Fram price & Ending stocks Fitting-Jun 16

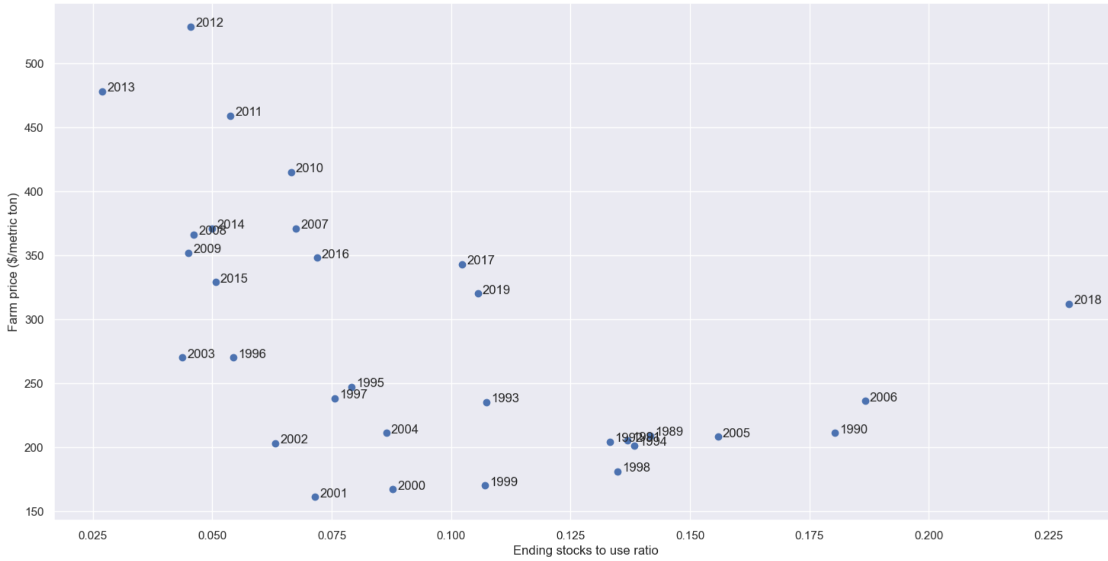
June 22, 2021

[2]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.metrics import r2_score
import scipy.stats as stats

sns.set() # Seaborn's default settings look much nicer than matplotlib

df = pd.read_csv('Regression price.csv', usecols=['Farm price ($/metric␣
↪ton)','Ending stocks to use ratio','Year'])[['Ending stocks to use␣
↪ratio','Farm price ($/metric ton)','Year']]

#Data = df[['Ending stocks to use ratio','Farm price ($/metric ton)','Year']]
#Data = Data.sort_values(by=['Ending stocks to use ratio'], ascending=True)
#Data.head()

df.head()
```

[2]:

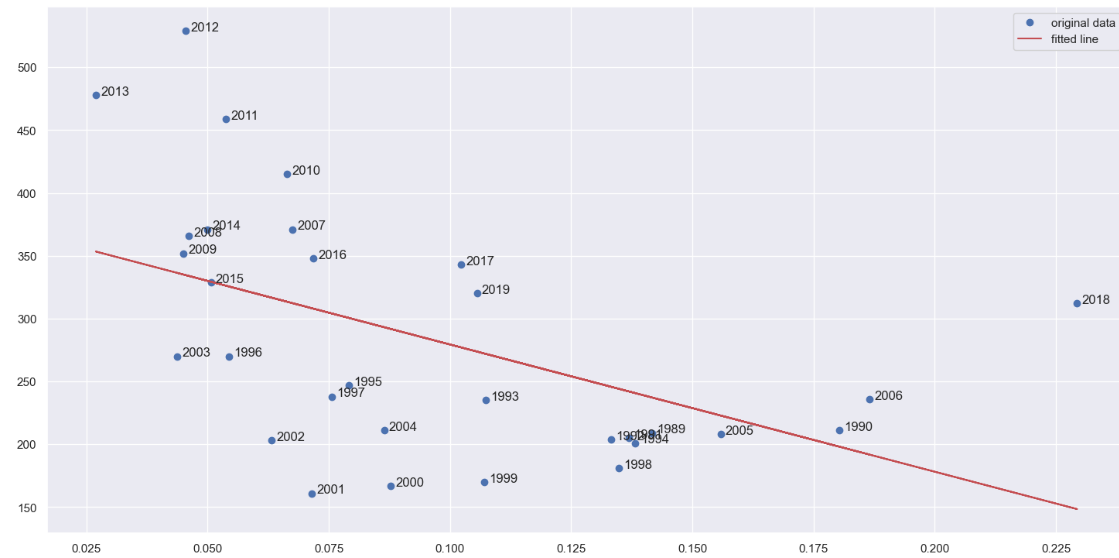
	Ending stocks to use ratio	Farm price ($/metric ton)	Year
0	0.105551	320	2019
1	0.229341	312	2018
2	0.102234	343	2017
3	0.071867	348	2016
4	0.050657	329	2015

[3]:
```python
plt.figure(figsize=(18,9),dpi=100)
plt.plot(df.iloc[:,0], df.iloc[:,1], 'o', label='original data')

