In [1]:

```
import os
import datetime
import numpy as np
import pandas as pd


try:
    import matplotlib.pyplot as plt
    """
    the output of plotting commands is displayed inline within frontends like the Jupyt
er notebook,
    directly below the code cell that produced it. The resulting plots will then also b
e stored in the notebook document.
    """
    %matplotlib inline
except:
    pass
```

merge and pre-process data

Nasdaq.com
NYSE
Yahoo

In [2]:

```
files = os.listdir('data')
print("There are {} csv files:".format(len(files)))
for name in files:
    print(name[:-4])
```

```
There are 31 csv files:
AAPL
AXP
BA
CAT
CSCO
CVX
DIS
DWDP
GS
HD
IBM
INTC
JNJ
JPM
KO
MCD
MMM
MRK
MSFT
NKE
PFE
PG
TRV
UNH
UTX
V
VZ
WBA
WMT
XOM
^DJI
```

In [3]:

```
dow_jones = pd.read_csv('data/^DJI.csv')
aapl = pd.read_csv('data/AAPL.csv')
```

In [4]:

```
print("dimensions of Dow Jones Industrial Average: ", dow_jones.shape)
dow_jones.head()
```

```
dimensions of Dow Jones Industrial Average:  (2518, 7)
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volum
0	2008-11-24	8048.089844	8599.019531	8048.089844	8443.389648	8443.389648	4918900C
1	2008-11-25	8445.139648	8607.379883	8281.459961	8479.469727	8479.469727	3740200C
2	2008-11-26	8464.490234	8726.610352	8311.169922	8726.610352	8726.610352	2839200C
3	2008-11-28	8724.700195	8831.349609	8672.690430	8829.040039	8829.040039	1555100C
4	2008-12-01	8826.889648	8827.049805	8141.359863	8149.089844	8149.089844	3210100C

Index

In [5]:

```
print("dimensions of Dow Jones components: ", aapl.shape)
aapl.tail()
```

```
dimensions of Dow Jones components:  (2518, 6)
```

Out[5]:

AAPL

	date	close	volume	open	high	low
2513	12/1/2008	12.7043	230862519	13.0429	13.1814	12.7033
2514	11/28/2008	13.2386	75301403	13.5286	13.5371	13.1229
2515	11/26/2008	13.5714	224858127	12.8457	13.6071	12.8357
2516	11/25/2008	12.9714	308776603	13.5186	13.5300	12.5943
2517	11/24/2008	13.2786	360468748	12.1729	13.5414	12.1200

In [6]:

```
aapl['Date'] = pd.to_datetime(aapl.date)
aapl = aapl.sort_values(by=['Date'])
aapl.head()
```

Out[6]:

	date	close	volume	open	high	low	Date
2517	11/24/2008	13.2786	360468748	12.1729	13.5414	12.1200	2008-11-24
2516	11/25/2008	12.9714	308776603	13.5186	13.5300	12.5943	2008-11-25
2515	11/26/2008	13.5714	224858127	12.8457	13.6071	12.8357	2008-11-26
2514	11/28/2008	13.2386	75301403	13.5286	13.5371	13.1229	2008-11-28
2513	12/1/2008	12.7043	230862519	13.0429	13.1814	12.7033	2008-12-01

In [7]:

```
asset_prices = pd.DataFrame(columns=['AAPL','AXP','BA','CAT','CSCO','CVX','DIS','DWDP',
'GS','HD',
                                     'IBM','INTC','JNJ','JPM','KO','MCD','MMM','MRK','MS
FT','NKE',
                                     'PFE','PG','TRV','UNH','UTX','V','VZ','WBA','WMT',
'XOM',
                                     'DJI'],
                            index=dow_jones.Date)
# special cases
asset_prices.AAPL = np.array(aapl.close)
asset_prices.DJI = np.array(dow_jones['Adj Close'])
dwdp = pd.read_csv('data/DWDP.csv')
asset_prices.DWDP = np.array(dwdp['Adj Close'])
mrk = pd.read_csv('data/MRK.csv')
asset_prices.MRK = np.array(mrk['Adj Close'])
```

In [8]:

```
files_left = sorted(list(set(files) - set(['AAPL.csv', 'DWDP.csv', 'MRK.csv', '^DJI.cs
v'])))
print(len(files_left), files_left)
```

```
27 ['AXP.csv', 'BA.csv', 'CAT.csv', 'CSCO.csv', 'CVX.csv', 'DIS.csv', 'GS.
csv', 'HD.csv', 'IBM.csv', 'INTC.csv', 'JNJ.csv', 'JPM.csv', 'KO.csv', 'MC
D.csv', 'MMM.csv', 'MSFT.csv', 'NKE.csv', 'PFE.csv', 'PG.csv', 'TRV.csv',
'UNH.csv', 'UTX.csv', 'V.csv', 'VZ.csv', 'WBA.csv', 'WMT.csv', 'XOM.csv']
```

In [9]:

```
for name in files_left:
    df = pd.read_csv('data/'+name)
    df['Date'] = pd.to_datetime(df.date)
    df = df.sort_values(by=['Date'])
    asset_prices[name[:-4]] = np.array(df.close)
```

In [28]:

