more about making out of sample predictions here:
https://machinelearningmastery.com/make-sample-forecasts-arima-python/

James Lucas July 27, 2018 at 2:21 pm #

I keep getting errors

TypeError Traceback (most recent call last)

in ()

9 obs = test[i]

10 history.append(obs)

—> 11 print(‘>Predicted=%.3f, Expected=%3.f’ % (yhat, obs))

TypeError: must be real number, not ellipsis

IE: # predict

yhat = …

those 3 dots (ellipsis) are throwing errors.

The code has … throughout. Any idea how to fix this e

Jason Brownlee July 28, 2018 at 6:28 am #

The code with “…” is just example code, i

Perhaps a more careful read of the tutorial is in orc

Ben Kesby August 3, 2018 at 2:53 am #

I’m having an issue with the PACF plot for the residuals in part 6.3 Plotting The Residuals. I have followed your code exactly with the ACF plot resulting in a perfect match of the one displayed here tower the PACF is plotting values nearing 8 at around 36 – 38 lags. Any idea what might be causing this?

JP August 24, 2018 at 8:17 pm #

REPLY

Hi Jason,

Thanks so much for this tutorial. One thing though. As series.from_csv is depricated, the date format gets lost when opening the dataset, for exemple when trying to generate the seasonal line plots. Are you aware of a workaround that would keep the date format while using read_csv instead of from_csv?

Thanks!

Jason Brownlee August 25, 2018 at 5:48 am #

REPLY

Yes you can use pandas.read_csv() with the same arguments.

Alexandra September 12, 2018 at 8:40 pm #

REPLY

How can I improve the readability of Seasonal Per Year Line Plots (especially when it comes to axes) ? I would be grateful for your help.

Jason Brownlee September 13, 2018 at 8:01 am #

REPLY

Perhaps create one plot per figure?

Alexandra September 13, 2018 at 4:44 pm

I'd rather have a comparison betweer

Jason Brownlee September 14, 20

Perhaps plot them all on the sam
Perhaps calculate the different and plot tha

Chintan September 18, 2018 at 2:16 am #

REPLY

Hi Jason,
I was able to predict and chart looks nice. Just wondering how can we predict for the future, I mean to say if I wanted to see for next month prediction how are we able to do that?

Thanks,
Chintan

Jason Brownlee September 18, 2018 at 6:20 am #

REPLY

I show how here:
https://machinelearningmastery.com/make-sample-forecasts-arima-python/

Pranay October 9, 2018 at 9:26 pm #

REPLY

Hi Jason, I deal with daily set data. What is th
not throwing best_arima out when I set months_in_yea

Jason Brownlee October 10, 2018 at 6:09 am #

REPLY

Why are you making that change? I don't follow?

Patrick October 30, 2018 at 2:07 am #

REPLY

Dear Jason,

thank you so much for your tutorial. I would like to appl
The time interval is not monthly based, as in your exan
collection is about every three seconds.

Now I would like to ask some questions.

Do you think that this model could work equally well ?

What should I put in place of 'months_in_year' ? For n
dataset.

Is it normal that with 2000+ points the matrix analysis r
taking ages ? (e.g. one triplet after 30 min) If yes, how

Thank you again,
Best wishes,
Patrick.

Jason Brownlee October 30, 2018 at 6:08 am #

REPLY

Try it and see.

2K points may be too many for the method, perhaps try reducing to a few hundred at max.

Markus December 25, 2018 at 12:20 am #

REPLY

Hi

Where does the column 'A'comes from by the code above which is:

groups = series['1964':'1970'].groupby(TimeGrouper('A'))
y

And printing out groups contains only the first 6 months of each year, why?

Jason Brownlee December 25, 2018 at 7:23 am #

REPLY

Right there:
https://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases

Markus December 25, 2018 at 3:10 am #

REPLY

Hi

What would be wrong if instead of defining the difference function by yourself, we would pass over d with the value of 12:

model = ARIMA(diff, order=(0,12,1))

Wouldn't that be the same?

Thanks!

Jason Brownlee December 25, 2018 at 7:24

It should be.

Anthony January 10, 2019 at 10:33 pm #

REPLY

Hi Jeson in running your code I get this error:
raise ValueError("The computed initial MA coefficients are not "
ValueError: The computed initial MA coefficients are not invertible
You should induce invertibility, choose a different model order, or you can pass your own start_params.
How can I solve it? Thank you

Jason Brownlee January 11, 2019 at 7:47 am #

REPLY

Sorry to hear that.

Perhaps confirm your statsmodels and other libraries are up to date?
Perhaps confirm that you copied all of the code in order?
Perhaps try an alternative model configuration?

Anthony January 15, 2019 at 9:50 pm #

Ok I changed ARIMA parameters and now it works thank you!

Jason Brownlee January 16, 2019 at 5:47 am #

REPLY

Glad it hear it!

aravind January 12, 2019 at 9:29 pm #

REPLY

This is my data: like….