for i in range(len(df)):
    plt.annotate(df.iloc[i,2], (df.iloc[i,0], df.iloc[i,1]), xytext=(df.
↪iloc[i,0]+0.001, df.iloc[i,1]))

plt.xlabel('Ending stocks to use ratio')
plt.ylabel('Farm price ($/metric ton)')

plt.show()
#fig.savefig('price&ratio.png', dpi=300)
```




```
[4]: # linear regression
import scipy
# slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
res = scipy.stats.linregress(df.iloc[:,0], df.iloc[:,1])
print(f"R-squared: {res.rvalue**2:.6f}")
print(f"p-value: {res.pvalue**2:.6f}")

plt.figure(figsize=(18,9),dpi=100)
plt.plot(df.iloc[:,0], df.iloc[:,1], 'o', label='original data')
plt.plot(df.iloc[:,0], res.intercept + res.slope*df.iloc[:,0], 'r',␣
 ↪label='fitted line')

for i in range(len(df)):
    plt.annotate(df.iloc[i,2], (df.iloc[i,0], df.iloc[i,1]), xytext=(df.
 ↪iloc[i,0]+0.001, df.iloc[i,1]))

plt.legend()
plt.show()
```

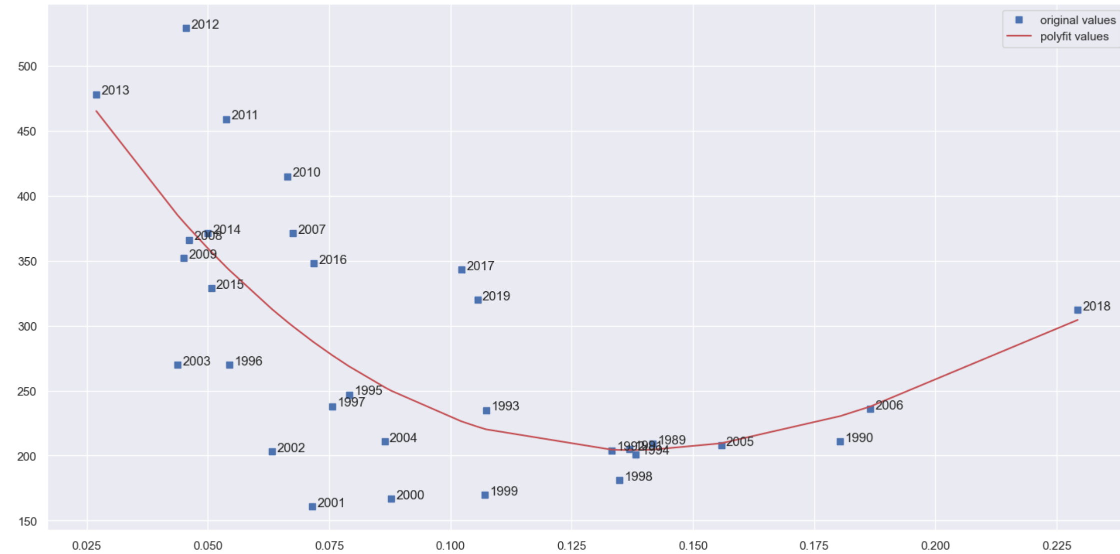
```
R-squared: 0.255919
p-value: 0.000014
```




