Calcific Aortic Valve Stenosis Analysis

Ryan Byrne

In [3]:
```
%load_ext autoreload
%autoreload 2
import sys,os; sys.path.append(os.environ['BMESAHMETDIR']); import bmes
bmes.pipinstall('sklearn')
bmes.pipinstall('GEOparse')
bmes.pipinstall('scipy')
bmes.pipinstall('statsmodels')
bmes.pipinstall('pathlib')
bmes.pipinstall('numpy')
bmes.pipinstall('pandas')
bmes.pipinstall('matplotlib')
bmes.pipinstall('yellowbrick')

import pandas as pd
import GEOparse
from scipy import stats
from statsmodels.stats.multitest import multipletests
from statsmodels.stats import multitest
from pathlib import Path
import numpy as np
import math
from scipy import stats
from mne.stats import bonferroni_correction, fdr_correction
from sklearn.feature_selection import RFE
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
from  matplotlib.pyplot import text
from yellowbrick.model_selection import RFECV
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Download or Open GSE Training Sets

In [4]:
```
#Training Sets
filegeo1 = "GSE12644"
filegeo2 = "GSE51472"
arr = os.listdir(bmes.tempdir())

#Check if file exists in tempdir with file name
flag1 = 1
flag2 = 1
for file in arr:
    result1 = file.find(filegeo1)
    result2 = file.find(filegeo2)
    if (result1 == 0):
        flag1 = 0
    if (result2 == 0):
        flag2 = 0

#If flag1 is 1 then the file exists in bmes.tempdir
if (flag1 == 1):
    GSE12644 = GEOparse.get_GEO(geo=filegeo1, destdir=bmes.tempdir(), silent=True
#Else download file from ncbi
else:
    GSE12644 = GEOparse.get_GEO(filepath=bmes.tempdir() + '/' + filegeo1 + '_fami

#If flag1 is 1 then the file exists in bmes.tempdir
if (flag2 == 1):
    GSE51472 = GEOparse.get_GEO(geo=filegeo2, destdir=bmes.tempdir(), silent=True
#Else download file from ncbi
else:
    GSE51472 = GEOparse.get_GEO(filepath=bmes.tempdir() + '/' + filegeo2 + '_fami
```

```
C:\Users\user\anaconda3\lib\site-packages\GEOparse\GEOparse.py:401: DtypeWarnin
g: Columns (2) have mixed types. Specify dtype option on import or set low_memo
ry=False.
  return read_csv(StringIO(data), index_col=None, sep="\t")
C:\Users\user\anaconda3\lib\site-packages\GEOparse\GEOparse.py:401: DtypeWarnin
g: Columns (2) have mixed types. Specify dtype option on import or set low_memo
ry=False.
  return read_csv(StringIO(data), index_col=None, sep="\t")
```

Combine Training Datasets

In [9]:
```
#Set variable to store first training set GSE12644
Training_Set1 = None

#List for storing gsmid
gsenames = []

#Iterate through GSE12644 and add to Training_set1 table
for gsmid in GSE12644.gsms.keys():

    #Add to list of gsmids
    gsenames.append(gsmid)

    #get data from GSM
    gsmdata = GSE12644.gsms[gsmid].table.rename(columns={'VALUE':gsmid});

    #If Training_Set1 is empty set it to be gsmdata
    if Training_Set1 is None:
        Training_Set1=gsmdata;
    #If Training_Set1 is not empty concat gsm data to Training_Set1
    else:
        Training_Set1 = pd.concat([Training_Set1,gsmdata[gsmid]],axis=1);

#Set variable to store first training set GSE51472
Training_Set2 = None

#Iterate through GSE12644 and add to Training_set1 table
for gsmid in GSE51472.gsms.keys():

    #Add to list of gsmids
    gsenames.append(gsmid)

    #get data from GSM
    gsmdata = GSE51472.gsms[gsmid].table.rename(columns={'VALUE':gsmid});

    #Remove unneeded columns
    if "ABS_CALL" in gsmdata:
        del gsmdata["ABS_CALL"]
    if "DETECTION P-VALUE" in gsmdata:
        del gsmdata["DETECTION P-VALUE"]

    #If Training_Set2 is empty set it to be gsmdata
    if Training_Set2 is None:
        Training_Set2=gsmdata;
    #If Training_Set2 is not empty concat gsm data to Training_Set2
    else:
        Training_Set2 = pd.concat([Training_Set2,gsmdata[gsmid]],axis=1);

#Dataframe to store both training sets
Combined_Training_Set = pd.DataFrame(columns = gsenames)

