BI Norwegian Business School - campus Oslo

EBA 35301

Causality, Machine learning and Forecasting

Course paper

ID number: 0998783, 1029906, 1041400

Start: 08.03.2021 09.00

Finish: 17.03.2021 12.00

In [293…
```
#Import python modules

import pandas as pd
import numpy as np
import scipy as sp
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from matplotlib import pyplot
from functools import reduce
from IPython.display import Image
import statsmodels.api as sm
import sklearn as sk
from sklearn import linear_model
from sklearn.metrics import r2_score


plt.rcParams ["figure.figsize"] = [10,8]
```

Assignment A: Regression analysis and Monte Carlo

Load the data

In [294…
```
data = pd.read_csv('randomX.csv')
```

In [295…
```
# Quick view about the data set
data.head()
```

Out[295…

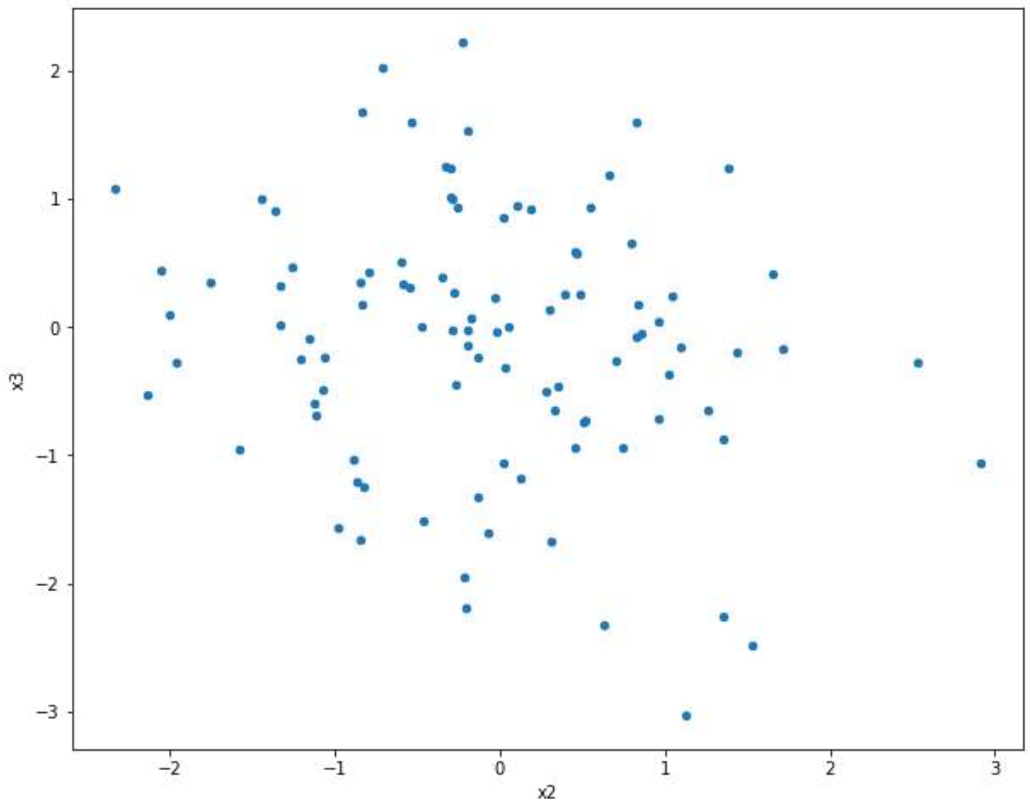
	x1	x2	x3
0	0.537667	0.840376	0.183227
1	1.833885	-0.888032	-1.029768
2	-2.258847	0.100093	0.949222
3	0.862173	-0.544529	0.307062
4	0.318765	0.303521	0.135175

In [296…
```
data.shape
```

Out[296… (100, 3)

In [297…
```
x1 = data['x1']
x2 = data['x2']
x3 = data['x3']
```

In [300…
```
data.plot(kind='scatter', x = 'x2', y = 'x3')
plt.show()
```



In [301…
```
# Checking the correlation (non significant) between the x-values

data.corr()
```

Out[301…

	x1	x2	x3
x1	1.000000	0.075423	0.131700
x2	0.075423	1.000000	-0.182852
x3	0.131700	-0.182852	1.000000

1a.) Plot a histogram for each of the estimated parameters as well as the R- squared. Into your histograms, also plot the corresponding median (of the respective parameter estimate/R-squared).

In [303…
```
# We now estimate on the model provided with multiple x-variables

x_all = data[['x1', 'x2', 'x3']]
```

In [304…
```
# We know the coefficients for the intercept and slope for x1
b0 = 0.5
b1 = 1.5

# We know that the error term has a mean of 0 and variance of 1
eps = np.random.normal(0,1, size = 100)

# Simulations
sims = 2000

y_all = []

beta_hat_vec = []
```

In [305…
```
# y equation
y11 = b0 + b1*x1 + x2 + x3 + (x3)**2 + eps


# The mean value of y11
mean = (y11.sum()) / 100
print('Mean value: ', mean)
```

```
Mean value:  1.4562302221518015
```

In [306…
```
# The mean value of x1, x2, and x3

x1_sum = (x1.sum())
x1_avg = x1_sum / 100
print('Mean of x1: ', x1_avg)

x2_sum = (x2.sum())
x2_avg = x2_sum / 100
print('Mean of x2: ', x2_avg)

x3_sum = (x3.sum())
x3_avg = x3_sum / 100
print('Mean of x2: ', x2_avg)
```

```
Mean of x1:  0.12308533750100568
Mean of x2:  -0.07268145828620218
Mean of x2:  -0.07268145828620218
```

In [307…
```
x_all.head(3)
```

Out[307…

	x1	x2	x3
0	0.537667	0.840376	0.183227
1	1.833885	-0.888032	-1.029768
2	-2.258847	0.100093	0.949222

In [308…
```
#Monte Carlo Simulations where sims = 2000
for i in range(1,sims):
```

```
# x_all

y2 = b0 + b1*x1 + x2 + x3 + (x3)**2 + eps

X2 = sm.add_constant(x_all)
model = sm.OLS(y2,X2)
results2 = model.fit()


y_all.append(y2)
```

In [309…

```
results2.summary()
```

Out[309…

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.640
Model:	OLS	**Adj. R-squared:**	0.629
Method:	Least Squares	**F-statistic:**	56.85
Date:	Wed, 17 Mar 2021	**Prob (F-statistic):**	3.27e-21
Time:	01:28:53	**Log-Likelihood:**	-186.38
No. Observations:	100	**AIC:**	380.8
Df Residuals:	96	**BIC:**	391.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.4316	0.162	8.823	0.000	1.110	1.754
x1	1.2808	0.140	9.173	0.000	1.004	1.558
x2	1.3240	0.163	8.132	0.000	1.001	1.647
x3	0.3323	0.162	2.052	0.043	0.011	0.654

Omnibus:	33.254	**Durbin-Watson:**	2.111
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	67.297
Skew:	1.306	**Prob(JB):**	2.44e-15
Kurtosis:	6.054	**Cond. No.**	1.41

Notes:

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

In [310…
```
results2.params
```

Out[310…
```
const    1.431630
x1       1.280797
x2       1.324031
x3       0.332284
dtype: float64
```

In [311…
```
#Estimated Beta coefficients of x1, x2, x3 from simulations

x1_corr = results2.params.x1
print('est_Beta 1 coefficient:', x1_corr)

x2_corr = results2.params.x2
print('est_Beta 2 coefficient:', x2_corr)

x3_corr = results2.params.x3
print('est_Beta 3 coefficient:', x3_corr)
```

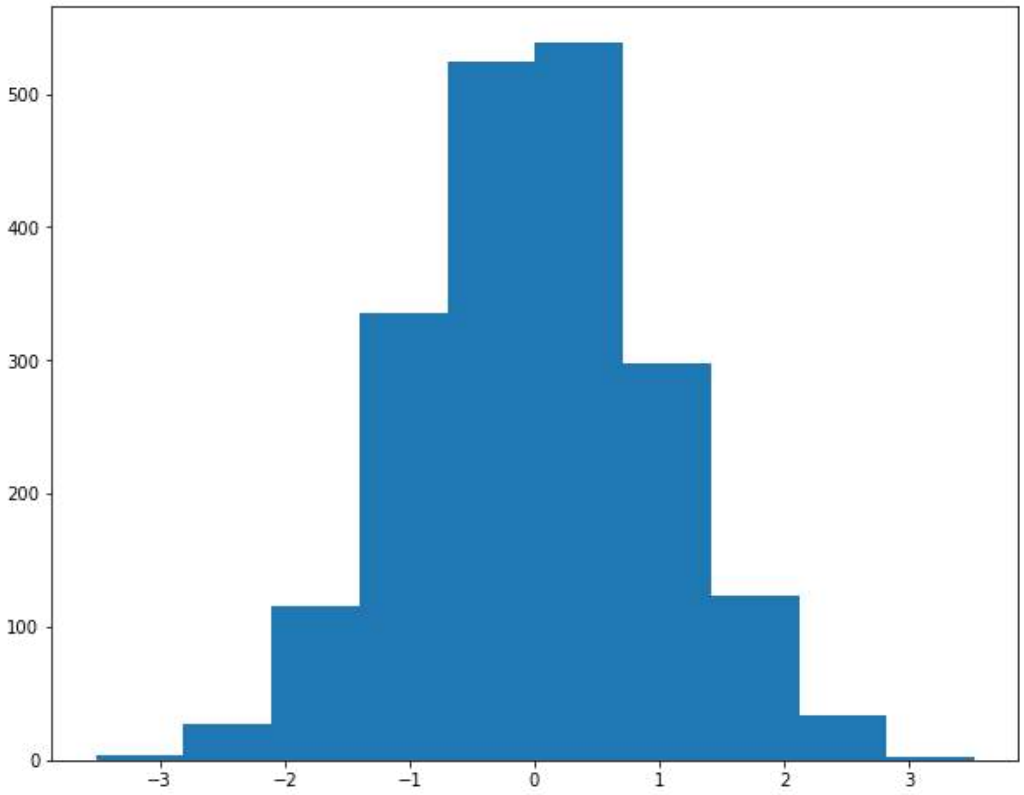
```
est_Beta 1 coefficient: 1.2807965218235866
est_Beta 2 coefficient: 1.3240307908306
est_Beta 3 coefficient: 0.3322835958738732
```

