23-1 2

October 4, 2024

[1333]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats
import warnings
from pandas.errors import SettingWithCopyWarning
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression

warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)

class BoxAn:
    def __init__(self,data:pd.
 ↪DataFrame,from_col=0,till_col=None,y_col=0,from_col_x=1):
        self.data=data.iloc[:,from_col:till_col]
        self.prev_data=data.iloc[:,from_col:till_col]
        self.y_col=data.iloc[:,y_col]
        self.x_col=data.iloc[:,from_col_x:till_col]

    def get_stat_min_max(self,data_describe):
        """gets quantile delta drom pd.DataFrame.describe()

        Args:
            data_describe (pd.DataFrame): pd.DataFrame.describe()

        Returns:
            tuple: (min,max)
        """
        stat_min = data_describe['25%'] - 1.5*(data_describe['75%'] -␣
 ↪data_describe['25%'])
        stat_max = data_describe['75%'] + 1.5*(data_describe['75%'] -␣
 ↪data_describe['25%'])

        return (stat_min, stat_max)

    def drop_bad_data(self,data=pd.DataFrame(),y_col=0):
        if data.shape==(0,0):
            data=self.data

        stat_data = data.describe()

        min_max = {column_name: self.get_stat_min_max(stat_data[column_name])␣
↪for column_name in data.describe().columns[y_col:]}

        for column_name in data.columns[y_col:]:
            median = data[np.logical_and(data[column_name] >␣
↪min_max[column_name][0], data[column_name] <␣
↪min_max[column_name][1])][column_name].median()

            data[column_name] = np.where(np.logical_or(data[column_name] <␣
↪min_max[column_name][0], data[column_name] >␣
↪min_max[column_name][1]),median,data[column_name])
        self.data=data
        return data

    def drop_max_y(self,data=pd.DataFrame(),y_col=0,times=5):
        if data.shape==(0,0):
            data=self.data
        name=data.columns[y_col]

        for i in range(times):

            data[name]=np.where(data[name]==max(data[name]),np.nan,data[name])

            data=data.dropna()
        self.data=data
        return data

    def remove_bad_data_x_times(self, data=pd.DataFrame(),times=1):
        if data.shape==(0,0):
            data=self.data

        for i in range(times):
            data=self.drop_bad_data(data)

        self.data=data
        return data

    def boxplots(self,data=pd.DataFrame(),figsize=(15,4)):
        """Makes boxplots for every column in dataframe

        Args:
            data (pd.DataFrame)
        """
        if data.shape==(0,0):
            data=self.data

        fig,ax = plt.subplots(1,data.shape[1],figsize=figsize)
        fig.tight_layout(pad=3)
        for i,j in zip(data.iloc[:,0:],ax):
            j.boxplot(data[i])
            j.set_title(i)
        plt.show()

    def regress_model_plot(self,X,Y,n=10_000,plotter = plt):

        x = np.linspace(X.min(),X.max(),n)
        b0,b1 = scipy.stats.linregress(X,Y).intercept,scipy.stats.linregress(X,Y).slope

        l = lambda x: b1 * x + b0
        y = [l(i) for i in x]
        plotter.plot(x,y)
        return x,y

    def scatterplots(self,data=pd.DataFrame(),figsize=(15,10), lines=False,y_col=0):
        if data.shape==(0,0):
            data=self.data

        shape = data.shape[1]-1

        dicts=dict()
        dicts2=dict()

        for i in range(1,shape+1):
            if shape%i==0:
                dicts[i]=shape//i

        for key, value in dicts.items():
            dicts2[abs(key-value)]=(key,value)

        shape_of_graph=dicts2[min(dicts2.keys())]

        fig,ax = plt.subplots(*shape_of_graph[::-1], figsize=figsize)



        for i,j in zip(data.iloc[:,y_col+1:].columns,ax.flatten()):
            j.scatter(data[i],data.iloc[:,y_col])
            j.set_title(i)
            if lines:
                self.regress_model_plot(data[i],data.iloc[:,y_col],plotter = j)
        plt.show()

    def scatterplot(self,data=pd.DataFrame(),figsize=(15,10),y_col=0):
        if data.shape==(0,0):
            data=self.data

        fig,ax = plt.subplots(figsize=figsize)

        for i in data.iloc[:,y_col+1:].columns:
            plt.scatter(data[i],data.iloc[:,y_col])

        plt.show()

    def corr_matrix(self,data=pd.DataFrame(),y_col=0,seaborn=True):

