Jane Downer
Math 484
A20452471

Final Project

In [1]:
```
import csv
import itertools
from   itertools import combinations
import matplotlib.pyplot as plt
import numpy as np
from   numpy import linalg
import pandas as pd
from   pandas import DataFrame
import patsy
from   patsy import dmatrices
import seaborn
import statsmodels as sm
from   statsmodels.stats.outliers_influence import variance_inflation_factor
from   statsmodels.tools.tools import add_constant
import statsmodels.api as sm
```

The functions below will be used throughout the document. --- Please scroll for responses.

In [2]:
```
### COLLINEARITY FUNCTIONS

def get_evals(data):
    C = corr_mat(data)
    evals,evecs = np.linalg.eigh(C)
    evals = [round(np.real_if_close(e).item(),7) for e in evals]
    evals = list(reversed(evals))
    return evals

def get_evecs(data):
    C = corr_mat(data)
    evals,evecs = np.linalg.eigh(C)
    Vs = [[round(v,5) for v in evecs[:,i].tolist()] for i in range(len(evecs))]
    Vs = list(reversed(Vs))
    V_names = []
    for i in range(len(data)-1):
```

```
        V_names.append('V' + str(i+1))
    return dict(zip(V_names,Vs))

def condition_indices(data):
    evals = get_evals(data)
    evals.sort(key = lambda x: np.abs(x),reverse=False)
    smallest = evals[0]
    Ks = []
    for i in range(len(evals)):
        Ks.append((np.sqrt(evals[i]/smallest)).round(3))
    return Ks

def condition_number(data):
    ks = condition_indices(data)
    return ks[-1]

def corr(x, y, **kwargs):
    coef = np.corrcoef(x, y)[0][1]
    label=''
    if np.abs(coef) <= 0.6:
        label = r'$\rho$ = ' + str(round(coef, 2))
    else:
        if np.abs(coef) > 0.7:
            label = r'$\rho$* = ' + str(round(coef, 2))
        if np.abs(coef) > 0.8:
            label = r'$\rho$** = ' + str(round(coef, 2))
        if np.abs(coef) > 0.9:
            label = r'$\rho$*** = ' + str(round(coef, 2))
    ax = plt.gca()
    ax.annotate(label, xy = (0.2, 0.8),
                size = 30, xycoords = ax.transAxes)

def VIFs(data):
    features = "+".join(data.columns[1:])
    y_col = data.columns[0]
    y, X = dmatrices(y_col + ' ~' + features, data, return_type='dataframe')
    vif = pd.DataFrame()
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif["Predictor"] = X.columns
    vif = vif.sort_values(by='VIF',axis=0, ascending = False)
    return vif
```

In [3]:

```
### MLR functions
```

```
def X_mat(data):
    x_df = data.iloc[:,1:]
    x_df.insert(0, "Ones", [1]*len(data), True)
    x_mat = np.array(x_df)
    return x_mat
def XTXinv_XT(data):
    X = X_mat(data)
    X_T = np.transpose(X)
    X_ = np.matmul(X_T,X)
    X_inv = np.linalg.inv(X_)
    X_mult = np.matmul(X_inv,X_T)
    return X_mult

def mlr(data):
    Y= np.matrix(data.iloc[:,0])
    Y = np.transpose(Y)
    X_term = XTXinv_XT(data)
    B = [b.item() for b in np.matmul(X_term,Y)]
    return  [round(value,3) for value in B]

def H(data):
    X = X_mat(data)
    X_term = XTXinv_XT(data)
    H = np.matmul(X,X_term)
    return H

def I(data):
    return(np.identity(len(data)))

def SSE_mlr(data):
    YT= np.matrix(data.iloc[:,0])
    Y = np.transpose(YT)
    I_H = np.subtract(I(data),H(data))
    right_term = np.matmul(I_H,Y)
    return np.matmul(YT,right_term).item()


def var_mlr(data):
    sse = SSE_mlr(data)
    n, p = len(data), len(data.columns)-1
    return (sse/(n-p-1))

def corr_mat(data):
    X = data.iloc[:,1:]
    return X.corr()
```

```
def predict(data):
    B = mlr(data)
    preds = []
    for i in range(len(data)):
        row = [1] + list(data.iloc[i,1:])
        preds.append(np.dot(row,B))
    return preds

def residuals(data):
    return [data.iloc[:,0][i] - predict(data)[i] for i in range(len(data.columns)-1)]
```

