In [1]:
```
# Group: Dominic Klusek, Johnathan Rozen
# CSC 732 HW# 1 Part 1
```

Description of Ionosphere Dataset

Data Set Information:

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this databse are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

Attribute Information

- There are 34 numeric (float values) attributes, all with values between -1.0 to 1.0, and are reading of the pulse numbers for the Goose Bay System
- The dataset contains 2 class 'g' for Good and 'b' for Bad
- The number of instances in each class are 'b' : 126 'g' : 225
  All data was captured using ther same setup of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts.

Listing 1a: Load libraries

In [2]:
```
# increase width of jupyter notebook cells
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [3]:
```
# Load Libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import seaborn as sns
import numpy as np
```

In [4]:
```
# pandas limites the output of functions to prevent overlarge outputs so setting th
ese options will force the library to print all columns
import pandas as pd
pd.set_option('display.max_columns', 500)

# prevent warning because some functions in sklearn have deprecation warning (ONLY
ENABLE FOR FINAL CODE)
import warnings
warnings.filterwarnings("ignore")
```

Listing 1b: Loading the Ionosphere Dataset

In [5]:
```
# load dataset
filename = 'Dataset/ionosphere.data'
names = ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'B10', 'B11', 'B12',
'B13', 'B14', 'B15', 'B16', 'B17', 'B18', 'B19', 'C20', 'C21', 'C22', 'C23', 'C24',
'C25', 'C26', 'C27', 'C28', 'C29', 'D30', 'D31', 'D32', 'D33', 'D34', 'class']
dataset = read_csv(filename, names=names, delimiter=',')
```

Listing 2: Dimensionsof thedataset. Peek at the data itself. Statistical summary of all attributes. Break down of the data by the class variable

In [6]:
```
# Print shape of dataset
print(dataset.shape)
```

```
(351, 35)
```

So the dataset has 351 examples, with 35 attributes for each data example

In [7]:
```
# Peak at first 20 lines of dataset
print(dataset.head(20)) # this is the final output for when we submit, but its ugly
#dataset.head(20)
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	\
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	
5	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	
6	1	0	0.97588	-0.10602	0.94601	-0.20800	0.92806	-0.28350	0.85996	
7	0	0	0.00000	0.00000	0.00000	0.00000	1.00000	-1.00000	0.00000	
8	1	0	0.96355	-0.07198	1.00000	-0.14333	1.00000	-0.21313	1.00000	
9	1	0	-0.01864	-0.08459	0.00000	0.00000	0.00000	0.00000	0.11470	
10	1	0	1.00000	0.06655	1.00000	-0.18388	1.00000	-0.27320	1.00000	
11	1	0	1.00000	-0.54210	1.00000	-1.00000	1.00000	-1.00000	1.00000	
12	1	0	1.00000	-0.16316	1.00000	-0.10169	0.99999	-0.15197	1.00000	
13	1	0	1.00000	-0.86701	1.00000	0.22280	0.85492	-0.39896	1.00000	
14	1	0	1.00000	0.07380	1.00000	0.03420	1.00000	-0.05563	1.00000	
15	1	0	0.50932	-0.93996	1.00000	0.26708	-0.03520	-1.00000	1.00000	
16	1	0	0.99645	0.06468	1.00000	-0.01236	0.97811	0.02498	0.96112	
17	0	0	0.00000	0.00000	-1.00000	-1.00000	1.00000	1.00000	-1.00000	
18	1	0	0.67065	0.02528	0.66626	0.05031	0.57197	0.18761	0.08776	
19	0	0	1.00000	-1.00000	0.00000	0.00000	0.00000	0.00000	1.00000	