```
#asset_prices.to_csv('data/close.csv')
asset_prices.head()
```

Out[28]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	DWDP	GS	HD	...	
Date												
2008-11-24	13.2786	21.18	40.75	36.34	16.40	74.30	22.20	13.190079	67.42	21.42	...	6
2008-11-25	12.9714	21.37	40.18	37.27	15.42	76.53	22.03	13.146806	71.78	22.25	...	6
2008-11-26	13.5714	22.30	41.28	39.33	16.39	79.93	22.50	13.644410	76.50	23.55	...	6
2008-11-28	13.2386	23.31	42.63	40.99	16.54	79.01	22.52	13.377581	78.99	23.11	...	6
2008-12-01	12.7043	19.64	39.88	36.58	14.96	72.02	20.33	12.930458	65.76	21.21	...	6

... DJI

5 rows × 31 columns

2518 x 31

In [29]:

```
asset_prices.tail()
```

Out[29]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	DWDP	GS	I
Date										
2018-11-16	193.53	109.46	335.95	129.96	46.35	119.06	116.19	59.189999	202.12	177.
2018-11-19	185.86	108.25	320.94	125.98	45.75	119.42	115.42	57.799999	198.22	173.
2018-11-20	176.98	106.09	317.70	122.27	44.49	116.10	111.87	56.369999	191.34	169.
2018-11-21	176.78	106.50	317.32	123.87	44.89	117.57	113.03	56.970001	192.60	169.
2018-11-23	172.29	105.74	312.32	122.32	44.54	113.60	112.08	56.430000	189.10	168.

5 rows × 31 columns

calculate log-returns

window = 1

In [11]:

```
np.log(asset_prices.shift(1)).head()
```

Out[11]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	DW
Date								
2008-11-24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-11-25	2.586154	3.053057	3.707456	3.592919	2.797281	4.308111	3.100092	2.579
2008-11-26	2.562747	3.061988	3.693369	3.618189	2.735665	4.337683	3.092405	2.576
2008-11-28	2.607965	3.104587	3.720378	3.671988	2.796671	4.381151	3.113515	2.613
2008-12-01	2.583137	3.148882	3.752558	3.713328	2.805782	4.369574	3.114404	2.593

5 rows × 31 columns

In [12]:

```
asset_returns = (np.log(asset_prices) - np.log(asset_prices.shift(1))).iloc[1:,:]
asset_returns.head()
```

Out[12]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	
Date								
2008-11-25	-0.023407	0.008931	-0.014086	0.025270	-0.061616	0.029572	-0.007687	-
2008-11-26	0.045218	0.042599	0.027009	0.053799	0.061006	0.043468	0.021110	(
2008-11-28	-0.024828	0.044296	0.032180	0.041341	0.009110	-0.011577	0.000888	-
2008-12-01	-0.041196	-0.171314	-0.066683	-0.113826	-0.100402	-0.092631	-0.102306	-
2008-12-02	0.039034	0.055460	0.020353	0.038085	0.023779	0.047718	0.054093	(

5 rows × 31 columns

In [13]:

```
len(asset_returns)
```

Out[13]:

2517 x 31

standarize the data

In [14]:

```
from sklearn.preprocessing import StandardScaler

# standardize the log-returns: mean = 0, sigma = 1
standardized_asset_returns = StandardScaler().fit_transform(asset_returns.values)
standardized_asset_returns = pd.DataFrame(data=standardized_asset_returns,
                                          index=asset_returns.index,
                                          columns=asset_returns.columns.values)
standardized_asset_returns.head()
```

Out[14]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS
Date							
2008-11-25	-1.445093	0.396634	-0.889603	1.265145	-3.665218	2.054946	-0.548920
2008-11-26	2.615029	2.007100	1.564708	2.721270	3.582246	3.026156	1.348636
2008-11-28	-1.529168	2.088282	1.873554	2.085400	0.514995	-0.820876	0.016158
2008-12-01	-2.497597	-8.225179	-4.030817	-5.834290	-5.957616	-6.485605	-6.783712
2008-12-02	2.249151	2.622298	1.167218	1.919213	1.381987	3.323185	3.522003

5 rows × 31 columns

In [15]:

```
print("mean of AAPL before and after standardization: ", asset_returns.AAPL. mean(), st
andardized_asset_returns.AAPL.mean())
print("std of AAPL before and after standardization: ", asset_returns.AAPL. std(), stan
dardized_asset_returns.AAPL.std())
```

