# PROJECT

# PROBLEM STATEMENT : To perform an analytics report on 100 years of Rainfall data

# Importing packages

In [24]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

# Reading the data

In [25]:
```
df=pd.read_csv(r"C:\Users\Lenovo\OneDrive\Desktop\Data Sets\rainfall in india 1
df
```

Out[25]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN & NICOBAR ISLANDS | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 |
| **1** | ANDAMAN & NICOBAR ISLANDS | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 |
| **2** | ANDAMAN & NICOBAR ISLANDS | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 |
| **3** | ANDAMAN & NICOBAR ISLANDS | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 |
| **4** | ANDAMAN & NICOBAR ISLANDS | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4111** | LAKSHADWEEP | 2011 | 5.1 | 2.8 | 3.1 | 85.9 | 107.2 | 153.6 | 350.2 | 254.0 | 255.2 | 117.4 |
| **4112** | LAKSHADWEEP | 2012 | 19.2 | 0.1 | 1.6 | 76.8 | 21.2 | 327.0 | 231.5 | 381.2 | 179.8 | 145.9 |
| **4113** | LAKSHADWEEP | 2013 | 26.2 | 34.4 | 37.5 | 5.3 | 88.3 | 426.2 | 296.4 | 154.4 | 180.0 | 72.8 |
| **4114** | LAKSHADWEEP | 2014 | 53.2 | 16.1 | 4.4 | 14.9 | 57.4 | 244.1 | 116.1 | 466.1 | 132.2 | 169.2 |
| **4115** | LAKSHADWEEP | 2015 | 2.2 | 0.5 | 3.7 | 87.1 | 133.1 | 296.6 | 257.5 | 146.4 | 160.4 | 165.4 |

4116 rows × 19 columns

# Data cleaning and preprocessing

In [26]:
```
df.shape
```

Out[26]: (4116, 19)