#Training Set 1 where all values are base gene expression
Training_Set1_Antilog = 2 ** Training_Set1.loc[:, Training_Set1.columns != 'ID_RE
Training_Set1_Antilog.insert(loc=0, column="ID_REF", value=Training_Set1['ID_REF'

#Combined Dataset that contains values that are base gene expression
Combined_Training_Set = pd.merge(Training_Set1_Antilog, Training_Set2, on="ID_REF
```

```
#Training Set 2 where all values are log 2 gene expression
Training_Set2_log2 = np.log2(Training_Set2.loc[:, Training_Set2.columns != 'ID_RE
Training_Set2_log2.insert(loc=0, column="ID_REF", value=Training_Set2['ID_REF'])

#Combined Dataset that contains values that are log base 2 gene expression
Combined_Training_Set_log2 = pd.merge(Training_Set1, Training_Set2_log2, on="ID_F
```

Download Testing Set

In [5]:
```
#Testing Set GSE geo
filegeo = "GSE83453"

#bmes.tempdir folder
arr = os.listdir(bmes.tempdir())

#Check if file exists in tempdir with file name
flag = 1
for file in arr:
    result = file.find(filegeo)
    if (result == 0):
        flag = 0

#If flag stays as 1 the file exists so open from bmes.tempdir
if (flag == 1):
    GSE83453 = GEOparse.get_GEO(geo=filegeo, destdir=bmes.tempdir(), silent=True)
#If flag is 0 it does not exists in bmes.tempdir so download
else:
    GSE83453 = GEOparse.get_GEO(filepath=bmes.tempdir() + '/' + filegeo + '_famil
```

Create Testing Set Dataframe

In [6]:
```
#Table for storing testing set
Testing_Set = None

#Iterate through GSE83453 and add to Testing_set table
for gsmid in GSE83453.gsms.keys():

    #get data from GSM
    gsmdata = GSE83453.gsms[gsmid].table.rename(columns={'VALUE':gsmid});

    #If Testing_Set is empty set it to be gsmdata
    if Testing_Set is None:
        Testing_Set=gsmdata;
    #If Training_Set is not empty concat gsm data to Training_Set
    else:
        Testing_Set = pd.concat([Testing_Set,gsmdata[gsmid]],axis=1);

Testing_Set.head()
```

Out[6]:

	ID_REF	GSM2203485	GSM2203486	GSM2203487	GSM2203488	GSM2203489	GSM22034
0	ILMN_1802380	10.209276	10.444209	10.483140	10.321388	10.038555	10.50505
1	ILMN_1893287	6.947164	6.984719	6.946075	6.933777	7.079490	7.01464
2	ILMN_3238331	6.936622	6.869156	7.108296	6.931772	6.730780	6.81546
3	ILMN_1736104	6.887272	6.994858	6.898660	6.979273	6.946823	7.03633
4	ILMN_1792389	7.120412	7.059341	6.925171	7.220840	7.263099	7.22544

5 rows × 28 columns

Convert from Illumina Reads to Gene Symbol

In [7]:
```
#Conversion of Illumina Reads to gene symbol done in R

#testing file with Illumina Reads converted to gene symbol
file = f"C:\\Users\\user\\Dropbox\\bmes543projects.20213\\AorticValveStenosis.Pra

#Open file
TestingSet_Symbol_file = pd.read_excel(file)

#Remove Illumina Reads that can not be converted
TestingSet_Symbol = TestingSet_Symbol_file.dropna()
#Remove ID_REF from the dataframe
Testing_Set = TestingSet_Symbol.loc[:, TestingSet_Symbol.columns != 'ID_REF']
```

Find DEGs (Differentially Expressed Genes).

In [10]:
```
#control group expression
control_group_exp = Combined_Training_Set[["GSM317342", "GSM317343", "GSM317344",

#convert control_group_exp from dataframe to array
control_group_exp = control_group_exp.to_numpy()

#CAVs group expression
CAVS_group_exp = Combined_Training_Set[["GSM317347","GSM317348", "GSM317349", "GS

#convert CAVS_group_exp from dataframe to array
CAVS_group_exp = CAVS_group_exp.to_numpy()

#control group log 2 expression
control_group_log2 = Combined_Training_Set_log2[["GSM317342", "GSM317343", "GSM31

#convert control_group_exp from dataframe to array
control_group_log2 = control_group_log2.to_numpy()

#CAVs group log 2 expression
CAVS_group_log2 = Combined_Training_Set_log2[["GSM317347","GSM317348", "GSM317349