In [312…
```
# R-squared
r2 = results2.rsquared
print('R^2: ', r2)
```

```
R^2:  0.6398363437333559
```

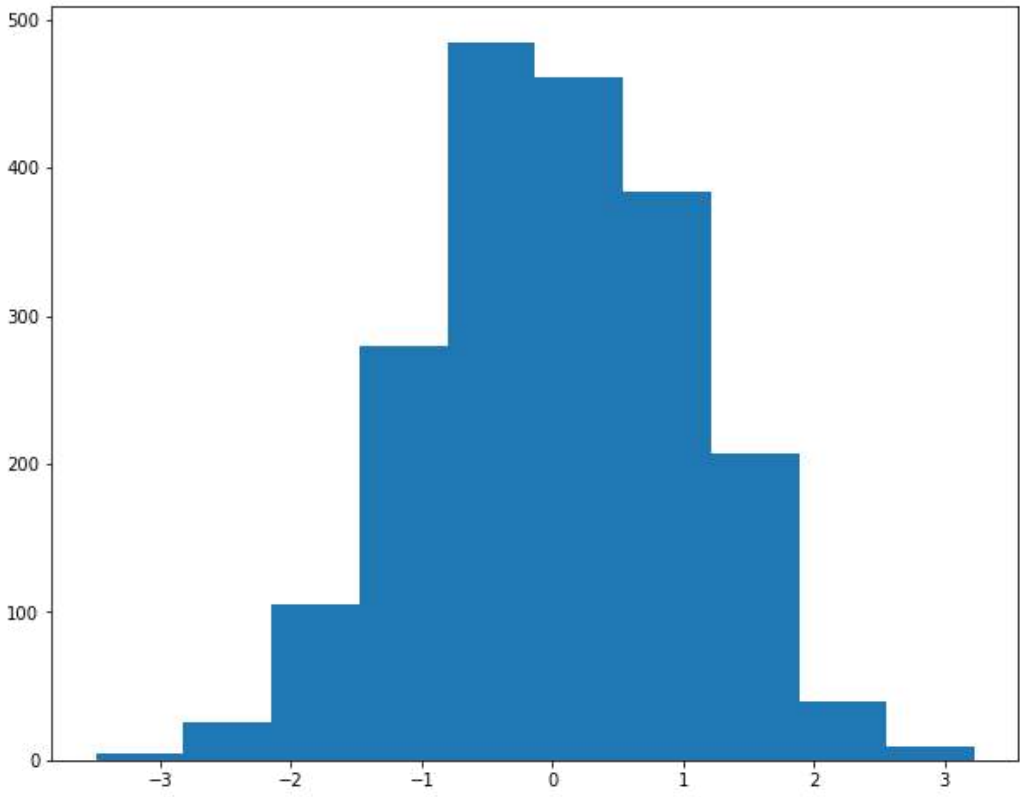
In [313…
```
# Histogram of the beta 1 coefficient

x1_hist = x1_corr
x1_hist = np.random.normal(size = 2000)

plt.figure()
plt.hist(x1_hist)
```

Out[313…
```
(array([  3.,  27., 115., 335., 524., 539., 298., 124.,  33.,
2.]),
 array([-3.51323567, -2.80968032, -2.10612497, -1.40256962, -0.69
901427,
         0.00454108,  0.70809643,  1.41165178,  2.11520713,  2.81
876248,
         3.52231783]),
 <BarContainer object of 10 artists>)
```



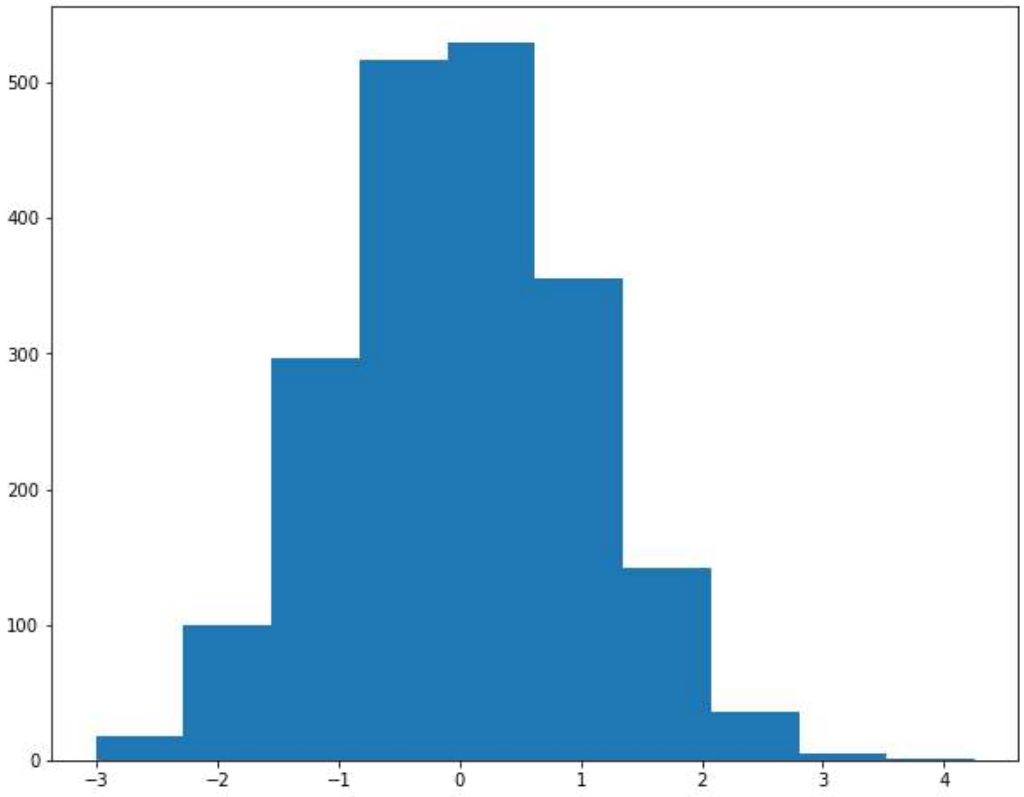
In [314…
```
# Histogram of the beta 2 coefficient
x2_hist = x2_corr
x2_hist = np.random.normal(size=2000)

plt.figure()
plt.hist(x2_hist)
```

Out[314…
```
(array([  4.,  25., 105., 280., 485., 461., 384., 207.,  40.,
9.]),
 array([-3.49281121, -2.8209501 , -2.149089  , -1.47722789, -0.80
536679,
       -0.13350569,  0.53835542,  1.21021652,  1.88207763,  2.55
393873,
        3.22579984]),
 <BarContainer object of 10 artists>)
```



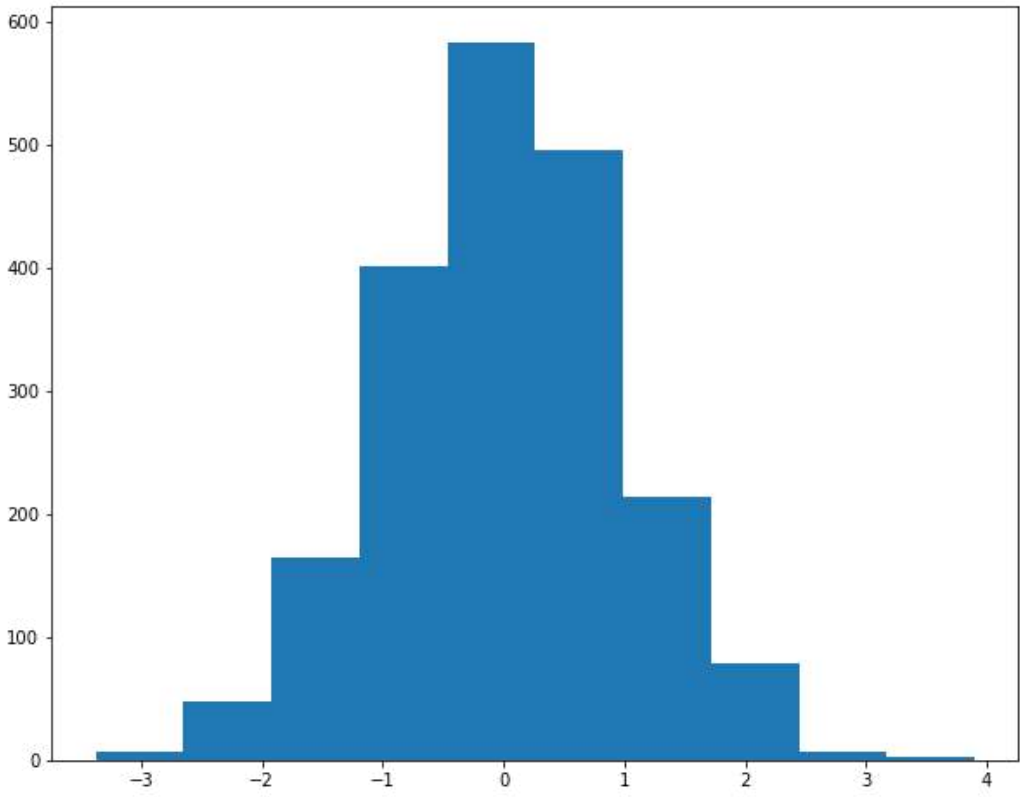
In [315…
```
# Histogram of the beta 3 coefficient

x3_hist = x3_corr
x3_hist = np.random.normal(size=2000)

plt.figure()
plt.hist(x3_hist)
```

Out[315…
```
(array([ 18., 100., 296., 517., 530., 355., 142.,  36.,   5.,
1.]),
 array([-3.00839642, -2.28229447, -1.55619252, -0.83009057, -0.10
398862,
         0.62211333,  1.34821528,  2.07431723,  2.80041918,  3.52
652113,
         4.25262309]),
 <BarContainer object of 10 artists>)
```



In [316…
```
# Plot the corresponding median (of the respective parameter esti

r2 = np.random.normal(size=2000)

plt.figure()
plt.hist(r2)
```

Out[316…
```
(array([  7.,  48., 164., 401., 583., 495., 213.,  79.,   7.,
3.]),
 array([-3.37429874, -2.64719999, -1.92010125, -1.1930025 , -0.46
590375,
         0.26119499,  0.98829374,  1.71539249,  2.44249123,  3.16
958998,
         3.89668873]),
 <BarContainer object of 10 artists>)
```



Comments about the result:

From the graph above, we can say that most of the result are centered around 0.

