FRE7251 Course Project

May 12, 2020

1 Introduction

1.1 Data

I selected CVS Health Corp (Ticker: CVS) to use MACD strategy to do analysis. I downloaded 5-year daily Adj Close data from 4/21/2011 to 4/20/2015 on Yahoo Finance.

1.2 MACD Strategy

Moving Average Convergence/Divergence (MACD) is a momentum indicator widely popular among practitioners. In MACD, momentum is calculated as the difference between a fast EMA and a slow EMA. An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points:

$$ema(P_t, \beta) = \beta * P_t + (1 - \beta) * ema(P_{t-1}, \beta)$$

$$\beta = \frac{2}{n+1}, \beta < 1$$

starting from short moving average (sma)

The typical fast and slow periods are 12 and 26 days, respectively. The difference

$$MACD_t = ema(P_t, 12) - ema(P_t, 26)$$

is called the *MACD line*. Its exponential smoothing is called the *signal line*:

$$signal\ line = ema(MACD_t, 9)$$

Because the signal line evolves slower than the MACD line, their crossovers can be interpreted as trading signals. Namely, buying opportunity appears when the MACD line crosses from below to above the signal line. On the contrary, crossing the signal line by the MACD line from above can be seen as a selling signal.

2 Bootstrap

A universal method to the data snooping bias is data resampling. Bootstrap is the most popular resampling procedure, which implies creating multiple data sets by randomly drawing the elements of a given sample with replacement.
A more wise approach implies estimating a mathematical models that fits the given sample and

bootstrapping the model's residuals. Typical models used for stock prices are the random walk with drift, the AR models and the GARCH models.
In my project, I used the AR(1) model. The step of Bootstrap is stated below:
1. Calculate the logarithmic returns of stock price, $r_t = log(P_t) - log(P_{t-1})$ 2. Estimate with OLS (Ordinary Least Squares) α and β for

$$r_t = \alpha + \beta r_{t-1} + \epsilon_t$$
$$\epsilon_t = IID(0, \sigma^2)$$
$$\sigma^2 = Var(\epsilon_t)$$

The result of OLS is shown below, we can conclude that both $\alpha = 0.0012$ and $\beta = -0.0598$ are significant because p-value is less than 0.1.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.004
Model:	OLS	Adj. R-squared:	0.003
Method:	Least Squares	F-statistic:	3.591
Date:	Tue, 12 May 2020	Prob (F-statistic):	0.0584
Time:	10:07:04	Log-Likelihood:	3074.6
No. Observations:	1002	AIC:	-6145.
Df Residuals:	1000	BIC:	-6135.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0012	0.000	3.233	0.001	0.000	0.002
x1	-0.0598	0.032	-1.895	0.058	-0.122	0.002

Omnibus:	129.154	Durbin-Watson:	1.992
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1336.359
Skew:	-0.010	Prob(JB):	6.51e-291
Kurtosis:	8.658	Cond. No.	88.7

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

3. Calculate residuals $e_t = r_t - \alpha - \beta r_{t-1}$ 4. Generate new sample with bootstrapped residuals $\check{r}_t = \alpha + \beta \check{r}_{t-1} + \check{e}_t$, where $\check{e}_t$ is randomly selected from e_t every time. 5. Compile new price sample $\check{P}_t = \check{P}_{t-1} \exp(\check{r}_t), \check{P}_0 = P_0$. The new price sample is used to test the robustness of MACD strategy. 6. Repeat step 4 and 5 100 times.