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4116 entries, 0 to 4115
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   SUBDIVISION  4116 non-null   object 
 1   YEAR         4116 non-null   int64  
 2   JAN          4112 non-null   float64
 3   FEB          4113 non-null   float64
 4   MAR          4110 non-null   float64
 5   APR          4112 non-null   float64
 6   MAY          4113 non-null   float64
 7   JUN          4111 non-null   float64
 8   JUL          4109 non-null   float64
 9   AUG          4112 non-null   float64
 10  SEP          4110 non-null   float64
 11  OCT          4109 non-null   float64
 12  NOV          4105 non-null   float64
 13  DEC          4106 non-null   float64
 14  ANNUAL       4090 non-null   float64
 15  Jan-Feb      4110 non-null   float64
 16  Mar-May      4107 non-null   float64
 17  Jun-Sep      4106 non-null   float64
 18  Oct-Dec      4103 non-null   float64
dtypes: float64(17), int64(1), object(1)
memory usage: 611.1+ KB
```

In [28]:
```
df.describe()
```

Out[28]:

| | YEAR | JAN | FEB | MAR | APR | MAY | JUN |
|---|---|---|---|---|---|---|---|
| **count** | 4116.000000 | 4112.000000 | 4113.000000 | 4110.000000 | 4112.000000 | 4113.000000 | 4111.00000 |
| **mean** | 1958.218659 | 18.957320 | 21.805325 | 27.359197 | 43.127432 | 85.745417 | 230.23444 |
| **std** | 33.140898 | 33.585371 | 35.909488 | 46.959424 | 67.831168 | 123.234904 | 234.71075 |
| **min** | 1901.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.40000 |
| **25%** | 1930.000000 | 0.600000 | 0.600000 | 1.000000 | 3.000000 | 8.600000 | 70.35000 |
| **50%** | 1958.000000 | 6.000000 | 6.700000 | 7.800000 | 15.700000 | 36.600000 | 138.70000 |
| **75%** | 1987.000000 | 22.200000 | 26.800000 | 31.300000 | 49.950000 | 97.200000 | 305.15000 |
| **max** | 2015.000000 | 583.700000 | 403.500000 | 605.600000 | 595.100000 | 1168.600000 | 1609.90000 |

In [29]:
```
df.head()
```

Out[29]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN & NICOBAR ISLANDS | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 | 558.2 |
| **1** | ANDAMAN & NICOBAR ISLANDS | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 | 359.0 |
| **2** | ANDAMAN & NICOBAR ISLANDS | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 | 284.4 |
| **3** | ANDAMAN & NICOBAR ISLANDS | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 | 308.7 |
| **4** | ANDAMAN & NICOBAR ISLANDS | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 | 25.4 |

In [30]:
```
df.tail()
```

Out[30]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4111** | LAKSHADWEEP | 2011 | 5.1 | 2.8 | 3.1 | 85.9 | 107.2 | 153.6 | 350.2 | 254.0 | 255.2 | 117.4 | 18 |
| **4112** | LAKSHADWEEP | 2012 | 19.2 | 0.1 | 1.6 | 76.8 | 21.2 | 327.0 | 231.5 | 381.2 | 179.8 | 145.9 | 1 |