```
[5]: #  Polynomial regression
f1 = np.polyfit(df.iloc[:,0], df.iloc[:,1], 3)
# p1
p1 = np.poly1d(f1)
print('p1 is :\n',p1)

plt.figure(figsize=(18,9),dpi=100)
plt.plot(df.iloc[:,0], df.iloc[:,1], 's',label='original values')

for i in range(len(df)):
    plt.annotate(df.iloc[i,2], (df.iloc[i,0], df.iloc[i,1]), xytext=(df.
 ↪iloc[i,0]+0.001, df.iloc[i,1]))

Yvals = p1(df.iloc[:,0]) # y
Y = pd.Series(Yvals, name ='YPred')
Pred = pd.concat([df.iloc[:,0],Y,df.iloc[:,2]], axis=1)
Pred = Pred.sort_values(by='Ending stocks to use ratio') #  X

plt.plot(Pred.iloc[:,0], Pred.iloc[:,1], 'r',label='polyfit values')
plt.legend()

coefficient_of_dermination = r2_score(df.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))
r, p_values = stats.pearsonr(df.iloc[:,1],Yvals)
print('p-values:' + str(p_values))
print(stats.kruskal(df.iloc[:,1],Yvals))
```

```
p1 is :
           3             2
```

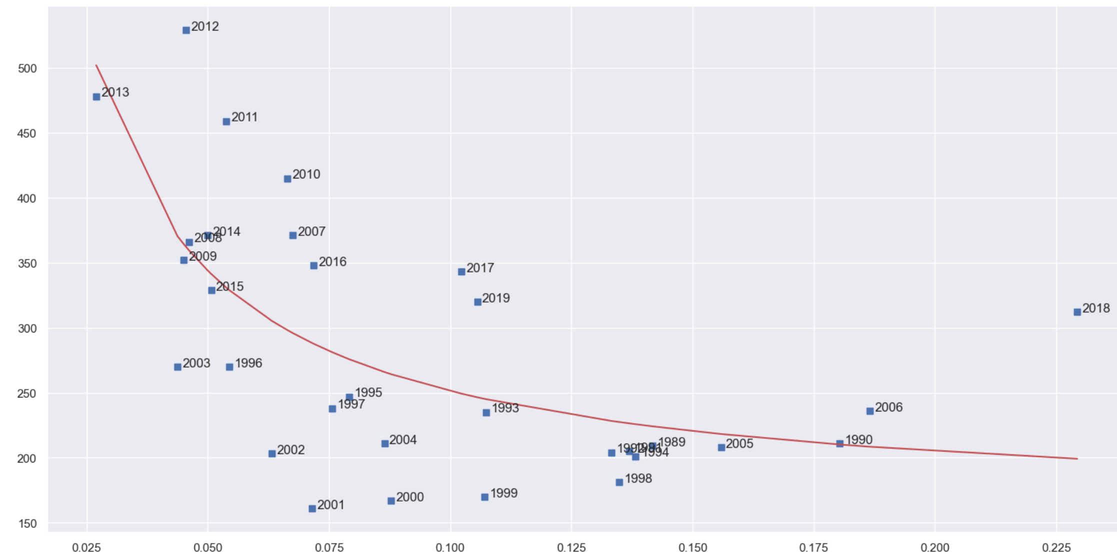
```
-4.706e+04 x + 3.558e+04 x - 7114 x + 632.1
R-squared:0.5098773688313567
p-values:6.4576631974460546e-06
KruskalResult(statistic=0.3413864761143225, pvalue=0.5590300128748782)
```



[6]:
```python
from scipy.optimize import curve_fit #nonlinear least squares
#
def func(x, a, b):
    return a + b*(1/x)

# popt           pcov
popt, pcov = curve_fit(func, df.iloc[:,0], df.iloc[:,1])
a = popt[0]
b = popt[1]
Yvals = func(df.iloc[:,0],a,b) # y

plt.figure(figsize=(18,9),dpi=100)
plt.plot(df.iloc[:,0], df.iloc[:,1], 's',label='original values')
for i in range(len(df)):
    plt.annotate(df.iloc[i,2], (df.iloc[i,0], df.iloc[i,1]), xytext=(df.
 ↪iloc[i,0]+0.001, df.iloc[i,1]))