2a.) Construct 95% confidence intervals for each parameter based on the simulations. Which of the coefficients are significant?

The formula for CI is the the following

In [317…
```
# compute std-error for x1
x1_se = results2.bse.x1

# confidence interval for beta 1 coefficient (x1_coeff)

x1_upper = x1_corr + stats.norm.ppf(0.975)*x1_se
x1_lower = x1_corr - stats.norm.ppf(0.975)*x1_se

#Beta 1 CI
CI_1 = print('Lower Limit:',  x1_lower , ';', 'Upper Limit:', x1_
```

```
Lower Limit: 1.0071408273385347 ; Upper Limit: 1.5544522163086385
```

In [318…
```
# compute std-error for x2
x2_se = results2.bse.x2
```

```
# confidence interval for beta 2 coefficient (x2_coeff)

x2_upper = x2_corr + stats.norm.ppf(0.975)*x2_se
x2_lower = x2_corr - stats.norm.ppf(0.975)*x2_se

#Beta 2 CI
CI_2 = print('Lower Limit:',  x2_lower , ';', 'Upper Limit:', x2_
```

```
Lower Limit: 1.0049089999437606 ; Upper Limit: 1.6431525817174393
```

In [319…

```
# compute std-error for x3
x3_se = results2.bse.x3

# compute confidence intervals for beta 3 coefficient

x3_upper = x3_corr + stats.norm.ppf(0.975)*x3_se
x3_lower = x3_corr - stats.norm.ppf(0.975)*x3_se

#Beta 3 CI
CI_3 = print('Lower Limit:',  x3_lower , ';', 'Upper Limit:', x3_
```

```
Lower Limit: 0.01483236142515304 ; Upper Limit: 0.649734830322593
4
```

In [320…

```
beta1_CI = print('Beta 1 95% CI:', x1_upper, '<', 'est_beta_1', '
beta2_CI = print('Beta 2 95% CI:', x2_upper, '<', 'est_beta_2', '
beta3_CI = print('Beta 3 95% CI:', x3_upper, '<', 'est_beta_3', '
```

```
Beta 1 95% CI: 1.5544522163086385 < est_beta_1 < 1.00714082733853
47
Beta 2 95% CI: 1.6431525817174393 < est_beta_2 < 1.00490899994376
06
Beta 3 95% CI: 0.6497348303225934 < est_beta_3 < 0.01483236142515
304
```

In [321…

```
# The difference between the upper and lower limit of estimated B
print('est_beta_1 difference:', x1_upper - x1_lower)

# The difference between the upper and lower limit of estimated B
print('est_beta_2 difference:', x2_upper - x2_lower)

# The difference between the upper and lower limit of estimated B
print('est_beta_3 difference:', x3_upper - x3_lower)
```

```
est_beta_1 difference: 0.5473113889701038
est_beta_2 difference: 0.6382435817736787
est_beta_3 difference: 0.6349024688974403
```

Answer:

To answer the question: Which of the coefficients are significant, we calculated the difference of the corresponding confidence interval for every beta. From above, we can therefore conclude that beta_1 is the most significant coeffiencient among the 3 as it has the lowest difference between

the upper and lower limit. I.e. the estimated beta_1 is close to the true beta_1 value.

3a.) Redo the Monte Carlo experiment, but now estimate the true model structure for each simulation. Make a histogram of the β1 estimate. How does it compare to the one you produced in A.i? Do you find a difference? Provide an explanation why/why not.

In [322…
```
# We know the coefficients for the intercept and slope for x1
b0 = 0.5
b1 = 1.5

# We know that the error term has a mean of 0 and variance of 1
eps = np.random.normal(0,1, size = 100)

# Simulations
sims = 2000

y1_est = []
```

In [323…
```
for i in range(1,sims):

    y1 = b0 + b1*x1 + eps

    X2 = sm.add_constant(x1)
    model = sm.OLS(y2,X2)
    results1 = model.fit()


    y1_est.append(y1)
```

In [324…
```
results1.summary()
```

Out[324…

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.391
Model:	OLS	**Adj. R-squared:**	0.385
Method:	Least Squares	**F-statistic:**	62.89
Date:	Wed, 17 Mar 2021	**Prob (F-statistic):**	3.57e-12
Time:	01:32:21	**Log-Likelihood:**	-212.65
No. Observations:	100	**AIC:**	429.3
Df Residuals:	98	**BIC:**	434.5
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.2832	0.206	6.226	0.000	0.874	1.692
x1	1.4054	0.177	7.931	0.000	1.054	1.757

Omnibus:	10.682	**Durbin-Watson:**	2.141
Prob(Omnibus):	0.005	**Jarque-Bera (JB):**	11.684
Skew:	0.637	**Prob(JB):**	0.00290
Kurtosis:	4.087	**Cond. No.**	1.20

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

In [325…
```
results1.params
```

Out[325…
```
const    1.283246
x1       1.405403
dtype: float64
```

In [326…
```
# estimated Beta 1 coefficient of x1 from DGP function

x11_corr = results1.params.x1

print('estimated beta 1 coefficient (DGP function):', x11_corr)
```

```
estimated beta 1 coefficient (DGP function): 1.405403181856381
```

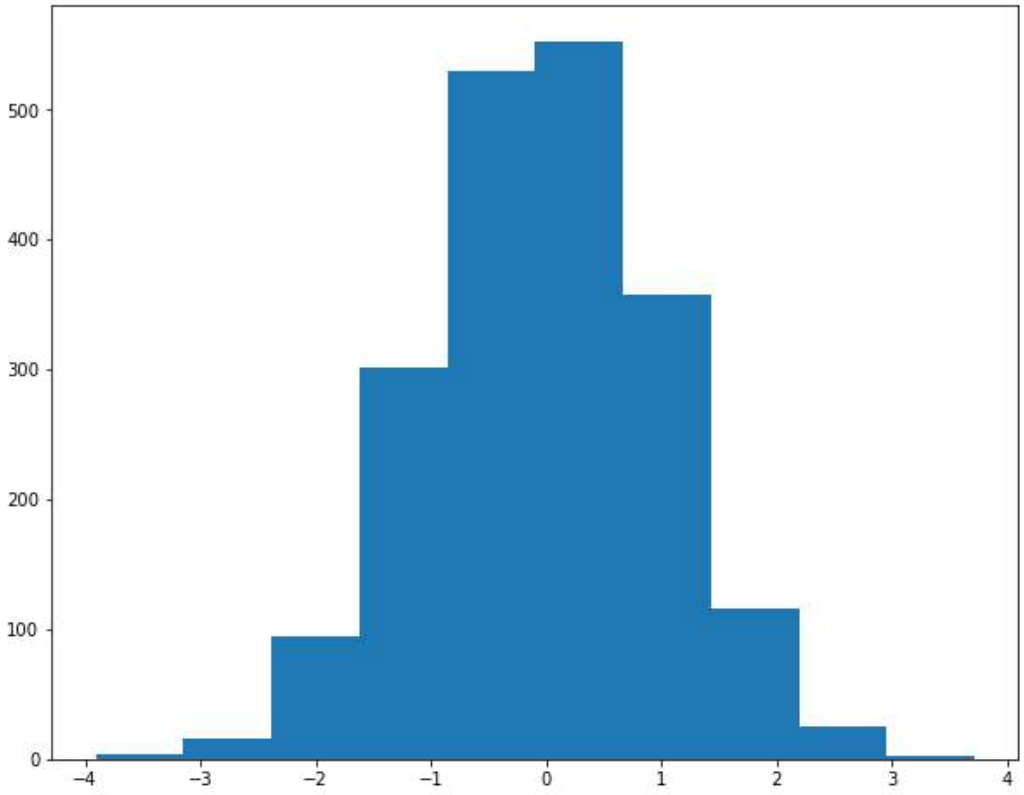
In [327…
```
# R-squared of DGP function

r22 = results1.rsquared

print('R^2 of DGP function:', r22)
```

```
R^2 of DGP function: 0.3909017532126836
```

In [328…
```
x11_corr = np.random.normal(size = 2000)

plt.figure()
plt.hist(x11_corr)
```

Out[328…
```
(array([  4.,  16.,  94., 302., 530., 553., 358., 116.,  25.,
2.]),
 array([-3.91060964, -3.14782   , -2.38503035, -1.6222407 , -0.85
945105,
        -0.0966614 ,  0.66612825,  1.4289179 ,  2.19170754,  2.95
449719,
```

```
        3.71728684]),
<BarContainer object of 10 artists>)
```



Answer:

From above our group found out that, there is a difference between beta 1 based on the True model (true DGP) and Beta 1 based on the other model. We notice that Beta 1 of the DGP function is approx. equal to 1.5, while on the other model has a beta 1 coefficient of 1.28. I.e. the estimated beta_1 coefficient in the DGP function is approx. equal to our true beta_1 coefficient which 1.5 than the other model.

