In [16]:
```
# impor necessary packages
from numpy import *
from matplotlib.pyplot import *
from pandas import *
from pandas import DataFrame
import csv
from matplotlib import style
style.use("ggplot")
from scipy.interpolate import *
from sklearn import datasets
import numpy as np
from numpy.polynomial.polynomial import polyfit
import matplotlib.pyplot as plt

from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
```

In [3]:
```
pwd()
```

Out[3]: '/Users/pacome'

In [4]:
```
import pandas as pd
data = pd.read_csv('crude-file.csv')
data.head(10)
```

Out[4]:

	Volume	temperature
0	1	-0.5
1	2	36.1
2	3	49.2
3	4	68.7
4	5	76.7
5	6	91.8
6	7	99.2
7	8	103.5
8	9	119.4
9	10	125.7

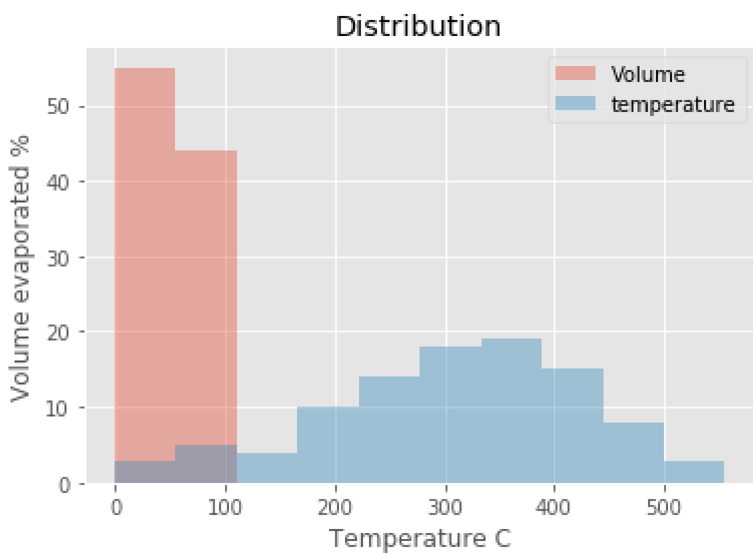
In [5]:
```
data.describe()
```

Out[5]:

	Volume	temperature
count	99.000000	99.000000
mean	50.000000	307.874747
std	28.722813	118.706769
min	1.000000	-0.500000
25%	25.500000	235.100000
50%	50.000000	320.000000
75%	74.500000	393.450000
max	99.000000	555.400000

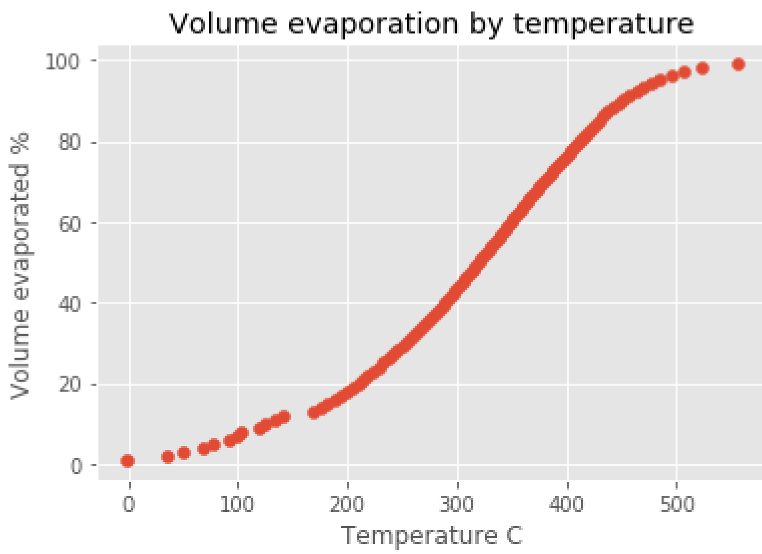
In [6]:
```
#plot the distribution of the data
data.plot(kind='hist', alpha=.4, legend=True) # alpha for transparency
plt.xlabel('Temperature C')
plt.ylabel('Volume evaporated %')
plt.title('Distribution')
```

Out[6]: Text(0.5,1,'Distribution')