#convert CAVS_group_log2 from dataframe to array
CAVS_group_log2 = CAVS_group_log2.to_numpy()

#Calcuate the p-value between control and CAVS groups
tresult = stats.ttest_ind(control_group_exp.transpose(), CAVS_group_exp.transpose

#index p-values as list
pvals = list(tresult.pvalue)

#Find bonferroni correction of pvalues - all fail the bonferroni test so use base
reject_bonferroni, pval_bonferroni = bonferroni_correction(pvals, alpha=0.05)

#log fold change is the difference in means on the log scale
log2fc = CAVS_group_log2.mean(axis=1) - control_group_log2.mean(axis=1);

#Dataframe for storing differential expression
different_genes = pd.DataFrame()

#Insert gene ID
different_genes.insert(loc=0,
          column='ID_REF',
          value=Combined_Training_Set["ID_REF"])

#Insert Fold change
different_genes.insert(loc=1,
          column='fold change',
          value=log2fc)

#Insert p-value
different_genes.insert(loc=2,
          column='p-value',
          value=pvals)

#Filter genes that have a p-value less than 0.01 and a log2 fold change of 1.5
most_different_genes = different_genes.loc[(different_genes['p-value'] <0.05) & (
```

```
#Report the top 10 different genes
print(most_different_genes.sort_values(by=['p-value'])[0:10])

#export to excel
most_different_genes.to_excel("DEGs.xlsx", index=False)
```

```
             ID_REF  fold change   p-value
25673     238378_at     1.138471  0.000099
20402  1568574_x_at     1.045688  0.000278
1427      223503_at     1.181758  0.000413
27197     219890_at     1.647355  0.000463
14065   220161_s_at    -1.234748  0.000505
38682     222953_at    -1.407767  0.000925
51999     205624_at     1.607493  0.001084
11115   216920_s_at     1.143176  0.001114
44501   207610_s_at     1.275867  0.001220
778       242943_at     1.016538  0.001272
```

DEG Pathway Enrichment

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjamini
	GOTERM_BP_DIRECT	immune response	RT		33	14.7	4.3E-17	5.5E-14
	GOTERM_BP_DIRECT	inflammatory response	RT		24	10.7	4.2E-11	2.0E-8
	GOTERM_BP_DIRECT	neutrophil chemotaxis	RT		13	5.8	4.5E-11	2.0E-8
	GOTERM_BP_DIRECT	adaptive immune response	RT		21	9.4	2.7E-8	8.7E-6
	GOTERM_BP_DIRECT	chemokine-mediated signaling pathway	RT		9	4.0	6.7E-7	1.7E-4
	GOTERM_BP_DIRECT	response to bacterium	RT		10	4.5	2.3E-6	4.6E-4
	GOTERM_BP_DIRECT	ossification	RT		9	4.0	2.5E-6	4.6E-4
	GOTERM_BP_DIRECT	defense response to bacterium	RT		13	5.8	5.4E-6	8.8E-4
	GOTERM_BP_DIRECT	extracellular matrix organization	RT		11	4.9	6.4E-6	9.3E-4
	GOTERM_BP_DIRECT	killing of cells of other organism	RT		8	3.6	8.4E-6	1.1E-3
	GOTERM_BP_DIRECT	cell surface receptor signaling pathway	RT		14	6.2	1.8E-5	2.0E-3
	GOTERM_BP_DIRECT	antimicrobial humoral immune response mediated by antimicrobial peptide	RT		9	4.0	1.9E-5	2.0E-3
	GOTERM_BP_DIRECT	chemotaxis	RT		9	4.0	4.5E-5	4.5E-3
	GOTERM_BP_DIRECT	natural killer cell activation	RT		5	2.2	1.4E-4	1.3E-2
	GOTERM_BP_DIRECT	cell-cell signaling	RT		11	4.9	1.5E-4	1.3E-2
	GOTERM_BP_DIRECT	cellular response to interleukin-1	RT		7	3.1	3.4E-4	2.8E-2
	GOTERM_BP_DIRECT	negative regulation of calcium ion transport	RT		4	1.8	3.7E-4	2.9E-2
	GOTERM_BP_DIRECT	immunoglobulin production	RT		7	3.1	4.8E-4	3.5E-2
	GOTERM_BP_DIRECT	positive regulation of ERK1 and ERK2 cascade	RT		10	4.5	6.3E-4	4.3E-2
	GOTERM_BP_DIRECT	proteolysis	RT		14	6.2	7.6E-4	4.9E-2
	GOTERM_BP_DIRECT	cellular response to lipopolysaccharide	RT		9	4.0	8.5E-4	5.3E-2
	GOTERM_BP_DIRECT	monocyte chemotaxis	RT		5	2.2	1.2E-3	7.2E-2
	GOTERM_BP_DIRECT	cytolysis	RT		4	1.8	1.3E-3	7.2E-2
	GOTERM_BP_DIRECT	extracellular matrix disassembly	RT		5	2.2	1.3E-3	7.2E-2
	GOTERM_BP_DIRECT	positive regulation of interferon-gamma production	RT		6	2.7	1.4E-3	7.4E-2
	GOTERM_BP_DIRECT	positive regulation of cell proliferation	RT		15	6.7	1.5E-3	7.4E-2
	GOTERM_BP_DIRECT	positive regulation of angiogenesis	RT		8	3.6	1.5E-3	7.4E-2
	GOTERM_BP_DIRECT	neuron projection development	RT		7	3.1	1.9E-3	8.5E-2

Find DEIRG

In [11]:
```
#Open immune gene list from immport.org
file = f"C:\\Users\\user\\Dropbox\\bmes543projects.20213\\AorticValveStenosis.Pra
geneList_Table = pd.read_table(file,sep="\t")