	B10	B11	B12	B13	B14	B15	B16	B17	\
0	0.03760	0.85243	-0.17755	0.59755	-0.44945	0.60536	-0.38223	0.84356	
1	-0.04549	0.50874	-0.67743	0.34432	-0.69707	-0.51685	-0.97515	0.05499	
2	0.01198	0.73082	0.05346	0.85443	0.00827	0.54591	0.00299	0.83775	
3	0.00000	0.00000	0.00000	0.00000	0.00000	-1.00000	0.14516	0.54094	
4	-0.16399	0.52798	-0.20275	0.56409	-0.00712	0.34395	-0.27457	0.52940	
5	0.06637	0.03786	-0.06302	0.00000	0.00000	-0.04572	-0.15540	-0.00343	
6	-0.27342	0.79766	-0.47929	0.78225	-0.50764	0.74628	-0.61436	0.57945	
7	0.00000	-1.00000	-1.00000	0.00000	0.00000	0.00000	0.00000	1.00000	
8	-0.36174	0.92570	-0.43569	0.94510	-0.40668	0.90392	-0.46381	0.98305	
9	-0.26810	-0.45663	-0.38172	0.00000	0.00000	-0.33656	0.38602	-0.37133	
10	-0.43107	1.00000	-0.41349	0.96232	-0.51874	0.90711	-0.59017	0.89230	
11	0.36217	1.00000	-0.41119	1.00000	1.00000	1.00000	-1.00000	1.00000	
12	-0.19277	0.94055	-0.35151	0.95735	-0.29785	0.93719	-0.34412	0.94486	
13	-0.12090	1.00000	0.35147	1.00000	0.07772	1.00000	-0.14767	1.00000	
14	0.08764	1.00000	0.19651	1.00000	0.20328	1.00000	0.12785	1.00000	
15	-1.00000	0.43685	-1.00000	0.00000	0.00000	-1.00000	-0.34265	-0.37681	
16	0.02312	0.99274	0.07808	0.89323	0.10346	0.94212	0.05269	0.88809	
17	1.00000	-1.00000	1.00000	1.00000	-1.00000	1.00000	1.00000	-1.00000	
18	0.34081	0.63621	0.12131	0.62099	0.14285	0.78637	0.10976	0.58373	
19	1.00000	1.00000	-1.00000	-0.71875	1.00000	0.00000	0.00000	-1.00000	

	B18	B19	C20	C21	C22	C23	C24	C25	\
0	-0.38542	0.58212	-0.32192	0.56971	-0.29674	0.36946	-0.47357	0.56811	
1	-0.62237	0.33109	-1.00000	-0.13151	-0.45300	-0.18056	-0.35734	-0.20332	
2	-0.13644	0.75535	-0.08540	0.70887	-0.27502	0.43385	-0.12062	0.57528	
3	-0.39330	-1.00000	-0.54467	-0.69975	1.00000	0.00000	0.00000	1.00000	
4	-0.21780	0.45107	-0.17813	0.05982	-0.35575	0.02309	-0.52879	0.03286	
5	-0.10196	-0.11575	-0.05414	0.01838	0.03669	0.01519	0.00888	0.03513	
6	-0.68086	0.37852	-0.73641	0.36324	-0.76562	0.31898	-0.79753	0.22792	
7	1.00000	-1.00000	-1.00000	0.00000	0.00000	0.00000	0.00000	1.00000	
8	-0.35257	0.84537	-0.66020	0.75346	-0.60589	0.69637	-0.64225	0.85106	
9	0.15018	0.63728	0.22115	0.00000	0.00000	0.00000	0.00000	-0.14803	
10	-0.66474	0.69876	-0.70997	0.70645	-0.76320	0.63081	-0.80544	0.55867	
11	-0.29354	1.00000	-0.93599	1.00000	1.00000	1.00000	1.00000	1.00000	
12	-0.28106	0.90137	-0.43383	0.86043	-0.47308	0.82987	-0.51220	0.84080	
13	-1.00000	1.00000	-1.00000	0.61831	0.15803	1.00000	0.62349	1.00000	
14	0.10561	1.00000	0.27087	1.00000	0.44758	1.00000	0.41750	1.00000	
15	0.03623	1.00000	-1.00000	0.00000	0.00000	0.00000	0.00000	-0.16253	
16	0.11120	0.86104	0.08631	0.81633	0.11830	0.83668	0.14442	0.81329	
17	-1.00000	-1.00000	1.00000	1.00000	-1.00000	-1.00000	1.00000	-1.00000	
18	0.18151	0.14395	0.41224	0.53888	0.21326	0.51420	0.22625	0.48838	

Looking at the head values it can be seen that values range from -1 to 1 for all attributes besides the class label which is either a 'b' or 'g'.