In [4]:

```
### STEPWISE REGRESSION FUNCTIONS

def round_sci(number):
    return float('{:0.3e}'.format(number))

def new_base(data,prev_base,best_pred,first_round = False):
    if first_round == True:
        new_base = pd.DataFrame({'const':[1]*len(data)})
    else:
        new_base = pd.concat([prev_base,data[best_pred]],axis=1)
    return new_base

def results_of_adding(data, best_pred_prev=[], best_pvals_prev=[],
                      base=None, alpha_E=0.15, alpha_R=0.15, add_msg = True):
    ignore = ['const'] if 'const' in data.columns else []
    ignore += ['BIO'] if best_pred_prev == [] else (['BIO'] + best_pred_prev)
    new_options = [c for c in data.columns if c not in ignore]
    X_list      = [pd.concat([base,data[c]],axis=1) for c in
                   data.columns if c in new_options]
    y = data.iloc[:,0]

    best_prev_d   = dict(zip(best_pred_prev,best_pvals_prev))
    all_d = dict(zip(new_options,[None]*len(new_options)))
    update_d = dict(zip(ignore[1:],best_pvals_prev))
    added_changes = {}
    better_worse_d = dict(zip(ignore[1:],['']*len(ignore[1:])))
    new_pred_prev = []

    added_new_p = 1
    if add_msg == True:
        print("Considering:")
    for X in X_list:
```

```
        added = X.columns[-1]
        columns    = X.columns[1:]
        this_model = sm.OLS(y, X).fit()
        for c in [col for col in columns if col not in best_pred_prev]:
            output = this_model.pvalues
            others_affected = [(p, output[p]) for p in best_pred_prev]
            added_changes[c] = others_affected
            pval = output[c]
            all_d[c] = pval
            msg = "( > alpha_E)" if pval > alpha_E else ""
            if add_msg == True:
                print((c.ljust(6) + ' p: ' + str(round_sci(pval))).ljust(25) + msg)
        for c in best_pred_prev:
            output = this_model.pvalues
            pair = (added,c)
            pval = output[c]
            added_changes[pair] = pval
            v_prev, v_now = best_prev_d[c], pval
            better_worse_d[added] = 'better' if v_prev <= v_now else 'worse'

    items = list(all_d.items()).copy()
    items.sort(key = lambda x: x[1], reverse = False)
    (best_pred, best_pval) = list(items)[0]
    if add_msg == True:
        print("\n" + 'Best this round: '.ljust(18) + str(best_pred))

    worse = True
    outside_worse = False
    while best_pvals_prev != [] and worse == True and len(best_prev_d) > 0:
        worse = False
        for i in range(len(best_prev_d.keys())):
            [k, v_prev] = [list(best_prev_d.keys())[i],
                           list(best_prev_d.values())[i]]
            v_now_options = added_changes[best_pred]
            v_now = [v for v in v_now_options if v[0] == k][0][1]
            if v_now > v_prev:
                worse = True
                outside_worse = True
                best_pval_s = '{:0.3e}'.format(v_now)
                bad_news = (('    ' + str(k) +
                             " p-value is worse than before: ").ljust(40) +
                            "(" + '{:0.3e}'.format((v_now))  + "  >  " +
                            '{:0.3e}'.format((v_prev)) + ".)")
                print(bad_news)
```

```
        exclude = []
        this_worse=True
        former = ''
        if len(items) > 1 and worse == True:
            del all_d[best_pred]
            items = list(all_d.items()).copy()
            items.sort(key = lambda x: x[1], reverse = False)
            former, (best_pred, best_pval) = best_pred, items[0]
            exclude.append(former)
            print('\n' + 'Next best option: '.ljust(18) + str(best_pred))
            best_pred_prev  = list(best_prev_d.keys())
            best_pvals_prev = list(best_prev_d.values())
            data_new = data[[c for c in data.columns if c not in exclude]]
            results_of_adding(data_new, best_pred_prev, best_pvals_prev, base, add_msg = False)
        if this_worse == True:
            worse = False
        if former == '':
            pass
    message = ''

    if outside_worse == False:
        messsage = 'Done! Add ' + str(best_pred)
    else:
        message = '\nCannot add any predictors -- Consider previous model.'
    update_d[best_pred] = best_pval
    new_pred_prev = list(update_d.keys())
    new_pval_prev = list(update_d.values())
    new_base = pd.concat([base,data[best_pred]],axis=1)
    return new_pred_prev, new_pval_prev, new_base, message
```

In [5]:

```
def rSubset(arr, r):
    return list(combinations(arr, r))

def get_Cps(subset_list,data,p):
    Cp_values = []
    n = len(data)
    X_ = sm.add_constant(data.iloc[:,1:])
    full_model = sm.OLS(data.BIO, X_).fit()
    resid_full = full_model.resid
    resid_full = [r**2 for r in list(resid_full)]
    MSE = np.mean(resid_full)
    for subset in subset_list:
        X             = sm.add_constant(data[[subset[0],subset[1]]])
        with_y        = pd.concat([data.BIO,X.iloc[:,1:]],axis=1)
```

```
        columns    = X.columns[1:]
        reduced_model = sm.OLS(with_y.BIO, X).fit()
        reduced_resid = reduced_model.resid
        resid_reduced = reduced_model.resid
        sum(resid_reduced)
        resid_reduced = [r**2 for r in list(resid_reduced)]
        SSE_p = np.sum(resid_reduced)
        Cp_values.append(SSE_p/MSE + (2*p)-n)
    return Cp_values

def pairs(Cp_list, subset_list, p):
    by_distance = [(Cp,  np.abs(Cp-p)) for Cp in Cp_list]
    by_distance.sort(key=lambda x: x[1],reverse= False)
    labels = [str(s[0]) + ' & ' + str(s[1]) for s in subset_list]
    pairs  = list(zip(labels,[v[0] for v in by_distance]))
    return [(p[0],p[1]) for p in pairs]
```