```
mean of AAPL before and after standardization:  0.0010182858111799883 -4.4
10897992153979e-18
std of AAPL before and after standardization:  0.01690543258877368 1.00019
87083973913
```

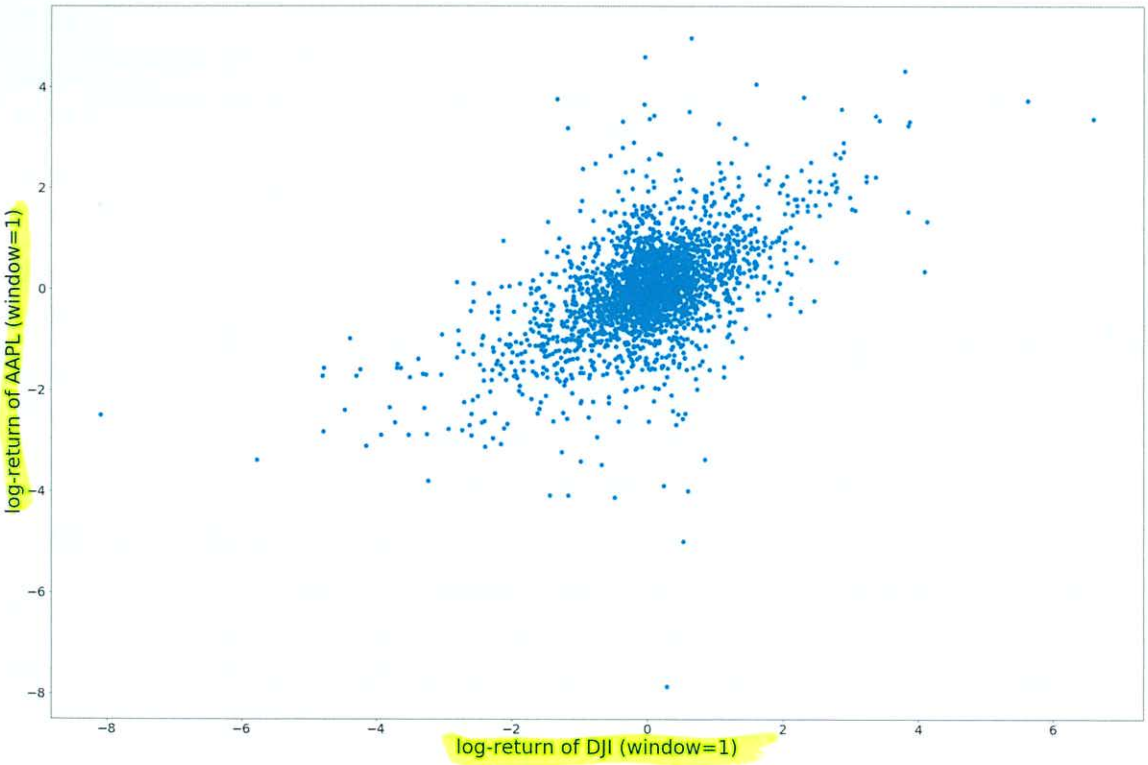
$\boxed{\text{mean} = 0, \ \sigma = 1}$

insights of data by plotting correlations

In [16]:

```
fig, ax_list = plt.subplots(1, 1, figsize=(30,20))
plt.suptitle("Correlation of AAPL returns and DJI returns", fontsize = 38, fontweight=
'bold')
ax_list.scatter(standardized_asset_returns['DJI'].values.reshape((2517, 1)), standardiz
ed_asset_returns['AAPL'].values.reshape((2517, 1)))
plt.xlabel("log-return of DJI (window=1)", fontsize=30)
plt.ylabel("log-return of AAPL (window=1)", fontsize=30)
ax_list.tick_params(labelsize=20)
plt.show()
```

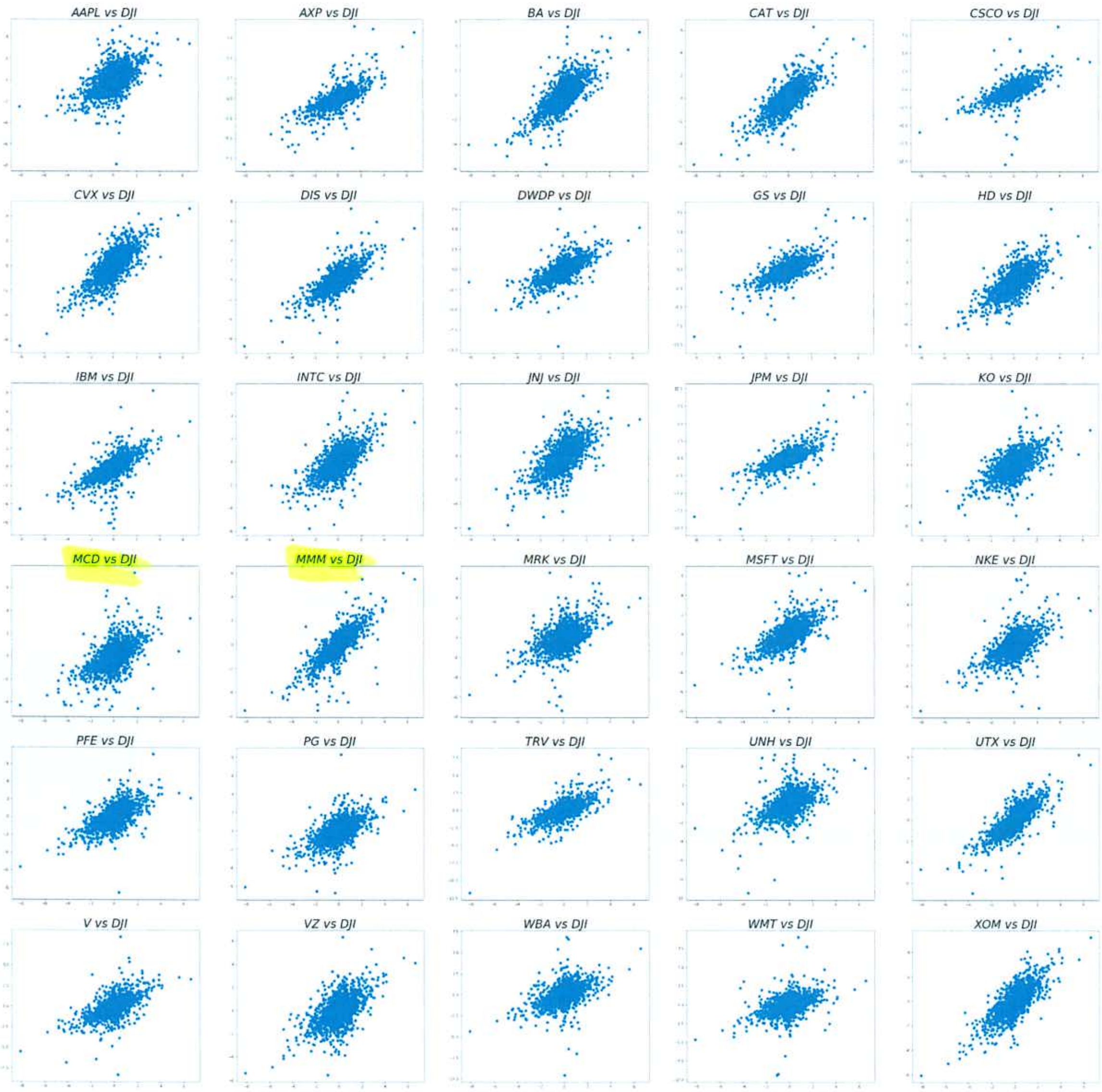
Correlation of AAPL returns and DJI returns



In [17]:

```
fig, ax_list = plt.subplots(6, 5, figsize=(50,50))
plt.suptitle("Correlation of component returns and DJI returns", fontsize = 48, fontwei
ght='bold')

dji = standardized_asset_returns['DJI'].values.reshape((2517, 1))
column_names = standardized_asset_returns.columns.values
for i in range(standardized_asset_returns.shape[1]-1):
    ax_list[int(i/5)][i%5].set_title('{} vs DJI'.format(column_names[i]), fontstyle='it
alic', fontsize = 30)
    ax_list[int(i/5)][i%5].scatter(dji, standardized_asset_returns.iloc[:,i].values.res
hape((2517, 1)))
plt.show()
```

Correlation of component returns and DJI returns



split into training and testing data set

In [18]:

```
train_percentage = 0.7

df_train = standardized_asset_returns.iloc[:int(standardized_asset_returns.shape[0]*0.7
), :]
df_test = standardized_asset_returns.iloc[int(standardized_asset_returns.shape[0]*0.7
):, :]
print("dimensions of training set: ", df_train.shape)
print("dimensions of testing set: ", df_test.shape)
df_train.tail()
```

(70%)

```
dimensions of training set:  (1761, 31)
dimensions of testing set:  (756, 31)
```

Out[18]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	
Date								
2015-11-17	-0.312104	-0.399105	0.634227	-0.754911	0.020647	-0.333504	0.076877	-
2015-11-18	1.784146	0.663958	0.844407	0.654906	0.656030	0.888338	1.088354	'
2015-11-19	0.686617	0.359009	0.337088	-0.242824	0.518882	-1.065635	0.274770	(
2015-11-20	0.198201	-0.241454	0.015671	0.785331	0.406858	-1.426396	0.708230	-
2015-11-23	-0.833974	-0.149597	-0.473567	-0.110779	-0.324355	0.769010	-0.400072	-

5 rows × 31 columns

building CAMP linear regression model

In [19]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

tickers = standardized_asset_returns.columns.values[:-1]
lr = LinearRegression()
alphas = [0.] * len(tickers)
betas = [0.] * len(tickers)
r2_in_sample = [0.] * len(tickers)
r2_out_sample = [0.] * len(tickers)
tickers
```