Name Month Qty Unit
Wire Rods Total 2007-JAN 93798 t
Wire Rods Total 2007-FEB 86621 t
Wire Rods Total 2007-MAR 93118 t

My code is :

```
import pandas as pd
from sklearn.metrics import mean_squared_error
from math import sqrt
# load data
path_to_file = "C:/Users\ARAVIND\Desktop\jupyter not
data = pd.read_csv(path_to_file, encoding='utf-8')
# prepare data
X = data.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
# predict
yhat = history[-1]
predictions.append(yhat)
# observation
obs = test[i]
history.append(obs)
print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)
```

when i run this command it shows “Value error: could not convert string to float”… so could anyone tell how to convert string to float according to my dataset.. i want to convert the columns which is “Name, Month and Unit” to float.

Jason Brownlee January 13, 2019 at 5:41 am #

REPLY

Perhaps remove the text and date data?

aravind January 14, 2019 at 3:36 pm #

REPLY

ok sir.. thank you

Mridul March 20, 2019 at 5:41 pm #

Hi Jason, Thanks for this amazing tutorial.

However, I get the below error when I am trying to run

ValueError: The computed initial MA coefficients are no
You should induce invertibility, choose a different mode
pass your own start_params.

Mridul March 20, 2019 at 5:45 pm #

REPLY

I just saw that this has already been answered and it worked when I followed that answer. Thanks!

Jason Brownlee March 21, 2019 at 7:59 am #

REPLY

Perhaps try a different configuration of the q/d/p variables for the model?

Kaws March 20, 2019 at 7:42 pm #

REPLY

This is really a helpful tutorial. Thank you Jason!!

And I have a small question. I got TypeError: a float is required error after i executed this code –

```
history = [x for x in train]
predictions = list()
for i in range(len(test)):
# predict
yhat = …
predictions.append(yhat)
# observation
obs = test[i]
history.append(obs)
print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
```

Can you help me out?

Jason Brownlee March 21, 2019 at 8:02 am #

Perhaps confirm that you have loaded the

Varun April 15, 2019 at 4:23 pm #

Hi Jason,

What should be the approach when we need to provide long term forecasts ~ 12 months with exogenous variables using a technique like ARIMAX? Should we forecast the covariates and then add it in the model?

Regards,
Varun

Jason Brownlee April 16, 2019 at 6:45 am #

REPLY

Perhaps you can frame your model to predict +12 months given only the observations available?

Varun April 26, 2019 at 7:22 pm #

REPLY

Hi Jason,

Just to be clear, if I add regressors and train the model, I would require future values right? E.g. xreg argument in auto.arima etc. How can I forecast +12 months without utilizing regressors that I have used for training?

Start Machine Learning

Jason Brownlee April 27, 2019 at 6:28 am # REPLY

You could train a new productive model that only requires t-12 data to make predictions.

Park1 April 24, 2019 at 8:14 pm # REPLY

Hi!

It's amazing, thank you!

Could you give all files of this project. Cannot build it on my own

Jason Brownlee April 25, 2019 at 8:10 am #

You can copy them from the article, here's
https://machinelearningmastery.com/faq/single-faq

Prince Tiwari May 15, 2019 at 9:31 pm #

Hi!

It's amazing, thank you!

Actually i want develop a model which determine how many calls we can expect to come into our call center on a daily basis ?

Jason Brownlee May 16, 2019 at 6:31 am # REPLY

Sounds great.

Perhaps start here:
https://machinelearningmastery.com/start-here/#timeseries

Prince Tiwari May 22, 2019 at 10:12 pm # REPLY

Hi Jason Brownlee ,

Thank you so much for responded, i need to one more help

Could you please explain how to use Seasonality in AR

Start Machine Learning

Thanks

REPLY

Jason Brownlee May 23, 2019 at 6:03 am #

You can use the blog search box.

Here is an example:
https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/

Here is another:
https://machinelearningmastery.com/how-to-grid-search-sarima-model-hyperparameters-for-time-series-forecasting-in-python/

Adi June 27, 2019 at 2:37 am #

Hi Jason,

That's a very helpful article. My time series is a daily P
bought everyday in a walmart). There are missing date
forecast is the bottles of pepsi sold on each day for the
algorithm?

Thanks.

REPLY

Jason Brownlee June 27, 2019 at 7:59 am #

Probably a linear model like SARIMA or ETS.

I have some suggestions here:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

REPLY

Nagaraj July 9, 2019 at 4:21 pm #

Hi Jason,
I'm getting the error like this, what does this error mean?

TypeError Traceback (most recent call last)
in
test = …
predictions = …
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print('RMSE: %.3f' % rmse)

TypeError: Expected sequence or array-like, got

REPLY

Jason Brownlee July 10, 2019 at 8:03 am #

That is surprising, did you copy of all of the code exactly?

