STAT902 Forecasting Competition

By

Name: Jinyang Li
ID: 20788352

TABLE OF CONTENTS

I. SCENARIO 1: HYDROLOGICAL FORECAST

1. Model Specification

For this problem, we use $SARIMA(1,0,2)\times(1,1,2)_{12}$ to forecast the 1-month to 24-month ahead forecast of the monthly resolution of the level of body water. The definition for the SARIMA model is as below:

- SARIMA Model:

  x_t is said to follow an SARIMA (Seasonal Autoregressive Integrated Moving Average) model of orders p, d, q, P, D, Q and seasonal periods s if

  $$\Phi_P(B^s)\phi(B)(1-B^s)^D(1-B)^d x_t = \Theta(B^s)\theta(B)w_t$$

  This is abbreviated as x_t follows $SARIMA(p,d,q)\times(P,D,Q)_s$

2. Model Selection and Estimation

The problem is this scenario is to forecast the level change of body water in monthly frequency. Intuitively, we may guess to use an $ARIMA$ model as it can captures a suite of different standard temporal structures in time series data. We know that the human body may have periodic adjustment on a yearly basis, we may further guess to use a $SARIMA$ model for better fitting and prediction.

Fig 1.1 shows the monthly resolution of the level of body water. It shows some extend of non-stationarity. By the ADF test in the notebook for scenario 1, we get the p-value of 0.079825, which indicating a non-stationarity in significance level of both 1% and 5%. Therefore, it confirms our use of the differencing in $SARIMA$ model.

Fig 1.2 shows the ACF of the data, in which we notice a seasonal effect of lag 12. This confirms out guess on seasonal differencing. The autocorrelation that remains in the residuals of the seasonally differenced data is then modeled using $ARMA(p,q)$ models.

We have tried to fit the data with $SARIMA$ model using multiple parameter combinations (as below) and chosen the best by the lowest AIC/BIC.