        if data.shape==(0,0):
            data=self.data
        if seaborn:
            sns.heatmap(data.iloc[:,y_col:].corr(),xticklabels=True,␣
↪yticklabels=True,annot=True, cmap='coolwarm')
            plt.show()
        else:
            return

    def drop_too_correlated(self,data=pd.DataFrame(),y_col=0, max_correl=0.9):
        if data.shape==(0,0):
            data=self.data

        cor_df=data.corr()
        k_flag=0
        for i in range(1,len(cor_df.columns)):
            for j in range(i+1,len(cor_df.columns)):
                if cor_df.iloc[i,j]>=max_correl:
                    data.pop(*cor_df[cor_df.iloc[:,y_col]==min(cor_df.
↪iloc[y_col,i],cor_df.iloc[y_col,j])].index)
                    k_flag=1
                    break
            if k_flag==1:
                break
        self.data=data
        self.corr_matrix()

    def drop_too_correlated_no_breaks(self,data=pd.DataFrame(),y_col=0,␣
↪max_correl=0.9):
        if data.shape==(0,0):
            data=self.data

        cor_df=data.corr()
        k_flag=0
        for i in range(1,len(cor_df.columns)):
            for j in range(i+1,len(cor_df.columns)):
                if cor_df.iloc[i,j]>=max_correl:
                    data.pop(*cor_df[cor_df.iloc[:,y_col]==min(cor_df.
  ↪iloc[y_col,i],cor_df.iloc[y_col,j])].index)

        self.data=data
        self.corr_matrix()

    def drop_too_correlated_x_times(self,times=1,data=pd.DataFrame(),y_col=0,␣
  ↪max_correl=0.9):
        if data.shape==(0,0):
            data=self.data

        for i in range(times):
            self.drop_too_correlated(data,y_col, max_correl)

    def corr_type(self,r)->str:

        ans=''
        if r > 0:
            ans += '    '
        elif r < 0:
                ans += '     '

        if abs(r) == 0:
                ans += '         '
        elif abs(r) < 0.3:
                ans += '        '
        elif 0.3 <= abs(r) < 0.5:
                ans += '   '
        elif 0.5 <= abs(r) < 0.7:
                ans += '      '
        elif 0.7 <= abs(r) < 1:
                ans += '     '
        else:
                ans += '         '
        return ans

    def t_test(self,data=pd.DataFrame,col1=0,col2=1,untrust=0.05):
        return scipy.stats.ttest_rel(data.iloc[:, col1], data.iloc[:, col2])[1]␣
  ↪< untrust
```

```
    def define_corr(self,data=pd.DataFrame(),y_col=0,x_col=1,prin=True)->str:

        if data.shape==(0,0):
            data=self.data

        r=data.iloc[:,x_col].corr( data.iloc[:,y_col])
        names=data.columns
        ans=self.corr_type(r)

        if self.t_test(data,y_col,x_col):
                ans += f'(             {r:.3f} -           )'
        else:
                ans += f'(             {r:.3f} -             )'

        if prin:
            print(f'       {names[y_col]}   {names[x_col]} - {ans}\n')
        else:
            return ans

    def correlations(self,data=pd.DataFrame()):

        if data.shape==(0,0):
            data=self.data

        cor_df=data.corr()

        for i in range(len(cor_df.columns)):
            for j in range(i+1,len(cor_df.columns)):

                self.define_corr(y_col=i,x_col=j)

    def sm_make_model(self,data=pd.DataFrame(),y_col=0):
        if data.shape==(0,0):
            data=self.data

        self.y_col=data.iloc[:,y_col]
        self.x_col=pd.concat([pd.Series([1 for i in range(self.data.iloc[:
↪,y_col+1:].shape[0])], index=self.data.iloc[:,y_col+1:].index, name='X0'),␣
↪self.data.iloc[:,y_col+1:]], axis=1)

        self.model = sm.OLS(self.y_col.values, self.x_col.values).fit()
        return self.model.summary()

    def sm_get_max_p(self, data=pd.DataFrame(),untrust = 0.05, use_max=False):
        if data.shape==(0,0):
            data=self.data
        dropper=[]
        d = {data.columns[i]: self.model.pvalues[i] for i in range(0, len(data.
↪columns))}
        for k, v in d.items():
            if use_max:
                if v == max(d.values()) and v > untrust:
                    dropper.append(k)
            else:
                if  v > untrust:
                    dropper.append(k)
        return dropper

    def sm_remove_columns(self,times=1,data=pd.DataFrame(),y_col=0):
        if data.shape==(0,0):
            data=self.data

