ECON 430 Code Reference

December 6, 2021

1 Likelihood Inference

We will estimate the parameter (λ) of an Exponential Distribution using 3 different methods:
>MLE
>Method of Moments
>Bootstrapping

[107]:
```
# Generate a random data set of size = 100 from the exponential distribution␣
↪with mean = 1/2
exp_data = np.random.exponential(1/2, size=1000)

# Construct histogram of exp_data
num_bins = 11
fig, axs = plt.subplots(1, 1)
axs.hist(exp_data, num_bins, color ='white', edgecolor = 'black', density =␣
↪True)

# Set the range of x in the second subplot
axs.set_xlim(0,3)
axs.set_title('Density Plot', fontsize = 14, fontweight = 'bold')

# Draw the density line
sns.kdeplot(exp_data)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[107]:
```
<AxesSubplot:title={'center':'Density Plot'}, ylabel='Density'>
```

1.0.1 Maximum Likelihood Estimator (MLE)

Suppose we are given the data we generated above. That is, we collected the data above. We know that it was generated from the exponential distribution, but do not know the scale parameter, λ. We will numerically approximate the MLE for λ by evaluating the likelihood function at 1000 equispaced points in (1,3]. Also, plot the likelihood function.

[102]:
```
# Defining log-like function to estimate MLE
def MLE(scale, data):
```

```
    return len(data)*np.log(scale) - scale*sum(data)

# Generating 1000 equsipaced points in (1,3]
Scale = np.arange(1, 3, 1/1000)

# Setting list of likelihoods to MLE function
Likelihoods = MLE(Scale, exp_data)

# Print maximizing value of "Likelihoods"
print(Scale[Likelihoods.argmax()])

# Plot of log-likelhihood function
plt.plot(Scale, Likelihoods, color = 'red')
plt.title('Log-Likelihood Function', fontsize = 14, fontweight = 'bold')
```

```
2.0909999999988
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[102]:

```
Text(0.5, 1.0, 'Log-Likelihood Function')
```

1.0.2 Method of Moments (MOM)

What is the estimated λ using the method of moments? Note: You need to first solve the analytical solution.

[101]:

```
# Remember that E(X) = 1/(Lambda) in an exponential distribution
# and that E(Lamda) = 1/(Mean)
EstimatedLambda = 1/np.mean(exp_data)
print("The estimated lamda using the method of moments is", EstimatedLambda)
```

```
The estimated lamda using the method of moments is 2.0911038819033716
```

1.0.3 Bootstrapping Likelihood

Suppose now that we are given the data we generated above, but the distribution itself is unknown. What is the estimated mean by bootstrapping and its standard error?

[104]:

```
# Transform exp_data into a DataFrame
exp_df = pd.DataFrame(exp_data)

# Defining function to estimate mean of data via bootstrapping
def bootstrap_mean(data):

    # Constructing empty list of 0 to later fill with entries
    BootstrapList = [0 for x in range(100)]

    for i in range(100):
```

```
        temp = data.sample(n=1000, replace=True)
        sum_mean = np.mean(temp)
        BootstrapList[i] = sum_mean

    BestMean = np.mean(BootstrapList)
    BestSE = np.sqrt(np.var(BootstrapList))
    BootstrapStoredValues = [BestMean, BestSE]

    return BootstrapStoredValues

bootstrap_mean(exp_df)

print("The estimated mu is", bootstrap_mean(exp_df)[0])
print("The standard error of the estimator is", bootstrap_mean(exp_df)[1])
```

```
The estimated mu is 0.4812097226220871
The standard error of the estimator is 0.015338206362375954
```

2 Stochastic Processes

2.0.1 Simple Random Walk

[94]:
```
from random import seed
from random import random
from matplotlib import pyplot as plt

random_walk = list()
random_walk.append(-1 if random() < 0.5 else 1)
for i in range(1, 1000):
    movement = -1 if random() < 0.5 else 1
    value = random_walk[i-1] + movement
    random_walk.append(value)
plt.plot(random_walk)
plt.show()
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

2.0.2 Simple Random Walk Without Loops

[96]:
```
import matplotlib.pyplot as plt
import numpy as np

fig2, axes2 = plt.subplots(1,1)
axes2.plot(np.random.randn(50).cumsum(),'k-', label = 'Sample Data1')
axes2.legend(loc = 'best')
axes2.set_title('Random Walk Example')
```

```
axes2.set_xlabel('Index')
axes2.set_ylabel('Value')
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[96]: Text(0, 0.5, 'Value')

2.0.3 Markov Chain

Consider a worker who, at any given time t, is either unemployed (state 0) or employed (state 1).

Suppose that, over a one month period,

1 An unemployed worker finds a job with probability $\alpha \in (0, 1)$
2 An employed worker loses her job and becomes unemployed with probability $\beta \in (0, 1)$

In terms of a Markov model, we have

$$S = \{0, 1\}, \quad P(0, 1) = \alpha, \quad P(1, 0) = \beta$$

We can write out the transition probabilities in matrix form as

$$P = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix}$$

Once we have the values α and β, we can address a range of questions, such as

What is the average duration of unemployment?
Over the long-run, what fraction of time does a worker find herself unemployed?
Conditional on employment, what is the probability of becoming unemployed at least once over the next 12 months?

[108]:
```
import quantecon as qe
from quantecon import MarkovChain

import quantecon as qe
from quantecon import MarkovChain

# Transition Probability Matrix
P = [[0.4, 0.6],
     [0.2, 0.8]]

mc = qe.MarkovChain(P, state_values=('unemployed', 'employed'))
mc.simulate(ts_length=10000, init='employed')
```

[108]:
```
array(['employed', 'unemployed', 'unemployed', …, 'employed',
       'employed', 'unemployed'], dtype='<U10')
```

[109]:
```
# Check if the MC is irreducible (i.e., can reach any state from any other␣
↪state)
mc.is_irreducible
```

[109]: True

[112]:
```
# Check if the MC is periodic or aperiodic
mc.is_aperiodic
```

[112]: True

```
C:\Users\tanne\anaconda3\lib\site-
packages\scipy\stats\_continuous_distns.py:4530: IntegrationWarning: The
integral is probably divergent, or slowly convergent.
  intg = integrate.quad(f, -xi, np.pi/2, **intg_kwargs)[0]
```

[114]:
```
# Check the stationary (steady-state) distribution of P
mc.stationary_distributions
# the first value is the probability of being unemployed, and the second one,␣
↪the probability of being employed
# Note: Prob of unemployment = p = alpha/(alpha + beta), so as beta -> 0, p ->␣
↪0, and as alpha -> 0, p -> 1
```

[114]: array([[0.25, 0.75]])

[115]:
```
 =  = 0.1
N = 10000
p =  / ( + )

P = ((1 - ,       ),              # Careful: P and p are distinct
     (    ,   1 - ))
mc = MarkovChain(P)

fig, ax = plt.subplots(figsize=(7, 4))
ax.set_ylim(-0.3, 0.3)
ax.grid()
ax.hlines(0, 0, N, lw=2, alpha=0.6)   # Horizonal line at zero

for x0, col in ((0, 'blue'), (1, 'green')):
    # Generate time series for worker that starts at x0
    X = mc.simulate(N, init=x0)
    # Compute fraction of time spent unemployed, for each n
    X_bar = (X == 0).cumsum() / (1 + np.arange(N, dtype=float))
    # Plot
    ax.fill_between(range(N), np.zeros(N), X_bar - p, color=col, alpha=0.1)
    ax.plot(X_bar - p, color=col, label=f'$X_0 = \, {x0} $')
    # Overlay in black--make lines clearer
    ax.plot(X_bar - p, 'k-', alpha=0.6)
```

```
ax.legend(loc='upper right')
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

3 Linear Regression

3.0.1 Evaluating Linearity of the OLS Model

[82]:
```
import statsmodels.formula.api as smf

df = woo.dataWoo('wage1')

# Specifying the type of OLS Model
OLS_Model = smf.ols(formula = 'wage ~ exper', data = df)

# Fitting the OLS Model
# "dir(OLS_Fit)" to loot at directory of other available attributes
OLS_Fit = OLS_Model.fit()

# Showing the OLS Model fit summary
print(OLS_Fit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   wage   R-squared:                       0.013
Model:                            OLS   Adj. R-squared:                  0.011
Method:                 Least Squares   F-statistic:                     6.766
Date:                Mon, 06 Dec 2021   Prob (F-statistic):            0.00955
Time:                        18:32:50   Log-Likelihood:                -1429.7
No. Observations:                 526   AIC:                             2863.
Df Residuals:                     524   BIC:                             2872.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      5.3733      0.257     20.908      0.000       4.868       5.878
exper          0.0307      0.012      2.601      0.010       0.008       0.054
==============================================================================
Omnibus:                      222.603   Durbin-Watson:                   1.808
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              851.542
Skew:                           1.962   Prob(JB):                    1.23e-185
Kurtosis:                       7.843   Cond. No.                         35.0
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[83]:
```python
import statsmodels.stats.api as sms

Name = ["t-stat","p-value"]
Test = sms.linear_harvey_collier(OLS_Fit)
print("Linearity Test Results:")
print(list(zip(Name, Test)))
```

```
Linearity Test Results:
[('t-stat', -3.2525062855138223), ('p-value', 0.0012178806000892786)]
```

3.0.2 Evaluating Normality of Residuals of the OLS Model

[85]:
```python
import statsmodels.stats.api as sms

Name = ["Jarque-Bera", "Chi^2 two-tail prob.", "Skew", "Kurtosis"]
Test = sms.jarque_bera(OLS_Fit.resid)
print("Jarque-Bera Results:")
print(list(zip(Name, Test)))
```

```
Jarque-Bera Results:
[('Jarque-Bera', 851.5424757340136), ('Chi^2 two-tail prob.',
1.229987906093463e-185), ('Skew', 1.962381539797588), ('Kurtosis',
7.8425080125868005)]
```

3.0.3 Evaluating Heteroskedasticity of the OLS Model

[86]:
```python
import statsmodels.stats.api as sms

Name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
Test = sms.het_breuschpagan(OLS_Fit.resid, OLS_Fit.model.exog)
print("Breush-Pagan Results:")
print(list(zip(Name, Test)))
```

```
Breush-Pagan Results:
[('Lagrange multiplier statistic', 5.73647968384148), ('p-value',
0.016616063038612978), ('f-value', 5.777678497439663), ('f p-value',
0.016577290872715044)]
```

3.0.4 Box-Cox Transformations

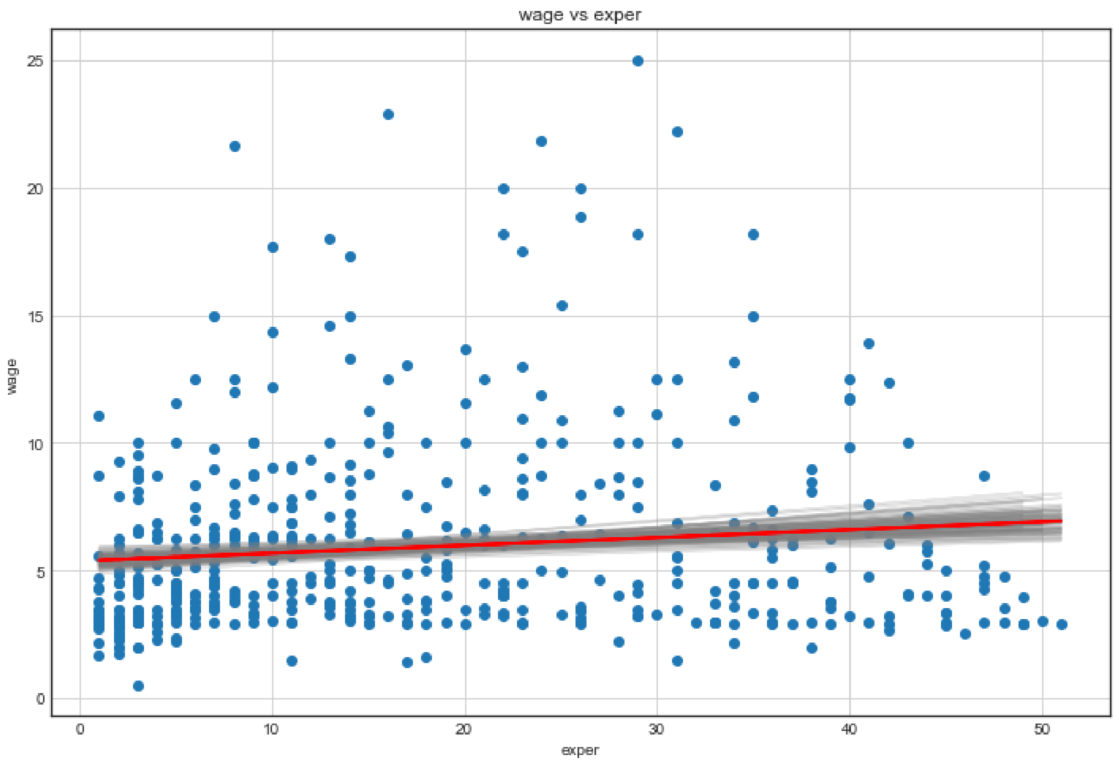
[ ]:
```python
# Transformation of non-normal dependent variable into a normally-distributed␣
↪shape
bc_wage,lambda_wage = sp.stats.boxcox(df["wage"])
print(lambda_wage)
```

```
sns.histplot(bc_wage)
plt.title("Box-Cox Transformed: wage")
plt.show()

bc_exper,lambda_exper = sp.stats.boxcox(df["exper"])
print(lambda_exper)

sns.histplot(bc_exper)
plt.title("Box-Cox Transformed: exper")
plt.show()
```

3.0.5 Bootstrap Estimates

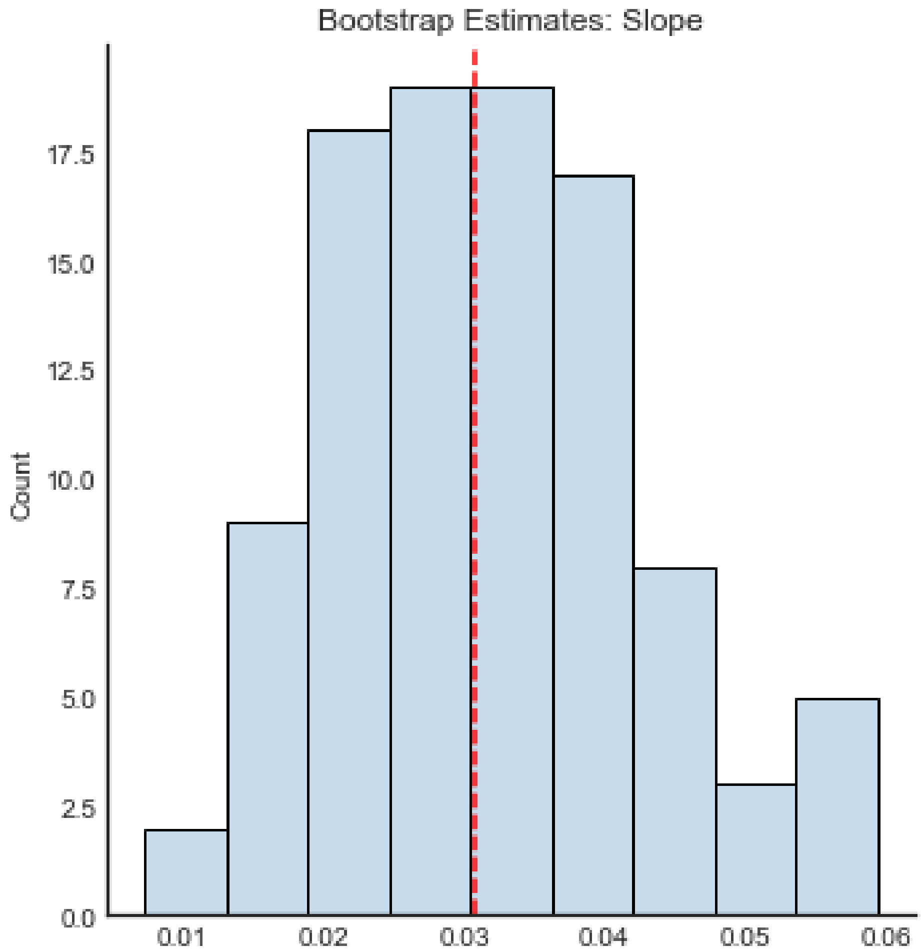
[87]:
```
# resample with replacement each row
boot_slopes = []
boot_interc = []
boot_adjR2 = []
n_boots = 100
n_points = df.shape[0]
plt.figure()

for _ in range(n_boots):
 # sample the rows, same size, with replacement
    sample_df = df.sample(n=n_points, replace=True)

 # fit a linear regression
    ols_model_temp = smf.ols(formula = 'wage ~ exper', data=sample_df)
    results_temp = ols_model_temp.fit()

 # append coefficients
    boot_interc.append(results_temp.params[0])
    boot_slopes.append(results_temp.params[1])
    boot_adjR2.append(results_temp.rsquared_adj)

 # plot a greyed out line
    y_pred_temp = ols_model_temp.fit().predict(sample_df['exper'])
    plt.plot(sample_df['exper'], y_pred_temp, color='grey', alpha=0.2)

# add data points
y_pred = OLS_Model.fit().predict(df['exper'])
plt.scatter(df['exper'], df['wage'])
plt.plot(df['exper'], y_pred, linewidth=2,color = 'red')
plt.grid(True)
plt.xlabel('exper')
plt.ylabel('wage')
plt.title('wage vs exper')
```

```
plt.show()
```



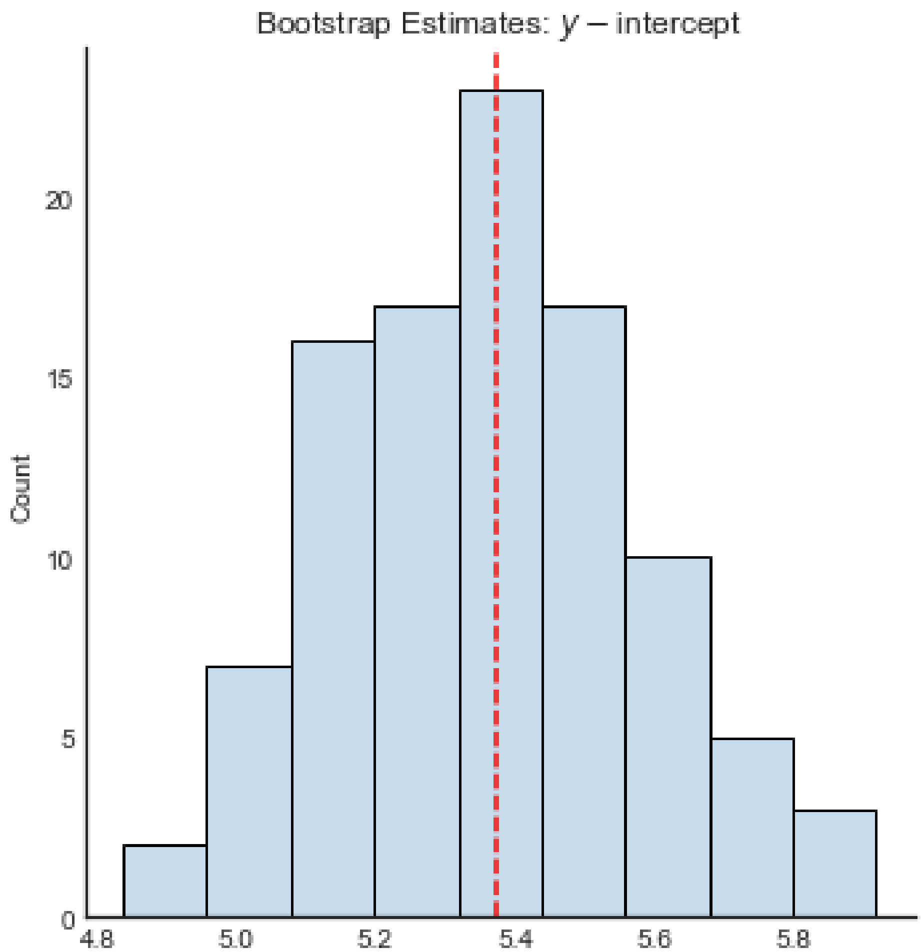
3.0.6 Bootstrap Estimates (Slope)

[88]:
```
sns.displot(boot_slopes, alpha = 0.25)
plt.axvline(x=0.0307,color='red', linestyle='--')
plt.title('Bootstrap Estimates: Slope')
plt.show()
```