| **4113** | LAKSHADWEEP | 2013 | 26.2 | 34.4 | 37.5 | 5.3 | 88.3 | 426.2 | 296.4 | 154.4 | 180.0 | 72.8 | 7 |
| **4114** | LAKSHADWEEP | 2014 | 53.2 | 16.1 | 4.4 | 14.9 | 57.4 | 244.1 | 116.1 | 466.1 | 132.2 | 169.2 | 5 |
| **4115** | LAKSHADWEEP | 2015 | 2.2 | 0.5 | 3.7 | 87.1 | 133.1 | 296.6 | 257.5 | 146.4 | 160.4 | 165.4 | 23 |

In [31]:
```
df["SUBDIVISION"].value_counts()
```

Out[31]:
```
SUBDIVISION
WEST MADHYA PRADESH                   115
EAST RAJASTHAN                        115
COASTAL KARNATAKA                     115
TAMIL NADU                            115
RAYALSEEMA                            115
TELANGANA                             115
COASTAL ANDHRA PRADESH                115
CHHATTISGARH                          115
VIDARBHA                              115
MATATHWADA                            115
MADHYA MAHARASHTRA                    115
KONKAN & GOA                          115
SAURASHTRA & KUTCH                    115
GUJARAT REGION                        115
EAST MADHYA PRADESH                   115
KERALA                                115
WEST RAJASTHAN                        115
SOUTH INTERIOR KARNATAKA              115
JAMMU & KASHMIR                       115
HIMACHAL PRADESH                      115
PUNJAB                                115
HARYANA DELHI & CHANDIGARH            115
UTTARAKHAND                           115
WEST UTTAR PRADESH                    115
EAST UTTAR PRADESH                    115
BIHAR                                 115
JHARKHAND                             115
ORISSA                                115
GANGETIC WEST BENGAL                  115
SUB HIMALAYAN WEST BENGAL & SIKKIM    115
NAGA MANI MIZO TRIPURA                115
ASSAM & MEGHALAYA                     115
NORTH INTERIOR KARNATAKA              115
LAKSHADWEEP                           114
ANDAMAN & NICOBAR ISLANDS             110
ARUNACHAL PRADESH                      97
Name: count, dtype: int64
```

In [32]:
```
states={"SUBDIVISION":{
"WEST MADHYA PRADESH":1,
"EAST RAJASTHAN":2,
"COASTAL KARNATAKA":3,
"TAMIL NADU":4,
"RAYALSEEMA":5,
"TELANGANA":6,
"COASTAL ANDHRA PRADESH":7,
"CHHATTISGARH":8,
"VIDARBHA":9,
"MATATHWADA":10,
"MADHYA MAHARASHTRA":11,
"KONKAN & GOA":12,
"SAURASHTRA & KUTCH":13,
"GUJARAT REGION":14,
"EAST MADHYA PRADESH":15,
"KERALA":16,
"WEST RAJASTHAN":17,
"SOUTH INTERIOR KARNATAKA":18,
"JAMMU & KASHMIR":19,
"HIMACHAL PRADESH":20,
"PUNJAB":21,
"HARYANA DELHI & CHANDIGARH":22,
"UTTARAKHAND":23,
"WEST UTTAR PRADESH":24,
"EAST UTTAR PRADESH":25,
"BIHAR":26,
"JHARKHAND":27,
"ORISSA":28,
"GANGETIC WEST BENGAL":29,
"SUB HIMALAYAN WEST BENGAL & SIKKIM":30,
"NAGA MANI MIZO TRIPURA":31,
"ASSAM & MEGHALAYA":32,
"NORTH INTERIOR KARNATAKA":33,
"LAKSHADWEEP":34,
"ANDAMAN & NICOBAR ISLANDS":35,
"ARUNACHAL PRADESH":36}}
df=df.replace(states)
df
```