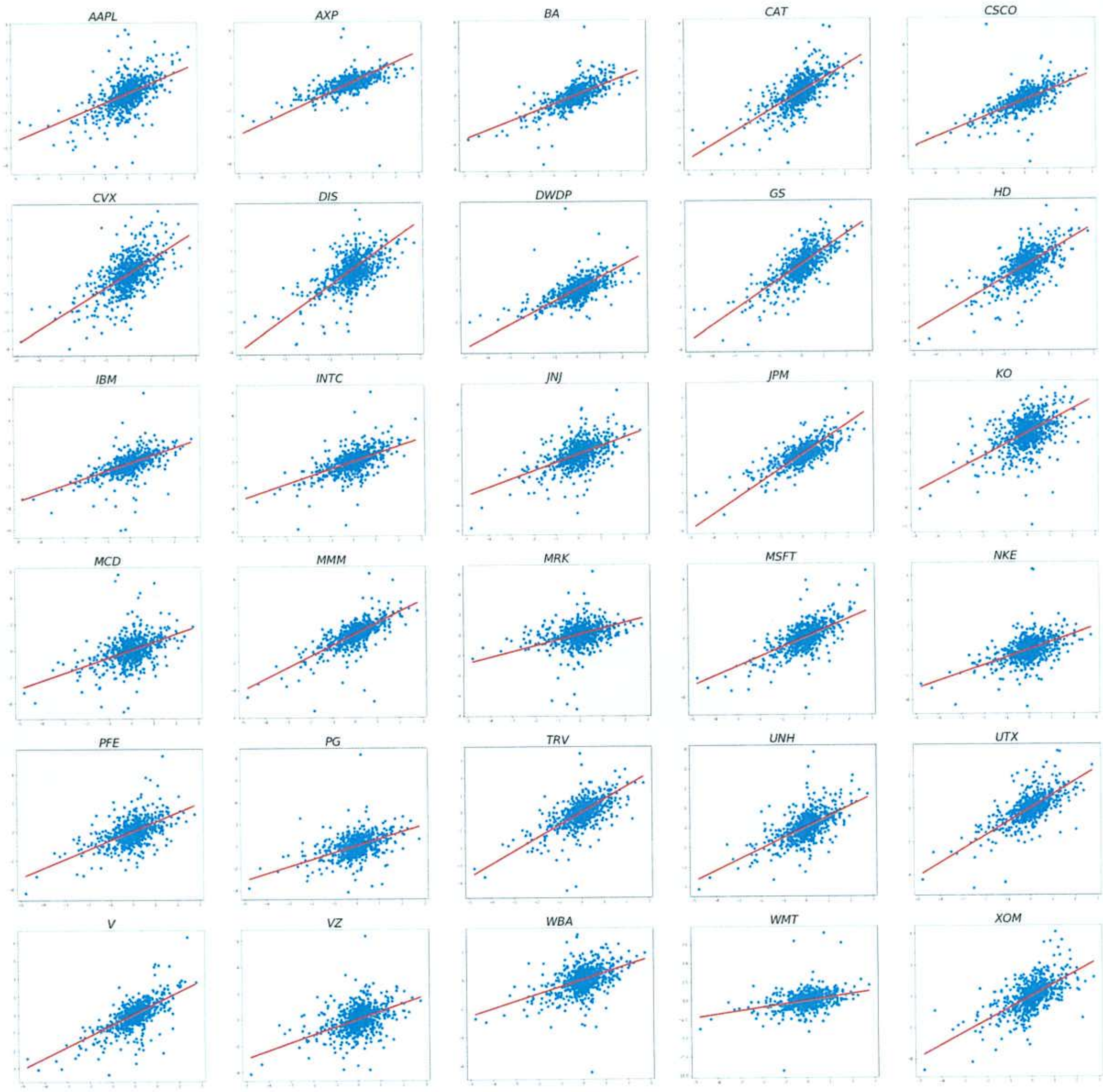
Out[19]:

```
array(['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'CVX', 'DIS', 'DWDP', 'GS',
       'HD', 'IBM', 'INTC', 'JNJ', 'JPM', 'KO', 'MCD', 'MMM', 'MRK',
       'MSFT', 'NKE', 'PFE', 'PG', 'TRV', 'UNH', 'UTX', 'V', 'VZ', 'WBA',
       'WMT', 'XOM'], dtype=object)
```

In [20]:

```
fig, ax_list = plt.subplots(6, 5, figsize=(50,50))
plt.suptitle("Regression of DJI component returns of testing set", fontsize = 48, fontw
eight='bold')

x_axis = df_test['DJI'].values.reshape((756, 1))
for i, stock in enumerate(tickers):
    lr.fit(df_train['DJI'].values.reshape((1761, 1)), df_train[stock].values.reshape((1
761, 1)))
    alphas[i] = lr.intercept_[0]
    betas[i] = lr.coef_[0][0]

    predictions = lr.predict(df_train['DJI'].values.reshape((1761, 1)))
    r2_in_sample[i] = r2_score(df_train[stock], predictions)

    predictions = lr.predict(df_test['DJI'].values.reshape((756, 1)))
    r2_out_sample[i] = r2_score(df_test[stock], predictions)

    ax_list[int(i/5)][i%5].set_title('{}'.format(stock), fontstyle='italic', fontsize =
 30)
    ax_list[int(i/5)][i%5].scatter(x_axis, df_test[stock].values.reshape((756, 1)))
    ax_list[int(i/5)][i%5].plot(x_axis, predictions, color='red', linewidth=3)

plt.show()
# R2 score: https://en.wikipedia.org/wiki/Coefficient_of_determination
```

