20104016

DEENA

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:
```
df=pd.read_csv("innovation_and_development_database.csv")
df
```

Out[2]:

	country	code	year	eap	eca	lac	mena	sha	sa	hi	...	y	stockpatEPO
0	Aruba	ABW	1960	0	0	1	0	0	0	0.0	...	NaN	NaN
1	Aruba	ABW	1961	0	0	1	0	0	0	0.0	...	NaN	NaN
2	Aruba	ABW	1962	0	0	1	0	0	0	0.0	...	NaN	NaN
3	Aruba	ABW	1963	0	0	1	0	0	0	0.0	...	NaN	NaN
4	Aruba	ABW	1964	0	0	1	0	0	0	0.0	...	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8290	Zimbabwe	ZWE	1998	0	0	0	0	1	0	0.0	...	8.290000e+09	8.0
8291	Zimbabwe	ZWE	1999	0	0	0	0	1	0	0.0	...	8.230000e+09	8.0
8292	Zimbabwe	ZWE	2000	0	0	0	0	1	0	0.0	...	7.830000e+09	8.0
8293	Zimbabwe	ZWE	2001	0	0	0	0	1	0	0.0	...	NaN	8.0
8294	Zimbabwe	ZWE	2002	0	0	0	0	1	0	0.0	...	NaN	NaN

8295 rows × 33 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.fillna(1)
df
```

Out[3]:

	country	code	year	eap	eca	lac	mena	sha	sa	hi	...	y	stockpatEPO
0	Aruba	ABW	1960	0	0	1	0	0	0	0.0	...	1.000000e+00	1.0
1	Aruba	ABW	1961	0	0	1	0	0	0	0.0	...	1.000000e+00	1.0
2	Aruba	ABW	1962	0	0	1	0	0	0	0.0	...	1.000000e+00	1.0
3	Aruba	ABW	1963	0	0	1	0	0	0	0.0	...	1.000000e+00	1.0
4	Aruba	ABW	1964	0	0	1	0	0	0	0.0	...	1.000000e+00	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8290	Zimbabwe	ZWE	1998	0	0	0	0	1	0	0.0	...	8.290000e+09	8.0
8291	Zimbabwe	ZWE	1999	0	0	0	0	1	0	0.0	...	8.230000e+09	8.0
8292	Zimbabwe	ZWE	2000	0	0	0	0	1	0	0.0	...	7.830000e+09	8.0
8293	Zimbabwe	ZWE	2001	0	0	0	0	1	0	0.0	...	1.000000e+00	8.0
8294	Zimbabwe	ZWE	2002	0	0	0	0	1	0	0.0	...	1.000000e+00	1.0

In [4]:
```
df.columns
```

Out[4]:
```
Index(['country', 'code', 'year', 'eap', 'eca', 'lac', 'mena', 'sha', 'sa',
       'hi', 'pat', 'patepo', 'royal', 'rdexp', 'rdper', 'rdfinabro',
       'rdfinprod', 'rdperfprod', 'rdperfhe', 'rdperfpub', 'lowrdexp',
       'lowrdfinprod', 'lowrdperfprod', 'y', 'stockpatEPO', 'poptotal',
       'labor', 'rdexpgdp', 'patgrantedstock', 'plantpatstock',
       'designpatstock', 'plantpat', 'designpat'],
      dtype='object')
```

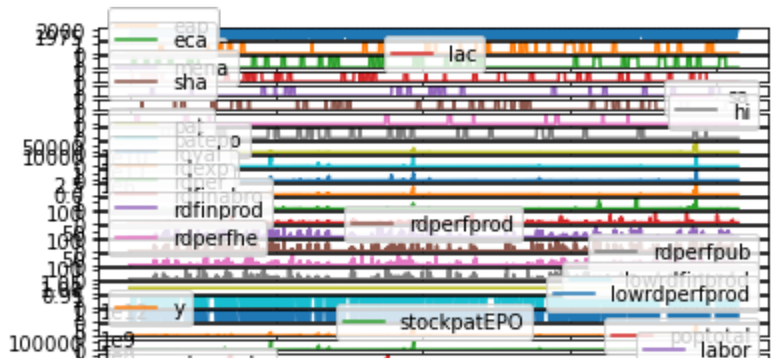
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8295 entries, 0 to 8294
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   country          8295 non-null   object 
 1   code             8295 non-null   object 
 2   year             8295 non-null   int64  
 3   eap              8295 non-null   int64  
 4   eca              8295 non-null   int64  
 5   lac              8295 non-null   int64  
 6   mena             8295 non-null   int64  
 7   sha              8295 non-null   int64  
 8   sa               8295 non-null   int64  
 9   hi               8295 non-null   float64
 10  pat              8295 non-null   float64
 11  patepo           8295 non-null   float64
 12  royal            8295 non-null   float64
 13  rdexp            8295 non-null   float64
 14  rdper            8295 non-null   float64
 15  rdfinabro        8295 non-null   float64
 16  rdfinprod        8295 non-null   float64
 17  rdperfprod       8295 non-null   float64
 18  rdperfhe         8295 non-null   float64
 19  rdperfpub        8295 non-null   float64
 20  lowrdexp         8295 non-null   float64
 21  lowrdfinprod     8295 non-null   float64
 22  lowrdperfprod    8295 non-null   float64
 23  y                8295 non-null   float64
 24  stockpatEPO      8295 non-null   float64
 25  poptotal         8295 non-null   float64
 26  labor            8295 non-null   float64
 27  rdexpgdp         8295 non-null   float64
 28  patgrantedstock  8295 non-null   float64
 29  plantpatstock    8295 non-null   float64
 30  designpatstock   8295 non-null   float64
 31  plantpat         8295 non-null   float64
 32  designpat        8295 non-null   float64
dtypes: float64(24), int64(7), object(2)
memory usage: 2.1+ MB
```