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMA: (0, 0, 1) x (0, 0, 1, 12)
SARIMA: (0, 0, 1) x (0, 0, 2, 12)
SARIMA: (0, 0, 2) x (0, 1, 0, 12)
SARIMA: (0, 0, 2) x (0, 1, 1, 12)
```

- In Python, the best parameter combination is $SARIMA(1,0,2)\times(1,1,2)_{12}$
- In R, by using the automatically selection function, the best parameter combination is $SARIMA(2,0,0)\times(1,1,0)_{12}$

In this problem, we fit the data by $SARIMA(1,0,2)\times(1,1,2)_{12}$ model.

Table 1.1 shows the summary of our fitted $SARIMA(1,0,2)\times(1,1,2)_{12}$ model.

3. Model Diagnostics

Standard model diagnostics can be found in **Fig 1.3**.

- The standardized residual looks stationary.
- By the histogram and Q-Q plot, the residual roughly follows normal distribution.
- By the ACF of residuals, they are uncorrelated.

Therefore, the residual is white noises and our time series model fit the data quite well.

4. Model Forecast

In this scenario, we want to forecast the level of body water in the future 24 months, as well as the 95% prediction and confidence bands.

Fig 1.5 is the visualization of the forecast. The right end of the whole data and the forecast is plotted for closer view. We can see a wider prediction interval than the confidence band. The reason from STAT850 is summarized as follow:

The difference between a prediction interval and a confidence interval is the standard error.

Confidence intervals

- tell you about how well you have determined the mean. Assume that the data really are randomly sampled from a pre-determined distribution. If you repeat many times and calculate a confidence interval of the mean from each sample, you'd expect about 95 % of those intervals to include the true value of the population mean. The key point is that the confidence interval tells you about the likely location of the true population parameter.
- The standard error for a confidence interval on the mean takes into account the uncertainty due to sampling.

Prediction intervals

- tell you where you can expect to see the next data point sampled. Assume that the data really are randomly sampled from a Gaussian distribution. Collect a sample of data and calculate a prediction interval. Then sample one more value from the population. If you do repeat many times, you'd expect that next value to lie within that prediction interval in 95% of the samples. The key point is that the prediction interval tells you about the distribution of values, not the uncertainty in determining the population mean.
- The standard error for a prediction interval on an individual observation takes into account the uncertainty due to sampling like above, but also takes into account the

variability of the individuals around the predicted mean. The standard error for the prediction interval will be wider than for the confidence interval and hence the prediction interval will be wider than the confidence interval.

II. SCENARIO 2: FINANCIAL RISK FORECAST

1. Model Specification

For this scenario, we need to forecast (lower) 15% quantiles 10 steps ahead for each stock price series. An example of stock price process is shown in **Fig 2.1**.

By financial market knowledge, we know the stock price is uncorrelated (**Fig 2.2**) but may still be serially dependent due to a dynamic conditional variance process. A time series exhibiting conditional heteroscedasticity, or autocorrelation in the squared series, is said to have *autoregressive conditional heteroscedastic* (ARCH) effects. As indicated by **Fig 2.1**, one can observe volatility clustering in some extend.

We use the Engle's LM test to test the ARCH effect in each stock series, which is a Lagrange multiplier test to assess the significance of ARCH effects. By the Engle's LM test, the p-values of all the series are less than 1%, therefore the null hypothesis that there is no ARCH effect can be rejected at 1% significance level, meaning that the ARCH affects are quite significant in the daily log returns.

Therefore, we may guess using GARCH type of model to fit the data. The definition of a GARCH model is as following:

Let w_t be a unit variance strong white noise process. x_t is said to follow a (strong) Generalized Autoregressive Conditionally Heteroscedastic model of orders p and q ($GARCH(p,q)$) if

$$x_t=\sigma_t w_t \qquad \sigma_t^2=\omega+\sum_{j=1}^{p}\alpha_j x_{t-j}^2+\sum_{l=1}^{q}\beta_j \sigma_{t-l}^2$$

5. Model Selection and Diagnostics

- Use $GARCH(1,1)$

We first fit the data using simple $GARCH(1,1)$ to see if any model adjustment is needed. **Table 2.1** shows the summary of the fitted $GARCH(1,1)$ model for stock 17. The fitted models for other series are similar so here we just talk about a specified one.

- The Ljung–Box test for standardized residuals looks good, but there is some evidence of serial correlation in standardized squared residuals.
- The ARCH LM test shows that we have eliminated the ARCH effect by $GARCH(1,1)$
- Nyblom test, which tests for coefficient stability (structural change), shows no evidence for unstable parameters.

- Sign Bias test, which examines the leverage effects, shows no or weak evidence of asymmetric effects.
- Adjusted Pearson Goodness-of-fit test, which tests for distribution goodness-of-fit, shows that the normal distribution in the model cannot be rejected.

By the above model diagnostics, fitting the stock series by $GARCH(1,1)$ is a good choice.

- Use $EGARCH(1,1)$

  We know that in the financial market, the stock is mostly fat-tailed distributed and there may exist leverage effect. So sometimes GARCH model may not be sufficient to capture all the features in stock time series.

  As a result, I also fit our data using $EGARCH(1,1)$ with student-t distribution whose result can be found in **Table 2.1**.

  I have also tested other asymmetric univariate GARCH model for the stock series. The result is similar to the $EGARCH$ one.

After comparing the model summaries for different models, we use $GARCH(1,1)$ with normal distribution as the final choice. The leverage effect is not significant as well as the fat-tail phenomenon. Therefore, the $GARCH(1,1)$ model is simple but already have the full capacity to fit the data.

6. Model Forecast

Multi-period forecasts can be produced for GARCH-type models using forward recursion. Some models, like EGARCH, that are non-linear in the sense that they do not normally have analytically tractable multi-period forecasts available.

There are three methods for forecasting using ARCH packages in Python:

- Analytical: multi-step analytical forecasts are only available for model which are linear in the square of the residual, such as GARCH or HARCH.
- Simulation: simulation-based forecasts are always available for any horizon, and is used mostly for horizons lager than 1 since the first out-of-sample forecast from an ARCH-type model is always fixed.
- Bootstrap: bootstrap-based forecasts are similar to simulation-based forecasts except that they make use of standardized residuals from the actual data used in estimation rather than assuming a specific distribution.

In the Notebook for scenario 2, we have implemented all the above three method for forecasting under $GARCH(1,1)$ and $EGARCH(1,1)$. Non-convergence issue exists when fitting some asymmetric GARCH model. We multiply the log-returns by 100 first and scale back after fitting. The situation is resolved for most cases but still remain in some certain stocks when fitting $EGARCH(1,1)$.

For output of our forecast, we choose the forecast result by using $GARCH(1,1)$ and applying the Bootstrap method to take the advantage of the non-parametric distribution of the actual data.

III. SCENARIO 3&4: IMPUTATION AND MULTIVARIATE TIME SERIES FORECASTING

As for scenario 3&4, we have data for monthly beer production, car production, steel production, gas consumption and electricity consumption. The imputation and forecasting will forecast on the month beer production.

1. Imputation

The scenario 3 asks for imputing (predicting) the missing values. There are 30 missing values in beer production from 1972-09 to 1975-02.

Below is a short summary about the data.

	Beer	*Car*	*Steel*	*Gas*	*Electricity*	*Temperature*
Start	1956-01	1961-07	1956-01	1956-01	1956-01	1943-11
End	1992-03	1992-03	1992-03	1992-03	1992-03	1992-03
# of values	435	369	435	435	435	581
# of missing values	30	NA	NA	NA	NA	NA
Total	435	369	435	435	435	581

We merged the 6 categories using left join on time index. The final dataset merged is of shape 435×6 .

Some basic imputation methods are as below:

- SoftImpute9: This method uses matrix completion via iterative soft-thresholded Singular Value Decomposition (SVD) to impute missing values.
- **KNN**: This method uses k-nearest neighbor to find similar samples and imputed unobserved data by weighted average of similar observations.
- Cubic Spline: This method uses cubic spline to interpolate each feature at different time steps.

- **MICE**: The Multiple Imputation by Chained Equations (MICE) method is widely used in practice, which uses chain equations to create multiple imputations for variables of different types.
- MF: Using matrix factorization (MF) to fill the missing items in the incomplete matrix by factorizing the matrix into two low-rank matrices.
- PCA: Imputing the missing values with the principal component analysis (PCA) model.
- MissForest: This is a non-parametric imputation method which uses random forests trained on the observed values to predict the missing values.

In Notebook for scenario 3, we have implemented the KNN and MICE methods, as well as a deep learning method called 'Datawig' which our final imputation output is based on.

Datawig trains machine learning models to impute missing values in tables. It has the advantages of fully making use of the information in the data and learns all parameters of the entire imputation pipeline automatically in an end-to-end fashion. Details on the underlying model can be found in Biessmann, Salinas et al. 2018.

By using Datawig, we imputed the missing value for the beer production as well as the car production value from 1956-01 to 1961-06 for the forecasting in scenario 4.

This deep learning model is evaluated using mean square error and r2 score. It has a MSE of 198.71 and r2 score of 0.84, which suggest a good regression on our data. One should notice that **we don't have prediction intervals for imputation** as the result of applying deep learning method.

2. Forecasting

Intuitively, we may think that the beer production can be correlated to the production of car, steel, the consumption of gas and electricity as well as the local temperature. Those are factors affecting the raw materials to produce beer, the transportation for sales, the energy to supply boiling, fermentation and filtration in the factories. The above is also verified by looking at the CCF plot between Beer and other categories as shown in **Fig 4.1**.

As for forecasting, we deploy the Long Short-Term Memory (LSTM) recurrent neural networks for multivariate time series forecasting. One can find more information about LSTM from wiki.

i. Reason we use LSTM
   - vs traditional time series model: ARMAs and ARIMAs are particularly simple models which are essentially linear update models plus some noise thrown in. With nonlinear activation functions, neural networks are approximations to nonlinear functions. LSTMs are thus essentially a nonlinear timeseries model, where the nonlinearity is learned from the data.
   - vs other machine learning algorithms: LSTM recurrent neural networks are capable of automatically learning features from sequence data, support multiple-variate data, and can output a variable length sequences that can be used for multi-step forecasting.

ii. Data preparation

Below is a screenshot of the data we got from the imputation step.

Date	Beer	Car	Steel	Gas	Electricity	Temp
1956-01-01	93.2	12700.116925	196.9	1709	1254	25.1
1956-02-01	96.0	12574.354195	192.1	1646	1290	25.3
1956-03-01	95.2	13050.102235	201.8	1794	1379	24.9
1956-04-01	77.1	11604.703762	186.9	1878	1346	23.9
1956-05-01	70.9	13700.668520	218.0	2173	1535	19.4

MAKE EACH CATEGORY DATA STATIOANRY

The Dickey Fuller test is one of the most popular statistical tests. It can be used to determine the presence of unit root in the series, and hence help us understand if the series is stationary or not. The null and alternate hypothesis of this test are:

- Null Hypothesis: The series has a unit root (value of a =1)
- Alternate Hypothesis: The series has no unit root.

If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary (we will understand more about difference stationary in the next section).

By **Fig 4.2, Fig 4.3** and Dickey Fuller test in the Notebook for scenario 4, some of our variables are non-stationary, namely ‘Steel’, ‘Gas’ and ‘Electricity’.

Therefore, the transformation to stationarity is needed for those variables. Typical technologies for stationarity transformation include differencing and log transformation. Details can be found in the Notebook for scenario 4.

DATA NORMALIZATION

For supervise learning, data needs to be normalized before feeding into the network. Which is a part of data preparation for training. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges which is the case in scenario 4.

Because the different features in our scenario do not have similar ranges of values, the gradients may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global/local minimum. To overcome the model learning problem, we normalize the data. We make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly.

iii. Model Specification

We define the LSTM with 50 neurons in the first hidden layer and 24 neurons in the output layer for predicting 24 steps ahead. The input shape will be 24 time-steps with 8 features.

We use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent.

iv. Model Evaluation

we keep track of both the training and test loss during training by setting the validation data argument in the fit() function. At the end of the run both the training and test loss are plotted as **Fig 4.4.**

We can see quick drop of both training and test losses indicating a proper hyperparameters tuning without overfitting problem.

The **prediction interval is not applicable** for the LSTM network. Instead we just use a $GARCH(1,1)$ model to plot a prediction interval for illustration purpose as shown in **Fig 4.5**.

IV. FIGURE

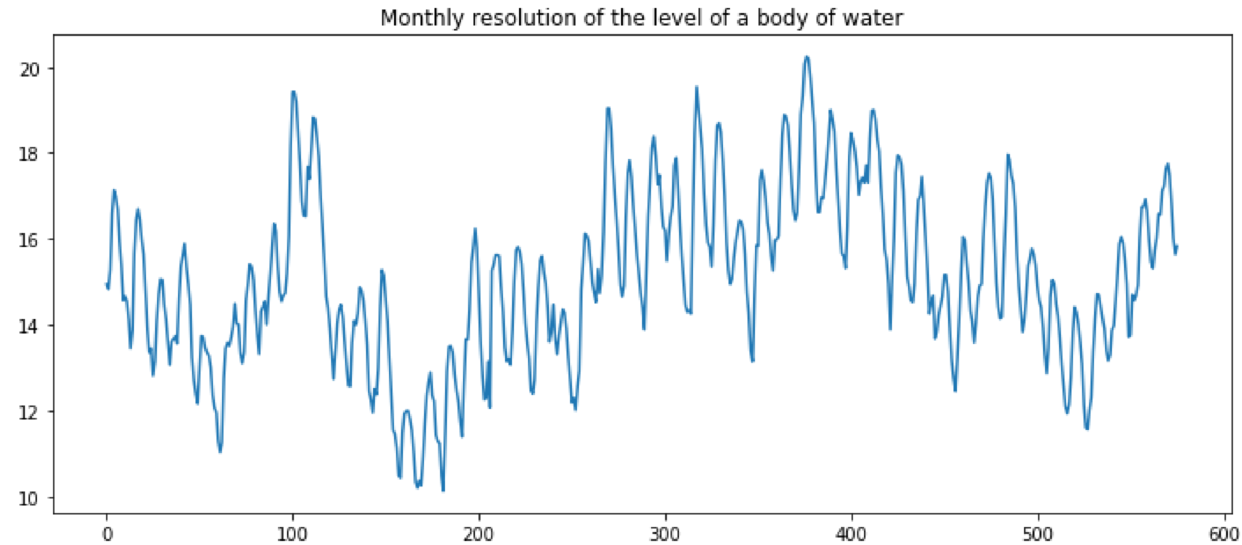


Fig. 1.1

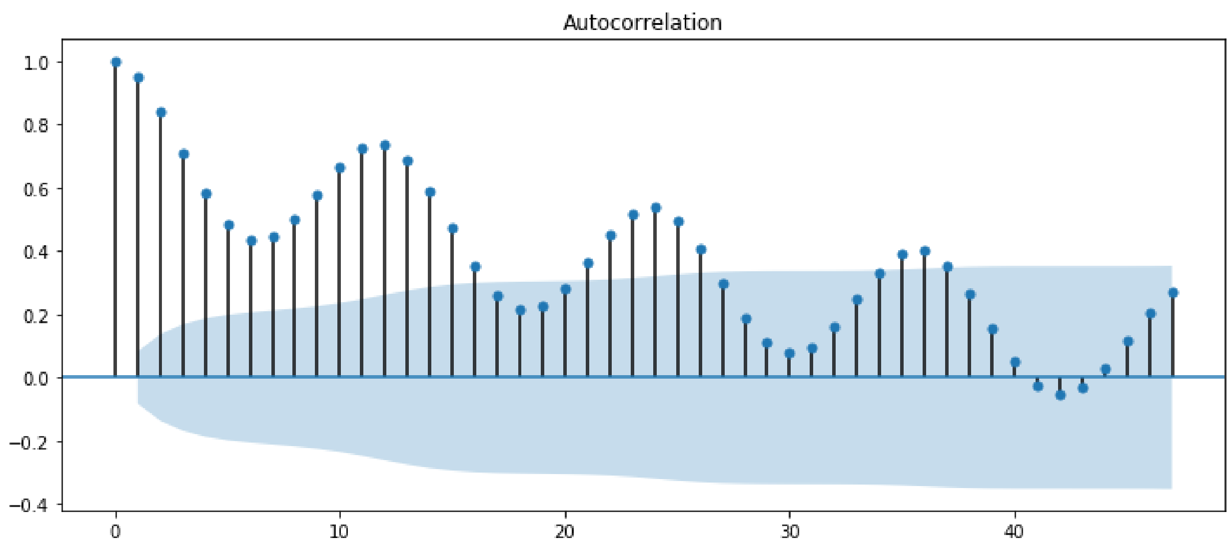
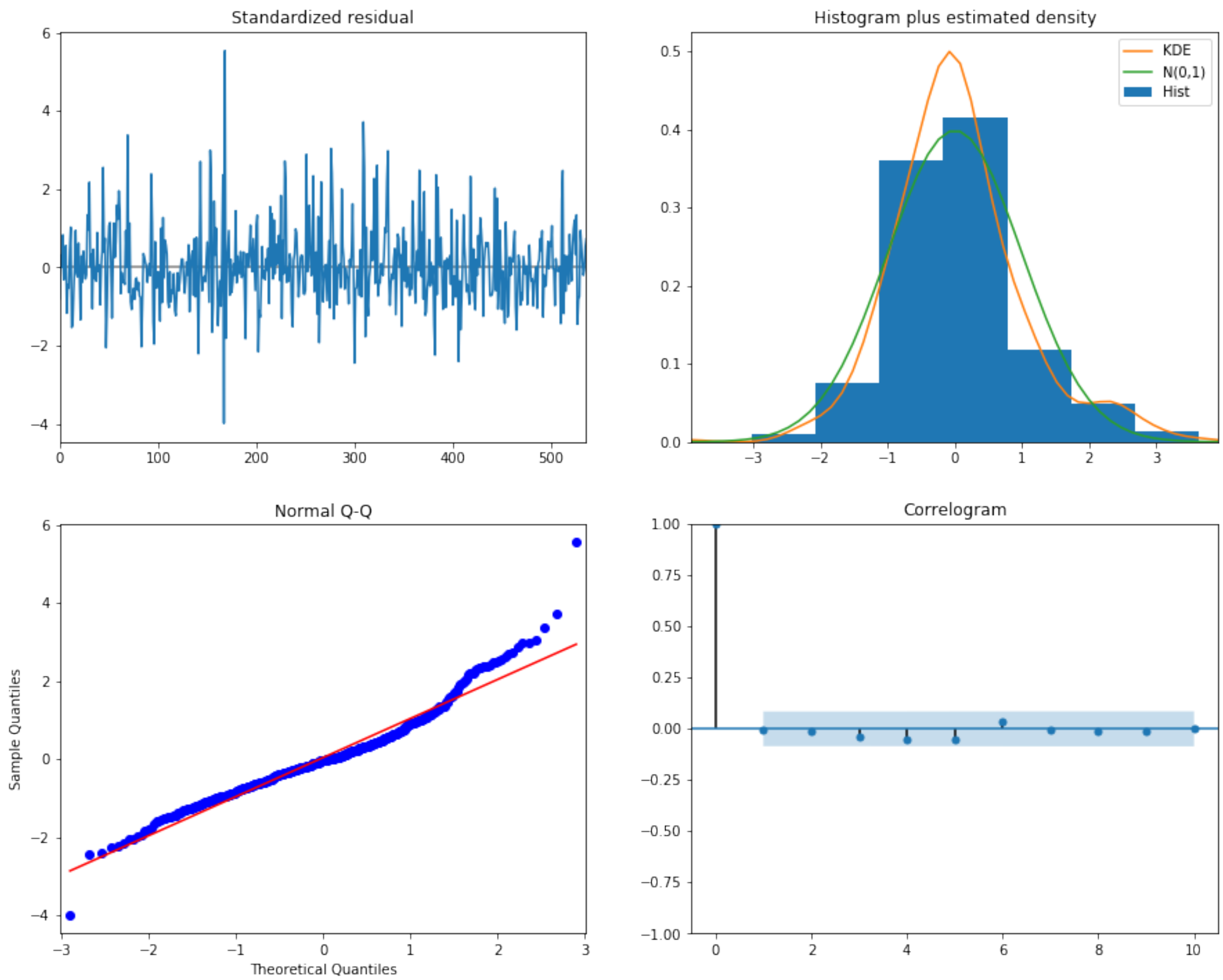
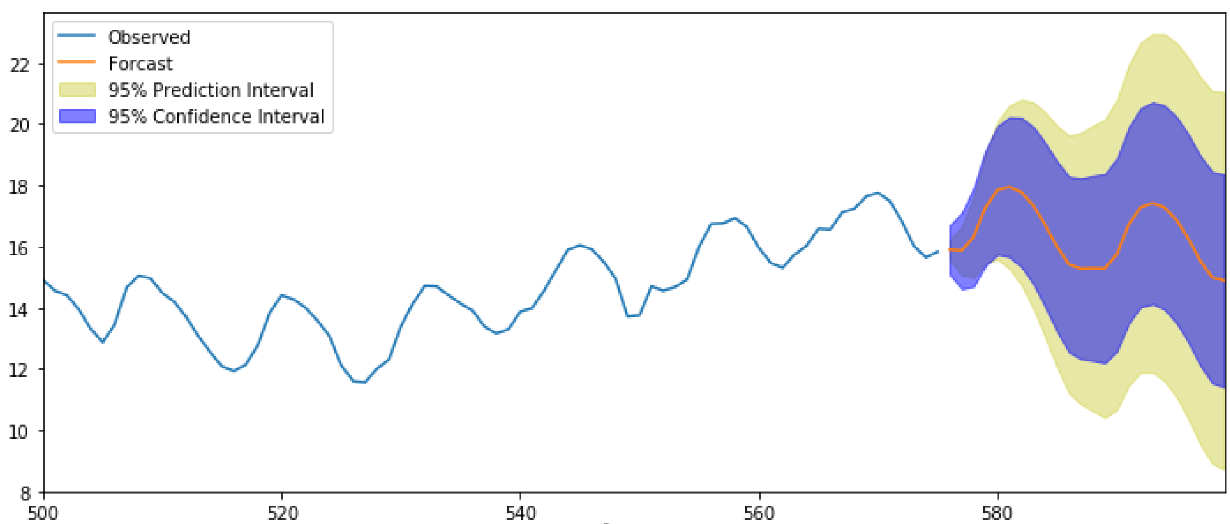


Fig. 1.2



Standardized residual
Histogram plus estimated density
KDE
N(0,1)
Hist
Normal Q-Q
Sample Quantiles
Theoretical Quantiles
Correlogram

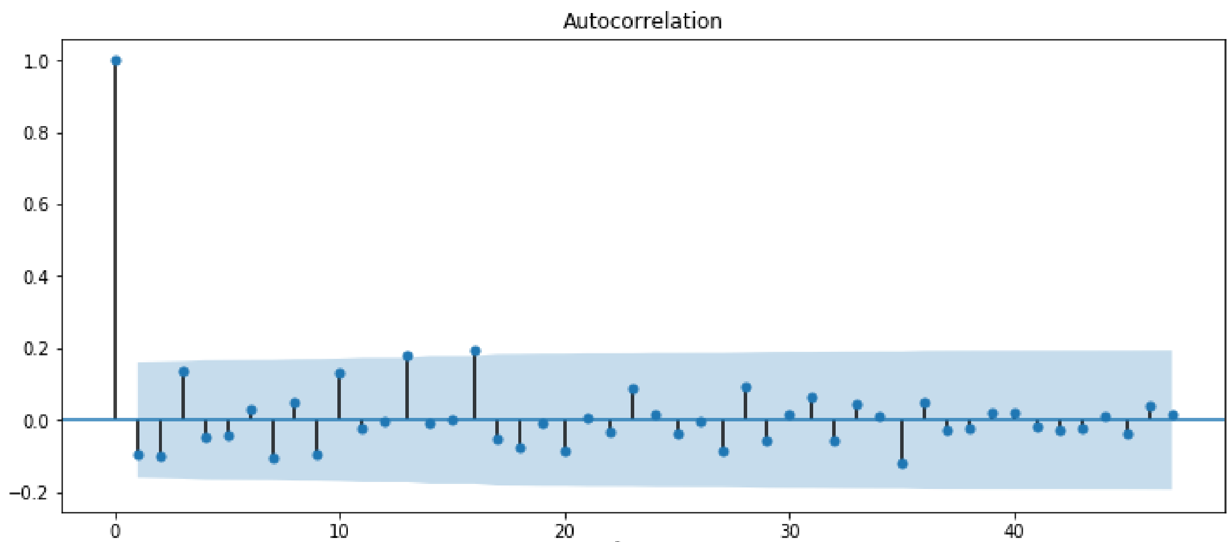
Fig. 1.3



Observed
Forcast
95% Prediction Interval
95% Confidence Interval

Fig. 1.5


Daily resolution log differenced price for stock 17
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0
20
40
60
80
100
120
140
Fig. 2.1


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
Fig. 2.2

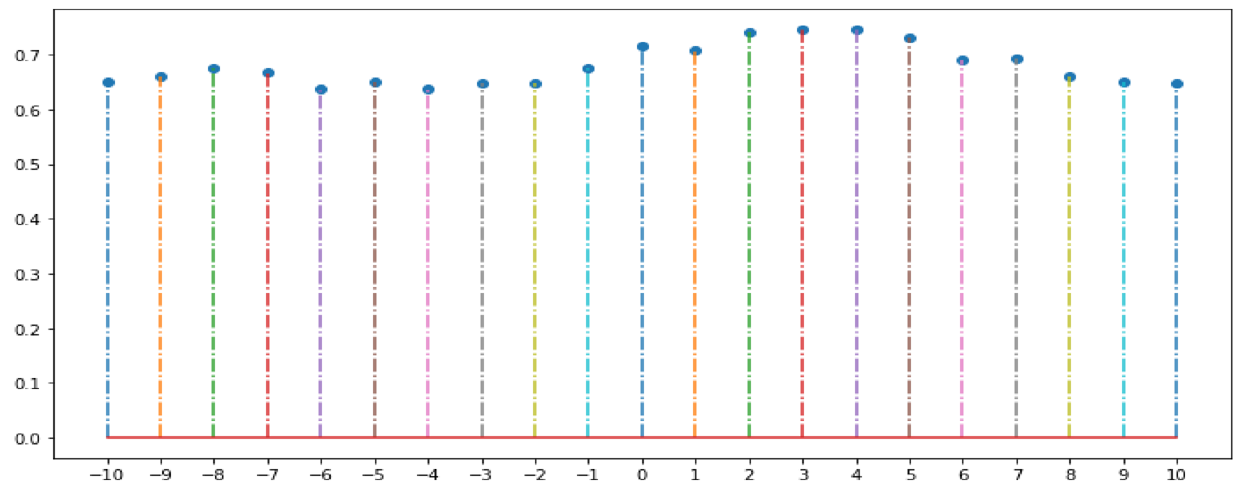

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10 −9 −8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7 8 9 10

Fig. 4.1

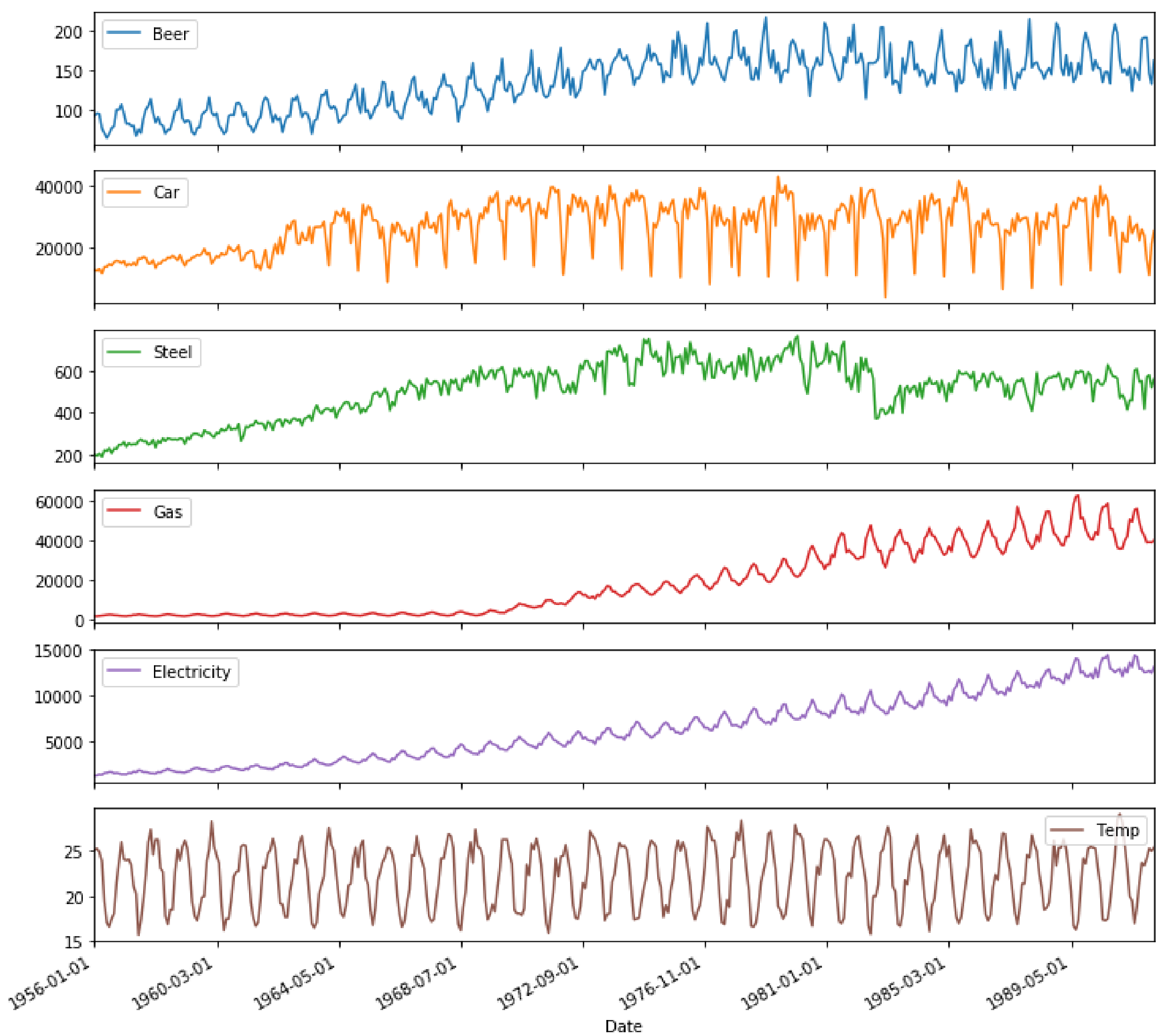

Beer
Car
Steel
Gas
Electricity
Temp
Date
1956-01-01
1960-03-01
1964-05-01
1968-07-01
1972-09-01
1976-11-01
1981-01-01
1985-03-01
1989-05-01

Fig. 4.2

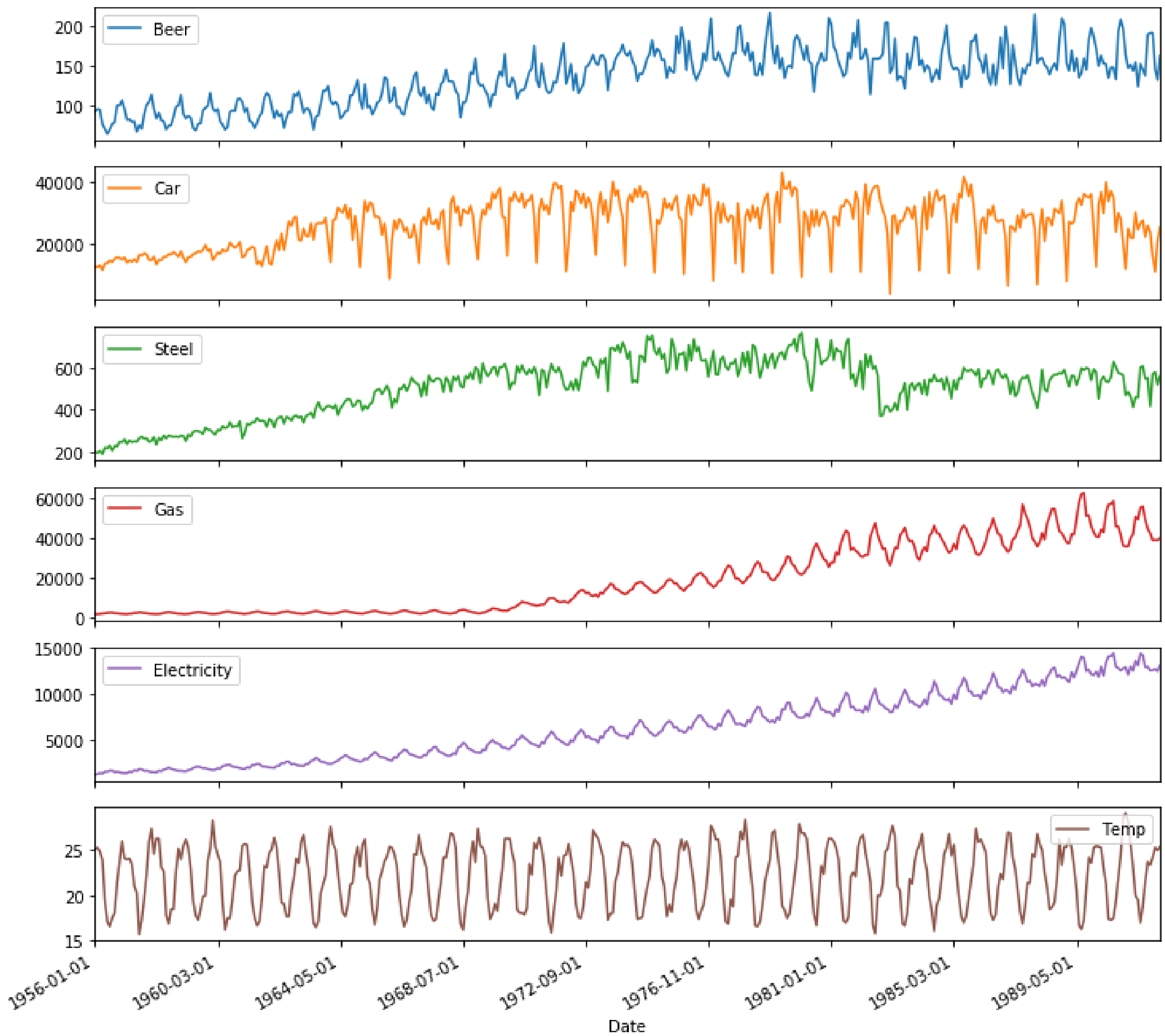

Beer
Car
Steel
Gas
Electricity
Temp
1956-01-01
1960-03-01
1964-05-01
1968-07-01
1972-09-01
1976-11-01
1981-01-01
1985-03-01
1989-05-01
Date

Fig. 4.2

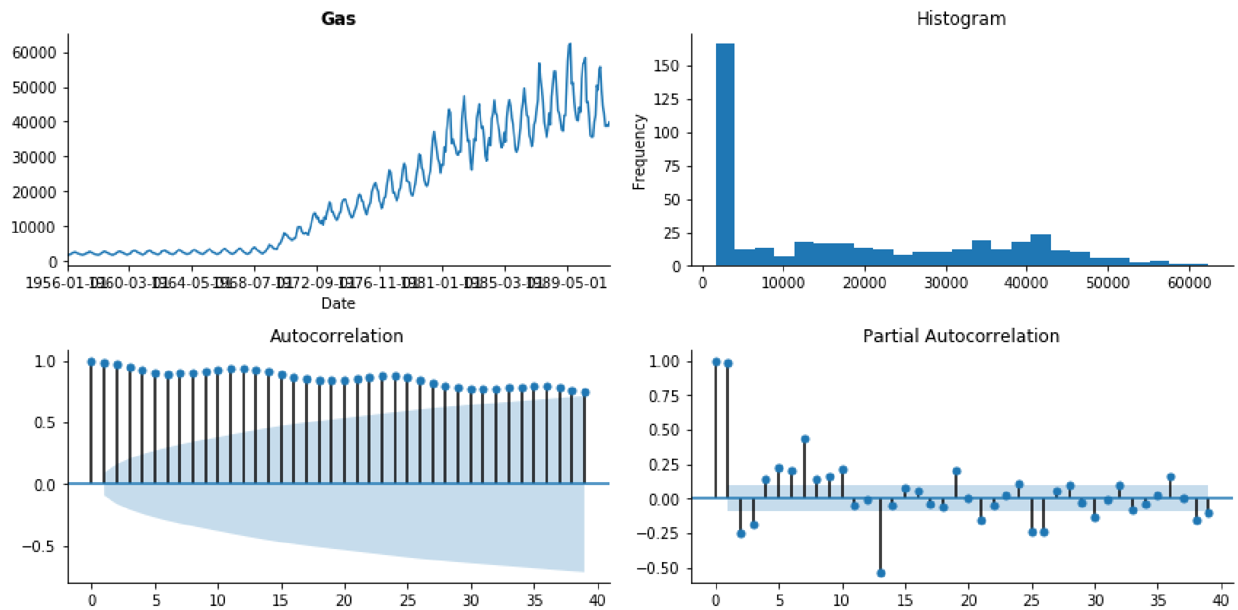

Gas
Date
Histogram
Frequency
Autocorrelation
Partial Autocorrelation

Fig. 4.3

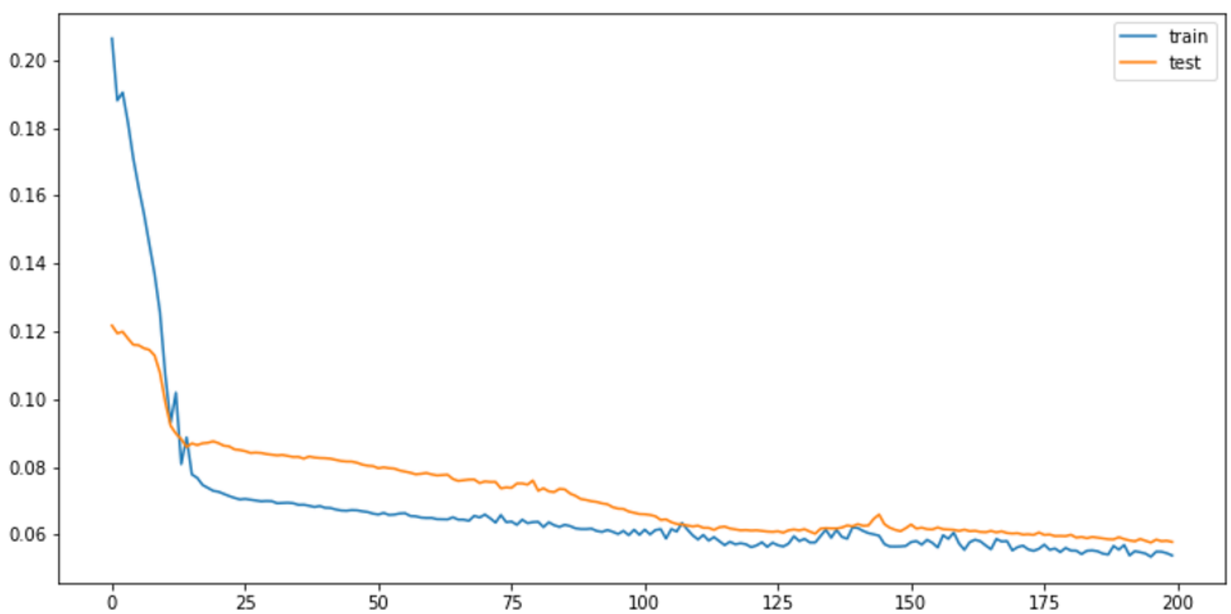

train
test

Fig 4.4

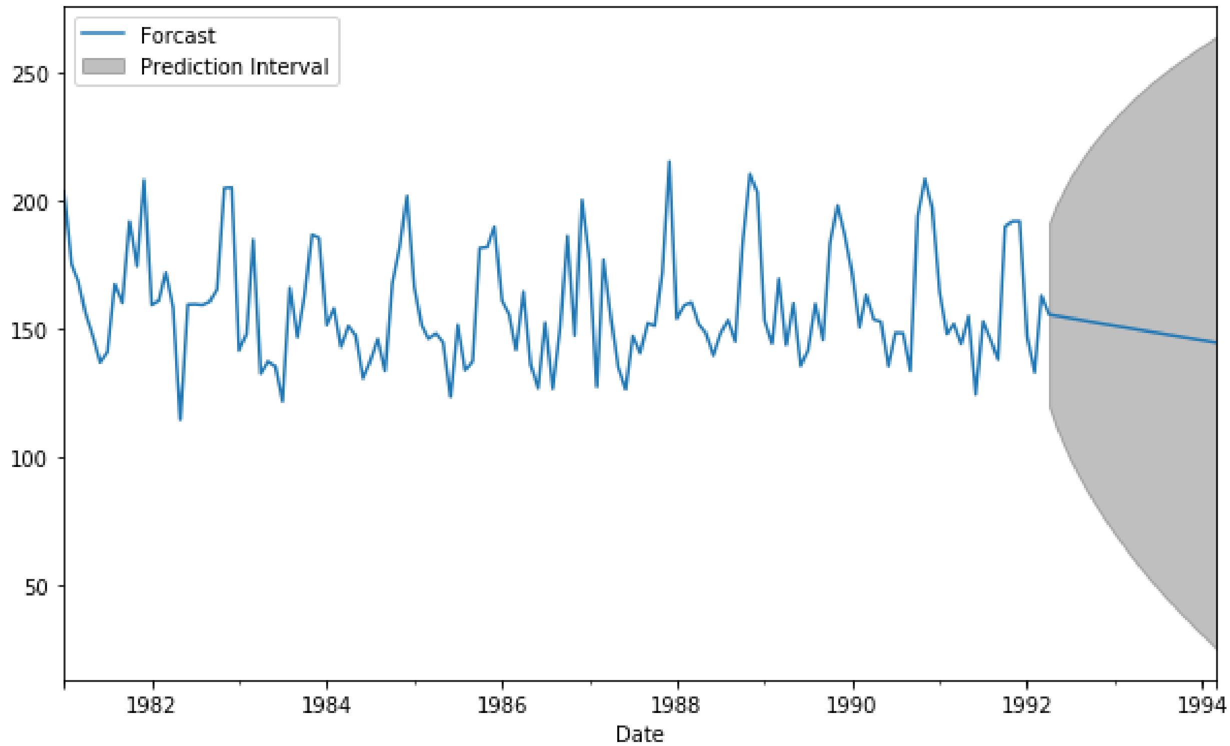

Forcast
Prediction Interval
250
200
150
100
50
1982
1984
1986
1988
1990
1992
1994
Date

Fig. 4.5

V. TABLE

Statespace Model Results

Dep. Variable:	Monthly Resolution	No. Observations:	576
Model:	SARIMAX(1, 0, 2)x(1, 1, 2, 12)	Log Likelihood	-291.171
Date:	Sun, 21 Apr 2019	AIC	596.342
Time:	23:53:03	BIC	626.344
Sample:	0 - 576	HQIC	608.079
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9550	0.017	57.066	0.000	0.922	0.988
ma.L1	0.2364	0.035	6.725	0.000	0.167	0.305
ma.L2	0.1352	0.042	3.205	0.001	0.053	0.218
ar.S.L12	-0.6150	0.287	-2.146	0.032	-1.177	-0.053
ma.S.L12	-0.4665	0.297	-1.571	0.116	-1.048	0.115
ma.S.L24	-0.6171	0.310	-1.988	0.047	-1.225	-0.009
sigma2	0.1512	0.010	15.901	0.000	0.133	0.170

Ljung-Box (Q):	51.15	Jarque-Bera (JB):	218.79
Prob(Q):	0.11	Prob(JB):	0.00
Heteroskedasticity (H):	0.67	Skew:	0.79
Prob(H) (two-sided):	0.01	Kurtosis:	5.70

Table 1.1

