a1

September 3, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import altair as alt
        import statsmodels.api as sm
        import statsmodels.formula.api as smf
        from math import log
        from itertools import combinations
        import copy
```

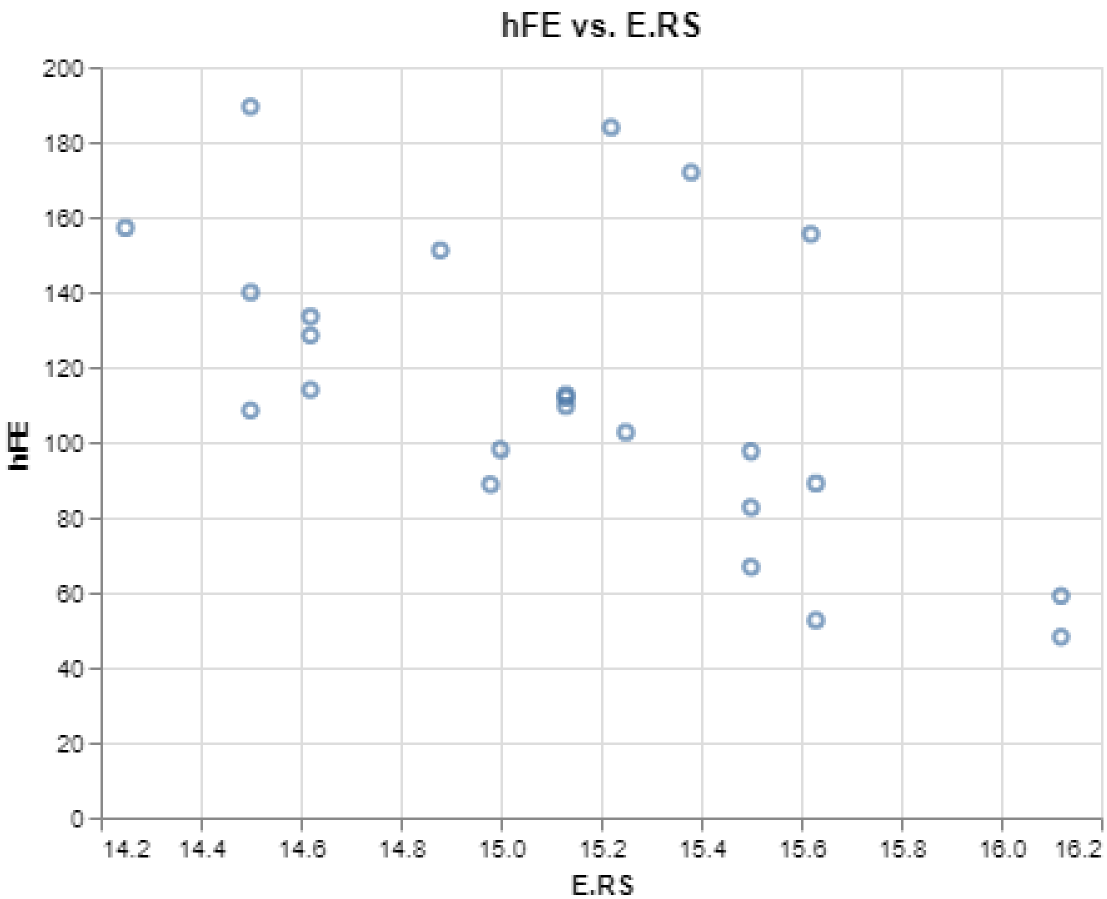
```
In [2]: semic = pd.read_csv('Data/semic.dat', sep='\s+')
        semic.columns=['ers', 'brs', 'ebrs', 'hfe']
        semic.head()
```

Out[2]:

	ers	brs	ebrs	hfe
0	14.62	226.0	7.000	128.40
1	15.63	220.0	3.375	52.62
2	14.62	217.4	6.375	113.90
3	15.00	220.0	6.000	98.01
4	14.50	226.5	7.625	139.90

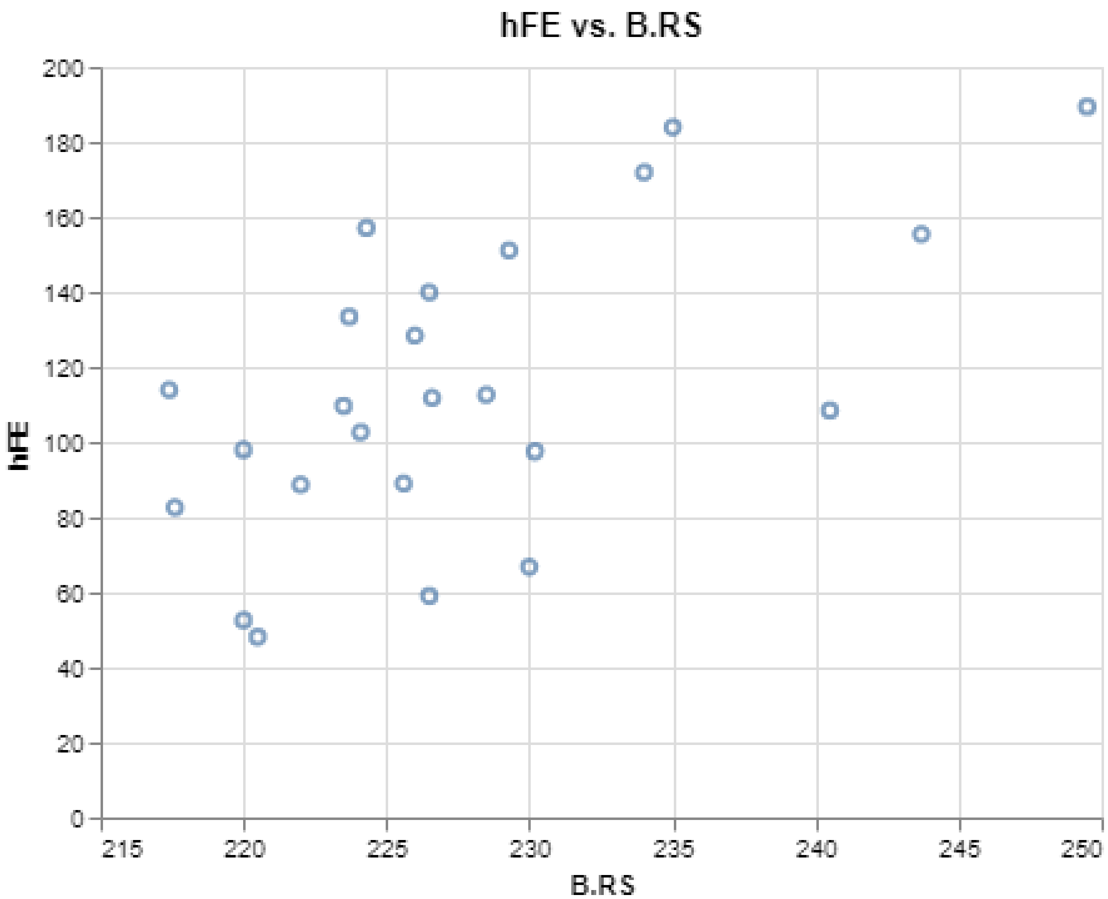
```
In [3]: alt.Chart(semic, title='hFE vs. E.RS').mark_point().encode(x=alt.X('ers', title='E.RS'
```

Out[3]:



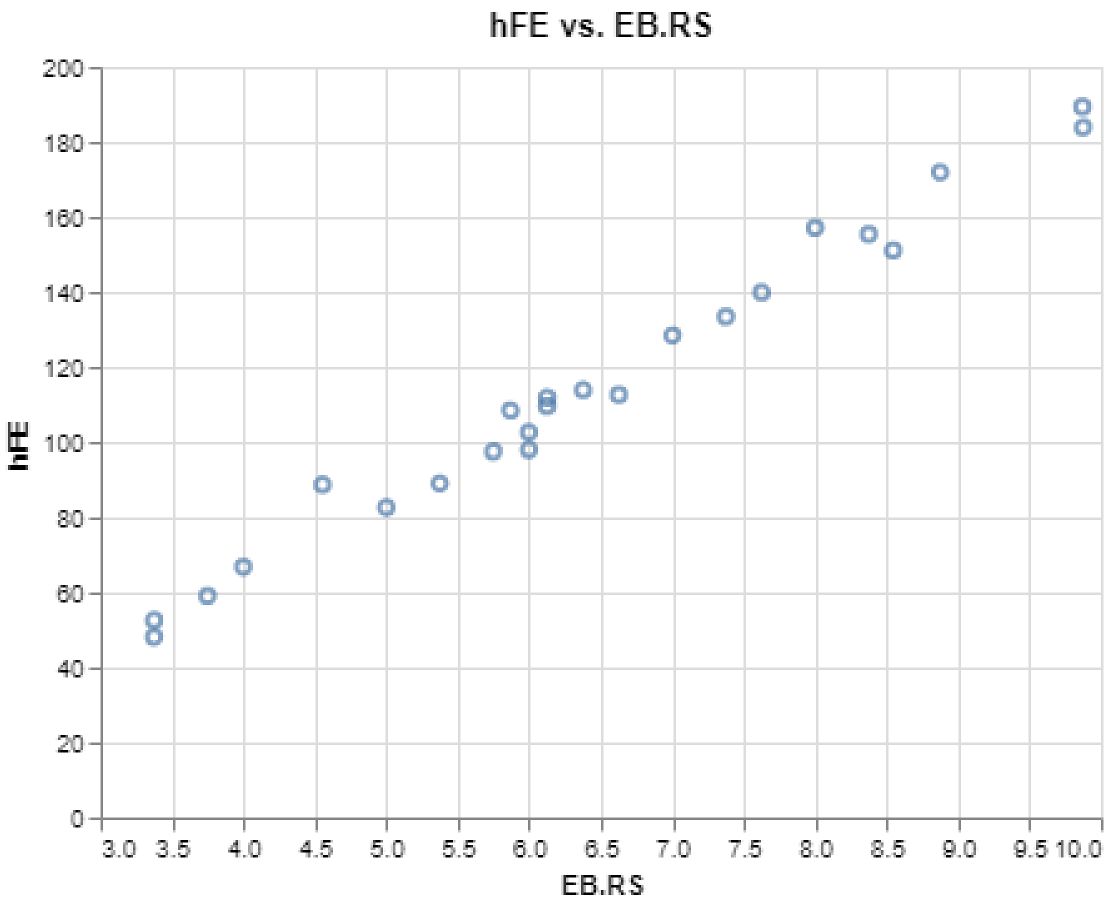