Line chart

In [6]:
```
df.plot.line(subplots=True)
```

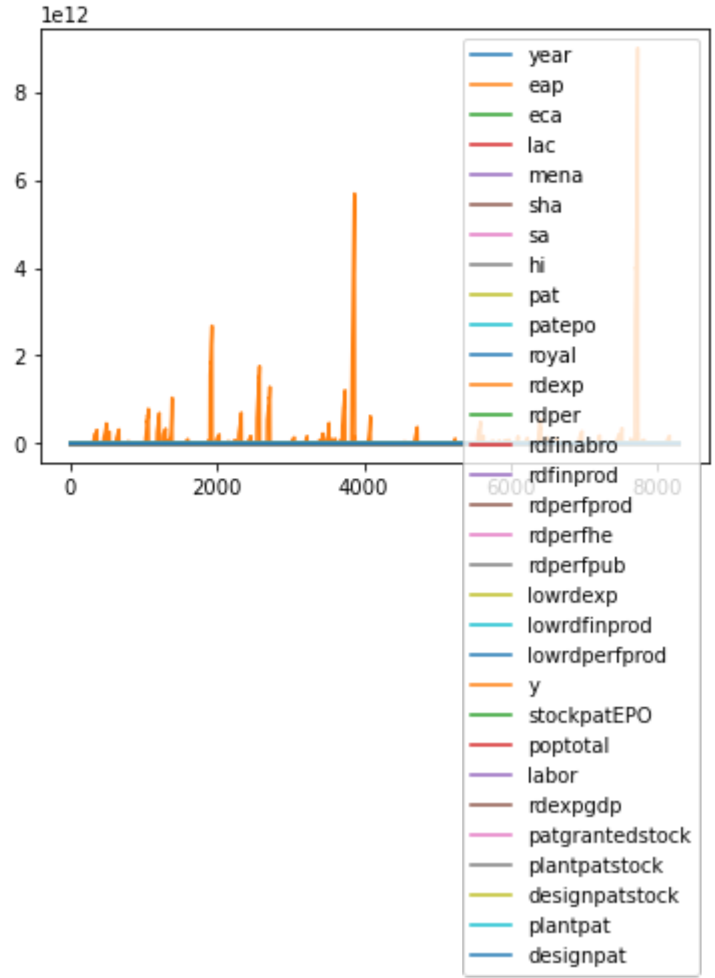
Out[6]:
```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [7]:
```
df.plot.line()
```

Out[7]: <AxesSubplot:>

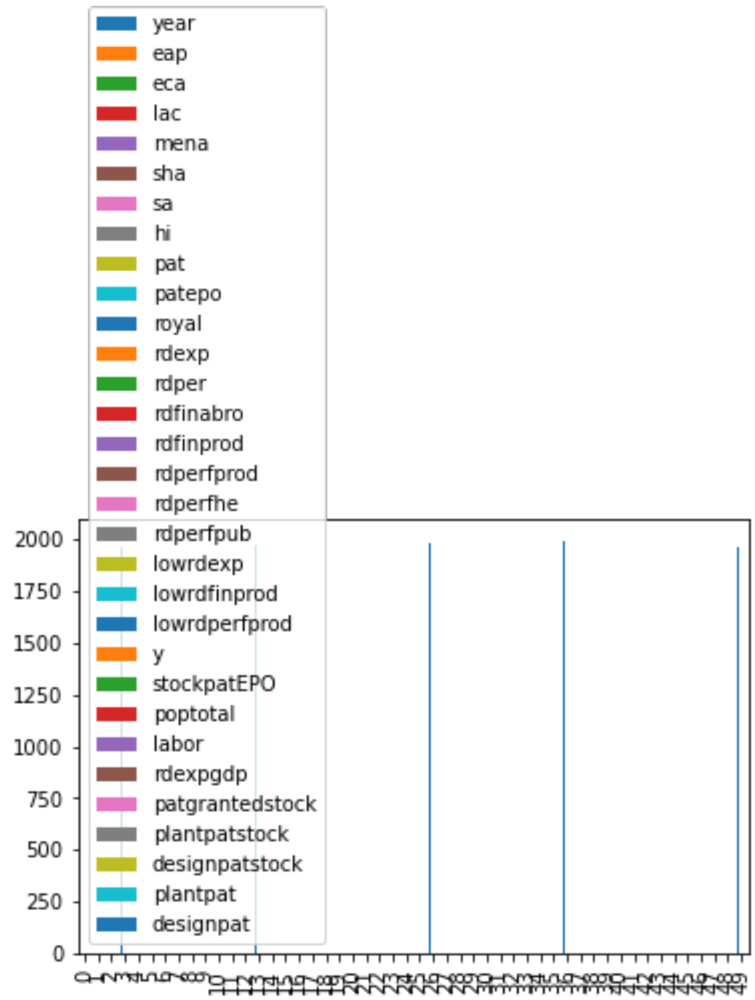


Bar chart

In [8]:
```
b=df[0:50]
```

In [9]:
```
b.plot.bar()
```

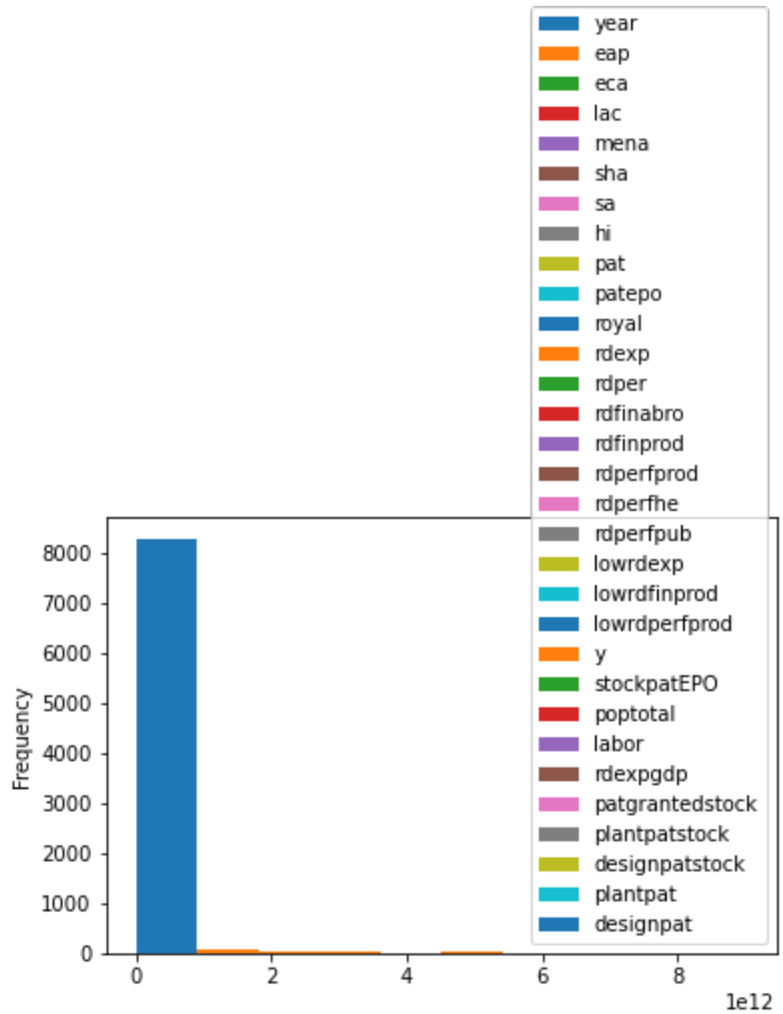
Out[9]: <AxesSubplot:>



Histogram

In [10]:
```
df.plot.hist()
```

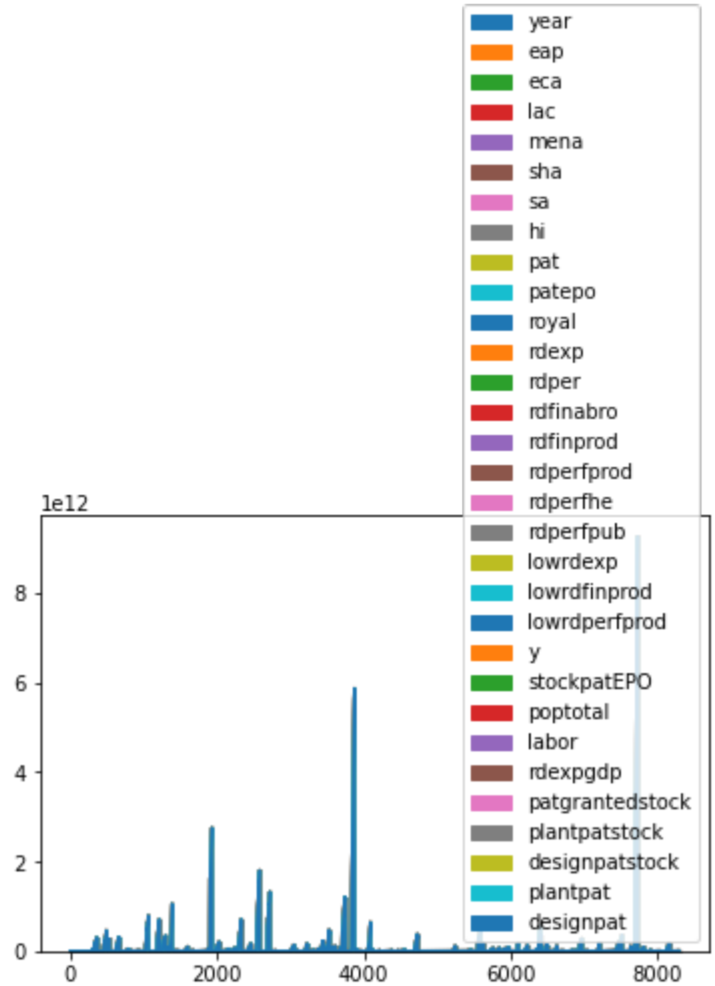
Out[10]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [11]:
```
df.plot.area()
```

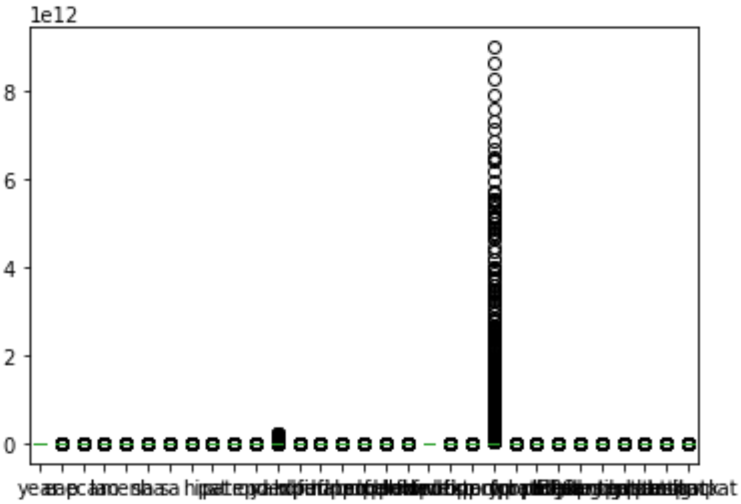
Out[11]: <AxesSubplot:>



Box chart

In [12]:
```
df.plot.box()
```

Out[12]: <AxesSubplot:>



Pie chart

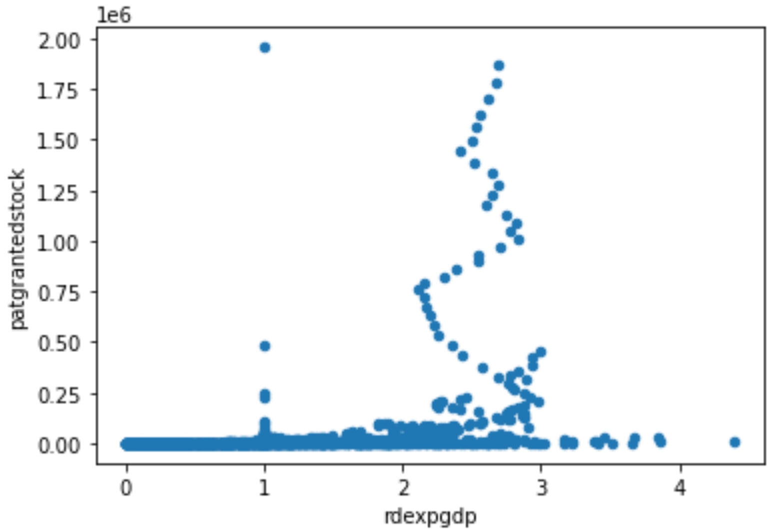
In [13]:
```
b.plot.pie(y='designpat',)
```

Out[13]: <AxesSubplot:ylabel='designpat'>

Scatter chart

In [14]:
```
df.plot.scatter( x='rdexpgdp', y= 'patgrantedstock')
```

Out[14]:
```
<AxesSubplot:xlabel='rdexpgdp', ylabel='patgrantedstock'>
```



In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8295 entries, 0 to 8294
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   country          8295 non-null   object 
 1   code             8295 non-null   object 
 2   year             8295 non-null   int64  
 3   eap              8295 non-null   int64  
 4   eca              8295 non-null   int64  
 5   lac              8295 non-null   int64  
 6   mena             8295 non-null   int64  
 7   sha              8295 non-null   int64  
 8   sa               8295 non-null   int64  
 9   hi               8295 non-null   float64
 10  pat              8295 non-null   float64
 11  patepo           8295 non-null   float64
 12  royal            8295 non-null   float64
 13  rdexp            8295 non-null   float64
```