```
*---------------------------------*
*          GARCH Model Fit        *
*---------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model     : sGARCH(1,1)
Mean Model      : ARFIMA(0,0,0)
Distribution    : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001493    0.001064   1.4027 0.160696
omega   0.000071    0.000025   2.8144 0.004887
alpha1  0.654661    0.215102   3.0435 0.002338
beta1   0.254466    0.127638   1.9937 0.046190

Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001493    0.000986   1.5133 0.130215
omega   0.000071    0.000020   3.6190 0.000296
alpha1  0.654661    0.354124   1.8487 0.064504
beta1   0.254466    0.128760   1.9763 0.048123

LogLikelihood : 417.2

Information Criteria
------------------------------------

Akaike       -5.5092
Bayes        -5.4289
Shibata      -5.5106
Hannan-Quinn -5.4766

Weighted LB Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.2503  0.6169
Lag[2*(p+q)+(p+q)-1][2]    1.8531  0.2884
Lag[4*(p+q)+(p+q)-1][5]    3.8189  0.2775
d.o.f=0
H0 : No serial correlation

Weighted LB Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.2642 0.60727
Lag[2*(p+q)+(p+q)-1][5]    7.1950 0.04650
Lag[4*(p+q)+(p+q)-1][9]    8.9281 0.08409
d.o.f=2

Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3]    0.2525 0.500 2.000  0.6153
ARCH Lag[5]    0.9880 1.440 1.667  0.7365
ARCH Lag[7]    1.3057 2.315 1.543  0.8594

Nyblom stability test
------------------------------------
Joint Statistic:  1.001
Individual Statistics:
mu     0.17790
omega  0.09422
alpha1 0.42092
beta1  0.10533

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:         1.07 1.24 1.6
Individual Statistic:    0.35 0.47 0.75

Sign Bias Test
------------------------------------
                   t-value   prob sig
Sign Bias           0.6900 0.4913
Negative Sign Bias  0.9523 0.3425
Positive Sign Bias  1.2794 0.2028
Joint Effect        3.2931 0.3486

Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20     13.73       0.7990
2    30     16.80       0.9652
3    40     27.07       0.9255
4    50     28.00       0.9931
```

Table 2.1

```
*---------------------------------*
*          GARCH Model Fit        *
*---------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model      : eGARCH(1,1)
Mean Model       : ARFIMA(0,0,0)
Distribution     : std
```

```
Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu     -0.001522    0.000001  -2344.4        0
omega  -0.250034    0.000124  -2014.3        0
alpha1 -0.189924    0.000113  -1680.4        0
beta1   0.967394    0.000408   2369.3        0
gamma1 -0.175202    0.000101  -1728.5        0
shape   7.549996    0.004866   1551.5        0
LogLikelihood : 414.9
```

```
Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu     -0.001522    0.000044  -34.278        0
omega  -0.250034    0.005819  -42.969        0
alpha1 -0.189924    0.002264  -83.875        0
beta1   0.967394    0.025697   37.647        0
gamma1 -0.175202    0.004702  -37.260        0
shape   7.549996    0.035720  211.369        0
```

```
Information Criteria
------------------------------------

Akaike       -5.4515
Bayes        -5.3310
Shibata      -5.4545
Hannan-Quinn -5.4025
```

```
Weighted LB Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                      1.632  0.2014
Lag[2*(p+q)+(p+q)-1][2]     1.946  0.2719
Lag[4*(p+q)+(p+q)-1][5]     2.379  0.5319
d.o.f=0
H0 : No serial correlation
```

```
Weighted LB Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                      1.422  0.2330
Lag[2*(p+q)+(p+q)-1][5]     2.399  0.5276
Lag[4*(p+q)+(p+q)-1][9]     2.864  0.7812
d.o.f=2
```

```
Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3]    0.7148 0.500 2.000  0.3979
ARCH Lag[5]    1.3229 1.440 1.667  0.6401
ARCH Lag[7]    1.4517 2.315 1.543  0.8310
```

```
Nyblom stability test
------------------------------------
Joint Statistic:  3.427
Individual Statistics:
mu     0.07197
omega  0.07216
alpha1 0.07541
beta1  0.04062
gamma1 0.07064
shape  0.07642

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:          1.49 1.68 2.12
Individual Statistic:     0.35 0.47 0.75
```

```
Sign Bias Test
------------------------------------
                   t-value   prob sig
Sign Bias           1.3048 0.1940
Negative Sign Bias  1.6545 0.1002
Positive Sign Bias  0.3969 0.6921
Joint Effect        3.9587 0.2660
```

```
Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20      34.0      0.01838
2    30      37.2      0.14118
3    40      54.8      0.04791
4    50      62.0      0.10057
```

Table 2.2

VI. REFERENCE

1. Biessmann, F., Salinas, D., Schelter, S., Schmidt, P., & Lange, D. (2018, October). Deep Learning for Missing Value Imputationin Tables with Non-Numerical Data. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 2017-2025). ACM.
2. Che, Z., Purushotham, S., Cho, K., Sontag, D., & Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, *8*(1), 6085.
3. Long short-term memory. In Wikipedia. Retrieved April 22, 2019, from https://en.wikipedia.org/wiki/Long_short-term_memory
4. Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.
5. White, I. R., Royston, P., & Wood, A. M. (2011). Multiple imputation using chained equations: issues and guidance for practice. Statistics in medicine, 30(4), 377-399.
6. Yong, Z., Youwen, L., & Shixiong, X. (2009). An improved KNN text classification algorithm based on clustering. Journal of computers, 4(3), 230-237.
7. Engle, R. (2001). GARCH 101: The use of ARCH/GARCH models in applied econometrics. Journal of economic perspectives, 15(4), 157-168.
8. Ho, S. L., & Xie, M. (1998). The use of ARIMA models for reliability forecasting and analysis. Computers & industrial engineering, 35(1-2), 213-216.
9. Surgailis, D., & Viano, M. C. (2002). Long memory properties and covariance structure of the EGARCH model. ESAIM: Probability and Statistics, 6, 311-329.
10. Breusch, T. S., & Pagan, A. R. (1979). A simple test for heteroscedasticity and random coefficient variation. Econometrica: Journal of the Econometric Society, 1287-1294.

VII. APPENDIX

1. Notebook for scenario 1
7. Notebook for scenario 2
8. Notebook for scenario 3
9. Notebook for scenario 4

Scenario1

April 22, 2019

```
In [1]: import csv
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from statsmodels.tsa.stattools import acf
        from statsmodels.graphics.tsaplots import plot_acf
        import itertools
        from statsmodels.tsa.statespace.sarimax import SARIMAX
        import statsmodels.api as sm
        import warnings
```

```
In [89]: #define function for ADF test
         from statsmodels.tsa.stattools import adfuller
         def adf_test(timeseries):
             #Perform Dickey-Fuller test:
             print ('Results of Dickey-Fuller Test:')
             dftest = adfuller(timeseries, autolag='AIC')
             dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
             for key,value in dftest[4].items():
                dfoutput['Critical Value (%s)'%key] = value
             print (dfoutput)
```

0.0.1 To run this code, please make sure 'hyd_post.txt' is in the same directionary.