```
In [4]: alt.Chart(semic, title='hFE vs. B.RS').mark_point().encode(x=alt.X('brs', title='B.RS'
```

Out[4]:




```
In [5]: alt.Chart(semic, title='hFE vs. EB.RS').mark_point().encode(x=alt.X('ebrs', title='EB.I
```

Out[5]:




```
In [6]: semicFit = smf.ols('hfe ~ ebrs + brs + ers', semic).fit()
        print(semicFit.summary())
        print(f'variance: {semicFit.scale}')
        print(f'Coefficients: \n{semicFit.params}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    hfe   R-squared:                       0.988
Model:                            OLS   Adj. R-squared:                  0.986
Method:                 Least Squares   F-statistic:                     530.2
Date:                Tue, 03 Sep 2019   Prob (F-statistic):           3.21e-19
Time:                        08:03:07   Log-Likelihood:                -69.287
No. Observations:                  24   AIC:                             146.6
Df Residuals:                      20   BIC:                             151.3
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
Intercept     16.0097     44.187      0.362      0.721     -76.163     108.182
ebrs          19.4235      0.802     24.213      0.000      17.750      21.097
brs            0.2349      0.159      1.476      0.155      -0.097       0.567
ers           -5.2407      2.422     -2.164      0.043     -10.292      -0.189
==============================================================================
Omnibus:                        4.041   Durbin-Watson:                   2.256
Prob(Omnibus):                  0.133   Jarque-Bera (JB):                2.320
Skew:                           0.693   Prob(JB):                        0.314
Kurtosis:                       3.630   Cond. No.                     1.04e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.04e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
variance: 22.609981340360555
Coefficients:
Intercept    16.009655
ebrs         19.423532
brs           0.234928
ers          -5.240723
dtype: float64
```

In [7]:
```
print(sm.stats.anova_lm(semicFit, type=2))
```

