# Problem Statement

1) With the given independent features predict the Youngs Modulus using regression

2) With the given lab test results predict whether the product quality is good or bad using classification

### Importing required libraries

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

# Data Processing
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.tools.tools import add_constant

# Modelling
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor

# Metics
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from statsmodels.tools.tools import add_constant

from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

In [2]:
```
# reading the file
df=pd.read_excel("Data.xlsx")
```

In [3]:
```
# Checking the basic structure of the data using head
df.head()
```

Out[3]:

| | Grade Name | Polymer Types | Primary Filler type | % of Primary filler | Secondary filler type | % of secondary filler | Orientation | Strain Rate(%/s) | Temperature | Youngs modulus (MPa) | Yield Strain (%) | Yield Stress (MPa) | Elongation at break (%) | Strength at break (MPa) | Quality | Unnamed: 15 | Input Factors |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | L1006 | L | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 52.0 | 4870.0 | 6.6 | 89.6 | 7.4 | 88.9 | Good | NaN | Target factors |
| **2** | L1006 | L | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 107.0 | 4190.0 | 7.3 | 70.0 | 7.7 | 69.6 | Good | NaN | NaN |
| **3** | L1006 | L | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 23.0 | 7270.0 | 4.0 | 118.0 | 5.9 | 117.0 | Good | NaN | NaN |
| **4** | L1006 | L | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 121.0 | 3870.0 | 10.0 | 77.6 | 10.0 | 77.6 | Good | NaN | NaN |

# EDA

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

Out[4]:
```
Index(['Grade Name', 'Polymer Types', 'Primary Filler type',
       '% of Primary filler', 'Secondary filler type', '% of secondary filler',
       'Orientation', 'Strain Rate(%/s)', 'Temperature',
       'Youngs modulus (MPa)', 'Yield Strain (%)', 'Yield Stress (MPa)',
       'Elongation at break (%)', 'Strength at break  (MPa)', 'Quality',
       'Unnamed: 15', 'Input Factors'],
      dtype='object')
```

In [5]:
```
#dropping the unwanted columns
df.drop(columns=['Grade Name','Unnamed: 15', 'Input Factors'],inplace=True)
```

In [6]:
```
# There are a total of 430 data points and 14 data features
df.shape
```

Out[6]:
```
(430, 14)
```

### Analysing the basic metrics

In [7]:
```
# Checking the data type of all the features
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Polymer Types             429 non-null    object 
 1   Primary Filler type       429 non-null    float64
 2   % of Primary filler       426 non-null    float64
 3   Secondary filler type     429 non-null    object 
 4   % of secondary filler     412 non-null    float64
 5   Orientation               429 non-null    float64
 6   Strain Rate(%/s)          429 non-null    float64
 7   Temperature               429 non-null    float64
 8   Youngs modulus (MPa)      428 non-null    float64
 9   Yield Strain (%)          427 non-null    float64
 10  Yield Stress (MPa)        428 non-null    float64
 11  Elongation at break (%)   427 non-null    float64
 12  Strength at break  (MPa)  428 non-null    float64
 13  Quality                   429 non-null    object 
dtypes: float64(11), object(3)
memory usage: 47.2+ KB
```

In [8]:
```
# Checking the stastistical value of each features
df.describe()
```

Out[8]:

| | Primary Filler type | % of Primary filler | % of secondary filler | Orientation | Strain Rate(%/s) | Temperature | Youngs modulus (MPa) | Yield Strain (%) | Yield Stress (MPa) | Elongation at break (%) | Strength at break (MPa) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 429.000000 | 426.000000 | 412.000000 | 429.000000 | 429.000000 | 429.000000 | 428.000000 | 427.000000 | 428.000000 | 427.000000 | 428.000000 |
| **mean** | 0.573427 | 2.152113 | 0.209709 | 0.839161 | 11.412170 | 43.156177 | 5689.282570 | 6.994333 | 80.817126 | 41.806169 | 78.160584 |
| **std** | 0.771782 | 3.021174 | 0.752678 | 8.095213 | 89.895071 | 49.386861 | 5707.299646 | 14.676442 | 48.180093 | 47.360811 | 49.655045 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000100 | -60.000000 | 1.860000 | 0.000000 | 0.000000 | 0.500000 | 1.850000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.083300 | 23.000000 | 2247.500000 | 2.400000 | 53.000000 | 2.569000 | 47.775000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.083300 | 23.000000 | 2665.000000 | 4.410000 | 65.400000 | 10.000000 | 61.900000 |
| **75%** | 1.000000 | 5.000000 | 0.000000 | 0.000000 | 0.833000 | 75.000000 | 7937.500000 | 6.320000 | 94.700000 | 83.615000 | 93.850000 |
| **max** | 2.000000 | 10.000000 | 4.000000 | 90.000000 | 833.300000 | 176.000000 | 32200.000000 | 151.000000 | 276.000000 | 158.000000 | 276.000000 |

In [9]:
```
#finding number of null values
df.isnull().sum()
```

Out[9]:
```
Polymer Types               1
Primary Filler type         1
% of Primary filler         4
Secondary filler type       1
% of secondary filler      18
Orientation                 1
Strain Rate(%/s)            1
Temperature                 1
Youngs modulus (MPa)        2
Yield Strain (%)            3
Yield Stress (MPa)          2
Elongation at break (%)     3
Strength at break  (MPa)    2
Quality                     1
dtype: int64
```

In [10]:
```
#dropping all the null values
df.dropna(inplace=True)
```

In [11]:
```
df.shape
# 21 records removed
```

Out[11]:
```
(409, 14)
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Polymer Types             0
Primary Filler type       0
% of Primary filler       0
Secondary filler type     0
% of secondary filler     0
Orientation               0
Strain Rate(%/s)          0
Temperature               0
Youngs modulus (MPa)      0
Yield Strain (%)          0
Yield Stress (MPa)        0
Elongation at break (%)   0
Strength at break  (MPa)  0
Quality                   0
dtype: int64
```

In [13]:
```
df.drop_duplicates(inplace=True)
```

In [14]:
```
df.shape
# 31 duplicate records removed
```

Out[14]:
```
(378, 14)
```

### Non-Graphical Analysis

In [15]:
```
df.nunique()
# mostly Secondary filler type, Quality, Polymer Types, Primary Filler type, Orientation  and Quality are categorical variable.
# Need to check the same
```