In [8]:
```
# Print statistical descroptions of the dataset
print(dataset.describe()) # final output for submission
#dataset.describe()
```

```
               A1     A2          A3          A4          A5          A6  \
count  351.000000  351.0  351.000000  351.000000  351.000000  351.000000
mean     0.891738    0.0    0.641342    0.044372    0.601068    0.115889
std      0.311155    0.0    0.497708    0.441435    0.519862    0.460810
min      0.000000    0.0   -1.000000   -1.000000   -1.000000   -1.000000
25%      1.000000    0.0    0.472135   -0.064735    0.412660   -0.024795
50%      1.000000    0.0    0.871110    0.016310    0.809200    0.022800
75%      1.000000    0.0    1.000000    0.194185    1.000000    0.334655
max      1.000000    0.0    1.000000    1.000000    1.000000    1.000000

               A7          A8          A9         B10         B11         B12  \
count  351.000000  351.000000  351.000000  351.000000  351.000000  351.000000
mean     0.550095    0.119360    0.511848    0.181345    0.476183    0.155040
std      0.492654    0.520750    0.507066    0.483851    0.563496    0.494817
min     -1.000000   -1.000000   -1.000000   -1.000000   -1.000000   -1.000000
25%      0.211310   -0.054840    0.087110   -0.048075    0.021120   -0.065265
50%      0.728730    0.014710    0.684210    0.018290    0.667980    0.028250
75%      0.969240    0.445675    0.953240    0.534195    0.957895    0.482375
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              B13         B14         B15         B16         B17         B18  \
count  351.000000  351.000000  351.000000  351.000000  351.000000  351.000000
mean     0.400801    0.093414    0.344159    0.071132    0.381949   -0.003617
std      0.622186    0.494873    0.652828    0.458371    0.618020    0.496762
min     -1.000000   -1.000000   -1.000000   -1.000000   -1.000000   -1.000000
25%      0.000000   -0.073725    0.000000   -0.081705    0.000000   -0.225690
50%      0.644070    0.030270    0.601940    0.000000    0.590910    0.000000
75%      0.955505    0.374860    0.919330    0.308975    0.935705    0.195285
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              B19         C20         C21         C22         C23         C24  \
count  351.000000  351.000000  351.000000  351.000000  351.000000  351.000000
mean     0.359390   -0.024025    0.336695    0.008296    0.362475   -0.057406
std      0.626267    0.519076    0.609828    0.518166    0.603767    0.527456
min     -1.000000   -1.000000   -1.000000   -1.000000   -1.000000   -1.000000
25%      0.000000   -0.234670    0.000000   -0.243870    0.000000   -0.366885
50%      0.576190    0.000000    0.499090    0.000000    0.531760    0.000000
75%      0.899265    0.134370    0.894865    0.188760    0.911235    0.164630
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              C25         C26         C27         C28         C29         D30  \
count  351.000000  351.000000  351.000000  351.000000  351.000000  351.000000
mean     0.396135   -0.071187    0.541641   -0.069538    0.378445   -0.027907
std      0.578451    0.508495    0.516205    0.550025    0.575886    0.507974
min     -1.000000   -1.000000   -1.000000   -1.000000   -1.000000   -1.000000
25%      0.000000   -0.332390    0.286435   -0.443165    0.000000   -0.236885
50%      0.553890   -0.015050    0.708240   -0.017690    0.496640    0.000000
75%      0.905240    0.156765    0.999945    0.153535    0.883465    0.154075
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              D31         D32         D33         D34
count  351.000000  351.000000  351.000000  351.000000
mean     0.352514   -0.003794    0.349364    0.014480
std      0.571483    0.513574    0.522663    0.468337
min     -1.000000   -1.000000   -1.000000   -1.000000
25%      0.000000   -0.242595    0.000000   -0.165350
50%      0.442770    0.000000    0.409560    0.000000
75%      0.857620    0.200120    0.813765    0.171660
max      1.000000    1.000000    1.000000    1.000000
```

