Case Study_1-MTP

July 15, 2021

#importing packages

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
```

1 Part I: Company Background (2 pts)

[2]:
```python
#Company's info
co = "MTP"
Get_Information = yf.Ticker(co)

# get all key value pairs that are available
for key, value in Get_Information.info.items():
    print(key, ":", value)
```



```
zip : CF24 0AA
sector : Healthcare
fullTimeEmployees : 18
longBusinessSummary : Midatech Pharma plc focuses on the research and
development of oncology and rare disease products in the United Kingdom, rest of
Europe, and internationally. The company is developing MTX110, a direct delivery
treatment for diffuse intrinsic pontine glioma, medulloblastomas, and
glioblastoma multiforme; MTX114, an immuno-suppressant for topical application
in psoriasis; and MTD211 and MTD219 for central nervous system and transplant
anti-rejection indications. It also offers drug delivery platforms, such as
Q-Sphera, a polymer microsphere microtechnology used for sustained release drug
delivery; MidaSolve, an oligosaccharide nanotechnology used to solubilize drugs
so that they can be administered in liquid form directly and locally into
tumors; and MidaCore, a gold nanoparticle used for targeting sites of disease by
using chemotherapeutic agents or immunotherapeutic agents. The company was
founded in 2000 and is headquartered in Cardiff, the United Kingdom.
city : Cardiff
phone : 44 1235 888 300
country : United Kingdom
companyOfficers : []
website : http://www.midatechpharma.com
```

maxAge : 1
address1 : Oddfellows House
industry : Biotechnology
address2 : 19 Newport Road
ebitdaMargins : 0
profitMargins : 0
grossMargins : 0
operatingCashflow : -9301000
revenueGrowth : None
operatingMargins : -23.93878
ebitda : -7112000
targetLowPrice : None
recommendationKey : none
grossProfits : -3985000
freeCashflow : -5172125
targetMedianPrice : None
currentPrice : 1.9
earningsGrowth : None
currentRatio : 3.103
returnOnAssets : -0.25211

numberOfAnalystOpinions : None
targetMeanPrice : None
debtToEquity : 5.001
returnOnEquity : -1.68892

targetHighPrice : None
totalCash : 7546000
totalDebt : 336000
totalRevenue : 343000

totalCashPerShare : 0.595
financialCurrency : GBP
revenuePerShare : 0.04
quickRatio : 3.017
recommendationMean : None
exchange : NMS
shortName : Midatech Pharma PLC
longName : Midatech Pharma plc
exchangeTimezoneName : America/New_York
exchangeTimezoneShortName : EDT
isEsgPopulated : False
gmtOffSetMilliseconds : -14400000
quoteType : EQUITY
symbol : MTP
messageBoardId : finmb_278298574
market : us_market
annualHoldingsTurnover : None
enterpriseToRevenue : 51.784
beta3Year : None
enterpriseToEbitda : -2.497

52WeekChange : 0.49242425
morningStarRiskRating : None
forwardEps : 0
revenueQuarterlyGrowth : None
sharesOutstanding : 19693700
fundInceptionDate : None
annualReportExpenseRatio : None
totalAssets : None
bookValue : 0.535
sharesShort : 334703
sharesPercentSharesOut : 0.026400002
fundFamily : None
lastFiscalYearEnd : 1609372800
heldPercentInstitutions : 0.077020004
netIncomeToCommon : -22189000
trailingEps : -3.595
lastDividendValue : None
SandP52WeekChange : 0.35413873
priceToBook : 3.5514016
heldPercentInsiders : 0.00135
nextFiscalYearEnd : 1672444800
yield : None
mostRecentQuarter : 1609372800
shortRatio : 0.06
sharesShortPreviousMonthDate : 1622160000
floatShares : 11016067
beta : 1.678991
enterpriseValue : 17761982
priceHint : 4
threeYearAverageReturn : None
lastSplitDate : 1583193600
lastSplitFactor : 1:5
legalType : None
lastDividendDate : None
morningStarOverallRating : None
earningsQuarterlyGrowth : None
priceToSalesTrailing12Months : 109.58292
dateShortInterest : 1625011200
pegRatio : None
ytdReturn : None
forwardPE : None
lastCapGain : None
shortPercentOfFloat : None
sharesShortPriorMonth : 165517
impliedSharesOutstanding : None
category : None
fiveYearAverageReturn : None
previousClose : 1.97

```
regularMarketOpen : 1.96
twoHundredDayAverage : 2.2109044
trailingAnnualDividendYield : None
payoutRatio : 0
volume24Hr : None
regularMarketDayHigh : 1.9635
navPrice : None
averageDailyVolume10Day : 276966
regularMarketPreviousClose : 1.97
fiftyDayAverage : 2.1191177
trailingAnnualDividendRate : None
open : 1.96
toCurrency : None
averageVolume10days : 276966
expireDate : None
algorithm : None
dividendRate : None
exDividendDate : None
circulatingSupply : None
startDate : None
regularMarketDayLow : 1.9
currency : USD
regularMarketVolume : 125690
lastMarket : None
maxSupply : None
openInterest : None
marketCap : 37586940
volumeAllCurrencies : None
strikePrice : None
averageVolume : 2205290
dayLow : 1.9
ask : 1.99
askSize : 1100
volume : 125690
fiftyTwoWeekHigh : 7.07
fromCurrency : None
fiveYearAvgDividendYield : None
fiftyTwoWeekLow : 1.26
bid : 1.9
tradeable : False
dividendYield : None
bidSize : 2900
dayHigh : 1.9635
regularMarketPrice : 1.9
logo_url : https://logo.clearbit.com/midatechpharma.com
```

2 Part 2: Daily stock returns (3 pts)

[3]:
```python
#S&P500 =sp
sp = yf.download("^GSPC",
                          start='2015-9-1',
                          end='2021-6-30')
#Stock (Microsoft)= st
st = yf.download(co,
                          start='2015-9-1',
                          end='2021-6-30')
#Risk-free rate (Rf)
rf = yf.download("^IRX",
                          start='2015-9-1',
                          end='2021-6-30')
sp
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

[3]:
```
                   Open         High          Low        Close    Adj Close  \
Date
2015-08-31  1986.729980  1986.729980  1965.979980  1972.180054  1972.180054
2015-09-01  1970.089966  1970.089966  1903.069946  1913.849976  1913.849976
2015-09-02  1916.520020  1948.910034  1916.520020  1948.859985  1948.859985
2015-09-03  1950.790039  1975.010010  1944.719971  1951.130005  1951.130005
2015-09-04  1947.760010  1947.760010  1911.209961  1921.219971  1921.219971
…                   …            …            …            …            …
2021-06-23  4249.270020  4256.600098  4241.430176  4241.839844  4241.839844
2021-06-24  4256.970215  4271.279785  4256.970215  4266.490234  4266.490234
2021-06-25  4274.450195  4286.120117  4271.160156  4280.700195  4280.700195
2021-06-28  4284.899902  4292.140137  4274.669922  4290.609863  4290.609863
2021-06-29  4293.209961  4300.520020  4287.040039  4291.799805  4291.799805

                Volume
Date
2015-08-31  3915100000
2015-09-01  4371850000
2015-09-02  3742620000
2015-09-03  3520700000
2015-09-04  3167090000
…                    …
2021-06-23  3172440000
2021-06-24  3141680000
2021-06-25  6248390000
2021-06-28  3415610000
2021-06-29  3049560000
```

```
[1468 rows x 6 columns]
```