Out[15]:
```
Polymer Types              12
Primary Filler type         3
% of Primary filler        13
Secondary filler type       3
% of secondary filler       6
Orientation                 3
Strain Rate(%/s)           11
Temperature                40
Youngs modulus (MPa)      245
Yield Strain (%)          294
Yield Stress (MPa)        285
Elongation at break (%)   328
Strength at break  (MPa)  297
Quality                     3
dtype: int64
```

In [16]:
```
df['Secondary filler type'].unique()
```

Out[16]:
```
array([0, 3, 1], dtype=object)
```

In [17]:
```
df['Quality'].unique()
#expected  just 2 output but got 3
```

Out[17]:
```
array(['Good', 'Bad ', 'Bad'], dtype=object)
```

In [18]:
```
df['Quality'].replace('Bad ','Bad',inplace=True)
# replacing value with 'Bad ' with 'Bad'
```

In [19]:
```
df['Quality'].unique()
# Corrected
```

Out[19]:
```
array(['Good', 'Bad'], dtype=object)
```

In [20]:
```
df['Orientation'].unique()
# 3 types of orientation
```

Out[20]:
```
array([ 0., 45., 90.])
```

In [21]:
```
df['Primary Filler type'].unique()
# 3 types of fillers
```

Out[21]:
```
array([1., 2., 0.])
```

In [22]:
```
df['Polymer Types'].unique()
# 12 types of polymer types
```

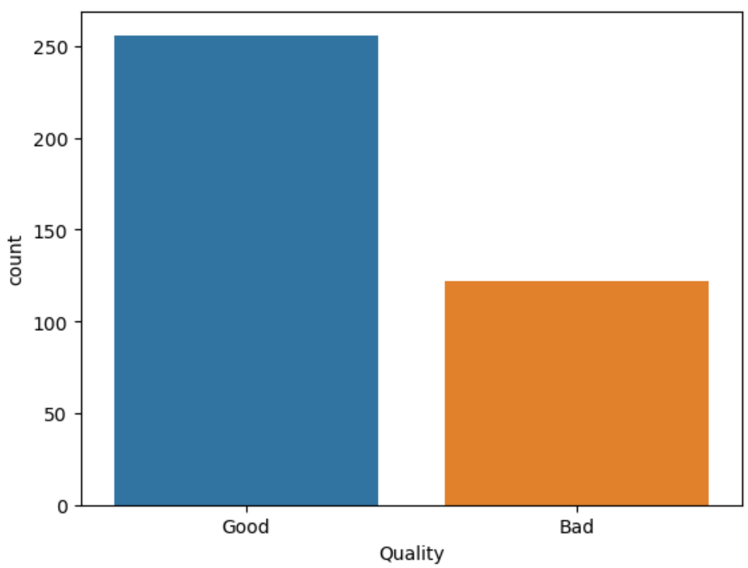
Out[22]:
```
array(['L', 'M', 'F', 'W', 'X', 'A', 'AA', 'S', 'XX', 'PR', 'H', 'CY'],
      dtype=object)
```

# Visual Analysis

In [23]:
```
sns.countplot(df['Quality'])
```

```
C:\Users\gokul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positi
onal argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[23]:
```
<AxesSubplot:xlabel='Quality', ylabel='count'>
```



**Classification data is unbalanced. Accuracy is not be the best metrice.**

**Classifying a bad quaility item as good quality can affect the reputation of the company.**

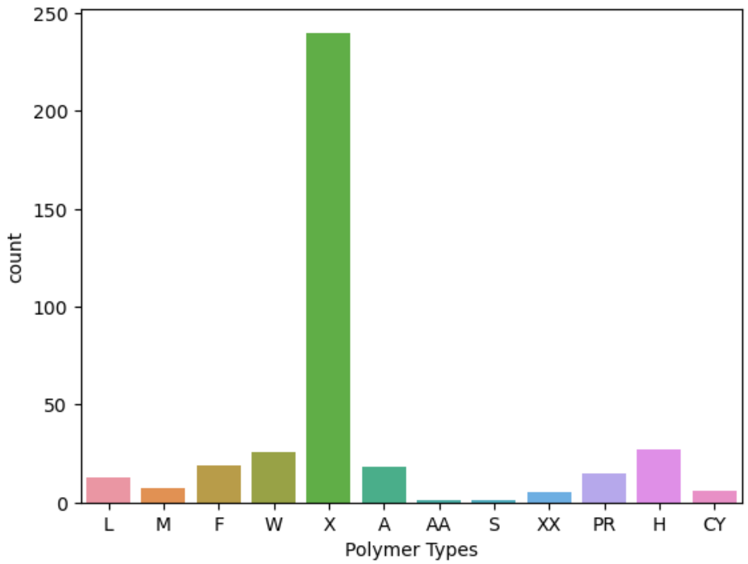
**So model with most Flase Negative is to be penalised**

**We need to optimize the algorithm for best Recall**

In [24]:
```
sns.countplot(df['Polymer Types'])
```

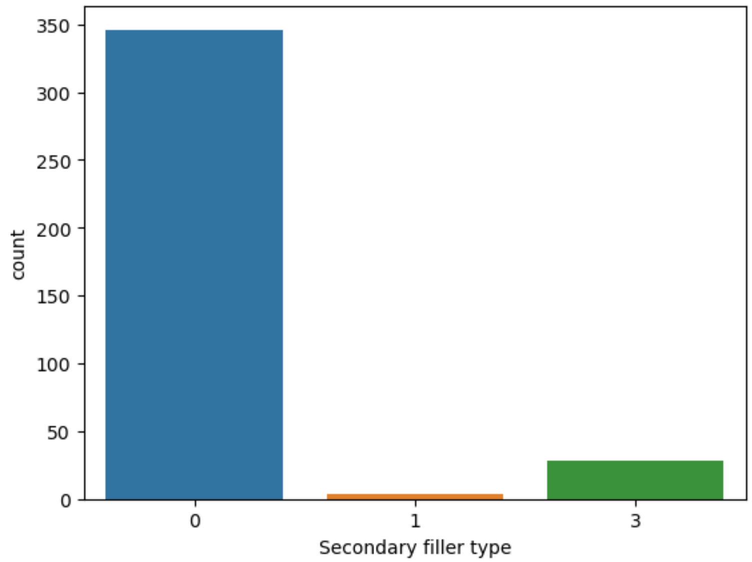
```
C:\Users\gokul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positi
onal argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[24]:
```
<AxesSubplot:xlabel='Polymer Types', ylabel='count'>
```




***Data mostly representing polymer types X***

In [25]:
```
sns.countplot(data=df,x='Secondary filler type')
```
Out[25]: <AxesSubplot:xlabel='Secondary filler type', ylabel='count'>