```
            df        sum_sq       mean_sq            F        PR(>F)
ebrs       1.0  35840.470457  35840.470457  1585.161435  1.598031e-20
brs        1.0     19.231423     19.231423     0.850572  3.673831e-01
ers        1.0    105.899826    105.899826     4.683764  4.272500e-02
Residual  20.0    452.199627     22.609981          NaN           NaN
```

In [8]:
```
p = semicFit.params
p[0] + 15.13*p[3] + 223.5*p[2] + 6.125*p[1]
```

Out[8]: 108.19315918035909

In [9]:
```
p = semicFit.predict({'ebrs':6.125, 'brs':223.5, 'ers':15.13})
p = semicFit.get_prediction({'ebrs':6.125, 'brs':223.5, 'ers':15.13})
```

In [10]:
```
p.conf_int(alpha=.01)
```

Out[10]: array([[105.01027994, 111.37603842]])

In [11]:
```
p.predicted_mean
```

Out[11]: array([108.19315918])

In [12]:
```
p.conf_int(obs=True, alpha=0.01)
```

```
Out[12]: array([[ 94.29423185, 122.09208651]])

In [13]: semicFit.conf_int(.01)

Out[13]:                     0           1
         Intercept -109.717212  141.736523
         ebrs        17.141004   21.706059
         brs         -0.217842    0.687699
         ers        -12.130860    1.649413

In [14]: semicFit.cov_params()

Out[14]:              Intercept      ebrs       brs        ers
         Intercept  1952.486049 -3.435156 -3.671093 -72.237543
         ebrs         -3.435156  0.643523 -0.079168   1.145233
         brs          -3.671093 -0.079168  0.025321  -0.104852
         ers         -72.237543  1.145233 -0.104852   5.863911

In [15]: semic.mean()

Out[15]: ers      15.138750
         brs     227.708333
         ebrs      6.410000
         hfe     114.671667
         dtype: float64

In [16]: semic.head()

Out[16]:      ers    brs   ebrs     hfe
         0  14.62  226.0  7.000  128.40
         1  15.63  220.0  3.375   52.62
         2  14.62  217.4  6.375  113.90
         3  15.00  220.0  6.000   98.01
         4  14.50  226.5  7.625  139.90

In [17]: semicCntr = semic/semic.mean()
         semicCntr.head()

Out[17]:         ers       brs      ebrs       hfe
         0  0.965734  0.992498  1.092044  1.119719
         1  1.032450  0.966148  0.526521  0.458875
         2  0.965734  0.954730  0.994540  0.993271
         3  0.990835  0.966148  0.936037  0.854701
         4  0.957807  0.994694  1.189548  1.220005

In [18]: X = np.array(semicCntr[['ers','brs','ebrs']])
         S = np.matmul(X.transpose(), X)
         S
```

```
Out[18]: array([[24.02636101, 23.99594666, 23.87148981],
                [23.99594666, 24.02860805, 24.14206085],
                [23.87148981, 24.14206085, 26.0160813 ]])
```

```
In [19]: S.diagonal()**0.5
```

```
Out[19]: array([4.90166921, 4.90189841, 5.10059617])
```

```
In [20]: S
```

```
Out[20]: array([[24.02636101, 23.99594666, 23.87148981],
                [23.99594666, 24.02860805, 24.14206085],
                [23.87148981, 24.14206085, 26.0160813 ]])
```

```
In [21]: X[0,1]
```

```
Out[21]: 0.9924977127172918
```

```
In [22]: X[23]
```

```
Out[22]: array([1.03178928, 1.07022873, 1.30655226])
```

```
In [23]: r = np.zeros([3,3])
         for i in range(3):
             for j in range(3):
                 r[i,j] = S[i,j]/(np.sqrt(S[i,i]*S[j,j]))
         print(r)
```