Out[32]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 35 | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 | 55 |
| **1** | 35 | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 | 35 |
| **2** | 35 | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 | 28 |
| **3** | 35 | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 | 30 |
| **4** | 35 | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **4111** | 34 | 2011 | 5.1 | 2.8 | 3.1 | 85.9 | 107.2 | 153.6 | 350.2 | 254.0 | 255.2 | 117.4 | 18 |
| **4112** | 34 | 2012 | 19.2 | 0.1 | 1.6 | 76.8 | 21.2 | 327.0 | 231.5 | 381.2 | 179.8 | 145.9 | 1 |
| **4113** | 34 | 2013 | 26.2 | 34.4 | 37.5 | 5.3 | 88.3 | 426.2 | 296.4 | 154.4 | 180.0 | 72.8 | 7 |
| **4114** | 34 | 2014 | 53.2 | 16.1 | 4.4 | 14.9 | 57.4 | 244.1 | 116.1 | 466.1 | 132.2 | 169.2 | 5 |
| **4115** | 34 | 2015 | 2.2 | 0.5 | 3.7 | 87.1 | 133.1 | 296.6 | 257.5 | 146.4 | 160.4 | 165.4 | 23 |

4116 rows × 19 columns

In [33]:
```
df.isnull().sum()
```

Out[33]:
```
SUBDIVISION     0
YEAR            0
JAN             4
FEB             3
MAR             6
APR             4
MAY             3
JUN             5
JUL             7
AUG             4
SEP             6
OCT             7
NOV            11
DEC            10
ANNUAL         26
Jan-Feb         6
Mar-May         9
Jun-Sep        10
Oct-Dec        13
dtype: int64
```

In [34]:
```
df.fillna(method="ffill",inplace=True)
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['SUBDIVISION', 'YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JU
L',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'ANNUAL', 'Jan-Feb', 'Mar-May',
       'Jun-Sep', 'Oct-Dec'],
      dtype='object')
```

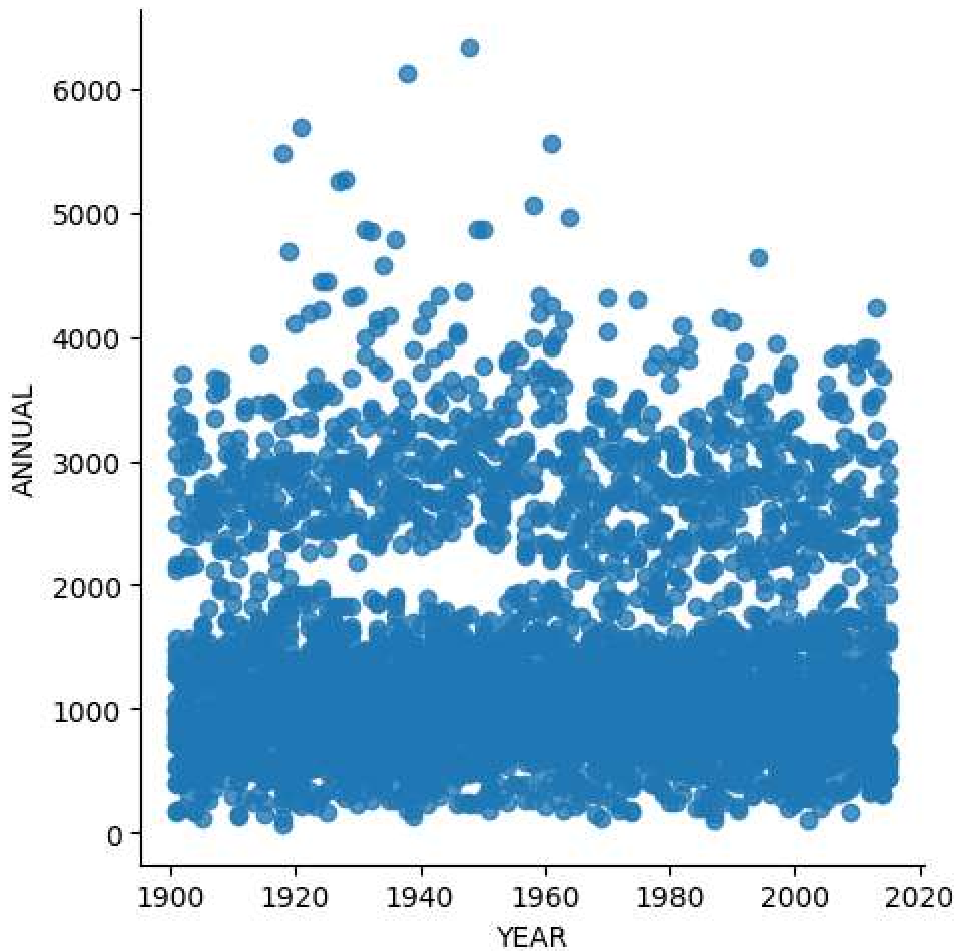
In [36]:
```
df.isnull().sum()
```

Out[36]:
```
SUBDIVISION    0
YEAR           0
JAN            0
FEB            0
MAR            0
APR            0
MAY            0
JUN            0
JUL            0
AUG            0
SEP            0
OCT            0
NOV            0
DEC            0
ANNUAL         0
Jan-Feb        0
Mar-May        0
Jun-Sep        0
Oct-Dec        0
dtype: int64
```

# Data Visualization

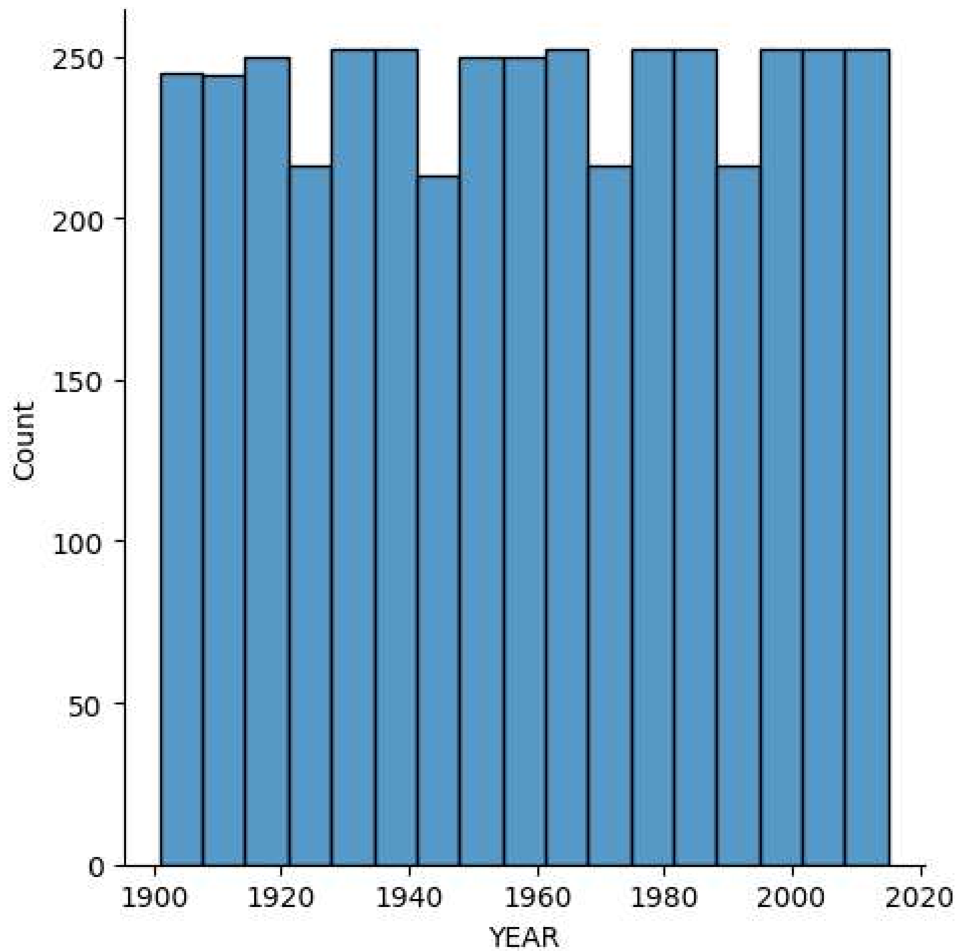
In [37]:
```
sns.lmplot(x="YEAR",y="ANNUAL",data=df,order=2,ci=None)
```

Out[37]: <seaborn.axisgrid.FacetGrid at 0x1cf10d4fb10>



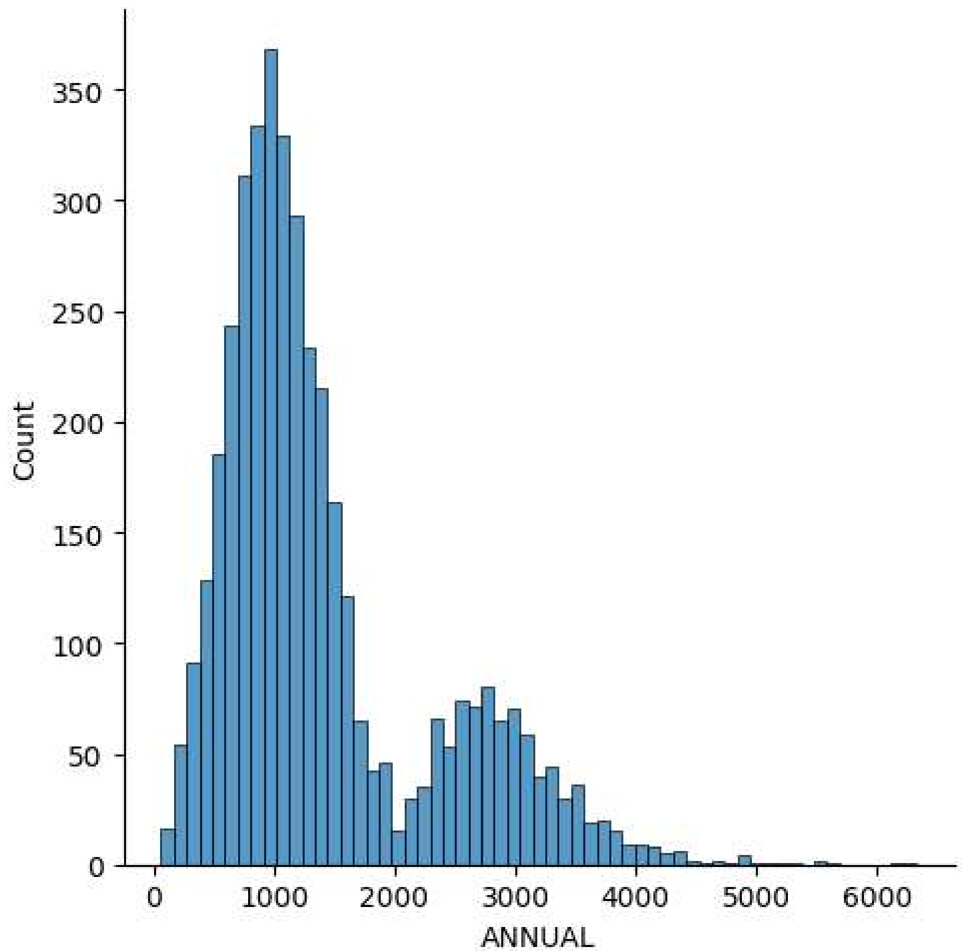
In [38]:
```
sns.displot(df['YEAR'])
```

Out[38]: <seaborn.axisgrid.FacetGrid at 0x1cf12f3e050>



In [39]:
```
sns.displot(df['ANNUAL'])
```

Out[39]:
```
<seaborn.axisgrid.FacetGrid at 0x1cf13131050>
```




# APPLYING LINEAR REGRESSION

In [40]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

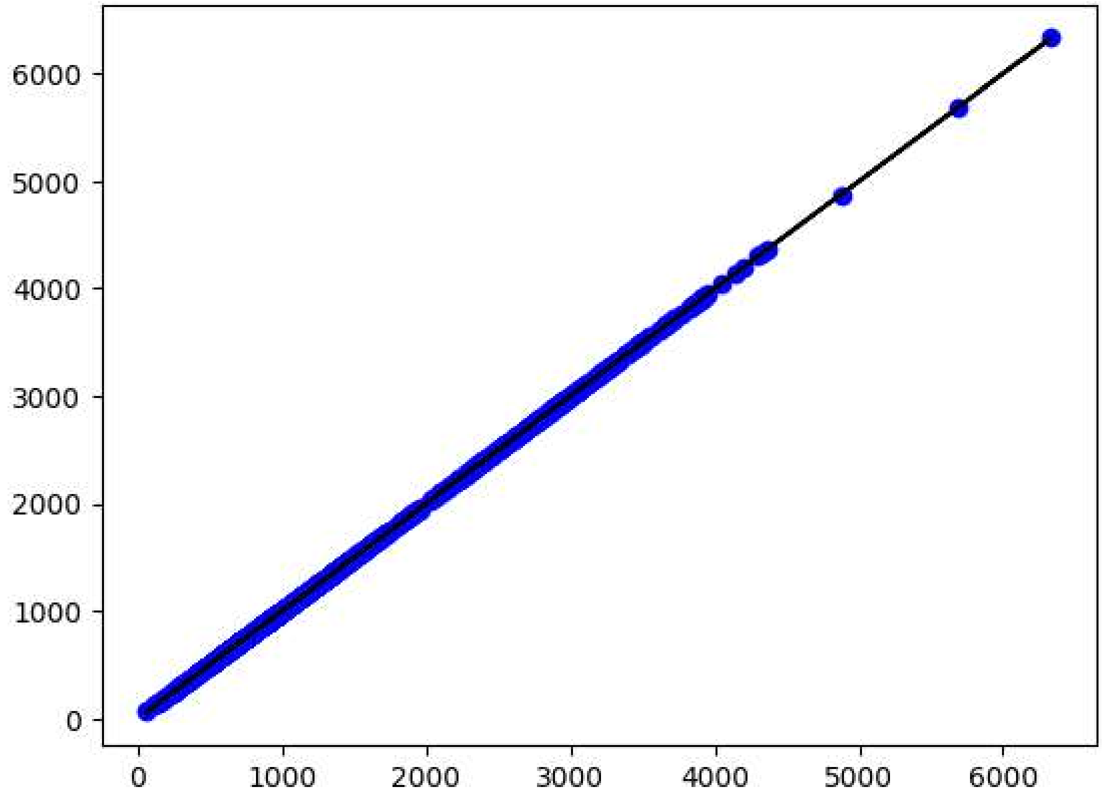
In [41]:
```
x=np.array(df['YEAR']).reshape(-1,1)
y=x=np.array(df['ANNUAL']).reshape(-1,1)
```

In [42]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30)
```