3.0.7 Bootstrap Estimates (Y-Intercept)

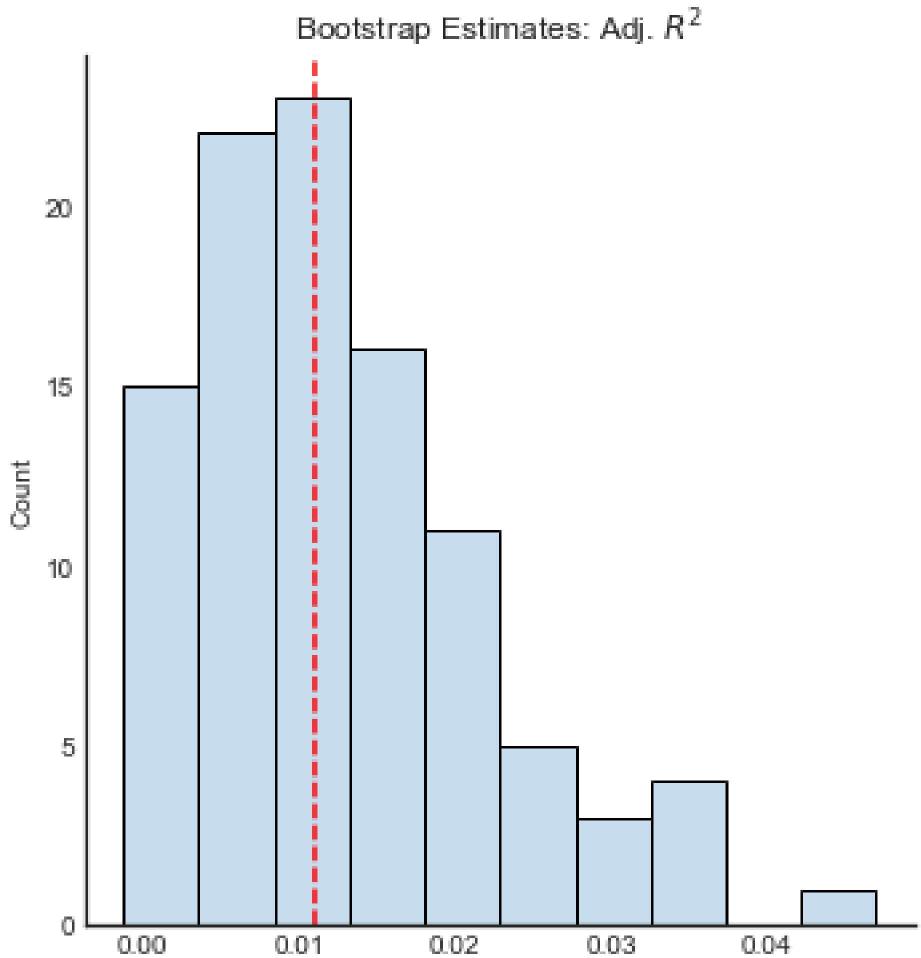
[89]:
```
sns.displot(boot_interc, alpha = 0.25)
plt.axvline(x=5.3733,color='red', linestyle='--')
plt.title('Bootstrap Estimates: $y-$intercept')
plt.show()
```



3.0.8 Bootstrap Estimates (Adj. R2)

[93]:
```
sns.displot(boot_adjR2, alpha = 0.25)
plt.axvline(x=0.011,color='red', linestyle='--')
plt.title('Bootstrap Estimates: Adj. $R^2$')
plt.show()
```

```
WARNING:root:SKIPPED triang distribution (taking more than 30 seconds)
```



3.0.9 Evaluating OLS Model Performance with Cross-Validation

[91]:
```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score

x = df[['exper']]
y = df[['wage']]
# Perform an OLS fit using all the data
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_
```

```
# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
↪random_state=0)

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance

print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

# Perform a 5-fold CV
# Use MSE as the scoring function (there are other options as shown here:
# https://scikit-learn.org/stable/modules/model_evaluation.html

regr = linear_model.LinearRegression()
scores = cross_val_score(regr, x, y, cv=5,␣
↪scoring='neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores)
```

```
MAE: 2.669962398476693
MSE: 12.05923472699901
RMSE: 3.4726408865586738
5-Fold CV MSE Scores: [-4.3982781  -3.8027313  -3.5078229  -2.95388475
-3.7262548 ]
```

3.0.10 Cullen-Frey Graph

[ ]:
```
# Estimates what distribution best fits your data
# Generate some data:
from scipy import stats
data = stats.gamma.rvs(2, loc=1.5, scale=2, size=1000)

# Fit various distributions:
from fitter import Fitter
f = Fitter(data)
f.fit()

# may take some time since by default, all distributions are tried
# but you call manually provide a smaller set of distributions
f.summary()
```

```
WARNING:root:SKIPPED pearson3 distribution (taking more than 30 seconds)
WARNING:root:SKIPPED powerlognorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED powernorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rdist distribution (taking more than 30 seconds)
WARNING:root:SKIPPED recipinvgauss distribution (taking more than 30 seconds)
```

4 Multiple Regression

4.0.1 Fitting a MR Model

[63]:
```python
import statsmodels.api as sm
import statsmodels as sms
import seaborn as sns
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import wooldridge as woo

df = woo.dataWoo('gpa1')

# Specify the Model
mr_mod = smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol', data=df)

# Fit the Model
mr_fit = mr_mod.fit()
# Type: dir(ols_fit) to look at other accessible attributes

# Look at the Model Fit Summary
print(mr_fit.summary())
```

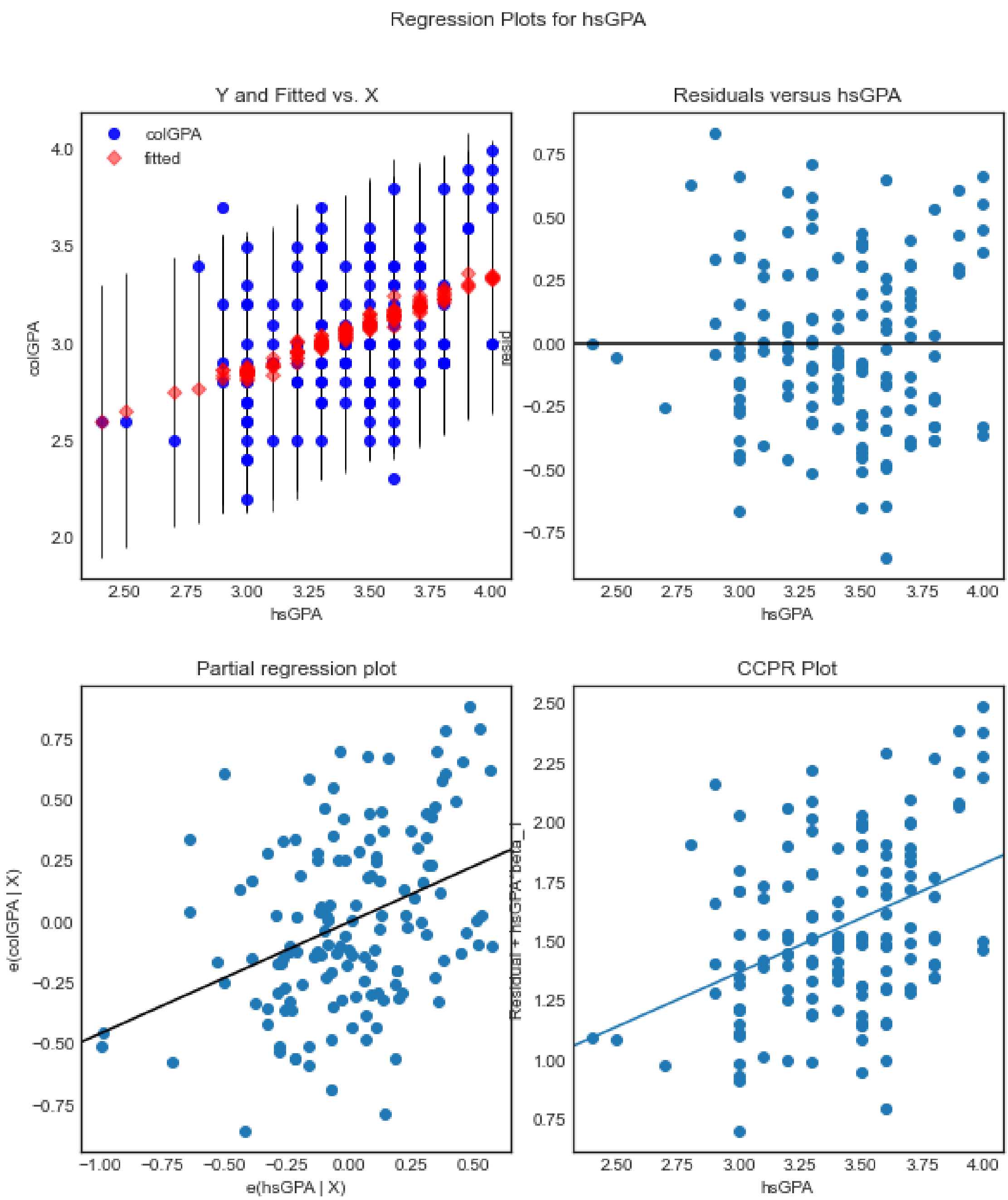
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 colGPA   R-squared:                       0.177
Model:                            OLS   Adj. R-squared:                  0.159
Method:                 Least Squares   F-statistic:                     9.819
Date:                Mon, 06 Dec 2021   Prob (F-statistic):           6.56e-06
Time:                        18:18:42   Log-Likelihood:                -46.526
No. Observations:                 141   AIC:                             101.1
Df Residuals:                     137   BIC:                             112.8
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.2787      0.343      3.729      0.000       0.601       1.957
hsGPA          0.4567      0.097      4.721      0.000       0.265       0.648
ACT            0.0088      0.011      0.795      0.428      -0.013       0.031
```

```
alcohol          0.0065       0.021       0.301       0.764      -0.036       0.049
==============================================================================
Omnibus:                        3.314   Durbin-Watson:                   1.876
Prob(Omnibus):                  0.191   Jarque-Bera (JB):                2.582
Skew:                           0.198   Prob(JB):                        0.275
Kurtosis:                       2.468   Cond. No.                         300.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

4.0.2 Diagnostic Plots

[64]:
```python
fig = sm.graphics.plot_regress_exog(mr_fit, "hsGPA")
fig.set_figheight(10)
fig.set_figwidth(8)
plt.show()
```

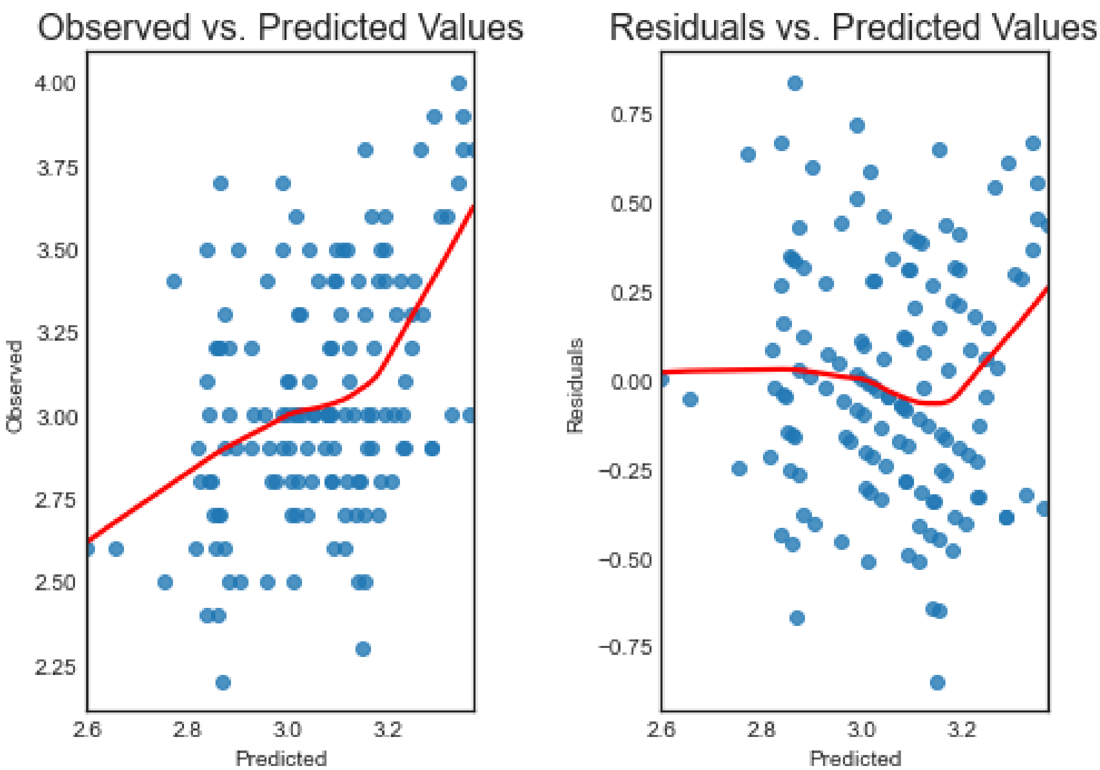
```
eval_env: 1
```



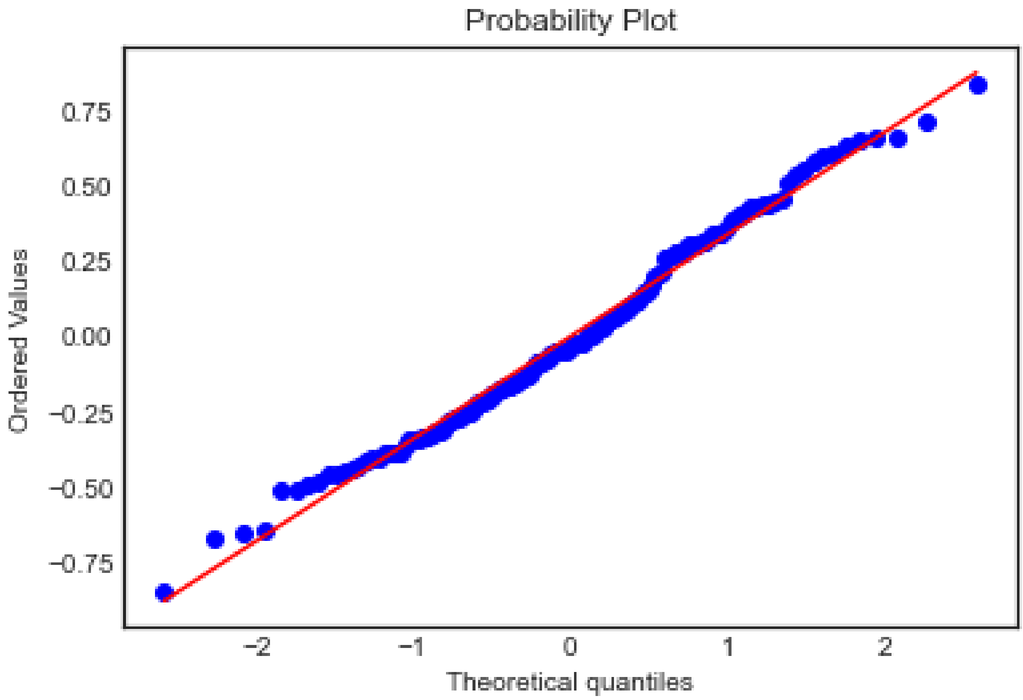
[65]:
```
# Plot y vs y_hat
fig, ax = plt.subplots(1,2,figsize=(8, 6))
fig.tight_layout(pad=6.0)
sns.regplot(x=mr_fit.fittedvalues, y=df['colGPA'], lowess=True, ax=ax[0],␣
↪line_kws={'color': 'red'})
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')
```

```
sns.regplot(x=mr_fit.fittedvalues, y=mr_fit.resid, lowess=True, ax=ax[1],␣
↪line_kws={'color': 'red'})
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

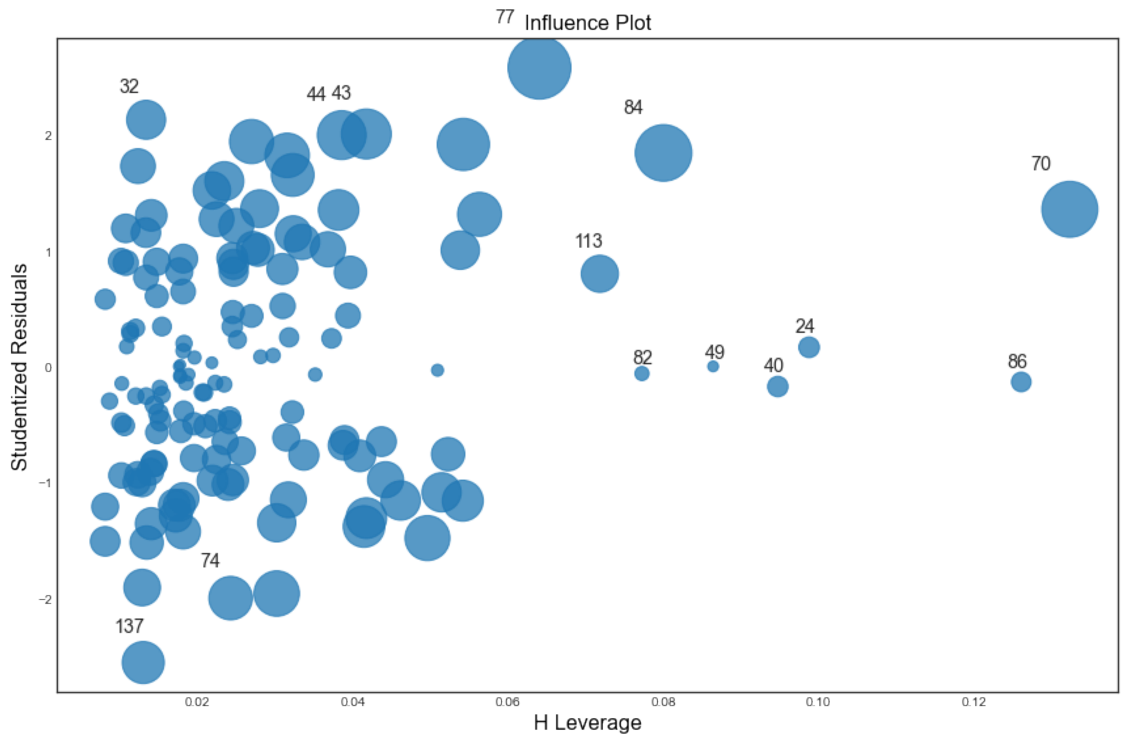
[65]: [Text(0.5, 37.5, 'Predicted'), Text(293.99090909090904, 0.5, 'Residuals')]

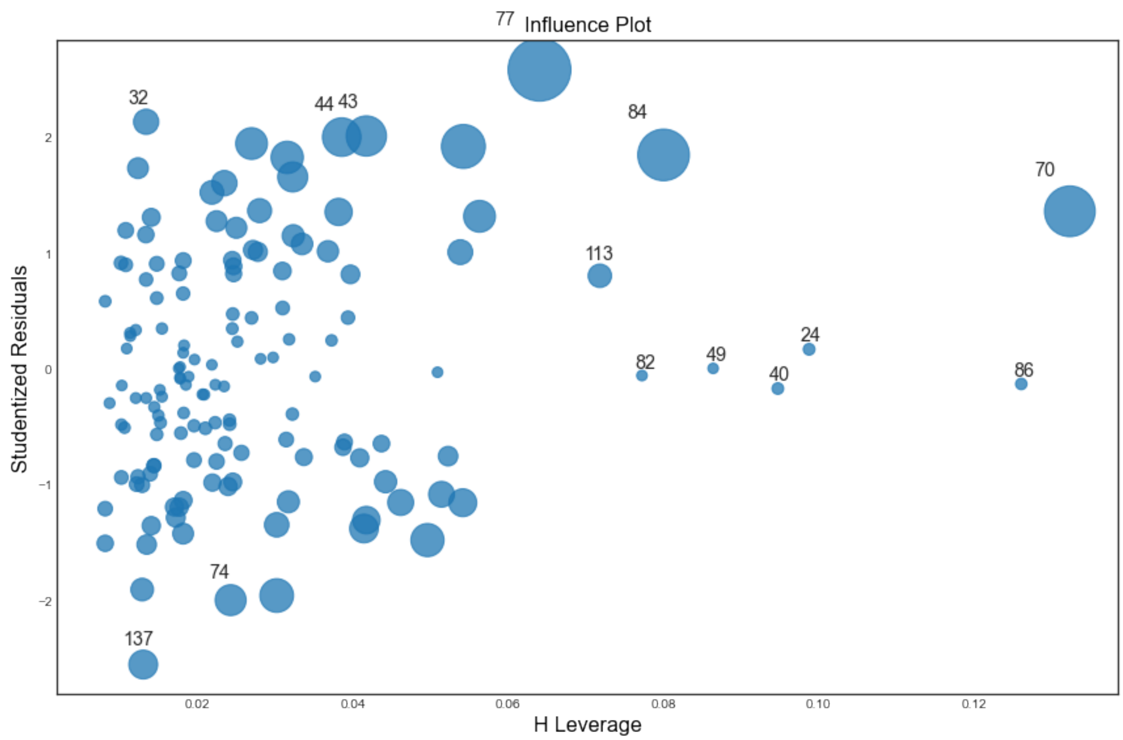


[66]:
```
# QQ Plot (Normal Probability)
import scipy as sp
figA, axA = plt.subplots(figsize=(6,4))
_, (__, ___, r) = sp.stats.probplot(mr_fit.resid, plot = axA, fit=True)
```



[67]:
```
# Outliers, high leverage, influential obs
figd, ax = plt.subplots(figsize=(12,8))
figd = sm.graphics.influence_plot(mr_fit, ax = ax, criterion="DFFITS")
figd.tight_layout(pad=1.0)

fige, ax = plt.subplots(figsize=(12,8))
fige = sm.graphics.influence_plot(mr_fit, ax = ax, criterion="cooks")
fige.tight_layout(pad=1.0)
```


Influence Plot
Studentized Residuals
H Leverage
77
32
44 43
84
70
113
24
82
49
40
86
74
137
2
1
0
−1
−2
0.02
0.04
0.06
0.08
0.10
0.12


Influence Plot
Studentized Residuals
H Leverage
77
32
44 43
84
70
113
24
82
49
40
86
74
137
2
1
0
−1
−2
0.02
0.04
0.06
0.08
0.10
0.12

4.0.3 Variance Inflation Factor (VIF)

[68]:
```python
# VIF: Test for multicolinearity
import statsmodels.stats.outliers_influence as smo
import patsy as pt

# extract matrices using patsy:
y, X = pt.dmatrices('colGPA ~ hsGPA + ACT + alcohol',
                    data=df, return_type='dataframe')

# get VIF:
K = X.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X.values, i)
print(f'VIF: \n{VIF}\n')
# VIF values are low enough that multicolinearity does not seem to be an issue
```

```
VIF:
[142.19442956   1.15044419   1.18179548   1.04266223]
```

4.0.4 Lagrange Multiplier (LM), or Breusch-Pagan (BP), Test

[72]:
```python
# Heteroskedasticity: Breush-Pagan --> Ho: var = constant
import statsmodels.stats.api as sms
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(mr_fit.resid, mr_fit.model.exog)
print("BP Results:")
print(list(zip(name, test)))
# Fail to reject Ho, therefore, heteroskedasticity does not appear to be an␣
↪issue.
```

```
BP Results:
[('Lagrange multiplier statistic', 1.112409529706819), ('p-value',
0.7740792499491564), ('f-value', 0.3631489756822489), ('f p-value',
0.7797099382125043)]
```

4.0.5 Model Misspecification: Ramsey RESET

[75]:
```python
# Model Misspecification
import statsmodels.regression.linear_model as rg
import statsmodels.stats.diagnostic as dg

test =  dg.linear_reset(mr_fit, power=2,  test_type='fitted', use_f = True)

print("Ramsey-RESET:")
print(test)
```

```
# Fail to reject Ho, therefore, the model seems to be correctly specified at␣
↪order 2.