```
[[1.         0.99868743 0.95480478]
 [0.99868743 1.         0.96558185]
 [0.95480478 0.96558185 1.        ]]
```

0.1 Question 2

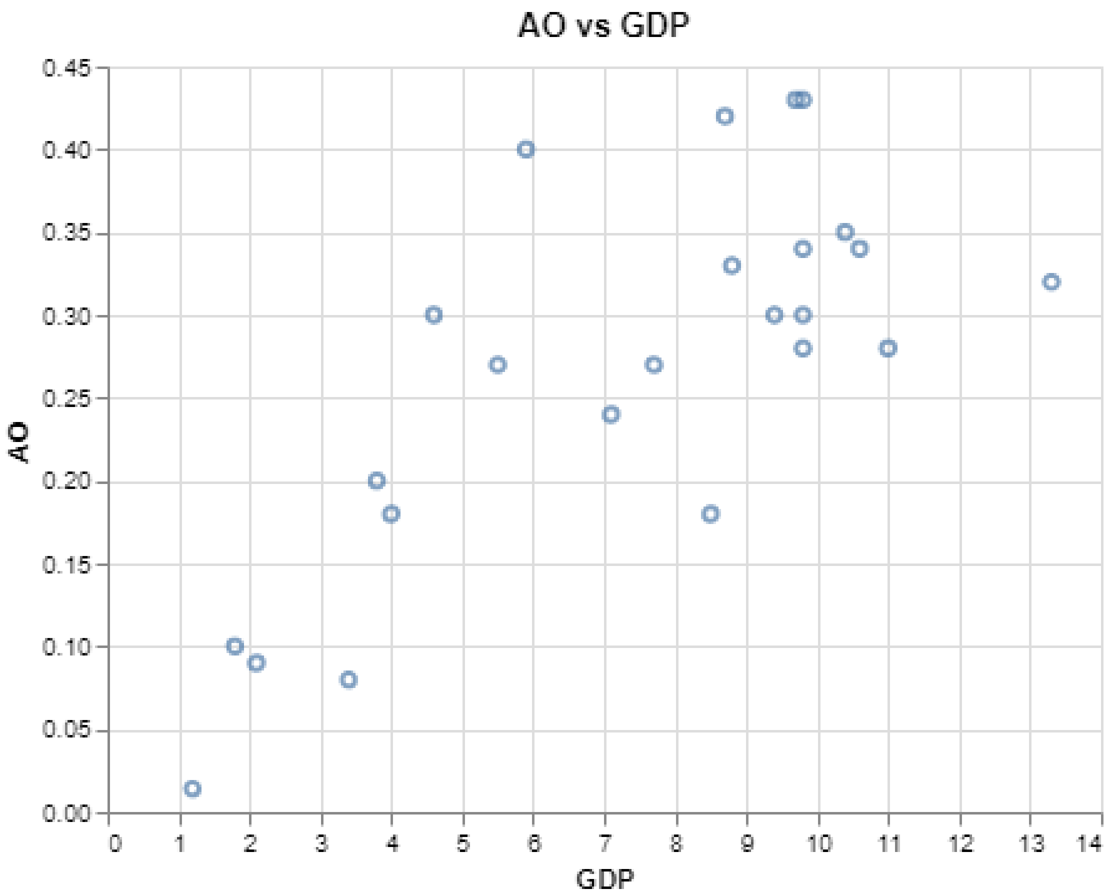
```
In [24]: cars = pd.read_csv('Data/car.dat', sep='\s+')
         cars.head()
```

Out[24]:

	Country	AO	POP	DEN	GDP	PR	CON	TR
0	Austria	0.27	7.5	89	7.7	49	1.11	2.6
1	Belgium	0.30	9.8	323	9.8	59	1.04	1.6
2	Canada	0.42	23.5	2	8.7	17	2.82	0.1
3	Denmark	0.28	5.1	119	11.0	56	1.21	1.9
4	Finland	0.24	4.8	16	7.1	49	1.22	2.2

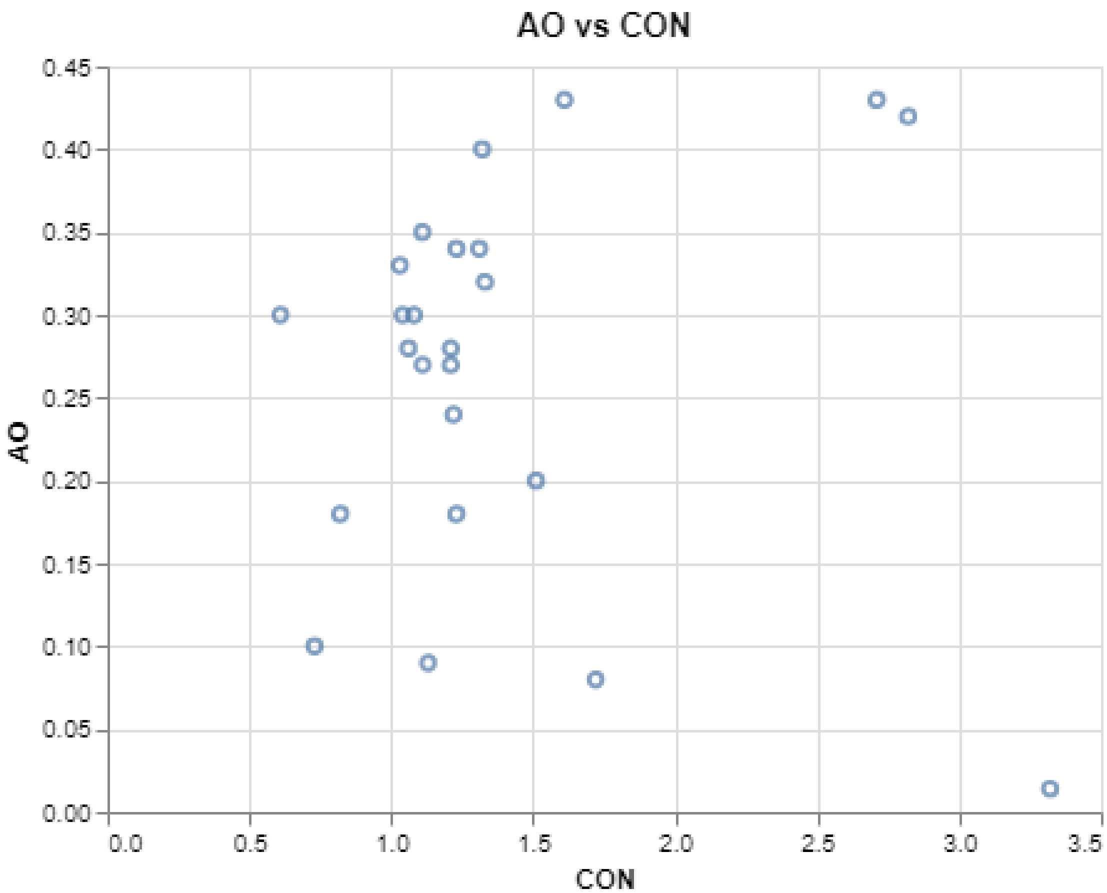
```
In [25]: alt.Chart(cars, title='AO vs GDP').mark_point().encode(x='GDP', y=alt.Y('AO', scale=al
```