[4]:
```
# Computing daily stock returns
R =100*np.log(st['Adj Close']/st['Adj Close'].shift(1)).dropna()
#Market Index returns: S&P500
M =100*np.log(sp['Adj Close']/sp['Adj Close'].shift(1)).dropna()
#Risk-free rate returns
Rf =(rf['Adj Close']/360).dropna()
Rf.drop(rf[rf["Adj Close"] == "." ].index, inplace=True)
```

3 Part 3:CAPM

Merging data files for CAPM

[5]:
```
dt =pd.merge(M,Rf, on='Date', how='left').dropna()
data = pd.merge(dt,R, on='Date', how='left').dropna()
data_cols=['M','Rf','R']
data.columns =data_cols
data
```


[5]:
```
                   M        Rf          R
Date
2015-12-08 -0.651105  0.000744 -15.733033
2015-12-09 -0.776909  0.000681  -4.561051
2015-12-10  0.224886  0.000639   5.511930
2015-12-11 -1.961387  0.000592  -0.791770
2015-12-14  0.474429  0.000550  -2.903436
…                …         …          …
2021-06-23 -0.108387  0.000111   1.754429
2021-06-24  0.579443  0.000119   0.865805
2021-06-25  0.332506  0.000119  -1.301536
2021-06-28  0.231229  0.000111  -2.655021
2021-06-29  0.027730  0.000111  -6.483757

[1387 rows x 3 columns]
```


Calculating excess returns for Stock and $S\&P500$

[6]:
```
data['R_p']= data['M']- data['Rf']
data['R_s']= data['R']- data['Rf']
data
```

[6]:
```
                   M        Rf          R       R_p        R_s
Date
2015-12-08 -0.651105  0.000744 -15.733033 -0.651850 -15.733778
2015-12-09 -0.776909  0.000681  -4.561051 -0.777589  -4.561732
2015-12-10  0.224886  0.000639   5.511930  0.224247   5.511291
2015-12-11 -1.961387  0.000592  -0.791770 -1.961978  -0.792361
```

```
2015-12-14  0.474429  0.000550  -2.903430  0.473879  -2.903980
…                …         …          …         …          …
2021-06-23 -0.108387  0.000111   1.754429 -0.108498   1.754318
2021-06-24  0.579443  0.000119   0.865805  0.579324   0.865686
2021-06-25  0.332506  0.000119  -1.301536  0.332387  -1.301655
2021-06-28  0.231229  0.000111  -2.655021  0.231118  -2.655132
2021-06-29  0.027730  0.000111  -6.483757  0.027619  -6.483868

[1387 rows x 5 columns]
```

Data : Remove N/A

[7]:
```python
data = data.dropna(subset=["R_p"])
data.to_csv("C:\\Users\\rluck\\OneDrive\\capm2.csv")
data.head()
```

[7]:
```
                   M        Rf          R       R_p        R_s
Date
2015-12-08 -0.651105  0.000744 -15.733033 -0.651850 -15.733778
2015-12-09 -0.776909  0.000681  -4.561051 -0.777589  -4.561732
2015-12-10  0.224886  0.000639   5.511930  0.224247   5.511291
2015-12-11 -1.961387  0.000592  -0.791770 -1.961978  -0.792361
2015-12-14  0.474429  0.000550  -2.903430  0.473879  -2.903980
```


[8]:
```python
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import linear_model
```

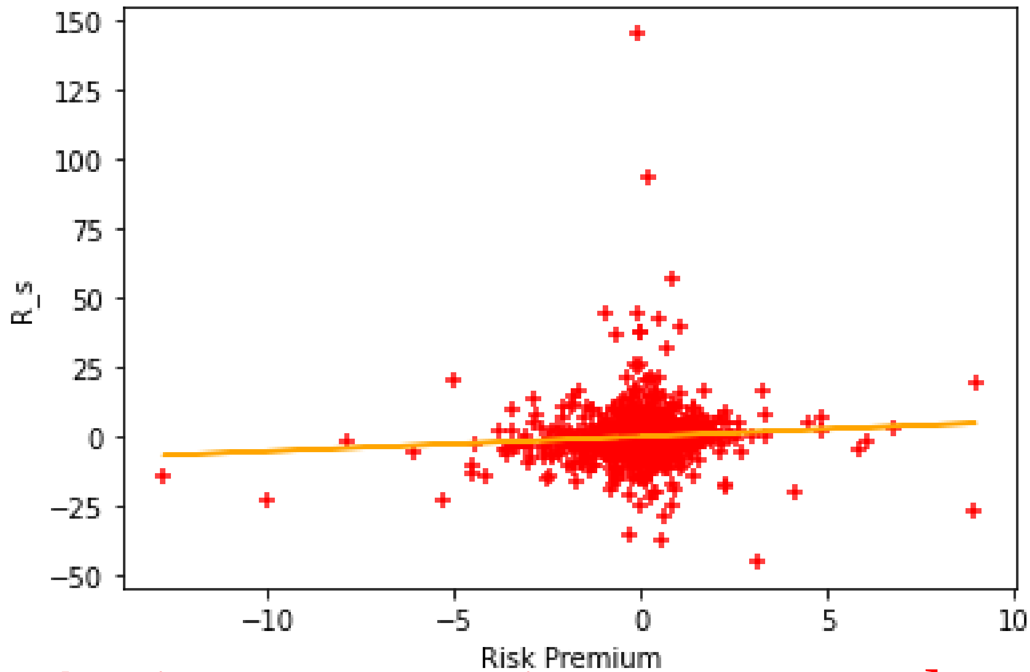
3(a): CAPM model (3pts)

I. Plotting stock's excess returns with market excess returns

[9]:
```python
#Regressing excess returns on gold (R_g-Rf) over risk-free rate against the␣
↪excess market return (Rp=Rm-rf)
reg = linear_model.LinearRegression()
X =data[['R_p']].dropna()
y =data['R_s'].dropna()
reg.fit(X,y)
predictions =reg.predict(X)
```

[10]:
```python
plt.xlabel('Risk Premium')
plt.ylabel('R_s')
plt.scatter(data.R_p,data.R_s,color='red',marker='+')
plt.plot(data.R_p,reg.predict(data[['R_p']]), color='orange')
```