# Note: We can do an added sanity check by estimating two more models:

#(a) include a quadratic term
print("Quadratic Model")
smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol + I(hsGPA**2)', data = df).
↪fit().summary()

print('')

#(b) include an interation term
print("Interaction Model")
smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol + hsGPA*alcohol', data = df).
↪fit().summary()
```

```
Ramsey-RESET:
<F test: F=3.233940008618943, p=0.07434504298656607, df_denom=136, df_num=1>
Quadratic Model

Interaction Model
```

[75]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                 colGPA   R-squared:                       0.180
Model:                            OLS   Adj. R-squared:                  0.156
Method:                 Least Squares   F-statistic:                     7.450
Date:                Mon, 06 Dec 2021   Prob (F-statistic):           1.86e-05
Time:                        18:23:13   Log-Likelihood:                -46.289
No. Observations:                 141   AIC:                             102.6
Df Residuals:                     136   BIC:                             117.3
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept         1.5631      0.543      2.879      0.005       0.489
2.637
hsGPA             0.3785      0.151      2.507      0.013       0.080
0.677
ACT               0.0079      0.011      0.713      0.477      -0.014
```

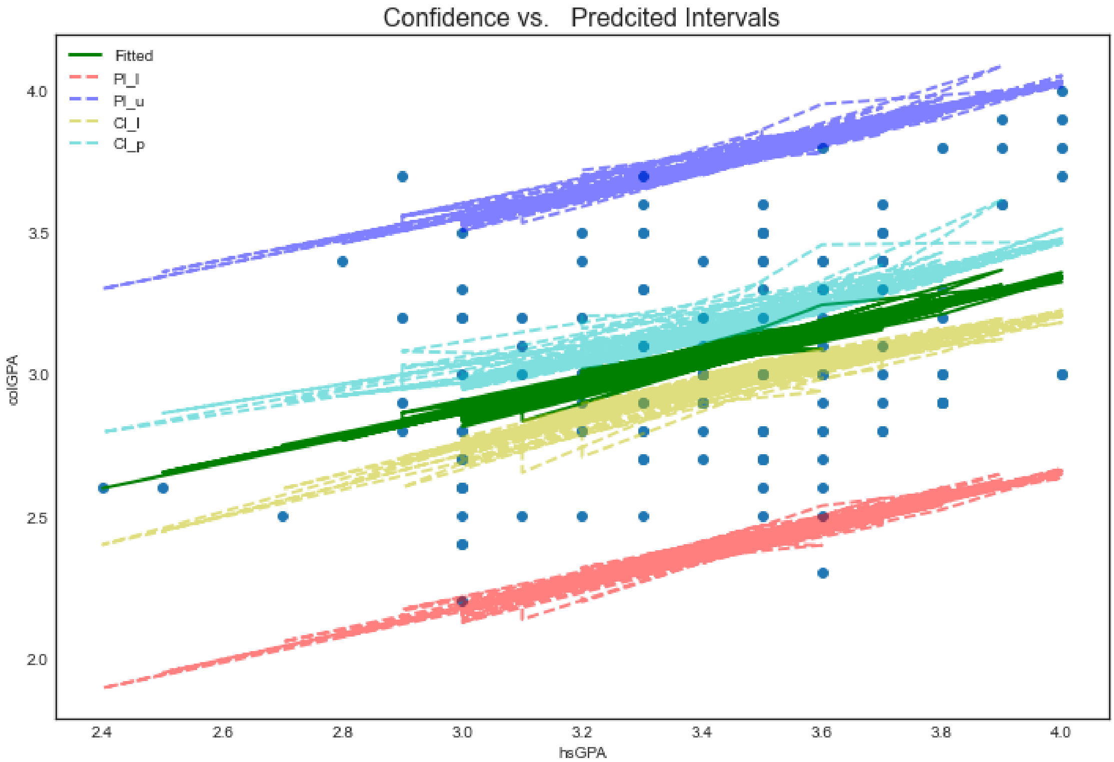
```
0.030
alcohol          -0.1275      0.199     -0.640      0.523      -0.521
0.266
hsGPA:alcohol     0.0398      0.059      0.676      0.500      -0.077
0.156
==============================================================================
Omnibus:                        2.550   Durbin-Watson:                   1.867
Prob(Omnibus):                  0.279   Jarque-Bera (JB):                2.348
Skew:                           0.231   Prob(JB):                        0.309
Kurtosis:                       2.568   Cond. No.                         517.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

4.0.6 Prediction vs. Confidence Intervals

[76]:
```python
from statsmodels.stats.outliers_influence import summary_table
from statsmodels.sandbox.regression.predstd import wls_prediction_std
import numpy as np
st, data, ss2 = summary_table(mr_fit, alpha=0.05)
prstd, iv_l, iv_u = wls_prediction_std(mr_fit)
fittedvalues = data[:, 2]
predict_mean_se  = data[:, 3]
predict_mean_ci_low, predict_mean_ci_upp = data[:, 4:6].T
predict_ci_low, predict_ci_upp = data[:, 6:8].T

# Check we got the right things
print(np.max(np.abs(mr_fit.fittedvalues - fittedvalues)))
print(np.max(np.abs(iv_l - predict_ci_low)))
print(np.max(np.abs(iv_u - predict_ci_upp)))
x = df['hsGPA']
y =  df['colGPA']
plt.plot(x, y, 'o')
plt.plot(x, fittedvalues, 'g-', lw=2, label = 'Fitted')
plt.plot(x, predict_ci_low, 'r--', lw=2,label ='PI_l', alpha = 0.5)
plt.plot(x, predict_ci_upp, 'b--', lw=2,label = 'PI_u', alpha = 0.5)
plt.plot(x, predict_mean_ci_low, 'y--', lw=2, label ='CI_l', alpha = 0.5)
plt.plot(x, predict_mean_ci_upp, 'c--', lw=2, label ='CI_p', alpha = 0.5)
plt.title('Confidence vs.   Predcited Intervals', fontsize=16)
plt.xlabel('hsGPA')
plt.ylabel('colGPA')
plt.legend()
plt.show()
```

```
0.0
```

```
0.0
0.0
```



4.0.7 Robust Estimation

[77]:
```python
# Compare the traditional SE vs. the  White SE
mr_mod = smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol', data=df)

# estimate default model (only for spring data):
results_default = mr_mod.fit()

table_default = pd.DataFrame({'b': round(results_default.params, 5),
                              'se': round(results_default.bse, 5),
                              't': round(results_default.tvalues, 5),
                              'pval': round(results_default.pvalues, 5)})
print(f'Default Estimaets & Std. Errors: \n{table_default}\n')

# estimate model with White SE (only for spring data):
results_white = mr_mod.fit(cov_type='HC1')

table_white = pd.DataFrame({'b': round(results_white.params, 5),
                            'se': round(results_white.bse, 5),
                            't': round(results_white.tvalues, 5),
```

```
                             'pval': round(results_white.pvalues, 5)})
print(f'White Estimaets & Std. Errors: \n{table_white}\n')
```

```
Default Estimaets & Std. Errors:
                 b       se        t     pval
Intercept  1.27871  0.34289  3.72925  0.00028
hsGPA      0.45674  0.09675  4.72099  0.00001
ACT        0.00877  0.01103  0.79521  0.42787
alcohol    0.00646  0.02143  0.30120  0.76372

White Estimaets & Std. Errors:
                 b       se        t     pval
Intercept  1.27871  0.35813  3.57052  0.00036
hsGPA      0.45674  0.09857  4.63372  0.00000
ACT        0.00877  0.01096  0.79992  0.42376
alcohol    0.00646  0.02303  0.28028  0.77926
```

4.0.8 Weighted Least-Squares (Known Form of the Variance)

[78]:
```
# Estimate model:
mr_mod = smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol', data=df)

results_ols= mr_mod.fit(cov_type='HC0')

table_ols = pd.DataFrame({'b': round(results_ols.params, 4),
                          'se': round(results_ols.bse, 4),
                          't': round(results_ols.tvalues, 4),
                          'pval': round(results_ols.pvalues, 4)})
print(f'table_ols: \n{table_ols}\n')

# WLS: here we use w = 1/x
wls_weight = list(1 / df['hsGPA'])
reg_wls = smf.wls(formula='colGPA ~ hsGPA + ACT + alcohol',
                  weights=wls_weight, data=df)
results_wls = reg_wls.fit()

# print regression table:
table_wls = pd.DataFrame({'b': round(results_wls.params, 4),
                          'se': round(results_wls.bse, 4),
                          't': round(results_wls.tvalues, 4),
                          'pval': round(results_wls.pvalues, 4)})
print(f'table_wls: \n{table_wls}\n')
```

```
table_ols:
                b      se       t    pval
Intercept  1.2787  0.3530  3.6223  0.0003
hsGPA      0.4567  0.0972  4.7009  0.0000
```

```
ACT         0.0088  0.0108  0.8115  0.4171
alcohol     0.0065  0.0227  0.2843  0.7761

table_wls:
                b      se       t    pval
Intercept  1.3358  0.3368  3.9660  0.0001
hsGPA      0.4423  0.0940  4.7080  0.0000
ACT        0.0086  0.0109  0.7871  0.4326
alcohol    0.0048  0.0212  0.2255  0.8219
```

4.0.9 Feasible Generalized Least-Squares (Unknown Form of the Variance)

[79]:

```python
# estimate model:
mr_mod = smf.ols(formula='colGPA ~ hsGPA + ACT + alcohol', data=df)

results_ols= mr_mod.fit()

# FGLS (estimation of the variance function):
df['logu2'] = np.log(results_ols.resid ** 2)
reg_fgls = smf.ols(formula='logu2 ~ hsGPA + ACT + alcohol', data=df)
results_fgls = reg_fgls.fit()
table_fgls = pd.DataFrame({'b': round(results_fgls.params, 4),
                           'se': round(results_fgls.bse, 4),
                           't': round(results_fgls.tvalues, 4),
                           'pval': round(results_fgls.pvalues, 4)})
print(f'FGLS (Variance Function): \n{table_fgls}\n')

# FGLS (WLS):
wls_weight = list(1 / np.exp(results_fgls.fittedvalues))
reg_wls = smf.wls(formula='colGPA ~ hsGPA + ACT + alcohol',
                  weights=wls_weight, data=df)
results_wls = reg_wls.fit()
table_wls = pd.DataFrame({'b': round(results_wls.params, 4),
                          'se': round(results_wls.bse, 4),
                          't': round(results_wls.tvalues, 4),
                          'pval': round(results_wls.pvalues, 4)})
print(f'FGLS Estimates: \n{table_wls}\n')
```

```
FGLS (Variance Function):
                b      se       t    pval
Intercept -9.5604  2.2352 -4.2771  0.0000
hsGPA      1.9063  0.6307  3.0226  0.0030
ACT       -0.0151  0.0719 -0.2102  0.8338
alcohol    0.0268  0.1397  0.1918  0.8482

FGLS Estimates:
                b      se       t    pval
```

```
Intercept  1.4892  0.3266  4.5597  0.0000
hsGPA      0.4033  0.0818  4.9298  0.0000
ACT        0.0084  0.0104  0.8078  0.4206
alcohol   -0.0053  0.0200 -0.2659  0.7907
```

4.0.10 Mallows' Cp

[ ]:
```python
def MallowCp(result, base):

    #Compute Cp
    return(sum(result.resid**2)/base.mse_resid - result.nobs + 2*result.params.
 ↪size)

def Mallow_Brute(Y, Z, Base):

    print("This function might take an extended period of time to run.")

    # Create a DataFrame to store the results
    results = pd.DataFrame(["IVs", "P", "CP"])
    count = 0

    for i in range(2, Z.shape[1]+1):

        # Compute every possible subset given p
        combo = list(itertools.combinations(Z.columns, i))

        for j in (combo):
            # fit model and compute CP
            mod = smf.ols(formula = 'Y ~ Z.loc[:,j]', data = EFI_Boruta).fit()
            cp = MallowCp(mod, Base)

            # Store values
            results = results.append({"IVs": j, "p": i, "CP": cp}, ignore_index␣
 ↪= True)

            # This can take a very long time to run
            # I can keep track of my function be adding a count
            count+= 1
            print(count)

    # Find distance from p and CP for each value
    results["distance"] = np.abs(results.p-results.CP)
    return(results)

# Finding model with combination of least paramaters and distance
results.sort_values(["distance", "CP"], axis = 0).head(15)
```

4.0.11 Boruta Shap

[ ]:
```
from BorutaShap import BorutaShap

# Identifying important attributes/predictors using BorutaShap
Feature_Selector = BorutaShap(importance_measure = 'shap', classification =␣
↪False)
Feature_Selector.fit(Regressors, Regressand, n_trials = 100, random_state = 0)
Feature_Selector.plot(which_features = 'all')
```

5 Qualitative-Limited Dependent Variable Models

5.0.1 Linear Probability Model

We could model the indicator variable y using the linear model, however, there are several problems:

– It implies marginal effects of changes in continuous explanatory variables are constant, which cannot be the case for a probability model

– This feature also can result in predicted probabilities outside the [0, 1] interval

– The linear probability model error term is heteroskedastic, so that a better estimator is generalized least squares

The underlying feature that causes these problems is that the linear probability model implicitly assumes that increases in x have a constant effect on $P(y = 1)$:

$dp/dx = _2$

[53]:
```
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# y = 1 (woman in the labor force), = 0 (otherwise)
# Estimate a linear probability model:
reg_lin = smf.ols(formula='inlf ~ nwifeinc + educ + exper +'
                          'I(exper**2) + age + kidslt6 + kidsge6',
                  data=mroz)
results_lin = reg_lin.fit(cov_type='HC3')

# Print regression table:
table = pd.DataFrame({'b': round(results_lin.params, 4),
                      'se': round(results_lin.bse, 4),
                      't': round(results_lin.tvalues, 4),
                      'pval': round(results_lin.pvalues, 4)})
print(f'table: \n{table}\n')

# We can check the y_hat values for two "extreme" cases:
```

```
# Recall that for this model y_hat may be outside [0, 1],
#  which inn theory can't happen

X_new = pd.DataFrame(
    {'nwifeinc': [100, 0], 'educ': [5, 17],
     'exper': [0, 30], 'age': [20, 52],
     'kidslt6': [2, 0], 'kidsge6': [0, 0]})
predictions = results_lin.predict(X_new)

print(f'predictions: \n{predictions}\n')
```

```
table:
                    b      se       t    pval
Intercept      0.5855  0.1536  3.8125  0.0001
nwifeinc      -0.0034  0.0016 -2.1852  0.0289
educ           0.0380  0.0073  5.1766  0.0000
exper          0.0395  0.0060  6.6001  0.0000
I(exper ** 2) -0.0006  0.0002 -2.9973  0.0027
age           -0.0161  0.0024 -6.6640  0.0000
kidslt6       -0.2618  0.0322 -8.1430  0.0000
kidsge6        0.0130  0.0137  0.9526  0.3408

predictions:
0   -0.410458
1    1.042808
dtype: float64
```

5.0.2 Probit Model

The probit statistical model expresses the probability $P(y = 1)$; where $y = (0, 1)$ for $p^{HAT} < 0.5$ and $p^{HAT} >= 0.5$, respectively.

[54]:
```
import wooldridge as woo
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# Estimate a probit model:
reg_probit = smf.probit(formula='inlf ~ nwifeinc + educ + exper +'
                                'I(exper**2) + age + kidslt6 + kidsge6',
                        data=mroz)
results_probit = reg_probit.fit(disp=0)
print(f'results_probit.summary(): \n{results_probit.summary()}\n')

# log likelihood value:
print(f'results_probit.llf: {results_probit.llf}\n')
```

```python
# McFadden's pseudo R2:
print(f'results_probit.prsquared: {results_probit.prsquared}\n')
```

```
results_probit.summary():
                          Probit Regression Results
==============================================================================
Dep. Variable:                   inlf   No. Observations:                  753
Model:                         Probit   Df Residuals:                      745
Method:                           MLE   Df Model:                            7
Date:                Mon, 06 Dec 2021   Pseudo R-squ.:                  0.2206
Time:                        18:12:02   Log-Likelihood:                -401.30
converged:                       True   LL-Null:                       -514.87
Covariance Type:            nonrobust   LLR p-value:                 2.009e-45
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept         0.2701      0.509      0.531      0.595      -0.727
1.267
nwifeinc         -0.0120      0.005     -2.484      0.013      -0.022
-0.003
educ              0.1309      0.025      5.183      0.000       0.081
0.180
exper             0.1233      0.019      6.590      0.000       0.087
0.160
I(exper ** 2)    -0.0019      0.001     -3.145      0.002      -0.003
-0.001
age              -0.0529      0.008     -6.235      0.000      -0.069
-0.036
kidslt6          -0.8683      0.119     -7.326      0.000      -1.101
-0.636
kidsge6           0.0360      0.043      0.828      0.408      -0.049
0.121
================================================================================
=

results_probit.llf: -401.3021931738952

results_probit.prsquared: 0.22058054372529368
```

5.0.3 Logit Model

Differs from probit model in the particular S-shaped curve used to constrain probabilities to the [0, 1] interval. Still expresses the probability $P(y = 1)$; where $y = (0, 1)$ for $p^{HAT} < 0.5$ and $p^{HAT} >= 0.5$, respectively. Estimation more inclined to be cleaner, but still likely to be close to probit.

[56]:
```python
import wooldridge as woo
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# Estimate a logit model:
reg_logit = smf.logit(formula='inlf ~ nwifeinc + educ + exper +'
                              'I(exper**2) + age + kidslt6 + kidsge6',
                      data=mroz)

# disp = 0 avoids printing out information during the estimation:
results_logit = reg_logit.fit(disp=0)
print(f'results_logit.summary(): \n{results_logit.summary()}\n')

# log likelihood value:
print(f'results_logit.llf: {results_logit.llf}\n')

# McFadden's pseudo R2:
print(f'results_logit.prsquared: {results_logit.prsquared}\n')
```

```
results_logit.summary():
                           Logit Regression Results
==============================================================================
Dep. Variable:                   inlf   No. Observations:                  753
Model:                          Logit   Df Residuals:                      745
Method:                           MLE   Df Model:                            7
Date:                Mon, 06 Dec 2021   Pseudo R-squ.:                  0.2197
Time:                        18:16:37   Log-Likelihood:                -401.77
converged:                       True   LL-Null:                       -514.87
Covariance Type:            nonrobust   LLR p-value:                 3.159e-45
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept         0.4255      0.860      0.494      0.621      -1.261
2.112
nwifeinc         -0.0213      0.008     -2.535      0.011      -0.038
-0.005
educ              0.2212      0.043      5.091      0.000       0.136
0.306
exper             0.2059      0.032      6.422      0.000       0.143
0.269
I(exper ** 2)    -0.0032      0.001     -3.104      0.002      -0.005
-0.001
```