In [44]:
```
regr=LinearRegression()
regr.fit(x_train,y_train)
print(regr.score(x_train,y_train))
```

```
1.0
```

In [46]:
```
y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='blue')
plt.plot(x_test,y_pred,color='black')
plt.show()
```




# Since we got 100% accuracy we can say that it is not a best model.Because no model is 100% accurate.Now we are going to implement Logistic Regression

# Applying Logistic Regression

In [62]:
```
from sklearn.linear_model import LogisticRegression
```

In [63]:
```
lr=LogisticRegression()
```

In [74]:
```
x=np.array(df['ANNUAL']).reshape(-1,1)
y=np.array(df['SUBDIVISION']).reshape(-1,1)
```

In [77]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=4
```

In [78]:
```
lr.fit(x_train,y_train)
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\utils\validation.py:1143: DataConversionWarning: A column-vector y was p
assed when a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
rge (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://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

Out[78]:
```
▸ LogisticRegression
```

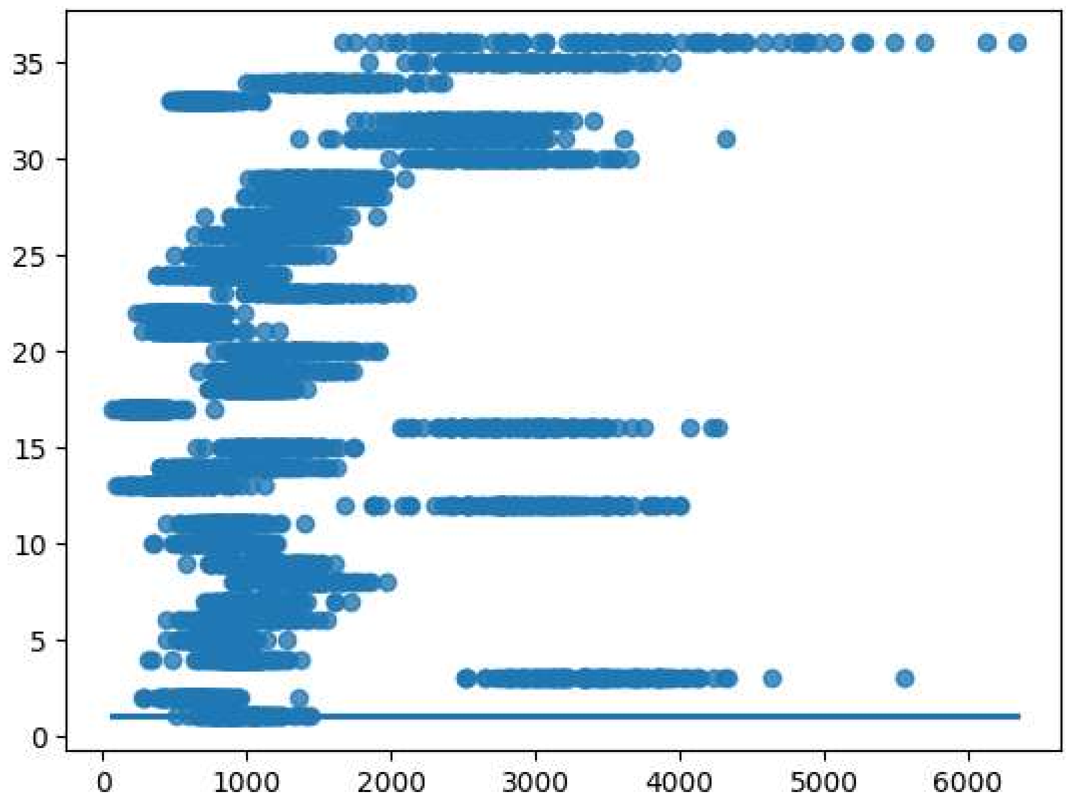
In [79]:
```
lr.score(x_test,y_test)
```

Out[79]:
```
0.1408906882591093
```

In [80]:
```
sns.regplot(x=x,y=y,data=df,logistic=True,ci=None)
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\sta
tsmodels\genmod\families\links.py:198: RuntimeWarning: overflow encountered i
n exp
  t = np.exp(-z)
```

Out[80]:
```
<Axes: >
```




# Here we did'nt got the good accuracy as well as graph also.Now we are going to implement clustering algorithm KMeans

# Applying Ridge Regression and Lasso Regression

In [81]:
```
from sklearn.linear_model import Lasso,Ridge
from sklearn.preprocessing import StandardScaler
```

In [82]:
```
features= df.columns[:2]
target= df.columns[:15]
```

In [83]:
```
x=np.array(df['YEAR']).reshape(-1,1)
y=np.array(df['ANNUAL']).reshape(-1,2)
```

In [84]:
```
x= df[features].values
y= df[target].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1
```

In [96]:
```
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X test is {}".format(x_test.shape))
```

```
The dimension of X_train is (2881, 2)
The dimension of X test is (1235, 2)
```

In [97]:
```
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

In [100]:
```
#Ridge Regression Model
ridgeReg=Ridge(alpha=10)

ridgeReg.fit(x_train,y_train)

#train and test score for ridge regression
train_score_ridge=ridgeReg.score(x_train,y_train)
test_score_ridge=ridgeReg.score(x_test,y_test)

print("The train score for lr model is {}".format(train_score_ridge))
print("The test score for lr model is {}".format(test_score_ridge))
```