Y = pd.Series(Yvals, name ='YPred')
Pred = pd.concat([df.iloc[:,0],Y,df.iloc[:,2]], axis=1)
Pred = Pred.sort_values(by='Ending stocks to use ratio') #  X

plt.plot(Pred.iloc[:,0], Pred.iloc[:,1], 'r',label='polyfit values')
```

```python
coefficient_of_dermination = r2_score(df.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))

r, p_values = stats.pearsonr(df.iloc[:,1],Yvals)
print('p-values:' + str(p_values))

print(stats.kruskal(df.iloc[:,1],Yvals))
#stats.friedmanchisquare(Y, Yvals)
```

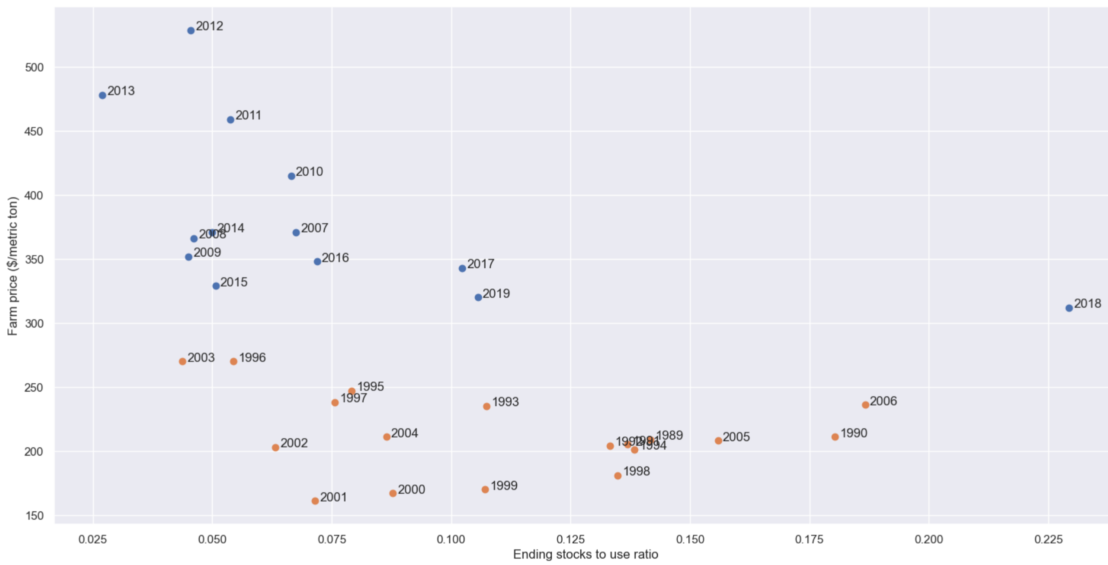
```
R-squared:0.447603519042595
p-values:3.8750032427526525e-05
KruskalResult(statistic=0.392527547873681, pvalue=0.5309737985847316)
```



[7]:

```python
####################################################################
 #Data   before07 & after07
####################################################################
XYAf08 = df.iloc[0:13,:]
XYBf08 = df.iloc[13:,:]
XYBf08 = XYBf08.reset_index(drop=True)

plt.figure(figsize=(18,9),dpi=100)
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,1], 'o', label='original data')
for i in range(len(XYAf08)):
    plt.annotate(XYAf08.iloc[i,2], (XYAf08.iloc[i,0], XYAf08.iloc[i,1]), xytext=(XYAf08.iloc[i,0]+0.001, XYAf08.iloc[i,1]))

plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,1],'o', label='original data')
for i in range(len(XYBf08)):
```