```
age                 -0.0880      0.015     -6.040      0.000      -0.117
-0.059
kidslt6             -1.4434      0.204     -7.090      0.000      -1.842
-1.044
kidsge6              0.0601      0.075      0.804      0.422      -0.086
0.207
================================================================================
=

results_logit.llf: -401.76515113438177

results_logit.prsquared: 0.21968137484058803
```

5.0.4 Long-Form and Short-Form Inferences

[57]:

```
import wooldridge as woo
import statsmodels.formula.api as smf
import scipy.stats as stats

mroz = woo.dataWoo('mroz')

# estimate probit model:
reg_probit = smf.probit(formula='inlf ~ nwifeinc + educ + exper +'
                                'I(exper**2) + age + kidslt6 + kidsge6',
                        data=mroz)
results_probit = reg_probit.fit(disp=0)

# test of overall significance (test statistic and pvalue):
llr1_manual = 2 * (results_probit.llf - results_probit.llnull)
print(f'llr1_manual: {llr1_manual}\n')
print(f'results_probit.llr: {results_probit.llr}\n')
print(f'results_probit.llr_pvalue: {results_probit.llr_pvalue}\n')

# automatic Wald test of H0 (experience and age are irrelevant):
hypotheses = ['exper=0', 'I(exper ** 2)=0', 'age=0']
waldstat = results_probit.wald_test(hypotheses)
teststat2_autom = waldstat.statistic
pval2_autom = waldstat.pvalue
print(f'teststat2_autom: {teststat2_autom}\n')
print(f'pval2_autom: {pval2_autom}\n')

# manual likelihood ratio statistic test
# of H0 (experience and age are irrelevant):
reg_probit_restr = smf.probit(formula='inlf ~ nwifeinc + educ +'
                                      'kidslt6 + kidsge6',
                              data=mroz)
```

```
results_probit_restr = reg_probit_restr.fit(disp=0)

llr2_manual = 2 * (results_probit.llf - results_probit_restr.llf)
pval2_manual = 1 - stats.chi2.cdf(llr2_manual, 3)
print(f'llr2_manual2: {llr2_manual}\n')
print(f'pval2_manual2: {pval2_manual}\n')
```

```
llr1_manual: 227.14202283719226

results_probit.llr: 227.14202283719226

results_probit.llr_pvalue: 2.0086732957628273e-45

teststat2_autom: [[110.91852003]]

pval2_autom: 6.960738406715968e-24

llr2_manual2: 127.03401014418034

pval2_manual2: 0.0


C:\Users\tanne\anaconda3\lib\site-packages\statsmodels\base\model.py:1889:
FutureWarning: The behavior of wald_test will change after 0.14 to returning
scalar test statistic values. To get the future behavior now, set scalar to
True. To silence this message while retaining the legacy behavior, set scalar to
False.
  warnings.warn(
```

5.0.5 Long-Form and Short-Form Inferences

[58]:
```
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# Estimate the models:

# Linear
reg_lin = smf.ols(formula='inlf ~ nwifeinc + educ + exper +'
                          'I(exper**2) + age + kidslt6 + kidsge6',
                  data=mroz)
results_lin = reg_lin.fit(cov_type='HC3')

# Logit
reg_logit = smf.logit(formula='inlf ~ nwifeinc + educ + exper +'
                              'I(exper**2) + age + kidslt6 + kidsge6',
```

```python
                      data=mroz)
results_logit = reg_logit.fit(disp=0)

# Probit
reg_probit = smf.probit(formula='inlf ~ nwifeinc + educ + exper +'
                                'I(exper**2) + age + kidslt6 + kidsge6',
                        data=mroz)
results_probit = reg_probit.fit(disp=0)

# predictions for two "extreme" women:
X_new = pd.DataFrame(
    {'nwifeinc': [100, 0], 'educ': [5, 17],
     'exper': [0, 30], 'age': [20, 52],
     'kidslt6': [2, 0], 'kidsge6': [0, 0]})
predictions_lin = results_lin.predict(X_new)
predictions_logit = results_logit.predict(X_new)
predictions_probit = results_probit.predict(X_new)

print(f'predictions_lin: \n{predictions_lin}\n')
print(f'predictions_logit: \n{predictions_logit}\n')
print(f'predictions_probit: \n{predictions_probit}\n')
```

```
predictions_lin:
0   -0.410458
1    1.042808
dtype: float64

predictions_logit:
0    0.005218
1    0.950049
dtype: float64

predictions_probit:
0    0.001065
1    0.959870
dtype: float64
```

5.0.6 Prediction Comparisons: Linear Model vs. Logit vs. Probit (Estimates)

Cannot compare across predictors, as the functions are not the same (linear vs. non-linear).

[59]:
```python
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')
```

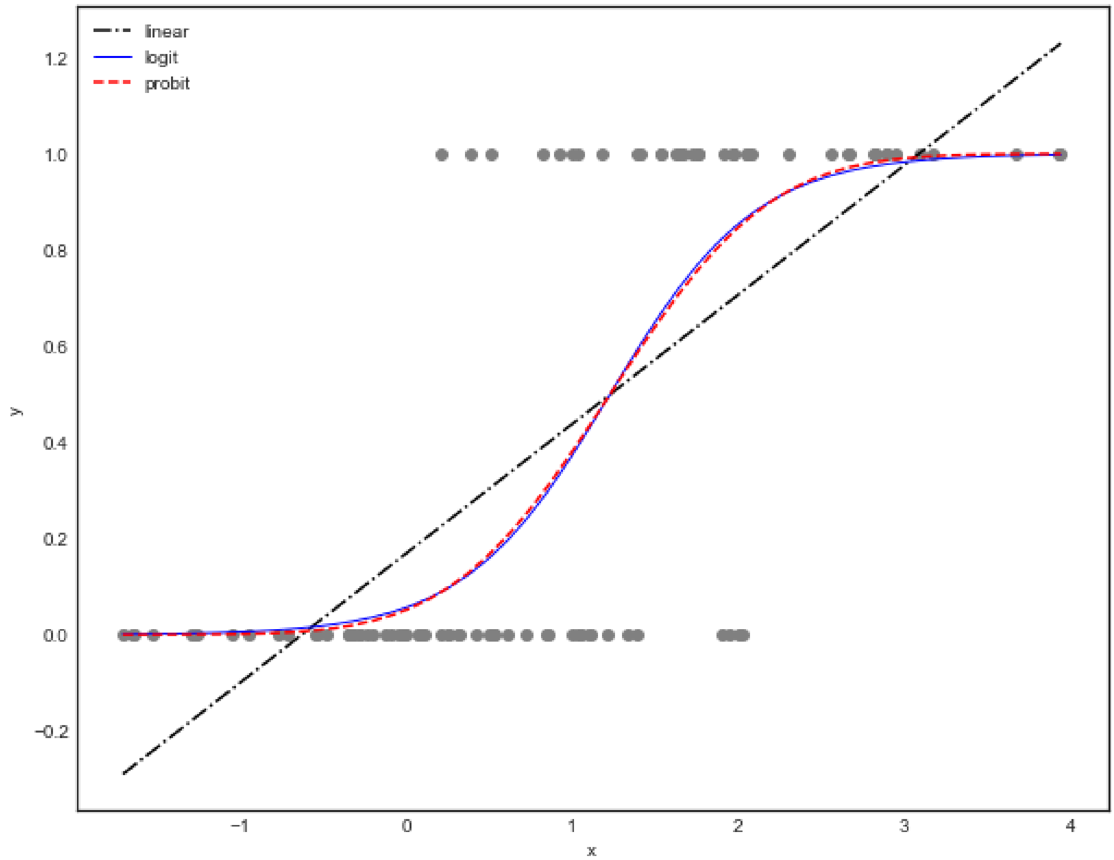
```python
# Estimate the models:

# Linear
reg_lin = smf.ols(formula='inlf ~ nwifeinc + educ + exper +'
                          'I(exper**2) + age + kidslt6 + kidsge6',
                  data=mroz)
results_lin = reg_lin.fit(cov_type='HC3')

# Logit
reg_logit = smf.logit(formula='inlf ~ nwifeinc + educ + exper +'
                              'I(exper**2) + age + kidslt6 + kidsge6',
                      data=mroz)
results_logit = reg_logit.fit(disp=0)

# Probit
reg_probit = smf.probit(formula='inlf ~ nwifeinc + educ + exper +'
                                'I(exper**2) + age + kidslt6 + kidsge6',
                        data=mroz)
results_probit = reg_probit.fit(disp=0)

# predictions for two "extreme" women:
X_new = pd.DataFrame(
    {'nwifeinc': [100, 0], 'educ': [5, 17],
     'exper': [0, 30], 'age': [20, 52],
     'kidslt6': [2, 0], 'kidsge6': [0, 0]})
predictions_lin = results_lin.predict(X_new)
predictions_logit = results_logit.predict(X_new)
predictions_probit = results_probit.predict(X_new)

print(f'predictions_lin: \n{predictions_lin}\n')
print(f'predictions_logit: \n{predictions_logit}\n')
print(f'predictions_probit: \n{predictions_probit}\n')
```

```
predictions_lin:
0   -0.410458
1    1.042808
dtype: float64

predictions_logit:
0    0.005218
1    0.950049
dtype: float64

predictions_probit:
0    0.001065
1    0.959870
dtype: float64
```

5.0.7 Prediction Comparisons: Linear Model vs. Logit vs. Probit (Predictions)

[60]:
```python
import pandas as pd
import numpy as np
import scipy.stats as stats
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt

# set the random seed:
np.random.seed(1234567)

y = stats.binom.rvs(1, 0.5, size=100)
x = stats.norm.rvs(0, 1, size=100) + 2 * y
sim_data = pd.DataFrame({'y': y, 'x': x})

# estimation:
reg_lin = smf.ols(formula='y ~ x', data=sim_data)
results_lin = reg_lin.fit()
reg_logit = smf.logit(formula='y ~ x', data=sim_data)
results_logit = reg_logit.fit(disp=0)
reg_probit = smf.probit(formula='y ~ x', data=sim_data)
results_probit = reg_probit.fit(disp=0)

# prediction for regular grid of x values:
X_new = pd.DataFrame({'x': np.linspace(min(x), max(x), 50)})
predictions_lin = results_lin.predict(X_new)
predictions_logit = results_logit.predict(X_new)
predictions_probit = results_probit.predict(X_new)

# scatter plot and fitted values:
fig, ax = plt.subplots(figsize=(10, 8))
plt.plot(x, y, color='grey', marker='o', linestyle='')
plt.plot(X_new['x'], predictions_lin,
         color='black', linestyle='-.', label='linear')
plt.plot(X_new['x'], predictions_logit,
         color='blue', linestyle='-', linewidth=1, label='logit')
plt.plot(X_new['x'], predictions_probit,
         color='red', linestyle='--', label='probit')
plt.ylabel('y')
plt.xlabel('x')
plt.legend()
```

[60]:
```
<matplotlib.legend.Legend at 0x22dd8220160>
```

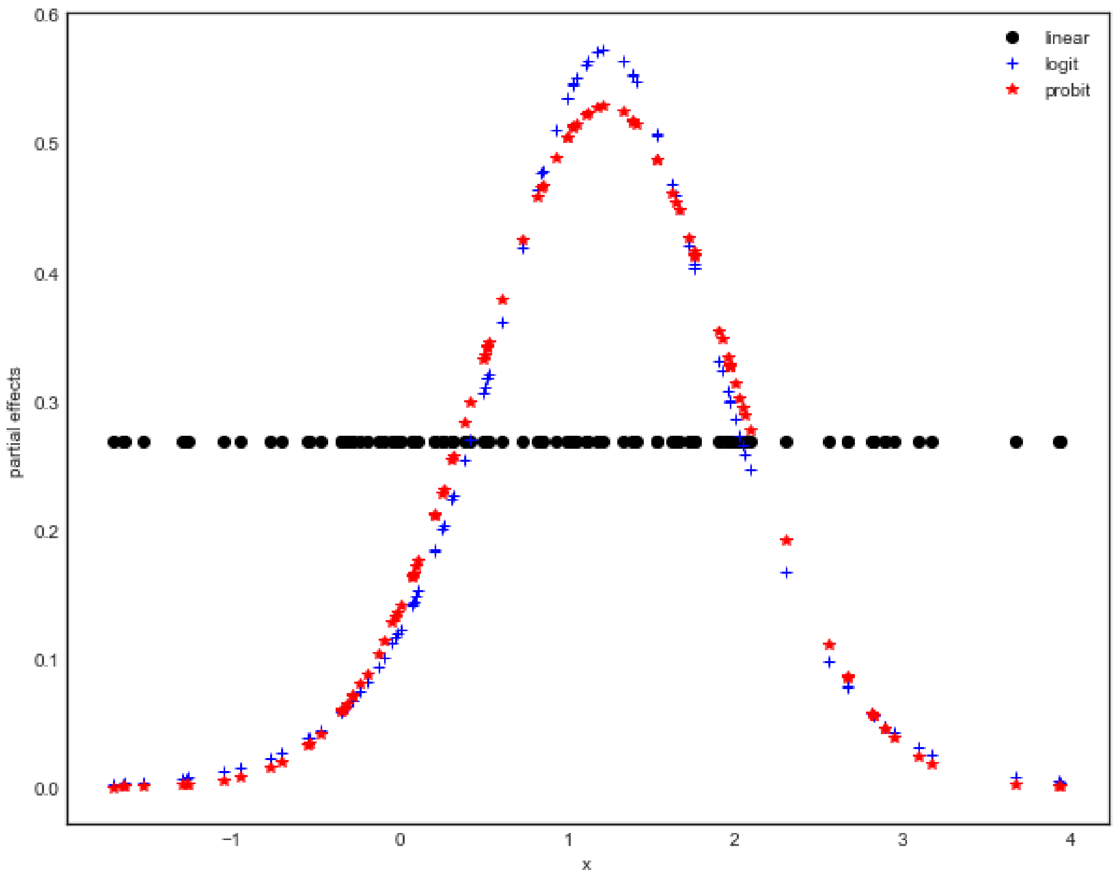


5.0.8 Prediction Comparisons: Linear Model vs. Logit vs. Probit (Marginal Effect Plots)

[61]:
```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import scipy.stats as stats

# set the random seed:
np.random.seed(1234567)

y = stats.binom.rvs(1, 0.5, size=100)
x = stats.norm.rvs(0, 1, size=100) + 2 * y
sim_data = pd.DataFrame({'y': y, 'x': x})

# estimation:
reg_lin = smf.ols(formula='y ~ x', data=sim_data)
results_lin = reg_lin.fit()
```

```python
reg_logit = smf.logit(formula='y ~ x', data=sim_data)
results_logit = reg_logit.fit(disp=0)
reg_probit = smf.probit(formula='y ~ x', data=sim_data)
results_probit = reg_probit.fit(disp=0)

# calculate partial effects:
PE_lin = np.repeat(results_lin.params['x'], 100)

xb_logit = results_logit.fittedvalues
factor_logit = stats.logistic.pdf(xb_logit)
PE_logit = results_logit.params['x'] * factor_logit

xb_probit = results_probit.fittedvalues
factor_probit = stats.norm.pdf(xb_probit)
PE_probit = results_probit.params['x'] * factor_probit


# plot APE's:
fig, ax = plt.subplots(figsize=(10, 8))
plt.plot(x, PE_lin, color='black',
         marker='o', linestyle='', label='linear')
plt.plot(x, PE_logit, color='blue',
         marker='+', linestyle='', label='logit')
plt.plot(x, PE_probit, color='red',
         marker='*', linestyle='', label='probit')
plt.ylabel('partial effects')
plt.xlabel('x')
plt.legend()
```

[61]:
```
<matplotlib.legend.Legend at 0x22dd674bf70>
```



5.0.9 Prediction Comparisons: Linear Model vs. Logit vs. Probit (Marginal Effect Estimates)

[62]:
```
import wooldridge as woo
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import scipy.stats as stats

mroz = woo.dataWoo('mroz')

# estimate models:
reg_lin = smf.ols(formula='inlf ~ nwifeinc + educ + exper + I(exper**2) +'
                          'age + kidslt6 + kidsge6', data=mroz)
results_lin = reg_lin.fit(cov_type='HC3')

reg_logit = smf.logit(formula='inlf ~ nwifeinc + educ + exper + I(exper**2) +'
                              'age + kidslt6 + kidsge6', data=mroz)
results_logit = reg_logit.fit(disp=0)
```

```
reg_probit = smf.probit(formula='inlf ~ nwifeinc + educ + exper + I(exper**2) +'
                               'age + kidslt6 + kidsge6', data=mroz)
results_probit = reg_probit.fit(disp=0)

# manual average partial effects:
APE_lin = np.array(results_lin.params)

xb_logit = results_logit.fittedvalues
factor_logit = np.mean(stats.logistic.pdf(xb_logit))
APE_logit_manual = results_logit.params * factor_logit

xb_probit = results_probit.fittedvalues
factor_probit = np.mean(stats.norm.pdf(xb_probit))
APE_probit_manual = results_probit.params * factor_probit

table_manual = pd.DataFrame({'APE_lin': np.round(APE_lin, 4),
                             'APE_logit_manual': np.round(APE_logit_manual, 4),
                             'APE_probit_manual': np.round(APE_probit_manual,␣
 ↪4)})
print(f'table_manual: \n{table_manual}\n')

# automatic average partial effects:
coef_names = np.array(results_lin.model.exog_names)
coef_names = np.delete(coef_names, 0)  # drop Intercept

APE_logit_autom = results_logit.get_margeff().margeff
APE_probit_autom = results_probit.get_margeff().margeff

table_auto = pd.DataFrame({'coef_names': coef_names,
                           'APE_logit_autom': np.round(APE_logit_autom, 4),
                           'APE_probit_autom': np.round(APE_probit_autom, 4)})
print(f'table_auto: \n{table_auto}\n')
```

```
table_manual:
               APE_lin  APE_logit_manual  APE_probit_manual
Intercept       0.5855            0.0760             0.0812
nwifeinc       -0.0034           -0.0038            -0.0036
educ            0.0380            0.0395             0.0394
exper           0.0395            0.0368             0.0371
I(exper ** 2)  -0.0006           -0.0006            -0.0006
age            -0.0161           -0.0157            -0.0159
kidslt6        -0.2618           -0.2578            -0.2612
kidsge6         0.0130            0.0107             0.0108

table_auto:
      coef_names  APE_logit_autom  APE_probit_autom
0       nwifeinc          -0.0038           -0.0036
```

```
1           educ          0.0395          0.0394
2          exper          0.0368          0.0371
3  I(exper ** 2)         -0.0006         -0.0006
4            age         -0.0157         -0.0159
5        kidslt6         -0.2578         -0.2612
6        kidsge6          0.0107          0.0108
```

6 Instrumental Variables and 2SLS

6.0.1 Instrumental Variables

[48]:
```python
import wooldridge as woo
import numpy as np
import pandas as pd
import linearmodels.iv as iv
import statsmodels.formula.api as smf

card = woo.dataWoo('card')

# checking for relevance with reduced form:
reg_redf = smf.ols(
    formula='educ ~ nearc4 + exper + I(exper**2) + black + smsa +'
    'south + smsa66 + reg662 + reg663 + reg664 + reg665 + reg666 +'
    'reg667 + reg668 + reg669', data=card)
results_redf = reg_redf.fit()

# print regression table:
table_redf = pd.DataFrame({'b': round(results_redf.params, 4),
                           'se': round(results_redf.bse, 4),
                           't': round(results_redf.tvalues, 4),
                           'pval': round(results_redf.pvalues, 4)})
print(f'table_redf: \n{table_redf}\n')

# OLS:
reg_ols = smf.ols(
    formula='np.log(wage) ~ educ + exper + I(exper**2) + black + smsa +'
    'south + smsa66 + reg662 + reg663 + reg664 + reg665 +'
    'reg666 + reg667 + reg668 + reg669', data=card)
results_ols = reg_ols.fit()

# print regression table:
table_ols = pd.DataFrame({'b': round(results_ols.params, 4),
                          'se': round(results_ols.bse, 4),
                          't': round(results_ols.tvalues, 4),
                          'pval': round(results_ols.pvalues, 4)})
print(f'table_ols: \n{table_ols}\n')
```

```python
# IV automatically:
reg_iv = iv.IV2SLS.from_formula(
    formula='np.log(wage)~ 1 + exper + I(exper**2) + black + smsa + '
            'south + smsa66 + reg662 + reg663 + reg664 + reg665 +'
            'reg666 + reg667 + reg668 + reg669 + [educ ~ nearc4]',
    data=card)
results_iv = reg_iv.fit(cov_type='unadjusted', debiased=True)

# print regression table:
table_iv = pd.DataFrame({'b': round(results_iv.params, 4),
                         'se': round(results_iv.std_errors, 4),
                         't': round(results_iv.tstats, 4),
                         'pval': round(results_iv.pvalues, 4)})
print(f'table_iv: \n{table_iv}\n')
```