```
In [90]: data = np.genfromtxt(fname='hyd_post.txt',delimiter=",",skip_header=True,dtype=np.floa
         df_data=pd.DataFrame(data=data,columns=['Monthly Resolution'])
```

```
In [91]: fig=plt.figure(figsize=(12,5))
         ax = fig.add_subplot(111)
         ax.set_title('Monthly resolution of the level of a body of water')
         ax.text(0.5,-0.15, "Fig. 1.1", size=12, ha="center", transform=ax.transAxes,weight='bo
         plt.plot(data)
         plt.show();
```

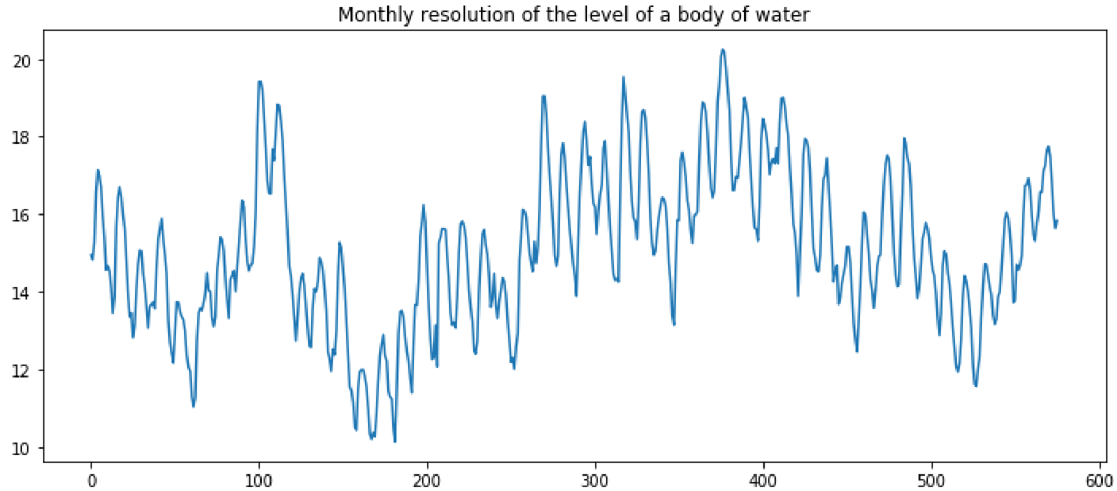


Fig. 1.1

```
In [94]: adf_test(data)

Results of Dickey-Fuller Test:
Test Statistic                  -2.667544
p-value                          0.079825
#Lags Used                      18.000000
Number of Observations Used    557.000000
Critical Value (1%)             -3.442145
Critical Value (5%)             -2.866743
Critical Value (10%)            -2.569541
dtype: float64
```

```
In [72]: fig = plt.figure(figsize=(12,5))
         ax = fig.add_subplot(111)
         ax.text(0.5,-0.15, "Fig. 1.2", size=12, ha="center", transform=ax.transAxes,weight='bo
         plot_acf(data,lags=np.arange(0,48),ax=ax)

Out[72]:
```

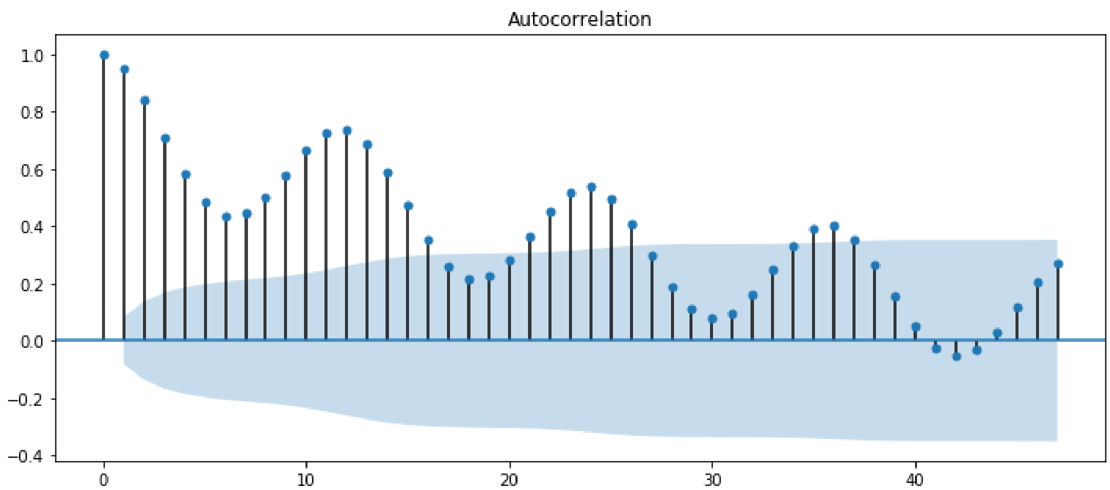


Fig. 1.2

Observed seasonal effect of lag 12

```
In [25]: # Define the p, d and q parameters to take any value between 0 and 3
         p = d = q = range(0, 3)

         # Generate all different combinations of p, q and q triplets
         pdq = list(itertools.product(p, d, q))

         # Generate all different combinations of seasonal p, q and q triplets
         seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

         print('Examples of parameter combinations for Seasonal ARIMA...')
         print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
         print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
         print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
         print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 0, 2, 12)
SARIMAX: (0, 0, 2) x (0, 1, 0, 12)
SARIMAX: (0, 0, 2) x (0, 1, 1, 12)
```

Choose the best parameter by the lowest AIC/BIC

```
In [28]: #Using grid search, we can identify the set of parameters that produces the best fitt
         #to our time series data. We can proceed to analyze this particular model in more dep
         warnings.filterwarnings("ignore") # specify to ignore warning messages
         AIC=10000
         BIC=10000
```

```
    order_aic=...
    seasonal_order_aic=...
    order_bic=...
    seasonal_order_bic=...
    for param in pdq:
        for param_seasonal in seasonal_pdq:
            try:
                mod = sm.tsa.statespace.SARIMAX(data,
                                                order=param,
                                                seasonal_order=param_seasonal,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                results = mod.fit()
                a=results.aic
                if a<=AIC:
                    AIC=a
                    order_aic=param
                    seasonal_order_aic=param_seasonal
                b=results.bic
                if b<=BIC:
                    BIC=b
                    order_bic=param
                    seasonal_order_bic=param_seasonal
                # print('ARIMA{}x{} - AIC:{}, BIC:{}'.format(param, param_seasonal, resul
            except:
                continue
```

```
In [29]: AIC,BIC
```

```
Out[29]: (596.3422624079133, 622.8894237518107)
```

```
In [30]: order_aic,seasonal_order_aic
```

```
Out[30]: ((1, 0, 2), (1, 1, 2, 12))
```

```
In [31]: order_bic,seasonal_order_bic
```

```
Out[31]: ((1, 0, 2), (0, 1, 2, 12))
```

The optimal parameters determined by AIC and BIC are different. Here we will use the one chosen by the lowest AIC by the model diagostic done below.

0.0.2 Using the optimal parameters calculated above

```
In [73]: mod = sm.tsa.statespace.SARIMAX(df_data,
                                         order=(1, 0, 2),
                                         seasonal_order=(1, 1, 2, 12),
                                         enforce_stationarity=False,
```

```
                                      enforce_invertibility=False)

results = mod.fit()
print(results.summary())
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                  Monthly Resolution   No. Observations:                  576
Model:             SARIMAX(1, 0, 2)x(1, 1, 2, 12)   Log Likelihood                -291.171
Date:                             Sun, 21 Apr 2019   AIC                            596.342
Time:                                     23:53:03   BIC                            626.344
Sample:                                          0   HQIC                           608.079
                                             - 576
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9550      0.017     57.066      0.000       0.922       0.988
ma.L1          0.2364      0.035      6.725      0.000       0.167       0.305
ma.L2          0.1352      0.042      3.205      0.001       0.053       0.218
ar.S.L12      -0.6150      0.287     -2.146      0.032      -1.177      -0.053
ma.S.L12      -0.4665      0.297     -1.571      0.116      -1.048       0.115
ma.S.L24      -0.6171      0.310     -1.988      0.047      -1.225      -0.009
sigma2         0.1512      0.010     15.901      0.000       0.133       0.170
===================================================================================
Ljung-Box (Q):                       51.15   Jarque-Bera (JB):               218.79
Prob(Q):                              0.11   Prob(JB):                         0.00
Heteroskedasticity (H):               0.67   Skew:                             0.79
Prob(H) (two-sided):                  0.01   Kurtosis:                         5.70
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
In [88]: results.plot_diagnostics(figsize=(15, 12))
         plt.text(-2,-1.4, "Fig. 1.3", size=12, ha="center",weight='bold')
         plt.show()
```

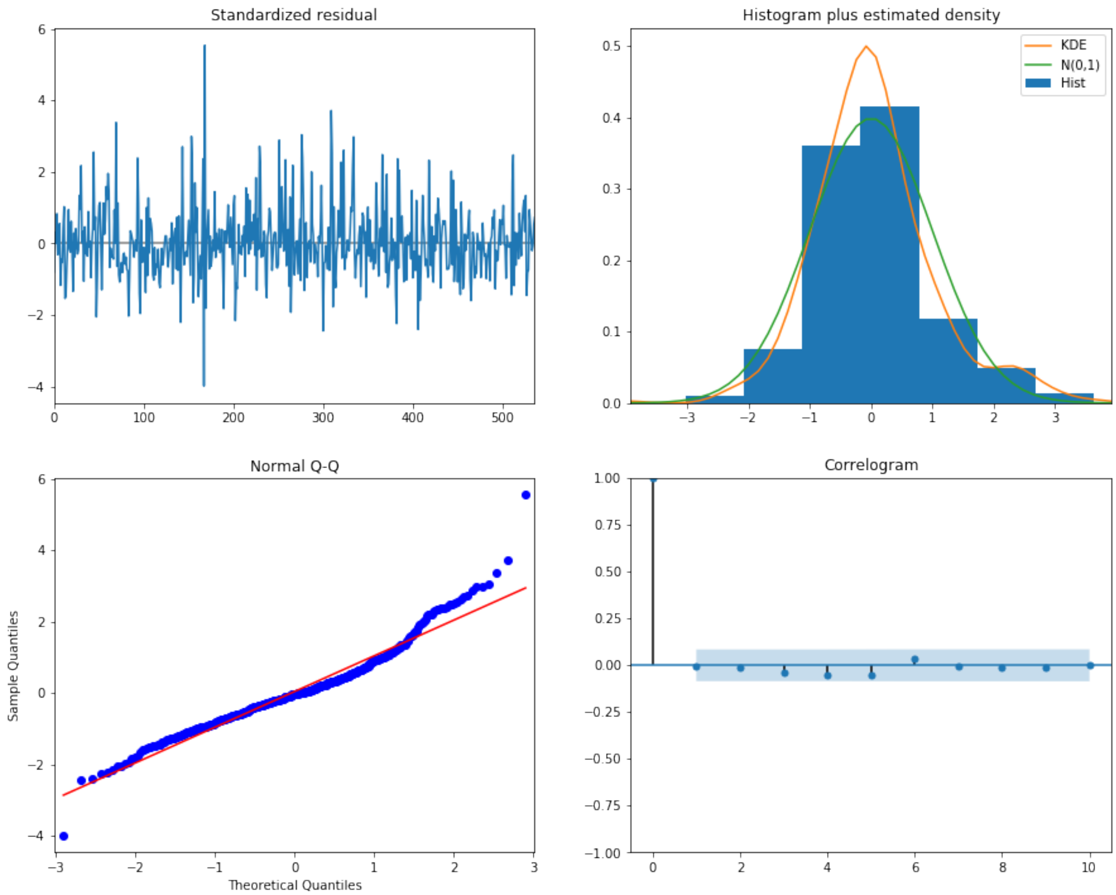


Fig. 1.3

0.0.3 Using the parameters got by auto.arima from R

```
In [11]: mod_R = sm.tsa.statespace.SARIMAX(df_data,
                                            order=(2, 0, 0),
                                            seasonal_order=(1, 1, 0, 12),
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)


         results_by_R = mod_R.fit()
         print(results_by_R.summary())
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                    Monthly Resolution   No. Observations:               576
Model:                SARIMAX(2, 0, 0)x(1, 1, 0, 12)   Log Likelihood             -380.958
Date:                               Sun, 21 Apr 2019   AIC                         769.916
Time:                                       22:42:24   BIC                         787.156
Sample:                                            0   HQIC                        776.653
```

```
                                        - 576
Covariance Type:                          opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1561      0.032     36.583      0.000       1.094       1.218
ar.L2         -0.2190      0.033     -6.622      0.000      -0.284      -0.154
ar.S.L12      -0.5314      0.030    -17.970      0.000      -0.589      -0.473
sigma2         0.2340      0.011     21.161      0.000       0.212       0.256
===================================================================================
Ljung-Box (Q):                      128.99   Jarque-Bera (JB):                99.11
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             0.37
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.94
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Html_file= open("a.html","w") Html_file.write(a) Html_file.close() imgkit.from_file('a.html', 'out.jpg')

```
In [12]: results_by_R.plot_diagnostics(figsize=(15, 12))
         plt.text(-2,-1.4, "Fig. 1.4", size=12, ha="center",weight='bold')
         plt.show()
```