```python
    plt.annotate(XYBf08.iloc[i,2], (XYBf08.iloc[i,0], XYBf08.iloc[i,1]), xytext=(XYBf08.iloc[i,0]+0.001, XYBf08.iloc[i,1]))
plt.xlabel('Ending stocks to use ratio')
plt.ylabel('Farm price ($/metric ton)')
fig = plt.gcf()
ax = plt.gca()
plt.show()
```

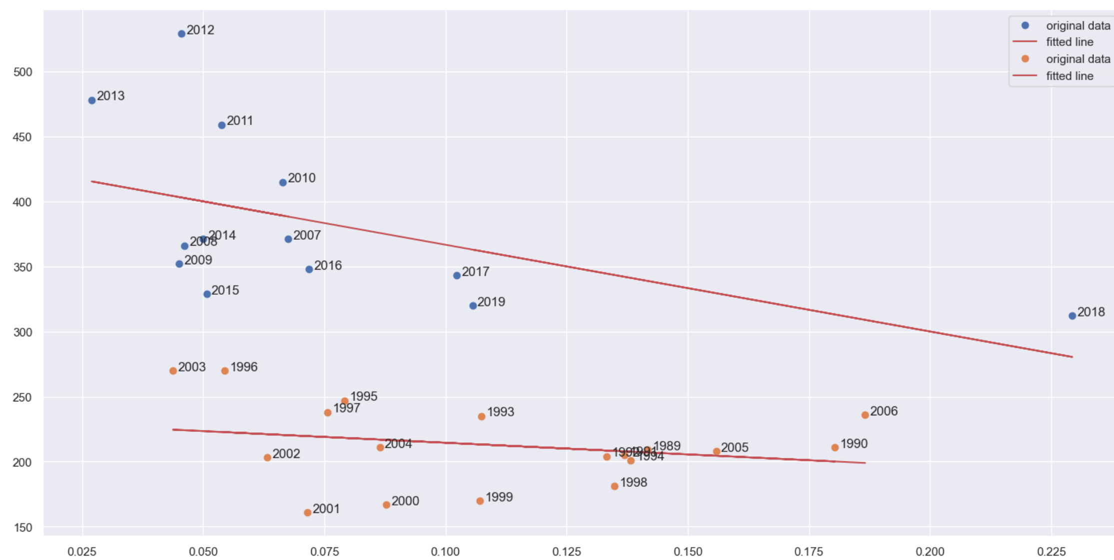


[8]:

```python
# linear regression
# slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
res = scipy.stats.linregress(XYAf08.iloc[:,0], XYAf08.iloc[:,1])
print(f"R-squared: {res.rvalue**2:.6f}")
print(f"p-value: {res.pvalue**2:.6f}")

plt.figure(figsize=(18,9),dpi=100)
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,1], 'o', label='original data')
plt.plot(XYAf08.iloc[:,0], res.intercept + res.slope*XYAf08.iloc[:,0], 'r', label='fitted line')
for i in range(len(XYAf08)):
    plt.annotate(XYAf08.iloc[i,2], (XYAf08.iloc[i,0], XYAf08.iloc[i,1]), xytext=(XYAf08.iloc[i,0]+0.001, XYAf08.iloc[i,1]))
plt.legend()

# XB YB
res = scipy.stats.linregress(XYBf08.iloc[:,0], XYBf08.iloc[:,1])
print(f"R-squared: {res.rvalue**2:.6f}")
print(f"p-value: {res.pvalue**2:.6f}")
```

```
plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,1], 'o', label='original data')
plt.plot(XYBf08.iloc[:,0], res.intercept + res.slope*XYBf08.iloc[:,0], 'r',␣
↪label='fitted line')
for i in range(len(XYBf08)):
    plt.annotate(XYBf08.iloc[i,2], (XYBf08.iloc[i,0], XYBf08.iloc[i,1]),␣
↪xytext=(XYBf08.iloc[i,0]+0.001, XYBf08.iloc[i,1]))
plt.legend()
plt.show()
```