[10]: `[<matplotlib.lines.Line2D at 0x1c10a1eb160>]`



[11]:

```
#model with intercept
X= sm.add_constant(X)
model = sm.OLS(y,X).fit()
predictions = model.predict(X)
j= (model.summary())
print(j)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    R_s   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.005
Method:                 Least Squares   F-statistic:                     8.203
Date:                Thu, 15 Jul 2021   Prob (F-statistic):            0.00425
Time:                        11:57:20   Log-Likelihood:                -4898.2
No. Observations:                1387   AIC:                             9800.
Df Residuals:                    1385   BIC:                             9811.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.4050      0.222     -1.821      0.069      -0.841       0.031
R_p            0.5343      0.187      2.864      0.004       0.168       0.900
==============================================================================
Omnibus:                     1609.181   Durbin-Watson:                   2.057
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           420127.290
Skew:                           5.437   Prob(JB):                         0.00
Kurtosis:                      87.566   Cond. No.                         1.20
==============================================================================

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

3(b) Interpretation Replicating Portfolio (3 pts)

DW-stats of 1.987 is close to 2.0, implying that there is no serial correlation.

Since p-value of the beta coefficient is less than 0.05, we reject the null hypothesis that beta is zero.

The CAPM equation for stock can be written as follows:

$R_s = 1.1931 * R_p + Rf$

where R_s is the return from the stock, $R_p = Rm - Rf$ is the market risk premium and Rf is the risk free rate of return

Replicating portfolio

If we want to replicate the returns from the, we can rearrange the above equation:

$R_g e = 1.1931 * Rm + (1 - 1.1931) * Rf$

$\Rightarrow$ We can buy 1.1931 of market portfolio (i.e.: S&P500 index fund) and then short 0.1931 T-Bill.

[12]:
```
#Determining Expected returns from replicating portfolio and variance
import statistics
Beta = model.params['R_p']
W_r = 1-Beta
ER = Beta*data['M'] + W_r*data['Rf']
name= ['Mean','Variance', 'Std Dev','S-ratio']
des=[ER.mean(),statistics.variance(ER), (statistics.variance(ER))**0.5,(ER.
 ↪mean()-Rf.mean())/(statistics.variance(ER))**0.5]
ret= (name,des)
ret
```

[12]:
```
(['Mean', 'Variance', 'Std Dev', 'S-ratio'],
 [0.030099160389495044,
  0.4052745106056766,
  0.6366117424346465,
  0.043130227785489576])
```

[13]:
```
# Comparing with stock' s expected returns and variance
name= ['Mean','Variance','Std Dev','S-ratio']
des=[R.mean(),statistics.variance(R), (statistics.variance(R))**0.5,(R.
 ↪mean()-Rf.mean())/(statistics.variance(R))**0.5]
ret= (name,des)
```

```
ret
```

[13]:

```
(['Mean', 'Variance', 'Std Dev', 'S-ratio'],
 [-0.3695181768222652,
  68.50889734292767,
  8.277010169314018,
  -0.044963111091656474])
```

3c: Stability Tests (3pts)

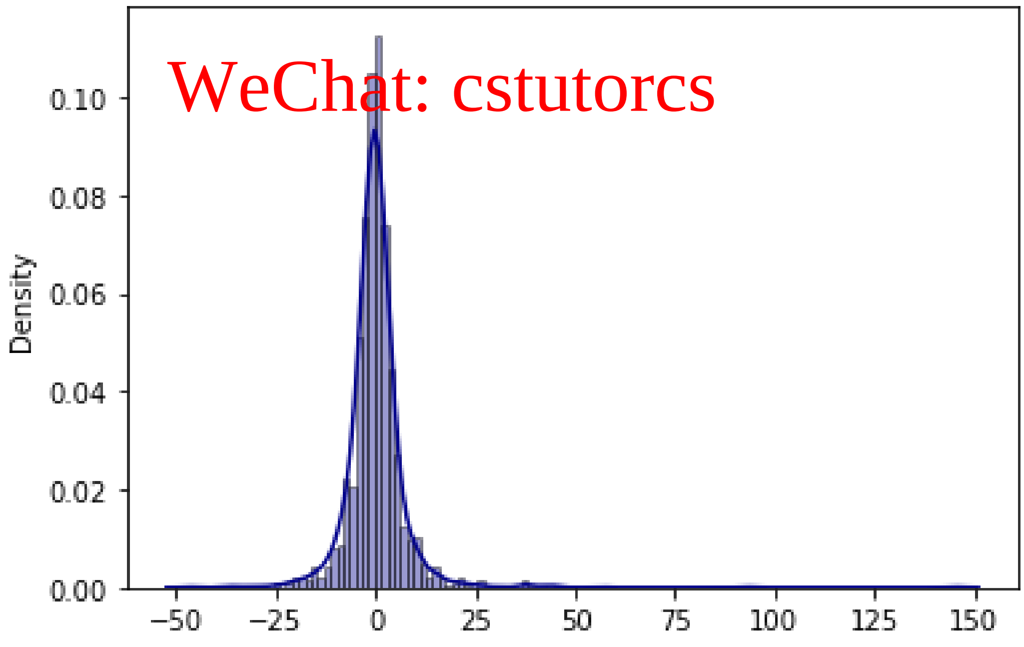
Residual Plots for stock

[14]:

```
residuals = model.resid
import seaborn as sns
sns.distplot(residuals,hist=True, kde=True, bins=int(120), color=␣
 ↪'darkblue',hist_kws={'edgecolor':'black'})
```

```
C:\Users\rluck\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```


[14]: <AxesSubplot:ylabel='Density'>



[15]:
```
from scipy import stats
JB_s= stats.jarque_bera(residuals)
JB_s
```

[15]:
```
Jarque_beraResult(statistic=420127.2898856596, pvalue=0.0)
```

The plot and JB test (p-value <0.05) rejects the null hypothesis of normality. It is clearly a non-normal distribution.

CUSUM test

[16]:
```
endog = data.R_s
Rp = data.R_p
exog = sm.add_constant(Rp)
mod = sm.RecursiveLS(endog,exog)
res_1 = mod.fit()
fig = res_1.plot_cusum(figsize=(10,6));
```

```
C:\Users\rluck\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
```


Cusum test of stability for GE shows stability of beta as it is within the 5% significance level band.

White Test of Heteroskedasticity for the stock

```
[17]: from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
from patsy import dmatrices
expr = 'y ~ X'
y, X = dmatrices(expr, data, return_type='dataframe')
olsr_results = smf.ols(expr, data).fit()
keys = ['Lagrange Multiplier statistic:', 'LM test\'s p-value:', 'F-statistic:
 ↪', 'F-test\'s p-value:']
results = het_white(olsr_results.resid, X)
lzip(keys, results)
```

```
[17]: [('Lagrange Multiplier statistic:', 0.771360820220893),
 ("LM test's p-value:", 0.6799878081863415),
 ('F-statistic:', 0.3850603518829708),
 ("F-test's p-value:", 0.6804824418633941)]
```

LM test statistic is 10.2669 and the corresponding p-value is 0

F-stats = 10.2669 and the corresponding p-value is 0

Since the p-value of the both LM and F-stats is less than 0.05, we reject the null hypothesis that there is no heteroskedasticity in the residuals. It infers that the heteroskedasticity exists and the standard errors need to be corrected.

BG test


```
[18]: import statsmodels.stats.diagnostic as dg
print (dg.acorr_breusch_godfrey(model, nlags= 2))
```



```
(15.723679286602392, 0.00038516455364831294, 7.929054683304046,
0.0003767458877534564)
```

T-statistic of Chi-squared = 2.20945 and the corresponding p-value = 0.3294.