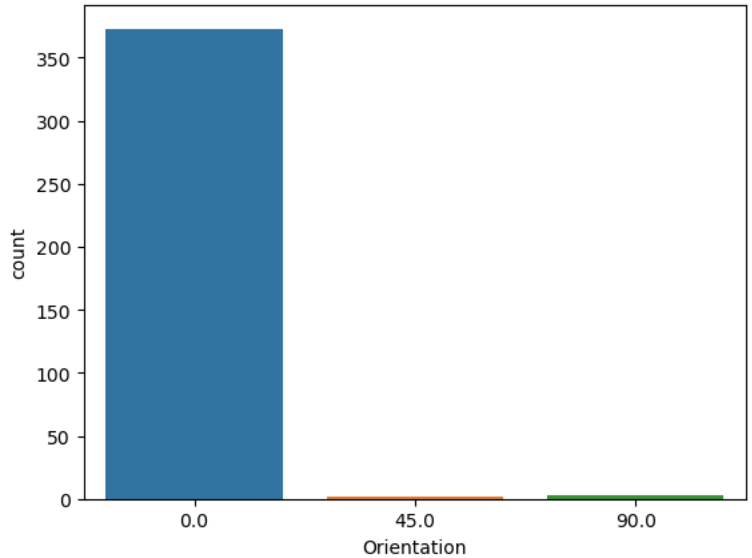



***Most polymer secondary filler material is not used***

In [26]:
```
sns.countplot(df['Orientation'])
```

```
C:\Users\gokul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positi
onal argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[26]:
```
<AxesSubplot:xlabel='Orientation', ylabel='count'>
```




***Fillers are mostly oriented at 0 degrees***

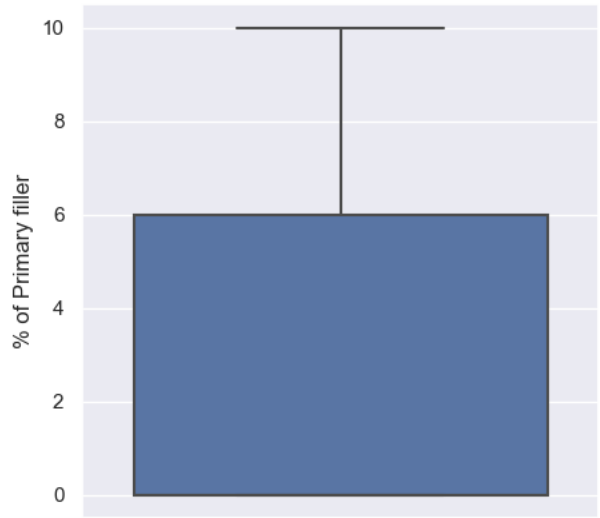
In [27]:
```
df.columns
```

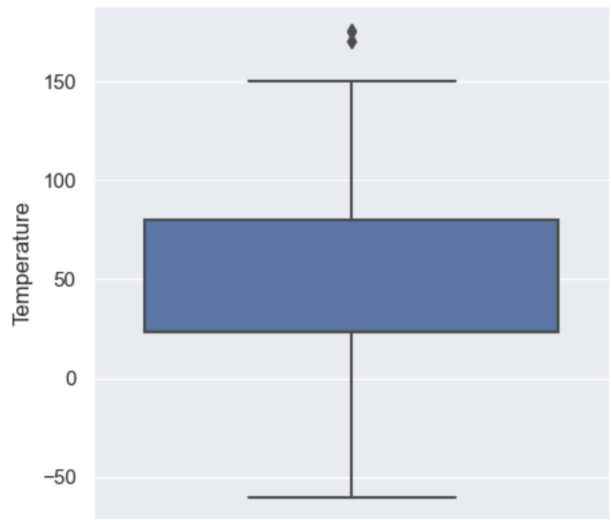
Out[27]:
```
Index(['Polymer Types', 'Primary Filler type', '% of Primary filler',
       'Secondary filler type', '% of secondary filler', 'Orientation',
       'Strain Rate(%/s)', 'Temperature', 'Youngs modulus (MPa)',
       'Yield Strain (%)', 'Yield Stress (MPa)', 'Elongation at break (%)',
       'Strength at break  (MPa)', 'Quality'],
      dtype='object')
```

### Check for outliers

In [28]:
```
sns.set(rc={'figure.figsize':(5,5)})
for i in ['% of Primary filler','Temperature']:
 a=sns.boxplot(data=df,y=i)
 plt.figure(i)
 print("Box Plot",a)
```

```
Box Plot AxesSubplot(0.125,0.11;0.775x0.77)
Box Plot AxesSubplot(0.125,0.11;0.775x0.77)
```




```
<Figure size 500x500 with 0 Axes>
```

In [29]:
```
for i in ['% of Primary filler','Temperature']:
    Q1 = np.percentile(df[i], 25, interpolation = 'midpoint')
    Q2 = np.percentile(df[i], 50, interpolation = 'midpoint')
    Q3 = np.percentile(df[i], 75, interpolation = 'midpoint')
    IQR = Q3 - Q1
    low_lim = Q1 - 1.5 * IQR
    up_lim = Q3 + 1.5 * IQR