```
table_redf:
                      b      se        t    pval
Intercept       16.6383  0.2406  69.1446  0.0000
nearc4           0.3199  0.0879   3.6408  0.0003
exper           -0.4125  0.0337 -12.2415  0.0000
I(exper ** 2)    0.0009  0.0017   0.5263  0.5987
black           -0.9355  0.0937  -9.9806  0.0000
smsa             0.4022  0.1048   3.8372  0.0001
south           -0.0516  0.1354  -0.3811  0.7032
smsa66           0.0255  0.1058   0.2409  0.8096
reg662          -0.0786  0.1871  -0.4203  0.6743
reg663          -0.0279  0.1834  -0.1524  0.8789
reg664           0.1172  0.2173   0.5394  0.5897
reg665          -0.2726  0.2184  -1.2481  0.2121
reg666          -0.3028  0.2371  -1.2773  0.2016
reg667          -0.2168  0.2344  -0.9250  0.3550
reg668           0.5239  0.2675   1.9587  0.0502
reg669           0.2103  0.2025   1.0386  0.2991

table_ols:
                     b      se        t    pval
Intercept       4.6208  0.0742  62.2476  0.0000
educ            0.0747  0.0035  21.3510  0.0000
exper           0.0848  0.0066  12.8063  0.0000
I(exper ** 2) -0.0023  0.0003  -7.2232  0.0000
black          -0.1990  0.0182 -10.9058  0.0000
smsa            0.1364  0.0201   6.7851  0.0000
south          -0.1480  0.0260  -5.6950  0.0000
smsa66          0.0262  0.0194   1.3493  0.1773
reg662          0.0964  0.0359   2.6845  0.0073
reg663          0.1445  0.0351   4.1151  0.0000
reg664          0.0551  0.0417   1.3221  0.1862
```

```
reg665         0.1280  0.0418   3.0599  0.0022
reg666         0.1405  0.0452   3.1056  0.0019
reg667         0.1180  0.0448   2.6334  0.0085
reg668        -0.0564  0.0513  -1.1010  0.2710
reg669         0.1186  0.0388   3.0536  0.0023

table_iv:
                    b      se       t    pval
Intercept      3.6662  0.9248  3.9641  0.0001
exper          0.1083  0.0237  4.5764  0.0000
I(exper ** 2) -0.0023  0.0003 -7.0014  0.0000
black         -0.1468  0.0539 -2.7231  0.0065
smsa           0.1118  0.0317  3.5313  0.0004
south         -0.1447  0.0273 -5.3023  0.0000
smsa66         0.0185  0.0216  0.8576  0.3912
reg662         0.1008  0.0377  2.6739  0.0075
reg663         0.1483  0.0368  4.0272  0.0001
reg664         0.0499  0.0437  1.1408  0.2541
reg665         0.1463  0.0471  3.1079  0.0019
reg666         0.1629  0.0519  3.1382  0.0017
reg667         0.1346  0.0494  2.7240  0.0065
reg668        -0.0831  0.0593 -1.4002  0.1616
reg669         0.1078  0.0418  2.5784  0.0100
educ           0.1315  0.0550  2.3926  0.0168
```

6.0.2 Two-Stage Least Squares

[49]:
```python
import wooldridge as woo
import numpy as np
import pandas as pd
import linearmodels.iv as iv
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# restrict to non-missing wage observations:
mroz = mroz.dropna(subset=['lwage'])

# 1st stage (reduced form):
reg_redf = smf.ols(formula='educ ~ exper + I(exper**2) + motheduc + fatheduc',
                   data=mroz)
results_redf = reg_redf.fit()
mroz['educ_fitted'] = results_redf.fittedvalues

# print regression table:
table_redf = pd.DataFrame({'b': round(results_redf.params, 4),
```

```python
                            'se': round(results_redf.bse, 4),
                            't': round(results_redf.tvalues, 4),
                            'pval': round(results_redf.pvalues, 4)})
print(f'table_redf: \n{table_redf}\n')

# 2nd stage:
reg_secstg = smf.ols(formula='np.log(wage) ~ educ_fitted + exper + I(exper**2)',
                     data=mroz)
results_secstg = reg_secstg.fit()

# print regression table:
table_secstg = pd.DataFrame({'b': round(results_secstg.params, 4),
                              'se': round(results_secstg.bse, 4),
                              't': round(results_secstg.tvalues, 4),
                              'pval': round(results_secstg.pvalues, 4)})
print(f'table_secstg: \n{table_secstg}\n')

# IV automatically:
reg_iv = iv.IV2SLS.from_formula(
    formula='np.log(wage) ~ 1 + exper + I(exper**2) +'
            '[educ  ~ motheduc + fatheduc]',
    data=mroz)
results_iv = reg_iv.fit(cov_type='unadjusted', debiased=True)

# print regression table:
table_iv = pd.DataFrame({'b': round(results_iv.params, 4),
                          'se': round(results_iv.std_errors, 4),
                          't': round(results_iv.tstats, 4),
                          'pval': round(results_iv.pvalues, 4)})
print(f'table_iv: \n{table_iv}\n')
```

```
table_redf:
                     b      se        t    pval
Intercept       9.1026  0.4266  21.3396  0.0000
exper           0.0452  0.0403   1.1236  0.2618
I(exper ** 2) -0.0010  0.0012  -0.8386  0.4022
motheduc        0.1576  0.0359   4.3906  0.0000
fatheduc        0.1895  0.0338   5.6152  0.0000

table_secstg:
                     b      se       t    pval
Intercept       0.0481  0.4198  0.1146  0.9088
educ_fitted     0.0614  0.0330  1.8626  0.0632
exper           0.0442  0.0141  3.1361  0.0018
I(exper ** 2) -0.0009  0.0004 -2.1344  0.0334

table_iv:
                     b      se       t    pval
```

```
Intercept      0.0481  0.4003  0.1202  0.9044
exper          0.0442  0.0134  3.2883  0.0011
I(exper ** 2) -0.0009  0.0004 -2.2380  0.0257
educ           0.0614  0.0314  1.9530  0.0515
```

6.0.3 Testing for Exogeneity of the Regressors

[50]:
```python
import wooldridge as woo
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

mroz = woo.dataWoo('mroz')

# restrict to non-missing wage observations:
mroz = mroz.dropna(subset=['lwage'])

# 1st stage (reduced form):
reg_redf = smf.ols(formula='educ ~ exper + I(exper**2) + motheduc + fatheduc',
                   data=mroz)
results_redf = reg_redf.fit()
mroz['resid'] = results_redf.resid

# 2nd stage:
reg_secstg = smf.ols(formula='np.log(wage)~ resid + educ + exper + I(exper**2)',
                     data=mroz)
results_secstg = reg_secstg.fit()

# print regression table:
table_secstg = pd.DataFrame({'b': round(results_secstg.params, 4),
                             'se': round(results_secstg.bse, 4),
                             't': round(results_secstg.tvalues, 4),
                             'pval': round(results_secstg.pvalues, 4)})
print(f'table_secstg: \n{table_secstg}\n')
```

```
table_secstg:
                    b      se       t    pval
Intercept      0.0481  0.3946  0.1219  0.9030
resid          0.0582  0.0348  1.6711  0.0954
educ           0.0614  0.0310  1.9815  0.0482
exper          0.0442  0.0132  3.3363  0.0009
I(exper ** 2) -0.0009  0.0004 -2.2706  0.0237
```

6.0.4 Testing Overidentifying Restrictions

[51]:
```python
import wooldridge as woo
import numpy as np
import pandas as pd
import linearmodels.iv as iv
import statsmodels.formula.api as smf
import scipy.stats as stats

mroz = woo.dataWoo('mroz')

# restrict to non-missing wage observations:
mroz = mroz.dropna(subset=['lwage'])

# IV regression:
reg_iv = iv.IV2SLS.from_formula(formula='np.log(wage) ~ 1 + exper + I(exper**2)␣
↪+'
                                        '[educ ~ motheduc + fatheduc]',␣
↪data=mroz)
results_iv = reg_iv.fit(cov_type='unadjusted', debiased=True)

# print regression table:
table_iv = pd.DataFrame({'b': round(results_iv.params, 4),
                         'se': round(results_iv.std_errors, 4),
                         't': round(results_iv.tstats, 4),
                         'pval': round(results_iv.pvalues, 4)})
print(f'table_iv: \n{table_iv}\n')

# auxiliary regression:
mroz['resid_iv'] = results_iv.resids
reg_aux = smf.ols(formula='resid_iv ~ exper + I(exper**2) + motheduc +␣
↪fatheduc',
                  data=mroz)
results_aux = reg_aux.fit()

# calculations for test:
r2 = results_aux.rsquared
n = results_aux.nobs
teststat = n * r2
pval = 1 - stats.chi2.cdf(teststat, 1)

print(f'r2: {r2}\n')
print(f'n: {n}\n')
print(f'teststat: {teststat}\n')
print(f'pval: {pval}\n')
```

```
table_iv:
                    b       se        t     pval
```

```
Intercept       0.0481  0.4003  0.1202  0.9044
exper           0.0442  0.0134  3.2883  0.0011
I(exper ** 2) -0.0009  0.0004 -2.2380  0.0257
educ            0.0614  0.0314  1.9530  0.0515

r2: 0.0008833444088021114

n: 428.0

teststat: 0.3780714069673037

pval: 0.5386371981604612
```

6.0.5 IVs with Panel Data

[52]:
```
import wooldridge as woo
import pandas as pd
import linearmodels.iv as iv

jtrain = woo.dataWoo('jtrain')

# define panel data (for 1987 and 1988 only):
jtrain_87_88 = jtrain.loc[(jtrain['year'] == 1987) | (jtrain['year'] == 1988), :
↪]
jtrain_87_88 = jtrain_87_88.set_index(['fcode', 'year'])

# manual computation of deviations of entity means:
jtrain_87_88['lscrap_diff1'] = \
    jtrain_87_88.sort_values(['fcode', 'year']).groupby('fcode')['lscrap'].
↪diff()
jtrain_87_88['hrsemp_diff1'] = \
    jtrain_87_88.sort_values(['fcode', 'year']).groupby('fcode')['hrsemp'].
↪diff()
jtrain_87_88['grant_diff1'] = \
    jtrain_87_88.sort_values(['fcode', 'year']).groupby('fcode')['grant'].diff()

# IV regression:
reg_iv = iv.IV2SLS.from_formula(
    formula='lscrap_diff1 ~ 1 + [hrsemp_diff1 ~ grant_diff1]',
    data=jtrain_87_88)
results_iv = reg_iv.fit(cov_type='unadjusted', debiased=True)

# print regression table:
table_iv = pd.DataFrame({'b': round(results_iv.params, 4),
                         'se': round(results_iv.std_errors, 4),
                         't': round(results_iv.tstats, 4),
```

```
                          'pval': round(results_iv.pvalues, 4)})
print(f'table_iv: \n{table_iv}\n')
```

```
table_iv:
                    b      se       t    pval
Intercept    -0.0327  0.1270 -0.2573  0.7982
hrsemp_diff1 -0.0142  0.0079 -1.7882  0.0808


C:\Users\tanne\anaconda3\lib\site-packages\linearmodels\shared\exceptions.py:35:
MissingValueWarning:
Inputs contain missing values. Dropping rows with missing observations.
  warnings.warn(missing_value_warning_msg, MissingValueWarning)
```

7 Treatment Effects and Panel Data

7.0.1 Pooled Cross-Sections

[1]:
```
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf

cps78_85 = woo.dataWoo('cps78_85')

# OLS results including interaction terms:
reg = smf.ols(formula='lwage ~ y85*(educ+female) + exper +'
                      'I((exper**2)/100) + union',
              data=cps78_85)
results = reg.fit()

# print regression table:
table = pd.DataFrame({'b': round(results.params, 4),
                      'se': round(results.bse, 4),
                      't': round(results.tvalues, 4),
                      'pval': round(results.pvalues, 4)})
print(f'table: \n{table}\n')
```

```
table:
                          b      se        t    pval
Intercept            0.4589  0.0934   4.9111  0.0000
y85                  0.1178  0.1238   0.9517  0.3415
educ                 0.0747  0.0067  11.1917  0.0000
female              -0.3167  0.0366  -8.6482  0.0000
y85:educ             0.0185  0.0094   1.9735  0.0487
y85:female           0.0851  0.0513   1.6576  0.0977
exper                0.0296  0.0036   8.2932  0.0000
I((exper ** 2) / 100) -0.0399  0.0078  -5.1513  0.0000
union                0.2021  0.0303   6.6722  0.0000
```

7.0.2 Difference-in-Differences

[2]:
```python
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf

kielmc = woo.dataWoo('kielmc')

# separate regressions for 1978 and 1981:
y78 = (kielmc['year'] == 1978)
reg78 = smf.ols(formula='rprice ~ nearinc', data=kielmc, subset=y78)
results78 = reg78.fit()

y81 = (kielmc['year'] == 1981)
reg81 = smf.ols(formula='rprice ~ nearinc', data=kielmc, subset=y81)
results81 = reg81.fit()

# joint regression including an interaction term:
reg_joint = smf.ols(formula='rprice ~ nearinc * C(year)', data=kielmc)
results_joint = reg_joint.fit()

# print regression tables:
table_78 = pd.DataFrame({'b': round(results78.params, 4),
                         'se': round(results78.bse, 4),
                         't': round(results78.tvalues, 4),
                         'pval': round(results78.pvalues, 4)})
print(f'table_78: \n{table_78}\n')

table_81 = pd.DataFrame({'b': round(results81.params, 4),
                         'se': round(results81.bse, 4),
                         't': round(results81.tvalues, 4),
                         'pval': round(results81.pvalues, 4)})
print(f'table_81: \n{table_81}\n')

table_joint = pd.DataFrame({'b': round(results_joint.params, 4),
                            'se': round(results_joint.bse, 4),
                            't': round(results_joint.tvalues, 4),
                            'pval': round(results_joint.pvalues, 4)})
print(f'table_joint: \n{table_joint}\n')
```

```
table_78:
                   b        se        t    pval
Intercept  82517.2276  2653.790  31.0941  0.0000
nearinc   -18824.3705  4744.594  -3.9675  0.0001

table_81:
                    b         se        t  pval
Intercept  101307.5136  3093.0267  32.7535   0.0
```

```
nearinc    -30688.2738  5827.7088  -5.2659   0.0

table_joint:
                                  b         se        t    pval
Intercept                82517.2276  2726.9101  30.2603  0.0000
C(year)[T.1981]          18790.2860  4050.0650   4.6395  0.0000
nearinc                 -18824.3705  4875.3221  -3.8612  0.0001
nearinc:C(year)[T.1981] -11863.9033  7456.6462  -1.5911  0.1126
```

7.0.3 First Difference Estimator

[3]:
```python
import wooldridge as woo
import numpy as np
import linearmodels as plm

crime4 = woo.dataWoo('crime4')
crime4 = crime4.set_index(['county', 'year'], drop=False)

# estimate FD model:
reg = plm.FirstDifferenceOLS.from_formula(
    formula='np.log(crmrte) ~ year + d83 + d84 + d85 + d86 + d87 +'
            'lprbarr + lprbconv + lprbpris + lavgsen + lpolpc',
    data=crime4)
results = reg.fit()
print(f'results: \n{results}\n')
```

```
results:
                     FirstDifferenceOLS Estimation Summary
================================================================================
Dep. Variable:         np.log(crmrte)   R-squared:                        0.4326
Estimator:         FirstDifferenceOLS   R-squared (Between):              0.6003
No. Observations:                 540   R-squared (Within):               0.4281
Date:                Mon, Dec 06 2021   R-squared (Overall):              0.6000
Time:                        13:59:29   Log-likelihood                    248.48
Cov. Estimator:            Unadjusted
                                        F-statistic:                      36.661
Entities:                          90   P-value                           0.0000
Avg Obs:                       7.0000   Distribution:                   F(11,529)
Min Obs:                       7.0000
Max Obs:                       7.0000   F-statistic (robust):             36.661
                                        P-value                           0.0000
Time periods:                       7   Distribution:                   F(11,529)
Avg Obs:                       90.000
Min Obs:                       90.000
Max Obs:                       90.000

                             Parameter Estimates
```

```
==============================================================================
              Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
------------------------------------------------------------------------------
year             0.0077     0.0171     0.4522     0.6513     -0.0258      0.0412
d83             -0.0999     0.0239    -4.1793     0.0000     -0.1468     -0.0529
d84             -0.1478     0.0413    -3.5806     0.0004     -0.2289     -0.0667
d85             -0.1524     0.0584    -2.6098     0.0093     -0.2671     -0.0377
d86             -0.1249     0.0760    -1.6433     0.1009     -0.2742      0.0244
d87             -0.0841     0.0940    -0.8944     0.3715     -0.2687      0.1006
lprbarr         -0.3275     0.0300    -10.924     0.0000     -0.3864     -0.2686
lprbconv        -0.2381     0.0182    -13.058     0.0000     -0.2739     -0.2023
lprbpris        -0.1650     0.0260    -6.3555     0.0000     -0.2161     -0.1140
lavgsen         -0.0218     0.0221    -0.9850     0.3251     -0.0652      0.0216
lpolpc           0.3984     0.0269     14.821     0.0000      0.3456      0.4512
==============================================================================
```

7.0.4 Fixed Effects Estimation

[4]:
```python
import wooldridge as woo
import pandas as pd
import linearmodels as plm

wagepan = woo.dataWoo('wagepan')
wagepan = wagepan.set_index(['nr', 'year'], drop=False)

# FE model estimation:
reg = plm.PanelOLS.from_formula(
    formula='lwage ~ married + union + C(year)*educ + EntityEffects',
    data=wagepan, drop_absorbed=True)
results = reg.fit()

# print regression table:
table = pd.DataFrame({'b': round(results.params, 4),
                      'se': round(results.std_errors, 4),
                      't': round(results.tstats, 4),
                      'pval': round(results.pvalues, 4)})
print(f'table: \n{table}\n')
```

```
table:
                            b      se        t    pval
C(year)[1980]          1.3625  0.0162  83.9031  0.0000
C(year)[1981]          1.3400  0.1452   9.2307  0.0000
C(year)[1982]          1.3567  0.1451   9.3481  0.0000
C(year)[1983]          1.3729  0.1452   9.4561  0.0000
C(year)[1984]          1.4468  0.1452   9.9617  0.0000
C(year)[1985]          1.4122  0.1451   9.7315  0.0000
C(year)[1986]          1.4281  0.1451   9.8404  0.0000
```

```
C(year)[1987]          1.4529  0.1452  10.0061  0.0000
married                0.0548  0.0184   2.9773  0.0029
union                  0.0830  0.0194   4.2671  0.0000
C(year)[T.1981]:educ   0.0116  0.0123   0.9448  0.3448
C(year)[T.1982]:educ   0.0148  0.0123   1.2061  0.2279
C(year)[T.1983]:educ   0.0171  0.0123   1.3959  0.1628
C(year)[T.1984]:educ   0.0166  0.0123   1.3521  0.1764
C(year)[T.1985]:educ   0.0237  0.0123   1.9316  0.0535
C(year)[T.1986]:educ   0.0274  0.0123   2.2334  0.0256
C(year)[T.1987]:educ   0.0304  0.0123   2.4798  0.0132


C:\Users\tanne\anaconda3\lib\site-packages\linearmodels\panel\model.py:1796:
AbsorbingEffectWarning:
Variables have been fully absorbed and have removed from the regression:

educ

  warnings.warn(
```

7.0.5 Panel Data Inspection

[6]:
```
import wooldridge as woo

wagepan = woo.dataWoo('wagepan')

# print relevant dimensions for panel:
N = wagepan.shape[0]
T = wagepan['year'].drop_duplicates().shape[0]
n = wagepan['nr'].drop_duplicates().shape[0]
print(f'N: {N}\n')
print(f'T: {T}\n')
print(f'n: {n}\n')

# check non-varying variables

# (I) across time and within individuals by calculating individual
# specific variances for each variable:
isv_nr = (wagepan.groupby('nr').var() == 0)  # True, if variance is zero
# choose variables where all grouped variances are zero:
noVar_nr = isv_nr.all(axis=0)  # which cols are completely True
print(f'isv_nr.columns[noVar_nr]: \n{isv_nr.columns[noVar_nr]}\n')

# (II) across individuals within one point in time for each variable:
isv_t = (wagepan.groupby('year').var() == 0)
noVar_t = isv_t.all(axis=0)
print(f'isv_t.columns[noVar_t]: \n{isv_t.columns[noVar_t]}\n')
```

```
N: 4360

T: 8

n: 545

isv_nr.columns[noVar_nr]:
Index(['black', 'hisp', 'educ'], dtype='object')

isv_t.columns[noVar_t]:
Index(['d81', 'd82', 'd83', 'd84', 'd85', 'd86', 'd87'], dtype='object')
```

7.0.6 Pooled, Fixed and Random Effects Comparison

[7]:
```python
import wooldridge as woo
import pandas as pd
import linearmodels as plm

wagepan = woo.dataWoo('wagepan')

# estimate different models:
wagepan = wagepan.set_index(['nr', 'year'], drop=False)

reg_ols = plm.PooledOLS.from_formula(
    formula='lwage ~ educ + black + hisp + exper + I(exper**2) +'
            'married + union + C(year)', data=wagepan)
results_ols = reg_ols.fit()

reg_re = plm.RandomEffects.from_formula(
    formula='lwage ~ educ + black + hisp + exper + I(exper**2) +'
            'married + union + C(year)', data=wagepan)
results_re = reg_re.fit()

reg_fe = plm.PanelOLS.from_formula(
    formula='lwage ~ I(exper**2) + married + union +'
            'C(year) + EntityEffects', data=wagepan)
results_fe = reg_fe.fit()