Regression of DJI component returnsof testing set



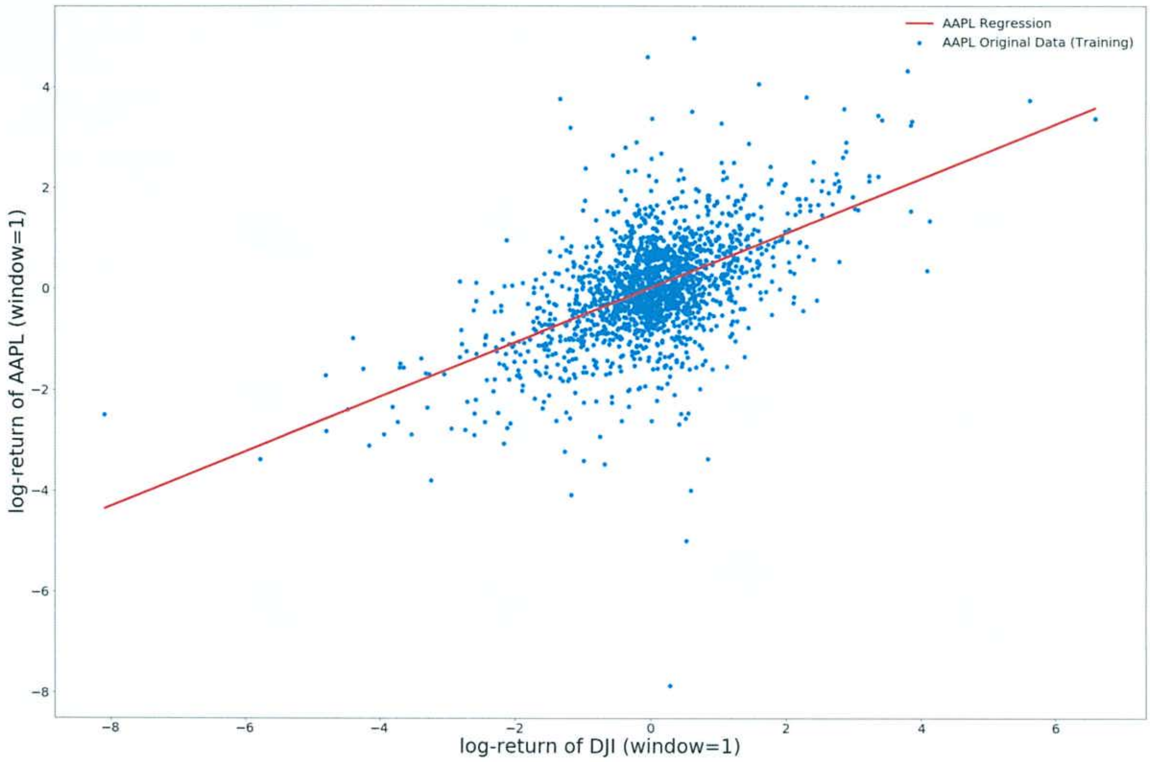
In [21]:

```
lr.fit(df_train['DJI'].values.reshape((1761, 1)), df_train['AAPL'].values.reshape((1761
, 1)))
alpha = lr.intercept_[0]
beta = lr.coef_[0][0]
predictions = lr.predict(df_train['DJI'].values.reshape((1761, 1)))
```

In [22]:

```
fig, ax_list = plt.subplots(1, 1, figsize=(30,20))
plt.suptitle("CAMP of the training set", fontsize = 38, fontweight='bold')
ax_list.scatter(df_train['DJI'].values.reshape((1761, 1)), df_train['AAPL'].values.resh
ape((1761, 1)), label='AAPL Original Data (Training)')
ax_list.plot(df_train['DJI'].values.reshape((1761, 1)), predictions, label='AAPL Regres
sion', color='red', linewidth=3)
plt.xlabel("log-return of DJI (window=1)", fontsize=30)
plt.ylabel("log-return of AAPL (window=1)", fontsize=30)
ax_list.tick_params(labelsize=20)
plt.legend(fontsize=20)
plt.show()
```