One of the reason for this is that in the model we simulated in 1a.) we considered more predictors than the DGP function, and we know that R^2 increases when we consider more predictors. So, though the fit of the linear function with multiple predictors (the one we simulated in 1a.)) yielded a higher R-squared (0.63) than the DGP function (0.39), the DGP function gives us an unbiased estimator of beta_1 coefficient and this in accordance to Gauss-Markov Theorem which assumes that the mean value of the estimated beta_1 coefficient is the true beta_1. That is, the distribution of the estimated beta_1 is always centered at the value of the true beta_1. This is shown in the

historgram above (note that we have have standardized our variables above, thereby, affecting the scaling).

4a.) Explain in which context it would be useful to replace the OLS estimator used above with a logistic regression.

Answer:

It would be useful to replace the OLS estimator with logistic regression, if we were to solve a problem where the predictive dependent variable y, would be a categorical variable. We could in this assignment fit a linear model with OLS, since the predictive outcome (y-variable) is a continous variable. If predicting the outcome (y-variable) was to be a categorical variable, it would be a violation to fit an OLS to this model, since the assumption of a linear relationship between the predictor(s) and the outcome would not be valid.

An example of this violation would be to fit a linear model where the outcome (y-variable) represents a categorical variable such as 'Boy or Girl'. We would (or could) as a solution, use the same independent random (continous) variables from the randomX data, to predict a binary outcome, which would be a result of replacing the OLS estimator with logistic regression. Thus, transforming the linear regression model to a logistic model, to make a predection of the binary outcome variable based on the predictors. Thus, a logit transformation.

5a.) What is the interpretation of the β2 coefficient above? What is the interpretation of β3 and β4?

(Hint: $\beta 3$ and $\beta 4$ should only be interpreted together. It might help you to first derive an expression for the effect of a unit increase in $X3$ on Y.)

Answer:

We approach a value for x3 of = -0.5, after deriving an expression for the effect of a unit increase in x3 on y. Regardless of the outcome of the graph or the model, resulting in a curvilinear outcome of the model due to the squared x3 variable, the coefficients b3 and b4 will remain the same, as they are attached to the same predictor x3 and interpreted together. This is a theory based on that: "The regression coefficient Beta_i is interpreted as the expected change in Y associated with a 1-unit increase in x_i where i = 1,...,n.,

while the other predictors are held constant, with their corresponding coefficient (Beta_i)".

From the simulated coefficients of beta_3(equivalent to beta_4) and the value derived for x3: (beta_3(or/-and 4)*x3 = 0.4945* -0.5 = -0,24725 --> A negative value that would result in a negative outcome of y.

The beta_2 coefficient will remain the same, or constant. It would not be affected by other predictors, than contributing to it's simulated constant of beta_2*x2 (1.3290x2)

6a.) Write shortly about how changing the number of Monte Carlo simulations and the number of observations (N) might affect your results in these experiments.

Answer:

Increasing the number of observation in our simulation, will give us a narrow confindence interval. This is a consequence of the law of large number which tells us that as our sample size N increases, the mean gets closer to the average of the whole population. So in relation to the assignment, if we increase the simulation to 10^4, then our estimated betahat_1 is going to be closer to the true value of beta_1.

Assignment B: Time Series

1b.) Load and plot the data. By just looking at the data, what time series properties would you say are prominent for this data?

Hint: To load the data use the below code.

In [329…

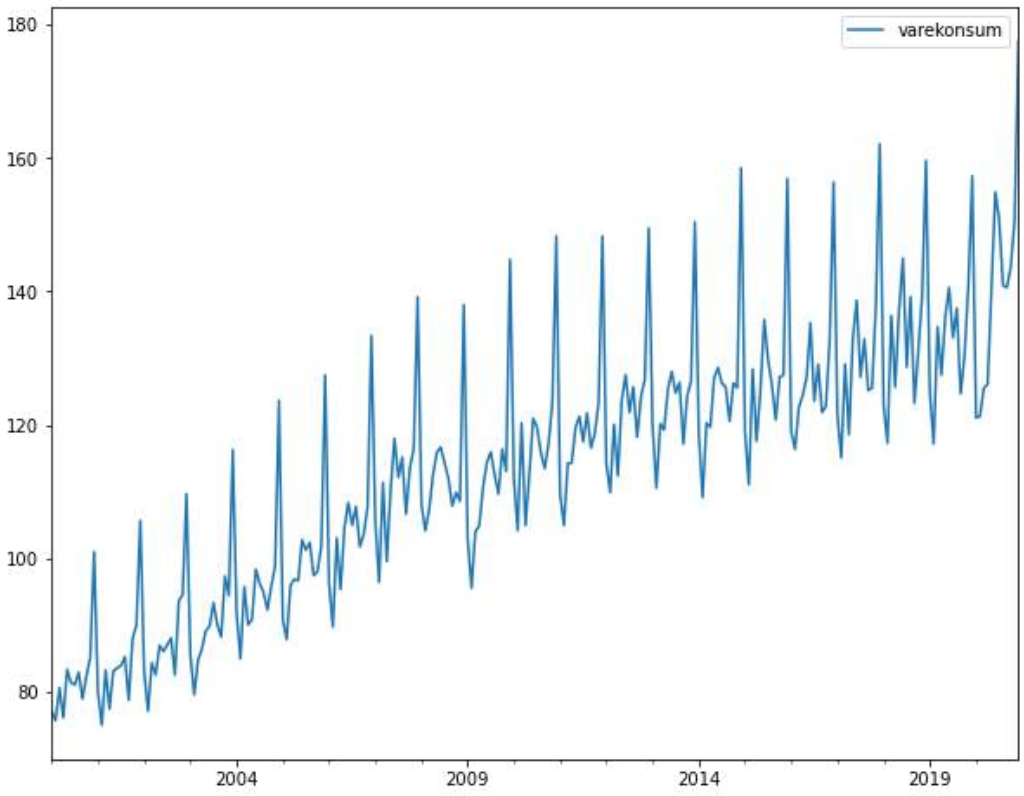
```
#Read in the data sheet
varekonsum = pd.read_excel ('varekonsum.xlsx', usecols="C:IT", he
varekonsum = varekonsum.dropna(axis=1)

#adjust column label
varekonsum.columns =['varekonsum']

#Replace the index with a machine readable format
varekonsum.index = pd.date_range(start='31/01/2000', end='31/12/2
varekonsum.head()
```

Out[329…

	varekonsum
2000-01-31	77.1
2000-02-29	75.8
2000-03-31	80.7
2000-04-30	76.2
2000-05-31	83.4

In [330…

```
#Plot the data
varekonsum.plot()
pyplot.show()
```



Answer:

It is quite clear that this timeseries is trending and has seasonal spikes. This data is not stationary, because we can see that it is not mean reverting.

These spikes tend to get high at the end of every quarter and this maybe due to household consumption increasing during the christmas season.

We can also see that it is trending upwards which can be due to increase in income. So households can afford to consume more each year.

2b.) Construct a dummy variable which has zeros for each month in the sample, but a one for each December month. Call this variable "dum".

In [331…
```
date = varekonsum.index

#add a new column where it's just the month
varekonsum['Month'] = pd.DatetimeIndex(date).month

varekonsum.head(3)
```

Out[331…

	varekonsum	Month
2000-01-31	77.1	1
2000-02-29	75.8	2
2000-03-31	80.7	3

In [332…
```
#Create dummy variable and add to the data frame
varekonsum['dum'] = (varekonsum['Month'] == 12).astype(int)
```

In [333…
```
varekonsum.head(24)
```

Out[333…

	varekonsum	Month	dum
2000-01-31	77.1	1	0
2000-02-29	75.8	2	0
2000-03-31	80.7	3	0
2000-04-30	76.2	4	0
2000-05-31	83.4	5	0
2000-06-30	81.4	6	0
2000-07-31	81.1	7	0
2000-08-31	83.0	8	0
2000-09-30	79.0	9	0
2000-10-31	82.4	10	0
2000-11-30	85.2	11	0
2000-12-31	101.0	12	1
2001-01-31	80.0	1	0
2001-02-28	75.1	2	0
2001-03-31	83.3	3	0

	varekonsum	Month	dum
2001-04-30	77.5	4	0
2001-05-31	83.2	5	0
2001-06-30	83.6	6	0
2001-07-31	84.0	7	0
2001-08-31	85.3	8	0
2001-09-30	78.8	9	0
2001-10-31	88.0	10	0
2001-11-30	90.1	11	0
2001-12-31	105.7	12	1

3b.) Run the following OLS regression:

$Yt = \beta 0 + \beta 1t + \beta 2dum + \varepsilon t$

and compute and plot $Yt - Yt = \varepsilon t$. What time series properties would you say are prominent for εt? Discuss your answer in relation to question 1b.

In [334…
```
dummy = varekonsum['dum'] == 1
dum = np.array([dummy])
```

In [335…
```
monthly = varekonsum['Month']
month = np.array([monthly])
```

Interpretation of the question:

Let y(t) be consumption at t, t be the month, and dum be 0 for other months and 1 for december and e_t be the stochastic part.