```
R-squared: 0.268240
p-value: 0.004877
R-squared: 0.056944
p-value: 0.115790
```



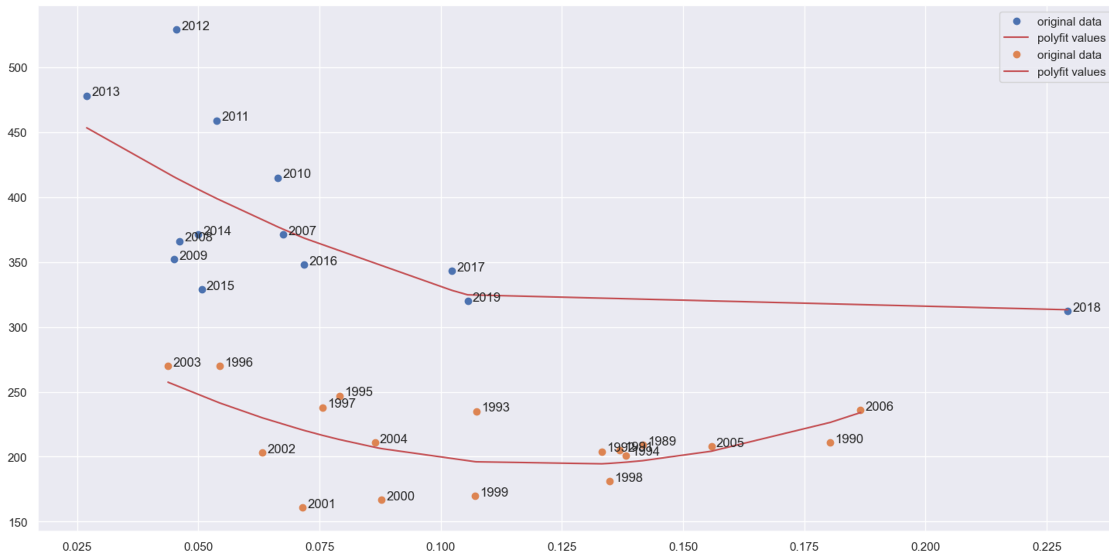
[9]:
```
#  Polynomial regression
items = 2 #Polynomial
# XA YA
f1 = np.polyfit(XYAf08.iloc[:,0], XYAf08.iloc[:,1], items)
# p1
p1 = np.poly1d(f1)
print('p1 is :\n',p1)

plt.figure(figsize=(18,9),dpi=100)
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,1], 'o', label='original data')
for i in range(len(XYAf08)):
    plt.annotate(XYAf08.iloc[i,2], (XYAf08.iloc[i,0], XYAf08.iloc[i,1]),␣
↪xytext=(XYAf08.iloc[i,0]+0.001, XYAf08.iloc[i,1]))
```

```
Yvals = p1(XYAf08.iloc[:,0]) # y
XYAf08['Pred'] = pd.Series(Yvals, name ='YPred')
XYAf08 = XYAf08.sort_values(by='Ending stocks to use ratio') # X
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,3], 'r',label='polyfit values')

coefficient_of_dermination = r2_score(XYAf08.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))
r, p_values = stats.pearsonr(XYAf08.iloc[:,1],Yvals)
print('p-values:' + str(p_values))
print(stats.kruskal(XYAf08.iloc[:,1],Yvals))


# XB YB
f2 = np.polyfit(XYBf08.iloc[:,0], XYBf08.iloc[:,1], items)
# p1
p2 = np.poly1d(f2)
print('p1 is :\n',p2)

plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,1], 'o', label='original data')
for i in range(len(XYBf08)):
    plt.annotate(XYBf08.iloc[i,2], (XYBf08.iloc[i,0], XYBf08.iloc[i,1]),␣
 ↪xytext=(XYBf08.iloc[i,0]+0.001, XYBf08.iloc[i,1]))

Yvals = p2(XYBf08.iloc[:,0]) # y
XYBf08['Pred'] = pd.Series(Yvals, name ='YPred')
XYBf08 = XYBf08.sort_values(by='Ending stocks to use ratio') # X
plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,3], 'r',label='polyfit values')

coefficient_of_dermination = r2_score(XYBf08.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))
r, p_values = stats.pearsonr(XYBf08.iloc[:,1],Yvals)
print('p-values:' + str(p_values))
print(stats.kruskal(XYBf08.iloc[:,1],Yvals))

plt.legend()
plt.show()
```

```
p1 is :
       2
7631 x - 2648 x + 519.2
R-squared:-0.7224864996167877
p-values:0.3760811232911454
KruskalResult(statistic=0.2900399873724069, pvalue=0.5901949065697367)
p1 is :
            2
1.009e+04 x - 2489 x + 346.9
R-squared:-0.2751195794184502
p-values:0.8143787730661286
```