```
The train score for lr model is 0.19734268994216755
The test score for lr model is 0.197092828044007
```

In [103]:
```
#lasso regression model
print("\nLasso Model: \n")
lasso=Lasso(alpha=10)
lasso.fit(x_train,y_train)
train_score_ls=lasso.score(x_train,y_train)
test_score_ls=lasso.score(x_test,y_test)

print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls))
```

```

Lasso Model: 

The train score for ls model is 0.11243395544012529
The test score for ls model is 0.11483605154265669
```

In [108]:
```
from sklearn.linear_model import RidgeCV

ridge_cv=RidgeCV(alphas=[0.0001,0.001,0.01,0.1,1,10]).fit(x_train,y_train)

print("The train score for ridge model is {}".format(ridge_cv.score(x_train,y_t
print("The train score for ridge model is {}".format(ridge_cv.score(x_test,y_te
```

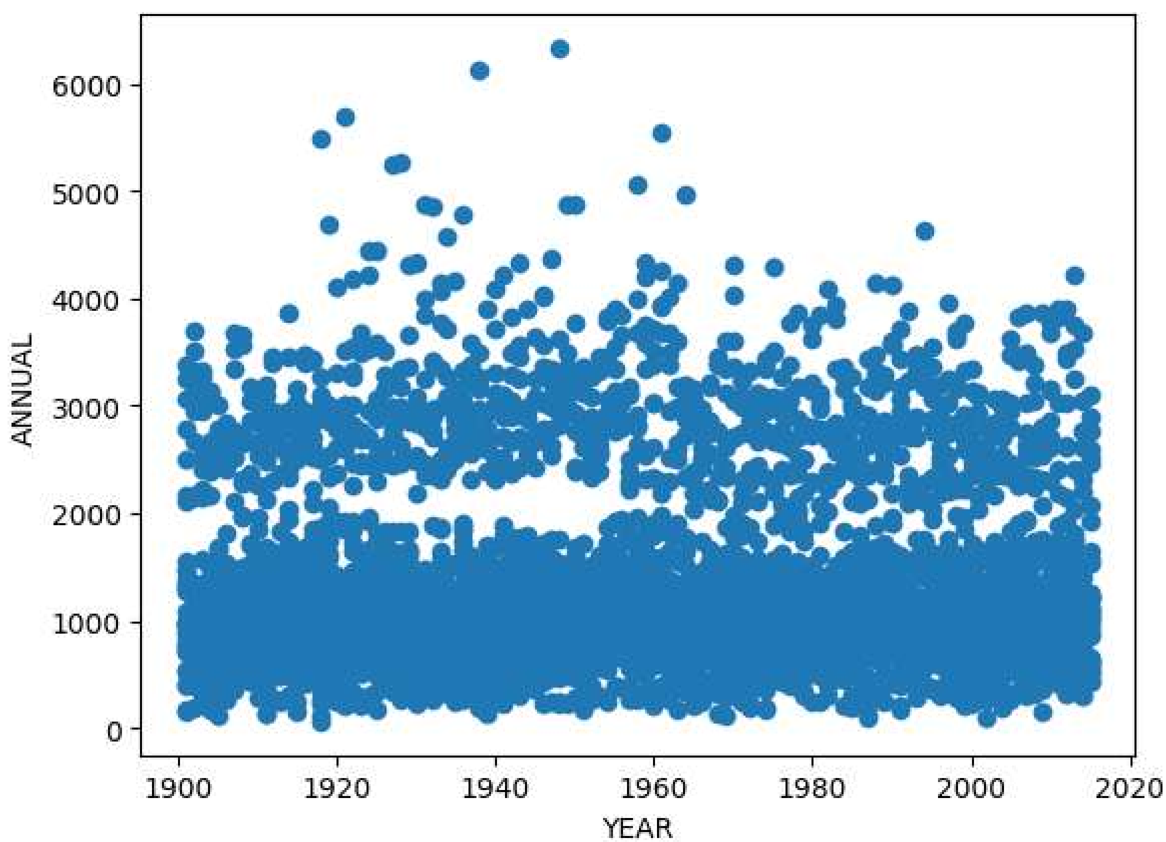
```
The train score for ridge model is 0.19734268994216667
The train score for ridge model is 0.1970928280440069
```

# Here we did'nt got the good accuracy.so now we are going to implement KMeans

# Applying KMeans

In [60]:
```
plt.scatter(df["YEAR"],df["ANNUAL"])
plt.xlabel("YEAR")
plt.ylabel("ANNUAL")
```

Out[60]: Text(0, 0.5, 'ANNUAL')



In [61]:
```
from sklearn.cluster import KMeans
```

In [41]:
```
km=KMeans()
km
```