In [6]:

```
### PLOTS

def var_exp():
    seaborn.set(font_scale=1)
    evals = get_evals(large_bin)
    var_exp     = [(i/sum(evals))*100 for i in sorted(evals, reverse=True)]
    cum_var_exp = np.cumsum(var_exp)

    plt.figure(figsize=(8, 6))
    plt.bar( range(len(evals)), var_exp,     label='Variance explained by single PC')
    plt.step(range(len(evals)), cum_var_exp, label='Variance explained by all PCs')
    plt.ylabel('Proportion of Variance Explained')
    plt.xlabel('Principal Components')
    plt.title('Effects of PCs on Variance')
    plt.legend(loc="lower center", bbox_to_anchor=(0.5, -0.35))
    plt.xticks(np.arange(0, 19,  1 ))
    plt.yticks(np.arange(0, 110, 10))
    plt.tight_layout()
    plt.rcParams['figure.facecolor'] = 'white'
    plt.show()
def Cps():
    subsets = rSubset(small_data.columns[1:],2)
    Cps = get_Cps(subsets,small_data,2)
    p   = pairs(Cps, subsets,2)
    (label1,y1),(label2,y2),(label3,y3) = p[0],p[1],p[2]
    seaborn.set(font_scale=1)
    plt.figure(figsize=(5, 5))
```

```
    plt.scatter([1]*len(Cps), Cps, alpha=0.5)
    plt.xlim(0,2)
    plt.ylim(min(Cps)-10,max(Cps)+10)
    plt.tight_layout()
    plt.title('Cp values across subsets',fontsize=17)
    plt.text(0.2, 3, 'p = 2')
    plt.annotate(label1,(1,y1),xytext=(10,+10),
                textcoords="offset points",  ha='left', va='center')
    plt.annotate(label2,(1,y2),xytext=(-50,0),
                 textcoords="offset points", ha='left', va='center')
    plt.annotate(label2,(1,y2),xytext=(10,-10),
                 textcoords="offset points", ha='left', va='center')
    plt.tick_params(axis='x', which='both',bottom=False,top=False,labelbottom=False)
    plt.axhline(y=2,color='red',linestyle='-',linewidth=1,label='p')
    plt.ylabel('Cp',fontsize=14)
    plt.rcParams['figure.facecolor'] = 'white'
    plt.show()
def pairwise():
    seaborn.set(font_scale=2)
    dropped_vars = large_bin[['BIO','pH','BUF','Ca','Mg']]

    grid = seaborn.PairGrid(data= dropped_vars,
                            vars = ['BIO','pH','BUF','Ca','Mg'],
                            height = 4)
    grid = grid.map_upper(corr)
    grid = grid.map_lower(plt.scatter,alpha = 0.5)
    grid = grid.map_diag(plt.hist, bins = 10, alpha = 0.5)
```

In [7]:

```
# FORMATTING

def formatting_1(col):
    if col.name == 'Kj':
        return ['background: yellow' if np.abs(c) > 10 else "" for c in col.values]
    elif col.name == 'Lambdaj':
        return ['background: orange' if c < 0.01 else "" for c in col.values]

def formatting_2(col):
    if col.name == 'V18':
        return ['background: yellow' if np.abs(c) > 0.2 else "" for c in col.values]
    if col.name == "Predictor":
        return ["" for c in col.values]

def formatting_3(col):
    if col.name == 'VIF':
```

```
        return ['background: yellow' if c > 15 else "" for c in col.values]
    if col.name == "Predictor":
        return ["" for c in col.values]
```

Reading in the Data

In [8]:

```
def read_in(file):
    with open(file, mode='r') as text_file:
        lines=(text_file.read()).splitlines()
        reader = csv.reader(lines)
        parsed_csv = list(reader)
    cols = parsed_csv[0][0].split()
    rows = [line[0].split() for line in parsed_csv[1:]]
    d = { i : [] for i in cols}
    for row in rows:
        for i in range(len(cols)):
            if cols[i] in ['Type','Loc']:
                val = str(row[i])
            else:
                val = float(row[i])
            (d[cols[i]]).append(val)
            i += 1
    del d['Obs']
    return pd.DataFrame(d)

f1,f2 = 'small_data.txt','large_data.txt'
small_data, large_data = read_in(f1).iloc[:,2:],read_in(f2)
display(small_data.head())
display(large_data.head())
```