To answer this question first let's estimate beta_0, beta_1, beta_2, since y(t), t, and dum can be obtained from the data set, before we run OLS

In [336…
```
#Let x1(t) be month and x2 be dum
x = varekonsum[['Month','dum']]
#add constant, beta_0
x = sm.add_constant(x)

#Dependent variable
y = varekonsum['varekonsum']
```

In [337…
```
reg = linear_model.LinearRegression()
reg.fit(x,y)
```

Out[337… LinearRegression()

In [338…
```
reg.coef_
```

Out[338… array([0.   , 1.13393939, 23.12926407])

In [339…
```
X = reg.intercept_
print(X)
```

104.41108225108225

Since we know now the estimated coefficients, we can run the regression

In [340…
```
model = sm.OLS(y, x).fit()
predictions = model.predict(x)
model.summary()
```

Out[340…

OLS Regression Results

Dep. Variable:	varekonsum	**R-squared:**	0.202
Model:	OLS	**Adj. R-squared:**	0.195
Method:	Least Squares	**F-statistic:**	31.44
Date:	Wed, 17 Mar 2021	**Prob (F-statistic):**	6.71e-13
Time:	01:44:56	**Log-Likelihood:**	-1083.5
No. Observations:	252	**AIC:**	2173.
Df Residuals:	249	**BIC:**	2184.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	104.4111	2.531	41.260	0.000	99.427	109.395
Month	1.1339	0.373	3.039	0.003	0.399	1.869
dum	23.1293	4.660	4.963	0.000	13.951	32.308

Omnibus:	20.532	**Durbin-Watson:**	0.141
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	9.266
Skew:	-0.248	**Prob(JB):**	0.00972
Kurtosis:	2.202	**Cond. No.**	31.2

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

Answer:

Let Consumption_t = B_0 + B_1_t + B_2dum + E_t where t is month and dum = 0 if not december or 1 if december

Running the OLS using sklean where x1= month and x2= dum and y = consumption we determine the following beta coefficients from the result summary. These are the following beta coefficients:

- B_0 = 104.41
- B_1 = 1.1339 ≈ 1
- B_2= 23.1292

From this, we know from theory that if B_1 = 1, then we have a random walk, that is to predict the consumption at t, is to know what was the consumption at t-1 in relation to this assignment. This can be mathemitcally expressed as Yt = Y_t-1 + E_t. This observation is relevant for the remaining questions.

On the other hand, to compute Y_t - Y_hat_t = E_t

We need to first estimate Y_hat_t, we can do this through the predict function from sklearn. Having calculated the predict function we can calculate the noise (see code below.) where:

- df['actual'] = Y_t
- df['predicted'] = Y_hat_t
- df['noise'] = E_t

the time series property that is prominent in the graph below has an upward seaonality pattern. In relation to 1b.) above, the graph below behaves similar to the actual y values.

In [341…
```
#Calculate y_hat_t
y_pred = reg.predict(x)
```

In [342…
```
#Create new data frame for just the actual and predicted consumpt
df = pd.DataFrame({'Actual': y, 'Predicted': y_pred})
```

In [343…
```
df
```

Out[343…

	Actual	Predicted
2000-01-31	77.1	105.545022
2000-02-29	75.8	106.678961
2000-03-31	80.7	107.812900
2000-04-30	76.2	108.946840
2000-05-31	83.4	110.080779
…	…	…
2020-08-31	140.9	113.482597
2020-09-30	140.6	114.616537
2020-10-31	143.7	115.750476
2020-11-30	150.5	116.884416
2020-12-31	177.5	141.147619

252 rows × 2 columns

In [344…
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y, y_pr
print('Mean Squared Error:', metrics.mean_squared_error(y, y_pred
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_er
```

```
Mean Absolute Error: 14.990494056208343
Mean Squared Error: 317.75072019514874
Root Mean Squared Error: 17.825563671175978
```

In [345…
```
#Caluculate the noise
df['noise'] = (df['Actual'] - df['Predicted']).astype(int)

df
```

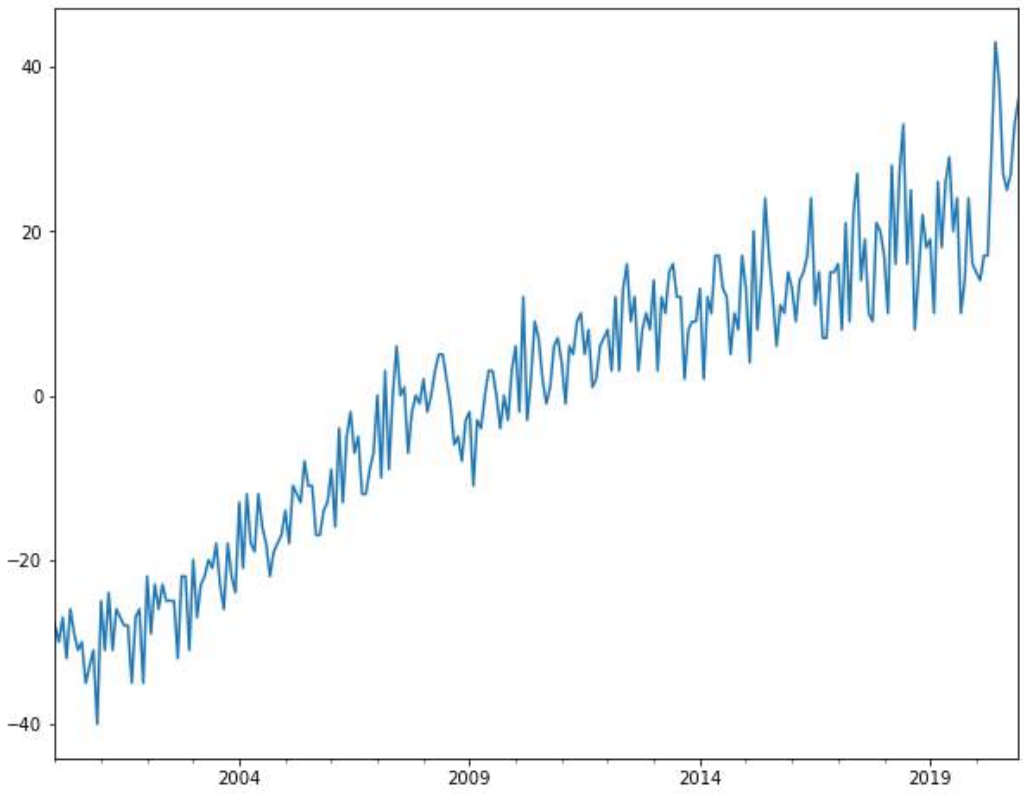
Out[345…

	Actual	Predicted	noise
2000-01-31	77.1	105.545022	-28
2000-02-29	75.8	106.678961	-30
2000-03-31	80.7	107.812900	-27
2000-04-30	76.2	108.946840	-32
2000-05-31	83.4	110.080779	-26
…	…	…	…
2020-08-31	140.9	113.482597	27

	Actual	Predicted	noise
2020-09-30	140.6	114.616537	25
2020-10-31	143.7	115.750476	27
2020-11-30	150.5	116.884416	33
2020-12-31	177.5	141.147619	36

252 rows × 3 columns

In [346…
```
df['noise'].plot()
pyplot.show()
```



4b.) Compute and plot $log(Yt)$ – $log(Yt$ – 12), i.e., the year-on-year growth in the data. Call this variable "d12y". Plot the series. What time series properties would you say are prominent for d12y? Does the data look stationary and does it have a seasonal component? Explain your answer.

In [347…
```
#Log Differencing
df['d12y'] = np.log(varekonsum['varekonsum']) - np.log(varekonsum
```

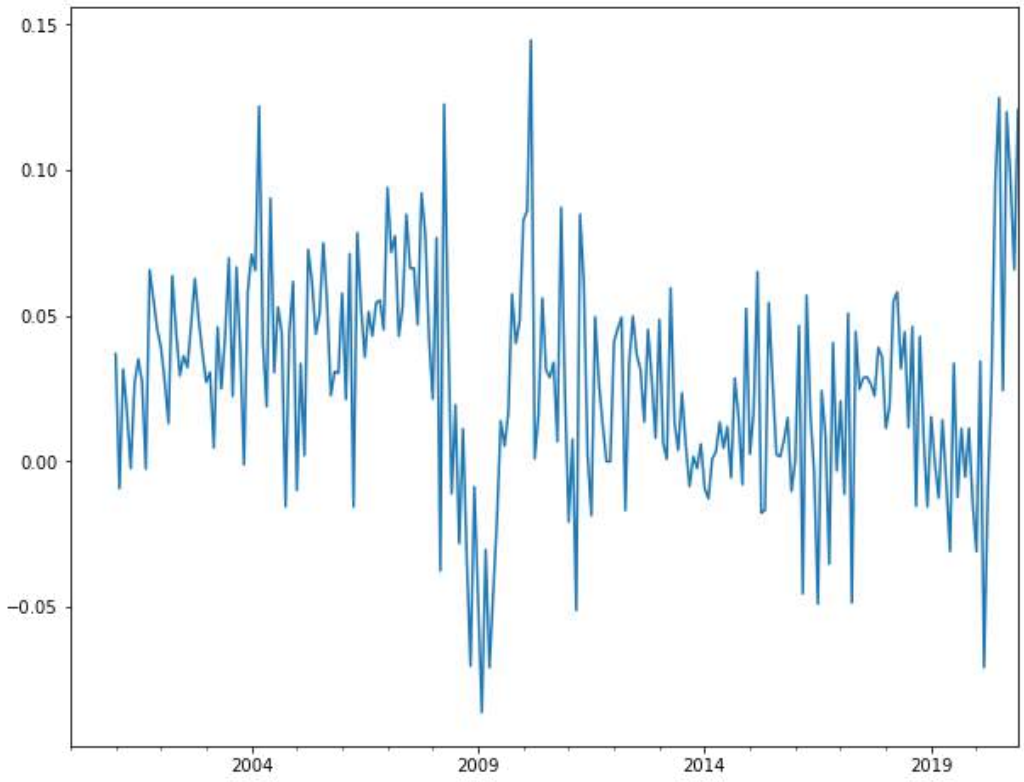
In [348…
```
df
```

Out[348…

	Actual	Predicted	noise	d12y
2000-01-31	77.1	105.545022	-28	NaN
2000-02-29	75.8	106.678961	-30	NaN
2000-03-31	80.7	107.812900	-27	NaN
2000-04-30	76.2	108.946840	-32	NaN
2000-05-31	83.4	110.080779	-26	NaN
...	...	...	...	...
2020-08-31	140.9	113.482597	27	0.024427
2020-09-30	140.6	114.616537	25	0.120008
2020-10-31	143.7	115.750476	27	0.097121
2020-11-30	150.5	116.884416	33	0.065913
2020-12-31	177.5	141.147619	36	0.120816

252 rows × 4 columns

In [349…
```
df['d12y'].plot()
pyplot.show()
```



Answer:

The time series properties that we can see from this plot firstly is the fact that ordering the list of observations periodically matters.

This data is stationary due to the fact that the process of generating the series did not change much with time.

We can see that there is no seasonality in this plot, since seasonality usually has repetetive patterns over intervals less than a year(weekly, monthly, quarterly). It is more logical to state that this plot has noise component since there is an irregular variety in the observation that we aren´t able to explain by the plot.

5b.) Estimate an AR(1) model for d12y. Is the AR model stationary? Use the BIC to find the optimal lag length for the AR(p) model for d12y. What do you get?

Hint: If you use the OLS class from the lectures/tutorial, compute the BIC using the formula $BIC = n \cdot ln(\sigma 2\varepsilon) + k \cdot ln(n)$ where "k" is the number of variables + 2.

In [350…
```
#Stats model has a built in function Auto regression which do the
from statsmodels.tsa.arima_model import ARMA
```

In [351…
```
#Since there are NaN values in our log differences we fill in 0.
d12y = df['d12y'].fillna(0)
```

In [352…
```
model = ARMA(d12y, order = (1,0,0))
model_fit = model.fit()
print(model_fit.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                    d12y   No. Observations:
252
Model:                     ARMA(1, 0)   Log Likelihood
490.416
Method:                        css-mle   S.D. of innovations
0.035
Date:                Wed, 17 Mar 2021   AIC
-974.832
Time:                        01:47:55   BIC
-964.244
Sample:                    01-31-2000   HQIC
-970.572
                         - 12-31-2020
==============================================================================
```