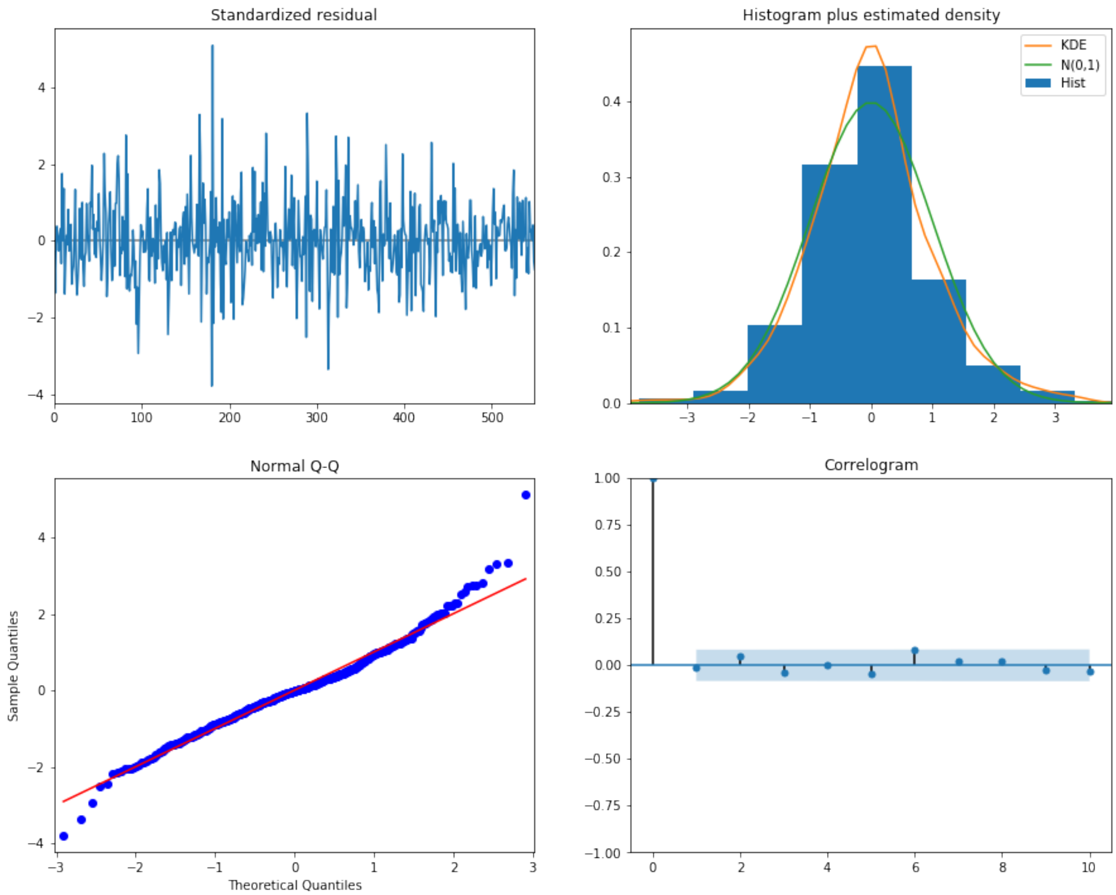


Fig. 1.4

I would prefer SARIMA(1, 0, 2)x(1, 1, 2, 12) by the Ljung-Box Test.

```
In [95]: # Get forecast 24 steps ahead in future
         pred_uc = results.get_forecast(steps=24)

         # Get 95% confidence intervals of forecasts
         pred_ci = pred_uc.conf_int(alpha=0.05)

In [96]: tocsv=pred_uc.predicted_mean.values
         np.savetxt('../Result in CSV/Li_Scenario1.csv', tocsv, delimiter=',')

In [97]: mean=pred_uc.predicted_mean.values
         var=pred_uc.var_pred_mean
         prediction_u=mean+1.96*var
         prediction_l=mean-1.96*var

In [98]: pred_pi = pred_uc.conf_int(alpha=0.05)
         pred_pi['lower Monthly Resolution']=prediction_l
         pred_pi['upper Monthly Resolution']=prediction_u
```

```
In [101]: ax = df_data[500:].plot(label='observed', figsize=(12, 5))
          pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
          ax.fill_between(pred_ci.index,
                          pred_pi.iloc[:, 0],
                          pred_pi.iloc[:, 1], color='y', alpha=.35,label='95% Prediction Inter
          ax.fill_between(pred_ci.index,
                          pred_ci.iloc[:, 0],
                          pred_ci.iloc[:, 1], color='b', alpha=.50,label='95% Confidence Inter
          ax.text(0.5,-0.1, "Fig. 1.5", size=12, ha="center", transform=ax.transAxes,weight='bo
          plt.legend(['Observed','Forcast','95% Prediction Interval','95% Confidence Interval']
          plt.show()
```

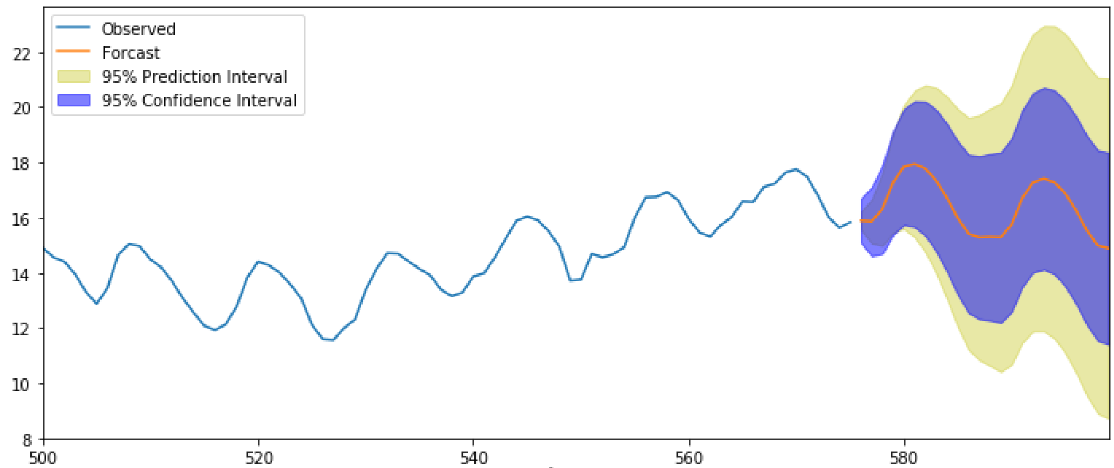


Fig. 1.5

Scenario2

April 22, 2019

```
In [1]: import csv
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from statsmodels.graphics.tsaplots import plot_acf
        import statsmodels.api as sm
        from statsmodels.stats.diagnostic import het_arch
        from statsmodels.tsa.stattools import q_stat
        from statsmodels.tsa.stattools import acf
        import warnings
        warnings.simplefilter('ignore')
        from arch import arch_model

        %matplotlib inline
```

0.0.1 To run this code, please make sure the 40 stocks files are stored in the a sub directory named "stock".

```
In [2]: def getForecast_GARCH(N):
            stock='stock/stock'+str(N)+'.txt'
            data = np.genfromtxt(fname=stock,delimiter=",",skip_header=True,dtype=np.float,usec
            df_data=pd.DataFrame(data=data,columns=['Price'])
            am = arch_model(df_data*100, p=1, o=1, q=1,vol='GARCH')
            res = am.fit(update_freq=5, disp='off')
            return res
```

```
In [16]: def getQuantile15_GARCH_simulation(N):
             res=getForecast_GARCH(N)
             forecasts = res.forecast(horizon=10, method='bootstrap',simulations=10000)
             quantile15=np.quantile(forecasts.simulations.values[-1],q=0.15,axis=0)/100
             return np.array(quantile15)
```

```
In [4]: def getQuantile15_GARCH_theoretical(N):
            res=getForecast_GARCH(N)
            forecasts = res.forecast(horizon=10, method='simulation',simulations=10000)
            s=np.sqrt(forecasts.variance.values[-1]/10000)
            quantile15=-1.03643*s
            return np.array(quantile15)
```

```
In [5]: def getForecast(N):
            stock='stock/stock'+str(N)+'.txt'
            data = np.genfromtxt(fname=stock,delimiter=",",skip_header=True,dtype=np.float,use
            df_data=pd.DataFrame(data=data,columns=['Price'])
            am = arch_model(df_data*100, p=1, o=1, q=1,vol='EGARCH', dist='StudentsT')
            res = am.fit(update_freq=5, disp='off')
            return res
```

```
In [6]: def getQuantile15(N):
            res=getForecast(N)
            forecasts = res.forecast(horizon=10, method='simulation',simulations=10000)
            quantile15=np.quantile(forecasts.simulations.values[-1],q=0.15,axis=0)/100
            return np.array(quantile15)
```

```
In [7]: N=17
        stock='stock/stock'+str(N)+'.txt'
        data = np.genfromtxt(fname=stock,delimiter=",",skip_header=True,dtype=np.float,usecols
        df_data=pd.DataFrame(data=data,columns=['Price'])
        fig=plt.figure(figsize=(12,5))
        ax = fig.add_subplot(111)
        ax.set_title(str('Daily resolution log differenced price for stock '+str(N)))
        ax.text(0.5,-0.12, "Fig. 2.1", size=12, ha="center", transform=ax.transAxes,weight='bol
        plt.plot(data)
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x29647116080>]
```

Fig. 2.1

Engle's LM test:

```
In [8]: print("p value is {:5.6f}".format(het_arch(data)[1]))
```

```
p value is 0.003577
```

The p-value is less than 1% hence the null hypothesis can be rejected at 1% significance level, meaning that the ARCH affects are quite significant in the daily log returns.

ACF

```
In [9]: fig = plt.figure(figsize=(12,5))
        ax = fig.add_subplot(111)
        ax.text(0.5,-0.1, "Fig. 2.2", size=12, ha="center", transform=ax.transAxes,weight='bold
        plot_acf(data,lags=np.arange(0,48),ax=ax);
```

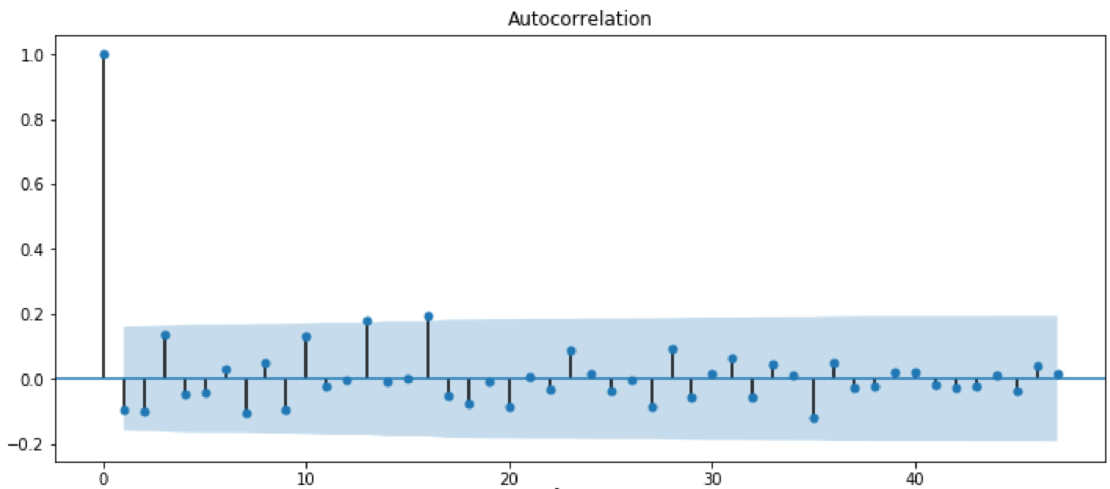


Fig. 2.2

```
In [41]: res=getForecast_GARCH(20)
         forecasts = res.forecast(horizon=10, method='bootstrap',simulations=10000)

In [42]: res.summary()

Out[42]: <class 'statsmodels.iolib.summary.Summary'>
         """
                           Constant Mean - GJR-GARCH Model Results
         ==============================================================================
         Dep. Variable:                  Price   R-squared:                      -0.000
         Mean Model:             Constant Mean   Adj. R-squared:                 -0.000
         Vol Model:                   GJR-GARCH   Log-Likelihood:                -224.652
         Distribution:                   Normal   AIC:                            459.304
         Method:             Maximum Likelihood   BIC:                            474.357
                                                  No. Observations:                   150
         Date:                 Mon, Apr 22 2019   Df Residuals:                       145
         Time:                         14:56:26   Df Model:                             5
```

```
                           Mean Model
==============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.1058  9.562e-02      1.107      0.269 [-8.161e-02,  0.293]
                        Volatility Model
=============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
-----------------------------------------------------------------------------
omega          0.5142      0.160      3.212  1.319e-03   [  0.200,  0.828]
alpha[1]   8.2717e-12  5.427e-02  1.524e-10      1.000   [ -0.106,  0.106]
gamma[1]       0.3308      0.515      0.642      0.521   [ -0.679,  1.341]
beta[1]        0.4403      0.179      2.456  1.405e-02 [8.893e-02,  0.792]
=============================================================================

Covariance estimator: robust
"""
```

The forecast result using GARCH Model.

1. By simulation(Bootstrap)

In [17]:
```
result_data_garch_simulation=[]
for i in np.arange(1,41):
    a=getQuantile15_GARCH_simulation(i)
    result_data_garch_simulation.append(a)
```

2. Theoretical

In [18]:
```
result_data_garch_theoretical=[]
for i in np.arange(1,41):
    a=getQuantile15_GARCH_theoretical(i)
    result_data_garch_theoretical.append(a)
```

In [48]:
```
result_data_garch_simulation[20]
```

Out[48]:
```
array([-0.02105246, -0.01891902, -0.01958589, -0.01846559, -0.01768079,
       -0.01676607, -0.01701528, -0.01646648, -0.01638114, -0.01646933])
```

In [49]:
```
result_data_garch_theoretical[20]
```

Out[49]:
```
array([-0.02181549, -0.02068851, -0.01970846, -0.01899656, -0.01841072,
       -0.01796263, -0.01760929, -0.01737189, -0.01716597, -0.01697496])
```

The forecast result using EGARCH Model.

In [25]:
```
result_data=[]
for i in np.arange(1,41):
    a=getQuantile15(i)
    result_data.append(a)
```

```
C:\Users\Jackie Li\Anaconda3\lib\site-packages\arch\univariate\base.py:577: ConvergenceWarning
The optimizer returned code 9. The message is:
Iteration limit exceeded
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
C:\Users\Jackie Li\Anaconda3\lib\site-packages\arch\univariate\base.py:577: ConvergenceWarning
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
C:\Users\Jackie Li\Anaconda3\lib\site-packages\arch\univariate\base.py:577: ConvergenceWarning
The optimizer returned code 9. The message is:
Iteration limit exceeded
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
C:\Users\Jackie Li\Anaconda3\lib\site-packages\arch\univariate\base.py:577: ConvergenceWarning
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
C:\Users\Jackie Li\Anaconda3\lib\site-packages\arch\univariate\base.py:577: ConvergenceWarning
The optimizer returned code 9. The message is:
Iteration limit exceeded
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
```

```
In [28]: result_data[10]
```

```
Out[28]: array([-0.01574469, -0.01516793, -0.01430382, -0.01433381, -0.01413956,
                -0.01380114, -0.01409021, -0.01405224, -0.01419223, -0.01415402])
```

prefer GARCH model

```
In [50]: result_data=np.array(result_data).T
         result_data_garch_simulation=np.array(result_data_garch_simulation).T
         result_data_garch_theoretical=np.array(result_data_garch_theoretical).T
```

```
In [54]: result_data.shape
```

```
Out[54]: (10, 40)
```

```
In [57]: np.savetxt('comparison/GARCH_Theoretical.csv',result_data_garch_theoretical,delimiter=
```

```
In [51]: forecasts.residual_variance.tail()/100
```

```
Out[51]:          h.01      h.02      h.03      h.04      h.05     h.06      h.07  \
        145       NaN       NaN       NaN       NaN       NaN      NaN       NaN
        146       NaN       NaN       NaN       NaN       NaN      NaN       NaN
        147       NaN       NaN       NaN       NaN       NaN      NaN       NaN
        148       NaN       NaN       NaN       NaN       NaN      NaN       NaN
        149  0.022734  0.018532  0.016057  0.014663  0.013827  0.01333  0.013108

                 h.08      h.09      h.10
        145       NaN       NaN       NaN
        146       NaN       NaN       NaN
        147       NaN       NaN       NaN
        148       NaN       NaN       NaN
        149  0.012898  0.012889  0.012699
```

assume normal distribution, 15% lower quantile

```
In [52]: s=np.sqrt(forecasts.variance.values[-1]/10000)
         -1.03643*s
```

```
Out[52]: array([-0.01562717, -0.01410903, -0.01313339, -0.01255042, -0.012187  ,
                -0.01196625, -0.01186607, -0.01177044, -0.01176677, -0.01167961])
```

By simulation, 15% lower quantile

```
In [53]: getQuantile15(20)
```

```
Out[53]: array([-0.01316212, -0.0119065 , -0.01132768, -0.01071698, -0.01022607,
                -0.01001358, -0.00941453, -0.00906662, -0.00917292, -0.00925284])
```

scenario3

April 22, 2019

```
In [3]: import pandas as pd
        from functools import reduce
        import numpy as np
        import impyute as impy
        from sklearn.metrics import mean_squared_error,r2_score
        import datawig
```

Data Preparation

```
In [2]: def readtodf(filename, colname):
            a=filename+'.txt'
            data = pd.read_csv(a, sep=",", header=(0))
            data.columns=['a','Date',colname]
            data=data.drop('a',axis=1)
            data['Date'] = pd.to_datetime(data['Date'])
            data = data.set_index('Date')

            return data
```

```
In [4]: target=readtodf('prod_target','Beer')
        prod_1=readtodf('prod_1','Car')
        prod_2=readtodf('prod_2','Steel')
        eng_1=readtodf('eng_1','Gas')
        eng_2=readtodf('eng_2','Electricity')
```

```
In [5]: temp = pd.read_csv('temp.txt', sep=",", header=(0))
        temp.columns=['num','year','month','Temp']
        temp=temp.drop('num',axis=1)
        temp['day']=1
        temp['Date']=pd.to_datetime(temp[['year', 'month', 'day']])
        temp=temp.drop(['year', 'month', 'day'],axis=1)
        temp = temp.set_index('Date')
```

```
In [6]: target.shape,prod_1.shape,prod_2.shape, eng_1.shape,eng_2.shape,temp.shape
```

```
Out[6]: ((435, 1), (369, 1), (435, 1), (435, 1), (435, 1), (581, 1))
```

```
In [7]:  # creat dataframe version of merged data
        dfs = [target, prod_1, prod_2, eng_1,eng_2,temp]
```

```
df_final = reduce(lambda left,right: pd.merge(left,right,left_index=True, right_index='
#df_final.to_csv('data_merged.csv')

# creat numpy version of merged data
np_final=np.array(df_final.values,dtype=np.float)
```

In [20]:
```
df_final.shape
```

Out[20]:
```
(435, 6)
```

In [19]:
```
df_final.head()
```

Out[19]:

Date	Beer	Car	Steel	Gas	Electricity	Temp
1956-01-01	93.2	NaN	196.9	1709	1254	25.1
1956-02-01	96.0	NaN	192.1	1646	1290	25.3
1956-03-01	95.2	NaN	201.8	1794	1379	24.9
1956-04-01	77.1	NaN	186.9	1878	1346	23.9
1956-05-01	70.9	NaN	218.0	2173	1535	19.4

0.0.1 Imputation

1.Mice The Multiple Imputation by Chained Equations (MICE) method is widely used in practice, which uses chain equations to create multiple imputations for variables of diferent types.