In [16]:
```
df.describe()
```

Out[16]:

	year	eap	eca	lac	mena	sha	
count	8295.000000	8295.000000	8295.000000	8295.000000	8295.000000	8295.000000	8295.0000
mean	1981.203014	0.094515	0.195901	0.176974	0.098614	0.150090	0.0265
std	12.421590	0.292561	0.396917	0.381670	0.298161	0.357182	0.1606
min	1960.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	1970.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
50%	1981.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
75%	1992.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
max	2002.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

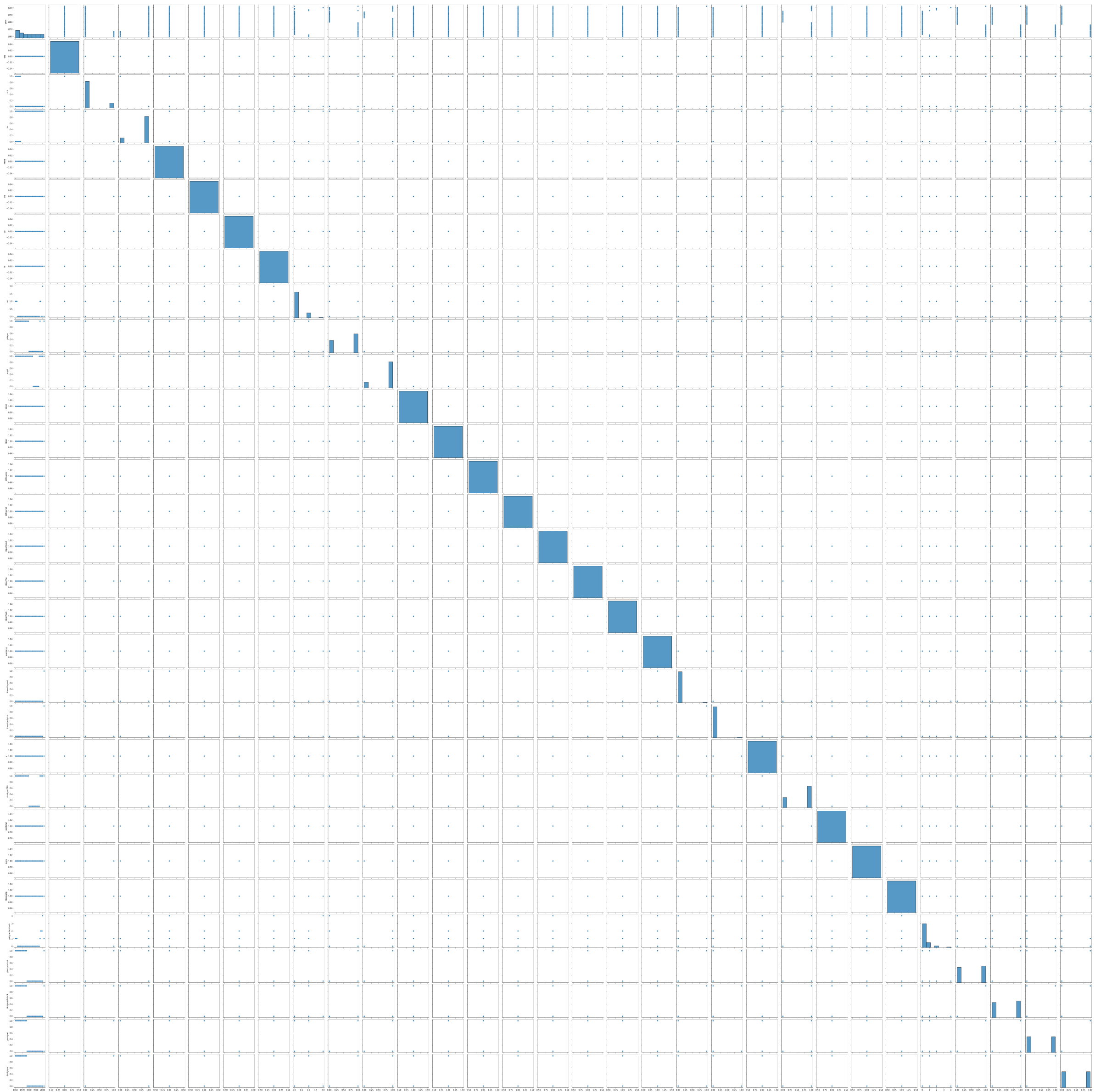
8 rows × 31 columns

In [17]:
```
df1=df[['year', 'eap', 'eca', 'lac', 'mena', 'sha', 'sa',
       'hi', 'pat', 'patepo', 'royal', 'rdexp', 'rdper', 'rdfinabro',
       'rdfinprod', 'rdperfprod', 'rdperfhe', 'rdperfpub', 'lowrdexp',
       'lowrdfinprod', 'lowrdperfprod', 'y', 'stockpatEPO', 'poptotal',
       'labor', 'rdexpgdp', 'patgrantedstock', 'plantpatstock',
       'designpatstock', 'plantpat', 'designpat']]
```