    df=df[(df[i]>low_lim) & (df[i]<up_lim)]
```

```
C:\Users\gokul\AppData\Local\Temp\ipykernel_51628\3063489896.py:2: DeprecationWarning: the `interpolation=` argument to percentile was renamed to `method=`, which has additio
nal options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to review the method they. (Deprecated NumPy 1.22)
  Q1 = np.percentile(df[i], 25, interpolation = 'midpoint')
C:\Users\gokul\AppData\Local\Temp\ipykernel_51628\3063489896.py:3: DeprecationWarning: the `interpolation=` argument to percentile was renamed to `method=`, which has additio
nal options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to review the method they. (Deprecated NumPy 1.22)
  Q2 = np.percentile(df[i], 50, interpolation = 'midpoint')
C:\Users\gokul\AppData\Local\Temp\ipykernel_51628\3063489896.py:4: DeprecationWarning: the `interpolation=` argument to percentile was renamed to `method=`, which has additio
nal options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to review the method they. (Deprecated NumPy 1.22)
  Q3 = np.percentile(df[i], 75, interpolation = 'midpoint')
```

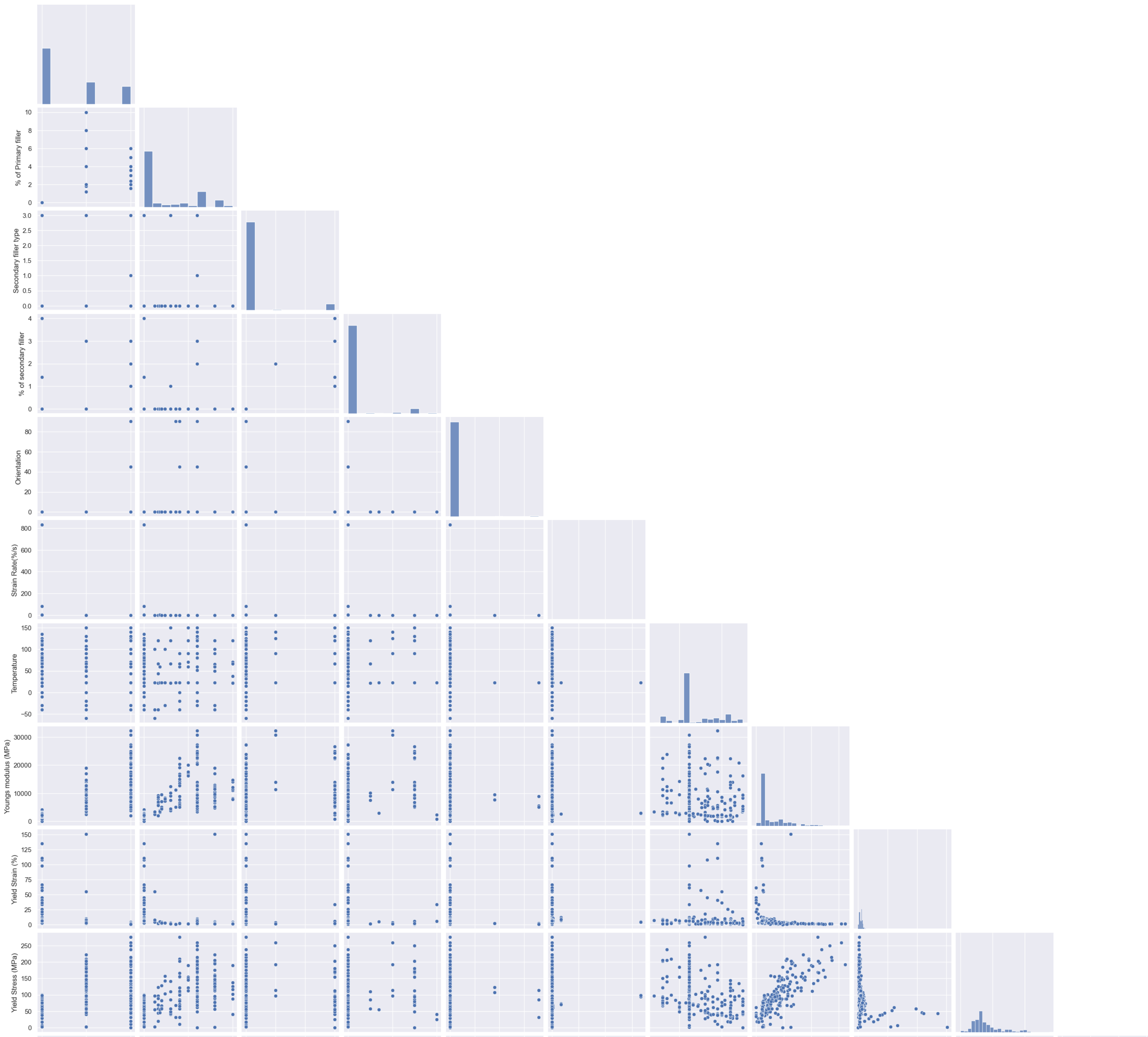
In [30]:
```
df.shape
```

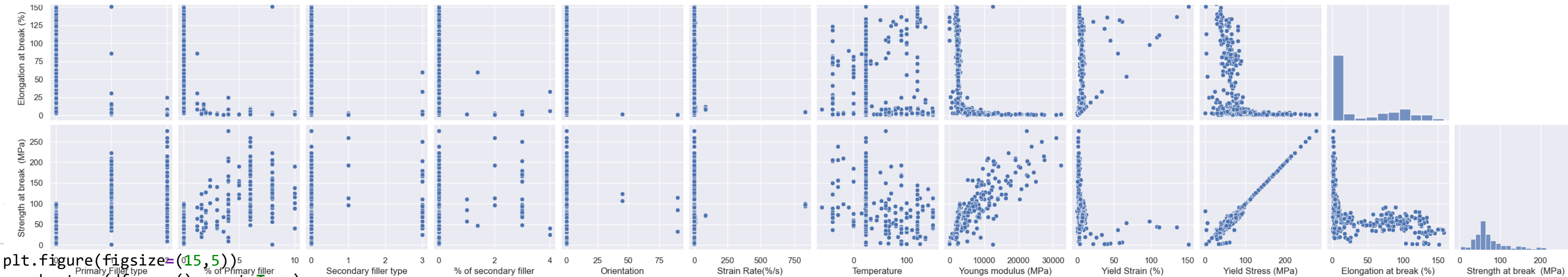
Out[30]:
```
(374, 14)
```

### Bivarient Analysis

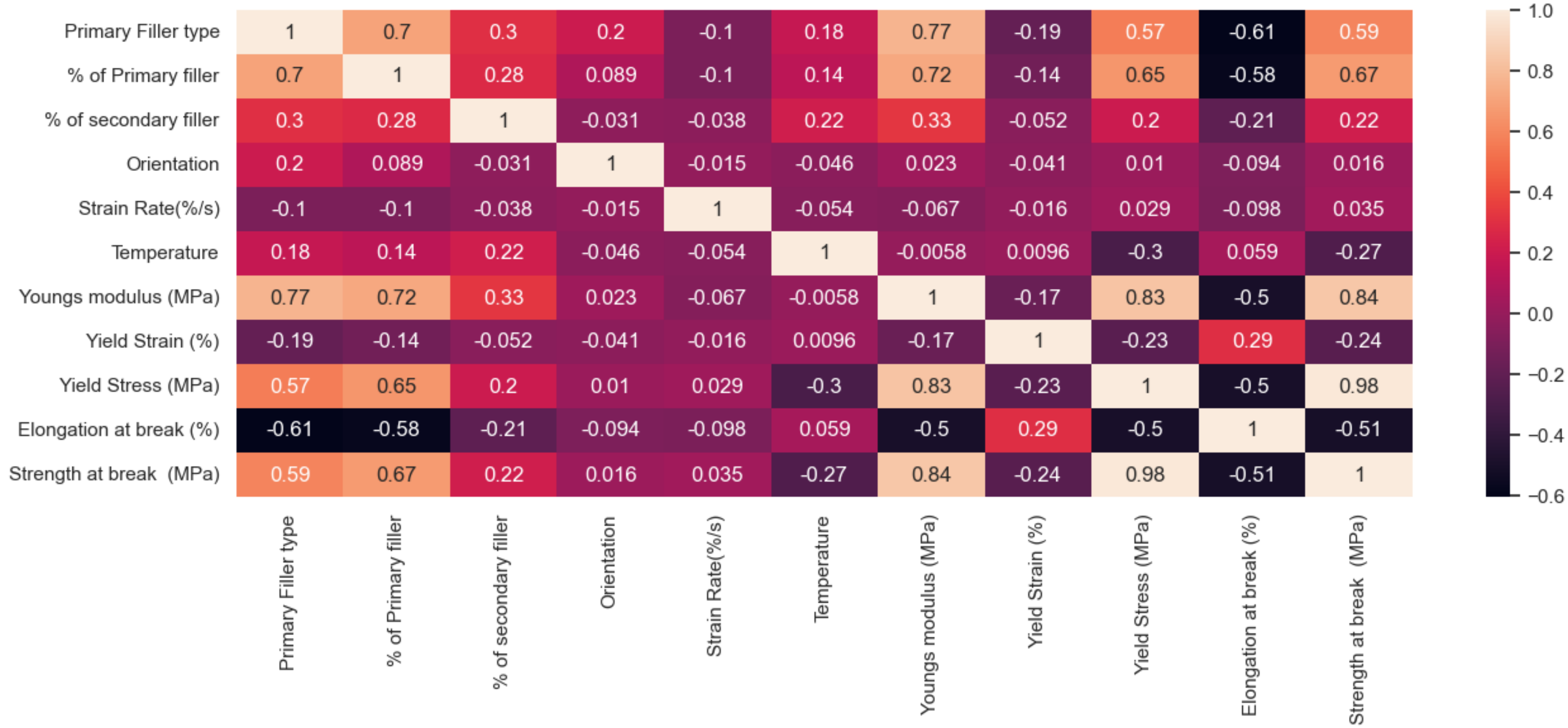
In [31]:
```
sns.pairplot(df,corner=True)
```

Out[31]:
```
<seaborn.axisgrid.PairGrid at 0x193cabacdc0>
```


% of Primary filler
Secondary filler type
% of secondary filler
Orientation
Strain Rate(%/s)
Temperature
Youngs modulus (MPa)
Yield Strain (%)
Yield Stress (MPa)



In [32]:
```
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True)
plt.show()
```




Primary filler has 0.77 correlation with youngs modulus

Youngs modulus and Strength at break have a correlaton of 0.84

Yield stress and Strength at break have of correlation of 0.99

## Data preparation for modeling

**Label encoding the data-Converting labels/words into numeric form**

In [33]:
```
le = LabelEncoder()
df['Polymer Types']=le.fit_transform(df['Polymer Types'])
df['Secondary filler type']=le.fit_transform(df['Secondary filler type'])
df['Quality']=le.fit_transform(df['Quality'])
```

In [34]:
```
# Checking the dataframe after the conversion
df.head()
```