```
=============
                 coef    std err          z      P>|z|      [0.02
5      0.975]
------------------------------------------------------------------
-------------
const          0.0257      0.003      7.752      0.000       0.01
9       0.032
ar.L1.d12y     0.3453      0.060      5.763      0.000       0.22
8       0.463
                                    Roots
==================================================================
============
                  Real          Imaginary           Modulus
Frequency
------------------------------------------------------------------
------------
AR.1            2.8960           +0.0000j            2.8960
0.0000
------------------------------------------------------------------
------------
```

```
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.
ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (no
te the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rel
ease.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fra
mework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until t
hey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.AR
MA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.AR
IMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
```

Answer:

The result above shows that the model is stationary, because the coefficient of the ar.L1.d12y above is within the range of -1 and 1 and this in accordance to the theory as well.

Assignment C: Regularization and dimension reduction

In this assignment, let y be dividend payments and x = the remaining variables (that is, 223 columns)

Preperation: Read the data and clean.

In [353…]
```
#Read CSV file

financial_data = pd.read_csv('2018_Financial_Data.csv')

#Check the data
financial_data.head(3)
```

Out[353…]

	Unnamed: 0	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	I Exper
0	CMCSA	9.450700e+10	0.1115	0.000000e+00	9.450700e+10	0.000000e
1	KMI	1.414400e+10	0.0320	7.288000e+09	6.856000e+09	0.000000e
2	INTC	7.084800e+10	0.1289	2.711100e+10	4.373700e+10	1.354300e

3 rows × 225 columns

In [354…]
```
#Set company name as index
financial_data = financial_data.set_index('Unnamed: 0')

#Convert sector to a categorical value
financial_data["Sector"] = financial_data["Sector"].astype('categ
financial_data["Sector"] = financial_data["Sector"].cat.codes
```

In [355…]
```
#Check the data again
financial_data.head(3)
```

Out[355…]

	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses
Unnamed: 0					

Unnamed: 0	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses
CMCSA	9.450700e+10	0.1115	0.000000e+00	9.450700e+10	0.000000e+00
KMI	1.414400e+10	0.0320	7.288000e+09	6.856000e+09	0.000000e+00
INTC	7.084800e+10	0.1289	2.711100e+10	4.373700e+10	1.354300e+10

3 rows × 224 columns

Comment:

The data above shows that our financial data has an index "Unnamed: 0", and the sector column was converted in to a categorical type. This is a contrast when we first read the data.

In [356…
```
#Checking the columns
financial_data.columns
```

Out[356…
```
Index(['Revenue', 'Revenue Growth', 'Cost of Revenue', 'Gross Pro
fit',
       'R&D Expenses', 'SG&A Expense', 'Operating Expenses',
       'Operating Income', 'Interest Expense', 'Earnings before T
ax',
       ...
       'Receivables growth', 'Inventory Growth', 'Asset Growth',
       'Book Value per Share Growth', 'Debt Growth', 'R&D Expense
Growth',
       'SG&A Expenses Growth', 'Sector', '2019 PRICE VAR [%]', 'C
lass'],
      dtype='object', length=224)
```

In [357…
```
#Check if there are any null values in the data
financial_data.isnull().any()
```

Out[357…
```
Revenue                 True
Revenue Growth          True
Cost of Revenue         True
Gross Profit            True
R&D Expenses            True
                       ...
R&D Expense Growth      True
SG&A Expenses Growth    True
Sector                 False
2019 PRICE VAR [%]     False
```

```
Class                    False
Length: 224, dtype: bool
```

Comment:

Above the group saw that we are handling a large datasets, so it is important to check if there are NaN values in the data because having NaN values can drastically impact our analysis. Our group found out that there are many NaN values in our data set so we did some research into how we can handle missing values. After some research we found 3 ways to handle missing data: (1) Drop the column (2) Fill in the column (that has a NaN values) with its mean value, and (3) Fill it with a zero value to ALL columns that has NaN values. We chose the latter as choosing (1) will only leave as with few predictor variables and choosing (2) is time consuming because there more that 200 columns that has to be fill in.

In [358…
```
#Fill in 0 to cells that are empty
new_fin_data = financial_data.fillna(0)
```

In [359…
```
#Check again if there are any nulls
new_fin_data.isnull().any()
```

Out[359…
```
Revenue                  False
Revenue Growth           False
Cost of Revenue          False
Gross Profit             False
R&D Expenses             False
                         ...
R&D Expense Growth       False
SG&A Expenses Growth     False
Sector                   False
2019 PRICE VAR [%]       False
Class                    False
Length: 224, dtype: bool
```

Now our data is ready, but our group wanted to iterrate that filling in 0 to NaN values may have impact in our analysis and model

Preparation: Constructing X and Y before answering all the questions.

In [360…
```
#Converting the datasets to an array

#Dependend variable
y = new_fin_data['Dividend payments'].to_numpy()
y = np.asarray(y)

#Independent variables from data except without dividend payments
X = new_fin_data.drop(['Dividend payments'], axis = 1)
```

```
X = X.to_numpy()
X = np.asarray(X)
```

In [361…
```
from sklearn.model_selection import train_test_split

#Split the data into 2, 80% of our data set is used for training
train_X, test_X, train_y, test_y = train_test_split(X , y, train_
```

In [362…
```
#NOTE: we need to center and scale the data i.e. standardized the
#Standardization ensures that our variables are on the same scale

from sklearn.preprocessing import StandardScaler

#Initialize the class
y_std_train = StandardScaler()
y1_std_test = StandardScaler()
X_std_train = StandardScaler()
X1_std_test = StandardScaler()

#Compute mean and std. used for computing the z-score
y_std_train.fit(train_y[:,None])
y1_std_test.fit(test_y[:,None])
X_std_train.fit(train_X)
X1_std_test.fit(test_X)

#Transform the data
y_std_train = y_std_train.transform(train_y[:,None])
y1_std_test = y1_std_test.transform(test_y[:,None])
X_std_train = X_std_train.transform(train_X)
X1_std_test = X1_std_test.transform(test_X)
```

1c.) Explain why estimating a OLS regression will fail in this case?

Answer:

Estimating an OLS Regression will fail in this case because it will likely violate one of the OLS assumptions, that is, in OLS regression there should be no Multi-collinearity (or perfect collinearity) i.e. there should be no linear relationship between the independent variables.

For example, in a simple linear regression with only one independent variable, this assumption will surely hold. However, in the case of this exercise we have 223 independent variables, thus, there is a high chance that a portion of our 223 independent variables may have a linear relationship between each other and therefore violates one of the OLS assumptions.

Furthermore, from theory we know that R^2 increases with more covariates i.e. it increases as we add more independent variable in our model because the error term is lessened. This often result into a misrepresentation of the relationship between the outcome variable and predictors. Moreover, if we use the same model but with new data sets (or Out-of-sample set) our model will break or give us dubious/uncertain result. This is shown in the following code below.

In the code below, as we split our data in to training and testing, we see that R^2 is highly positive for the training sets meaning an increase in our predictors will result in to an increase in our dividend payments. But clearly this is overfitting because as we run the R^2 for testing set it says something else which does not make sense.

In [363…
```
#Given the datasets: This is how many variables were used in trai
num_train = len(train_X)
num_test = len(test_X)
print("train set: ", num_train)
print("test set: ", num_test)
```

```
train set:  3513
test set:  879
```

In [364…
```
from sklearn import linear_model

#Do the regression
reg = linear_model.LinearRegression()
reg.fit(train_X,train_y)

#Calculate the accuracy score of the model for both the training
rSq_train = reg.score(train_X, train_y)
rSq_test = reg.score(test_X, test_y)
print('R^2 using the training set: ', rSq_train)
print('R^2 using the test set: ', rSq_test)
```