3 Conclusion

3.1 Result

1. According to the performance table below, during the 5 years, under MACD strategy, it has 33 round-trip rates using the actual data. Moreover, the total return in 5 years is 79.05%. The max drawdown is 11.56%.
2. As for the back-testing using bootstrap, the average total return is 62.50 which is close to the empirical data though there are some returns below it. The average maximum drawdown 15.83%. Also, the number percentage of winning trades is 45.00%, which demonstrates that most returns of round-trip trades are negative. All the indictor of bootstrap are similar with the result of empirical data. **Therefore, we can conclude that the bootstrap resampling confirms profitability of your strategy.**
3. In a nutshell, MACS strategy is a satisfied strategy which balances return and volatility. Though it is a litter risky, it is also profitable. To be honest, we may need more tests to ensure

	# of round-trip trades	Total return, %	Winning trades, %	Max drawdown, %
Empirical	33.0	79.05	63.64	11.56
B1	41.0	22.48	41.46	12.56
B2	41.0	44.46	31.71	20.34
B3	44.0	46.94	45.45	13.26
B4	36.0	58.53	50.00	17.98
B5	39.0	74.95	38.46	9.12
B6	38.0	77.95	42.11	12.56
B7	40.0	109.38	57.50	12.52
B8	42.0	5.07	30.95	36.15
B9	33.0	107.19	48.48	20.68
B10	41.0	11.89	39.02	28.78
B11	46.0	34.49	36.96	25.29
B12	39.0	72.93	41.03	20.53
B13	42.0	20.52	40.48	15.40
B14	42.0	37.39	33.33	26.14
B15	41.0	6.03	34.15	17.35
B16	40.0	66.71	47.50	18.63
B17	39.0	64.80	41.03	10.41

B18	41.0	36.16	43.90	21.15
B19	40.0	50.69	37.50	17.93
B20	39.0	65.50	41.03	7.15
B21	36.0	18.84	41.67	19.20
B22	42.0	-4.93	33.33	24.41
B23	36.0	55.26	55.56	16.46
B24	37.0	72.55	32.43	11.03
B25	50.0	17.95	36.00	11.10
B26	34.0	52.23	47.06	21.88
B27	39.0	76.55	48.72	11.21
B28	39.0	29.80	43.59	13.85
B29	36.0	155.58	63.89	10.14
B30	36.0	115.83	47.22	21.82
B31	35.0	46.76	48.57	15.23
B32	38.0	66.18	44.74	18.84
B33	44.0	46.07	45.45	14.75
B34	34.0	126.76	52.94	8.96
B35	39.0	98.85	46.15	10.06
B36	40.0	85.81	52.50	9.58

B37	44.0	1.86	27.27	14.80
B38	48.0	44.84	39.58	9.64
B39	37.0	46.83	51.35	9.09
B40	39.0	62.28	30.77	18.90
B41	43.0	23.71	37.21	17.55
B42	39.0	57.26	43.59	15.22
B43	45.0	11.82	33.33	20.95
B44	50.0	-6.02	28.00	20.73
B45	37.0	120.09	51.35	14.19
B46	38.0	88.14	50.00	11.00
B47	34.0	151.05	61.76	13.88
B48	34.0	59.32	44.12	21.40
B49	41.0	34.82	41.46	13.45
B50	41.0	29.95	36.59	12.84
B51	41.0	36.73	46.34	14.97
B52	41.0	61.14	46.34	13.37
B53	33.0	110.30	48.48	17.12
B54	43.0	-12.95	34.88	24.53
B55	43.0	58.23	51.16	20.24

B56	41.0	69.48	43.90	14.42
B57	44.0	9.57	40.91	9.14
B58	50.0	-5.08	32.00	14.26
B59	41.0	88.31	46.34	12.84
B60	40.0	71.57	50.00	28.53
B61	37.0	99.91	51.35	12.38
B62	36.0	92.74	63.89	13.63
B63	45.0	10.24	42.22	16.65
B64	47.0	-3.33	42.55	19.59
B65	42.0	0.59	38.10	10.47
B66	42.0	24.09	33.33	15.09
B67	42.0	75.32	45.24	14.78
B68	46.0	41.97	28.26	18.38
B69	30.0	196.08	60.00	13.09
B70	32.0	114.85	53.12	11.85
B71	44.0	27.85	47.73	14.37
B72	41.0	35.83	41.46	15.31
B73	42.0	19.34	33.33	26.36
B74	44.0	32.23	40.91	15.53