        list_on_drop = []

        list_on_drop.extend(self.sm_get_max_p(self.x_col))
        self.x_col=self.x_col.drop(list_on_drop, axis=1)

        if all([self.x_col.shape[i]!=0 for i in range(len(self.x_col.shape))])␣
↪and all([self.y_col.shape[i]!=0 for i in range(len(self.y_col.shape))]):


            self.data=pd.concat([self.y_col,self.x_col],axis=1)
            self.model = sm.OLS(self.y_col, self.x_col.values).fit()
            self.corr_matrix()
            self.p_values={self.x_col.columns[i]: self.model.pvalues[i] for i␣
↪in range(0, len(self.x_col.columns))}

    def sl_make_model(self,data=pd.DataFrame(),y_col=0):
        if data.shape==(0,0):
            data=self.data
        self.lin_model = LinearRegression().fit(self.data.iloc[:,y_col+1:],self.
↪data.iloc[:,y_col])
        return self.lin_model.score(self.data.iloc[:,y_col+1:],self.data.iloc[:
↪,y_col])

    def sl_plot(self,data=pd.DataFrame(),y_col=0):
        if data.shape==(0,0):
            data=self.data
        fig,ax = plt.subplots(1,figsize=(7,7))

        y_ = self.lin_model.predict(self.data.iloc[:,y_col+1:])

        for i in range(1,len(data.columns)):
            ax.scatter(data.iloc[:,y_col],data.iloc[:,i])

        ax.plot(sorted(data.iloc[:,y_col]),y_, color='green')
        plt.show()

    def complex_analysys(self,drop_y=0, replace_to_median=0):
        print('               .')
        self.boxplots()
        t=''
        if drop_y:
            t+=f'    {drop_y}           Y'
            self.drop_max_y(times=drop_y)
        if replace_to_median:
            t+=f'''
                                 .

             {replace_to_median}

             , . .                ,             .
            '''
            self.remove_bad_data_x_times(times=replace_to_median)

        self.boxplots()

        print('                 Y         .')
        self.scatterplots(lines=True)
        self.scatterplot(figsize=(6,4))

        print('                               .')
        self.corr_matrix()
        self.drop_too_correlated_no_breaks()
        self.correlations()

        print('                       .                         .')
        display(self.sm_make_model())

        print('            P-                   ')
        self.sm_remove_columns(times=1)

        print(f'               {self.sl_make_model()}.\n                ')
        self.sl_plot()

```

[1334]:
```
df = pd.read_excel('          №2   .xlsx', header=1,index_col=0,␣
 ↪usecols=[1,2,3,4,5,6,7,8])
df.columns=['Y', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6']
```

```
analysys = BoxAn(df)


analysys.data.head()
```

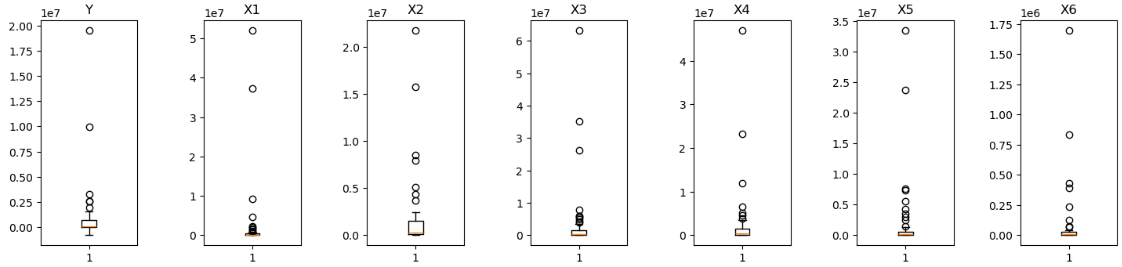
[1334]:

```
          Y        X1       X2        X3        X4        X5       X6
1   1440075     61749  1007355   4920199   5165712   3490541    31365
2      5146     17532    58110     50798     19595     23014        0
3     13612     20268    51271     18903     81072      8678       84
4       964       211     5827     13398      8446      4821        0
5  19513178  52034182  2411352  63269757  47002385  23780450  1696853
```

1 1. . , , . .

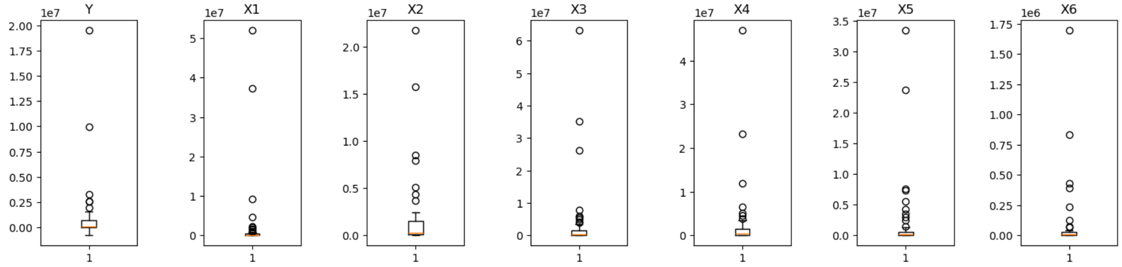
[1335]:

```
analysys.boxplots()
```



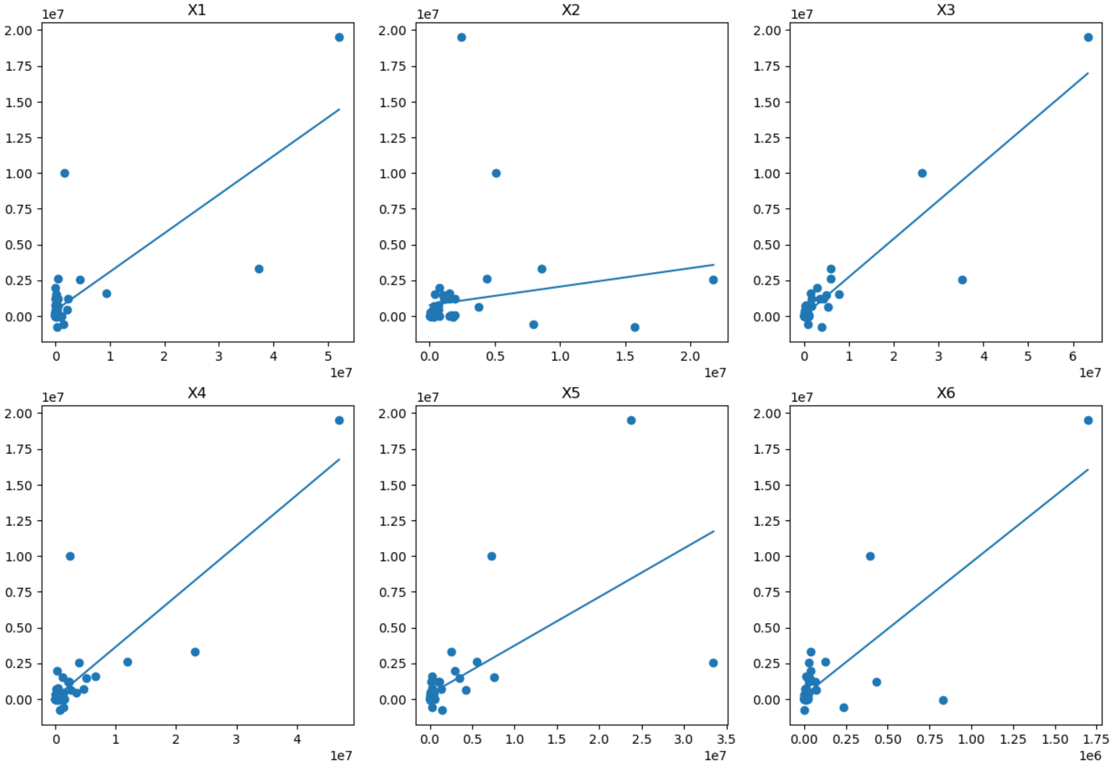
[1336]:

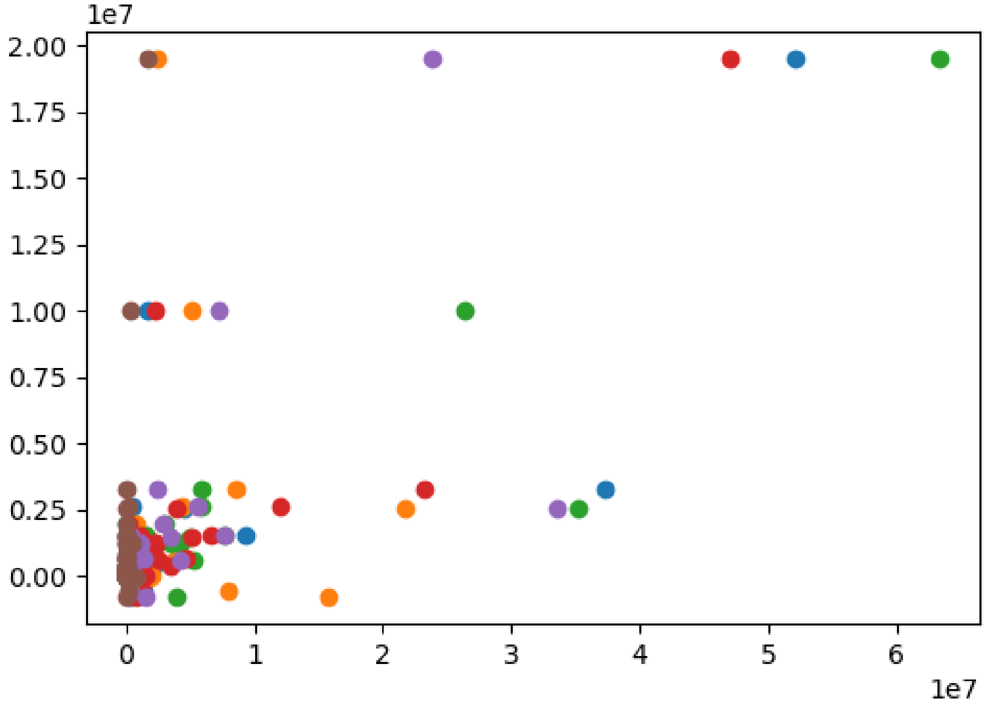
```
#analysys.drop_max_y(times=1)
#analysys.remove_bad_data_x_times()
analysys.boxplots()
```



2 2. . .