```
R^2 using the training set:  0.863106345928756
R^2 using the test set:  -2419555.765862895
```

2c.) Run a LASSO regression where you choose the penalization parameter using 5-fold cross validation

(Comment)

sklearn module has a built-in function Lasso CV which do the math for us, but to see how this was done "behind-the-scenes" see appendix.

In [365…
```
from sklearn.linear_model import LassoCV

#LASSO CV using the standardized training set. Note that we alrea
```

```
#from above, but we are only going to use the training set.
Lasso = LassoCV(cv = 5, fit_intercept=False, normalize=False, tol
print('The optimal alpha/lambda is: '+ str(Lasso.alpha_))
```

```
The optimal alpha/lambda is: 0.019569823014722057
```

In [366…

```
#Using the obtained optimal alpha/tuning parameter we can now do
lasso_reg = linear_model.Lasso(alpha = 0.019569823014722057, max_
lasso_reg.fit(X_std_train, y_std_train)

#Calculate the R^2 of the lasso regression for the training set
lasso_reg_train = lasso_reg.score(X_std_train, y_std_train)

#Calculate the R^2 of the lasso regression for the testing set i.
lasso_reg_test = lasso_reg.score(X1_std_test, y1_std_test)

#Result
print('R^2 for lasso regression using the training set: ', lasso_
print('R^2 for lasso regression using the test set: ', lasso_reg_
```

```
R^2 for lasso regression using the training set:  0.8098709824056
209
R^2 for lasso regression using the test set:  0.711370265618116
```

Comment about the result above:

As one can see, doing the lasso improve the accuracy of our model both in training set and testing set. In comparision to 1c., when we demonstrated the regression score. The formula below shows how this was achieved, the cost penalty/regularization parameter, lambda, that penalizes our parameters (beta coefficients) and make it close to zero, thereby affecting the predictors by "shriking" them.

$$\beta = argmin(-2/n * loglhd(\beta)) + \lambda * (\sum|\beta|)$$

3c.) Explain shortly why cross validation is used in 2c. Explain shortly why cross validation, and out-of-sample testing in general, is used to avoid in-sample over-fitting.

Answer:

Cross Validation is used in 2c because it helps us find the best/optimal tuning parameter (lambda). For every model training, we choose a lambda and compute the sum of the squared error (SSE) (we do this for each lambda). In the case above we've done it 1000 times, then the computer choose the lambda that has the least SSE from Cross Validation and in regards to 2c, that is, 0.019569823014722057 (the appendix also show this line of reasoning).

To answer the 2nd question: The group will first explain why in-sample over-fitting occurs. It occurs because as we train our model using all the dataset it "learns/identifies" the noise (i.e. random errors) instead of the signal (i.e. the pattern), this will lead into over-fitting because our model is too good, but it breaks when new data set is introduced as it only familiarizes the noise. This is exactly what happened in 1c. Overfitting produced a good R^2 in the train set but a misleading R^2 in the test set. To avoid this problem, we used Cross Validation (CV) or Out-of-sample (OOS) testing. CV prevents overfitting because by partitioning the dataset in the case above (5-folds) we can determine overall performance of the model as Mean squared error (MSE) examines each prediction in the model.

4c.) Make a table showing the regression output. Which predictors are chosen? What is the R-squared of this regression?

Answer:

As seen from the table below there are 37 predictors out of 223 that there corresponding beta coefficient wasn't "shrink" enough by the regularization parameter (lambda), however, this does not mean that the other predictors were remove, rather it was reduced in magnitude. The 37 predictors that were choosen are the features that have prominent impact in our outcome variable. These variables are good enough to explain the relationship between the independent and dependent variable. The R^2 of this OLS regression is 0.841 meaning there is a strong positive correlation between the 37 predictors and dividend payments i.e. as one of our predictors increase, dividend payments will also increase.

In [367…

```
#Read out the variables selected by Lasso

train_X_reduced = train_X[:,Lasso.coef_ != 0]

#Run the regression
model = sm.OLS(train_y[:,None], train_X_reduced)
results = model.fit()

#Show result
print(results.summary())
```

```
                                 OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared (uncentered):
0.841
```

```
Model:                            OLS   Adj. R-squared (uncentere
d):              0.840
Method:                 Least Squares   F-statistic:
497.8
Date:                Wed, 17 Mar 2021   Prob (F-statistic):
0.00
Time:                        01:55:51   Log-Likelihood:
-74252.
No. Observations:                3513   AIC:
1.486e+05
Df Residuals:                    3476   BIC:
1.488e+05
Df Model:                          37
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.02
5      0.975]
------------------------------------------------------------------------------
x1         -2.199e+06   1.52e+06     -1.442      0.149   -5.19e+0
6    7.91e+05
x2            -0.0147      0.002     -6.354      0.000      -0.01
9      -0.010
x3            -0.1225      0.012     -9.969      0.000      -0.14
7      -0.098
x4             0.0241      0.006      4.154      0.000       0.01
3       0.035
x5            -0.0895      0.018     -4.838      0.000      -0.12
6      -0.053
x6            -0.1701      0.027     -6.266      0.000      -0.22
3      -0.117
x7             0.0077      0.108      0.071      0.943      -0.20
4       0.219
x8            -0.2463      0.036     -6.904      0.000      -0.31
6      -0.176
x9             0.2059      0.140      1.476      0.140      -0.06
8       0.479
x10           -0.8409      0.173     -4.847      0.000      -1.18
1      -0.501
x11           -0.0078      0.003     -2.219      0.027      -0.01
5      -0.001
x12        -3.25e+07   4.19e+06     -7.752      0.000   -4.07e+0
7   -2.43e+07
x13           -0.0748      0.030     -2.477      0.013      -0.13
4      -0.016
x14            0.0698      0.037      1.889      0.059      -0.00
3       0.142
x15           -0.2674      0.136     -1.961      0.050      -0.53
5      -0.000
x16           -0.0149      0.002     -7.972      0.000      -0.01
9      -0.011
x17           -0.0270      0.002    -14.596      0.000      -0.03
1      -0.023
x18           -0.0808      0.007    -11.244      0.000      -0.09
5      -0.067
x19            0.0025      0.001      2.100      0.036       0.00
```

```
0       0.005
x20           -0.0057      0.001     -4.195      0.000      -0.00
8      -0.003
x21           -0.0055      0.002     -3.223      0.001      -0.00
9      -0.002
x22           -0.0267      0.005     -4.888      0.000      -0.03
7      -0.016
x23           -0.0176      0.007     -2.456      0.014      -0.03
2      -0.004
x24           -0.0296      0.004     -7.957      0.000      -0.03
7      -0.022
x25           -0.0115      0.001    -13.220      0.000      -0.01
3      -0.010
x26            0.1823      0.031      5.964      0.000       0.12
2       0.242
x27            1.2225      0.043     28.154      0.000       1.13
7       1.308
x28           -0.0150      0.009     -1.641      0.101      -0.03
3       0.003
x29           -0.0538      0.006     -8.604      0.000      -0.06
6      -0.042
x30           -0.0227      0.002     -9.566      0.000      -0.02
7      -0.018
x31           -0.0181      0.003     -5.273      0.000      -0.02
5      -0.011
x32        -1.539e+06    6.2e+05     -2.481      0.013   -2.75e+0
6   -3.23e+05
x33           -0.0049      0.000    -16.546      0.000      -0.00
5      -0.004
x34          7.83e-05      0.000      0.529      0.597      -0.00
0       0.000
x35         1.864e+08   7.12e+07      2.619      0.009    4.68e+0
7    3.26e+08
x36         9.313e+07   3.47e+07      2.684      0.007    2.51e+0
7    1.61e+08
x37         9.866e+07   3.38e+07      2.922      0.004    3.25e+0
7    1.65e+08
==============================================================================
Omnibus:                     3614.343   Durbin-Watson:
2.021
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3754845.155
Skew:                          -4.052   Prob(JB):
0.00
Kurtosis:                     162.958   Cond. No.
1.20e+12
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model
does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the erro
rs is correctly specified.
[3] The condition number is large, 1.2e+12. This might indicate t
```

```
hat there are
strong multicollinearity or other numerical problems.
```

5c.) Use all predictors and compute 1 common component using PCA. Plot this common component.

In [368…
```
from sklearn.decomposition import PCA
from sklearn import preprocessing
```

In [369…
```
#Initial the class
scaler = StandardScaler()

#compute mean and std. for X used for computing the z-score, then
#X_std = scaler.fit_transform(X)
X_std = scaler.fit(X)

X_std = X_std.transform(X)
```

In [370…
```
#Create a PCA Object
pca = PCA(n_components = 1)

#The math part: The math behind PCA is the eigen-vectors and eige
pca.fit(X_std)

#Transform to get a PCA coordinate with component 1
pca_data = pca.transform(X_std)
```

In [371…
```
shape_scaled = X_std.shape  #224 features(column)
shape_pca = pca_data.shape #1 features(column)

print("The scaled_X has a shape: ", shape_scaled, ", that is we a
print("While when we do PCA our shape becomes", shape_pca, ", tha
```