Out[34]:

| | Polymer Types | Primary Filler type | % of Primary filler | Secondary filler type | % of secondary filler | Orientation | Strain Rate(%/s) | Temperature | Youngs modulus (MPa) | Yield Strain (%) | Yield Stress (MPa) | Elongation at break (%) | Strength at break (MPa) | Quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 52.0 | 4870.0 | 6.6 | 89.6 | 7.4 | 88.9 | 1 |
| **2** | 5 | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 107.0 | 4190.0 | 7.3 | 70.0 | 7.7 | 69.6 | 1 |
| **3** | 5 | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 23.0 | 7270.0 | 4.0 | 118.0 | 5.9 | 117.0 | 1 |
| **4** | 5 | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 121.0 | 3870.0 | 10.0 | 77.6 | 10.0 | 77.6 | 1 |
| **5** | 5 | 1.0 | 6.0 | 0 | 0.0 | 0.0 | 0.0833 | 107.0 | 6530.0 | 10.1 | 84.1 | 10.1 | 84.1 | 1 |

***Splitting the data for regression and classification***

In [35]:
```
#Splitting the data for regression and classification
X_regression=df.iloc[:,0:8]
y_regression=df.iloc[:,8]
```

In [36]:
```
y_regression.head()
```

Out[36]:
```
1    4870.0
2    4190.0
3    7270.0
4    3870.0
5    6530.0
Name: Youngs modulus (MPa), dtype: float64
```

In [37]:
```
X_classification=df.iloc[:,8:13]
y_classification=df['Quality']
```

**Testing data for multi-collierity**

In [38]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['Features'] = X_regression.columns
vif['VIF'] = [variance_inflation_factor(X_regression.values, i) for i in range(X_regression.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[38]:

| | Features | VIF |
|---|---|---|
| 3 | Secondary filler type | 14.36 |
| 4 | % of secondary filler | 14.27 |
| 1 | Primary Filler type | 3.39 |
| 2 | % of Primary filler | 3.10 |
| 7 | Temperature | 1.87 |
| 0 | Polymer Types | 1.69 |
| 5 | Orientation | 1.08 |
| 6 | Strain Rate(%/s) | 1.03 |

***% of secondary filler have high inflation factor. So dropping and checking whether VIF reduces***

In [39]:
```
X_regression.drop(columns=['% of secondary filler'],inplace=True)
```

In [40]:
```
vif = pd.DataFrame()
vif['Features'] = X_regression.columns
vif['VIF'] = [variance_inflation_factor(X_regression.values, i) for i in range(X_regression.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[40]:

| | Features | VIF |
|---|---|---|
| 1 | Primary Filler type | 3.36 |
| 2 | % of Primary filler | 3.04 |
| 6 | Temperature | 1.86 |
| 0 | Polymer Types | 1.67 |
| 3 | Secondary filler type | 1.27 |
| 4 | Orientation | 1.08 |
| 5 | Strain Rate(%/s) | 1.03 |

***All vif has reduced to below 5***

In [41]:
```
X_regression= add_constant(X_regression)  #Statmodels default is without intercept, to add intercept we need to add constant
X_train, X_test, y_train, y_test = train_test_split(X_regression, y_regression, test_size=0.2)
```

```
C:\Users\gokul\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: In a future version of pandas all arguments of concat except for the argument 'obj
s' will be keyword-only.
  x = pd.concat(x[::order], 1)
```

In [42]:
```
print(X_train.shape,X_test.shape)
```

```
(299, 8) (75, 8)
```

In [43]:
```
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

# Modelling-Regression

## Linear regression

In [44]:
```
model1 = LinearRegression()
```

In [45]:
```
model1.fit(X_train,y_train)
y_hat1 = model1.predict(X_test)
r2_train=model1.score(X_train,y_train)
r2_test=model1.score(X_test,y_test)
adj_r2_train=1-(1-r2_train)*(X_train.shape[0]-1)/(X_train.shape[0]-X_train.shape[1]-1)
adj_r2_test=1-(1-r2_test)*(X_train.shape[0]-1)/(X_train.shape[0]-X_train.shape[1]-1)
mae=mean_absolute_error(y_hat1,y_test)
mse=mean_squared_error(y_hat1,y_test)
```

In [46]:
```
print("R2 score for training data ",r2_train)
print("R2 score for testing data ",r2_test)
print("Adjusted R2 score for training data ",adj_r2_train)
print("Adjusted R2 score for test data ",adj_r2_test )
print('Mean Absolute Error ',mae )
print('Mean Squard Error', mse)
```

```
R2 score for training data  0.7125838361808196
R2 score for testing data  0.7685602915079257
Adjusted R2 score for training data  0.7046551144202904
Adjusted R2 score for test data  0.7621757478253857
Mean Absolute Error  1492.8181318229028
Mean Squard Error 7060735.649583005
```

In [47]:
```
w0 = model1.intercept_
coff = pd.DataFrame()
coff['Features'] = X_regression.columns
coff['cofficient'] = model1.coef_
print('intercept:', w0)
print()
print(coff)
```

```
intercept: 5882.294949832776

                Features   cofficient
0                  const     0.000000
1          Polymer Types  1073.710103
2    Primary Filler type  3772.849048
3    % of Primary filler  2072.090637
4  Secondary filler type   715.072872
5            Orientation  -860.346529
6       Strain Rate(%/s)    85.572389
7            Temperature  -958.472368
```

***The bias is really high for this model but the varience does not exist. The model is not capturing complexity of the data. (Underfitting)***

***Going for polynomial regression***

## Polynomical regression

In [48]:
```
poly = PolynomialFeatures(3)
X_train1 = poly.fit(X_train)
X_train1 = poly.transform(X_train)
X_test1 = poly.transform(X_test)
model2 = LinearRegression()
model2.fit(X_train1,y_train)
y_hat1 = model2.predict(X_test1)
r2_train=model2.score(X_train1,y_train)
r2_test=model2.score(X_test1,y_test)
adj_r2_train=1-(1-r2_train)*(X_train1.shape[0]-1)/(X_train1.shape[0]-X_train1.shape[1]-1)
adj_r2_test=1-(1-r2_test)*(X_train1.shape[0]-1)/(X_train1.shape[0]-X_train1.shape[1]-1)
mae=mean_absolute_error(y_hat1,y_test)
mse=mean_squared_error(y_hat1,y_test)

print("R2 score for training data ",r2_train)
print("R2 score for testing data ",r2_test)
print("Adjusted R2 score for training data ",adj_r2_train)
print("Adjusted R2 score for test data ",adj_r2_test )
print('Mean Absolute Error ',mae )
print('Mean Squard Error', mse)
```

```
R2 score for training data  0.9589405797501032
R2 score for testing data  -116822918473.61108
Adjusted R2 score for training data  0.9080022012445921
Adjusted R2 score for test data  -261753606806.7752
Mean Absolute Error  222255702.7046875
Mean Squard Error 3.5640199797013775e+18
```

***Here polynomical factor were added to the model. The bias has reduced but varinece has increased. (Overfitting)***

***Tried out different iteration with different order of polynomial and 3 had given the best result***

## Lasso Regression

***L1 Regularisation***

In [49]:
```
model3 = Lasso(alpha=0.001)
model3.fit(X_train1,y_train)
y_hat = model3.predict(X_test1)
r2_train=model3.score(X_train1,y_train)
r2_test=model3.score(X_test1,y_test)
adj_r2_test=1-(1-r2_test)*(X_train1.shape[0]-1)/(X_train1.shape[0]-X_train1.shape[1]-1)
mae=mean_absolute_error(y_hat,y_test)
mse=mean_squared_error(y_hat,y_test)
```

```
C:\Users\gokul\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the num
ber of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.527e+08, tolerance: 1.039e+06
  model = cd_fast.enet_coordinate_descent(
```

In [50]:
```
print("R2 score for training data ",r2_train)
print("R2 score for testing data ",r2_test)
print("Adjusted R2 score for training data ",adj_r2_train)
print("Adjusted R2 score for test data ",adj_r2_test )
print('Mean Absolute Error ',mae )
print('Mean Squard Error', mse)
```

```
R2 score for training data  0.9513430275509782
R2 score for testing data  0.8759577932326935
Adjusted R2 score for training data  0.9080022012445921
Adjusted R2 score for test data  0.7220708449875388
Mean Absolute Error  1011.2924526486329
Mean Squard Error 3784265.185439689
```

***Varience has reduced but still exist. Going of for L2 regularisation***

## Ridge Regression

In [51]:
```
model4 = Ridge()
model4.fit(X_train1,y_train)
y_hat = model4.predict(X_test1)
r2_train=model4.score(X_train1,y_train)
r2_test=model4.score(X_test1,y_test)
adj_r2_test=1-(1-r2_test)*(X_train1.shape[0]-1)/(X_train1.shape[0]-X_train1.shape[1]-1)
mae=mean_absolute_error(y_hat,y_test)
mse=mean_squared_error(y_hat,y_test)
```

In [52]:
```
print("R2 score for training data ",r2_train)
print("R2 score for testing data ",r2_test)
print("Adjusted R2 score for training data ",adj_r2_train)
print("Adjusted R2 score for test data ",adj_r2_test )
print('Mean Absolute Error ',mae )
print('Mean Squard Error', mse)
```