EDA AND VISUALIZATION

In [18]:
```
sns.pairplot(df1[0:50])
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x16183092580>



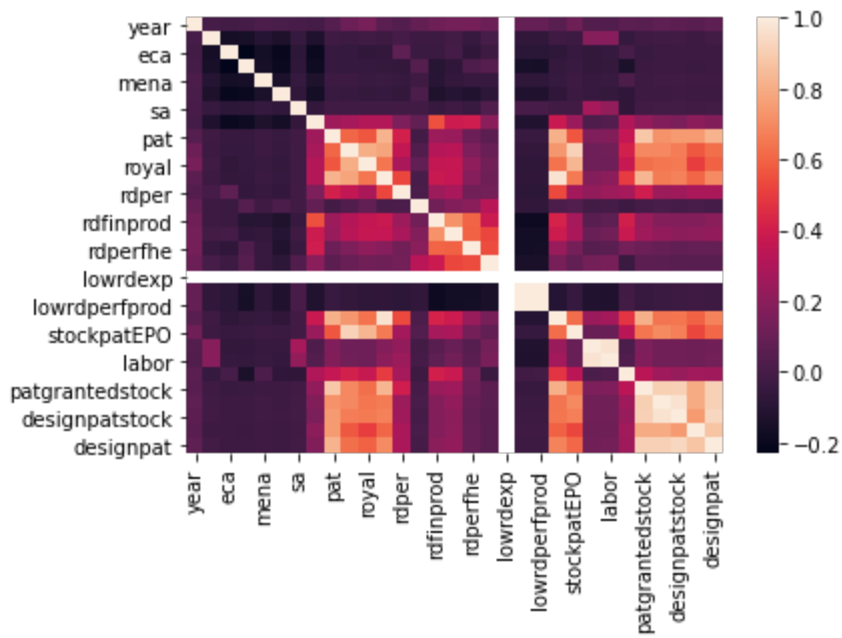
In [19]:
```
sns.distplot(df1['designpat'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]: <AxesSubplot:xlabel='designpat', ylabel='Density'>

In [20]:
```
sns.heatmap(df1.corr())
```

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
x=df[['year', 'eap', 'eca', 'lac', 'mena', 'sha', 'sa',
       'hi', 'pat', 'patepo', 'royal', 'rdexp', 'rdper', 'rdfinabro',
       'rdfinprod', 'rdperfprod', 'rdperfhe', 'rdperfpub', 'lowrdexp',
       'lowrdfinprod', 'lowrdperfprod', 'y', 'stockpatEPO', 'poptotal',
       'labor', 'rdexpgdp', 'patgrantedstock', 'plantpatstock',
       'designpatstock', 'plantpat']]
y=df['designpat']
```

In [22]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Linear Regression

In [23]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[23]: LinearRegression()

In [24]:
```
lr.intercept_
```

Out[24]: -764.4487900938358

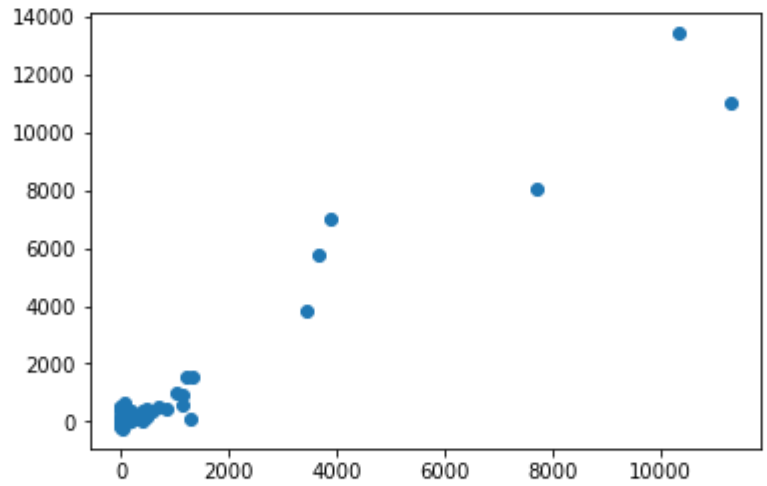
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
year	3.912868e-01
eap	5.333837e+00
eca	-5.922969e+00
lac	-3.187443e+00
mena	-2.360544e+00
sha	-2.628649e+00
sa	-2.353376e+00
hi	-1.965513e+01
pat	9.872656e-03
patepo	-5.267820e-03
royal	-3.015520e-08
rdexp	6.506588e-10
rdper	-1.735738e-05
rdfinabro	-1.350879e-01
rdfinprod	4.810888e-01
rdperfprod	1.392027e-01
rdperfhe	-1.862709e-02
rdperfpub	-1.106904e-01
lowrdexp	-6.039613e-14
lowrdfinprod	-2.905639e-01
lowrdperfprod	-1.879814e+00
y	-1.845861e-11
stockpatEPO	-3.208420e-04
poptotal	-1.301376e-08
labor	2.067889e-08
rdexpgdp	-1.324317e+01
patgrantedstock	1.005447e-04
plantpatstock	-8.437710e-01
designpatstock	8.631382e-02
plantpat	1.307181e+01

In [26]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[26]: <matplotlib.collections.PathCollection at 0x161b3d09940>



ACCURACY

In [27]:
```
lr.score(x_test,y_test)
```

Out[27]: 0.9162096408969369

In [28]:
```
lr.score(x_train,y_train)
```

Out[28]: 0.9634754764411437

Ridge and Lasso

In [29]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [30]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:14
7: LinAlgWarning: Ill-conditioned matrix (rcond=6.86535e-27): result may not
be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

Out[30]: Ridge(alpha=10)

Accuracy(Ridge)

In [31]:
```
rr.score(x_test,y_test)
```

Out[31]: 0.9162124778768015

In [32]:
```
en.score(x_train,y_train)
```

Out[32]: 0.9634750582706163

In [33]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[33]: Lasso(alpha=10)

In [34]:
```
la.score(x_train,y_train)
```

Out[34]: 0.9631243502492564

Accuracy(Lasso)

In [35]:
```
la.score(x_test,y_test)
```

Out[35]: 0.9165661481463434

ElasticNet

In [36]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[36]: ElasticNet()

In [37]:
```
en.coef_
```

Out[37]:
```
array([ 4.07138505e-01,  6.08336385e-01, -0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00, -1.25164599e+00,
        1.00103255e-02, -4.76758743e-03, -3.15108699e-08,  7.17148629e-10,
       -1.63658098e-05, -4.86910773e-02,  2.68159282e-01,  1.00464920e-01,
       -7.91213036e-03, -0.00000000e+00,  0.00000000e+00, -0.00000000e+00,
       -0.00000000e+00, -2.76853805e-11, -3.73949002e-04,  7.78322034e-10,
        2.11411623e-08, -6.83653979e-01,  1.15933568e-04, -8.48042740e-01,
        8.70159523e-02,  1.28954823e+01])
```

In [38]:
```
en.intercept_
```

Out[38]: -811.8228637770986

In [39]:
```
prediction=en.predict(x_test)
```

In [40]:
```
en.score(x_test,y_test)
```

Out[40]: 0.9162648062418464

Evaluation Metrics

In [41]:
```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test,prediction))
print(metrics.mean_squared_error(y_test,prediction))
print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
13.73174811011863
11614.398900657516
107.7701206302448
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
feature_matrix=df[['year', 'eap', 'eca', 'lac', 'mena', 'sha', 'sa',
       'hi', 'pat', 'patepo', 'royal', 'rdexp', 'rdper', 'rdfinabro',
       'rdfinprod', 'rdperfprod', 'rdperfhe', 'rdperfpub', 'lowrdexp',
       'lowrdfinprod', 'lowrdperfprod', 'y', 'stockpatEPO', 'poptotal',
       'labor', 'rdexpgdp', 'patgrantedstock', 'plantpatstock',
       'designpatstock', 'plantpat']]
target_vector=df['designpat']
```

In [44]:
```
feature_matrix.shape
```

Out[44]:
```
(8295, 30)
```

In [45]:
```
target_vector.shape
```

Out[45]:
```
(8295,)
```

In [46]:
```
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[48]:
```
LogisticRegression(max_iter=10000)
```