REPLY

Rohit Singh Adhikari August 2, 2019 at 3:01 am #

Hi Jason,
I need to build a forecast having addition predictor, wh

Jason Brownlee August 2, 2019 at 6:54 am #

Perhaps follow the process outlined in the

Ray August 7, 2019 at 9:58 am #

Hi Jason, nice article.
I have a question in the last code sample for validation.

1) What is the purpose of making the first prediction?
load model
#make first prediction

2) I remove the above two sections of the code and got the exact result

Does mean only order (0,0,1) and Bias(165.904728) matters and there is no need to save and load the model?

REPLY

Jason Brownlee August 7, 2019 at 2:20 pm #

It is an example of how we might use bias correction, in general.

REPLY

Kuba August 16, 2019 at 9:53 pm #

Hello,
I got exactly same results of the Augmented Dickey-Fu
It has a lot of spikes from lag 50 to 80, one even reach

Any ideas or previous experience why it can look that strange?
Thank you for sharing your knowledge with us.

REPLY

Jason Brownlee August 17, 2019 at 5:44 am #

They changed the plot recently.

Try scaling the number of time steps way down in the plot.

Jamie August 26, 2019 at 3:22 pm #

I have tried with earnest to work through your
One being this error. This could be the whole bone to r
using your code alone. I had to tinker with it – see expl

series = Series.from_csv('dataset.csv')
AttributeError: type object 'Series' has no attribute 'fror

Which I resloved by implementing

panda import pd
series = pd.read_csv('dataset.csv')
series.iloc[0]

Once I got over that hurdle – I then ran into this hurdle.

dataset = dataset.astype('float32')
ValueError: could not convert string to float: '1964-01'

More than like its something I am doing wrong.

Would it be at all possible to have access to the full code? My attempts in copying and pasting obviously are not helping me.

I think I have possibly left some intrinsic part of the code out. Or I have totally confused myself.

REPLY

Jason Brownlee August 27, 2019 at 6:27 am #

No problem, changed to:

```
from pandas import read_csv
series = read_csv('champagne.csv', header=0, index_col=0)
...
```

I have updated all examples in the tutorial.

Jamie August 27, 2019 at 9:42 am #

REPLY

That simple – gees… I will try that. Thanks for taking the time to answer my problem. Is your book available on amazon?

Jason Brownlee August 27, 2019 at 2:08 pm #

REPLY

My books are not on Amazon, only on my website, I explain why here (they take a massive cut):
https://machinelearningmastery.com/faq/single-faq/why-arent-your-books-on-amazon

Jamie August 27, 2019 at 3:59 pm #

It worked like a charm. Thanks.

Another issue that I have found is with the

from pandas import TimeGrouper
from pandas import DataFrame

It seems that pandas don't support TimeG
https://pandas.pydata.org/pandas-docs/sta
Removed the previously deprecated TimeGrouper (GH16942)
Removed the previously deprecated DataFrame.reindex_axis and Series.reindex_axis (GH17842)

Jason Brownlee August 28, 2019 at 6:29 am #

Well done!

You can use:

```
...
groups = series['1964':'1970'].groupby(Grouper(freq='A'))
```

I updated the example, thanks!

Arjun December 5, 2019 at 4:42 pm #

REPLY

Hi jason,
I was wondering if you have used fbprophet for sales prediction. We were fetching data directly from postgresql and we seem to be running into an error

Out of bounds nanosecond timestamp: 1-08-11 00:00:00

This seems to be something related with pandas version compatibility.
Could you please look into it and try to find the problem behind it?

Jason Brownlee December 6, 2019 at 5:12 am #

REPLY

I have not, sorry.

Mitra February 6, 2020 at 7:40 pm #

hey Jason,

I'm running the
series=read_csv(r'D:\industrial engineering\Thesis\mo
code and what I'm getting is a data frame, not a Series

Jason Brownlee February 7, 2020 at 8:13 am

Try adding squeeze=True argument.

Aviral Kumar March 17, 2020 at 1:56 am #

REPLY

dear sir
I am running following code in a rolling window framework however, i am not able to see results that come from the analysis. It displays only one value. Can you please let me know what and where i need to fix so that i can get those results:

```
##Entropy
from entropy import *
import numpy as np

np.random.seed(1234567)
x = np.random.rand(3000)
n = len(x)
result = list()
block = 250
for a in range(1, n-block+1):
DATA = x[a:a+249]
results = perm_entropy(DATA, order=3, normalize=True)
```

```
result.append(results)
return Series(result)
```

Jason Brownlee March 17, 2020 at 8:17 am #

REPLY

This is a common question that I answer here:
https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

Leave a Reply

Name (required)

Email (will not be published) (required)

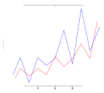
Website

SUBMIT COMMENT

Welcome!
My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:


How to Create an ARIMA Model for Time Series Forecasting in Python


How to Convert a Time Series to a Supervised Learning Problem in Python


Time Series Forecasting as Supervised Learning


11 Classical Time Series Forecasting Methods in Py


How To Backtest Machine Learning Models for Time

Start Machine Learning