In [8]:
```
ip_mice=impy.imputation.cs.mice(np_final)
np.savetxt('./Imputation Results/imputation_mice.csv',ip_mice,delimiter=",")
```

2.KNN

- This method uses k-nearest neighbor to fInd similar samples and imputed unobserved data by weighted average of similar observations.
- Basic idea: Impute array with a basic mean impute and then use the resulting complete array to construct a KDTree. Use this KDTree to compute nearest neighbours. After finding k nearest neighbours, take the weighted average of them. Basically, find the nearest row in terms of distance

In [10]:
```
ip_knn=impy.imputation.cs.fast_knn(np_final)
np.savetxt('./Imputation Results/imputation_knn.csv',ip_knn,delimiter=",")
```

3.DataWig

- "Deep" Learning for Missing Value Imputationin Tables with Non-Numerical Data
- Details on the underlying model can be found in Biessmann, Salinas et al. 2018

In [128]:
```
#Initialize a SimpleImputer model
imputer = datawig.SimpleImputer(
    input_columns=['Car','Steel','Gas','Electricity','Temp'], # column(s) containing
    output_column='Beer', # the column we'd like to impute values for
```

```
            output_path = 'imputer_model' # stores model data and metrics
            )

        #Using LSTMs instead of bag-of-words
        # data_encoder_cols = [NumericalEncoder('Car'), NumericalEncoder('Steel'),NumericalE
        #                      NumericalEncoder('Electricity'),NumericalEncoder('Temp')]
        # label_encoder_cols = [NumericalEncoder('Beer')]
        # data_featurizer_cols = [LSTMFeaturizer('Car'), LSTMFeaturizer('Steel'),LSTMFeaturi
        #                      LSTMFeaturizer('Electricity'),LSTMFeaturizer('Temp')]

        # imputer = Imputer(
        #     data_featurizers=data_featurizer_cols,
        #     label_encoders=label_encoder_cols,
        #     data_encoders=data_encoder_cols,
        #     output_path='imputer_model'
        # )
```

In [137]:
```
#Fit an imputer model on the train data
imputer.fit(train_df=df_final[df_final['Beer'].notnull()], num_epochs=300,learning_ra
```

```
2019-04-20 23:21:52,585 [INFO]  Assuming 5 numeric input columns: Car, Steel, Gas, Electricity
2019-04-20 23:21:52,589 [INFO]  Assuming 0 string input columns:
2019-04-20 23:21:52,593 [INFO]  No output column name provided for ColumnEncoder using Beer
2019-04-20 23:21:52,596 [INFO]  Assuming numeric output column: Beer
2019-04-20 23:21:52,599 [INFO]  Using [[cpu(0)]] as the context for training
2019-04-20 23:21:52,605 [INFO]  Detected 0 rows with missing labels                         fo
2019-04-20 23:21:52,608 [INFO]  Dropping 0/364 rows
2019-04-20 23:21:52,611 [INFO]  Detected 0 rows with missing labels                         fo
2019-04-20 23:21:52,614 [INFO]  Dropping 0/40 rows
2019-04-20 23:21:52,617 [INFO]  Train: 364, Test: 40
2019-04-20 23:21:52,619 [INFO]  Building Train Iterator with 364 elements
2019-04-20 23:21:52,637 [INFO]  Concatenating numeric columns ['Car', 'Steel', 'Gas', 'Electric
2019-04-20 23:21:52,640 [INFO]  Normalizing with StandardScaler
2019-04-20 23:21:52,646 [INFO]  Data Encoding - Encoded 365 rows of column
2019-04-20 23:21:52,651 [INFO]  Concatenating numeric columns ['Beer'] into Beer
2019-04-20 23:21:52,653 [INFO]  Normalizing with StandardScaler
2019-04-20 23:21:52,657 [INFO]  Label Encoding - Encoded 365 rows of column
2019-04-20 23:21:52,659 [INFO]  Building Test Iterator with 40 elements
2019-04-20 23:21:52,670 [INFO]  Concatenating numeric columns ['Car', 'Steel', 'Gas', 'Electric
2019-04-20 23:21:52,672 [INFO]  Normalizing with StandardScaler
2019-04-20 23:21:52,676 [INFO]  Data Encoding - Encoded 40 rows of column
2019-04-20 23:21:52,681 [INFO]  Concatenating numeric columns ['Beer'] into Beer
2019-04-20 23:21:52,685 [INFO]  Normalizing with StandardScaler
2019-04-20 23:21:52,690 [INFO]  Label Encoding - Encoded 40 rows of column
2019-04-20 23:21:52,693 [INFO]
========== start: fit model
2019-04-20 23:21:52,695 [WARNING]  Already bound, ignoring bind()
C:\Users\Jackie Li\Anaconda3\lib\site-packages\mxnet\module\base_module.py:503: UserWarning: Pa
```

```
  allow_missing=allow_missing, force_init=force_init)
2019-04-20 23:21:52,698 [WARNING]  optimizer already initialized, ignoring...
2019-04-20 23:21:52,755 [INFO]  Epoch[0] Batch [0-37]        Speed: 3709.87 samples/sec
2019-04-20 23:21:52,800 [INFO]  Epoch[0] Train-cross-entropy=0.934271
2019-04-20 23:21:52,803 [INFO]  Epoch[0] Train-Beer-accuracy=0.000000
2019-04-20 23:21:52,806 [INFO]  Epoch[0] Time cost=0.103
2019-04-20 23:21:52,848 [INFO]  Saved checkpoint to "imputer_model\model-0000.params"
2019-04-20 23:21:52,856 [INFO]  Epoch[0] Validation-cross-entropy=1.502696
2019-04-20 23:21:52,858 [INFO]  Epoch[0] Validation-Beer-accuracy=0.000000
2019-04-20 23:21:52,907 [INFO]  Epoch[1] Batch [0-37]        Speed: 4122.24 samples/sec
2019-04-20 23:21:52,958 [INFO]  Epoch[1] Train-cross-entropy=0.922294
2019-04-20 23:21:52,960 [INFO]  Epoch[1] Train-Beer-accuracy=0.000000
2019-04-20 23:21:52,962 [INFO]  Epoch[1] Time cost=0.102
2019-04-20 23:21:52,984 [INFO]  Saved checkpoint to "imputer_model\model-0001.params"
2019-04-20 23:21:52,991 [INFO]  Epoch[1] Validation-cross-entropy=1.512594
2019-04-20 23:21:52,993 [INFO]  Epoch[1] Validation-Beer-accuracy=0.000000
2019-04-20 23:21:53,046 [INFO]  Epoch[2] Batch [0-37]        Speed: 3785.68 samples/sec
2019-04-20 23:21:53,094 [INFO]  Epoch[2] Train-cross-entropy=0.911299
2019-04-20 23:21:53,096 [INFO]  Epoch[2] Train-Beer-accuracy=0.000000
2019-04-20 23:21:53,098 [INFO]  Epoch[2] Time cost=0.103
2019-04-20 23:21:53,114 [INFO]  Saved checkpoint to "imputer_model\model-0002.params"
2019-04-20 23:21:53,123 [INFO]  Epoch[2] Validation-cross-entropy=1.523891
2019-04-20 23:21:53,125 [INFO]  Epoch[2] Validation-Beer-accuracy=0.000000
2019-04-20 23:21:53,178 [INFO]  Epoch[3] Batch [0-37]        Speed: 3785.64 samples/sec
2019-04-20 23:21:53,224 [INFO]  Epoch[3] Train-cross-entropy=0.900580
2019-04-20 23:21:53,226 [INFO]  Epoch[3] Train-Beer-accuracy=0.000000
2019-04-20 23:21:53,228 [INFO]  Epoch[3] Time cost=0.102
2019-04-20 23:21:53,251 [INFO]  Saved checkpoint to "imputer_model\model-0003.params"
2019-04-20 23:21:53,258 [INFO]  No improvement detected for 3 epochs compared to 1.50269575417
2019-04-20 23:21:53,261 [INFO]  Stopping training, patience reached
2019-04-20 23:21:53,263 [INFO]
========== done (0.5714719295501709 s) fit model
2019-04-20 23:21:53,276 [INFO]  Expected calibration error: 100.0%
2019-04-20 23:21:53,282 [INFO]  Expected calibration error after calibration: 100.0%
2019-04-20 23:21:53,301 [INFO]  save metrics in imputer_model\fit-test-metrics.json
2019-04-20 23:21:53,311 [INFO]  Keeping imputer_model\model-0000.params
2019-04-20 23:21:53,314 [INFO]  Deleting imputer_model\model-0001.params
2019-04-20 23:21:53,321 [INFO]  Deleting imputer_model\model-0002.params
2019-04-20 23:21:53,325 [INFO]  Deleting imputer_model\model-0003.params
```

```
Out[137]: <datawig.simple_imputer.SimpleImputer at 0x28ee200f6a0>
```

```
In [138]: #Impute missing values and return original dataframe with predictions
          imputed = imputer.predict(df_final)
          #imputed.to_csv('./Imputation Results/imputation_Datawig.csv')
```

```
2019-04-20 23:21:55,699 [INFO]  Concatenating numeric columns ['Car', 'Steel', 'Gas', 'Electric
2019-04-20 23:21:55,701 [INFO]  Normalizing with StandardScaler
```

```
2019-04-20 23:21:55,706 [INFO]  Data Encoding - Encoded 435 rows of column
2019-04-20 23:21:55,711 [INFO]  Concatenating numeric columns ['Beer'] into Beer
2019-04-20 23:21:55,714 [INFO]  Normalizing with StandardScaler
2019-04-20 23:21:55,718 [INFO]  Label Encoding - Encoded 435 rows of column
2019-04-20 23:21:55,776 [INFO]  Top-k only for CategoricalEncoder, dropping Beer, <class 'dataw
2019-04-20 23:21:55,779 [INFO]  Precision filtering only for CategoricalEncoder returning
```

In [139]:
```
predictions=imputed[imputed['Beer'].notnull()]
```

In [140]:
```
#Calculate MSE score
MSE = mean_squared_error(predictions['Beer'].values, predictions['Beer_imputed'].valu

#Calculate r2 score
r2=r2_score(predictions['Beer'].values, predictions['Beer_imputed'].values)

MSE,r2
```

Out[140]: (198.71291211265773, 0.836286238218155)

In [59]:
```
imputed_data=imputed.copy()
imputed_data.loc['1972-09-01':'1975-02-01','Beer']=imputed.loc['1972-09-01':'1975-02-0
imputed_data=imputed_data.drop('Beer_imputed',axis=1);
```

In [60]:
```
imputed_data.head()
```

Out[60]:

Date	Beer	Car	Steel	Gas	Electricity	Temp
1956-01-01	93.2	NaN	196.9	1709	1254	25.1
1956-02-01	96.0	NaN	192.1	1646	1290	25.3
1956-03-01	95.2	NaN	201.8	1794	1379	24.9
1956-04-01	77.1	NaN	186.9	1878	1346	23.9
1956-05-01	70.9	NaN	218.0	2173	1535	19.4

In [54]:
```
#Initialize a SimpleImputer model
imputer = datawig.SimpleImputer(
    input_columns=['Beer','Steel','Gas','Electricity','Temp'], # column(s) containing
    output_column='Car', # the column we'd like to impute values for
    output_path = 'imputer_model' # stores model data and metrics
    )
```

In [55]:
```
#Fit an imputer model on the train data
imputer.fit(train_df=imputed_data[imputed_data['Car'].notnull()], num_epochs=300)
```

```
2019-04-20 22:44:42,708 [INFO]  Assuming 5 numeric input columns: Beer, Steel, Gas, Electricity
2019-04-20 22:44:42,710 [INFO]  Assuming 0 string input columns:
2019-04-20 22:44:42,712 [INFO]  No output column name provided for ColumnEncoder using Car
2019-04-20 22:44:42,713 [INFO]  Assuming numeric output column: Car
2019-04-20 22:44:42,715 [INFO]  Using [[cpu(0)]] as the context for training
```

```
2019-04-20 22:44:42,720 [INFO]  Fitting label encoder <class 'datawig.column_encoders.Numerical
2019-04-20 22:44:42,728 [INFO]  Detected 0 rows with missing labels                          fo
2019-04-20 22:44:42,730 [INFO]  Dropping 0/332 rows
2019-04-20 22:44:42,733 [INFO]  Detected 0 rows with missing labels                          fo
2019-04-20 22:44:42,735 [INFO]  Dropping 0/36 rows
2019-04-20 22:44:42,738 [INFO]  Train: 332, Test: 36
2019-04-20 22:44:42,739 [INFO]  Fitting data encoder <class 'datawig.column_encoders.Numerical
2019-04-20 22:44:42,750 [INFO]  Building Train Iterator with 332 elements
2019-04-20 22:44:42,767 [INFO]  Concatenating numeric columns ['Beer', 'Steel', 'Gas', 'Electr
2019-04-20 22:44:42,768 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:42,773 [INFO]  Data Encoding - Encoded 336 rows of column
2019-04-20 22:44:42,778 [INFO]  Concatenating numeric columns ['Car'] into Car
2019-04-20 22:44:42,779 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:42,782 [INFO]  Label Encoding - Encoded 336 rows of column
2019-04-20 22:44:42,783 [INFO]  Building Test Iterator with 36 elements
2019-04-20 22:44:42,816 [INFO]  Concatenating numeric columns ['Beer', 'Steel', 'Gas', 'Electr
2019-04-20 22:44:42,817 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:42,820 [INFO]  Data Encoding - Encoded 48 rows of column
2019-04-20 22:44:42,823 [INFO]  Concatenating numeric columns ['Car'] into Car
2019-04-20 22:44:42,825 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:42,829 [INFO]  Label Encoding - Encoded 48 rows of column
2019-04-20 22:44:42,831 [INFO]  Concatenating all 1 latent symbols
2019-04-20 22:44:42,832 [INFO]  Constructing numerical loss for column Car
2019-04-20 22:44:42,835 [INFO]  Building output symbols
2019-04-20 22:44:42,840 [INFO]
========== start: fit model
2019-04-20 22:44:42,842 [WARNING]  Already bound, ignoring bind()
2019-04-20 22:44:42,870 [INFO]  Epoch[0] Batch [0-11]        Speed: 8823.68 samples/sec
2019-04-20 22:44:42,886 [INFO]  Epoch[0] Train-cross-entropy=13.688891
2019-04-20 22:44:42,888 [INFO]  Epoch[0] Train-Car-accuracy=0.000000
2019-04-20 22:44:42,890 [INFO]  Epoch[0] Time cost=0.043
2019-04-20 22:44:42,909 [INFO]  Saved checkpoint to "imputer_model\model-0000.params"
2019-04-20 22:44:42,914 [INFO]  Epoch[0] Validation-cross-entropy=10.143172
2019-04-20 22:44:42,916 [INFO]  Epoch[0] Validation-Car-accuracy=0.000000
2019-04-20 22:44:42,940 [INFO]  Epoch[1] Batch [0-11]        Speed: 8403.51 samples/sec
2019-04-20 22:44:42,960 [INFO]  Epoch[1] Train-cross-entropy=10.606893
2019-04-20 22:44:42,962 [INFO]  Epoch[1] Train-Car-accuracy=0.000000
2019-04-20 22:44:42,963 [INFO]  Epoch[1] Time cost=0.046
2019-04-20 22:44:42,978 [INFO]  Saved checkpoint to "imputer_model\model-0001.params"
2019-04-20 22:44:42,984 [INFO]  Epoch[1] Validation-cross-entropy=10.206568
2019-04-20 22:44:42,985 [INFO]  Epoch[1] Validation-Car-accuracy=0.000000
2019-04-20 22:44:43,006 [INFO]  Epoch[2] Batch [0-11]        Speed: 9804.33 samples/sec
2019-04-20 22:44:43,022 [INFO]  Epoch[2] Train-cross-entropy=10.107667
2019-04-20 22:44:43,024 [INFO]  Epoch[2] Train-Car-accuracy=0.000000
2019-04-20 22:44:43,027 [INFO]  Epoch[2] Time cost=0.041
2019-04-20 22:44:43,048 [INFO]  Saved checkpoint to "imputer_model\model-0002.params"
2019-04-20 22:44:43,053 [INFO]  Epoch[2] Validation-cross-entropy=10.667156
2019-04-20 22:44:43,055 [INFO]  Epoch[2] Validation-Car-accuracy=0.000000
```

```
2019-04-20 22:44:43,077 [INFO]  Epoch[3] Batch [0-11]       Speed: 9810.06 samples/sec
2019-04-20 22:44:43,094 [INFO]  Epoch[3] Train-cross-entropy=9.943596
2019-04-20 22:44:43,095 [INFO]  Epoch[3] Train-Car-accuracy=0.000000
2019-04-20 22:44:43,096 [INFO]  Epoch[3] Time cost=0.040
2019-04-20 22:44:43,113 [INFO]  Saved checkpoint to "imputer_model\model-0003.params"
2019-04-20 22:44:43,118 [INFO]  No improvement detected for 3 epochs compared to 10.14317186673
2019-04-20 22:44:43,119 [INFO]  Stopping training, patience reached
2019-04-20 22:44:43,121 [INFO]
========== done (0.2812483310699463 s) fit model
2019-04-20 22:44:43,129 [INFO]  Expected calibration error: 100.0%
2019-04-20 22:44:43,136 [INFO]  Expected calibration error after calibration: 100.0%
2019-04-20 22:44:43,144 [INFO]  save metrics in imputer_model\fit-test-metrics.json
2019-04-20 22:44:43,155 [INFO]  Keeping imputer_model\model-0000.params
2019-04-20 22:44:43,157 [INFO]  Deleting imputer_model\model-0001.params
2019-04-20 22:44:43,161 [INFO]  Deleting imputer_model\model-0002.params
2019-04-20 22:44:43,164 [INFO]  Deleting imputer_model\model-0003.params
```