# print results:
theta_hat = results_re.theta.iloc[0, 0]
print(f'theta_hat: {theta_hat}\n')

table_ols = pd.DataFrame({'b': round(results_ols.params, 4),
                          'se': round(results_ols.std_errors, 4),
                          't': round(results_ols.tstats, 4),
                          'pval': round(results_ols.pvalues, 4)})
```

```python
print(f'table_ols: \n{table_ols}\n')

table_re = pd.DataFrame({'b': round(results_re.params, 4),
                         'se': round(results_re.std_errors, 4),
                         't': round(results_re.tstats, 4),
                         'pval': round(results_re.pvalues, 4)})
print(f'table_re: \n{table_re}\n')

table_fe = pd.DataFrame({'b': round(results_fe.params, 4),
                         'se': round(results_fe.std_errors, 4),
                         't': round(results_fe.tstats, 4),
                         'pval': round(results_fe.pvalues, 4)})
print(f'table_fe: \n{table_fe}\n')
```

```
theta_hat: 0.6450593029243452

table_ols:
                     b      se        t    pval
C(year)[1980]   0.0921  0.0783   1.1761  0.2396
C(year)[1981]   0.1504  0.0838   1.7935  0.0730
C(year)[1982]   0.1548  0.0893   1.7335  0.0831
C(year)[1983]   0.1541  0.0944   1.6323  0.1027
C(year)[1984]   0.1825  0.0990   1.8437  0.0653
C(year)[1985]   0.2013  0.1031   1.9523  0.0510
C(year)[1986]   0.2340  0.1068   2.1920  0.0284
C(year)[1987]   0.2659  0.1100   2.4166  0.0157
educ            0.0913  0.0052  17.4419  0.0000
black          -0.1392  0.0236  -5.9049  0.0000
hisp            0.0160  0.0208   0.7703  0.4412
exper           0.0672  0.0137   4.9095  0.0000
I(exper ** 2) -0.0024  0.0008  -2.9413  0.0033
married         0.1083  0.0157   6.8997  0.0000
union           0.1825  0.0172  10.6349  0.0000

table_re:
                     b      se       t    pval
C(year)[1980]   0.0234  0.1514  0.1546  0.8771
C(year)[1981]   0.0638  0.1601  0.3988  0.6901
C(year)[1982]   0.0543  0.1690  0.3211  0.7481
C(year)[1983]   0.0436  0.1780  0.2450  0.8065
C(year)[1984]   0.0664  0.1871  0.3551  0.7225
C(year)[1985]   0.0811  0.1961  0.4136  0.6792
C(year)[1986]   0.1152  0.2052  0.5617  0.5744
C(year)[1987]   0.1583  0.2143  0.7386  0.4602
educ            0.0919  0.0107  8.5744  0.0000
black          -0.1394  0.0480 -2.9054  0.0037
hisp            0.0217  0.0428  0.5078  0.6116
exper           0.1058  0.0154  6.8706  0.0000
```

```
I(exper ** 2) -0.0047  0.0007 -6.8623  0.0000
married         0.0638  0.0168  3.8035  0.0001
union           0.1059  0.0179  5.9289  0.0000

table_fe:
                     b      se        t    pval
C(year)[1980]   1.4260  0.0183  77.7484  0.0000
C(year)[1981]   1.5772  0.0216  72.9656  0.0000
C(year)[1982]   1.6790  0.0265  63.2583  0.0000
C(year)[1983]   1.7805  0.0333  53.4392  0.0000
C(year)[1984]   1.9161  0.0417  45.9816  0.0000
C(year)[1985]   2.0435  0.0515  39.6460  0.0000
C(year)[1986]   2.1915  0.0630  34.7714  0.0000
C(year)[1987]   2.3510  0.0762  30.8669  0.0000
I(exper ** 2) -0.0052  0.0007  -7.3612  0.0000
married         0.0467  0.0183   2.5494  0.0108
union           0.0800  0.0193   4.1430  0.0000
```

7.0.7 Correlated Random Effects

[8]:
```python
import wooldridge as woo
import pandas as pd
import statsmodels.formula.api as smf
import linearmodels as plm

wagepan = woo.dataWoo('wagepan')
wagepan['t'] = wagepan['year']
wagepan['entity'] = wagepan['nr']
wagepan = wagepan.set_index(['nr'])

# include group specific means:
wagepan['married_b'] = wagepan.groupby('nr').mean()['married']
wagepan['union_b'] = wagepan.groupby('nr').mean()['union']
wagepan = wagepan.set_index(['year'], append=True)

# estimate FE parameters in 3 different ways:
reg_we = plm.PanelOLS.from_formula(
    formula='lwage ~ married + union + C(t)*educ + EntityEffects',
    drop_absorbed=True, data=wagepan)
results_we = reg_we.fit()

reg_dum = smf.ols(
    formula='lwage ~ married + union + C(t)*educ + C(entity)',
    data=wagepan)
results_dum = reg_dum.fit()
```

```python
# estimate CRE:
reg_cre = plm.RandomEffects.from_formula(
    formula='lwage ~ married + union + C(t)*educ + married_b + union_b',
    data=wagepan)
results_cre = reg_cre.fit()

# compare to RE estimates:
reg_re = plm.RandomEffects.from_formula(
    formula='lwage ~ married + union + C(t)*educ',
    data=wagepan)
results_re = reg_re.fit()

var_selection = ['married', 'union', 'C(t)[T.1982]:educ']

# print results:
table = pd.DataFrame({'b_we': round(results_we.params[var_selection], 4),
                      'b_dum': round(results_dum.params[var_selection], 4),
                      'b_cre': round(results_cre.params[var_selection], 4),
                      'b_re': round(results_re.params[var_selection], 4)})
print(f'table: \n{table}\n')

# CRE Test:

# RE test as an Wald test on the CRE specific coefficients:
wtest = results_cre.wald_test(formula='married_b = union_b = 0')
print(f'wtest: \n{wtest}\n')
```

```
C:\Users\tanne\anaconda3\lib\site-packages\linearmodels\panel\model.py:1796:
AbsorbingEffectWarning:
Variables have been fully absorbed and have removed from the regression:

educ

  warnings.warn(
```

```
table:
                     b_we   b_dum   b_cre    b_re
married            0.0548  0.0548  0.0548  0.0773
union              0.0830  0.0830  0.0830  0.1075
C(t)[T.1982]:educ  0.0148  0.0148  0.0148  0.0143

wtest:
Linear Equality Hypothesis Test
H0: Linear equality constraint is valid
Statistic: 19.4058
P-value: 0.0001
Distributed: chi2(2)
```

7.0.8 Robust (Clustered) Standard Errors

[9]:
```python
import wooldridge as woo
import numpy as np
import pandas as pd
import linearmodels as plm

crime4 = woo.dataWoo('crime4')
crime4 = crime4.set_index(['county', 'year'], drop=False)

# estimate FD model:
reg = plm.FirstDifferenceOLS.from_formula(
    formula='np.log(crmrte) ~ year + d83 + d84 + d85 + d86 + d87 +'
            'lprbarr + lprbconv + lprbpris + lavgsen + lpolpc',
    data=crime4)

# regression with standard SE:
results_default = reg.fit()

# regression with "clustered" SE:
results_cluster = reg.fit(cov_type='clustered', cluster_entity=True,
                          debiased=False)

# regression with "clustered" SE (small-sample correction):
results_css = reg.fit(cov_type='clustered', cluster_entity=True)

# print results:
table = pd.DataFrame({'b': round(results_default.params, 4),
                      'se_default': round(results_default.std_errors, 4),
                      'se_cluster': round(results_cluster.std_errors, 4),
                      'se_css': round(results_css.std_errors, 4)})
print(f'table: \n{table}\n')
```

```
table:
                b  se_default  se_cluster  se_css
year       0.0077      0.0171      0.0136  0.0137
d83       -0.0999      0.0239      0.0219  0.0222
d84       -0.1478      0.0413      0.0356  0.0359
d85       -0.1524      0.0584      0.0505  0.0511
d86       -0.1249      0.0760      0.0624  0.0630
d87       -0.0841      0.0940      0.0773  0.0781
lprbarr   -0.3275      0.0300      0.0556  0.0562
lprbconv  -0.2381      0.0182      0.0390  0.0394
lprbpris  -0.1650      0.0260      0.0451  0.0456
lavgsen   -0.0218      0.0221      0.0254  0.0257
lpolpc     0.3984      0.0269      0.1014  0.1025
```

7.0.9 Hausman Test

[10]:
```
import wooldridge as woo
import numpy as np
import linearmodels as plm
import scipy.stats as stats

wagepan = woo.dataWoo('wagepan')
wagepan = wagepan.set_index(['nr', 'year'], drop=False)

# estimation of FE and RE:
reg_fe = plm.PanelOLS.from_formula(formula='lwage ~ I(exper**2) + married +'
                                           'union + C(year) + EntityEffects',
                                   data=wagepan)
results_fe = reg_fe.fit()
b_fe = results_fe.params
b_fe_cov = results_fe.cov

reg_re = plm.RandomEffects.from_formula(
    formula='lwage ~ educ + black + hisp + exper + I(exper**2)'
            '+ married + union + C(year)', data=wagepan)
results_re = reg_re.fit()
b_re = results_re.params
b_re_cov = results_re.cov

# Hausman test of FE vs. RE
# (I) find overlapping coefficients:
common_coef = set(results_fe.params.index).intersection(results_re.params.index)

# (II) calculate differences between FE and RE:
b_diff = np.array(results_fe.params[common_coef] - results_re.
 ↪params[common_coef])
df = len(b_diff)
b_diff.reshape((df, 1))
b_cov_diff = np.array(b_fe_cov.loc[common_coef, common_coef] -
                      b_re_cov.loc[common_coef, common_coef])
b_cov_diff.reshape((df, df))

# (III) calculate test statistic:
stat = abs(np.transpose(b_diff) @ np.linalg.inv(b_cov_diff) @ b_diff)
pval = 1 - stats.chi2.cdf(stat, df)

print(f'stat: {stat}\n')
print(f'pval: {pval}\n')
```

```
stat: 43.42707117572976

pval: 9.150613851094391e-06
```

8 Time Series

8.0.1 Importing Time Series

[2]:
```
import pandas as pd
data = pd.read_csv('a10.csv',parse_dates=True, index_col="index")
data.head()
data.index.freq='MS'
data.head()

# Added sanity checks
# data.isna().sum()
# data.index.nunique()
# pd.crosstab(index=data.index, columns=data.index.month)
```

[2]:
```
               value
index
1991-07-01  3.526591
1991-08-01  3.180891
1991-09-01  3.252221
1991-10-01  3.611003
1991-11-01  3.565869
```

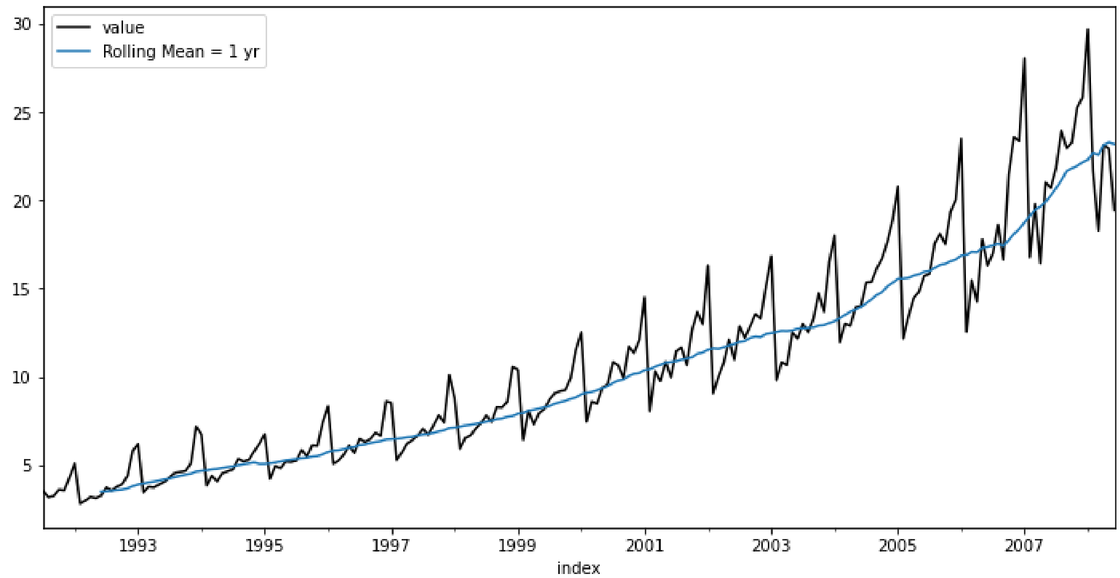
8.0.2 Conditions for Stationarity

1. $E[X_t] = Constant = \mu$
2. $V(X_t) = Constant = \sigma^2$
3. $CV(X_t, X_{t+h}) = f(h) \neq f(t)$

Cannot model a non-stationary process whose properties change with time (at least not easily). Want to reduce the residuals to effectively white noise, so we can fit a model.

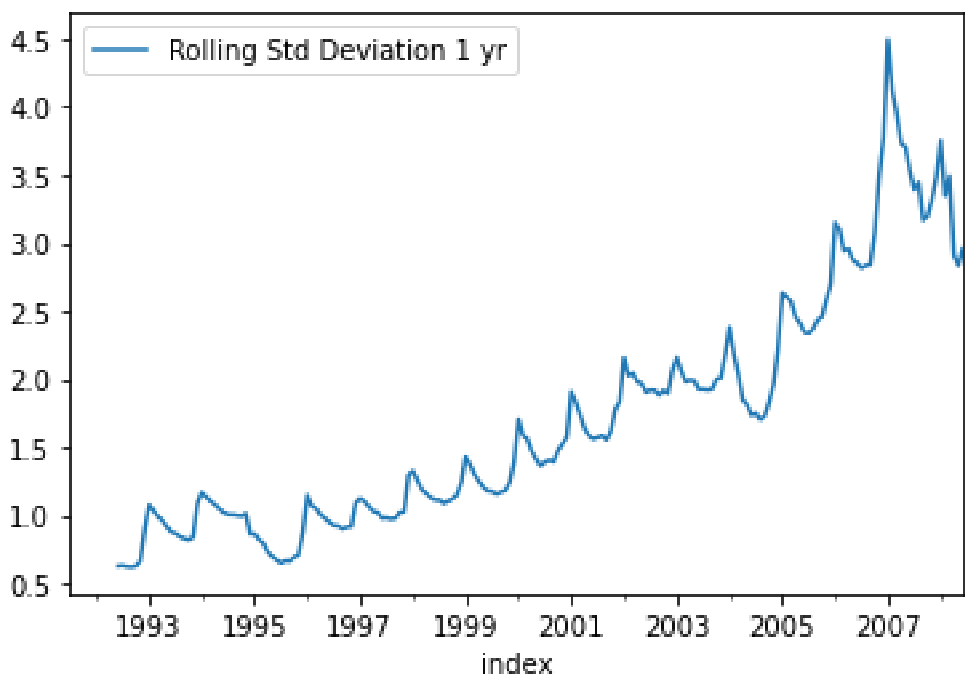
[3]:
```
#  Mean Reversion (constant mean?)
data.plot(figsize=(12,6), legend=True, label="a10", cmap='gray')
data["value"].rolling(12, center=False).mean().plot(legend=True, label="Rolling␣
↪Mean = 1 yr");
print("Mean is:", data["value"].mean())
```

```
Mean is: 10.694429582156861
```



[4]:
```
# Constant variance?
data["value"].rolling(12).std().plot(legend=True, label="Rolling Std Deviation␣
↪1 yr");
print("S.D is:", data["value"].std())
```

S.D is: 5.956997805897407



8.0.3 Summary of Time Series Analysis

1. **Inspect the data.**
2. **Plot the first order conditions.**
3. **Test the series for evidence of stationarity via an ADF test.**
4. **Multiplicatively or additively decompose the series.**
5. **Plot the ACF and PACF of the series.**
6. **From the decomposition, set the following elements of the series to variables:**
   - 6a. Original series
   - 6b. Trend
   - 6c. Seasonality
   - 6d. Cycles/residuals/irregularities
7. **If present, then remove the seasonality component via one of the following methods:**
   - 7a. Additive seasonality – Original series - seasonal component
   - 7b. Multiplicative seasonality – Original series / seasonal component
8. **If present, then remove the trend component in one of the following methods:**
   - 8a. No previous seasonal adjustment – Original series - trend component
   - 8b. Previous seasonal adjustment – Seasonally-adjusted series - trend component
9. **Check cycles for evidence of stationarity via an ADF test**
10. **If necessary, stationarize (and then test) the cycles through one of the following methods:**
    - 10a. Take the first difference of the cycles.
    - 10b. Take the log of the residuals, then take the first difference of the logged cycles.
11. **Plot the ACF and PACF of the stationarized cycles (or whatever series is most current).**
12. **Implement one of the following methods to fit a model to the cycles:**
    - 12a. ARMA
    - 12b. Auto-ARIMA
    - 12c. Manual ARIMA
13. **Verify that the residuals of the fitted model are approximate to white noise by plotting their ACF and PACF.**
14. **Split dataset into two halves composing a training and testing set.**

15. **Forecast the cycles.**
16. **Forecast the original series.**

8.0.4 First Order Properties

First Order Properties:

1. **Data inspection**
2. **Trend**

   Deterministic trend – Trend evolves in a predictable way

   Stochastic trend – Trend evolves in an unpredictable way (e.g. Random walk)
3. **Seasonality**

   Additive seasonality – Seasonal fluctations do not vary much with time, $y_t = S_t + T_t + R_t$

   Multiplicative seasonality – Seasonal fluctations do vary with time, $y_t = (S_t)(T_t)(R_t)$
4. **Cycles**

   Deterministic cycles – Cycles evolve in a predictable way

   Stochastic cycles – Cycles evolve in an unpredictable way (e.g. Random walk)

8.0.5 Second Order Properties

Second Order Properties:

1. Series decomposition

   Multiplicative decomposition

   Additive decomposition
2. ACF and PACF

   ACF –

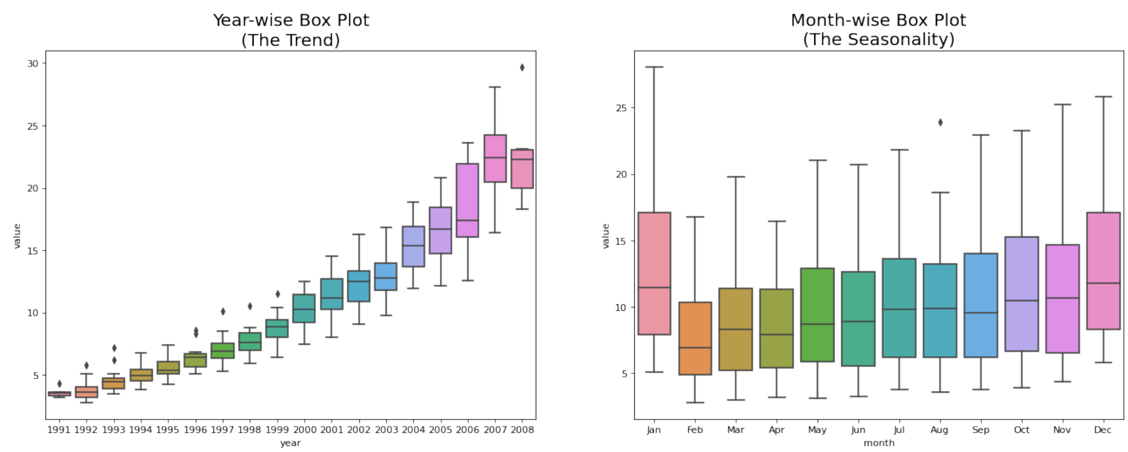
   PACF –
3. Stationarity

   (See: Time Series, Conditions for Stationarity)

8.0.6 Trend and Seasonality

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the seasonanl and annual variation

df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.
 ↪csv', parse_dates=['date'], index_col='date')
df.reset_index(inplace=True)
```

```python
# Prepare data
df['year'] = [d.year for d in df.date]
df['month'] = [d.strftime('%b') for d in df.date]
years = df['year'].unique()

# Draw Plot
fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80)
sns.boxplot(x='year', y='value', data=df, ax=axes[0])
sns.boxplot(x='month', y='value', data=df.loc[~df.year.isin([1991, 2008]), :])

# Set Title
axes[0].set_title('Year-wise Box Plot\n(The Trend)', fontsize=18);
axes[1].set_title('Month-wise Box Plot\n(The Seasonality)', fontsize=18)
plt.show()
```