In [7]:
```
#plot my data
y_profile = data.Volume
x_profile = data.temperature

plt.scatter(x_profile, y_profile)
plt.xlabel('Temperature C')
plt.ylabel('Volume evaporated %')
plt.title('Volume evaporation by temperature')
plt.show
```

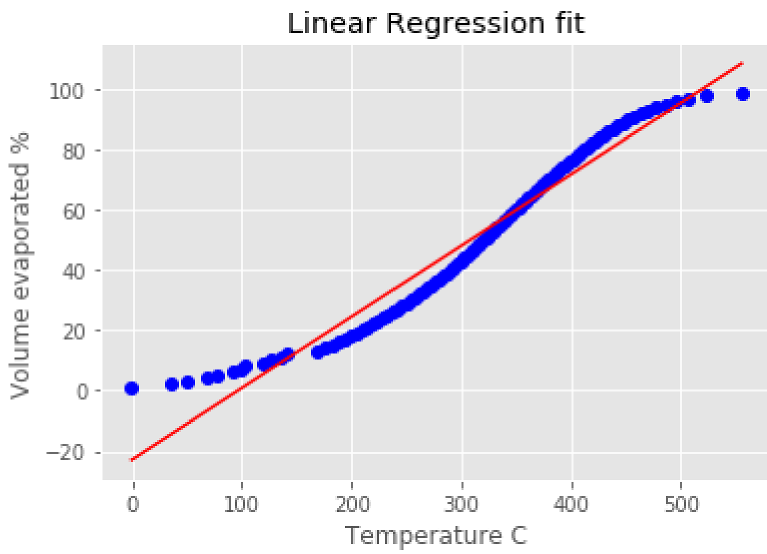
Out[7]: <function matplotlib.pyplot.show(*args, **kw)>



In [8]:

```
#simple linear reg model, see fit
b, m = polyfit(x_profile, y_profile, 1)
fit = m*x_profile + b
plt.plot(x_profile, y_profile, 'bo')
plt.plot(x_profile, fit,'r-')
plt.xlabel('Temperature C')
plt.ylabel('Volume evaporated %')
plt.title('Linear Regression fit')
```

Out[8]: Text(0.5,1,'Linear Regression fit')