```
Out[25]:
```



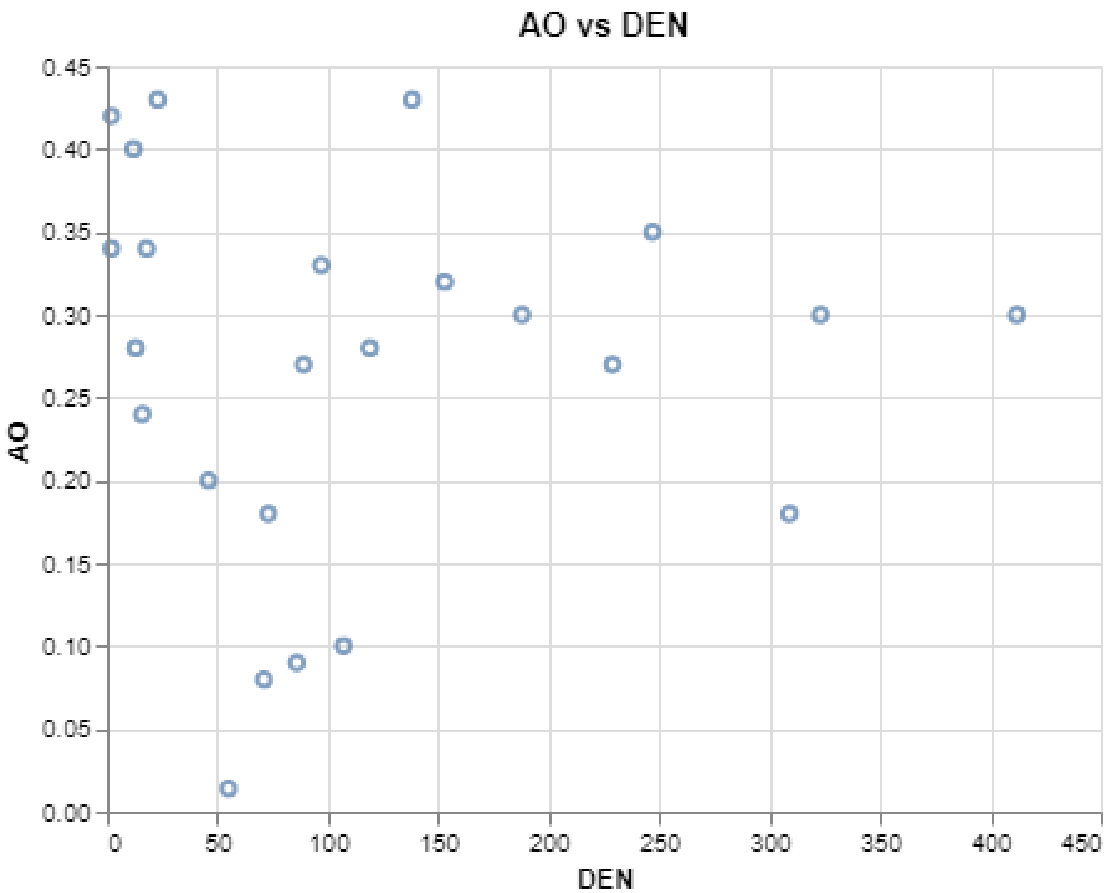
In [26]:
```
alt.Chart(cars, title='AO vs CON').mark_point().encode(x='CON', y=alt.Y('AO', scale=al
```

Out[26]:



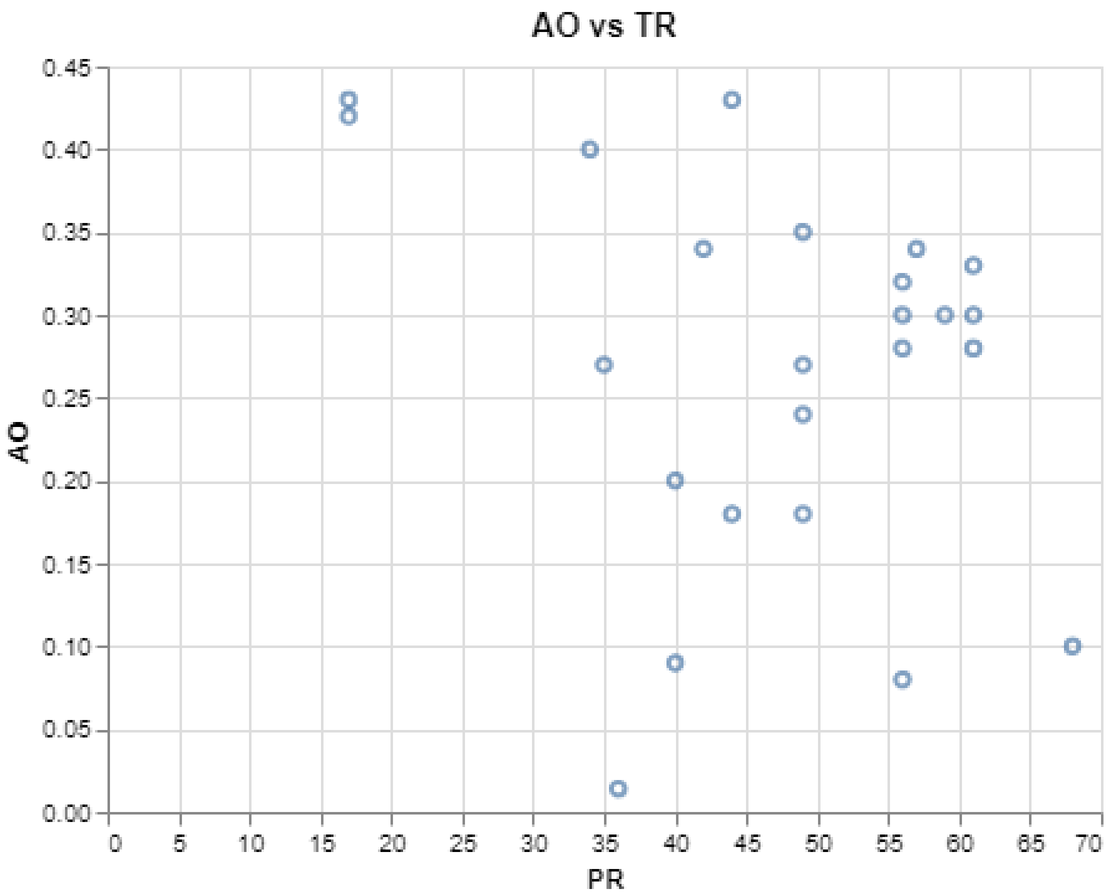

```
In [27]: alt.Chart(cars, title='AO vs DEN').mark_point().encode(x='DEN', y=alt.Y('AO', scale=al
```

Out[27]:




```
In [28]: alt.Chart(cars, title='AO vs TR').mark_point().encode(x='PR', y=alt.Y('AO', scale=alt
```

Out[28]:




```
In [29]: #code for forward_selected from https://planspace.org/20150423-forward_selection_with_
         def forward_selected(data, response):
             """Linear model designed by forward selection.