	BIO	SAL	pH	K	Na	Zn
0	676.0	33.0	5.00	1441.67	35184.5	16.4524
1	516.0	35.0	4.75	1299.19	28170.4	13.9852
2	1052.0	32.0	4.20	1154.27	26455.0	15.3276
3	868.0	30.0	4.40	1045.15	25072.9	17.3128
4	1008.0	33.0	5.55	521.62	31664.2	22.3312

	Loc	Type	BIO	H2S	SAL	Eh7	pH	BUF	P	K	Ca	Mg	Na	Mn	Zn	Cu	NH

	Loc	Type	BIO	H2S	SAL	Eh7	pH	BUF	P	K	Ca	Mg	Na	Mn	Zn	Cu	NH
0	OI	DVEG	676.0	-610.0	33.0	-290.0	5.00	2.34	20.238	1441.67	2150.00	5169.05	35184.5	14.2857	16.4524	5.02381	59.52
1	OI	DVEG	516.0	-570.0	35.0	-268.0	4.75	2.66	15.591	1299.19	1844.76	4358.03	28170.4	7.7285	13.9852	4.19019	51.37
2	OI	DVEG	1052.0	-610.0	32.0	-282.0	4.20	4.18	18.716	1154.27	1750.36	4041.27	26455.0	17.8066	15.3276	4.79221	68.78
3	OI	DVEG	868.0	-560.0	30.0	-232.0	4.40	3.60	22.821	1045.15	1674.36	3966.08	25072.9	49.1538	17.3128	4.09487	82.25
4	OI	DVEG	1008.0	-610.0	33.0	-318.0	5.55	1.90	37.843	521.62	3360.02	4609.39	31664.2	30.5229	22.3312	4.60131	70.90

Part I: Large Data

Consider the 14-predictor data set (LINTHALL.txt). Use the ordinary least square estimation to estimate the regression coefficients. Run the collinearity diagnostics and identify if there is any collinearity.

Replace categorical predictors with dummy variables

In [9]:

```
small_bin, large_bin = read_in(f1), read_in(f2)

new_cols = [pd.get_dummies(large_bin[c],drop_first=True) for c in ['Type','Loc']]
for i in range(len(new_cols)):
    large_bin[new_cols[i].columns]=new_cols[i]
large_bin = large_bin[large_bin.columns[2:]]

display(large_bin.head())
```

	BIO	H2S	SAL	Eh7	pH	BUF	P	K	Ca	Mg	Na	Mn	Zn	Cu	NH4	SHRT	TA
0	676.0	-610.0	33.0	-290.0	5.00	2.34	20.238	1441.67	2150.00	5169.05	35184.5	14.2857	16.4524	5.02381	59.524	0	
1	516.0	-570.0	35.0	-268.0	4.75	2.66	15.591	1299.19	1844.76	4358.03	28170.4	7.7285	13.9852	4.19019	51.378	0	
2	1052.0	-610.0	32.0	-282.0	4.20	4.18	18.716	1154.27	1750.36	4041.27	26455.0	17.8066	15.3276	4.79221	68.788	0	
3	868.0	-560.0	30.0	-232.0	4.40	3.60	22.821	1045.15	1674.36	3966.08	25072.9	49.1538	17.3128	4.09487	82.256	0	
4	1008.0	-610.0	33.0	-318.0	5.55	1.90	37.843	521.62	3360.02	4609.39	31664.2	30.5229	22.3312	4.60131	70.904	0	

Linear Regression:

(a) Coefficients:

In [10]:
```
display(mlr(large_bin))
```

```
[64.853,
 -0.872,
 -3.78,
 -0.135,
 47.813,
 32.505,
 0.154,
 -0.174,
 0.157,
 -0.149,
 -0.004,
 -7.333,
 -5.279,
 180.758,
 1.977,
 -465.798,
 444.338,
 -399.586,
 116.622]
```

(b) More thorough summary:

In [11]:
```
d = large_bin.copy()
X = d.iloc[:,1:]
X = sm.add_constant(X)
y = d.iloc[:,0]
regr = sm.OLS(y, X).fit()

regr.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	BIO	**R-squared:**	0.872
Model:	OLS	**Adj. R-squared:**	0.784
Method:	Least Squares	**F-statistic:**	9.863
Date:	Sun, 05 Dec 2021	**Prob (F-statistic):**	1.79e-07
Time:	23:49:34	**Log-Likelihood:**	-309.20
No. Observations:	45	**AIC:**	656.4

Df Residuals:	26	**BIC:**	690.7
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	64.8525	3245.420	0.020	0.984	-6606.205	6735.910
H2S	-0.8720	2.713	-0.321	0.750	-6.448	4.704
SAL	-3.7795	24.557	-0.154	0.879	-54.257	46.698
Eh7	-0.1347	2.028	-0.066	0.948	-4.304	4.035
pH	47.8125	351.410	0.136	0.893	-674.522	770.147
BUF	32.5055	113.370	0.287	0.777	-200.529	265.540
P	0.1545	2.628	0.059	0.954	-5.248	5.557
K	-0.1743	0.587	-0.297	0.769	-1.381	1.032
Ca	0.1565	0.151	1.038	0.309	-0.153	0.466
Mg	-0.1493	0.337	-0.443	0.662	-0.843	0.544
Na	-0.0035	0.023	-0.155	0.878	-0.050	0.043
Mn	-7.3326	6.195	-1.184	0.247	-20.066	5.401
Zn	-5.2789	23.086	-0.229	0.821	-52.733	42.175
Cu	180.7575	112.171	1.611	0.119	-49.813	411.328
NH4	1.9772	3.340	0.592	0.559	-4.889	8.843
SHRT	-465.7983	189.506	-2.458	0.021	-855.333	-76.263
TALL	444.3379	437.120	1.017	0.319	-454.176	1342.852
SI	-399.5857	445.947	-0.896	0.378	-1316.242	517.071
SM	116.6219	337.766	0.345	0.733	-577.666	810.910

Omnibus:	5.684	**Durbin-Watson:**	1.884
Prob(Omnibus):	0.058	**Jarque-Bera (JB):**	7.823
Skew:	0.054	**Prob(JB):**	0.0200
Kurtosis:	5.040	**Cond. No.**	1.31e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.31e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Collinearity:

In [12]:
```
ks      = condition_indices(large_bin)
evals   = get_evals(large_bin)
d       = pd.DataFrame({'Kj': ks})
d.index = d.index + 1
d.insert(1,'Lambdaj',evals,True)
d.style.apply(formatting_1)
```

Out[12]:

	Kj	Lambdaj
1	1.000000	5.663963
2	1.546000	4.453990
3	2.231000	2.391811