B75	35.0	155.90	54.29	15.48
B76	43.0	66.12	48.84	13.14
B77	36.0	67.15	38.89	23.38
B78	38.0	84.29	39.47	10.57
B79	39.0	-3.06	33.33	21.61
B80	40.0	47.42	45.00	24.75
B81	43.0	25.93	48.84	12.56
B82	37.0	83.64	43.24	12.48
B83	53.0	7.05	33.96	14.67
B84	36.0	86.64	61.11	11.42
B85	41.0	12.08	36.59	20.78
B86	40.0	15.17	42.50	19.10
B87	40.0	66.72	40.00	12.93
B88	37.0	112.54	62.16	17.64
B89	37.0	42.01	43.24	17.28
B90	35.0	37.49	48.57	20.26
B91	35.0	41.79	45.71	16.48
B92	39.0	83.15	38.46	15.94
B93	36.0	51.94	50.00	7.46
B94	42.0	54.11	40.48	17.63
B95	42.0	91.67	45.24	15.94
B96	41.0	116.52	43.90	11.18
B97	36.0	82.73	44.44	13.27
B98	42.0	85.97	47.62	13.90
B99	40.0	33.22	42.50	15.71
B100	38.0	84.05	52.63	22.72
Average	39.9	57.09	43.91	16.17

the solidity of MACD strategy, for example, taking the transaction fee into consideration.

4. To check if the 100 total return list is a normal distribution, we use Kolmogorov-Smirnov test of scipy.stats.kstest function, which returns the D-value and the p-value. According to the result below, we can conclude that the return serise is a normal distribution.

KS Test

```
1  stats.kstest(total_return_list, 'norm')
```

```
KstestResult(statistic=1.0, pvalue=0.0)
```

5. To check if mean total return differs from zero and the average return, we use Student's T-Test of stats.ttest_1samp, which returns t-statistic and the p-value. As for the test comparing to 0, p-value is 14.72006344 which is much less than 0.05, so it differs from 0 significantly. AS for the test to check if the return list equals to the average return, we also use T-Test. However, the p-value is 0.9651512 which is larger than 0.05. Hence, we cannot conclude that the return of MACD strategy equals to 62.5% siginificantly.

Student's T-Test

```
1  stats.ttest_1samp(total_return_list, 0)
```

```
Ttest_1sampResult(statistic=array([ 14.72006344]), pvalue=array([  8.91787212e-27]))
```

```
1  stats.ttest_1samp(total_return_list[1:-1], 62.5)
```

```
Ttest_1sampResult(statistic=array([-0.0438022]), pvalue=array([ 0.9651512]))
```

3.2 Improvement

1. Though AR(1) model is popular in stock return regression model, the R square of the OLS is quite small, which means it does not take all factor related to stock return into consideration. So we can use more sophisticated model, such as ARIMA and GARCH models.
2. In our model, we cannot short selling and use other products to hedge. Therefore, we may miss some opportunities to make a profit.
3. The main disadvantage of the MACD indicator is that it is subjective to the user. Like many technical indicators, the MACD has settings that can be changed to give almost limitless numbers of variations which means results will always differ from person to person. To be more specific, we set typical fast and slow periods as 12 and 26 days, it may be arbitrary. Therefore, we can use Grid Search to find the optimal parameters which can balance profit and risk efficiently.
4. Though considering transaction fee may decrease the return, we are supposed to consider it. Otherwise, our model is a little unrealistic.