```
R2 score for training data  0.9508036795560211
R2 score for testing data  0.8766399405963757
Adjusted R2 score for training data  0.9080022012445921
Adjusted R2 score for test data  0.7235992653963907
Mean Absolute Error  1024.2407661550856
Mean Squard Error 3763454.313180984
```

**Produced almost similar result with little bit improvement. This can be the final model**

In [53]:
```
model4.coef_.max()
```

Out[53]:
```
2023.8381325362604
```

In [54]:
```
round(model4.coef_.argmax()/3)
```

Out[54]:
```
6
```

***Young's modulus is heavy dependent on Strain Rate(%/s) variable***

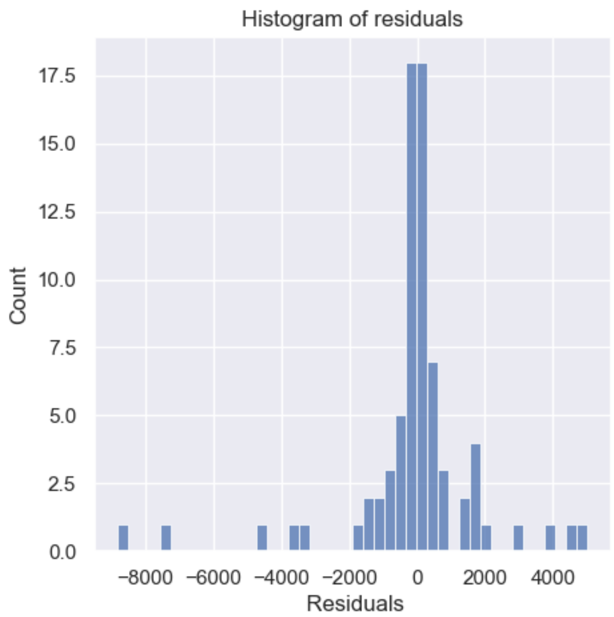
## Check for assumptions for Linear Regression

### Normality of residuals

In [55]:
```
errors = y_hat - y_test
```

In [56]:
```
sns.histplot(errors)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
```

Out[56]: Text(0.5, 1.0, 'Histogram of residuals')



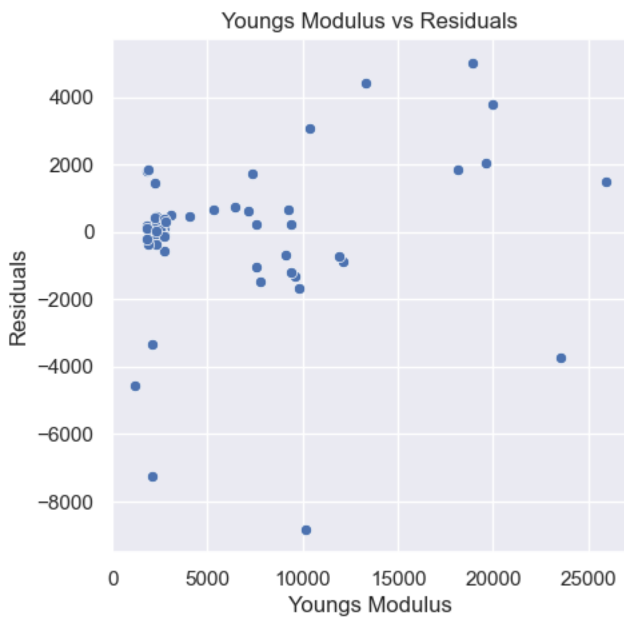

Errors are normally distributed

**Check for Heteroskedasticity and Linearity of variable**

In [57]:
```
sns.scatterplot(y_hat,errors)
plt.xlabel("Youngs Modulus")
plt.ylabel("Residuals")
plt.title("Youngs Modulus vs Residuals")
```

```
C:\Users\gokul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid pos
itional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[57]:
```
Text(0.5, 1.0, 'Youngs Modulus vs Residuals')
```




No Heteroskedasticity as there is not specific pattern

Point are equally distributed on the both side of 0.

In [ ]:

**Lasso Regression is the final model with accuracy of 89% and with least bias and varience**

# Modelling-Classification

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X_classification, y_classification, test_size=0.2)
# X_train=np.array(X_train).reshape(-1,)
# X_test=np.array(X_test).reshape(-1,)
# X_train.shape
```

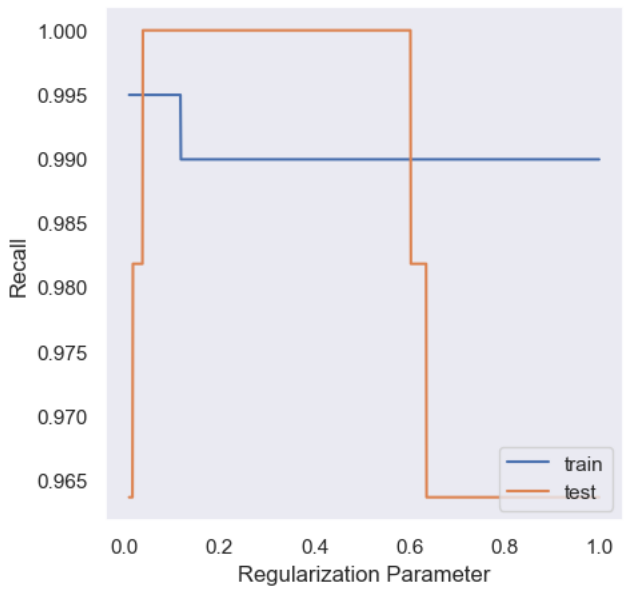
## Logistice Regression

In [59]:
```
sc1 = StandardScaler()
sc1.fit(X_train)
X_train = sc1.transform(X_train)
X_test = sc1.transform(X_test)
```

In [60]:
```
train_scores = []
test_scores = []
for la in np.arange(0.01, 1.0, 0.001):
    model5 = LogisticRegression(C=1/la)
    model5.fit(X_train, y_train)
    y_hat_train=model5.predict(X_train)
    y_hat_test=model5.predict(X_test)
    train_score=recall_score(y_train,y_hat_train)
    test_score=recall_score(y_test,y_hat_test)
    train_scores.append(train_score)
    test_scores.append(test_score)
```

In [61]:
```
plt.figure()
plt.plot(list(np.arange(0.01, 1.0, 0.001)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 1.0, 0.001)), test_scores, label="test")
plt.legend(loc='lower right')

plt.xlabel("Regularization Parameter")
plt.ylabel("Recall")
plt.grid()
plt.show()
```



In [62]:
```
# Best regulariation factor
la=0.01+(np.argmax(train_scores)+1)*0.001
```

In [63]:
```
#best model
model5 = LogisticRegression(C=1/la)
```

In [64]:
```
model5.fit(X_train, y_train)
```

Out[64]:
```
LogisticRegression(C=90.90909090909092)
```

In [65]:
```
y_hat_train=model5.predict(X_train)
y_hat_test=model5.predict(X_test)
```

In [66]:
```
model5.coef_
```

Out[66]:
```
array([[ 0.27179087, -0.0542791 , -2.07931657, -0.06917588,  1.96120284]])
```

Strength at break (MPa) has high correlation with quality

In [67]:
```
print('f1 score for train data is ', f1_score(y_train, y_hat_train))
print('Best R2 score for train data is ', max(train_scores))
```