#Put IRGs into list
geneList = geneList_Table["Symbol"].tolist()

#File containing the converted Gene List
file = f"C:\\Users\\user\\Dropbox\\bmes543projects.20213\\AorticValveStenosis.Pra
convert_Table = pd.read_table(file,sep="\t")

#variable to store gene symbols
matched = []

#Iterate through genes in dataframe and get the gene symbol convertion if it exis
for i in range(len(most_different_genes)):

    #Indexed row
    row = most_different_genes.iloc[i]

    #gene id
    geneName = row["ID_REF"]

    #Gene symbol of gene id
    geneSymbol = convert_Table["To"].loc[convert_Table["From"] == geneName.upper(

    #If does not exists add empty characters to list
    if geneSymbol.empty:
        matched.append("")
    #If exists add to list
    else:
        matched.append(geneSymbol.iloc[0])

#Add column to data frame containing gene symbol
most_different_genes.insert(loc=1,
          column='Gene_Symbol',
          value=matched)

#DEIRGs are genes remaining that exist in IRG gene list
DEIRG = most_different_genes.query('Gene_Symbol in @geneList')

#Read DEIRGs into excel file
DEIRG.to_excel("DEIRG.xlsx", index=False)
```

DEIRG Pathway Enrichment

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjam
☐	GOTERM_BP_DIRECT	immune response	RT		23	37.1	8.3E-21	5.5E-1
☐	GOTERM_BP_DIRECT	inflammatory response	RT		21	33.9	1.5E-19	5.1E-1
☐	GOTERM_BP_DIRECT	neutrophil chemotaxis	RT		12	19.4	7.3E-16	1.6E-1
☐	GOTERM_BP_DIRECT	chemokine-mediated signaling pathway	RT		9	14.5	3.5E-11	5.8E-9
☐	GOTERM_BP_DIRECT	antimicrobial humoral immune response mediated by antimicrobial peptide	RT		9	14.5	1.3E-9	1.7E-7
☐	GOTERM_BP_DIRECT	killing of cells of other organism	RT		8	12.9	1.7E-9	1.8E-7
☐	GOTERM_BP_DIRECT	chemotaxis	RT		9	14.5	3.4E-9	3.2E-7
☐	GOTERM_BP_DIRECT	adaptive immune response	RT		12	19.4	5.7E-8	4.7E-6
☐	GOTERM_BP_DIRECT	immunoglobulin production	RT		7	11.3	4.2E-7	2.8E-5
☐	GOTERM_BP_DIRECT	defense response to bacterium	RT		9	14.5	4.2E-7	2.8E-5