4	2.428000	1.809418
5	2.718000	0.908225
6	3.736000	0.797425
7	4.475000	0.450855
8	5.195000	0.392973
9	6.066000	0.379933
10	7.771000	0.231561
11	7.903000	0.169804
12	8.465000	0.125981
13	11.258000	0.087823

	Kj	Lambdaj
14	12.014000	0.046499
15	16.958000	0.037089
16	19.497000	0.031316
17	26.606000	0.015040
18	30.003000	0.006292

There are four condition indices over 10. This suggests four sets of collinearity. The last eigenvalue is small, indicating that we should look at the last eigenvector for more information. We continue to investigate the collinearity in the data in Part II.

Part II: Principal Components

V18 is associated with a small eigenvalue, so we look there first.

In [13]:
```
v = VIFs(large_bin)
v.index = v.index-1
v.iloc[1:,1]
d = pd.DataFrame(get_evecs(large_bin)).iloc[:,17:]
d.insert(0,'Predictor',v.iloc[1:,1],True)
d.index = d.Predictor
d = d.drop(columns=['Predictor'])
d.style.apply(formatting_2)
```

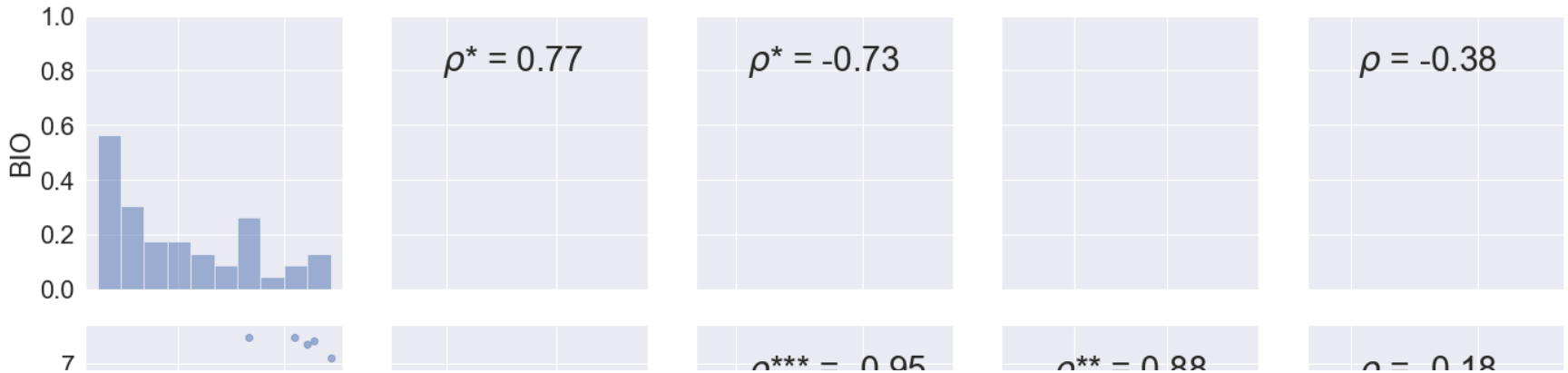
Out[13]:

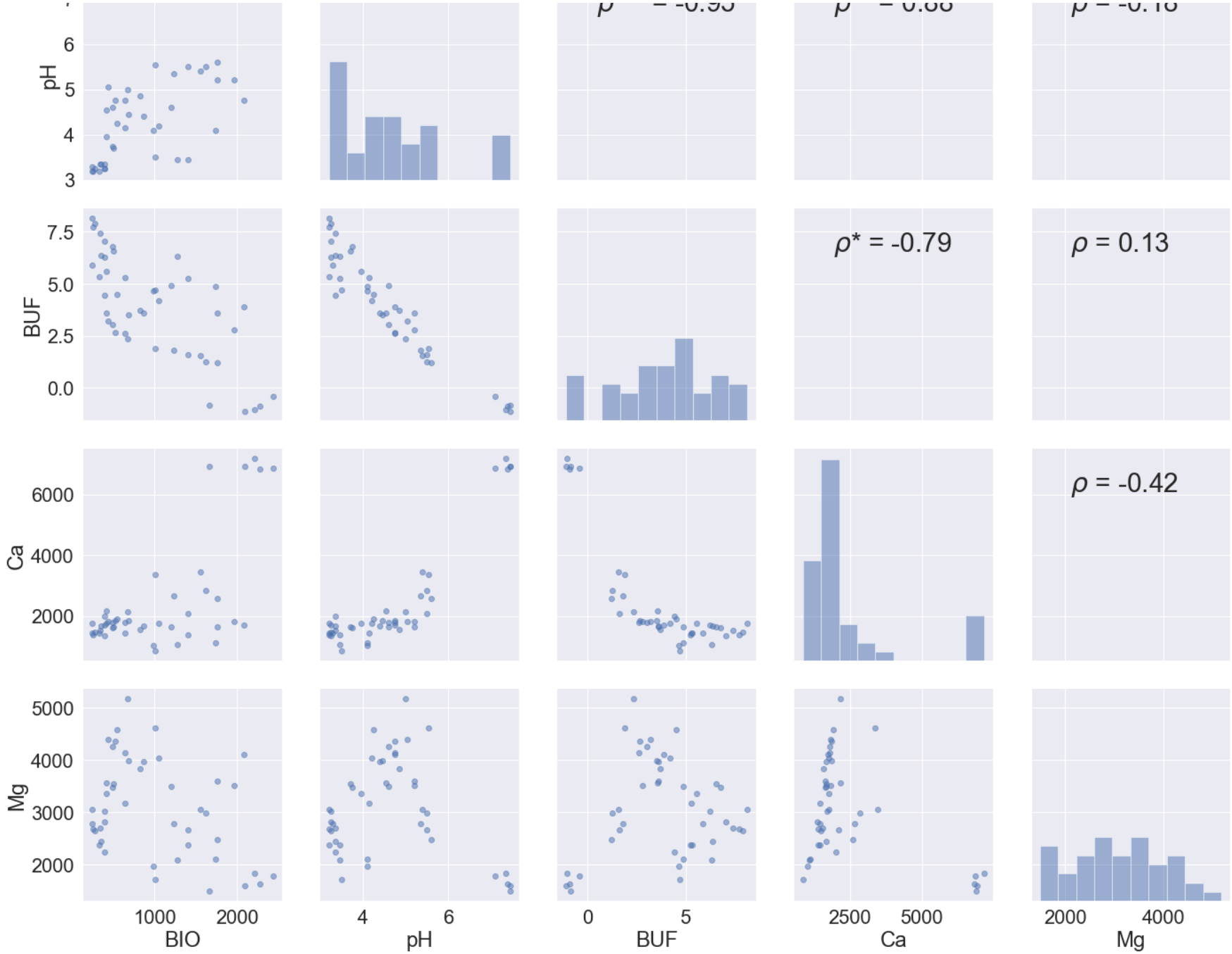
Predictor	V18
H2S	0.060750
SAL	-0.056720
Eh7	0.018400
pH	-0.723460
BUF	-0.349310

	V18
Predictor	
P	-0.001590
K	-0.139340
Ca	0.272270
Mg	0.371320
Na	-0.094190
Mn	-0.187250
Zn	-0.042930
Cu	-0.012950
NH4	0.024550
SHRT	0.053540
TALL	0.180420
SI	0.121010
SM	0.138970

The relatively large magnitudes associated with predictors `[pH, BUF, Ca, MG]` indicate they may be contributing to collinearity in the data.

In [14]:
```
pairwise()
```





The pairwise plot above and the corresponding correlation coefficients confirm that the predictors mentioned above have collinear relationships.

`['BIO','pH','BUF','Ca','Mg']` also have VIFs over 15, which is consistent with this result.

In [15]:
```
v = VIFs(large_bin).iloc[1:,:]
p =  v.Predictor
v.insert(0,'Predictor',v.iloc[0:,1],True)
v       = v.drop(columns=['Predictor'])
v.index = p
print("Predictors with VIFs over 15:")
v.style.apply(formatting_3)
```