Out[55]: <datawig.simple_imputer.SimpleImputer at 0x28edbbe9860>

In [56]:
```
#Impute missing values and return original dataframe with predictions
imputed_car = imputer.predict(imputed_data)
#imputed.to_csv('./Imputation Results/imputation_Datawig.csv')
```

```
2019-04-20 22:44:56,438 [INFO]  Concatenating numeric columns ['Beer', 'Steel', 'Gas', 'Electri
2019-04-20 22:44:56,439 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:56,443 [INFO]  Data Encoding - Encoded 448 rows of column
2019-04-20 22:44:56,448 [INFO]  Concatenating numeric columns ['Car'] into Car
2019-04-20 22:44:56,451 [INFO]  Normalizing with StandardScaler
2019-04-20 22:44:56,454 [INFO]  Label Encoding - Encoded 448 rows of column
2019-04-20 22:44:56,472 [INFO]  Top-k only for CategoricalEncoder, dropping Car, <class 'datawi
2019-04-20 22:44:56,473 [INFO]  Precision filtering only for CategoricalEncoder returning
```

In [142]:
```
imputed_data_final=imputed_car.copy()
imputed_data_final.loc['1956-01-01':'1961-06-01','Car']=\
imputed_car.loc['1956-01-01':'1961-06-01']['Car_imputed'].values
imputed_data_final=imputed_data_final.drop('Car_imputed',axis=1)
imputed_data_final.to_csv('data_merged_final.csv');
```

scenario4

April 22, 2019

```
In [19]: import pandas as pd
         from statsmodels.tsa.stattools import ccf
         import matplotlib.pyplot as plt
         from scipy.signal import correlate
         import numpy as np
         import statsmodels as sm
         from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
         import seaborn as sns
         from sklearn.preprocessing import MinMaxScaler
         from pandas import DataFrame
         from pandas import concat
         from keras.models import Sequential
         from keras.layers import Dense
         from keras.layers import LSTM
         from numpy import concatenate
         from math import sqrt
         from sklearn.metrics import mean_squared_error

         %matplotlib inline
```

```
In [3]: # convert series to supervised learning
        def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
            n_vars = 1 if type(data) is list else data.shape[1]
            df = DataFrame(data)
            cols, names = list(), list()
            # input sequence (t-n, ... t-1)
            for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
            # forecast sequence (t, t+1, ... t+n)
            for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                    names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                    names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
            # put it all together
            agg = concat(cols, axis=1)
```

```
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [4]:
```
#define function for ADF test
from statsmodels.tsa.stattools import adfuller
def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

In [36]:
```
def tsplot(y, title, lags=None, figsize=(12, 6)):
    fig = plt.figure(figsize=figsize)
    layout = (2, 2)
    ts_ax = plt.subplot2grid(layout, (0, 0))
    hist_ax = plt.subplot2grid(layout, (0, 1))
    acf_ax = plt.subplot2grid(layout, (1, 0))
    pacf_ax = plt.subplot2grid(layout, (1, 1))
    plt.text(1,-1.4, "Fig. 4.3", size=12, ha="center",weight='bold');
    y.plot(ax=ts_ax)
    ts_ax.set_title(title, fontsize=12, fontweight='bold')
    y.plot(ax=hist_ax, kind='hist', bins=25)
    hist_ax.set_title('Histogram')
    plot_acf(y, lags=lags, ax=acf_ax)
    plot_pacf(y, lags=lags, ax=pacf_ax)
    sns.despine()
    plt.tight_layout()
    plt.show()
    return ts_ax, acf_ax, pacf_ax
```

In [6]:
```
dataset = pd.read_csv('../Scenario3/data_merged_final.csv',index_col=0)
```

In [7]:
```
dataset.head()
```

Out[7]:

	Beer	Car	Steel	Gas	Electricity	Temp
Date						
1956-01-01	93.2	12700.116925	196.9	1709	1254	25.1
1956-02-01	96.0	12574.354195	192.1	1646	1290	25.3
1956-03-01	95.2	13050.102235	201.8	1794	1379	24.9
1956-04-01	77.1	11604.703762	186.9	1878	1346	23.9
1956-05-01	70.9	13700.668520	218.0	2173	1535	19.4

In [8]:
```
CsI=dataset['Beer']
WLS=dataset['Steel']
```

In [38]:
```
import scipy.signal as ss
import numpy as np
import matplotlib.pyplot as plt

maxlags = 10
result = result = ss.correlate(CsI - np.mean(CsI), WLS - np.mean(WLS), method='direct
lo = (len(result)-1)//2-10 #just get +/- 10 elements around lag 0
hi = (len(result)-1)//2+11

locs = np.arange(lo, hi)
# for loc in locs:
#     print(str(loc)+'\t:\t'+str(result[loc]))

#Make a plot like ccf
f, ax = plt.subplots(figsize=(12,6))
ax.stem(np.arange(-10,11), result[lo:hi], '-.')
ax.set_xticks(np.arange(-10,11))
ax.text(0.5,-0.15, "Fig. 4.1", size=12, ha="center", transform=ax.transAxes,weight='bo

plt.show()
```

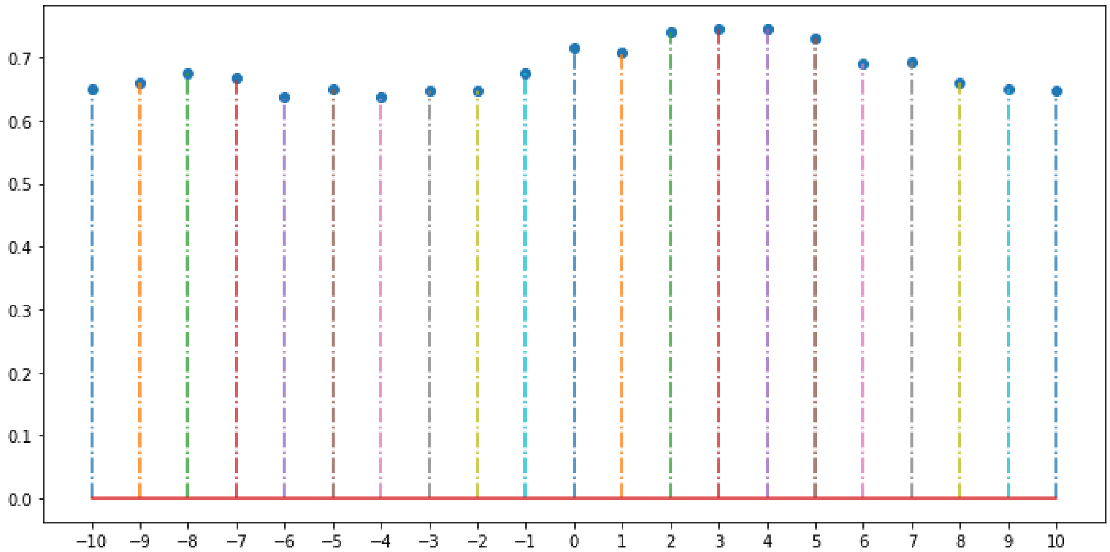


Fig. 4.1

In [29]:
```
dataset.plot(subplots=True,figsize=(12,12))
plt.text(6,-1.4, "Fig. 4.2", size=12, ha="center",weight='bold');
```

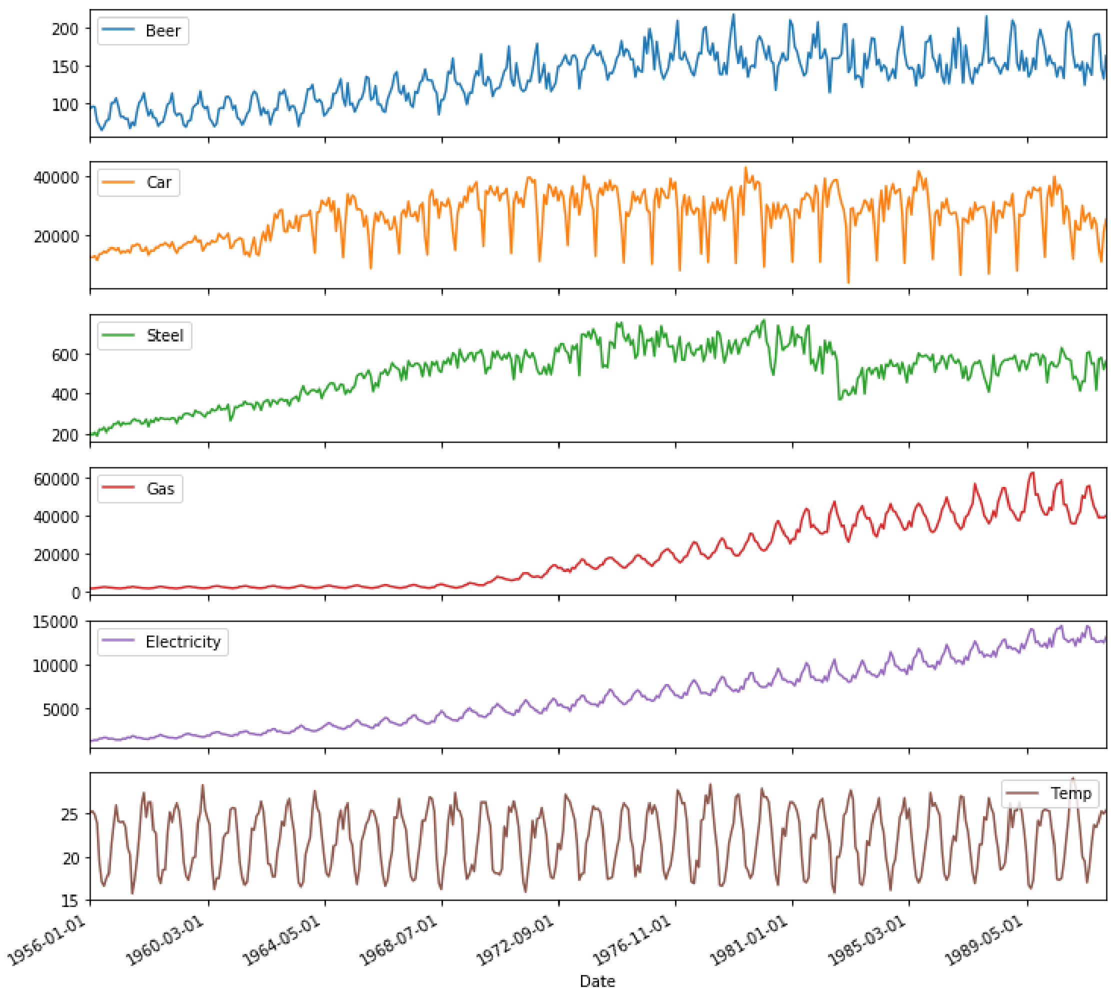


Fig. 4.2

```
In [37]: tsplot(dataset['Gas'],'Gas',lags=np.arange(0,40))
```

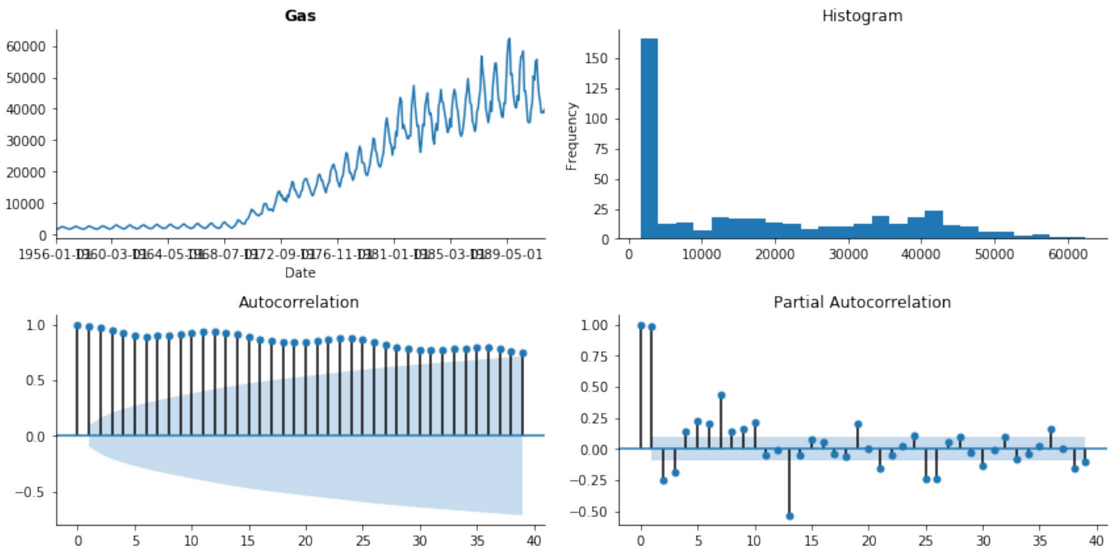


Fig. 4.3

```
Out[37]: (<matplotlib.axes._subplots.AxesSubplot at 0x1dffa719128>,
          <matplotlib.axes._subplots.AxesSubplot at 0x1dff7f155c0>,
          <matplotlib.axes._subplots.AxesSubplot at 0x1dff9e589e8>)
```

```
In [39]: adf_test(dataset['Temp'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                -6.333879e+00
p-value                        2.862096e-08
#Lags Used                     1.800000e+01
Number of Observations Used    4.160000e+02
Critical Value (1%)           -3.446168e+00
Critical Value (5%)           -2.868513e+00
Critical Value (10%)          -2.570484e+00
dtype: float64
```

For temperature, the test statistic < critical value, which implies that the series is stationary.

```
In [40]: adf_test(dataset['Car'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.737860
p-value                          0.067713
#Lags Used                      13.000000
Number of Observations Used    421.000000
Critical Value (1%)             -3.445979
```

```
Critical Value (5%)             -2.868430
Critical Value (10%)            -2.570440
dtype: float64
```

For car, the test statistic < critical value in 10% significance level, which implies that the series is stationary. The cas has growth in the beginning but tends to stationay later.

```
In [41]: adf_test(dataset['Steel'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.252618
p-value                          0.187705
#Lags Used                      12.000000
Number of Observations Used    422.000000
Critical Value (1%)             -3.445941
Critical Value (5%)             -2.868413
Critical Value (10%)            -2.570431
dtype: float64
```

```
In [42]: adf_test(dataset['Gas'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                   0.205778
p-value                          0.972584
#Lags Used                      17.000000
Number of Observations Used    417.000000
Critical Value (1%)             -3.446129
Critical Value (5%)             -2.868496
Critical Value (10%)            -2.570475
dtype: float64
```

```
In [43]: adf_test(dataset['Electricity'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                   1.563761
p-value                          0.997745
#Lags Used                      17.000000
Number of Observations Used    417.000000
Critical Value (1%)             -3.446129
Critical Value (5%)             -2.868496
Critical Value (10%)            -2.570475
dtype: float64
```

For the above three, transformation to stationary is needed.

```
In [44]: dataset['Steel_log'] = np.log(dataset['Steel'])
         dataset['Steel_log_diff'] = dataset['Steel_log'] - dataset['Steel_log'].shift(1)
         adfuller(dataset['Steel_log_diff'].dropna())
```

```
Out[44]: (-6.287739626234377,
          3.662730800933948e-08,
          13,
          420,
          {'1%': -3.4460159927788574,
           '10%': -2.570448781179138,
           '5%': -2.868446209372638},
          -897.0581197796027)
```