CAMP of the training set



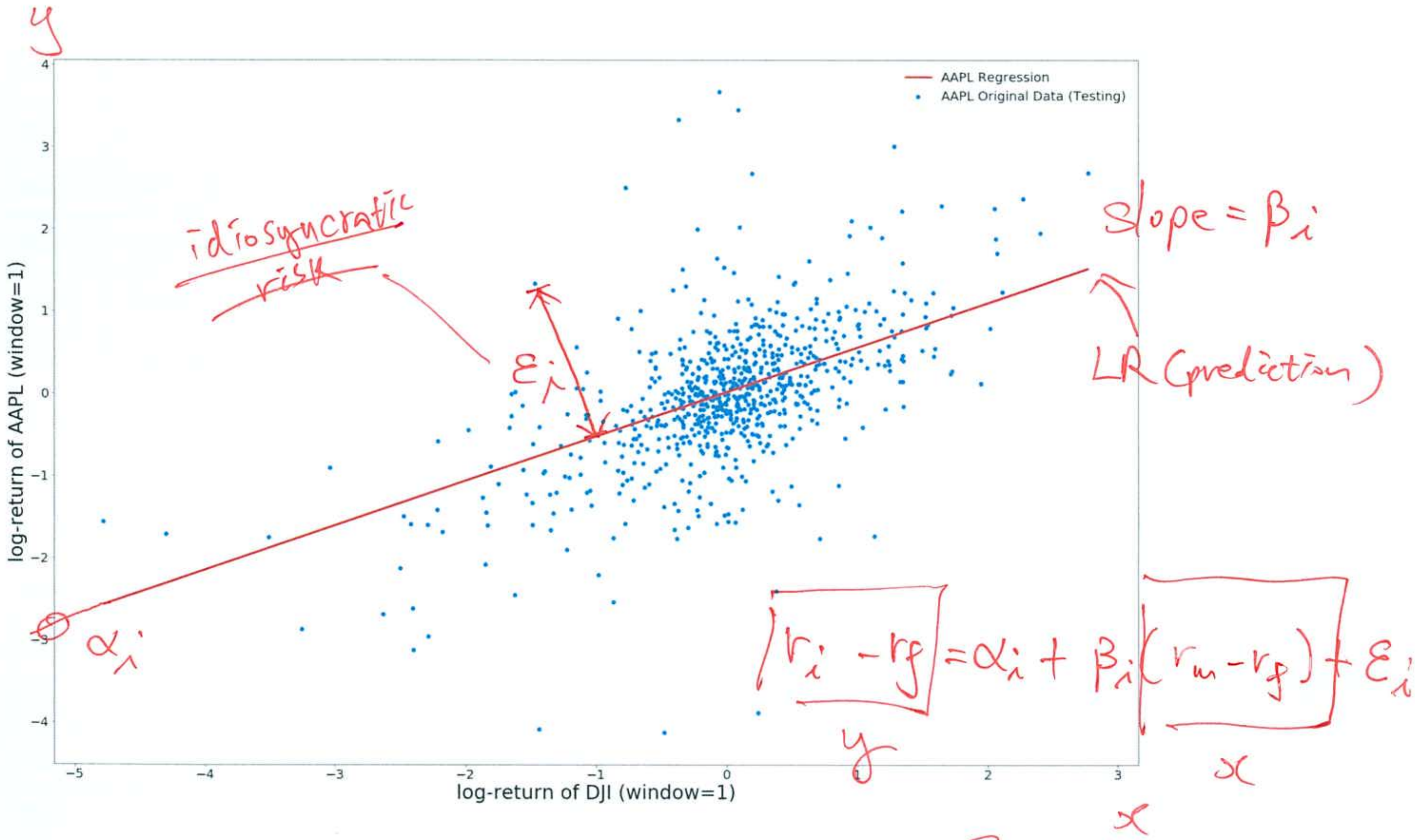
In [23]:

```
lr.fit(df_train['DJI'].values.reshape((1761, 1)), df_train['AAPL'].values.reshape((1761
, 1)))
alpha = lr.intercept_[0]
beta = lr.coef_[0][0]
predictions = lr.predict(df_test['DJI'].values.reshape((756, 1)))
```

In [24]:

```
#date = np.array([datetime.datetime.strptime(t, '%Y-%m-%d').date() for t in np.array(df
_train.index)])

fig, ax_list = plt.subplots(1, 1, figsize=(30,20))
plt.suptitle("CAMP of the testing set", fontsize = 38, fontweight='bold')
ax_list.scatter(df_test['DJI'].values.reshape((756, 1)), df_test['AAPL'].values.reshape
((756, 1)), label='AAPL Original Data (Testing)')
ax_list.plot(df_test['DJI'].values.reshape((756, 1)), predictions, label='AAPL Regressi
on', color='red', linewidth=3)
plt.xlabel("log-return of DJI (window=1)", fontsize=30)
plt.ylabel("log-return of AAPL (window=1)", fontsize=30)
ax_list.tick_params(labelsize=20)
plt.legend(fontsize=20)
plt.show()
```

CAMP of the testing set



$$\underbrace{\boxed{r_i - r_f}}_{y} = \alpha_i + \beta_i \underbrace{\boxed{(r_m - r_f)}}_{x} + \varepsilon_i$$

$$\beta_i = \frac{\sigma_{im}}{\sigma_m^2}$$

model scores and interpretation

In [25]:

```
df_results = pd.DataFrame({'Alpha':alphas, 'Beta':betas, 'R2 in-sample':r2_in_sample,
'R2 out-sample':r2_out_sample},
                          index=tickers)
df_results
```