F-statistics = 1.109 and the corresponding p-value = 0.3301

Since p-value exceeds 0.05, we fail to reject the null hypothesis, thus inferring there is no autocorrelation at order less than or equal to 2.0

4 Part 4: APT

4a: 3-factor APT model (3 pts)

Merging data files for APT

```
[19]: print(data)
```

```
                   M        Rf          R       R_p        R_s
Date
2015-12-08 -0.651105  0.000744 -15.733033 -0.651850 -15.733778
2015-12-09 -0.776909  0.000681  -4.561051 -0.777589  -4.561732
2015-12-10  0.224886  0.000639   5.511930  0.224247   5.511291
```

```
2015-12-11 -1.961387  0.000592  -0.791770 -1.961978  -0.792361
2015-12-14  0.474429  0.000550  -2.903430  0.473879  -2.903980
…                …         …         …         …         …
2021-06-23 -0.108387  0.000111   1.754429 -0.108498   1.754318
2021-06-24  0.579443  0.000119   0.865805  0.579324   0.865686
2021-06-25  0.332506  0.000119  -1.301536  0.332387  -1.301655
2021-06-28  0.231229  0.000111  -2.655021  0.231118  -2.655132
2021-06-29  0.027730  0.000111  -6.483757  0.027619  -6.483868

[1387 rows x 5 columns]
```

[20]:
```python
#Reading Fama file
#SMB
fama= pd.read_excel("C:\\Users\\rluck\\OneDrive\\fama_1.xlsx")
fama
```

[20]:
```
            Date  Mkt-RF0  SMB0  HML0    RF0
0     1926-07-01     0.10 -0.24 -0.28  0.009
1     1926-07-02     0.45 -0.32 -0.08  0.009
2     1926-07-06     0.17  0.27 -0.35  0.009
3     1926-07-07     0.09 -0.59  0.03  0.009
4     1926-07-08     0.21 -0.36  0.15  0.009
…            …        …     …     …      …
24993 2021-05-24     1.00 -0.38 -0.69  0.000
24994 2021-05-25    -0.30 -0.60 -1.22  0.000
24995 2021-05-26     0.46  1.77  0.52  0.000
24996 2021-05-27     0.28  0.80  0.95  0.000
24997 2021-05-28     0.04 -0.30 -0.27  0.000

[24998 rows x 5 columns]
```

[21]:
```python
#Set date as index
fama = fama.set_index('Date')
fama.index.astype(str)
```

[21]:
```
Index(['1926-07-01', '1926-07-02', '1926-07-06', '1926-07-07', '1926-07-08',
       '1926-07-09', '1926-07-10', '1926-07-12', '1926-07-13', '1926-07-14',
       …
       '2021-05-17', '2021-05-18', '2021-05-19', '2021-05-20', '2021-05-21',
       '2021-05-24', '2021-05-25', '2021-05-26', '2021-05-27', '2021-05-28'],
      dtype='object', name='Date', length=24998)
```

[22]:
```python
data.index.astype(str)
```

[22]:
```
Index(['2015-12-08', '2015-12-09', '2015-12-10', '2015-12-11', '2015-12-14',
       '2015-12-15', '2015-12-16', '2015-12-17', '2015-12-18', '2015-12-21',
       …
       '2021-06-16', '2021-06-17', '2021-06-18', '2021-06-21', '2021-06-22',
```

```
       '2021-06-23', '2021-06-24', '2021-06-25', '2021-06-28', '2021-06-29'],
      dtype='object', name='Date', length=1387)
```

[23]:

```python
fama
```

[23]:

```
            Mkt-RF0  SMB0  HML0    RF0
Date
1926-07-01     0.10 -0.24 -0.28  0.009
1926-07-02     0.45 -0.32 -0.08  0.009
1926-07-06     0.17  0.27 -0.35  0.009
1926-07-07     0.09 -0.59  0.03  0.009
1926-07-08     0.21 -0.36  0.15  0.009
…               …     …     …     …
2021-05-24     1.00 -0.38 -0.69  0.000
2021-05-25    -0.30 -0.60 -1.22  0.000
2021-05-26     0.46  1.77  0.52  0.000
2021-05-27     0.28  0.80  0.95  0.000
2021-05-28     0.04 -0.30 -0.27  0.000

[24998 rows x 4 columns]
```


[24]:

```python
dta= pd.merge(data,fama,left_index=True, right_index =True).dropna()
dta_cols=['M','Rf','R','R_p','R_s','Mkt-RF','SMB','HML','RF']
dta.columns =dta_cols
dta.dropna()
```


[24]:


```
                   M        Rf          R       R_p        R_s  Mkt-RF   SMB  \
Date
2015-12-08 -0.651105  0.000744 -15.733033 -0.651850 -15.733778   -0.59  0.49
2015-12-09 -0.776909  0.000681  -4.561051 -0.777589  -4.561732   -0.83 -0.34
2015-12-10  0.224886  0.000639   5.511930  0.224247   5.511291    0.30  0.10
2015-12-11 -1.961387  0.000592  -0.791770 -1.961978  -0.792361   -2.03 -0.21
2015-12-14  0.474429  0.000550  -2.903430  0.473879  -2.903980    0.29 -1.04
…                …         …          …         …          …      …     …
2021-05-24  0.986250  0.000008   1.904818  0.986241   1.904809    1.00 -0.38
2021-05-25 -0.212755  0.000028  -0.472814 -0.212782  -0.472841   -0.30 -0.60
2021-05-26  0.187506  0.000014  -1.432004  0.187492  -1.432018    0.46  1.77
2021-05-27  0.116464  0.000014  -5.433430  0.116450  -5.433444    0.28  0.80
2021-05-28  0.076859  0.000022  -3.093033  0.076836  -3.093055    0.04 -0.30

             HML   RF
Date
2015-12-08 -1.21  0.0
2015-12-09  0.42  0.0
2015-12-10 -0.20  0.0
2015-12-11 -0.05  0.0
2015-12-14 -0.18  0.0
```

```
…            …   …
2021-05-24 -0.69  0.0
2021-05-25 -1.22  0.0
2021-05-26  0.52  0.0
2021-05-27  0.95  0.0
2021-05-28 -0.27  0.0

[1366 rows x 9 columns]
```

[25]:
```
dta.dropna(subset = ["SMB"], inplace=True)
dta.dropna(subset = ["HML"], inplace=True)
```

OLS Regression to determine beta under APT (3-factor Model)