```
[1337]: analysys.scatterplots(lines=True)
        analysys.scatterplot(figsize=(6,4))
```



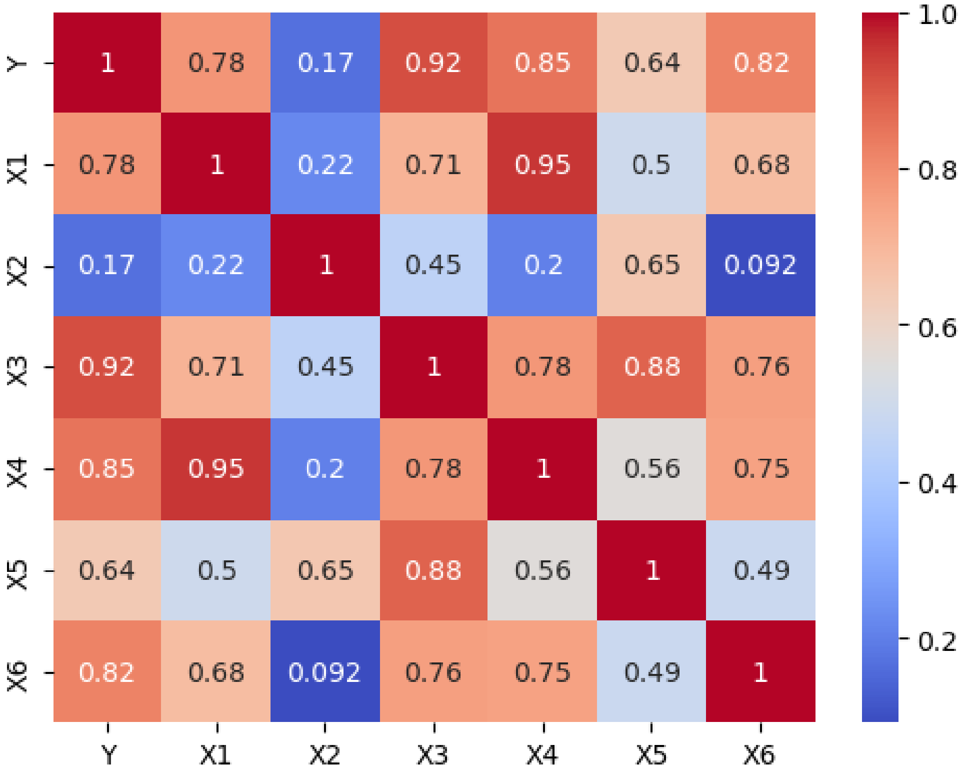


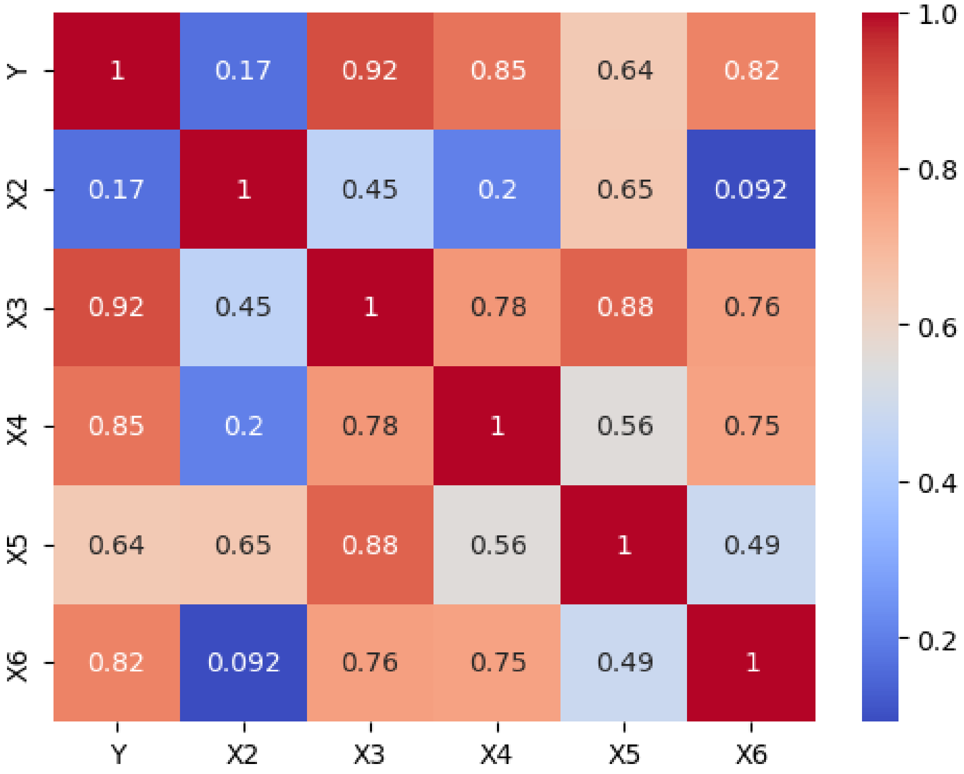
3 3. .

[1338]:

```
analysys.corr_matrix()

analysys.drop_too_correlated_no_breaks()
```


Y 1 0.78 0.17 0.92 0.85 0.64 0.82
X1 0.78 1 0.22 0.71 0.95 0.5 0.68
X2 0.17 0.22 1 0.45 0.2 0.65 0.092
X3 0.92 0.71 0.45 1 0.78 0.88 0.76
X4 0.85 0.95 0.2 0.78 1 0.56 0.75
X5 0.64 0.5 0.65 0.88 0.56 1 0.49
X6 0.82 0.68 0.092 0.76 0.75 0.49 1
Y X1 X2 X3 X4 X5 X6
1.0
0.8
0.6
0.4
0.2


Y 1 0.17 0.92 0.85 0.64 0.82
X2 0.17 1 0.45 0.2 0.65 0.092
X3 0.92 0.45 1 0.78 0.88 0.76
X4 0.85 0.2 0.78 1 0.56 0.75
X5 0.64 0.65 0.88 0.56 1 0.49
X6 0.82 0.092 0.76 0.75 0.49 1
Y X2 X3 X4 X5 X6
1.0
0.8
0.6
0.4
0.2

[1339]:
```
analysys.correlations()
```

```
   Y  X2 -              (             0.167 -
         )

   Y  X3 -           (            0.916 -
       )

   Y  X4 -           (            0.850 -
       )

   Y  X5 -            (             0.643 -
         )

   Y  X6 -           (            0.820 -
       )

   X2  X3 -         (             0.448 -
 )

   X2  X4 -              (             0.205 -
         )

   X2  X5 -            (             0.651 -
         )

   X2  X6 -              (             0.092 -
       )

   X2  X4 -           (             0.777 -
         )

   X3  X5 -           (             0.881 -
         )

   X3  X6 -           (             0.761 -
       )

   X4  X5 -            (             0.556 -
         )

   X4  X6 -           (             0.753 -
       )