4 Source Code

```
In [1]: import numpy as np
        import pandas as pd
        import statsmodels.api as sm
        from scipy import stats
```

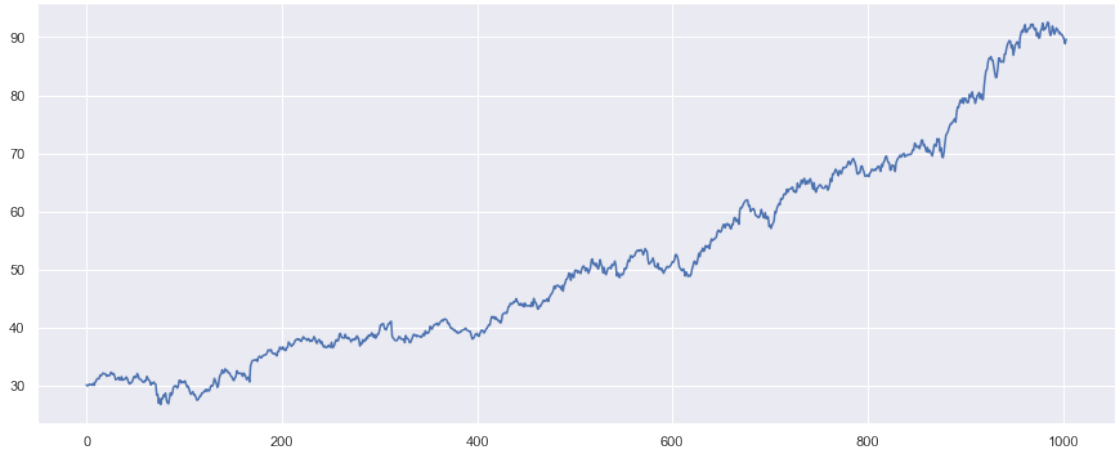
```
import random
random.seed(0)
import math
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [2]:
```
df = pd.read_csv('CVS.csv')
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2011-04-21	36.240002	36.340000	36.130001	36.209999	30.202391	4603300
1	2011-04-25	36.049999	36.220001	35.930000	36.020000	30.043913	3879700
2	2011-04-26	36.139999	36.630001	36.080002	36.090000	30.102303	9300000
3	2011-04-27	36.160000	36.480000	36.049999	36.369999	30.335857	7989700
4	2011-04-28	36.250000	36.320000	36.020000	36.250000	30.235748	8127800

In [3]:
```
plt.figure(figsize=(15,6))
plt.plot(df['Adj Close'])
plt.show()
```



In [4]:
```
def getSignal(df, priceName):
    df['EMA_12day'] = df[[priceName]].ewm(span=12, min_periods=12, adjust=False).mean()
    df['EMA_26day'] = df[[priceName]].ewm(span=26, min_periods=26, adjust=False).mean()
    df['MACD'] = df['EMA_12day'] - df['EMA_26day']
    df['Signal_line'] = df[['MACD']].ewm(span=9, min_periods=9, adjust=False).mean()
    df['Signal'] = np.where((df['MACD']-df['Signal_line'])>0, 'Buy', 'Sell')
    return df
```

In [5]:
```
def Strategy(df, priceName):
    df['Action'] = '-'
    num_trades = 0
```

```
    num_pos_trades = 0
    # return_list is used to calculate the return of each round
    return_list = []
    # close is uesd to save the adj close during trading period
    close = pd.Series(data=None)
    # Suppose we can't buy sell
    # Find the first buy signal
    begin = df[df['Signal']=='Buy'].index.tolist()[0]

    for i in range(begin, len(df)):
        if df['Signal'].iloc[i]=='Buy' and df['Signal'].iloc[i-1]=='Sell':
            df.loc[i, 'Action']='Buy'
            mark = i
        if df['Signal'].iloc[i]=='Sell' and df['Signal'].iloc[i-1]=='Buy':
            df['Action'].iloc[i]='Sell'
            close = pd.Series.append(close, df[priceName].iloc[mark:i+1],
                                     ignore_index=True)
            num_trades += 1
            r = df.loc[i, priceName] / df.loc[mark, priceName] - 1
            return_list.append(r)
            if r > 0:
                num_pos_trades += 1

    return_list = pd.DataFrame(return_list, columns={'return'})
    return df, num_trades, num_pos_trades, return_list, close
```

```
In [6]: def calCumReturn(return_list):
    cum_return = (1+return_list).cumprod()
    total_return_pct = 100*(cum_return.iloc[-1] - 1)
    total_return_pct = total_return_pct[0].round(2)
    return total_return_pct
```

```
In [7]: def calWinTrades(num_pos_trades, num_trades):
    win_trades = num_pos_trades/num_trades
    win_trades_pct = 100*win_trades
    return win_trades_pct
```

```
In [8]: def calDrawDown(df, close):
    df['drawdown'] = pd.DataFrame(close)/pd.DataFrame(close).cummax()-1
    max_drawdown = - df['drawdown'].min()
    max_drawdown_pct = max_drawdown*100
    return max_drawdown_pct
```