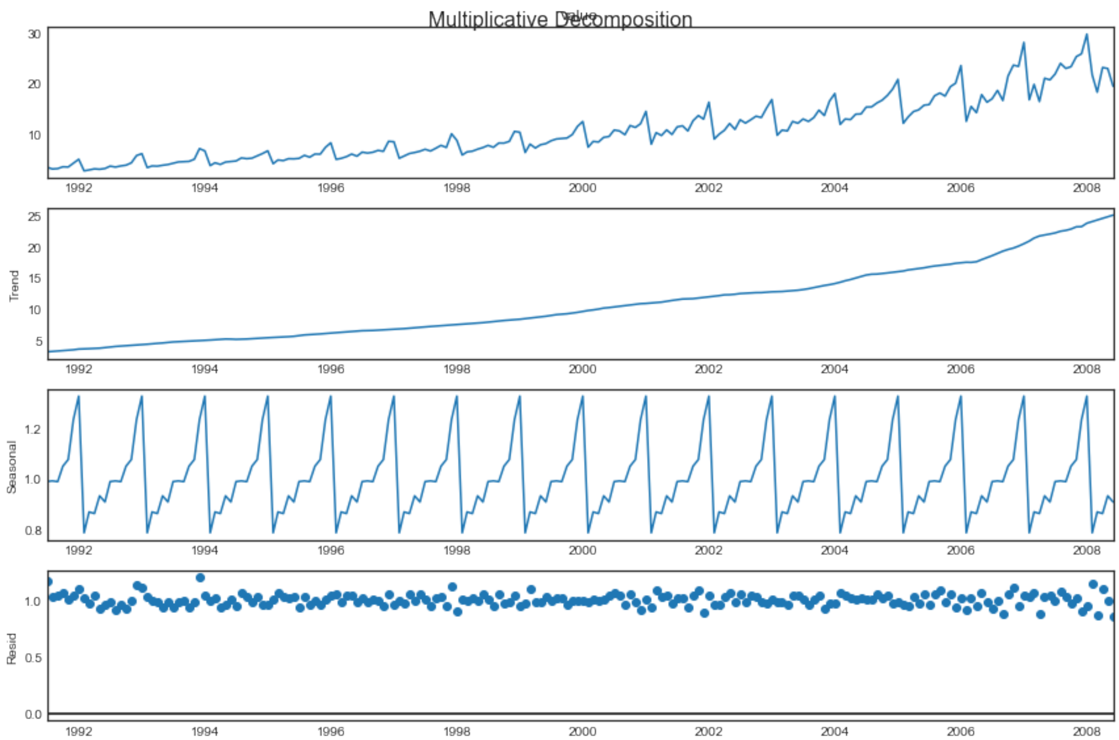


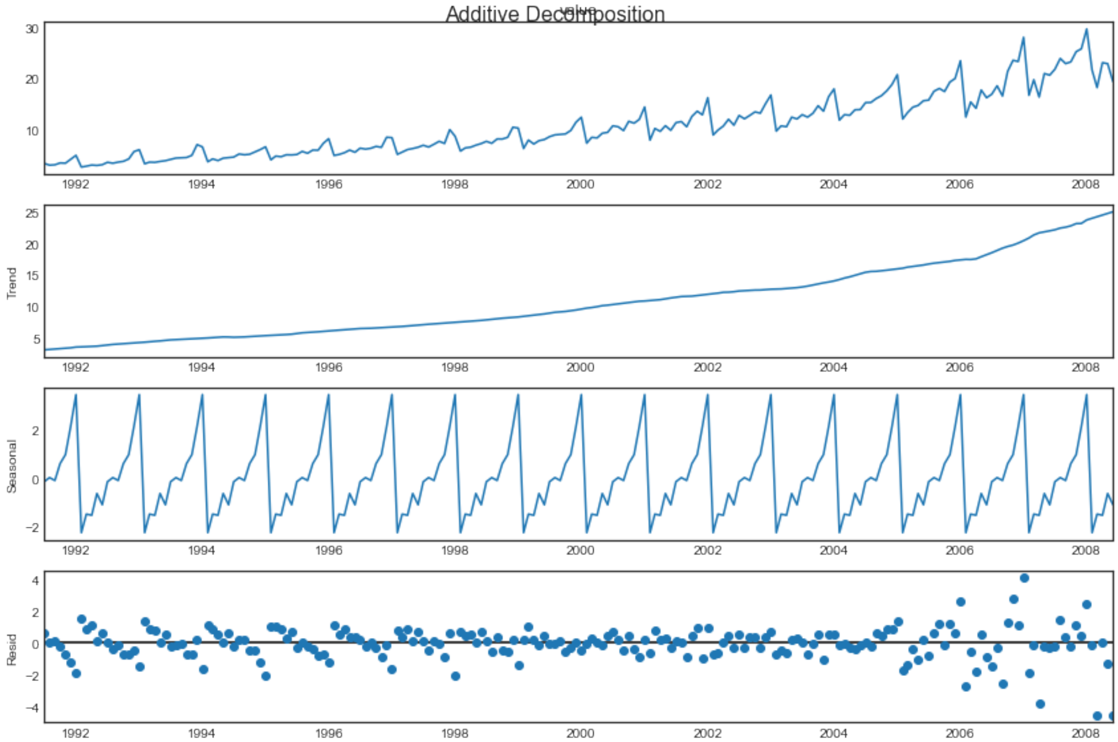
8.0.7 Decomposition

[6]:
```python
from statsmodels.tsa.seasonal import seasonal_decompose
#Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt
plt.style.use('seaborn-white')
%matplotlib inline

# Multiplicative Decomposition
decomposeM = seasonal_decompose(data["value"],model='multiplicative',␣
 ↪extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
#decomposeM.plot();
```

```
decomposeM.plot().suptitle('Multiplicative Decomposition', fontsize=16)


# Additive Decomposition
decomposeA = seasonal_decompose(data["value"],model='additive',␣
 ↪extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
decomposeA.plot().suptitle('Additive Decomposition', fontsize=16)
```

[6]: Text(0.5, 0.98, 'Additive Decomposition')





8.0.8 Extracting Elements of Decomposition

[14]:
```
df_reconstructed = pd.concat([decomposeM.seasonal, decomposeM.trend, decomposeM.
↪resid, decomposeM.observed], axis=1)
df_reconstructed.columns = ['seas', 'trend', 'resid', 'actual_values']
df_reconstructed.head()
```

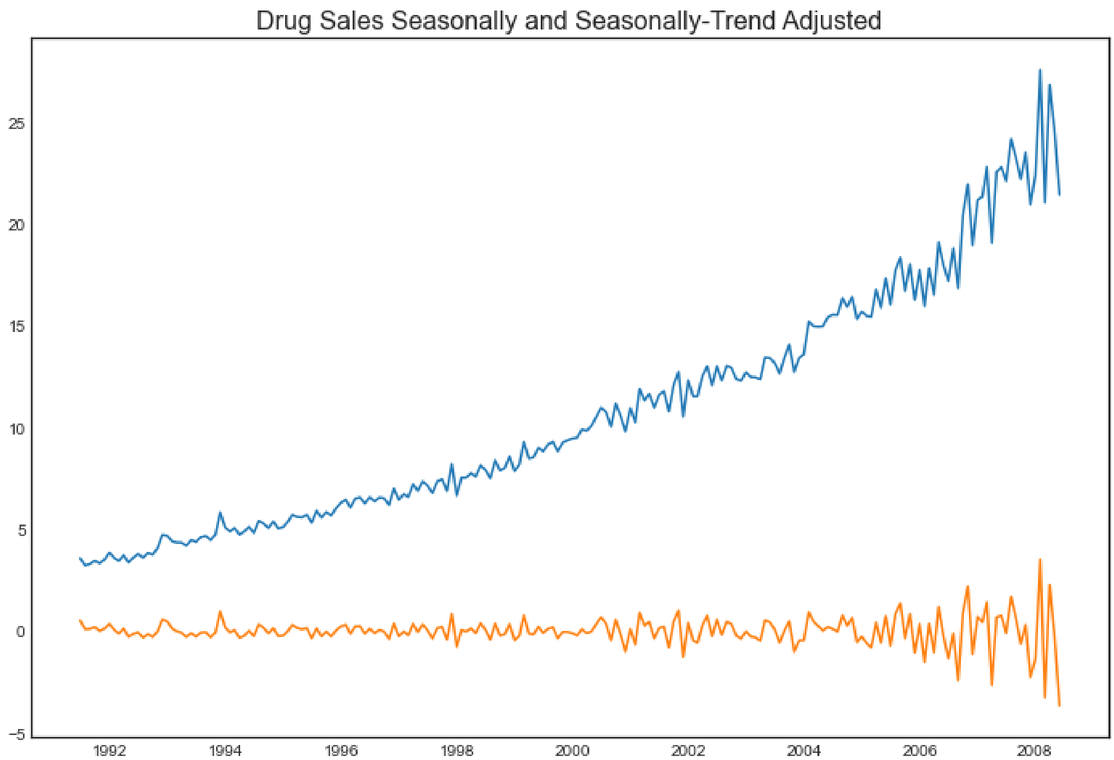
[14]:

index	seas	trend	resid	actual_values
1991-07-01	0.987845	3.060085	1.166629	3.526591
1991-08-01	0.990481	3.124765	1.027745	3.180891
1991-09-01	0.987476	3.189445	1.032615	3.252221
1991-10-01	1.048329	3.254125	1.058513	3.611003
1991-11-01	1.074527	3.318805	0.999923	3.565869

8.0.9 Removing Seasonal and Trend Components

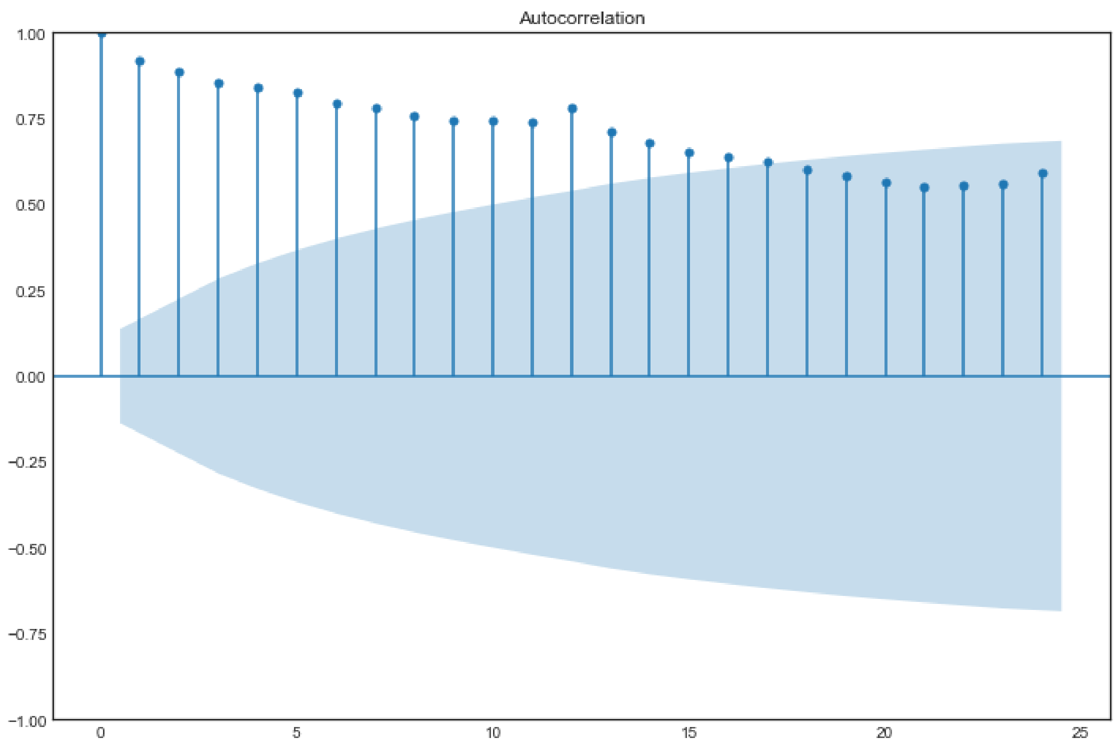
[9]:
```
# Removing seasonal component
Deseasonalized = decomposeM.observed / decomposeM.seasonal
plt.plot(Deseasonalized)

# Removing trend component
cycles = Deseasonalized - decomposeM.trend
```

```
plt.plot(cycles)
plt.title('Drug Sales Seasonally and Seasonally-Trend Adjusted', fontsize=16)
plt.show()
```



8.0.10 Autocorrelation Function (ACF)

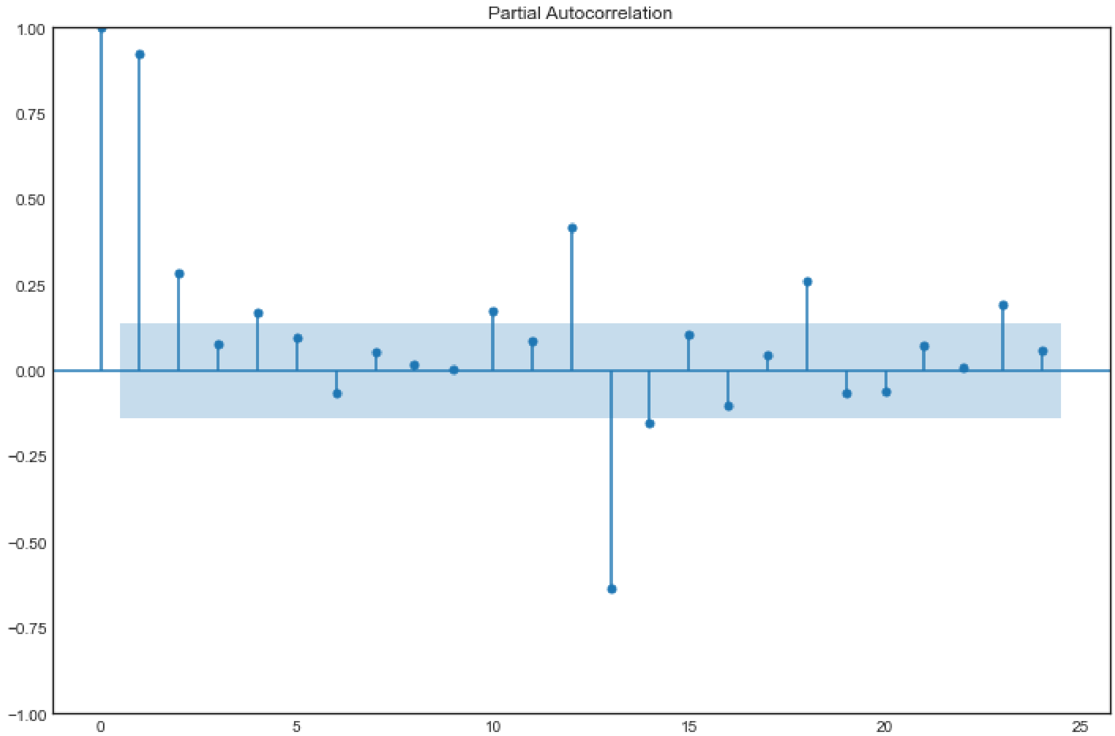
[10]:
```
# Plot the ACF
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf,␣
↪plot_pacf, quarter_plot
plot_acf(data);
```



8.0.11 Partial Autocorrelation Function (PACF)

[11]:
```
# Plot the PACF
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf,␣
↪plot_pacf, quarter_plot
plot_pacf(data);
```

```
C:\Users\tanne\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



8.0.12 ADF Test for Stationarity

[19]:
```
# ADF Test for Stationarity
from statsmodels.tsa.stattools import adfuller
adf = adfuller(cycles)[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else ", Series is␣
↪Non-Stationary")
```

```
p value:1.8854672080574665e-08 , Series is Stationary
```

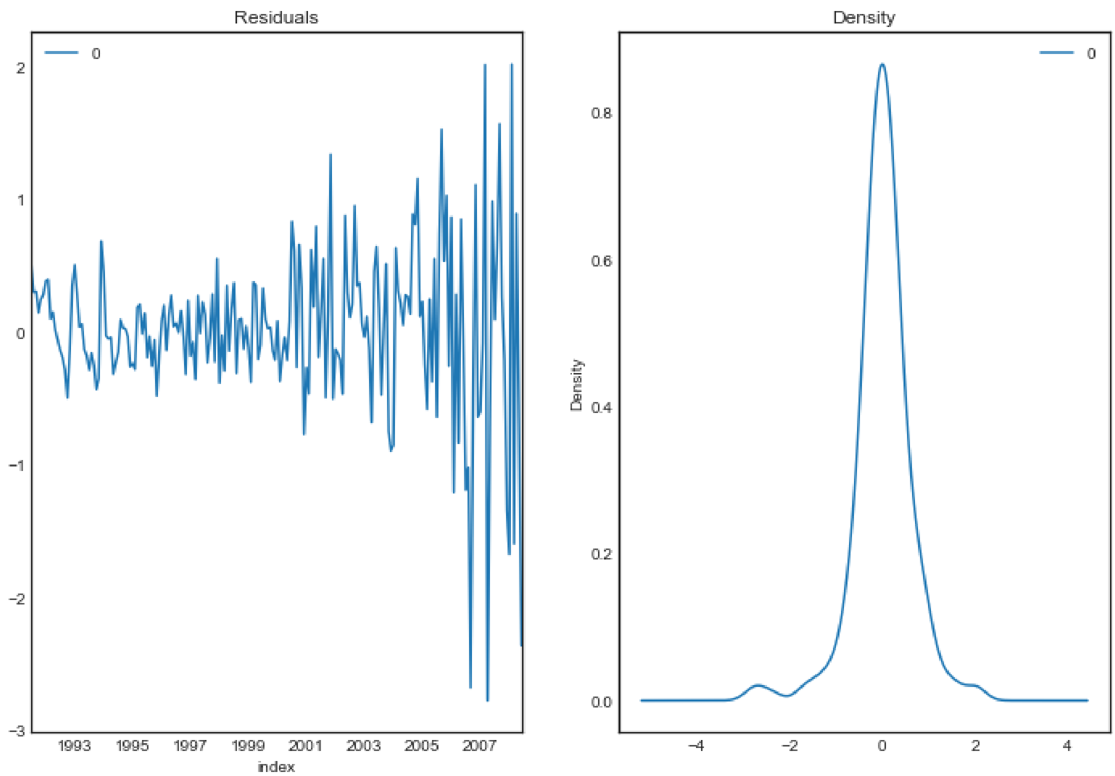
8.0.13 Fitting an ARMA Model

[20]:
```
# Fit and ARMA(p,q) model to the cycles
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA


# fit model
model = ARIMA(cycles, order=(4,0,3))
model_fit = model.fit()
print(model_fit.summary())

# plot residual erros
```

```python
residuals = pd.DataFrame(model_fit.resid)
#residuals.plot()
#residuals.plot(kind='kde')
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
print(residuals.describe())
```

```
C:\Users\tanne\anaconda3\lib\site-packages\statsmodels\base\model.py:604:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "

                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  204
Model:                 ARIMA(4, 0, 3)   Log Likelihood                -192.840
Date:                Mon, 06 Dec 2021   AIC                            403.679
Time:                        17:07:08   BIC                            433.542
Sample:                    07-01-1991   HQIC                           415.760
                         - 06-01-2008
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0006      0.003     -0.190      0.850      -0.007       0.005
ar.L1         -0.3195      0.085     -3.778      0.000      -0.485      -0.154
ar.L2         -0.1134      0.074     -1.523      0.128      -0.259       0.033
ar.L3          0.6178      0.092      6.728      0.000       0.438       0.798
ar.L4         -0.0918      0.070     -1.315      0.188      -0.229       0.045
ma.L1         -0.1808      0.104     -1.746      0.081      -0.384       0.022
ma.L2         -0.0728      0.086     -0.843      0.399      -0.242       0.096
ma.L3         -0.7284      0.107     -6.802      0.000      -0.938      -0.519
sigma2         0.3801      0.025     15.123      0.000       0.331       0.429
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
220.20
Prob(Q):                              0.98   Prob(JB):
0.00
Heteroskedasticity (H):              15.89   Skew:
-0.79
Prob(H) (two-sided):                  0.00   Kurtosis:
7.84
================================================================================
===
```

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



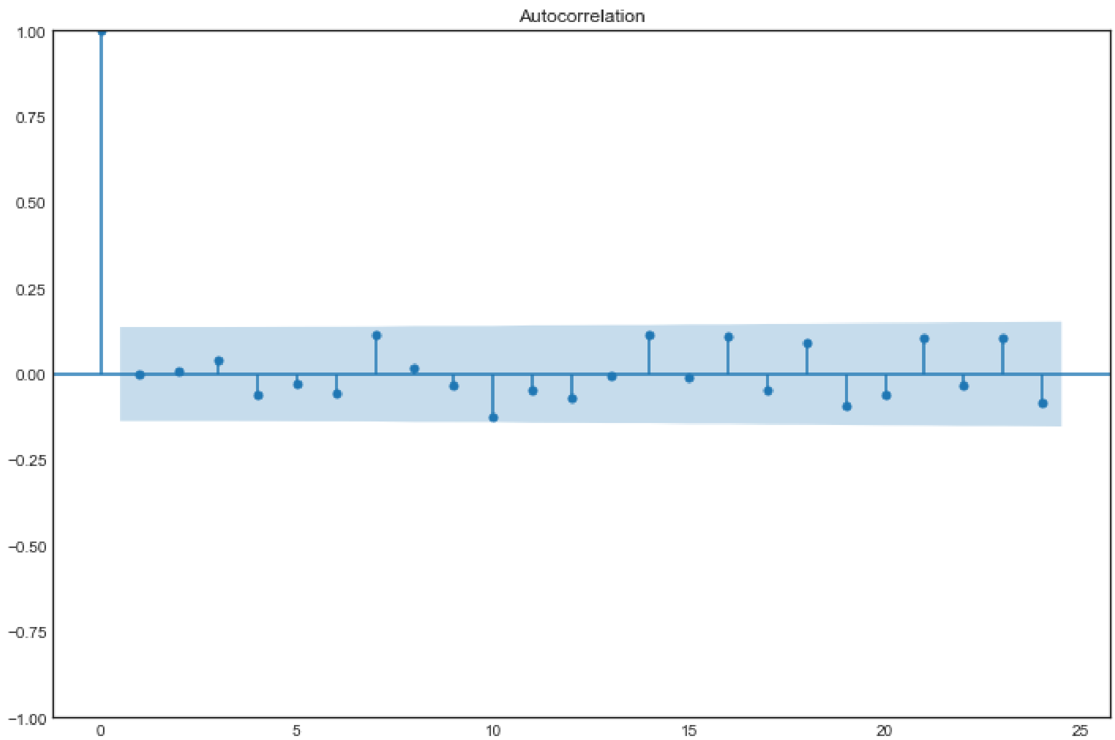