☐	GOTERM_BP_DIRECT	cell-cell signaling	RT		8	12.9	5.7E-6	3.4E-4
☐	GOTERM_BP_DIRECT	monocyte chemotaxis	RT		5	8.1	1.0E-5	5.7E-4
☐	GOTERM_BP_DIRECT	signal transduction	RT		15	24.2	1.1E-5	5.7E-4
☐	GOTERM_BP_DIRECT	lymphocyte chemotaxis	RT		4	6.5	1.5E-4	6.8E-3
☐	GOTERM_BP_DIRECT	cellular response to interleukin-1	RT		5	8.1	1.5E-4	6.8E-3
☐	GOTERM_BP_DIRECT	defense response to Gram-negative bacterium	RT		5	8.1	1.8E-4	7.5E-3
☐	GOTERM_BP_DIRECT	cellular response to interferon-gamma	RT		5	8.1	2.3E-4	9.0E-3
☐	GOTERM_BP_DIRECT	positive regulation of GTPase activity	RT		6	9.7	2.5E-4	9.1E-3
☐	GOTERM_BP_DIRECT	cellular response to lipopolysaccharide	RT		6	9.7	2.7E-4	9.2E-3
☐	GOTERM_BP_DIRECT	embryo implantation	RT		4	6.5	2.8E-4	9.2E-3
☐	GOTERM_BP_DIRECT	cell surface receptor signaling pathway	RT		7	11.3	2.9E-4	9.2E-3
☐	GOTERM_BP_DIRECT	chronic inflammatory response	RT		3	4.8	4.0E-4	1.2E-2
☐	GOTERM_BP_DIRECT	defense response to Gram-positive bacterium	RT		5	8.1	4.1E-4	1.2E-2
☐	GOTERM_BP_DIRECT	positive regulation of ERK1 and ERK2 cascade	RT		6	9.7	6.0E-4	1.6E-2
☐	GOTERM_BP_DIRECT	cellular response to virus	RT		4	6.5	6.2E-4	1.6E-2

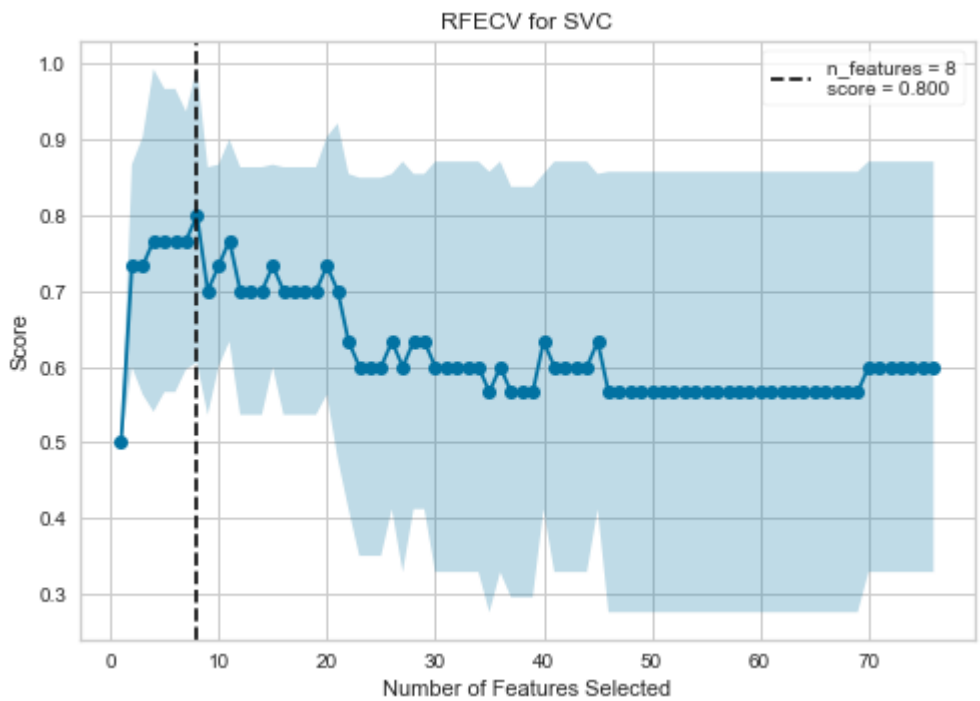
Transpose Training and Testing sets

In [11]:
```
#Control groups from Training Set
control_group = Combined_Training_Set_log2[["ID_REF", "GSM317342", "GSM317343", '

#CAVS groups from Training Set
CAVS_group = Combined_Training_Set_log2[["ID_REF", "GSM317347","GSM317348", "GSM3

#List of gene ids of DEIRGs
DEIRG_List_ID_REF = DEIRG["ID_REF"].tolist()

#List of gene symbols of DEIRGs
DEIRG_List_Symbol = DEIRG["Gene_Symbol"].tolist()

#Index genes that are in gene id list
DEIRG_Exp_Raw_control_Training = control_group.query('ID_REF in @DEIRG_List_ID_RE

#Add gene symbol to dataframe
DEIRG_Exp_Raw_control_Training.insert (1, "Gene_Symbol", DEIRG_List_Symbol)

#remove gene id
DEIRG_Exp_Raw_control_Training = DEIRG_Exp_Raw_control_Training.loc[:, DEIRG_Exp_