4.1 Bootstrap

```
In [9]: # Bootstrap using AR(1)
    # rt is the log retruns = log(Pt)-log(Pt-1)
    rt = np.log(df['Adj Close']).diff()
    X = rt[1:-1]
```

```
Y = rt[2:]
X = sm.add_constant(X)
model = sm.OLS(np.array(Y), np.array(X))
results = model.fit()
alpha = results.params[0]
beta = results.params[1]
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.004
Model:                            OLS   Adj. R-squared:                  0.003
Method:                 Least Squares   F-statistic:                     3.591
Date:                Tue, 12 May 2020   Prob (F-statistic):             0.0584
Time:                        23:00:07   Log-Likelihood:                 3074.6
No. Observations:                1002   AIC:                            -6145.
Df Residuals:                    1000   BIC:                            -6135.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0012      0.000      3.233      0.001       0.000       0.002
x1            -0.0598      0.032     -1.895      0.058      -0.122       0.002
==============================================================================
Omnibus:                      129.154   Durbin-Watson:                   1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1336.359
Skew:                          -0.010   Prob(JB):                    6.51e-291
Kurtosis:                       8.658   Cond. No.                         88.7
==============================================================================

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

```
//anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2389: FutureWarning: Method .p
  return ptp(axis=axis, out=out, **kwargs)
```

In [10]:
```
# Calculate residuals
rt_est = results.predict(sm.add_constant(rt[1:]))
rt_est = rt_est.reset_index(drop=True)
# In textbook, the standardized residuals = (rt - alpha - beta*rtX)/std(residual)
# rt_actual = rt.reset_index(drop=True)
rt_actual = rt[1:].reset_index(drop=True)
# residuals = (rt_actual - rt_est)/(rt_actual - rt_est).std()
residuals = (rt_actual - rt_est)
```

In [11]:
```
def createNewSample(alpha,beta , rt_actual, residuals, df):
    r_sample = [rt_actual[0]]
```

```
            new_price = []
            for i in range(0, len(residuals)):
                index = math.floor(random.random()*len(residuals))
                random_residuals = residuals[index]
                random_return = alpha + beta *r_sample[i-1] + random_residuals
                r_sample.append(random_return)
            new_price.append(df['Adj Close'].iloc[0])
            for i in range(1, len(residuals)+1):
                new_price.append(new_price[i-1] * np.exp(r_sample[i-1]))
            return pd.DataFrame(data=new_price, columns={'Price_sample'})
```

In [12]:
```
# Calculate using empirical data
data = getSignal(df, 'Adj Close')
data, num_trades, num_pos_trades, return_list, close = Strategy(data, 'Adj Close')
total_return_pct = calCumReturn(return_list)
win_trades_pct = calWinTrades(num_pos_trades, num_trades)
max_drawdown_pct = calDrawDown(data, close)
result = pd.DataFrame(data=[[num_trades, total_return_pct, win_trades_pct, max_drawdown