In [49]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,2
```

In [50]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[212.]
```

In [51]:
```
logn_classes
```

Out[51]:
```
array([0.0000e+00, 1.0000e+00, 2.0000e+00, 3.0000e+00, 4.0000e+00,
       5.0000e+00, 6.0000e+00, 7.0000e+00, 8.0000e+00, 9.0000e+00,
       1.0000e+01, 1.1000e+01, 1.2000e+01, 1.3000e+01, 1.4000e+01,
       1.5000e+01, 1.6000e+01, 1.7000e+01, 1.8000e+01, 1.9000e+01,
       2.0000e+01, 2.1000e+01, 2.2000e+01, 2.3000e+01, 2.4000e+01,
       2.5000e+01, 2.6000e+01, 2.7000e+01, 2.8000e+01, 2.9000e+01,
       3.0000e+01, 3.2000e+01, 3.3000e+01, 3.4000e+01, 3.7000e+01,
       3.8000e+01, 3.9000e+01, 4.0000e+01, 4.1000e+01, 4.2000e+01,
       4.3000e+01, 4.4000e+01, 4.5000e+01, 4.6000e+01, 4.7000e+01,
       4.8000e+01, 4.9000e+01, 5.0000e+01, 5.4000e+01, 5.5000e+01,
       5.6000e+01, 5.7000e+01, 5.8000e+01, 5.9000e+01, 6.0000e+01,
       6.1000e+01, 6.2000e+01, 6.3000e+01, 6.4000e+01, 6.5000e+01,
       6.6000e+01, 6.7000e+01, 6.9000e+01, 7.0000e+01, 7.1000e+01,
       7.2000e+01, 7.3000e+01, 7.4000e+01, 7.6000e+01, 7.7000e+01,
       7.8000e+01, 8.0000e+01, 8.1000e+01, 8.2000e+01, 8.3000e+01,
       8.4000e+01, 8.5000e+01, 8.6000e+01, 8.7000e+01, 8.8000e+01,
       8.9000e+01, 9.0000e+01, 9.1000e+01, 9.3000e+01, 9.4000e+01,
       9.5000e+01, 9.6000e+01, 9.7000e+01, 9.8000e+01, 9.9000e+01,
       1.0000e+02, 1.0100e+02, 1.0200e+02, 1.0300e+02, 1.0500e+02,
       1.0600e+02, 1.0700e+02, 1.0900e+02, 1.1000e+02, 1.1200e+02,
       1.1300e+02, 1.1400e+02, 1.1500e+02, 1.1700e+02, 1.1800e+02,
       1.1900e+02, 1.2000e+02, 1.2100e+02, 1.2300e+02, 1.2400e+02,
       1.2500e+02, 1.2600e+02, 1.2700e+02, 1.2900e+02, 1.3200e+02,
       1.3300e+02, 1.3400e+02, 1.3600e+02, 1.3800e+02, 1.3900e+02,
       1.4200e+02, 1.4300e+02, 1.4400e+02, 1.4700e+02, 1.5000e+02,
       1.5200e+02, 1.5300e+02, 1.5600e+02, 1.5900e+02, 1.6000e+02,
       1.6200e+02, 1.6300e+02, 1.6500e+02, 1.6900e+02, 1.7200e+02,
       1.7300e+02, 1.7600e+02, 1.7900e+02, 1.8000e+02, 1.8100e+02,
       1.8200e+02, 1.8400e+02, 1.8500e+02, 1.8600e+02, 1.9000e+02,
       1.9300e+02, 1.9500e+02, 1.9600e+02, 1.9700e+02, 2.0100e+02,
       2.0200e+02, 2.0500e+02, 2.0800e+02, 2.1100e+02, 2.1200e+02,
       2.1300e+02, 2.1500e+02, 2.1600e+02, 2.2100e+02, 2.2200e+02,
       2.2700e+02, 2.2800e+02, 2.3000e+02, 2.3100e+02, 2.3400e+02,
       2.3700e+02, 2.3900e+02, 2.4000e+02, 2.4300e+02, 2.4700e+02,
       2.5000e+02, 2.5300e+02, 2.5400e+02, 2.5700e+02, 2.5800e+02,
       2.6000e+02, 2.6500e+02, 2.7500e+02, 2.8200e+02, 3.0000e+02,
       3.0600e+02, 3.2000e+02, 3.3000e+02, 3.3800e+02, 3.4100e+02,
       3.5000e+02, 3.5600e+02, 3.6000e+02, 3.6800e+02, 3.7000e+02,
       3.7200e+02, 3.8200e+02, 3.9000e+02, 3.9600e+02, 4.0100e+02,
       4.1000e+02, 4.1800e+02, 4.3800e+02, 4.3900e+02, 4.6600e+02,
       4.8200e+02, 4.8400e+02, 4.8500e+02, 5.0300e+02, 5.0500e+02,
       5.0900e+02, 5.2200e+02, 5.3900e+02, 5.4700e+02, 5.7600e+02,
       5.8800e+02, 6.2000e+02, 6.9800e+02, 7.0500e+02, 7.9500e+02,
       8.3300e+02, 8.6100e+02, 9.3600e+02, 9.4800e+02, 1.0260e+03,
       1.0370e+03, 1.0560e+03, 1.1350e+03, 1.1370e+03, 1.1490e+03,
       1.1680e+03, 1.2070e+03, 1.2940e+03, 1.3070e+03, 1.3100e+03,
       1.3640e+03, 1.4970e+03, 1.5460e+03, 2.4690e+03, 3.0520e+03,
       3.0550e+03, 3.0650e+03, 3.2780e+03, 3.4280e+03, 3.4460e+03,
       3.4750e+03, 3.5460e+03, 3.5700e+03, 3.6450e+03, 3.8830e+03,
       3.9020e+03, 5.0690e+03, 6.0130e+03, 6.0750e+03, 7.4160e+03,
       7.6970e+03, 7.7470e+03, 7.8630e+03, 8.2510e+03, 9.3250e+03,
       9.6540e+03, 9.9130e+03, 1.0346e+04, 1.1285e+04])
```