   X5  X6 -         (             0.485 -
    )
```

4 4. . . .

[1340]:
```
analysys.sm_make_model()
```

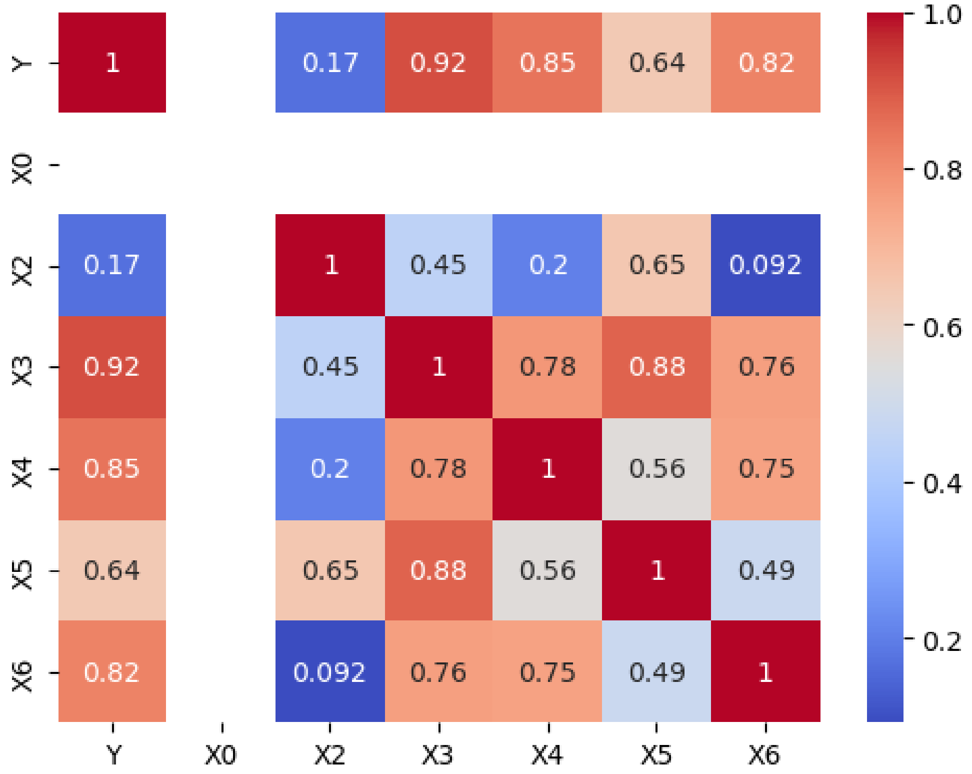
[1340]:
```
Dep. Variable:          y                  R-squared:              0.973
Model:                  OLS                Adj. R-squared:         0.970
Method:                 Least Squares      F-statistic:            343.8
Date:                   Fri, 04 Oct 2024   Prob (F-statistic):     2.40e-36
Time:                   20:46:53           Log-Likelihood:         -783.84
No. Observations:       54                 AIC:                    1580.
Df Residuals:           48                 BIC:                    1592.
Df Model:               5
Covariance Type:        nonrobust

           coef        std err   t         P> |t|   [0.025      0.975]
const      1.974e+05   8e+04     2.467     0.017    3.65e+04    3.58e+05
x1         -0.0893     0.026     -3.444    0.001    -0.141      -0.037
x2         0.4197      0.028     15.049    0.000    0.364       0.476
x3         0.0652      0.018     3.672     0.001    0.029       0.101
x4         -0.3116     0.039     -7.925    0.000    -0.391      -0.233
x5         -1.0911     0.535     -2.040    0.047    -2.167      -0.016

    Omnibus:           30.020   Durbin-Watson:       2.108
    Prob(Omnibus):     0.000    Jarque-Bera (JB):    57.816
    Skew:              1.772    Prob(JB):            2.79e-13
    Kurtosis:          6.624    Cond. No.            1.56e+07
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.56e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

[1341]:
```
analysys.sm_remove_columns(times=1)
```



```
C:\Users\ivant\AppData\Local\Temp\ipykernel_30216\1131384224.py:280:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  self.p_values={self.x_col.columns[i]: self.model.pvalues[i] for i in range(0,
len(self.x_col.columns))}
```

[1342]:
```
analysys.model.summary()
```

[1342]:

Dep. Variable:	Y	**R-squared:**	0.973
Model:	OLS	**Adj. R-squared:**	0.970
Method:	Least Squares	**F-statistic:**	343.8
Date:	Fri, 04 Oct 2024	**Prob (F-statistic):**	2.40e-36
Time:	20:46:53	**Log-Likelihood:**	-783.84
No. Observations:	54	**AIC:**	1580.
Df Residuals:	48	**BIC:**	1592.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	1.974e+05	8e+04	2.467	0.017	3.65e+04	3.58e+05
x1	-0.0893	0.026	-3.444	0.001	-0.141	-0.037
x2	0.4197	0.028	15.049	0.000	0.364	0.476
x3	0.0652	0.018	3.672	0.001	0.029	0.101
x4	-0.3116	0.039	-7.925	0.000	-0.391	-0.233
x5	-1.0911	0.535	-2.040	0.047	-2.167	-0.016