Looking at statisitical decription of the distibution of the data could be helpful for later analysis or data pre-processing

In [9]:
```
# Print class distribution of dataset
print(dataset.groupby('class').size())
```

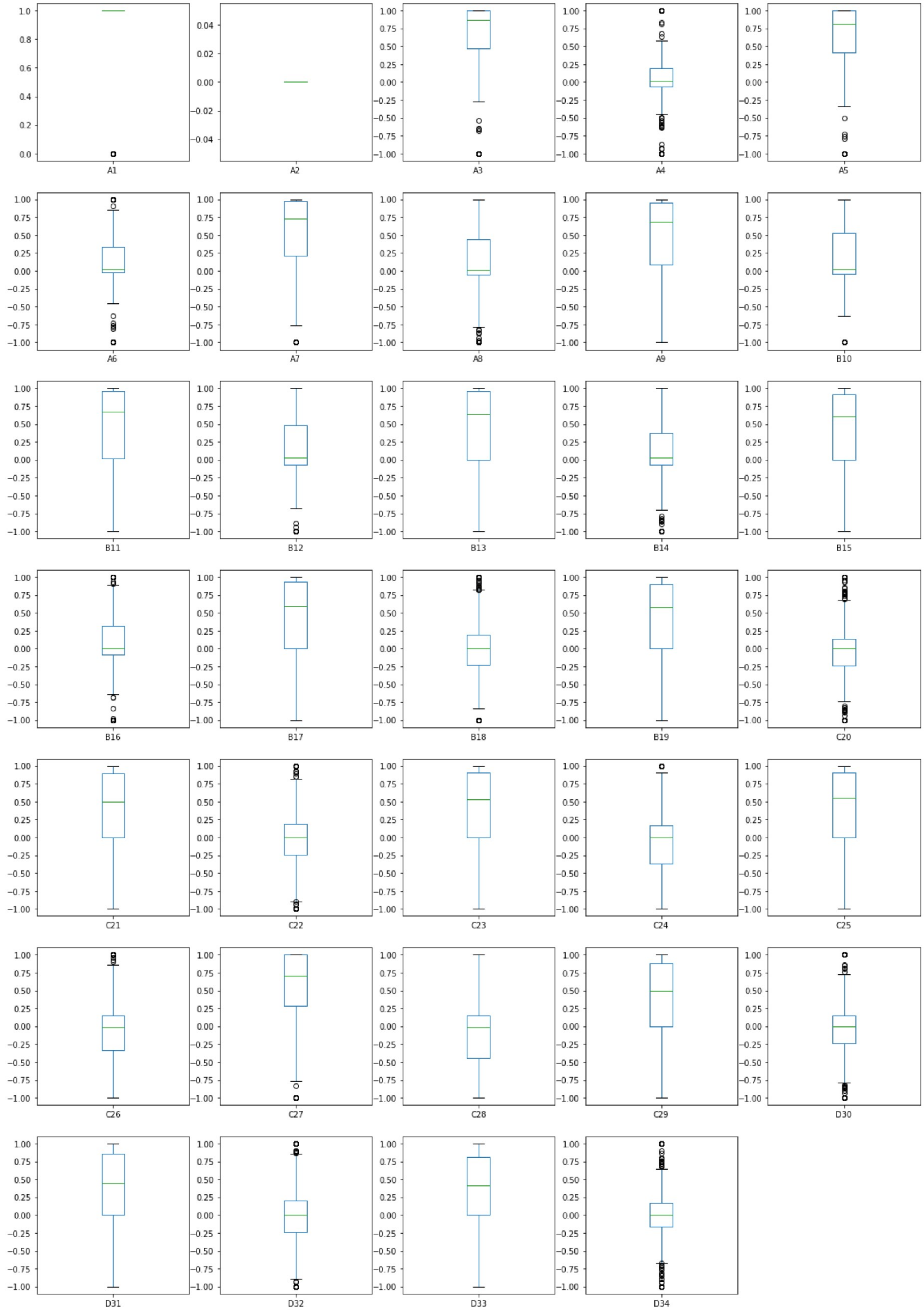
```
class
b    126
g    225
dtype: int64
```

The dataset has a slight imbalance with the 'g' class having roughly 2x the amount of examples than the 'b' class; however, with such a small dataset this imbalance shouldn't affect our model's performance in a significant manner.