```
                0
count  204.000000
mean     0.013189
std      0.619696
min     -2.778334
25%     -0.234164
50%      0.031481
75%      0.284163
max      2.015127
```

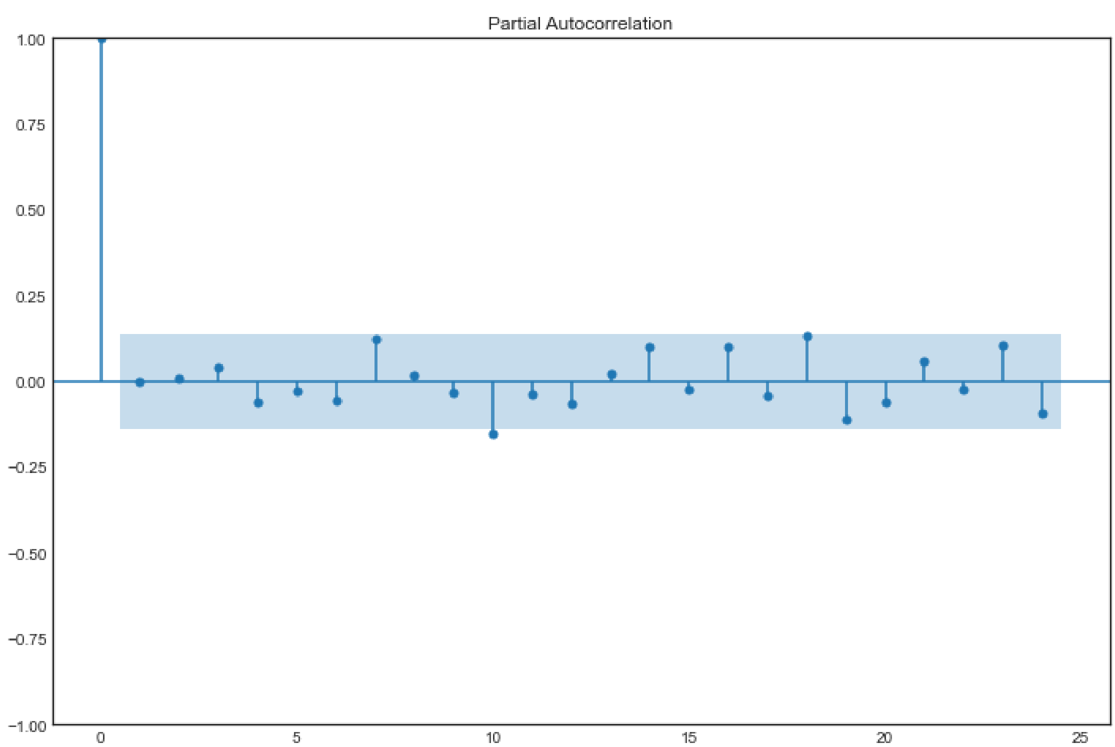
[21]:

```python
# Verify that all the dynamics have been accounted for
# We should now get white noise for the ACF and PACF
plot_acf(residuals);
plot_pacf(residuals);
```

```
C:\Users\tanne\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
```

```
warnings.warn(
```



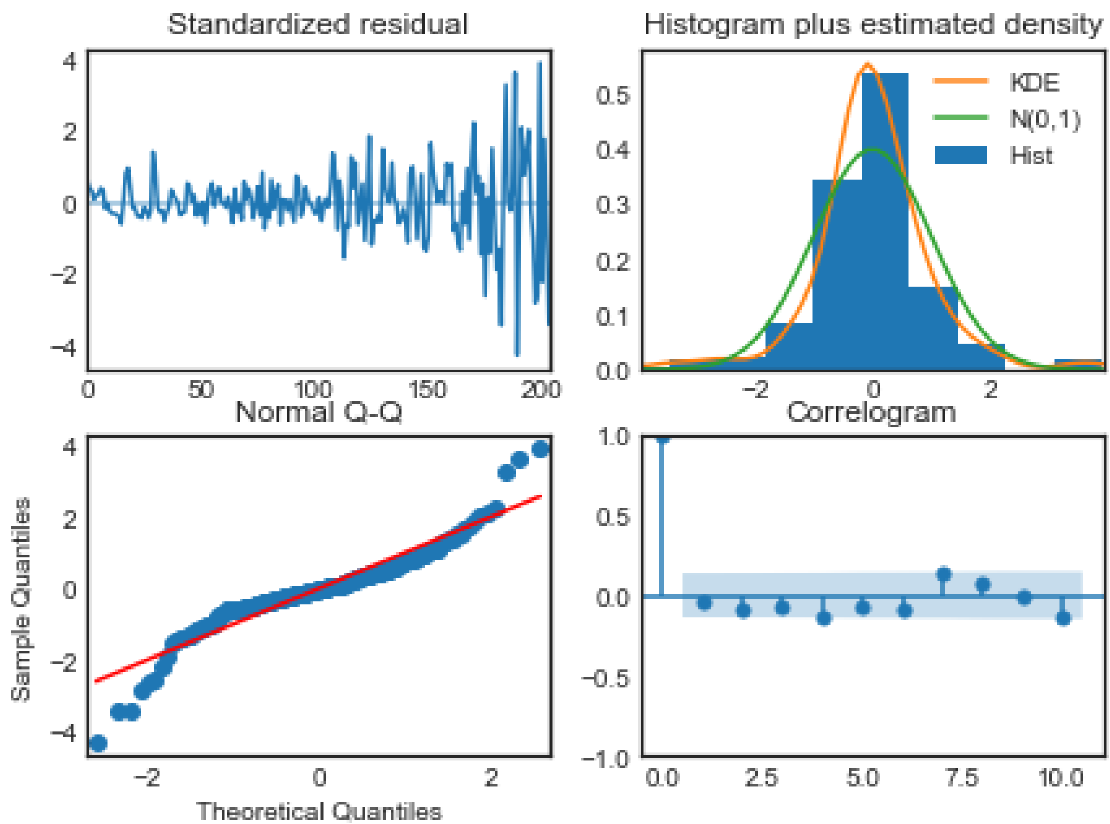


8.0.14 Fitting an Auto-ARIMA Model

[22]:
```python
from statsmodels.tsa.arima_model import ARIMA
import pmdarima as pm
model = pm.auto_arima(cycles, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=5, max_q=5, # maximum p and q
                      m=12,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)

print(model.summary())
model.plot_diagnostics(figsize=(7,5))
plt.show()
```

```
C:\Users\tanne\anaconda3\lib\site-packages\pmdarima\arima\_validation.py:62:
UserWarning: m (12) set for non-seasonal fit. Setting to 0
  warnings.warn("m (%i) set for non-seasonal fit. Setting to 0" % m)

Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=inf, Time=0.12 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=461.030, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=438.011, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=434.030, Time=0.01 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=435.762, Time=0.03 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=432.689, Time=0.06 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=420.580, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=425.741, Time=0.08 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=inf, Time=0.26 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=420.772, Time=0.18 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=434.287, Time=0.07 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=425.764, Time=0.07 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=inf, Time=0.31 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=422.526, Time=0.15 sec

Best model:  ARIMA(2,0,2)(0,0,0)[0]
Total fit time: 1.459 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  204
```

```
Model:               SARIMAX(2, 0, 2)   Log Likelihood                -205.290
Date:                Mon, 06 Dec 2021   AIC                            420.580
Time:                        17:07:46   BIC                            437.171
Sample:                             0   HQIC                           427.292
                                - 204
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.1252      0.052    -21.671      0.000      -1.227      -1.023
ar.L2         -0.8968      0.040    -22.490      0.000      -0.975      -0.819
ma.L1          0.8100      0.067     12.023      0.000       0.678       0.942
ma.L2          0.6838      0.068     10.044      0.000       0.550       0.817
sigma2         0.4356      0.026     16.902      0.000       0.385       0.486
================================================================================
===
Ljung-Box (L1) (Q):                   0.32   Jarque-Bera (JB):
156.72
Prob(Q):                              0.57   Prob(JB):
0.00
Heteroskedasticity (H):              15.59   Skew:
-0.18
Prob(H) (two-sided):                  0.00   Kurtosis:
7.28
================================================================================
===

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

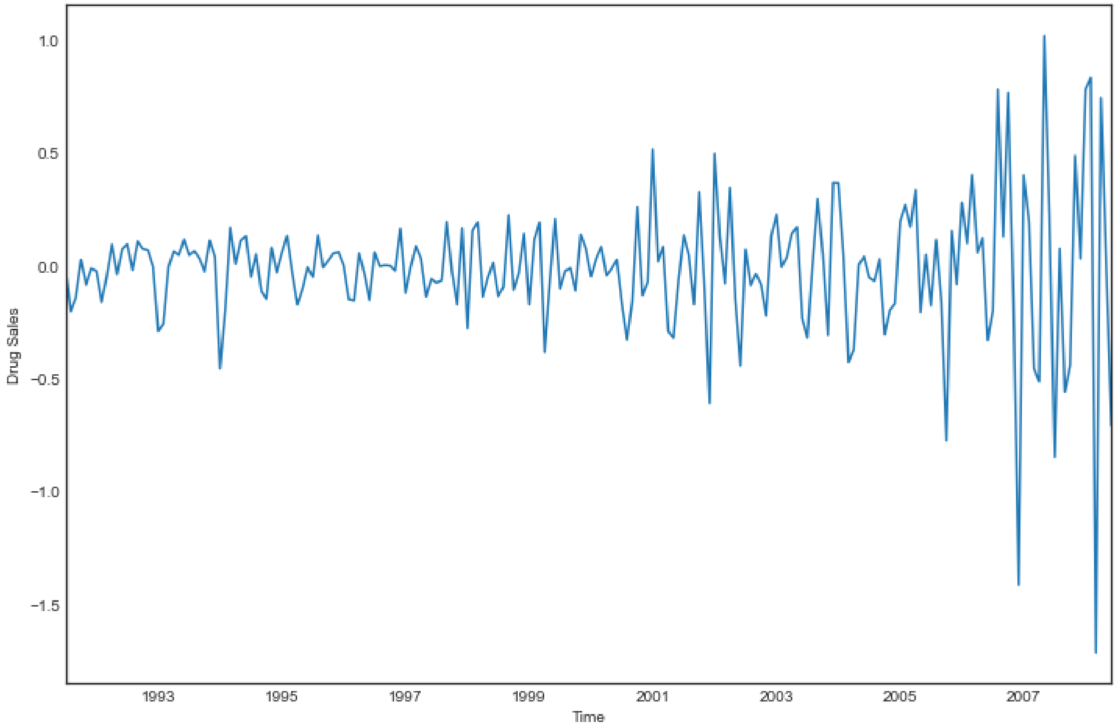


8.0.15 Fitting a Manual ARIMA Model

[23]:
```python
from statsmodels.tsa.arima.model import ARIMA
# Best Fit ARIMA Model = (3, 0, 0)
model = ARIMA(cycles, order=(3,0,0))
model_fit = model.fit()
print(model_fit.summary())

# Plot Actual vs Fitted
#model_fit.plot_predict(dynamic=False)
model_fit.predict().plot()
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  204
Model:                 ARIMA(3, 0, 0)   Log Likelihood                -211.833
Date:                Mon, 06 Dec 2021   AIC                            433.665
Time:                        17:08:22   BIC                            450.256
```

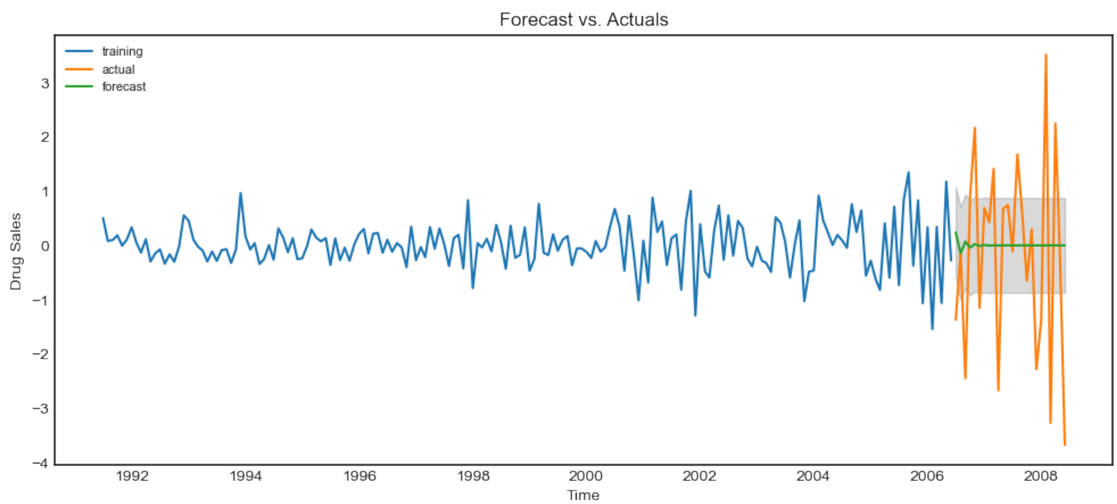
```
Sample:                       07-01-1991   HQIC                           440.376
                            - 06-01-2008
Covariance Type:                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0093      0.040     -0.230      0.818      -0.089       0.070
ar.L1         -0.4103      0.044     -9.306      0.000      -0.497      -0.324
ar.L2         -0.0984      0.045     -2.182      0.029      -0.187      -0.010
ar.L3          0.1736      0.050      3.463      0.001       0.075       0.272
sigma2         0.4664      0.029     16.108      0.000       0.410       0.523
================================================================================
===
Ljung-Box (L1) (Q):                   0.46   Jarque-Bera (JB):
187.34
Prob(Q):                              0.50   Prob(JB):
0.00
Heteroskedasticity (H):              14.25   Skew:
-0.47
Prob(H) (two-sided):                  0.00   Kurtosis:
7.60
================================================================================
===

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



8.0.16 Comparing Model Forecast Against Actual Values

[28]:
```
# Model Evaluation using
# Out-of-Time Cross validation

from statsmodels.tsa.stattools import acf

# Create Training and Test
train = cycles[:180]
test = cycles[180:]

# Build Model
model = ARIMA(train, order=(3,0,0))
fitted = model.fit()

# Forecast
#fc, se,  conf = fitted.forecast(24, alpha=0.05)  # 95% conf
fc = fitted.forecast(24, alpha=0.05)

# Make as pandas series
fc_series = pd.Series(fc, index=test.index)

lower_series = fitted.get_forecast(24).conf_int()["lower y"]
```

```
upper_series = fitted.get_forecast(24).conf_int()["upper y"]

# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k',␣
 ↪alpha=.15)
plt.title('Forecast vs. Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```



8.0.17 Model Forecast of Original Series

[46]:
```
# Seasonal - fit stepwise auto-ARIMA
smodel = pm.auto_arima(data, start_p=1, start_q=1,
                         test='adf',
                         max_p=3, max_q=3, m=12,
                         start_P=0, seasonal=True,
                         d=None, D=1, trace=True,
                         error_action='ignore',
                         suppress_warnings=True,
                         stepwise=True)


smodel.summary()
```

```
Performing stepwise search to minimize aic
```

```
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=534.818, Time=0.49 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=624.061, Time=0.01 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=596.068, Time=0.15 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=611.475, Time=0.13 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=757.274, Time=0.01 sec
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=559.407, Time=0.13 sec
 ARIMA(1,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=0.67 sec
 ARIMA(1,0,1)(0,1,2)[12] intercept   : AIC=536.817, Time=0.95 sec
 ARIMA(1,0,1)(1,1,0)[12] intercept   : AIC=543.106, Time=0.42 sec
 ARIMA(1,0,1)(1,1,2)[12] intercept   : AIC=537.834, Time=1.29 sec
 ARIMA(1,0,0)(0,1,1)[12] intercept   : AIC=594.467, Time=0.11 sec
 ARIMA(2,0,1)(0,1,1)[12] intercept   : AIC=529.829, Time=0.47 sec
 ARIMA(2,0,1)(0,1,0)[12] intercept   : AIC=555.198, Time=0.16 sec
 ARIMA(2,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=0.80 sec
 ARIMA(2,0,1)(0,1,2)[12] intercept   : AIC=531.168, Time=0.84 sec
 ARIMA(2,0,1)(1,1,0)[12] intercept   : AIC=534.757, Time=0.41 sec
 ARIMA(2,0,1)(1,1,2)[12] intercept   : AIC=524.211, Time=1.59 sec
 ARIMA(2,0,1)(2,1,2)[12] intercept   : AIC=533.034, Time=1.45 sec
 ARIMA(2,0,1)(2,1,1)[12] intercept   : AIC=531.779, Time=1.69 sec
 ARIMA(2,0,0)(1,1,2)[12] intercept   : AIC=545.610, Time=1.18 sec
 ARIMA(3,0,1)(1,1,2)[12] intercept   : AIC=inf, Time=1.69 sec
 ARIMA(2,0,2)(1,1,2)[12] intercept   : AIC=529.398, Time=1.89 sec
 ARIMA(1,0,0)(1,1,2)[12] intercept   : AIC=593.876, Time=0.74 sec
 ARIMA(1,0,2)(1,1,2)[12] intercept   : AIC=523.971, Time=1.61 sec
 ARIMA(1,0,2)(0,1,2)[12] intercept   : AIC=532.804, Time=0.84 sec
 ARIMA(1,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=0.80 sec
 ARIMA(1,0,2)(2,1,2)[12] intercept   : AIC=534.472, Time=1.42 sec
 ARIMA(1,0,2)(0,1,1)[12] intercept   : AIC=531.170, Time=0.46 sec
 ARIMA(1,0,2)(2,1,1)[12] intercept   : AIC=532.815, Time=1.38 sec
 ARIMA(0,0,2)(1,1,2)[12] intercept   : AIC=583.617, Time=0.81 sec
 ARIMA(1,0,3)(1,1,2)[12] intercept   : AIC=inf, Time=1.54 sec
 ARIMA(0,0,1)(1,1,2)[12] intercept   : AIC=614.924, Time=0.66 sec
 ARIMA(0,0,3)(1,1,2)[12] intercept   : AIC=566.118, Time=1.03 sec
 ARIMA(2,0,3)(1,1,2)[12] intercept   : AIC=527.229, Time=1.73 sec
 ARIMA(1,0,2)(1,1,2)[12]             : AIC=524.136, Time=1.08 sec

Best model:  ARIMA(1,0,2)(1,1,2)[12] intercept
Total fit time: 30.646 seconds
```

[46]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
================================================================================
==========
Dep. Variable:                                    y   No. Observations:
204
Model:             SARIMAX(1, 0, 2)x(1, 1, 2, 12)   Log Likelihood
```

```
-253.986
Date:                            Mon, 06 Dec 2021   AIC
523.971
Time:                                    18:02:59   BIC
550.031
Sample:                                         0   HQIC
534.526
                                             - 204
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0053      0.009      0.567      0.571      -0.013       0.024
ar.L1          0.9761      0.019     51.367      0.000       0.939       1.013
ma.L1         -0.9023      0.057    -15.705      0.000      -1.015      -0.790
ma.L2          0.2137      0.059      3.625      0.000       0.098       0.329
ar.S.L12       0.8434      0.165      5.125      0.000       0.521       1.166
ma.S.L12      -1.5660      0.184     -8.517      0.000      -1.926      -1.206
ma.S.L24       0.7452      0.112      6.659      0.000       0.526       0.965
sigma2         0.7648      0.066     11.544      0.000       0.635       0.895
================================================================================
===
Ljung-Box (L1) (Q):                   0.34   Jarque-Bera (JB):
143.81
Prob(Q):                              0.56   Prob(JB):
0.00
Heteroskedasticity (H):              14.40   Skew:
0.28
Prob(H) (two-sided):                  0.00   Kurtosis:
7.20
================================================================================
===

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

[47]:

```
# Forecast
n_periods = 24
fitted, confint = smodel.predict(n_periods=n_periods, return_conf_int=True)
index_of_fc = pd.date_range(data.index[-1], periods = n_periods, freq='MS')

# make series for plotting purpose
fitted_series = pd.Series(fitted, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)
```

```
# Plot
plt.plot(data)
plt.plot(fitted_series, color='darkgreen')
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k', alpha=.15)

plt.title("SARIMA - Final Forecast of a10 - Drug Sales")
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```