#Index genes that are in gene id list
DEIRG_Exp_Raw_CAVS_Training = CAVS_group.query('ID_REF in @DEIRG_List_ID_REF')

#Add gene symbol to dataframe
DEIRG_Exp_Raw_CAVS_Training.insert (1, "Gene_Symbol", DEIRG_List_Symbol)

#remove gene id
DEIRG_Exp_Raw_CAVS_Training = DEIRG_Exp_Raw_CAVS_Training.loc[:, DEIRG_Exp_Raw_CA

#Dataframe with headers being the DEIRGs
DEIRG_Exp_control_Training = pd.DataFrame(columns = DEIRG_List_Symbol)
DEIRG_Exp_CAVS_Training = pd.DataFrame(columns = DEIRG_List_Symbol)

#Iterate through columns in dataframe and add to DEIRG_Exp_control_Training and D
for column in DEIRG_Exp_control_Training.columns:
    expression_control = DEIRG_Exp_Raw_control_Training.loc[DEIRG_Exp_Raw_control
    DEIRG_Exp_control_Training[column] = expression_control.values[0][:]

    expression_CAVS = DEIRG_Exp_Raw_CAVS_Training.loc[DEIRG_Exp_Raw_CAVS_Training
    DEIRG_Exp_CAVS_Training[column] = expression_CAVS.values[0][:]

#concatenate them dataframes on top of eachother thus top is control, bottom is C
Feature_Vector_Training_df_all = pd.concat([DEIRG_Exp_control_Training, DEIRG_Exp

Feature_Vector_Training_df = Feature_Vector_Training_df_all.loc[:,~Feature_Vector

#Control group in testing set
Control_group = Testing_Set[["Gene_Symbol", "GSM2203504", "GSM2203505", "GSM22035

#CAVS group in testing set
CAVS_group = Testing_Set[["Gene_Symbol", "GSM2203495","GSM2203496", "GSM2203497",

#Dataframe with headers being the DEIRGs
DEIRG_Exp_control_Testing = pd.DataFrame(columns = DEIRG_List_Symbol)
DEIRG_Exp_CAVS_Testing = pd.DataFrame(columns = DEIRG_List_Symbol)
```

```
#Index genes that are in gene symbol list
DEIRG_Exp_Raw_control_Testing = Control_group.query('Gene_Symbol in @DEIRG_List_S

#Index genes that are in gene symbol list
DEIRG_Exp_Raw_CAVS_Testing = CAVS_group.query('Gene_Symbol in @DEIRG_List_Symbol'

#Iterate through columns in dataframe and add to DEIRG_Exp_control_Training and D
#Some genes are not in the testing set so they are ignored
for column in DEIRG_Exp_control_Testing.columns:
    expression_control = DEIRG_Exp_Raw_control_Testing.loc[DEIRG_Exp_Raw_control_
    if len(expression_control) != 0:
        DEIRG_Exp_control_Testing[column] = expression_control.values[0][:]

    expression_CAVS = DEIRG_Exp_Raw_CAVS_Testing.loc[DEIRG_Exp_Raw_CAVS_Testing['
    if len(expression_CAVS) != 0:
        DEIRG_Exp_CAVS_Testing[column] = expression_CAVS.values[0][:]

#concatenate them dataframes on top of eachother thus top is control, bottom is C
Feature_Vector_Testing_df = pd.concat([DEIRG_Exp_control_Testing, DEIRG_Exp_CAVS_

#drop columns that were unmatched
Feature_Vector_Testing_df = Feature_Vector_Testing_df.dropna(axis = 1)

#Remove duplicate genes
Feature_Vector_Testing_df = Feature_Vector_Testing_df.loc[:,~Feature_Vector_Testi

#columns in both testing and training dataframes
columnsTrain = list(Feature_Vector_Training_df.columns)
columnsTest = list(Feature_Vector_Testing_df.columns)
indexes = []

#indexs of genes that were unmatched in testing file
for i in range(len(columnsTrain)):
    if (columnsTrain[i] in columnsTest):
        indexes.append(i)

#Sepertate training set that contains matching genes in the tesin
Feature_Vector_Training_df = Feature_Vector_Training_df.iloc[:, indexes]
```

Find Optimal Number of Features

In [12]:
```
#Class values for svm model 0 is healthy 1 is CAVs
#Training classes
Health_Training = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
#Testing classes
Health_Testing = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

#linear svm model
clf = svm.SVC(kernel='linear')