Listing 3: Univariate plots to better understand each attribute. Multivariate plots to better understand the relationships between attributes.

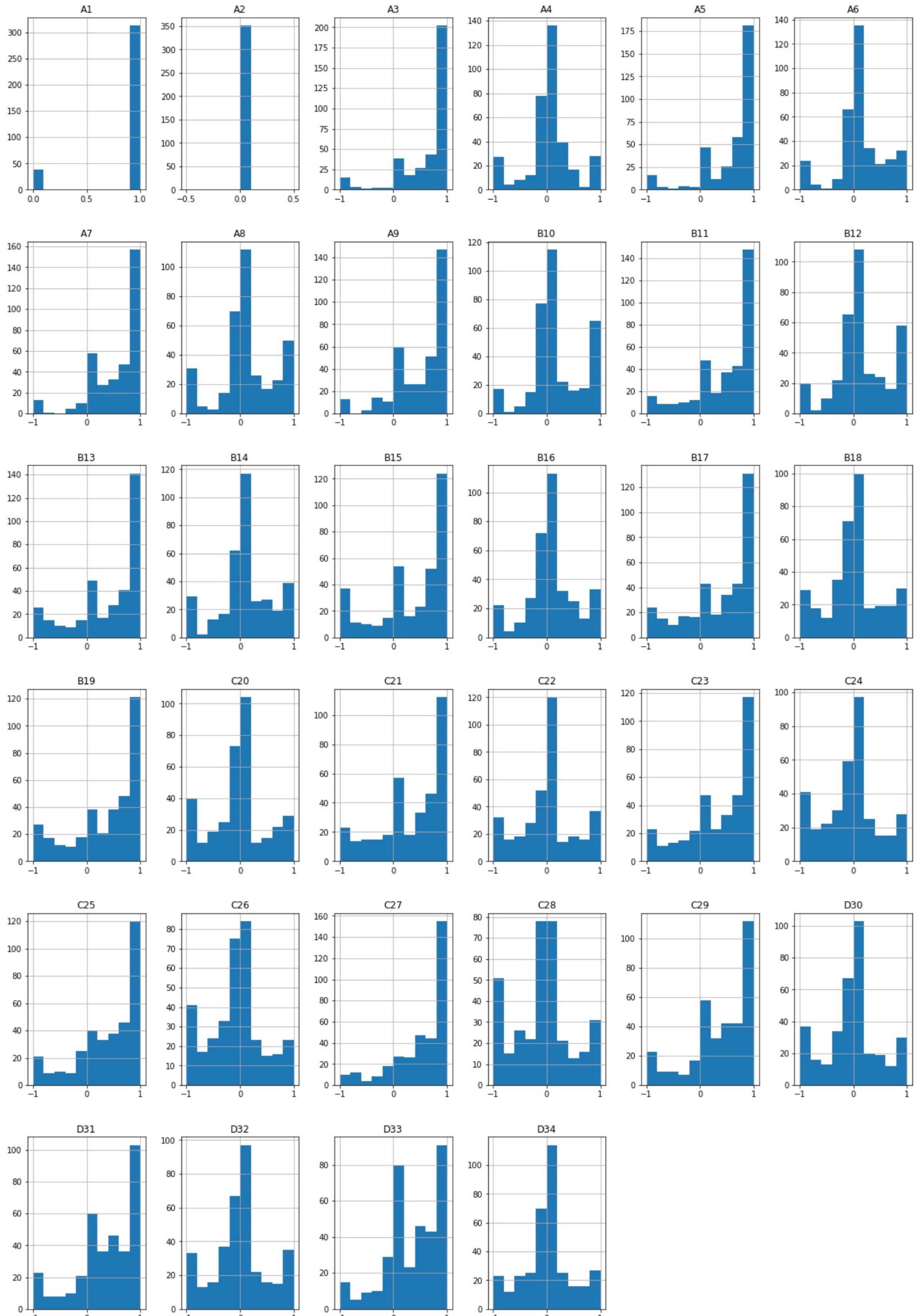
In [10]:

```
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(7,5), sharex=False, sharey=False, f
igsize=(20,30))
plt.show()
```