             Parameters:
             -----------
             data : pandas DataFrame with all possible predictors and response

             response: string, name of response column in data

             Returns:
             --------
             model: an "optimal" fitted statsmodels linear model
                    with an intercept
                    selected by forward selection
                    evaluated by adjusted R-squared
             """
             remaining = set(data.columns)
             remaining.remove(response)
```

```
        selected = []
        current_score, best_new_score = 0.0, 0.0
        while remaining and current_score == best_new_score:
            scores_with_candidates = []
            for candidate in remaining:
                formula = "{} ~ {} + 1".format(response,
                                               ' + '.join(selected + [candidate]))
                score = smf.ols(formula, data).fit().rsquared_adj
                scores_with_candidates.append((score, candidate))
            scores_with_candidates.sort()
            best_new_score, best_candidate = scores_with_candidates.pop()
            if current_score < best_new_score:
                remaining.remove(best_candidate)
                selected.append(best_candidate)
                current_score = best_new_score
        formula = "{} ~ {} + 1".format(response,
                                       ' + '.join(selected))
        model = smf.ols(formula, data).fit()
        return model
```

```
In [47]: def backward_selected(data, response):
        cols = set(data.columns)
        cols.remove(response)
        cols = list(cols)
        currentScore, bestNewScore = 0.0, 0.0
        loop = True

        while loop and currentScore == bestNewScore:
            scoresAndCandidates = []
            rCols = copy.deepcopy(cols)
            for col in rCols:
                rCols.remove(col)
                formula = "{} ~ {}".format(response, ' + '.join(rCols))
                score = smf.ols(formula, data).fit().rsquared_adj
                scoresAndCandidates.append((score, col))
            scoresAndCandidates.sort()
            bestNewScore, bestCandidate = scoresAndCandidates.pop()
            if currentScore < bestNewScore:
                cols.remove(bestCandidate)
                currentScore = bestNewScore
            else:
                loop = False
        formula = f"{response} ~ {' + '.join(cols)}"
        model = smf.ols(formula, data).fit()
        return model
```

```
In [48]: m = backward_selected(cars[['AO', 'POP', 'DEN', 'GDP', 'PR', 'CON', 'TR']], 'AO')
        print(m.summary())
        print(f'Variance: {m.scale}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     AO   R-squared:                       0.843
Model:                            OLS   Adj. R-squared:                  0.799
Method:                 Least Squares   F-statistic:                     19.28
Date:                Tue, 03 Sep 2019   Prob (F-statistic):           1.16e-06
Time:                        08:07:55   Log-Likelihood:                 40.693
No. Observations:                  24   AIC:                            -69.39
Df Residuals:                      18   BIC:                            -62.32
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.4834      0.089      5.412      0.000       0.296       0.671
CON           -0.1007      0.026     -3.881      0.001      -0.155      -0.046
DEN        -3.403e-05      0.000     -0.323      0.750      -0.000       0.000
PR            -0.0044      0.001     -3.773      0.001      -0.007      -0.002
TR            -0.0612      0.017     -3.580      0.002      -0.097      -0.025
GDP            0.0308      0.003      8.932      0.000       0.024       0.038
==============================================================================
Omnibus:                        3.592   Durbin-Watson:                   2.322
Prob(Omnibus):                  0.166   Jarque-Bera (JB):                2.081
Skew:                           0.686   Prob(JB):                        0.353
Kurtosis:                       3.446   Cond. No.                     1.47e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.47e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
Variance: 0.002628573282679365
```

```
In [33]: m = forward_selected(cars[['AO', 'POP', 'DEN', 'GDP', 'PR', 'CON', 'TR']], 'AO')
         print(m.summary())
         print(f'Variance: {m.scale}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     AO   R-squared:                       0.842
Model:                            OLS   Adj. R-squared:                  0.808
Method:                 Least Squares   F-statistic:                     25.26
Date:                Tue, 03 Sep 2019   Prob (F-statistic):           2.23e-07
Time:                        08:05:19   Log-Likelihood:                 40.624
No. Observations:                  24   AIC:                            -71.25
Df Residuals:                      19   BIC:                            -65.36
Df Model:                           4
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.4831      0.087      5.541      0.000       0.301       0.666
GDP            0.0307      0.003      9.163      0.000       0.024       0.038
TR            -0.0623      0.016     -3.804      0.001      -0.097      -0.028
CON           -0.1001      0.025     -3.963      0.001      -0.153      -0.047
PR            -0.0045      0.001     -3.930      0.001      -0.007      -0.002
==============================================================================
Omnibus:                        3.521   Durbin-Watson:                   2.434
Prob(Omnibus):                  0.172   Jarque-Bera (JB):                2.151
Skew:                           0.717   Prob(JB):                        0.341
Kurtosis:                       3.310   Cond. No.                         437.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Variance: 0.0025046815666849172
```

In [34]:
```
def calcPress(X, Y, Ypred):
    X = np.matrix(X)
    h = np.matmul(X, np.matmul(np.linalg.inv(np.matmul(X.transpose(), X)), X.transpose
    press = 0
    #for i in range(lenth(Y)):
    press = press + (((Y - Ypred)**2 / (1-np.diagonal(h)))**2).sum()
    return press
```

In [35]:
```
def calcYpred(model, inputs):
    y = model.predict(inputs)
    return y
```

In [36]:
```
np.diagonal([[1,2],[3,4]])
```

Out[36]:
```
array([1, 4])
```

In [37]:
```
%%time
variables = ['GDP', 'PR', 'CON', 'TR']
#variables = variables.drop('y')
bestFormula = ''
bestAIC = 9999999
maxAIC = 0

fullModelMSE = smf.ols('AO ~ POP + DEN + GDP + PR + CON + TR', cars).fit().mse_resid

for i in range(1, len(variables)+1):
    combs = combinations(variables, i)
```

```
            for c in combs:
                formula = 'AO ~'
                for v in c:
                    formula = formula + f' + {v}'
                formula = formula.replace('~ + ', '~ ')
                fit = smf.ols(formula, cars).fit()
                fit.predict()
                if fit.aic < bestAIC:
                    bestAIC = fit.aic
                    bestFormula = formula
                if fit.aic > maxAIC:
                    maxAIC = fit.aic
                ypred = calcYpred(fit, cars[[x for x in c]])
                print(f'{formula} : ')
                press = str(calcPress(cars[[x for x in c]], cars.AO, ypred))[:6]
                cm = ((fit.mse_resid * fit.df_resid) / fullModelMSE) - len(cars) + 2*len(fit.p

                print(f'AIC: {str(fit.aic)[:8]} - PRESS: {press} - RSquared: {str(fit.rsquare
```