#5 fold cross fold
cv = StratifiedKFold(5)

#Perform cross validated recursive feature elimination
visualizer = RFECV(clf, cv=cv)

#fit RFECV
visualizer.fit(Feature_Vector_Training_df_all, Health_Training)

#Show RFECV
visualizer.show()
```

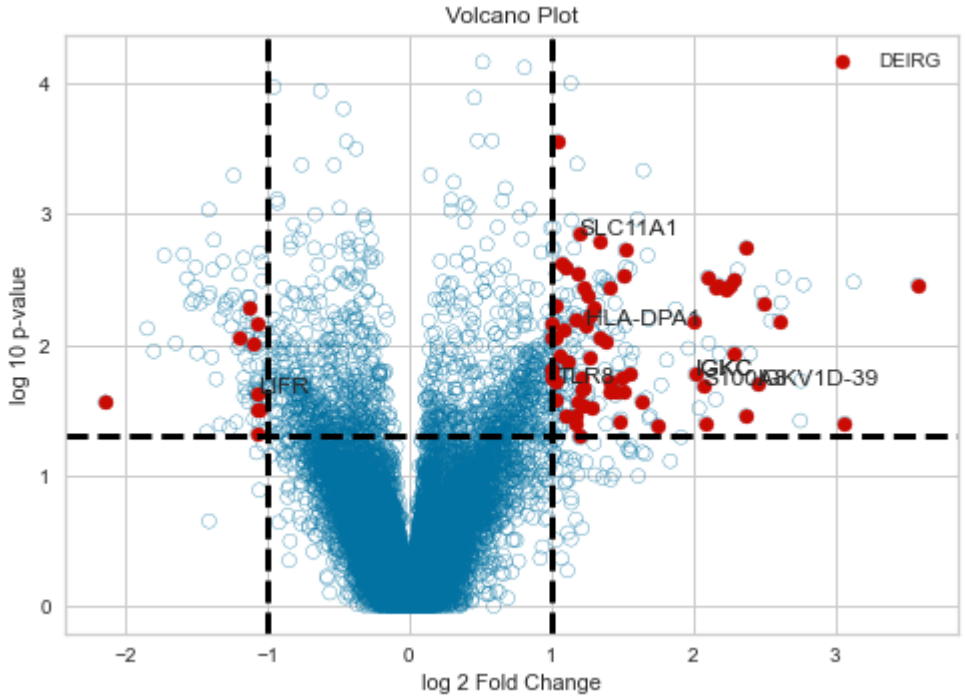


Out[12]:

```
<AxesSubplot:title={'center':'RFECV for SVC'}, xlabel='Number of Features Sel
ected', ylabel='Score'>
```

Volcano Plot With Selected Features

In [15]:
```
#Create scatter plot of all differential expression
plt.scatter(different_genes['fold change'], -np.log10(different_genes['p-value']))

#x axis label
plt.xlabel("log 2 Fold Change")

#y axis label
plt.ylabel("log 10 p-value")

#add plot title
plt.title("Volcano Plot")

#Add scatter plot for DEIRG
DEIRG_Scat = plt.scatter(DEIRG['fold change'], -np.log10(DEIRG['p-value']), facec

#Adding lines to define boundries
plt.axvline(x=-1, color="black", linestyle="--", linewidth=3.0)
plt.axvline(x=1, color="black", linestyle="--", linewidth=3.0)
plt.axhline(y=-math.log10(0.05), color="black", linestyle="--", linewidth=3.0)

#Add label to list
DEIRG_Scat.set_label('DEIRG')

#Add legend
plt.legend()

#The top 8 genes as found by RFE
top_8_ind = visualizer.get_support(indices=True)

#names of genes that are selected
top_8 = list(Feature_Vector_Training_df_all.iloc[:,top_8_ind].columns)

#Add label to all genes selected
for gene in top_8:

    #Genes expression value
    expression = DEIRG.loc[DEIRG['Gene_Symbol'] == gene, 'fold change']

    #Gene p-value
    pval = DEIRG.loc[DEIRG['Gene_Symbol'] == gene, 'p-value']

    #Label gene with name
    text(expression.values[0], -np.log10(pval.values[0]), gene, fontsize=12)

#Show plot
plt.show()
```



Build and Test SVM-RFE Model

In [16]:
```
# file for storing accuracies
avg_acc = []

#list of number of features to select from 1 to 50
numFeat_to_select = list(range(1,50+1))

#list of selected Features
selected_Features = {}

for num in numFeat_to_select:

    #accuracy of iteration
    avg_acc_iter = []
    for i in range(1):