[26]:
```
import statsmodels.api as sm
#X & y Variables defined
X_1 = dta[["R_p","SMB","HML"]]
X_1 = sm.add_constant(X_1)
y= dta["R"]-dta["Rf"]
#OLS model
model = sm.OLS(y,X_1).fit()
Q= model.summary()
print(Q)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.011
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     5.212
Date:                Thu, 15 Jul 2021   Prob (F-statistic):            0.00140
Time:                        11:57:23   Log-Likelihood:                -4817.8
No. Observations:                1366   AIC:                             9644.
Df Residuals:                    1362   BIC:                             9664.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.4183      0.223     -1.873      0.061      -0.856       0.020
R_p            0.5017      0.188      2.667      0.008       0.133       0.871
SMB            0.9378      0.355      2.643      0.008       0.242       1.634
HML           -0.0254      0.246     -0.103      0.918      -0.509       0.458
==============================================================================
Omnibus:                     1609.387   Durbin-Watson:                   2.060
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           448491.937
Skew:                           5.568   Prob(JB):                         0.00
Kurtosis:                      91.067   Cond. No.                         1.95
==============================================================================
```

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

4b: Regression with oil price change and forex (3 pts)

[27]:
```python
#Crude oil
oil = yf.download("CL=F",
                       start='2015-9-1',
                       end='2021-6-30')
R_o =100*np.log(oil['Close']/oil['Close'].shift(1)).dropna()
R_o
#Fx: AUD/USD (AUD per USD)
forex = yf.download("AUD=X",
                       start='2015-9-1',
                       end='2021-6-30')
R_for=100*np.log(forex['Close']/forex['Close'].shift(1)).dropna()
forex
```

```
[*********************100%***********************]  1 of 1 completed
C:\Users\rluck\anaconda3\lib\site-packages\pandas\core\arraylike.py:358:
RuntimeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
[*********************100%***********************]  1 of 1 completed
```


[27]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2015-08-31	1.400600	1.411400	1.396800	1.400400	1.400400	0
2015-09-01	1.406400	1.424500	1.398600	1.406600	1.406600	0
2015-09-02	1.426100	1.431500	1.420000	1.426700	1.426700	0
2015-09-03	1.419600	1.429000	1.416200	1.419000	1.419000	0
2015-09-04	1.423900	1.446800	1.423900	1.424700	1.424700	0
…	…	…	…	…	…	…
2021-06-23	1.323574	1.326559	1.315789	1.323660	1.323660	0
2021-06-24	1.320170	1.321320	1.317000	1.320230	1.320230	0
2021-06-25	1.319090	1.319090	1.312500	1.318900	1.318900	0
2021-06-28	1.316656	1.323469	1.315097	1.316656	1.316656	0
2021-06-29	1.321266	1.331820	1.320800	1.321283	1.321283	0

[1499 rows x 6 columns]

[28]:
```python
# Merging files
dt1 =pd.merge(dt,R, on='Date', how='left').dropna()
dt2 = pd.merge(dt1,R_o, on='Date', how='left').dropna()
dt3 = pd.merge(dt2,R_for, on='Date', how='left').dropna()
dt3_cols=['M','Rf','R','R_o','R_for']
dt3.columns =dt3_cols
```

```
dt3
```

[28]:

```
                   M        Rf          R       R_o     R_for
Date
2015-12-08 -0.651105  0.000744 -15.733033 -0.372548  1.138862
2015-12-09 -0.776909  0.000681  -4.561051 -0.937461  0.468564
2015-12-10  0.224886  0.000639   5.511930 -1.082266 -0.142439
2015-12-11 -1.961387  0.000592  -0.791770 -3.150300 -0.507760
2015-12-14  0.474429  0.000550  -2.903430  1.918598  1.486960
…                …         …          …         …         …
2021-06-23 -0.108387  0.000111   1.754429  0.027377 -0.263916
2021-06-24  0.579443  0.000119   0.865805  0.300589 -0.259467
2021-06-25  0.332506  0.000119  -1.301536  1.017993 -0.100792
2021-06-28  0.231229  0.000111  -2.655021 -1.551473 -0.170286
2021-06-29  0.027730  0.000111  -6.483757  0.095962  0.350804

[1362 rows x 5 columns]
```

[29]:

```
#X & y Variables defined
X_2 = dt3[['R_o','R_for']]
X_2 = sm.add_constant(X_2)
y= dt3['R']-dt3['Rf']
#OLS model
model_1 = sm.OLS(y,X_2).fit()
R= model_1.summary()
print(R)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.005
Method:                 Least Squares   F-statistic:                     4.186
Date:                Thu, 15 Jul 2021   Prob (F-statistic):             0.0154
Time:                        11:57:24   Log-Likelihood:                -4815.1
No. Observations:                1362   AIC:                             9636.
Df Residuals:                    1359   BIC:                             9652.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3999      0.225     -1.775      0.076      -0.842       0.042
R_o            0.1568      0.071      2.219      0.027       0.018       0.295
R_for         -0.6833      0.367     -1.864      0.063      -1.402       0.036
==============================================================================
Omnibus:                     1579.862   Durbin-Watson:                   2.055
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           406830.864
Skew:                           5.434   Prob(JB):                         0.00
```

```
Kurtosis:                      86.968   Cond. No.                        5.19
==============================================================================

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

5 PART 5 Model Selection

5a and b: ADF test of stationarity and unit root (2 pts)+ (2 pts)

[30]:
```
from statsmodels.tsa.stattools import adfuller
#ADF Test under (i) Constant (no linear trend)
X = dta['R'].values
result = adfuller(X, maxlag=None, regression='c', autolag='BIC', store=False,␣
 ↪regresults=False)
print(f'ADF Statistic: {result[0]}')
print( f'n_lags: {result[1]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
        print('\t%s:%.3f'%(key,value))
if result[0] < result [4] ["1%"]:
        print("Reject Ho_ Time Series is then stationary")
else:
        print("Failed to Reject Ho_ Time Series is then non-stationary")
```