```
AO ~ GDP :
AIC: -52.8908 - PRESS: 0.0019 - RSquared: 0.5633 - m: 2 - Cm: 29.026
AO ~ PR :
AIC: -34.7088 - PRESS: 0.0105 - RSquared: 0.0685 - m: 2 - Cm: 84.578
AO ~ CON :
AIC: -33.0052 - PRESS: 0.0127 - RSquared: 6.7618 - m: 2 - Cm: 92.271
AO ~ TR :
AIC: -33.1466 - PRESS: 0.0105 - RSquared: 0.0058 - m: 2 - Cm: 91.612
AO ~ GDP + PR :
AIC: -57.2986 - PRESS: 0.0015 - RSquared: 0.6656 - m: 3 - Cm: 19.538
AO ~ GDP + CON :
AIC: -51.1158 - PRESS: 0.0026 - RSquared: 0.5674 - m: 3 - Cm: 30.568
AO ~ GDP + TR :
AIC: -58.3450 - PRESS: 0.0012 - RSquared: 0.6799 - m: 3 - Cm: 17.936
AO ~ PR + CON :
AIC: -34.5919 - PRESS: 0.0079 - RSquared: 0.1388 - m: 3 - Cm: 78.686
AO ~ PR + TR :
AIC: -32.7925 - PRESS: 0.0115 - RSquared: 0.0717 - m: 3 - Cm: 86.214
AO ~ CON + TR :
AIC: -31.2181 - PRESS: 0.0158 - RSquared: 0.0088 - m: 3 - Cm: 93.280
AO ~ GDP + PR + CON :
AIC: -59.6559 - PRESS: 0.0009 - RSquared: 0.7211 - m: 4 - Cm: 15.305
AO ~ GDP + PR + TR :
AIC: -58.7871 - PRESS: 0.0015 - RSquared: 0.7108 - m: 4 - Cm: 16.459
AO ~ GDP + CON + TR :
AIC: -58.9709 - PRESS: 0.0008 - RSquared: 0.7130 - m: 4 - Cm: 16.212
AO ~ PR + CON + TR :
AIC: -32.6888 - PRESS: 0.0086 - RSquared: 0.1422 - m: 4 - Cm: 80.297
```

```
AO ~ GDP + PR + CON + TR :
AIC: -71.2479 - PRESS: 0.0004 - RSquared: 0.8417 - m: 5 - Cm: 3.7693
Wall time: 589 ms
```

In [38]:
```python
print(bestFormula)
print(bestAIC)
print(maxAIC)
```

```
AO ~ GDP + PR + CON + TR
-71.2479549532554
0
```

0.2 Question 3

In [39]:
```python
jelly = pd.read_csv('Data/jelly.dat', sep='\s+')
jelly['Site Name'] = jelly.Site.map(lambda x: 'Salamander Bay' if x==2 else 'Dangar Is
jelly.head()
```

Out[39]:

	Breadth	Length	Site	Site Name
0	6.5	8.5	1	Dangar Island
1	6.0	9.0	1	Dangar Island
2	6.5	9.0	1	Dangar Island
3	7.0	9.0	1	Dangar Island
4	8.0	9.5	1	Dangar Island

In [40]:
```python
j1Fit = smf.ols('Breadth ~ Length', jelly[jelly.Site==1]).fit()
print(j1Fit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Breadth   R-squared:                       0.915
Model:                            OLS   Adj. R-squared:                  0.910
Method:                 Least Squares   F-statistic:                     214.6
Date:                Tue, 03 Sep 2019   Prob (F-statistic):           3.72e-12
Time:                        08:05:56   Log-Likelihood:                -29.794
No. Observations:                  22   AIC:                             63.59
Df Residuals:                      20   BIC:                             65.77
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -2.9374      0.920     -3.194      0.005      -4.856      -1.019
Length         1.0534      0.072     14.650      0.000       0.903       1.203
==============================================================================
Omnibus:                        0.410   Durbin-Watson:                   2.254
Prob(Omnibus):                  0.815   Jarque-Bera (JB):                0.043
```

```
Skew:                          -0.108   Prob(JB):                        0.979
Kurtosis:                       3.005   Cond. No.                         56.5
==============================================================================

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

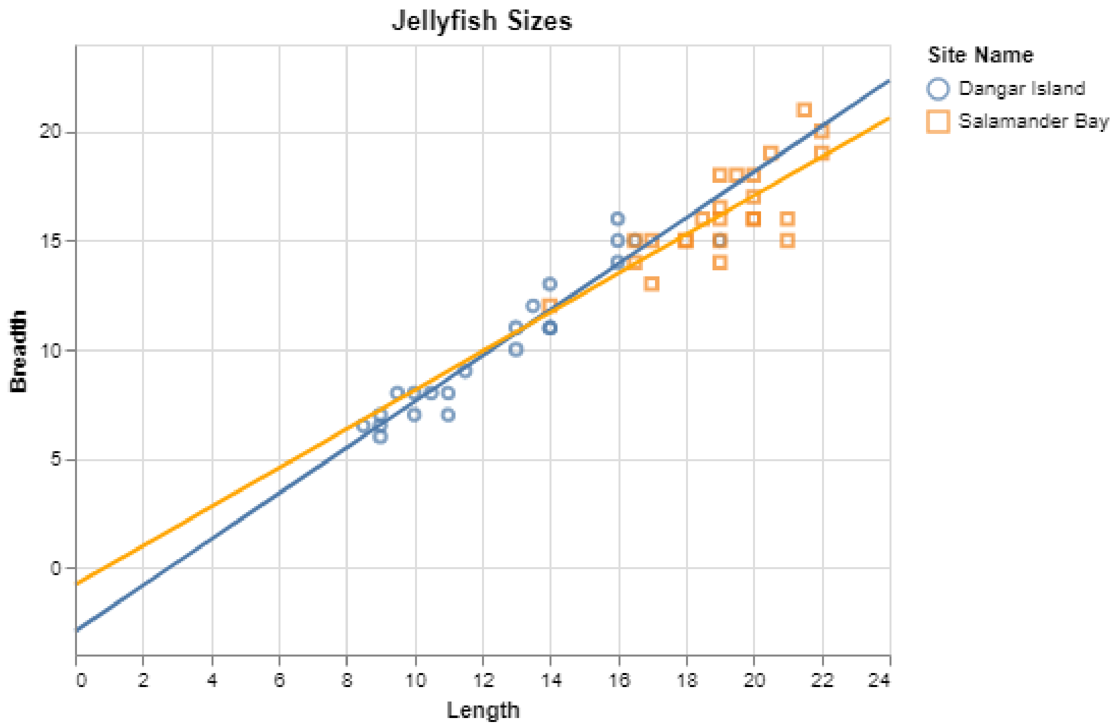
In [41]:
```
j2Fit = smf.ols('Breadth ~ Length', jelly[jelly.Site==2]).fit()
print(j2Fit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Breadth   R-squared:                       0.625
Model:                            OLS   Adj. R-squared:                  0.608
Method:                 Least Squares   F-statistic:                     36.68
Date:                Tue, 03 Sep 2019   Prob (F-statistic):           4.28e-06
Time:                        08:05:56   Log-Likelihood:                -40.551
No. Observations:                  24   AIC:                             85.10
Df Residuals:                      22   BIC:                             87.46
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.8003      2.826     -0.283      0.780      -6.661       5.060
Length         0.8924      0.147      6.056      0.000       0.587       1.198
==============================================================================
Omnibus:                        0.436   Durbin-Watson:                   1.849
Prob(Omnibus):                  0.804   Jarque-Bera (JB):                0.405
Skew:                          -0.274   Prob(JB):                        0.817
Kurtosis:                       2.676   Cond. No.                         194.
==============================================================================

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

In [42]:
```
alt.Chart(jelly, title='Jellyfish Sizes').mark_point().encode(x='Length', y='Breadth'
alt.Chart(pd.DataFrame({'Breadth':[j1Fit.params[0] + x*j1Fit.params[1] for x in range
                        'Length':[x for x in range(25)]})).mark_line().encode(x='Leng
alt.Chart(pd.DataFrame({'Breadth':[j2Fit.params[0] + x*j2Fit.params[1] for x in range
                        'Length':[x for x in range(25)]})).mark_line(color='orange').
```

Out[42]:



0.3 Question 4