Out[25]:

systematic risk

	Alpha	Beta	R2 in-sample	R2 out-sample
AAPL	0.012884	0.540214	0.302781	0.323307
AXP	0.002493	0.778564	0.555189	0.210315
BA	-0.004761	0.721516	0.561934	0.509038
CAT	-0.005464	0.765893	0.609088	0.494331
CSCO	-0.006435	0.669297	0.445433	0.459051
CVX	-0.004447	0.762649	0.639511	0.361433
DIS	0.020287	0.797565	0.617386	0.267576
DWDP	0.007085	0.733037	0.498082	0.329535
GS	0.008293	0.726647	0.511405	0.566066
HD	0.014941	0.706463	0.501144	0.411090
IBM	0.012013	0.688678	0.532743	0.362576
INTC	0.002183	0.654399	0.473203	0.377840
JNJ	-0.003915	0.653170	0.490554	0.320031
JPM	-0.001773	0.791462	0.564337	0.463749
KO	0.005496	0.634532	0.409841	0.220611
MCD	-0.005943	0.584590	0.387535	0.201514
MMM	0.005207	0.806897	0.678940	0.520118
MRK	0.014005	0.563051	0.371092	0.045215
MSFT	-0.006016	0.646649	0.437439	0.466641
NKE	0.015417	0.620695	0.416318	0.227720
PFE	-0.001233	0.643521	0.421723	0.288668
PG	-0.004812	0.627081	0.429854	0.177887
TRV	0.009836	0.740250	0.536368	0.379249
UNH	-0.002195	0.545247	0.280003	0.392731
UTX	0.000392	0.817094	0.677376	0.471482
V	0.006846	0.611793	0.365240	0.503265
VZ	-0.002187	0.598259	0.400715	0.156400
WBA	0.012990	0.508173	0.268765	0.213071
WMT	-0.013979	0.452810	0.263426	0.136823
XOM	0.001936	0.776527	0.648714	0.355225

coefficient of determination

$$R_2 = 1 - \underbrace{\frac{SS_{residuals}}{SS_{total}}}_{FVU}$$

Smaller Rs

→ more non-systematic risk

(other factors?)

In [26]:

```
df_test.head()
```

Out[26]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	
Date								
2015-11-24	0.504823	-0.436185	0.076354	0.247754	-0.369226	1.020990	-0.858538	0.2
2015-11-25	-0.484795	0.009493	-0.540500	0.039683	-0.088518	-0.379988	0.358623	-1.2
2015-11-27	-0.170627	0.076081	-0.243084	-0.217741	0.149865	-0.397403	-2.037950	0.0
2015-11-30	0.185322	-0.170569	-0.661077	0.990044	-0.175093	0.719068	-0.999390	0.0
2015-12-01	-0.542321	0.341900	0.884637	-0.796187	0.666562	0.870385	1.063255	1.0

5 rows × 31 columns

In [30]:

```
# df_test - df_results.Alpha works cuz df_results.index==df_test.columns.values is Tru
e; otherwise new columns will be created
df_market = pd.DataFrame(index=df_test.index,
                         columns=tickers,
                         data=np.vstack([df_test.DJI.values]*30).T)
df_unexplained = df_test.iloc[:,:-1] - df_results.Alpha - df_results.Beta * df_market
df_unexplained.head()
```

Out[30]:

$$\varepsilon_i = \log(\text{return}_i) - \alpha_i - \beta_i(\log(\text{return}_m))$$

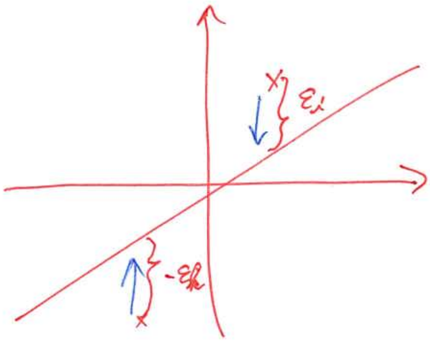
	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	
Date								
2015-11-24	0.455229	-0.491586	0.032085	0.201171	-0.408273	0.973611	-0.933024	0.2
2015-11-25	-0.478551	0.034568	-0.510192	0.072266	-0.058384	-0.348536	0.366576	-1.2
2015-11-27	-0.160723	0.106432	-0.207885	-0.179968	0.184535	-0.360783	-2.024592	0.0
2015-11-30	0.480861	0.271443	-0.244382	1.432778	0.213463	1.158934	-0.564324	0.4
2015-12-01	-1.046035	-0.367984	0.233841	-1.486601	0.064885	0.181903	0.318313	0.4

5 rows × 30 columns

Interpretation
$+\varepsilon_i$:
$-\varepsilon_k$:

Markov Equilibrium

overpriced
↓
sell
↓
price ↓
(back to equil.)



underpriced
↓
people buy
↓
price ↑
(back to equil.)

In [34]:

```
time = np.array([datetime.datetime.strptime(t, '%Y-%m-%d').date() for t in np.array(df_
unexplained.index)])
fig, ax_list = plt.subplots(1, 1, figsize=(30,20))
plt.suptitle("Unexplained price of AAPL in the testing set", fontsize = 38, fontweight=
'bold')
ax_list.plot(time, df_unexplained['AAPL'].values.reshape((756, 1)))
#plt.xlabel("Log-return of DJI (window=1)", fontsize=30)
#plt.ylabel("Log-return of AAPL (window=1)", fontsize=30)
ax_list.tick_params(labelsize=20)
#plt.legend(fontsize=20)
plt.show()
```

Unexplained price of AAPL in the testing set