```
ADF Statistic: -29.11193109566119
n_lags: 0.0
p-value: 0.0
        1%:-3.435
        5%:-2.864
        10%:-2.568
Reject Ho_ Time Series is then stationary
```

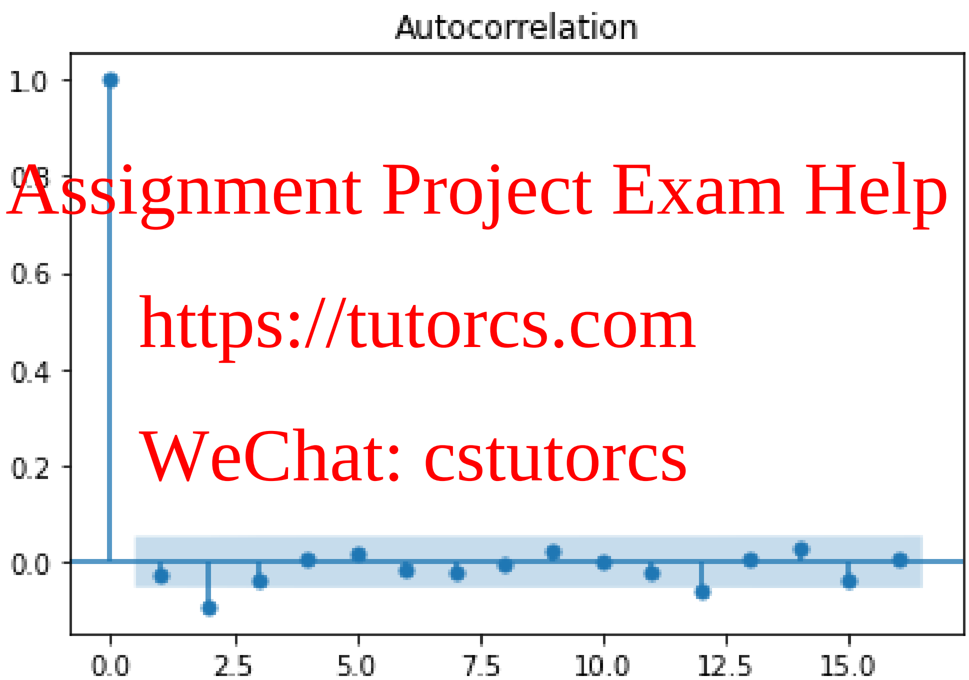
[31]:
```
#ADF Test under (i) Constant (no linear trend)
X = dta['R'].values
result = adfuller(X, maxlag=None, regression='ct', autolag='BIC', store=False,␣
 ↪regresults=False)
print(f'ADF Statistic: {result[0]}')
print(f'n_lags: {result[1]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
        print('\t%s:%.3f'%(key,value))
if result[0] < result [4] ["1%"]:
        print("Reject Ho_ Time Series is then stationary")
else:
        print("Failed to Reject Ho_ Time Series is then non-stationary")
```

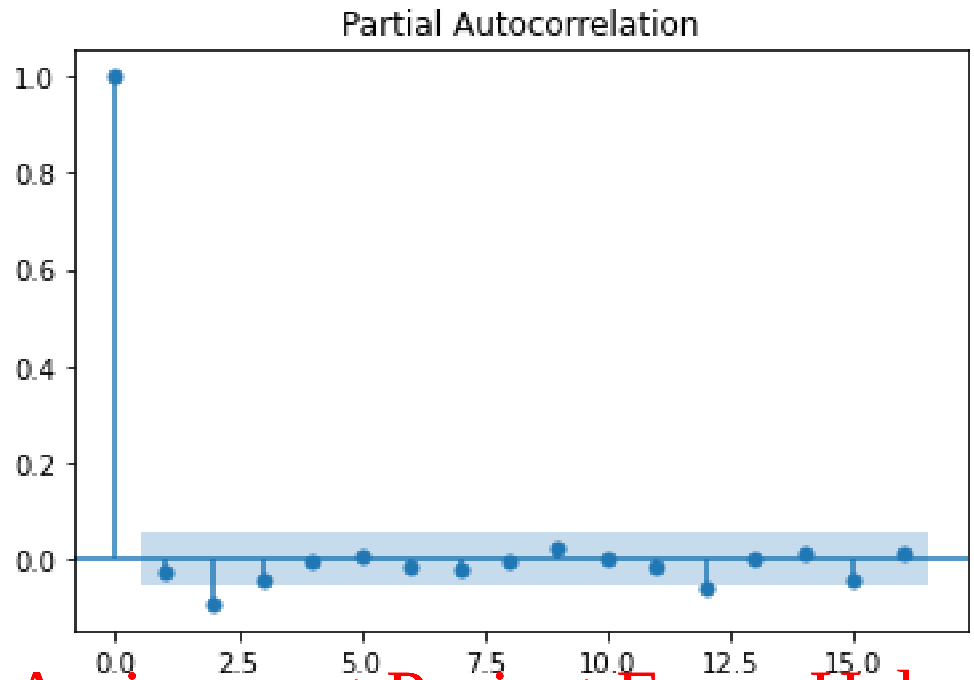
```
ADF Statistic: -29.10154849104698
n_lags: 0.0
p-value: 0.0
        1%:-3.965
        5%:-3.414
        10%:-3.129
Reject Ho_ Time Series is then stationary
```

Correlogram of returns

[32]:

```python
#running ACF and PACF
sm.graphics.tsa.plot_acf(dta.R.values.squeeze(),lags=16)
sm.graphics.tsa.plot_pacf(dta.R.values.squeeze(),lags=16)
plt.show()
```





[33]:
```
# Generating the Q tables
import numpy as np
r,q,p = sm.tsa.acf(dta.R.values.squeeze(), qstat=True)
data = np.c_[range(1,41), r[1:], q, p]
table = pd.DataFrame(data, columns=['lag', "AC", "Q", "Prob(>Q)"])
print (table.set_index('lag'))
```

```
            AC          Q  Prob(>Q)
lag
1.0  -0.029220   1.168840  0.279640
2.0  -0.092812  12.970213  0.001526
3.0  -0.039924  15.155483  0.001688
4.0   0.008045  15.244275  0.004220
5.0   0.014531  15.534191  0.008308
6.0  -0.013871  15.798550  0.014877
7.0  -0.021083  16.409763  0.021625
8.0  -0.002339  16.417290  0.036782
9.0   0.024707  17.257917  0.044827
10.0  0.001471  17.260899  0.068787
11.0 -0.019457  17.783007  0.086754
12.0 -0.062455  23.166382  0.026346
13.0  0.007370  23.241407  0.038869
14.0  0.025730  24.156476  0.043867
15.0 -0.040519  26.427360  0.033764
16.0  0.007294  26.501001  0.047374
```

```
17.0  0.003251  26.515640  0.065565
18.0  0.009370  26.637351  0.086053
19.0 -0.055411  30.896936  0.041436
20.0  0.000034  30.896937  0.056566
21.0 -0.035010  32.599836  0.050847
22.0 -0.014185  32.879591  0.063588
23.0  0.043225  35.479374  0.046582
24.0 -0.001798  35.483877  0.061541
25.0 -0.016080  35.844167  0.073998
26.0 -0.002067  35.850125  0.094460
27.0  0.033873  37.451380  0.086946
28.0 -0.012057  37.654415  0.105117
29.0 -0.035565  39.422262  0.093836
30.0 -0.021269  40.054995  0.103804
31.0 -0.000871  40.056058  0.127781
32.0  0.000791  40.056933  0.155048
33.0 -0.006296  40.112506  0.183985
34.0 -0.010301  40.261380  0.212747
35.0  0.024078  41.075340  0.221651
36.0 -0.049729  44.549981  0.155152
37.0  0.035265  46.298645  0.140558
38.0  0.017985  46.753794  0.155967
39.0  0.024373  47.590337  0.162680
40.0  0.007819  47.676497  0.188311
```