In [52]:
```
logr.score(fs,target_vector)
```

Out[52]: 0.8764315852923448

In [53]:
```
logr.predict_proba(observation)[0][0]
```

Out[53]: 0.0

In [54]:
```
logr.predict_proba(observation)
```

Out[54]:
```
array([[0.00000000e+00, 6.46875839e-85, 4.18064698e-86, 4.97640492e-80,
        8.59945817e-83, 2.37734566e-56, 3.98520696e-68, 2.90827307e-74,
        3.93470301e-68, 7.84564082e-60, 4.43221229e-85, 6.56263221e-55,
        1.03039805e-73, 1.08013371e-59, 1.08112169e-50, 7.94176532e-68,
        1.81569452e-50, 2.60212594e-61, 4.31128405e-50, 7.36175743e-49,
        3.51381920e-52, 8.03683141e-50, 4.61455481e-44, 9.29966791e-71,
        2.15616849e-32, 1.47123590e-51, 3.22397464e-25, 1.13656197e-42,
        3.84756439e-36, 3.92528441e-58, 2.57777087e-38, 4.08345965e-33,
        4.43714065e-49, 1.95954619e-43, 1.45852110e-54, 1.32840786e-50,
        1.61612437e-22, 2.16834024e-29, 1.26908838e-34, 8.98132380e-30,
        2.17424290e-44, 4.15451744e-15, 6.05958093e-36, 3.76411606e-12,
        1.64039866e-26, 5.78586665e-31, 1.98698848e-18, 1.60937793e-35,
        3.86587379e-25, 7.65725299e-09, 1.68641093e-36, 7.44871984e-19,
        9.91118520e-31, 3.27302787e-26, 4.54761528e-32, 3.46546939e-26,
        3.43068716e-38, 2.09551086e-46, 1.47074020e-24, 5.60652672e-29,
        4.46188339e-14, 5.23448116e-35, 4.90515430e-13, 4.19574969e-43,
        1.88139168e-38, 5.87960896e-41, 1.68589460e-22, 1.06641334e-28,
        4.31086801e-32, 2.11026858e-21, 1.36350787e-13, 2.68466590e-17,
        2.06256309e-27, 4.64149327e-37, 6.11020490e-11, 1.77915082e-04,
        1.04327068e-33, 9.27239443e-25, 1.64059124e-19, 1.41867503e-26,
        3.45576309e-25, 3.78644588e-25, 5.53565304e-21, 1.10810312e-32,
        4.17326226e-30, 2.85198531e-14, 1.70870740e-16, 2.37696717e-31,
        4.57363954e-20, 1.10518235e-31, 7.04942289e-18, 7.09899878e-15,
        6.18243195e-20, 5.14010433e-12, 1.30131822e-26, 2.75215467e-28,
        3.56175114e-23, 1.43488680e-20, 1.49026166e-35, 2.31955497e-19,
        2.80943383e-28, 3.26844661e-25, 1.98767004e-25, 2.90381563e-47,
        4.16144030e-38, 2.15517707e-26, 5.16987221e-29, 3.01843396e-43,
        2.43209628e-24, 1.67542453e-22, 1.49536949e-44, 4.37752030e-24,
        1.19030241e-27, 1.26335802e-12, 3.25590978e-18, 1.46940286e-36,
        6.21194731e-25, 1.02987622e-16, 2.55264374e-31, 3.08002770e-28,
        2.73592092e-19, 1.09563396e-20, 7.28661834e-15, 8.63266119e-24,
        6.55186889e-19, 3.67493263e-23, 5.37165816e-11, 1.42090765e-29,
        1.96504387e-26, 3.15806890e-35, 4.67423646e-41, 1.03249746e-42,
        2.06352084e-18, 4.80872013e-11, 2.38144854e-22, 1.44623746e-19,
        1.12561121e-23, 1.27398771e-17, 5.88348615e-41, 9.14398660e-13,
        2.69631717e-12, 3.53809810e-06, 1.35670563e-37, 4.00635816e-07,
        3.66333929e-26, 3.21990721e-11, 4.34472713e-18, 3.82940747e-11,
        4.48980290e-12, 3.70126759e-10, 3.42435599e-16, 2.23506481e-29,
        2.06547055e-22, 1.65143853e-23, 9.99292856e-01, 1.89931562e-40,
        9.95178307e-09, 4.04014787e-33, 6.45046777e-19, 6.39481242e-17,
        1.65711204e-21, 1.66902668e-13, 5.85102749e-09, 3.58553149e-20,
        7.63863608e-14, 2.16528932e-16, 2.31182283e-19, 5.19372698e-10,
        2.85136582e-18, 2.20030544e-23, 8.40363376e-15, 1.52062520e-10,
        2.11702071e-34, 9.43588493e-11, 2.38638165e-34, 6.84723725e-16,
        7.27402502e-24, 1.34617982e-10, 7.45356898e-05, 2.01297770e-13,
        6.90786546e-16, 2.47182409e-26, 4.93804370e-15, 1.42551905e-13,
        2.26831073e-18, 4.95819552e-17, 2.54256226e-26, 3.33888563e-11,
        1.20758731e-31, 2.15710411e-24, 4.34686023e-07, 1.48682527e-40,
        1.80994770e-16, 1.56738491e-25, 5.46790333e-18, 3.54425763e-04,
        1.27586197e-10, 3.05700233e-12, 1.93730817e-21, 5.51384567e-23,
        2.69380119e-17, 3.77031259e-10, 1.01278030e-07, 5.00823656e-20,
        9.23034863e-05, 1.50877311e-19, 3.49094704e-21, 2.04317551e-20,
        6.66023049e-12, 2.20980382e-22, 5.86801028e-22, 8.35022702e-20,
        5.26476071e-19, 2.53780151e-18, 6.88038395e-17, 1.31007494e-15,
```

```
       3.18385328e-17, 1.42025828e-18, 3.67892414e-17, 2.01043312e-14,
       6.10377056e-18, 5.68120253e-21, 2.01399099e-13, 7.29833544e-15,
       3.86518040e-15, 6.63952897e-26, 9.64773366e-12, 1.62949403e-24,
       9.21556898e-19, 2.45420341e-12, 5.20103815e-26, 1.31984657e-10,
       6.15952510e-20, 9.20354758e-19, 5.84975126e-21, 1.31131770e-18,
       8.45741933e-13, 1.03629389e-08, 1.79179668e-13, 2.82884116e-08,
       1.24337025e-11, 6.93118767e-12, 2.24629133e-13, 5.49609681e-07,
       6.10191031e-09, 9.96528220e-11, 3.87985456e-13, 2.53836642e-09,
       1.26770832e-06, 1.24209791e-08, 3.02700348e-08, 1.53830151e-06,
       2.39069364e-11, 2.79232187e-11, 2.18150800e-21, 2.44157090e-13,
       1.53833366e-08, 5.51628248e-12, 1.97159115e-09]])
```