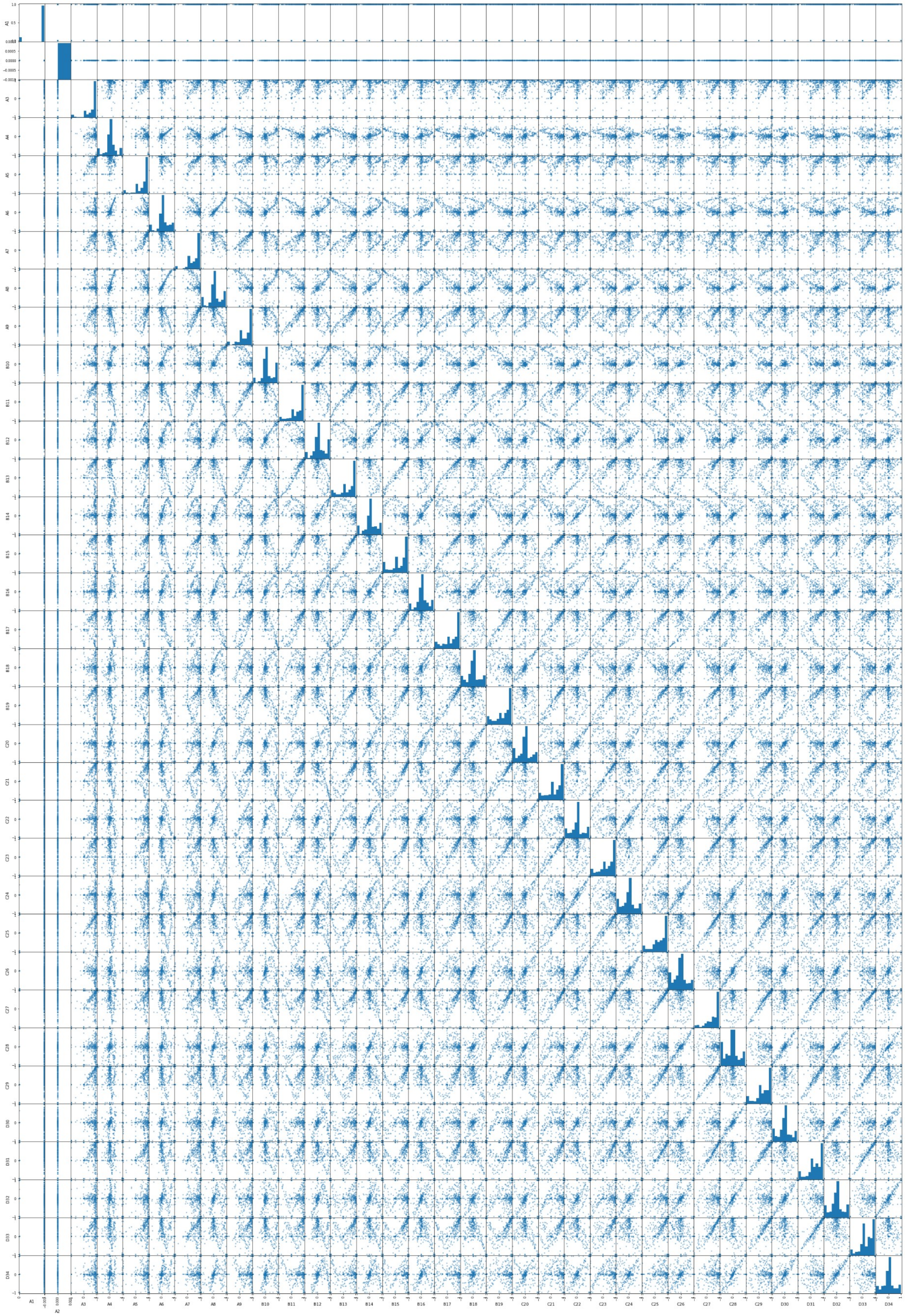
Box and whisker plots are a great way to visualize the statistics that are displayed when utilizing the describe function on a Pandas Dataframe. We can also see that there are not a large number of outlier points for most of the attributes, and that off numbered attributes are skewed towards 1, while even numbered attributes are most centered.