```
KruskalResult(statistic=0.06408055841630043, pvalue=0.8001589631104618)
```

```
<ipython-input-9-aa7264d86b99>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  XYAf08['Pred'] = pd.Series(Yvals, name ='YPred')
```

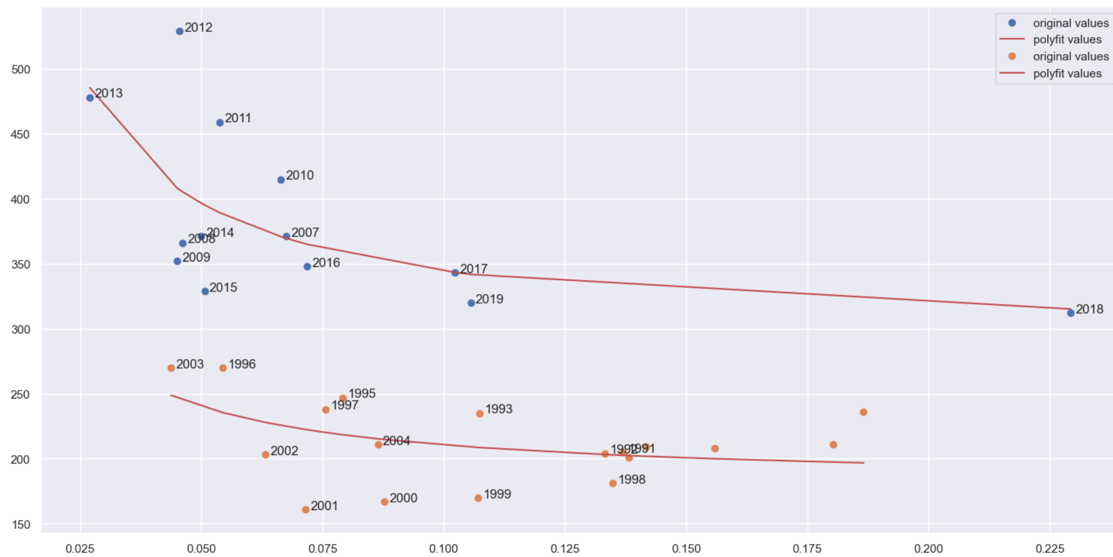


[10]:

```
from scipy.optimize import curve_fit #nonlinear least squares
#
def func(x, a, b):
    return a + b*(1/x)

# popt           pcov
popt, pcov = curve_fit(func, XYAf08.iloc[:,0], XYAf08.iloc[:,1])
a = popt[0]
b = popt[1]
Yvals = func(XYAf08.iloc[:,0],a,b) # y

plt.figure(figsize=(18,9),dpi=100)
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,1], 'o',label='original values')
for i in range(len(XYAf08)):
    plt.annotate(XYAf08.iloc[i,2], (XYAf08.iloc[i,0], XYAf08.iloc[i,1]),␣
 ↪xytext=(XYAf08.iloc[i,0]+0.001, XYAf08.iloc[i,1]))

XYAf08['Pred'] = pd.Series(Yvals, name ='YPred')
XYAf08 = XYAf08.sort_values(by='Ending stocks to use ratio') # X
```

```python
plt.plot(XYAf08.iloc[:,0], XYAf08.iloc[:,3], 'r',label='polyfit values')

coefficient_of_dermination = r2_score(XYAf08.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))

r, p_values = stats.pearsonr(XYAf08.iloc[:,1],Yvals)
print('p-values:' + str(p_values))

print(stats.kruskal(XYAf08.iloc[:,1],Yvals))
#stats.friedmanchisquare(Y, Yvals)

## Before
# popt           pcov
popt, pcov = curve_fit(func, XYBf08.iloc[:,0], XYBf08.iloc[:,1])
a = popt[0]
b = popt[1]
Yvals = func(XYBf08.iloc[:,0],a,b) # y

plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,1], 'o',label='original values')
for i in range(len(XYAf08)):
    plt.annotate(XYBf08.iloc[i,2], (XYBf08.iloc[i,0], XYBf08.iloc[i,1]),␣
↪xytext=(XYBf08.iloc[i,0]+0.001, XYBf08.iloc[i,1]))