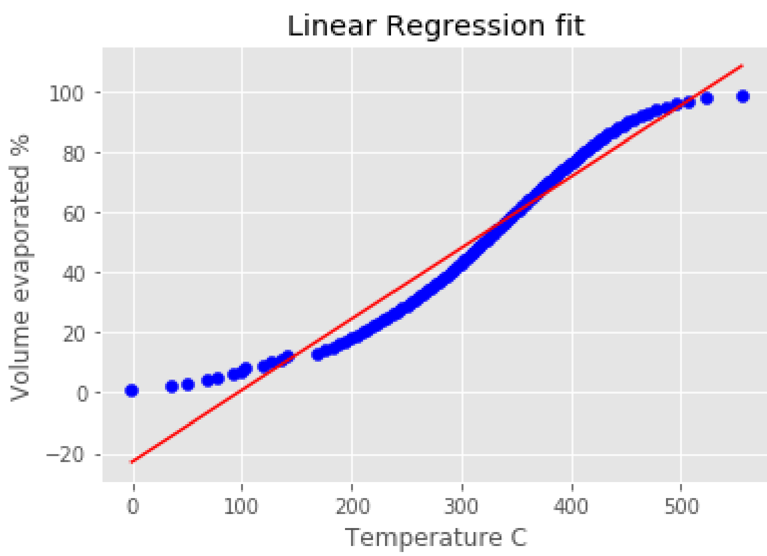


In [9]:
```
#results & Coefs trick
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats

constant = y_profile
y = sm.add_constant(constant)
est = sm.OLS(x_profile, y)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:            temperature   R-squared:
0.955
Model:                            OLS   Adj. R-squared:
0.954
Method:                 Least Squares   F-statistic:
2055.
Date:                Sun, 05 May 2019   Prob (F-statistic):           4.2
9e-67
Time:                        14:24:45   Log-Likelihood:               -4
59.44
No. Observations:                  99   AIC:
922.9
Df Residuals:                      97   BIC:
928.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const        105.9435      5.131     20.649      0.000      95.761      11
6.126
Volume         4.0386      0.089     45.333      0.000       3.862
4.215
==============================================================================
=====
Omnibus:                       35.028   Durbin-Watson:
0.055
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                7
5.409
Skew:                          -1.355   Prob(JB):                     4.2
2e-17
Kurtosis:                       6.308   Cond. No.
116.
==============================================================================
=====

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

In [10]:
```
X = data.temperature.values.reshape(len(data.temperature.values), 1)
y = data.Volume.values.reshape(len(data.temperature.values), 1)
```

In [11]:
```
#redo the linear regression with a built-in function to stay in line with fu
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Visualizing the Linear Regression results
def viz_linear():
    plt.scatter(X, y, color='blue')
    plt.plot(X, lin_reg.predict(X), color='red')
    plt.xlabel('Temperature C')
    plt.ylabel('Volume evaporated %')
    plt.title('Linear Regression fit')
    plt.show()
    return
viz_linear()
```

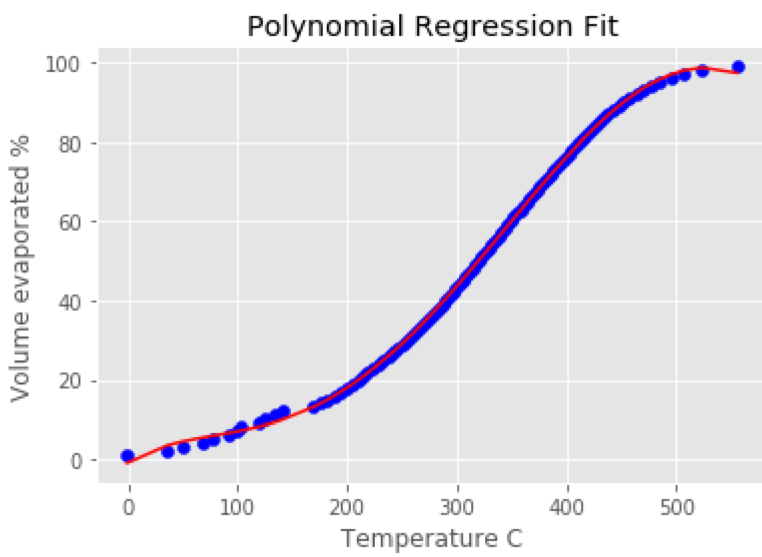


In [13]:
```
#build fundtion to return coef of goodness R**2 = 1-(SSYhat/Mean of Ys)

def squared_error(ys_orig,ys_line):
    return sum((ys_line - ys_orig) * (ys_line - ys_orig))

def coefficient_of_determination(ys_orig,ys_line):
    y_mean_line = [mean(ys_orig) for y in ys_orig]
    squared_error_regr = squared_error(ys_orig, ys_line)
    squared_error_y_mean = squared_error(ys_orig, y_mean_line)
    return 1 - (squared_error_regr/squared_error_y_mean)
```

In [14]:
```
#Linear model coef of goodness/Determination

r_squared = coefficient_of_determination(X, y)
print(r_squared)
```

```
0.9459294333911424
```

In [18]:
```
# Even though the R**2 valu is more satisfying at a glance we can easily dec
#isn't the best fit, let's try with a polynomial model to the nTh degree unt

# Visualising the Polynomial Regression results

poly = PolynomialFeatures(degree = 5)
X_poly = poly.fit_transform(X)

poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)

plt.scatter(X, y, color = 'blue')

plt.plot(X, lin2.predict(poly.fit_transform(X)), color = 'red')
plt.xlabel('Temperature C')
plt.ylabel('Volume evaporated %')
plt.title('Polynomial Regression Fit')

plt.show()
```



In [19]:
```
#Polynomial model coef of goodness/Determination at order 5
from sklearn.metrics import r2_score
y_true = y_profile
y_pred = y
```

In [20]:
```
r2_score(y_true, y_pred) ##1 ?? a perfect model is questionable
```

Out[20]: 1.0

In [ ]:
```
/*the Polynomial model fit our data better, the R**2 value has also increase
for the limitations of the model, by splitting my dataset two parts, testing
Train the model and test the model on the data to appreciate its accuracy.

The conditions concerning the residuals are assumed to be validated.
*/
```