```
C:\Users\rluck\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:657:
FutureWarning: The default number of lags is changing from 40 tomin(int(10 *
np.log10(nobs)), nobs - 1) after 0.12is released. Set the number of lags to an
integer to  silence this warning.
  warnings.warn(
C:\Users\rluck\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:667:
FutureWarning: fft=True will become the default after the release of the 0.12
release of statsmodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(
```

5c. ARMA(1,1): 3 pts

[34]:
```
#ARMA(1,1)
from statsmodels.tsa.arima.model import ARIMA
```

[35]:
```
arima=ARIMA(dta.R.values,exog=None, order=(1, 0, 1), seasonal_order=(0, 0, 0,␣
 ↪0), trend=None, enforce_stationarity=True, enforce_invertibility=True,␣
 ↪concentrate_scale=True)
results = arima.fit()
print(results.summary())
```

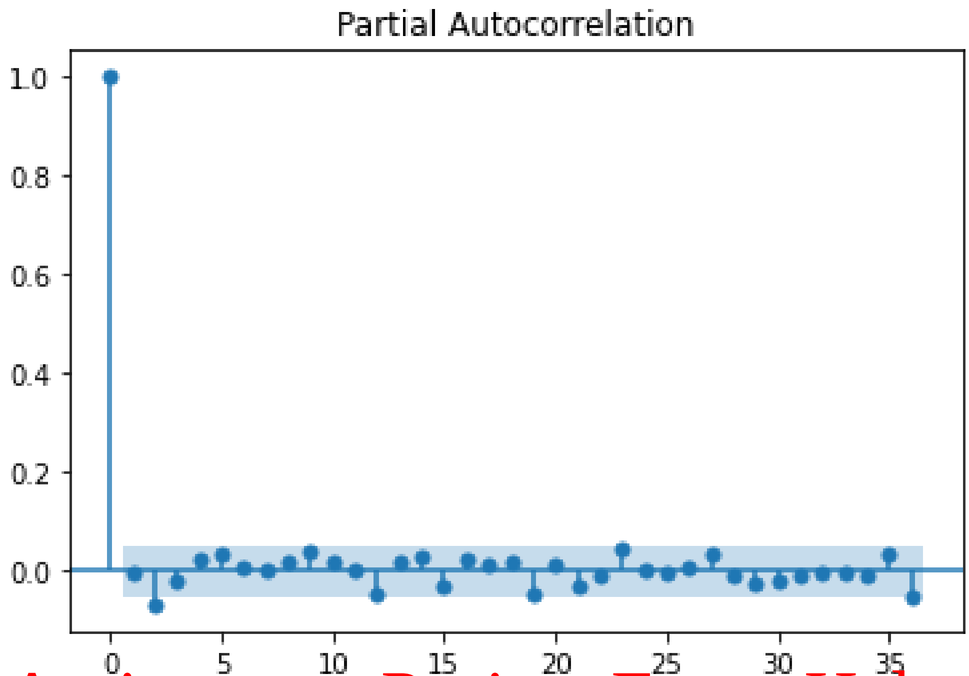
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1366
```

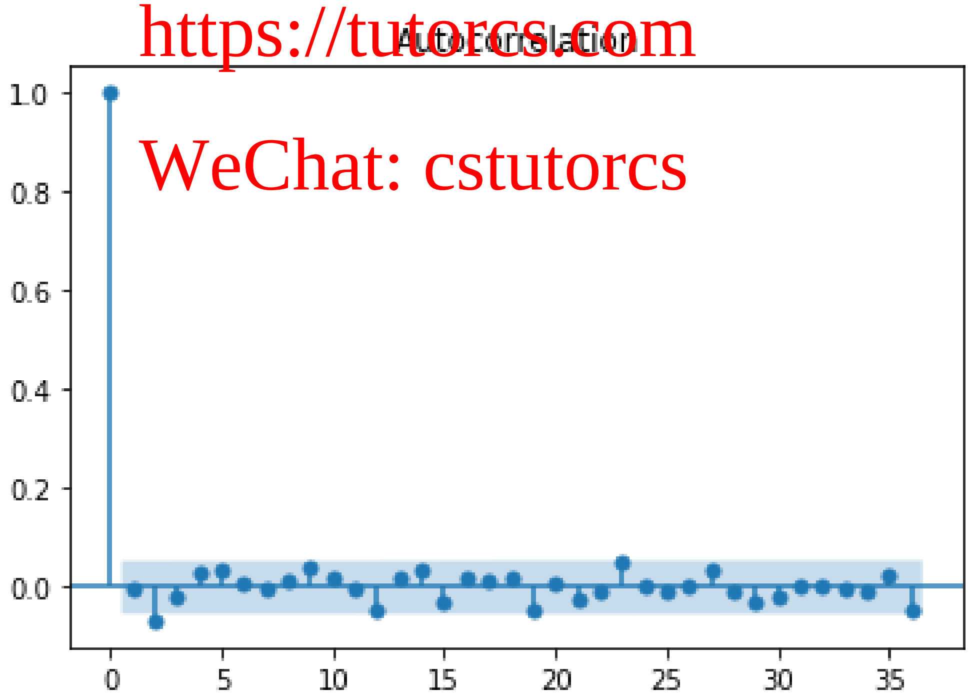
```
Model:                    ARIMA(1, 0, 1)   Log Likelihood               -4818.166
Date:                   Thu, 15 Jul 2021   AIC                           9644.332
Time:                           11:57:24   BIC                           9665.210
Sample:                                0   HQIC                          9652.146
                                  - 1366   Scale                           67.788
Covariance Type:                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3777      0.122     -3.099      0.002      -0.617      -0.139
ar.L1          0.9394      0.012     79.076      0.000       0.916       0.963
ma.L1         -0.9754      0.009   -108.061      0.000      -0.993      -0.958
================================================================================
===
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
419269.56
Prob(Q):                              0.89   Prob(JB):
0.00
Heteroskedasticity (H):               5.19   Skew:
5.54
Prob(H) (two-sided):                  0.00   Kurtosis:
88.11
================================================================================
===

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

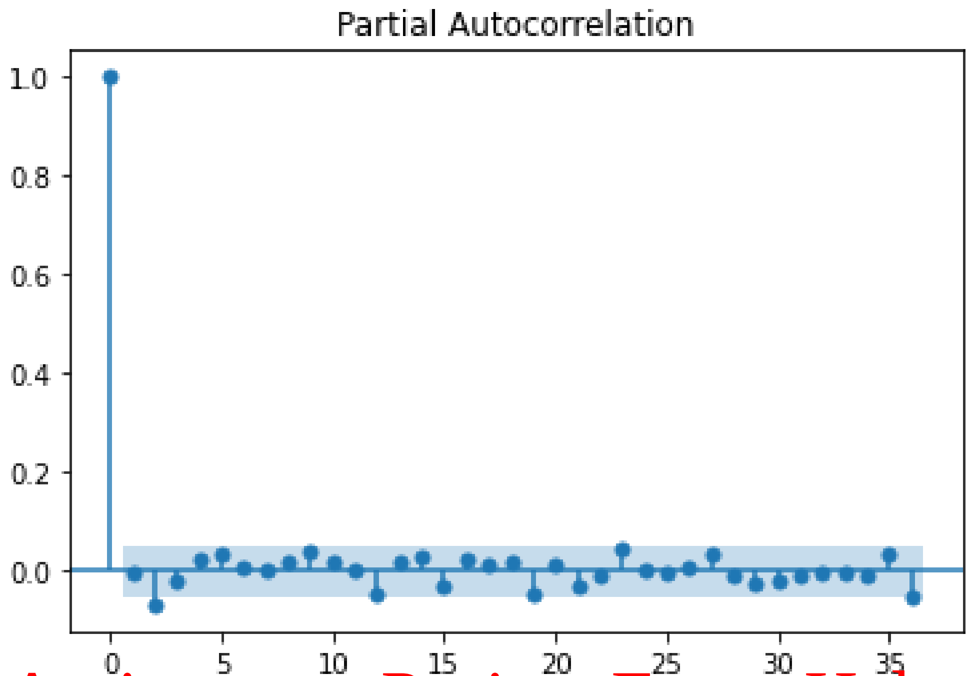

Diagnostic tests of ARMA (1,1)

[36]:
```python
dtr = results.resid
sm.graphics.tsa.plot_acf(dtr.squeeze(),lags=36)
sm.graphics.tsa.plot_pacf(dtr.squeeze(),lags=36)
```

[36]:


Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35



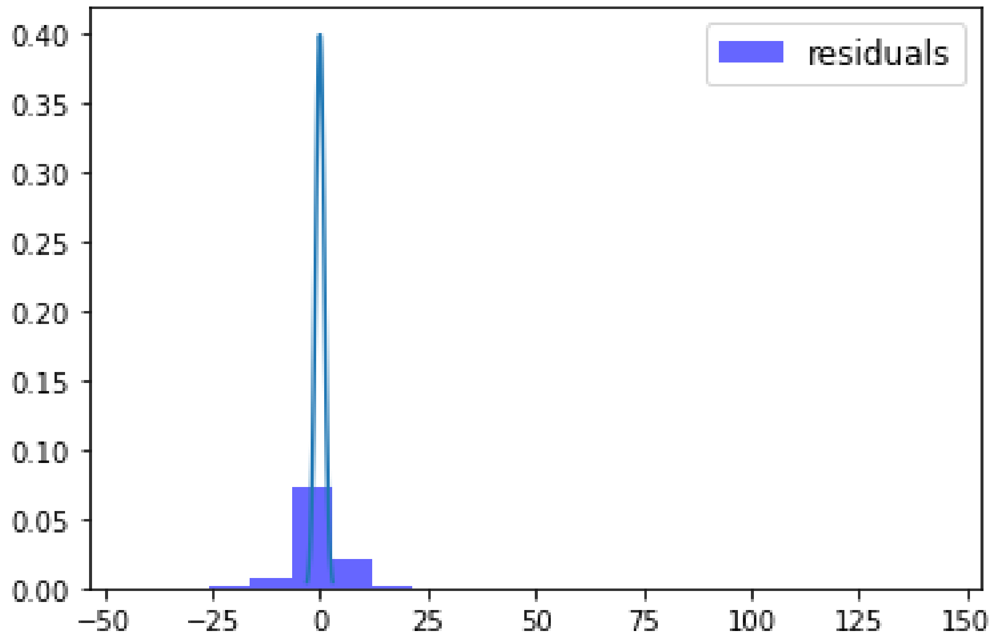
[37]:
```
from scipy import stats
stats.describe(dtr)
```

[37]:
```
DescribeResult(nobs=1366, minmax=(-44.30073924799657, 143.99388556743554),
mean=-0.017913363339356767, variance=67.84326034971882,
skewness=5.54192328655388, kurtosis=85.09471790281705)
```


[38]:
```
JB_resid= stats.jarque_bera(dtr)
JB_resid
```

[38]:
```
Jarque_beraResult(statistic=419132.20862370566, pvalue=0.0)
```

[39]:
```
#Plot histogram for residuals
import math
plt.hist(dtr,bins=20,label='residuals', density=True, alpha=0.6, color='b')
plt.legend(loc='best', fontsize='large')
#plotting the normal distribution curve
mu = 0
variance = 1
sigma = math.sqrt(variance)
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
plt.plot(x, stats.norm.pdf(x, mu, sigma))
plt.show()
```



BDS

[40]:
```
#computing the standardised residuals as residuals from ARMA(1,1) divided by
↪std error of the model
import statistics
var= statistics.variance(results.resid)
se= var**0.5
std_res=results.resid/se
```


[41]:
```
#Computing the BDS stats
import statsmodels.tsa.stattools as stat
bds = stat.bds(std_res,max_dim=2, epsilon=None, distance = 1.5)
print('bds_stat, pvalue:{}'.format(bds))
```