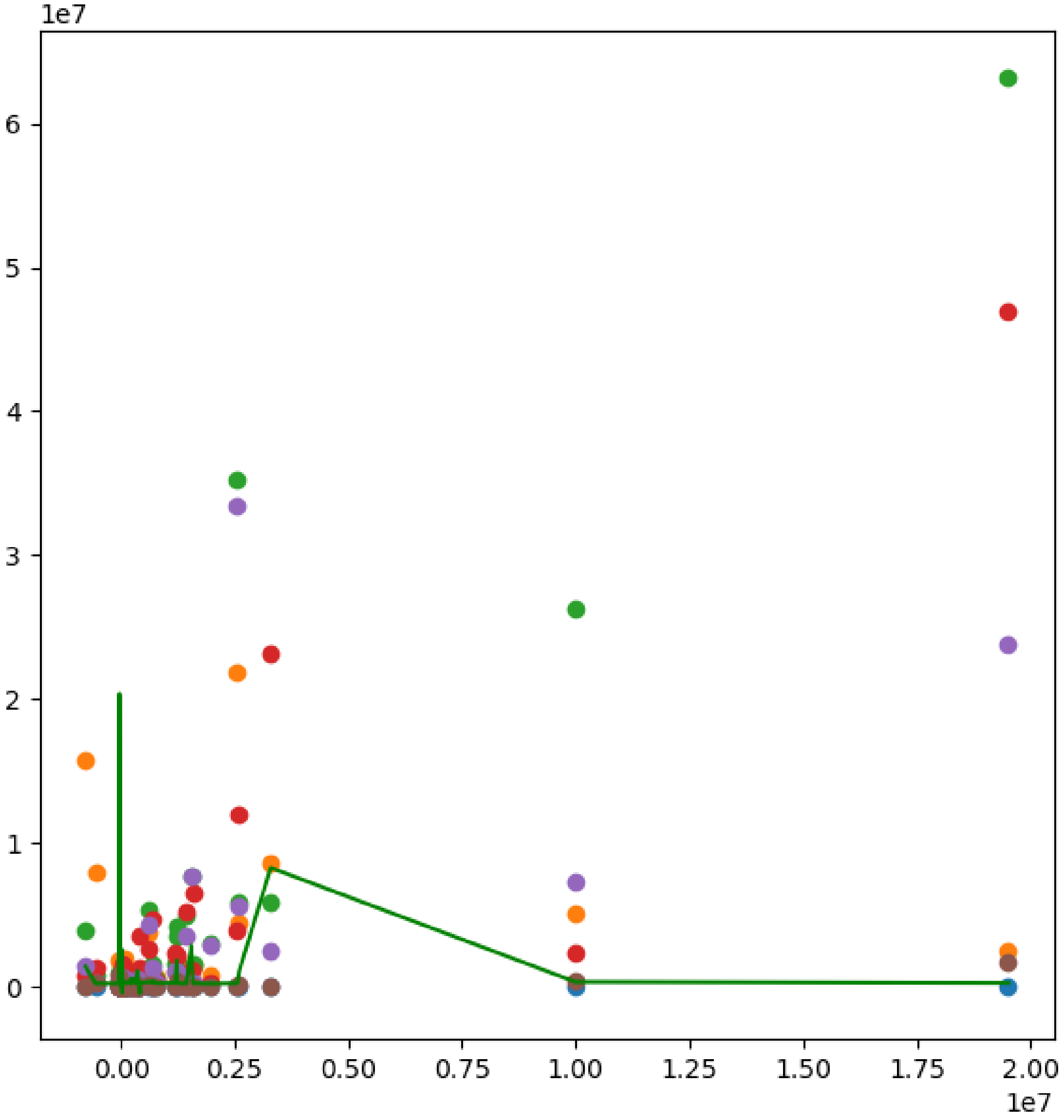
Omnibus:	30.020	**Durbin-Watson:**	2.108
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	57.816
Skew:	1.772	**Prob(JB):**	2.79e-13
Kurtosis:	6.624	**Cond. No.**	1.56e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.56e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

```
[1343]: analysys.sl_make_model()
```

```
[1343]: 0.9728332447664441
```

```
[1344]: analysys.sl_plot()
```


1e7
6
5
4
3
2
1
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7