```
f1 score for train data is  0.8065173116089613
Best R2 score for train data is  0.9949748743718593
```

In [68]:
```
print('f1 score for test data is ',f1_score(y_test, y_hat_test))
print('Best R2 score for test data is ', max(test_scores))
```

```
f1 score for test data is  0.828125
Best R2 score for test data is  1.0
```

## Logistice Regression using polynominial feature

In [69]:
```
poly = PolynomialFeatures(4)
X_train2 = poly.fit(X_train)
X_train2 = poly.transform(X_train)
X_test2 = poly.transform(X_test)
```

In [70]:
```
train_scores = []
test_scores = []
for la in np.arange(0.01, 1.0, 0.01):
    model5 = LogisticRegression(C=1/la)
    model5.fit(X_train2, y_train)
    y_hat_train=model5.predict(X_train2)
    y_hat_test=model5.predict(X_test2)
    train_score=recall_score(y_train,y_hat_train)
    test_score=recall_score(y_test,y_hat_test)
    train_scores.append(train_score)
    test_scores.append(test_score)
```

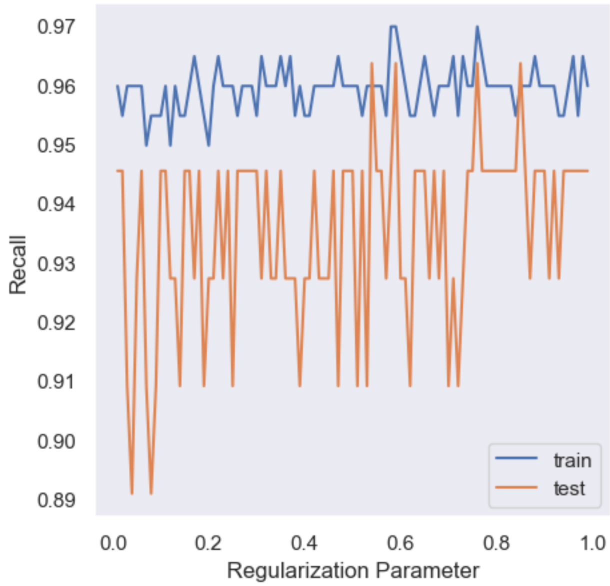
```
C:\Users\gokul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\gokul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\gokul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

In [71]:
```
plt.figure()
plt.plot(list(np.arange(0.01, 1.0, 0.01)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 1.0, 0.01)), test_scores, label="test")
plt.legend(loc='lower right')

plt.xlabel("Regularization Parameter")
plt.ylabel("Recall")
plt.grid()
plt.show()
```




In [72]:
```
# Best regulirisation factor
la=0.01+(np.argmax(train_scores)+1)*0.001
```

In [73]:
```
model6 = LogisticRegression(C=1/la)
model6.fit(X_train2, y_train)
y_hat_train=model5.predict(X_train2)
y_hat_test=model5.predict(X_test2)
```

```
C:\Users\gokul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [74]:
```
print('f1 score for train data is ', f1_score(y_train, y_hat_train))
print('Best R2 score for train data is ', max(train_scores))
```

```
f1 score for train data is  0.8179871520342612
Best R2 score for train data is  0.9698492462311558
```

In [75]:
```
print('f1 score for test data is ',f1_score(y_test, y_hat_test))
print('Best R2 score for test data is ', max(test_scores))
```

```
f1 score for test data is  0.8387096774193549
Best R2 score for test data is  0.9636363636363636
```

## Decision Tree

In [76]:
```
model6 = tree.DecisionTreeClassifier()
```

In [77]:
```
model6.fit(X_train, y_train)
```

Out[77]:
```
DecisionTreeClassifier()
```

In [78]:
```
y_hat_train=model6.predict(X_train)
y_hat_test=model6.predict(X_test)
train_score=recall_score(y_train,y_hat_train)
test_score=recall_score(y_test,y_hat_test)
```

In [79]:
```
print('recall for decision tree on training data = ',train_score)
```

```
recall for decision tree on training data =  0.9899497487437185
```

In [80]:
```
print('recall for decision tree on test data = ',test_score)
```

```
recall for decision tree on test data =  0.7454545454545455
```

Recall is really lower for test data

## Random Forest

In [81]:
```
model7=RandomForestRegressor(n_estimators=100)
model7.fit(X_train, y_train)
```

Out[81]:
```
RandomForestRegressor()
```

In [82]:
```
y_hat_train=model7.predict(X_train)
y_hat_test=model7.predict(X_test)


y_hat_train=y_hat_train.round()
y_hat_test=y_hat_test.round()

train_score=recall_score(y_train,y_hat_train)
test_score=recall_score(y_test,y_hat_test)
```

In [83]:
```
print('recall for Random Forest on training data = ',train_score)
```

```
recall for Random Forest on training data =  0.9949748743718593
```

In [84]:
```
print('recall for Random Forest on test data = ',test_score)
```

```
recall for Random Forest on test data =  0.8909090909090909
```

Recall is lower than logistic regression

**Logistic Regression is the final model with best recall and with best f1 score on test data (0.85)**

# Bussiness Insights

1) Ridge Regression is the best model for predicting Youngs Modulus with an accuracy of 89% and with the least bias and variance

2) Logistic Regression is the final model with the best recall and with the best f1 score on test data (0.85)

3) Yield Stress and Strength at break are linearly correlated

4) Youngs modulus and strength at break are linearly correlated

5) Larger majority of data is for good-quality items. So data is imbalanced.

6) Since data is imbalanced we can not go for accuracy. Classifying a bad-quality item as good quality can affect the reputation of the company. So a model with more False Negatives is to be penalized. We need to optimize the algorithm for best Recall.

7) Multi-collinearity existed between the Secondary filler type and % of secondary filler. So, % of secondary filler (for linear regression Multi-collinearity should not exist.

8) Youngs modulus has high correlation with Strain Rate(%/s), Primaray filler and % primary filler.

9) Strength at break (MPa) have a high correlation with the quality of the item. Strength has a high correlation with Yield stress and youngs modulus

# Recommendation

1) Use Ridge Regression with polynomial features of order 3 for predicting youngs modulus

2) Use Logistic regression to predict the quality of the product

3) Strength at break (MPa) should be high for best-quality items

In [ ]:
```

```