In [11]:
```
# histograms
dataset.hist(figsize=(20,30))
plt.show()
```



Histograms show the frequency of various data values in are are useful when trying to determine the uniformity of data. Looking at the univariate histograms we can see that non of the attributes have a uniform histogram and we can see the skews that were evident in the box and whisker plots.

In [12]:
```
# scatter plot matrix
scatter_matrix(dataset, figsize=(40,60))
plt.show()
```



A scatter matrix is an excellent way to see how clustered attributes are to one another, and could be used to visually observe the correlation that attributes have to one another. Looking at the scatter plots odd and even attributes have a positive correlation to other odd and even attributes respectively. While odd and even attributes have a negative correlation to one another most likely due to each phase number being represented by 2 attributes each,

Listing 4: Separate out a validation dataset. Build 5 different models to predict species from flower measurements. Select the best model.

In [13]:
```
# Split-out validation dataset
array = dataset.values
X = array[:,0:34]
Y = array[:,34]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=val
idation_size, random_state=seed)
```

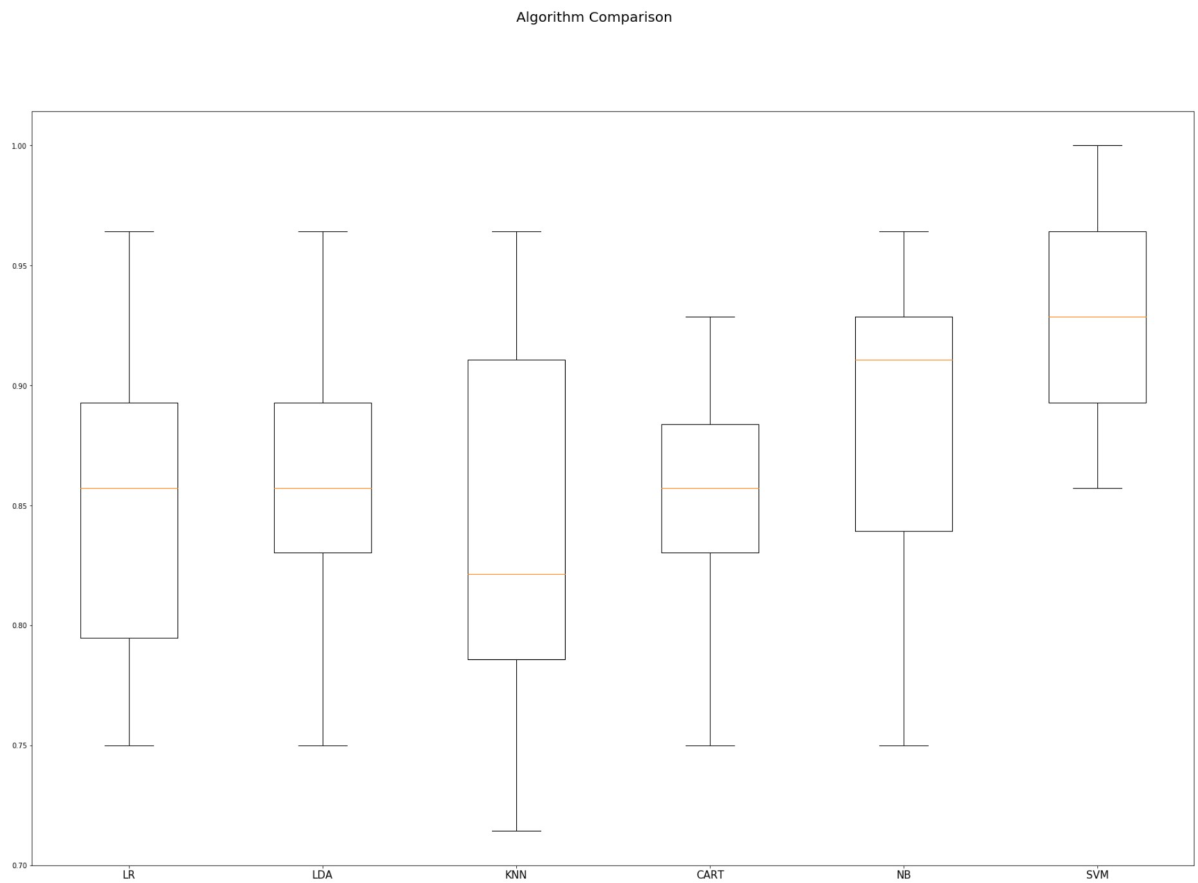
In [14]:
```
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

In [15]:
```
# evaluate each model in turn,store performance, and output general performance of
models
results = []
model_names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accura
cy')
    results.append(cv_results)
    model_names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.853571 (0.064780)
LDA: 0.867857 (0.061962)
KNN: 0.835714 (0.078571)
CART: 0.853571 (0.054046)
NB: 0.882143 (0.067857)
SVM: 0.925000 (0.046429)
```

Looking at the training results shows that most of the models performed similarly to one another with the exception of SVM which seems to be abnle to overcome some limitation of the other models with the help of learning a kernel to change the dimensionality of the data to a better representation.

In [16]:
```
# Compare Algorithms
fig = plt.figure(figsize=(30,20))
fig.suptitle('Algorithm Comparison', fontsize=20)
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(model_names, fontdict={'fontsize': 15})
plt.show()
```