```
In [43]: cloud = pd.read_csv('Data\cloud.dat', sep='\s+')
         #cloud.A = cloud.A + 1
         for col in ['S', 'C', 'P', 'E']:
             cloud[f'A{col}'] = cloud.A * cloud[col]
         cloud.head()
```

Out[43]:

	A	T	S	C	P	E	y	AS	AC	AP	AE
0	0	0	1.75	13.4	0.274	2	2.61	0.00	0.0	0.000	0
1	1	3	4.10	3.9	0.198	2	1.81	4.10	3.9	0.198	2
2	0	4	2.35	5.3	0.526	1	1.78	0.00	0.0	0.000	0
3	1	6	4.25	7.1	0.250	1	0.83	4.25	7.1	0.250	1
4	0	9	1.60	6.9	0.018	2	1.28	0.00	0.0	0.000	0

```
In [44]: cFit = smf.ols('y ~ A + T + S + C + P + E + AS + AC + AP + AE', cloud).fit()
         print(cFit.summary())
         print(cFit.scale)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.879
Model:                            OLS   Adj. R-squared:                  0.744
Method:                 Least Squares   F-statistic:                     6.518
```

```
Date:                Tue, 03 Sep 2019   Prob (F-statistic):            0.00473
Time:                        08:05:57   Log-Likelihood:               -0.43477
No. Observations:                  20   AIC:                             22.87
Df Residuals:                       9   BIC:                             33.82
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.3469      0.971     -0.357      0.729      -2.544       1.850
A              3.4404      1.213      2.835      0.020       0.695       6.185
T             -0.0147      0.006     -2.435      0.038      -0.028      -0.001
S              0.1267      0.226      0.560      0.589      -0.385       0.639
C              0.0746      0.041      1.825      0.101      -0.018       0.167
P              1.1913      0.677      1.759      0.112      -0.341       2.723
E              0.6372      0.367      1.735      0.117      -0.194       1.468
AS            -0.8519      0.277     -3.077      0.013      -1.478      -0.226
AC            -0.0588      0.078     -0.751      0.472      -0.236       0.118
AP             0.5972      3.313      0.180      0.861      -6.898       8.092
AE             0.0675      0.483      0.140      0.892      -1.025       1.160
==============================================================================
Omnibus:                        0.629   Durbin-Watson:                   2.691
Prob(Omnibus):                  0.730   Jarque-Bera (JB):                0.586
Skew:                           0.357   Prob(JB):                        0.746
Kurtosis:                       2.562   Cond. No.                     1.70e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.7e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
0.13589230416186726
```

In [45]:
```python
s = '= '

for p in ['A' , 'T' , 'S' , 'C' , 'P' , 'E' , 'AS' , 'AC' , 'AP' , 'AE']:
    symbol = '+' if cFit.params[p] >=0 else ''
    s = s + f'{symbol}{str(cFit.params[p])[:6]}*{p} '

s
```

Out[45]:
```
'= +3.4404*A -0.014*T +0.1267*S +0.0745*C +1.1913*P +0.6372*E -0.851*AS -0.058*AC +0.5
```

In [46]:
```python
cloud['residuals'] = cloud.y - cyPred
cloud['normResiduals'] = cFit.get_influence().resid_studentized_internal
cloud['yFit'] = cyPred
```

```
---------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-46-b15db7622b86> in <module>()
----> 1 cloud['residuals'] = cloud.y - cyPred
      2 cloud['normResiduals'] = cFit.get_influence().resid_studentized_internal
      3 cloud['yFit'] = cyPred


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

```
In [ ]: cFit.fvalue
```

```
In [ ]: cFit.get_influence().resid_studentized_internal
```

```
In [ ]: sm.qqplot(cFit.get_influence().resid_studentized_internal, line='45')
```

```
In [ ]: from statsmodels.graphics.gofplots import ProbPlot as pPlot
        from matplotlib import pyplot as plt
```

```
In [ ]: %matplotlib inline
        fig = pPlot(cyPred-cloud.y)
        plt.show()
```

```
In [ ]: cloud.head()
```

```
In [ ]: alt.Chart(cloud).mark_point().encode(x='yFit', y='normResiduals')
```

```
In [ ]: alt.Chart(cloud).mark_point().encode(x='A', y='normResiduals')
```

```
In [ ]: alt.Chart(cloud).mark_point().encode(x='AS', y='normResiduals')
```