# 100 times bootstrap
total_return_list = [total_return_pct]
for i in range(100):
    Price_sample = createNewSample(alpha,beta, rt_actual, residuals, df)
    data = getSignal(Price_sample, 'Price_sample')
    data, num_trades, num_pos_trades, return_list, close = Strategy(Price_sample, 'Pric
    total_return_pct = calCumReturn(return_list)
    total_return_list.append(total_return_pct)
    win_trades_pct = calWinTrades(num_pos_trades, num_trades)
    max_drawdown_pct = calDrawDown(data, close)

    Bresult = pd.DataFrame(data=[[num_trades, total_return_pct, win_trades_pct, max_dra
    result = pd.concat([result, Bresult], axis=0)
```

```
//anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:190: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  self._setitem_with_indexer(indexer, value)
```

In [13]:
```
result.loc['Average'] = result.mean()
result = result.rename(columns={0: '# of round-trip trades', 1: 'Total return, %', 2: '
result = result.round(decimals=2)
pd.set_option('display.max_rows', None)
result
```

Out[13]:

	# of round-trip trades	Total return, %	Winning trades, %	\
Empirical	33.0	79.05	63.64	
B1	41.0	22.48	41.46	

B2	41.0	44.46	31.71
B3	44.0	46.94	45.45
B4	36.0	58.53	50.00
B5	39.0	74.95	38.46
B6	38.0	77.95	42.11
B7	40.0	109.38	57.50
B8	42.0	5.07	30.95
B9	33.0	107.19	48.48
B10	41.0	11.89	39.02
B11	46.0	34.49	36.96
B12	39.0	72.93	41.03
B13	42.0	20.52	40.48
B14	42.0	37.39	33.33
B15	41.0	6.03	34.15
B16	40.0	66.71	47.50
B17	39.0	64.80	41.03
B18	41.0	36.16	43.90
B19	40.0	50.69	37.50
B20	39.0	65.50	41.03
B21	36.0	18.84	41.67
B22	42.0	-4.93	33.33
B23	36.0	55.26	55.56
B24	37.0	72.55	32.43
B25	50.0	17.95	36.00
B26	34.0	52.23	47.06
B27	39.0	76.55	48.72
B28	39.0	29.80	43.59
B29	36.0	155.58	63.89
B30	36.0	115.83	47.22
B31	35.0	46.76	48.57
B32	38.0	66.18	44.74
B33	44.0	46.07	45.45
B34	34.0	126.76	52.94
B35	39.0	98.85	46.15
B36	40.0	85.81	52.50
B37	44.0	1.86	27.27
B38	48.0	44.84	39.58
B39	37.0	46.83	51.35
B40	39.0	62.28	30.77
B41	43.0	23.71	37.21
B42	39.0	57.26	43.59
B43	45.0	11.82	33.33
B44	50.0	-6.02	28.00
B45	37.0	120.09	51.35
B46	38.0	88.14	50.00
B47	34.0	151.05	61.76
B48	34.0	59.32	44.12
B49	41.0	34.82	41.46

B50	41.0	29.95	36.59
B51	41.0	36.73	46.34
B52	41.0	61.14	46.34
B53	33.0	110.30	48.48
B54	43.0	-12.95	34.88
B55	43.0	58.23	51.16
B56	41.0	69.48	43.90
B57	44.0	9.57	40.91
B58	50.0	-5.08	32.00
B59	41.0	88.31	46.34
B60	40.0	71.57	50.00
B61	37.0	99.91	51.35
B62	36.0	92.74	63.89
B63	45.0	10.24	42.22
B64	47.0	-3.33	42.55
B65	42.0	0.59	38.10
B66	42.0	24.09	33.33
B67	42.0	75.32	45.24
B68	46.0	41.97	28.26
B69	30.0	196.08	60.00
B70	32.0	114.85	53.12
B71	44.0	27.85	47.73
B72	41.0	35.83	41.46
B73	42.0	19.34	33.33
B74	44.0	32.23	40.91
B75	35.0	155.90	54.29
B76	43.0	66.12	48.84
B77	36.0	67.15	38.89
B78	38.0	84.29	39.47
B79	39.0	-3.06	33.33
B80	40.0	47.42	45.00
B81	43.0	25.93	48.84
B82	37.0	83.64	43.24
B83	53.0	7.05	33.96
B84	36.0	86.64	61.11
B85	41.0	12.08	36.59
B86	40.0	15.17	42.50
B87	40.0	66.72	40.00
B88	37.0	112.54	62.16
B89	37.0	42.01	43.24
B90	35.0	37.49	48.57
B91	35.0	41.79	45.71
B92	39.0	83.15	38.46
B93	36.0	51.94	50.00
B94	42.0	54.11	40.48
B95	42.0	91.67	45.24
B96	41.0	116.52	43.90
B97	36.0	82.73	44.44

B98	42.0	85.97	47.62
B99	40.0	33.22	42.50
B100	38.0	84.05	52.63
Average	39.9	57.09	43.91

	Max drawdown, %
Empirical	11.56
B1	12.56
B2	20.34
B3	13.26
B4	17.98
B5	9.12
B6	12.56
B7	12.52
B8	36.15
B9	20.68
B10	28.78
B11	25.29
B12	20.53
B13	15.40
B14	26.14
B15	17.35
B16	18.63
B17	10.41
B18	21.15
B19	17.93
B20	7.15
B21	19.20
B22	24.41
B23	16.46
B24	11.03
B25	11.10
B26	21.88
B27	11.21
B28	13.85
B29	10.14
B30	21.82
B31	15.23
B32	18.84
B33	14.75
B34	8.96
B35	10.06
B36	9.58
B37	14.80
B38	9.64
B39	9.09
B40	18.90
B41	17.55

B42	15.22
B43	20.95
B44	20.73
B45	14.19
B46	11.00
B47	13.88
B48	21.40
B49	13.45
B50	12.84
B51	14.97
B52	13.37
B53	17.12
B54	24.53
B55	20.24
B56	14.42
B57	9.14
B58	14.26
B59	12.84
B60	28.53
B61	12.38
B62	13.63
B63	16.65
B64	19.59
B65	10.47
B66	15.09
B67	14.78
B68	18.38
B69	13.09
B70	11.85
B71	14.37
B72	15.31
B73	26.36
B74	15.53
B75	15.48
B76	13.14
B77	23.38
B78	10.57
B79	21.61
B80	24.75
B81	12.56
B82	12.48
B83	14.67
B84	11.42
B85	20.78
B86	19.10
B87	12.93
B88	17.64
B89	17.28