Use log transform for steel.

```
In [45]: dataset['Gas_diff_seas'] = dataset['Gas'] - dataset['Gas'].shift(12)
         dataset['Gas_diff'] = dataset['Gas_diff_seas'] - dataset['Gas_diff_seas'].shift(1)
```

```
In [46]: adfuller(dataset['Gas_diff'].dropna())
```

```
Out[46]: (-7.423805825441933,
          6.629388763872179e-11,
          18,
          403,
          {'1%': -3.4466811208382437,
           '10%': -2.5706046655665635,
           '5%': -2.8687386420385494},
          6970.282319837302)
```

For gas, first seasonal difference. then difference by 1.

```
In [47]: dataset['Electricity_diff_seas'] = dataset['Electricity'] - dataset['Electricity'].shi
         dataset['Electricity_diff'] = dataset['Electricity_diff_seas'] - dataset['Electricity_
```

```
In [48]: adfuller(dataset['Electricity_diff'].dropna())
```

```
Out[48]: (-6.32664313158624,
          2.9751670322101644e-08,
          17,
          404,
          {'1%': -3.44664043608676,
           '10%': -2.5705951311145965,
           '5%': -2.868720756230461},
          5275.094116151984)
```

```
In [49]: dataset_stationary=dataset.drop(['Steel','Gas','Electricity',\
                                          'Steel_log','Gas_diff_seas','Electricity_diff_seas']
```

```
In [50]: dataset_stationary.head()
```

```
Out[50]:             Beer          Car  Temp  Steel_log_diff  Gas_diff  \
         Date
         1957-02-01  82.8  13985.911224  24.0       -0.092787       0.0
         1957-03-01  83.3  14767.651312  24.1        0.069590      84.0
         1957-04-01  80.0  14270.452770  23.5       -0.030697     -63.0
         1957-05-01  80.4  15046.366905  21.1        0.014658      75.0
         1957-06-01  67.5  14187.495032  20.3       -0.008525    -180.0

                     Electricity_diff
         Date
         1957-02-01             -58.0
         1957-03-01              67.0
         1957-04-01              -8.0
         1957-05-01               2.0
         1957-06-01             -97.0
```

```
In [51]: values = dataset_stationary.values
         # normalize features
         scaler = MinMaxScaler(feature_range=(0, 1))
         scaled = scaler.fit_transform(values)
```

```
In [451]: # specify the number of lag hours
          n_days = 24
          n_features = 6

          # frame as supervised learning
          reframed = series_to_supervised(scaled, n_days, n_days)

          reframed.to_csv('input_LSTM.csv')
```

```
In [428]: # split into train and test sets
          values = reframed.values
          n_train_days = 330
          train = values[:n_train_days, :]
          test = values[n_train_days:, :]
```

```
In [429]: # split into input and outputs
          n_obs = n_days * n_features
          train_X, train_y = train[:, :n_obs], train[:,n_obs::n_features]
          test_X, test_y = test[:, :n_obs], test[:,n_obs::n_features]
          print(train_X.shape, len(train_X), train_y.shape)

(330, 144) 330 (330, 24)
```

```
In [430]: # reshape input to be 3D [samples, timesteps, features]
          train_X = train_X.reshape((train_X.shape[0], n_days, n_features))
          test_X = test_X.reshape((test_X.shape[0], n_days, n_features))
          print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(330, 24, 6) (330, 24) (45, 24, 6) (45, 24)
```

```
In [473]: train_X_all, train_y_all = values[:, :n_obs], values[:,n_obs::n_features]
          train_X_all = train_X_all.reshape((train_X_all.shape[0], n_days, n_features))
```

```
In [476]: train_X_all.shape,train_y_all.shape
```

```
Out[476]: ((375, 24, 6), (375, 24))
```

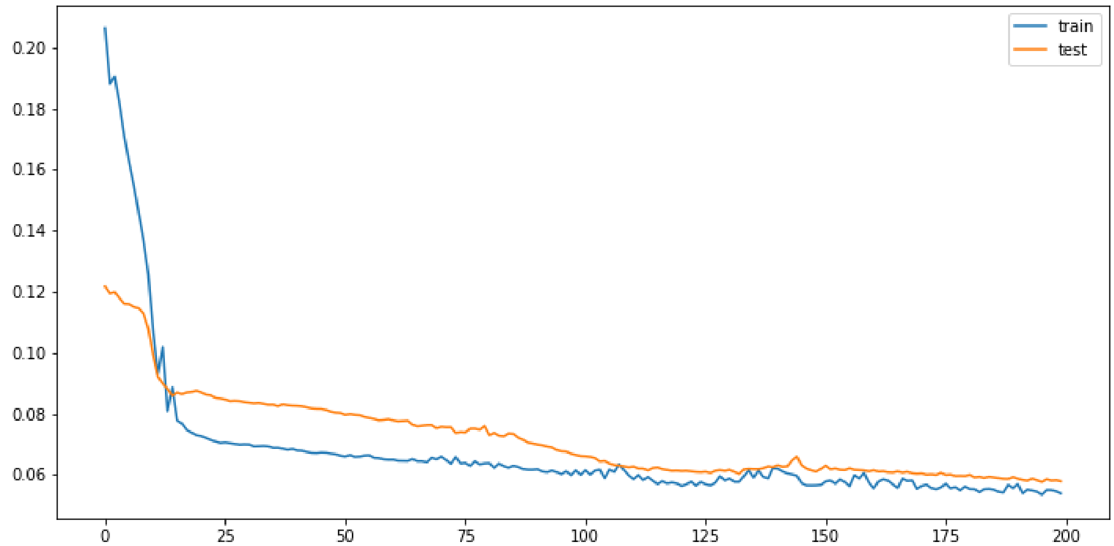
```
In [477]: # design network
          model = Sequential()
          model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
          model.add(Dense(n_days))
          model.compile(loss='mae', optimizer='adam')
          # fit network
          history = model.fit(train_X_all, train_y_all, epochs=200, batch_size=10, validation_d
```

```
Train on 375 samples, validate on 45 samples
Epoch 1/200
 - 1s - loss: 0.2063 - val_loss: 0.1217
Epoch 2/200
 - 0s - loss: 0.1881 - val_loss: 0.1194
Epoch 3/200
 - 0s - loss: 0.1904 - val_loss: 0.1199
Epoch 4/200
 - 0s - loss: 0.1815 - val_loss: 0.1178
Epoch 5/200
 - 0s - loss: 0.1710 - val_loss: 0.1160
Epoch 6/200
 - 0s - loss: 0.1624 - val_loss: 0.1159
Epoch 7/200
 - 0s - loss: 0.1547 - val_loss: 0.1150
Epoch 8/200
 - 0s - loss: 0.1457 - val_loss: 0.1145
Epoch 9/200
 - 0s - loss: 0.1368 - val_loss: 0.1129
Epoch 10/200
 - 0s - loss: 0.1256 - val_loss: 0.1078
Epoch 11/200
 - 0s - loss: 0.1072 - val_loss: 0.0992
Epoch 12/200
 - 0s - loss: 0.0930 - val_loss: 0.0921
Epoch 13/200
 - 0s - loss: 0.1019 - val_loss: 0.0898
Epoch 14/200
 - 0s - loss: 0.0808 - val_loss: 0.0882
Epoch 15/200
 - 0s - loss: 0.0887 - val_loss: 0.0859
```

```
Epoch 184/200
 - 1s - loss: 0.0552 - val_loss: 0.0589
Epoch 185/200
 - 1s - loss: 0.0554 - val_loss: 0.0593
Epoch 186/200
 - 1s - loss: 0.0551 - val_loss: 0.0591
Epoch 187/200
 - 1s - loss: 0.0544 - val_loss: 0.0589
Epoch 188/200
 - 1s - loss: 0.0541 - val_loss: 0.0587
Epoch 189/200
 - 1s - loss: 0.0567 - val_loss: 0.0586
Epoch 190/200
 - 1s - loss: 0.0556 - val_loss: 0.0592
Epoch 191/200
 - 1s - loss: 0.0570 - val_loss: 0.0587
Epoch 192/200
 - 1s - loss: 0.0539 - val_loss: 0.0583
Epoch 193/200
 - 1s - loss: 0.0551 - val_loss: 0.0581
Epoch 194/200
 - 0s - loss: 0.0548 - val_loss: 0.0587
Epoch 195/200
 - 0s - loss: 0.0544 - val_loss: 0.0582
Epoch 196/200
 - 0s - loss: 0.0535 - val_loss: 0.0577
Epoch 197/200
 - 0s - loss: 0.0550 - val_loss: 0.0585
Epoch 198/200
 - 0s - loss: 0.0550 - val_loss: 0.0581
Epoch 199/200
 - 0s - loss: 0.0545 - val_loss: 0.0582
Epoch 200/200
 - 0s - loss: 0.0539 - val_loss: 0.0579
```

```
In [478]: # plot history
          fig=plt.figure(figsize=(12,6))
          plt.plot(history.history['loss'], label='train')
          plt.plot(history.history['val_loss'], label='test')
          plt.text(6,-1.4, "Fig. 4.3", size=12, ha="center",weight='bold');
          plt.legend()
```

```
Out[478]: <matplotlib.legend.Legend at 0x1cb69ba8438>
```




```
In [479]: test_y[-1:]

Out[479]: array([[0.57351963, 0.56819694, 0.45242848, 0.53892216, 0.53825682,
                  0.43978709, 0.84231537, 0.93878909, 0.86360612, 0.64471058,
                  0.53626081, 0.56220892, 0.50964737, 0.582169  , 0.37924152,
                  0.56886228, 0.52228876, 0.46906188, 0.81503659, 0.82834331,
                  0.82834331, 0.52894212, 0.43579508, 0.63539587]])

In [480]: test_X.shape,yhat.shape,test_X_res.shape,test_y.shape

Out[480]: ((45, 24, 6), (1, 24), (45, 144), (45, 24))

In [481]: yhat=model.predict(test_X[-1:])
          test_X_res = test_X.reshape((test_X.shape[0], n_days*n_features))

          # invert scaling for forecast

          yhat_last24=[]
          ytrue_last24=[]

          for i in np.arange(n_days):
              inv_yhat = concatenate((yhat[-1:,[i]], test_X_res[-1:, -5:]), axis=1)
              inv_yhat = scaler.inverse_transform(inv_yhat)
              inv_yhat = inv_yhat[:,0]
              yhat_last24.append(inv_yhat)

          for i in np.arange(n_days):
              inv_y = concatenate((test_y[-1:,[i]], test_X_res[-1:, -5:]), axis=1)
              inv_y = scaler.inverse_transform(inv_y)
```

```
        inv_y = inv_y[:,0]
        ytrue_last24.append(inv_y)
```

In [482]:
```
yhat=model.predict(np.reshape(test[-1,n_obs:],newshape=(1,n_days,n_features)))
test_X_res = test_X.reshape((test_X.shape[0], n_days*n_features))

# invert scaling for forecast

yhat_future=[]

for i in np.arange(n_days):
    inv_yhat = concatenate((yhat[-1:,[i]], test_X_res[-1:, -5:]), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:,0]
    yhat_future.append(inv_yhat)
```

In [483]:
```
yhat_future
```

Out[483]:
```
[array([152.51552664]),
 array([138.85869391]),
 array([134.16951433]),
 array([144.89787746]),
 array([148.69925299]),
 array([151.05538591]),
 array([163.66366187]),
 array([196.96757017]),
 array([185.83429722]),
 array([156.73270562]),
 array([149.34777342]),
 array([161.02483206]),
 array([146.99964051]),
 array([142.90194209]),
 array([133.63986527]),
 array([148.69771212]),
 array([151.19717333]),
 array([146.61835447]),
 array([175.58979585]),
 array([198.8434158]),
 array([192.79560513]),
 array([155.76620046]),
 array([150.37765156]),
 array([162.13171813])]
```

In [485]:
```
np.savetxt('../Result in CSV/Li_Scenario4.csv', yhat_future, delimiter=',')
```

0.0.1 Prediction Interval

In [52]:
```
import csv
import itertools
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import statsmodels.api as sm
```

In [53]:
```
Beer=pd.read_csv('../Scenario3/data_merged_final.csv',sep=',',index_col=0,usecols=[0,
```

In [55]:
```
mod = sm.tsa.statespace.SARIMAX(Beer,
                                order=(1, 0, 1),
                                enforce_stationarity=False,
                                enforce_invertibility=False,)

results = mod.fit()
print(results.summary())
```

```
                           Statespace Model Results
==============================================================================
Dep. Variable:                   Beer   No. Observations:                  435
Model:               SARIMAX(1, 0, 1)   Log Likelihood               -1872.418
Date:                Mon, 22 Apr 2019   AIC                           3750.837
Time:                        22:36:31   BIC                           3763.049
Sample:                    01-01-1956   HQIC                          3755.658
                         - 03-01-1992
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9968      0.004    237.065      0.000       0.989       1.005
ma.L1         -0.3107      0.044     -7.066      0.000      -0.397      -0.225
sigma2       335.4263     21.510     15.594      0.000     293.267     377.585
===================================================================================
Ljung-Box (Q):                      824.88   Jarque-Bera (JB):                 3.08
Prob(Q):                              0.00   Prob(JB):                         0.21
Heteroskedasticity (H):               3.70   Skew:                             0.04
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.41
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).


C:\Users\Jackie Li\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:171: ValueWar
  % freq, ValueWarning)
C:\Users\Jackie Li\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:191: FutureWa
  start=index[0], end=index[-1], freq=freq)
```

In [57]:
```
# Get forecast 24 steps ahead in future
pred_uc = results.get_forecast(steps=24)

# Get 95% confidence intervals of forecasts
pred_ci = pred_uc.conf_int(alpha=0.05)
```

```
C:\Users\Jackie Li\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:320: FutureWa
  freq=base_index.freq)
```

```
In [58]: s=pd.DataFrame(pred_uc.predicted_mean)
         s.columns=['Beer']
         s.set_index(s.columns[0])
         s.index.name='Date'
         s.index=s.index.to_period('D')
```

```
In [59]: Beer.index = pd.to_datetime(Beer.index)
         Beer.index=Beer.index.to_period('D')
```

```
In [64]: fig, ax = plt.subplots(figsize=(10,6))
         Beer.append(s)[300:].plot(ax=ax)
         ax.fill_between(pred_ci.index,
                         pred_ci.iloc[:, 0],
                         pred_ci.iloc[:, 1], color='k', alpha=.25)
         ax.text(0.5,-0.15, "Fig. 4.5", size=12, ha="center", transform=ax.transAxes,weight='bo
         plt.legend(['Forcast','Prediction Interval'],loc=2)
         plt.show()
```

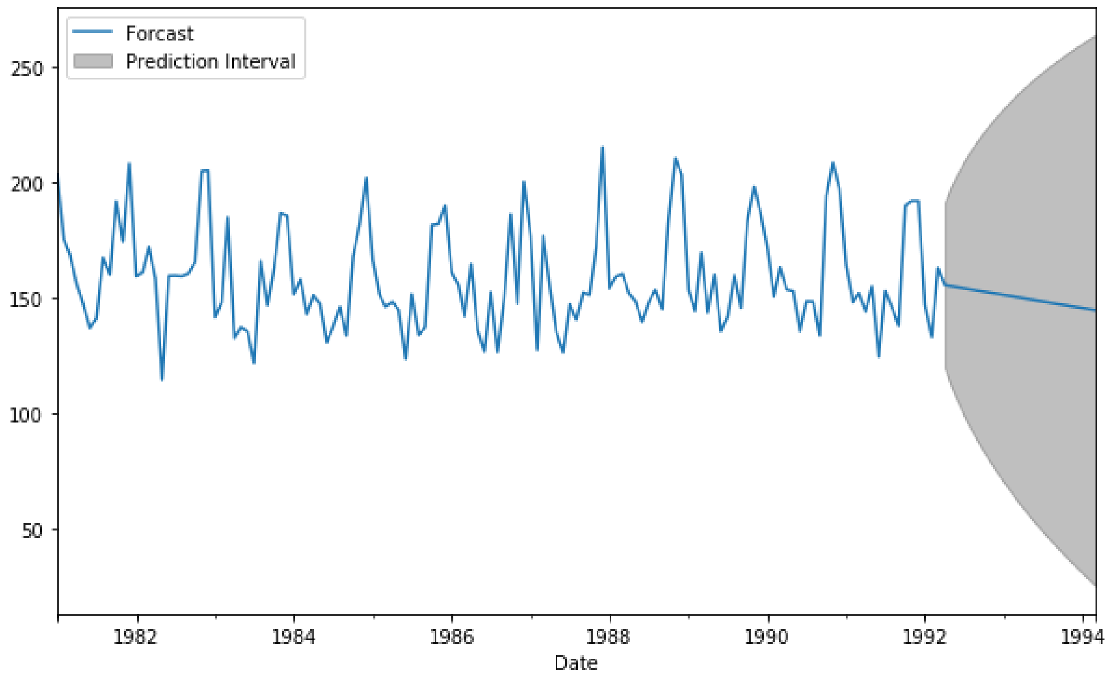


Fig. 4.5