```
bds_stat, pvalue:(array(10.2358456), array(1.36993146e-24))
```

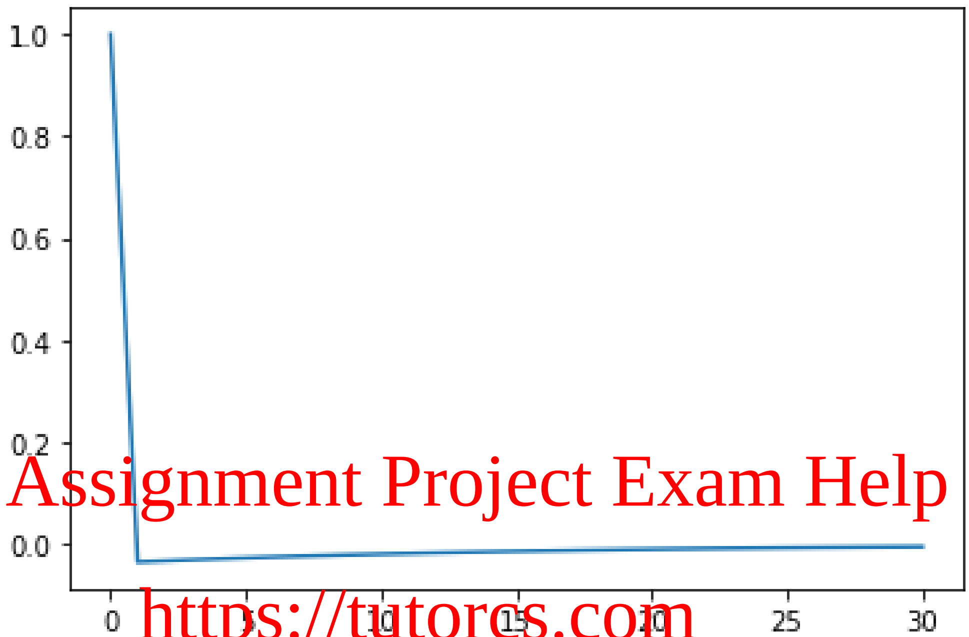
5d: Impulse Response Function (3 pts)

[42]:
```
irf= results.impulse_responses(30)
irf
```

[42]:
```
array([ 1.        , -0.03597641, -0.03379713, -0.03174987, -0.02982662,
       -0.02801987, -0.02632256, -0.02472807, -0.02323017, -0.021823  ,
       -0.02050107, -0.01925922, -0.01809259, -0.01699663, -0.01596706,
       -0.01499985, -0.01409123, -0.01323766, -0.01243578, -0.01168249,
       -0.01097482, -0.01031002, -0.00968549, -0.00909879, -0.00854763,
       -0.00802986, -0.00754345, -0.0070865 , -0.00665724, -0.00625397,
```

```
       -0.00587514])
```

[43]:
```
y = np.array(irf)
plt.plot(y)
plt.show()
```



[ ]:
```
```

[ ]:
```
```