        #linear svm model
        clf = svm.SVC(kernel='linear')

        #RFE model
        RFESelector = RFE(estimator=clf, n_features_to_select=num, step=1)

        #model pipeline
        pipe = Pipeline([('rfe', RFESelector), ('svc', clf)])

        #fit pipeline
        pipe.fit(Feature_Vector_Training_df, Health_Training)

        #Index of Features selected
        column = RFESelector.get_support(indices=True)

        #Features selected
        Selected_Features_SVC = list(Feature_Vector_Training_df.iloc[:,column].co

        #number of matches during testing
        Num_Correct = 0

        #Iterate through testing subjects
        for j in range(len(Health_Testing)):

            #row of subject being tested
            row = Feature_Vector_Testing_df.iloc[j,:]

            #Feature vector for predicting
            FeatureVec = pd.DataFrame(columns = Feature_Vector_Training_df.colum

            #add row to dataframe
            FeatureVec.loc[len(FeatureVec.index)]  = row

            #predict using featureVec dataframe
            result = pipe.predict(FeatureVec)

            #if result matches subjects group
            if (result[0] == Health_Testing[j]):
                Num_Correct += 1

        #accuracy of model
```

```
        accuracy = Num_Correct / len(Health_Testing)

        #Add cross fold accuracy to list of accuracies for numfeature iteration
        avg_acc_iter.append(accuracy)

    #add selected features to dictionary
    selected_Features[num] = Selected_Features_SVC

    #mean of iterations
    iter_avg = sum(avg_acc_iter) / len(avg_acc_iter)

    #add accuracy
    avg_acc.append(iter_avg)

print(selected_Features[5])
```

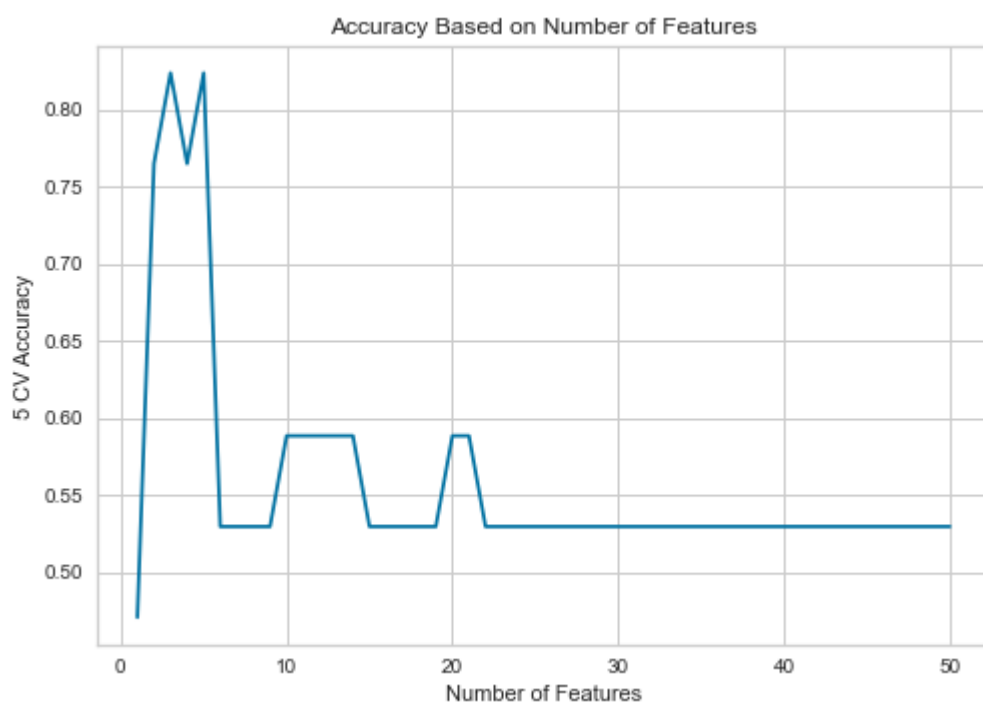
```
['LCK', 'TLR8', 'LIFR', 'ADIPOQ', 'S100A8']
```

Type *Markdown* and LaTeX: α^2

Results of training

Type *Markdown* and LaTeX: α^2

In [17]:
```
#plot accuracies for different number of selected features
plt.plot(numFeat_to_select, avg_acc)
plt.xlabel('Number of Features')
plt.ylabel('5 CV Accuracy')
plt.title('Accuracy Based on Number of Features')
plt.show()
```



In [ ]:

```

```