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

In [57]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [58]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
666: UserWarning: The least populated class in y has only 1 members, which is
less than n_splits=2.
  warnings.warn(("The least populated class in y has only %d"
```

Out[58]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [59]:
```
grid_search.best_score_
```

Out[59]:
```
0.8825353083017569
```

In [60]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['variab
'variable11', 'variable12', 'variable13', 'variable14', 'variable15', 'variabl
'variable21', 'variable22', 'variable23', 'variable24', 'variable25', 'variabl
'variable31', 'variable32', 'variable33', 'variable34', 'variable35', 'variabl
'variable41', 'variable42', 'variable43', 'variable44', 'variable45', 'variabl
'variable51', 'variable52', 'variable53', 'variable54', 'variable55', 'variabl
'variable61', 'variable62', 'variable63', 'variable64', 'variable65', 'variabl
'variable71', 'variable72', 'variable73', 'variable74', 'variable75', 'variabl
'variable81', 'variable82', 'variable83', 'variable84', 'variable85', 'variabl
'variable91', 'variable92', 'variable93', 'variable94', 'variable95', 'variabl
```

Out[62]:
```
[Text(1906.5, 1993.2, 'stockpatEPO <= 0.5\ngini = 0.591\nsamples = 3662\nvalu
e = [2635, 2615, 45, 37, 30, 19, 25, 11, 16, 16, 6, 5\n10, 5, 0, 3, 5, 11, 2,
5, 4, 6, 2, 2, 3, 5\n6, 8, 2, 5, 1, 3, 1, 1, 4, 0, 3, 3, 4, 1\n1, 1, 7, 3, 1,
4, 5, 3, 1, 3, 0, 0, 3, 0\n1, 1, 1, 0, 1, 3, 4, 1, 0, 2, 2, 2, 0, 0\n3, 3, 2,
3, 5, 2, 0, 1, 1, 0, 4, 4, 5, 3\n1, 2, 4, 3, 3, 2, 1, 2, 2, 2, 4, 1, 0, 1\n0,
2, 0, 0, 0, 1, 3, 2, 0, 2, 3, 3, 2, 0\n0, 1, 0, 3, 2, 2, 2, 3, 0, 2, 2, 1, 2,
1\n1, 1, 2, 1, 1, 0, 1, 1, 2, 1, 1, 0, 1, 2\n1, 2, 1, 1, 1, 2, 0, 2, 1, 3, 1,
4, 0, 1\n0, 0, 1, 1, 0, 2, 2, 1, 3, 2, 0, 0, 0, 0\n1, 1, 1, 0, 3, 1, 2, 0, 0,
1, 2, 1, 0, 1\n0, 0, 1, 1, 0, 1, 0, 0, 2, 0, 1, 1, 1, 1\n0, 2, 3, 2, 1, 0, 2,
1, 4, 0, 2, 1, 0, 0\n0]\nclass = variable1'),
 Text(837.0, 1630.8000000000002, 'pat <= 42.0\ngini = 0.145\nsamples = 634\nv
alue = [946, 52, 6, 8, 2, 0, 6, 1, 1, 0, 0, 1, 0, 0\n0, 0, 0, 0, 0, 0, 0,
2, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\
n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0,
```

Conclusion

Accuracy

In [63]:
```
print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
```

```
Linear Regression: 0.9162096408969369
Ridge Regression: 0.9162124778768015
Lasso Regression 0.9165661481463434
ElasticNet Regression: 0.9162648062418464
Logistic Regression: 0.8764315852923448
Random Forest: 0.8825353083017569
```

Lasso Regression is suitable for this dataset