Out[41]:
```
▾ KMeans
KMeans()
```

In [42]:
```
y_predicted=km.fit_predict(df[["YEAR","ANNUAL"]])
y_predicted
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
```

Out[42]:
```
array([3, 3, 1, ..., 7, 7, 7])
```

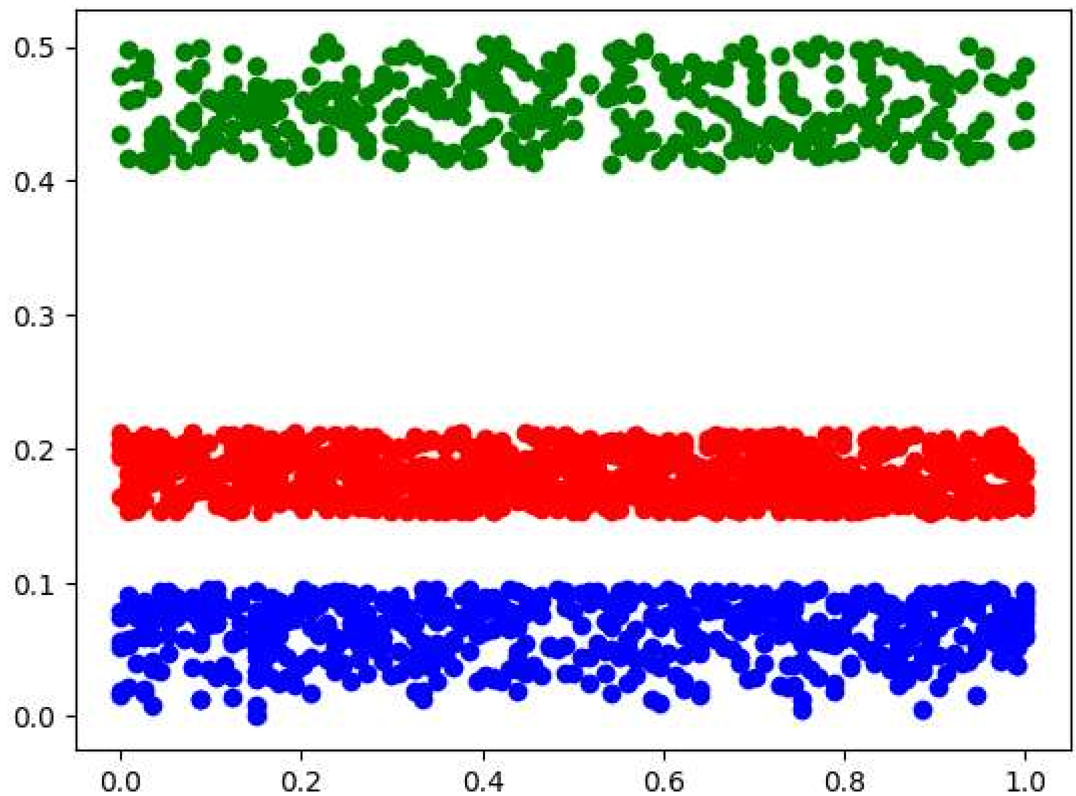
In [43]:
```
df["Cluster"]=y_predicted
df.head()
```

Out[43]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 35 | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 | 558.2 |
| **1** | 35 | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 | 359.0 |
| **2** | 35 | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 | 284.4 |
| **3** | 35 | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 | 308.7 |
| **4** | 35 | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 | 25.4 |

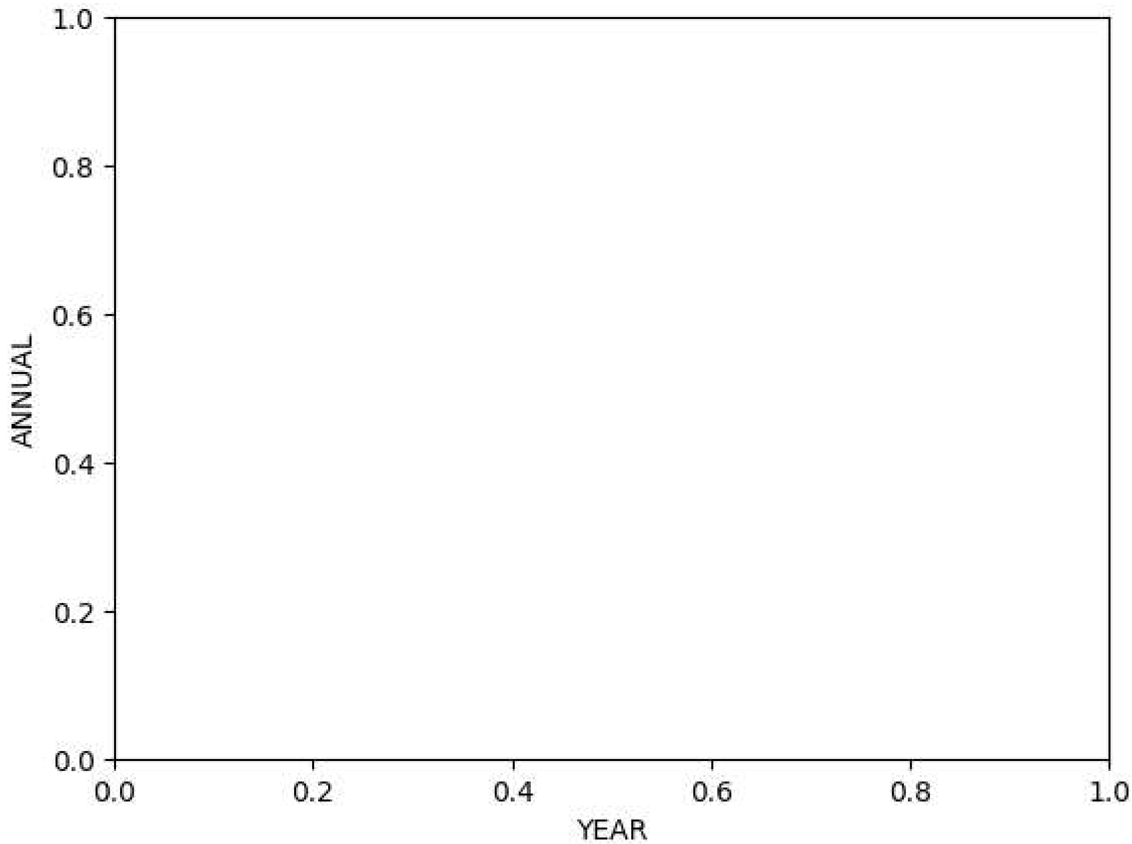
In [58]:
```
df1=df[df.Cluster==0]
df2=df[df.Cluster==1]
df3=df[df.Cluster==2]

plt.scatter(df1["YEAR"],df1["ANNUAL"],color="red")
plt.scatter(df2["YEAR"],df2["ANNUAL"],color="green")
plt.scatter(df3["YEAR"],df3["ANNUAL"],color="blue")
```

Out[58]: <matplotlib.collections.PathCollection at 0x1754edb1f10>



In [59]:
```
plt.xlabel("YEAR")
plt.ylabel("ANNUAL")
```

Out[59]:
```
Text(0, 0.5, 'ANNUAL')
```




In [60]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
```