```
B90                 20.26
B91                 16.48
B92                 15.94
B93                  7.46
B94                 17.63
B95                 15.94
B96                 11.18
B97                 13.27
B98                 13.90
B99                 15.71
B100                22.72
Average             16.17
```

5 P/L histogram

In [14]:
```
total_return_list = pd.DataFrame(total_return_list, columns={'total return'})

total_return_list = total_return_list.values
m = total_return_list.mean()
std = total_return_list.std()
normal_dist = np.random.normal(m, std, 100)
plt.figure(figsize=(15,6))
plt.title('Returns P/L %', fontsize=18)
plt.xlabel('Return of Empirical and Bootstraps, %', fontsize=18)
plt.ylabel('count', fontsize=18)
plt.hist([total_return_list, normal_dist], density = False, label =['CVS','Normal'], co
plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x12a250cf8>

5.1 KS Test

```
In [15]: stats.kstest(total_return_list, 'norm')
```

```
Out[15]: KstestResult(statistic=1.0, pvalue=0.0)
```

5.2 Student's T-Test

```
In [16]: stats.ttest_1samp(total_return_list, 0)
```

```
Out[16]: Ttest_1sampResult(statistic=array([14.16907109]), pvalue=array([1.18943035e-25]))
```

```
In [17]: stats.ttest_1samp(total_return_list[1:-1], 62.5)
```

```
Out[17]: Ttest_1sampResult(statistic=array([-1.4407649]), pvalue=array([0.15283855]))
```