The box and whisker plots further prove that SVM is the superior model for solving the classification problem. SVM had a higher median performance score, and the distibution of scores is much more concise than other box and whisker plots, and obtained a much higher performance score than other models. Many of the other models perform erratically and their performance suffers due to the large number of attributes.

Listing 5: Make Predictions on Validation Dataset

In [17]:
```
# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9014084507042254
[[17  6]
 [ 1 47]]
              precision    recall  f1-score   support

           b       0.94      0.74      0.83        23
           g       0.89      0.98      0.93        48

    accuracy                           0.90        71
   macro avg       0.92      0.86      0.88        71
weighted avg       0.91      0.90      0.90        71
```

In [18]:
```
# Make predictions on validation dataset
knn = SVC()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9436619718309859
[[20  3]
 [ 1 47]]
              precision    recall  f1-score   support

           b       0.95      0.87      0.91        23
           g       0.94      0.98      0.96        48

    accuracy                           0.94        71
   macro avg       0.95      0.92      0.93        71
weighted avg       0.94      0.94      0.94        71
```

When testing the two highest scoring models KNNs and SVM; we see that on our validation data neither the KNNs or SVM models suffer from overfitting and perform regularly; thus meaning that their performance is correctly reported by our graphs.

Overview of Results

- Looking at the box-and-whisker plots it can be seen that the odd numbered attributes are skewed toward 1.0, and the even numbered attributes are centered around 0. This is further proven by the histograms charts.
- Looking a the results for training the models that do not modify dimensionality of the data perform similarly to one another due to the lack of strong correlation or separability of the attributes. Looking at results when training the models a few conclusions can be infered from the graphs. odels that do not modify dimensionality such as LR, LDA, CART, and NB perform similarly to one another.
- While SVM, which changed the dimensionality of the data using a kernel performed significantly better on average compared to the rest of the models. KNNs was also able to perform significantly better than those linear models when classifying points.

In [ ]:

```
```