```
The scaled_X has a shape:  (4392, 223) , that is we are consideri
ng 223 columns
While when we do PCA our shape becomes (4392, 1) , that is we are
now looking at 1 component that caputurea most of the information
in our data set
```

In [372…
```
#Just for illustration
pca_var = pca.explained_variance_
pca_var_ratio = pca.explained_variance_ratio_

print('PCA Variance: ', pca_var)
print('PCA Variance Ratio: ', pca_var_ratio)
```

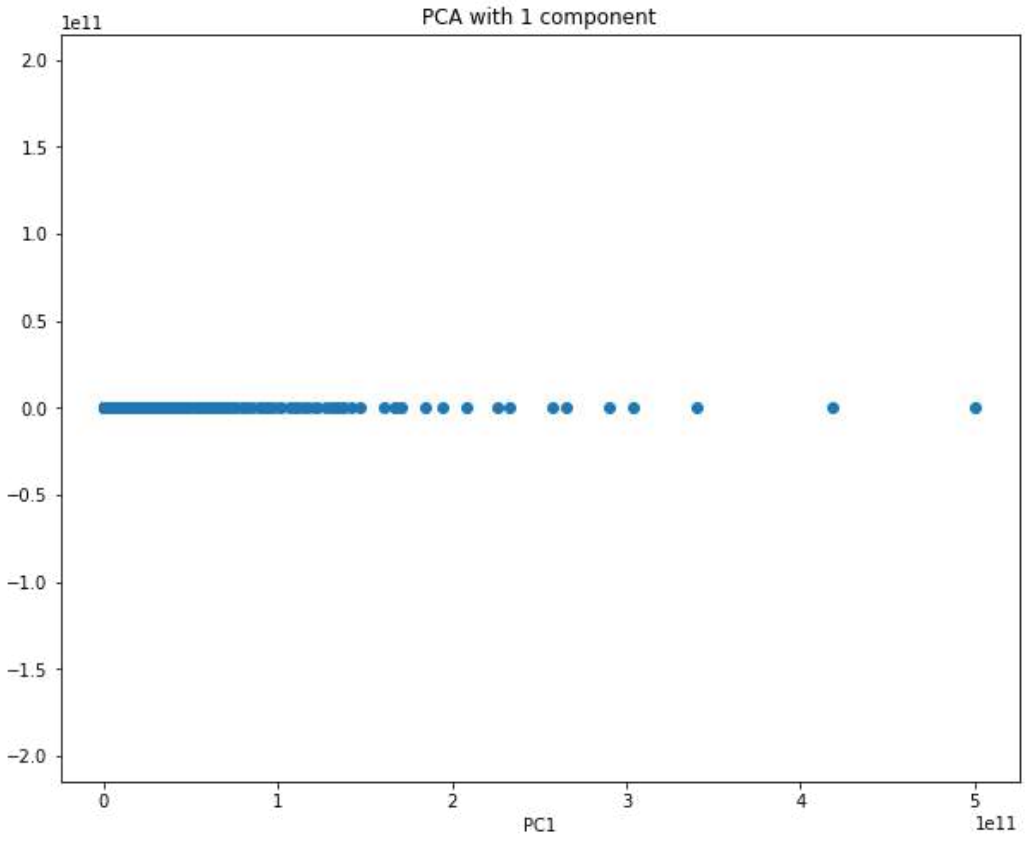
```
PCA Variance:  [23.93235675]
PCA Variance Ratio:  [0.10729555]
```

Comment:

From above, our PC(1) contains 10.7% of the variability of our features.

In [373…

```
#Scale back the data to the original representation
X_new = pca.inverse_transform(pca_data)


#plot
plt.scatter(X[:, 0], X_new[:, 1])


plt.title('PCA with 1 component')
plt.xlabel('PC1')
plt.axis('equal');
```



Comment:

In this assignment we were only ask to plot 1 common component. The algorithm chose the PCA component that has the biggest value, that is PC(1) which captures the most relevent information of the predictors.

6c.) Regress the outcome variable on the common component. What is the R-squared of this regression? Compare your result with what you got in 4c.

Answer:

As seen below, most of the data sets are clustered between 0 and 20 in the PC(1), running an OLS regression gives us a a positive correlation (R^2 = 0.605) between the PC (1) and Dividend Payments which is smaller than the what we got at 4c.

However, the PC(1) represents the data which the observation vary the most which avoids the issue of multi-collinearity, also the 1st component represents the most and relevant information about the predictors. Unlike LASSO where it penalize coefficients that it almost approximate some predictors to 0, while principal component analysis treated all predictors equally.

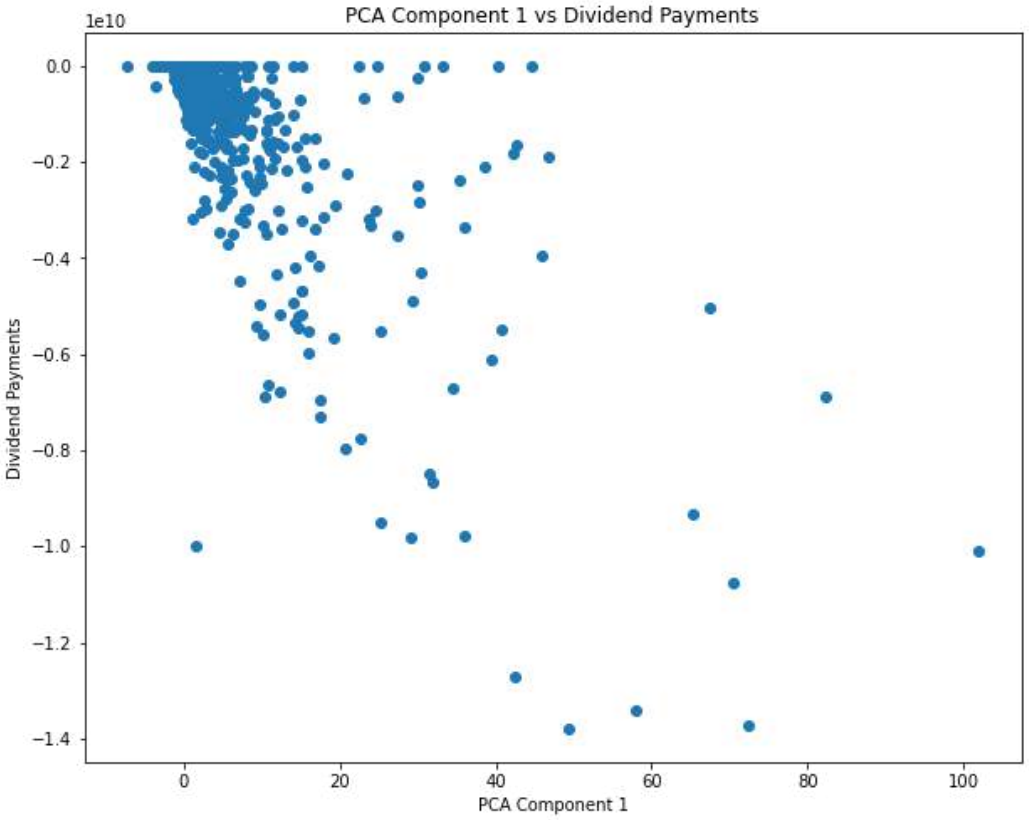
In summary, R^2 in Lasso is greater than R^2 in PCA, but in Lasso we "tune out" the other predictors which improves our model but PCA considered all predictors and "pushed/squeeze" the predictors toward a singular dimension (in this case) where our X's minimizes the error. So our group concludes that PCA is maybe more accurate than running lasso.

In [374…

```
#Plot
plt.scatter(pca_data[:, 0], y)

#Set title
plt.title('PCA Component 1 vs Dividend Payments')

# Set x-axis label
plt.xlabel('PCA Component 1')

# Set y-axis label
plt.ylabel('Dividend Payments')
```

Out[374…

```
Text(0, 0.5, 'Dividend Payments')
```



In [375…

```
#Run Regression
results = sm.OLS(y,sm.add_constant(pca_data)).fit()

#Show Result
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:
0.605
Model:                            OLS   Adj. R-squared:
0.605
Method:                 Least Squares   F-statistic:
6732.
Date:                Wed, 17 Mar 2021   Prob (F-statistic):
0.00
Time:                        01:58:10   Log-Likelihood:
-94513.
No. Observations:                4392   AIC:
1.890e+05
Df Residuals:                    4390   BIC:
1.890e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.02
```

```
5      0.975]
------------------------------------------------------------------------
-------------
const      -1.902e+08    8.1e+06    -23.495      0.000   -2.06e+0
8   -1.74e+08
x1         -1.358e+08   1.66e+06    -82.047      0.000   -1.39e+0
8   -1.33e+08
========================================================================
=============
Omnibus:                     4061.187   Durbin-Watson:
2.060
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1155567.338
Skew:                          -3.692   Prob(JB):
0.00
Kurtosis:                      82.120   Cond. No.
4.89
========================================================================
=============

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

7c.) In light of your answer in (6c), discuss shortly in which cases using PCA versus LASSO might be beneficial.

PCA is beneficial when we handle large data sets similar above because it avoids multicollinearity. Though, PCA reduces the dimension of the data, it did not remove any predictors rather it removes redundancy.

This is a strinking contrast between PCA and the Lasso. In Lasso, we drop/minimize predictors that is penalized by our lambda which improves the model's accuracy as seen above when we run the lasso R^2 between training and test set. Yet, there are still posibility that the X's chosen have interrellationship between each other, thus it will still likely violate the OLS assumption.

While in PCA, the issue of multicollinearity can be mitigated and still contain all information about the predictors.

Appendix

Lasso CV

* Visually, this is how the algorithm works.

In this assg. we are doing 5-folds from our train set. From the book, it was shown to us how mathematically we minimize a penalized deviance. That is

$$\hat{\beta} = \text{argmin} \left\{ -\frac{1}{n} \log \text{lhd}(\beta) + \lambda \sum_{k} c(\beta_k) \right\}$$

In 1c, in order to run a LASSO we have to pick the optimal λ, this is done through Cross Validation. The algorithm does it by

Suppose we have a grid of λ values. We choose a λ from the grid and fit in into equation above to determine $\hat{\beta}$ we using the train sets. Then we predict $\hat{y}$ using $\hat{\beta}$ and X values. After that, we calculate the sum of the squared errors (SSE) between $\hat{y} - y$. We repeat this process n times (in the assignment max_iter = 1000) for every λ in our grid of λ values, after repeating the process we pick λ that gives us the smallest SSE. In the assignment that lambda is 0.01956987... .

The obtained lambda will be then used to determine $\hat{\beta}$ that will either be highly penalized by the optimal λ that it will "time out" or just rightly penalized.