```
Predictors with VIFs over 15:
```

Out[15]:

Predictor	VIF
pH	89.703459
Mg	46.893230
BUF	37.716469
Ca	31.341308
SI	21.113701
TALL	20.286198
Zn	17.068175
K	14.249286
SM	12.112412
NH4	11.647088
Na	11.244450
Mn	10.742769
Cu	6.320638
SAL	3.897815
SHRT	3.812803
H2S	3.238793
Eh7	2.624919

Predictor	VIF
P	2.456062

Removing the four predictors mentioned above yields the following results:

In [16]:
```
to_drop = ['BIO','pH','BUF','Ca','Mg']
new_columns = ['BIO'] +[c for c in large_bin.columns if c not in to_drop]
reduced_round_1 = large_bin[new_columns]

print('Condition number of original dataset: ' + str(condition_number(large_bin)))
print('Condition number of reduced dataset:  ' + str(condition_number(reduced_round_1)))
print('\nRegression coefficients of reduced dataset: \n\n' + str(mlr(reduced_round_1)))
```

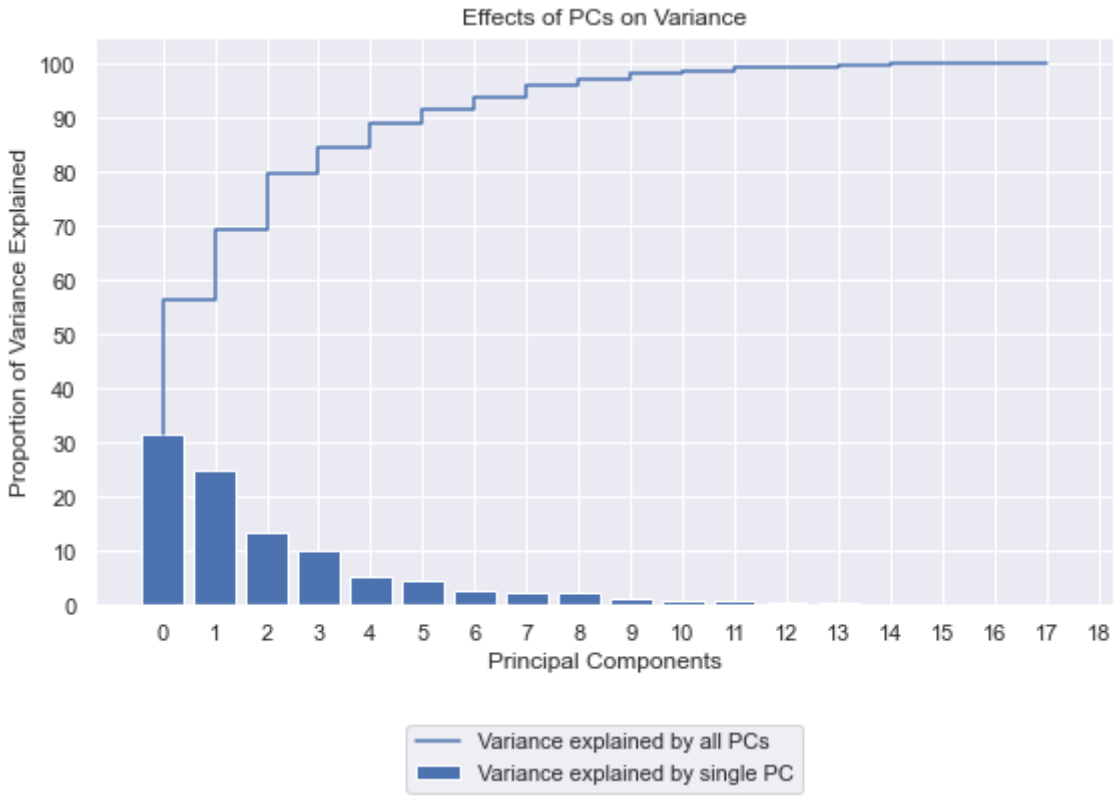
```
Condition number of original dataset: 30.003
Condition number of reduced dataset:  10.303

Regression coefficients of reduced dataset:

[1016.773, -1.177, -13.307, 1.294, -0.935, -0.623, 0.004, -7.919, -16.557, 216.972, -0.459, -296.352, 698.703,
173.233, 365.524]
```

These findings are consistent with the graph below, which shows that over 90% of the variance is explained by about half the predictors in the full model.

In [17]:
```
var_exp()
```



Part III

1. Stepwise Regression

In [18]:
```
small_data.head()
```

Out[18]:

	BIO	SAL	pH	K	Na	Zn
0	676.0	33.0	5.00	1441.67	35184.5	16.4524
1	516.0	35.0	4.75	1299.19	28170.4	13.9852
2	1052.0	32.0	4.20	1154.27	26455.0	15.3276
3	868.0	30.0	4.40	1045.15	25072.9	17.3128
4	1008.0	33.0	5.55	521.62	31664.2	22.3312

In [19]:

```
new_base0 = new_base(small_data,[],[],True)
best_pred1, p_best1, new_base1, message = results_of_adding(small_data, [], [], new_base0)
print(message)
best_pred2, p_best2, new_base2,message = results_of_adding(small_data, best_pred1, p_best1, new_base1)
print(message)
best_pred3, p_best3, new_base3,message = results_of_adding(small_data, best_pred2, p_best2, new_base2)
print(message)
```

```
Considering:
SAL     p: 0.5001          ( > alpha_E)
pH      p: 4.433e-10
K       p: 0.1775          ( > alpha_E)
Na      p: 0.0706
Zn      p: 4.566e-06

Best this round:  pH

Considering:
SAL     p: 0.517           ( > alpha_E)
K       p: 0.02106
Na      p: 0.01008
Zn      p: 0.3338          ( > alpha_E)

Best this round:  Na

Considering:
SAL     p: 0.7871          ( > alpha_E)
K       p: 0.6322          ( > alpha_E)
Zn      p: 0.4888          ( > alpha_E)

Best this round:  Zn
   pH p-value is worse than before:     (4.074e-06  >  4.433e-10.)
   Na p-value is worse than before:     (1.407e-02  >  1.008e-02.)

Next best option: K
   Na p-value is worse than before:     (2.212e-01  >  1.008e-02.)

Next best option: SAL
   Na p-value is worse than before:     (1.333e-02  >  1.008e-02.)

Cannot add any predictors -- Consider previous model.
```

2. Subset Selection

In [20]:

```
subsets = rSubset(small_data.columns[1:],2)
```

```
print('Possible two-predictor combinations:')
display(subsets)
```

```
Possible two-predictor combinations:
[('SAL', 'pH'),
 ('SAL', 'K'),
 ('SAL', 'Na'),
 ('SAL', 'Zn'),
 ('pH', 'K'),
 ('pH', 'Na'),
 ('pH', 'Zn'),
 ('K', 'Na'),
 ('K', 'Zn'),
 ('Na', 'Zn')]
```

In [21]:

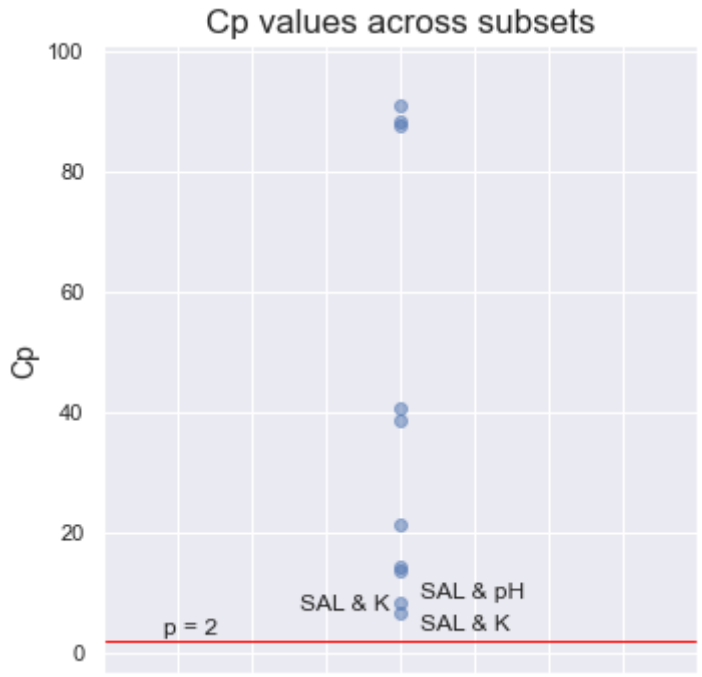
```
get_Cps(subsets,small_data,2)
```

Out[21]:

```
[14.307399904306834,
 91.00387407502203,
 87.63158745456639,
 21.388953525575403,
 8.146618594218388,
 6.632606331437131,
 13.626497840741223,
 88.08634936066045,
 40.55178954924526,
 38.48406779795141]
```

In [22]:

```
Cps()
```



The subsets ('SAL','K'), ('SAL','pH'), and ('SAL','K') are closest to p, which is 2. This suggests that these are the best subsets.