XYBf08['Pred'] = pd.Series(Yvals, name ='YPred')
XYBf08 = XYBf08.sort_values(by='Ending stocks to use ratio') #  X

plt.plot(XYBf08.iloc[:,0], XYBf08.iloc[:,3], 'r',label='polyfit values')

coefficient_of_dermination = r2_score(XYBf08.iloc[:,1], Yvals)
print('R-squared:' + str(coefficient_of_dermination))

r, p_values = stats.pearsonr(XYBf08.iloc[:,1],Yvals)
print('p-values:' + str(p_values))

print(stats.kruskal(XYBf08.iloc[:,1],Yvals))
#stats.friedmanchisquare(Y, Yvals)

plt.legend()
plt.show()
```

```
R-squared:0.3953587016871638
p-values:0.021335537268839622
KruskalResult(statistic=0.23742502367673746, pvalue=0.626072115506939)
R-squared:0.1976420460840974
p-values:0.0645323681005783
KruskalResult(statistic=0.025031468131356253, pvalue=0.8742886744742262)
```



[11]:
```
XYAf08
```

[11]:

	Ending stocks to use ratio	Farm price ($/metric ton)	Year	Pred
6	0.026969	478	2013	485.523357
10	0.045005	352	2009	408.145909
7	0.045455	529	2012	407.002925
11	0.046061	366	2008	405.495656
5	0.049904	371	2014	396.789340
4	0.050657	329	2015	395.239131
8	0.053801	459	2011	389.231462
9	0.066441	415	2010	370.819208
12	0.067551	371	2007	369.531659
3	0.071867	348	2016	364.903029
2	0.102234	343	2017	343.382175
0	0.105551	320	2019	341.781698
1	0.229341	312	2018	315.154451

[12]:
```
import tensorflow as tf
from tensorflow import keras

from keras.models import Sequential
from keras.layers import Dense

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(np.array([100* X, 1000 * X␣
↪** 2, 1e4 * X ** 3]).T, np.array(Y), test_size=0.2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-cdcdb8a0fb5d> in <module>
      9
     10
---> 11 X_train, X_test, y_train, y_test = train_test_split(np.array([100* X,␣
↪1000 * X ** 2, 1e4 * X ** 3]).T, np.array(Y), test_size=0.2)

NameError: name 'X' is not defined
```

[ ]:
```
coefficient_of_dermination = r2_score(YA, Yvals)
print('R-squared:' + str(coefficient_of_dermination))
r, p_values = stats.pearsonr(YA,Yvals)
print('p-values:' + str(p_values))
print(stats.kruskal(YA,Yvals))

# XB YB
f1 = np.polyfit(XB, YB, 2)
# p1
p1 = np.poly1d(f1)
print('p1 is :\n',p1)

plt.plot(XB, YB, 's',label='original values')
Yvals = p1(XB) # y
plt.plot(XB, Yvals, 'r',label='polyfit values')
plt.legend()

coefficient_of_dermination = r2_score(YB, Yvals)
print('R-squared:' + str(coefficient_of_dermination))
r, p_values = stats.pearsonr(YB,Yvals)
print('p-values:' + str(p_values))
print(stats.kruskal(YB,Yvals))
```

[ ]:
```
np.array([X_test, X_test ** 2]).T
```

[ ]:
```
X_test
```

[ ]:
```
XX = np.linspace(-2, 6, 200)
np.random.shuffle(XX)
YY = 0.5 * -XX * XX + 2 + XX + 0.15 * np.random.randn(200,)

X_train, X_test, y_train, y_test = train_test_split(XX, YY, test_size=0.2)
```

[ ]:
```
model = Sequential()
# model.add(Dense(5, activation='sigmoid'))
# model.add(Dense(1))
model.add(Dense(100, input_dim=3,  kernel_initializer='random_normal',
    bias_initializer='zeros', activation="sigmoid"))
model.add(Dense(100, kernel_initializer='random_normal',
    bias_initializer='zeros', activation="sigmoid"))
model.add(Dense(1,  kernel_initializer='random_normal',
    bias_initializer='zero'))
opt = keras.optimizers.Adam(learning_rate=0.003)
model.compile(loss='mse', optimizer=opt)
model.fit(X_train, y_train, epochs=10000, batch_size=10)
model.output_shape
```

[ ]:
```
model.get_weights()
```

[ ]:
```
y_pred = model.predict(X_test)
plt.scatter(X_train[:,0], y_train)
plt.scatter(X_test[:,0], y_test)
plt.plot(X_test[:,0], y_pred)
plt.show()
```

[ ]:
```
y_pred = model.predict(X_train)
plt.scatter(X_train[:,0], y_train)
plt.scatter(X_test[:,0], y_test)
plt.plot(X_train[:,0], y_pred)
plt.show()
```

[ ]:
```
y_pred
```

[ ]:
```
X_test
```

[ ]:
```

```