In [61]:
```
scaler.fit(df[["ANNUAL"]])
df["ANNUAL"]=scaler.transform(df[["ANNUAL"]])
df.head()
```

Out[61]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | ... | NOV | DEC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 35 | 0.000000 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | ... | 558.2 | 33.6 |
| **1** | 35 | 0.008772 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | ... | 359.0 | 160.5 |
| **2** | 35 | 0.017544 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | ... | 284.4 | 225.0 |
| **3** | 35 | 0.026316 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | ... | 308.7 | 40.1 |
| **4** | 35 | 0.035088 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | ... | 25.4 | 344.7 |

5 rows × 22 columns

In [62]:
```
scaler.fit(df[["YEAR"]])
df["YEAR"]=scaler.transform(df[["YEAR"]])
df.head()
```

Out[62]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | ... | NOV | DEC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 35 | 0.000000 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | ... | 558.2 | 33.6 |
| **1** | 35 | 0.008772 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | ... | 359.0 | 160.5 |
| **2** | 35 | 0.017544 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | ... | 284.4 | 225.0 |
| **3** | 35 | 0.026316 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | ... | 308.7 | 40.1 |
| **4** | 35 | 0.035088 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | ... | 25.4 | 344.7 |

5 rows × 22 columns

In [63]:
```
km=KMeans()
```

In [64]:
```
y_predicted=km.fit_predict(df[["YEAR","ANNUAL"]])
y_predicted
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
```

Out[64]:
```
array([7, 7, 7, ..., 1, 1, 1])
```

In [65]:
```
df["New Cluster"]=y_predicted
df.head()
```

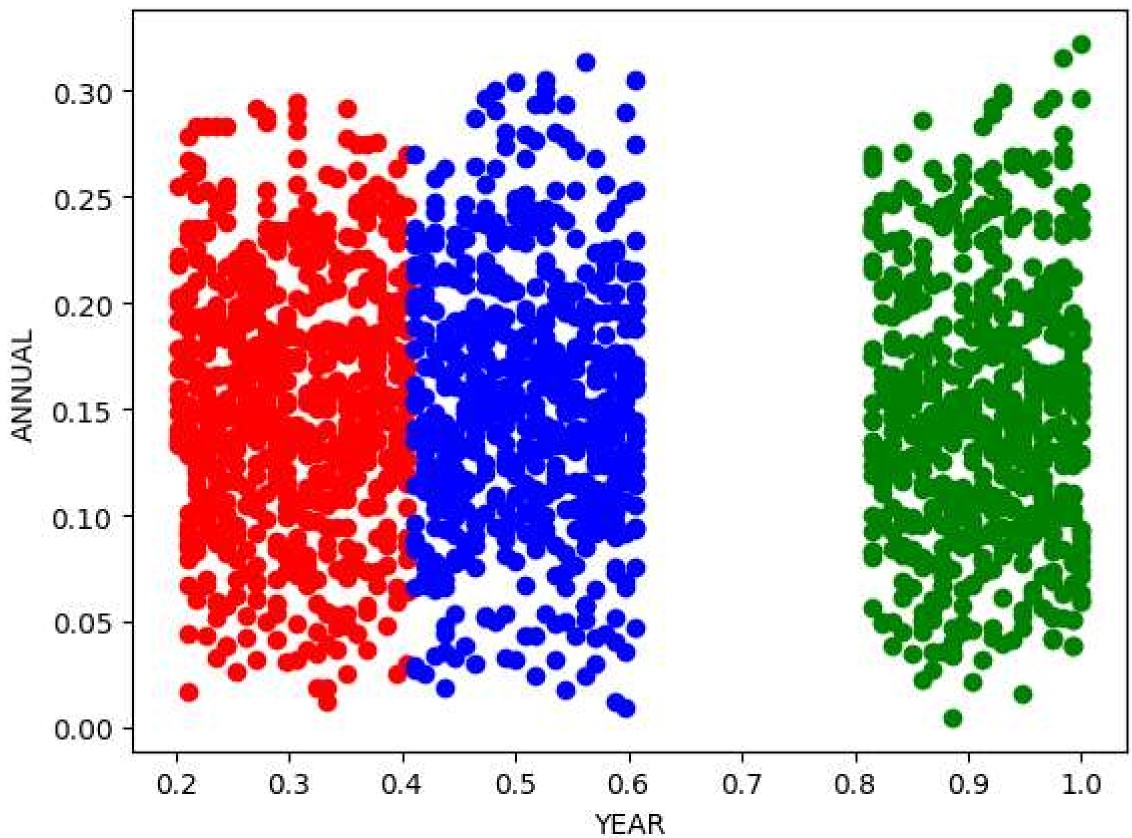
Out[65]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | ... | NOV | DEC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 35 | 0.000000 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | ... | 558.2 | 33.6 |
| **1** | 35 | 0.008772 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | ... | 359.0 | 160.5 |
| **2** | 35 | 0.017544 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | ... | 284.4 | 225.0 |
| **3** | 35 | 0.026316 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | ... | 308.7 | 40.1 |
| **4** | 35 | 0.035088 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | ... | 25.4 | 344.7 |

5 rows × 22 columns

In [66]:
```
df1=df[df["New Cluster"]==0]
df2=df[df["New Cluster"]==1]
df3=df[df["New Cluster"]==2]

plt.scatter(df1["YEAR"],df1["ANNUAL"],color="red")
plt.scatter(df2["YEAR"],df2["ANNUAL"],color="green")
plt.scatter(df3["YEAR"],df3["ANNUAL"],color="blue")

plt.xlabel("YEAR")
plt.ylabel("ANNUAL")
```

Out[66]:
```
Text(0, 0.5, 'ANNUAL')
```

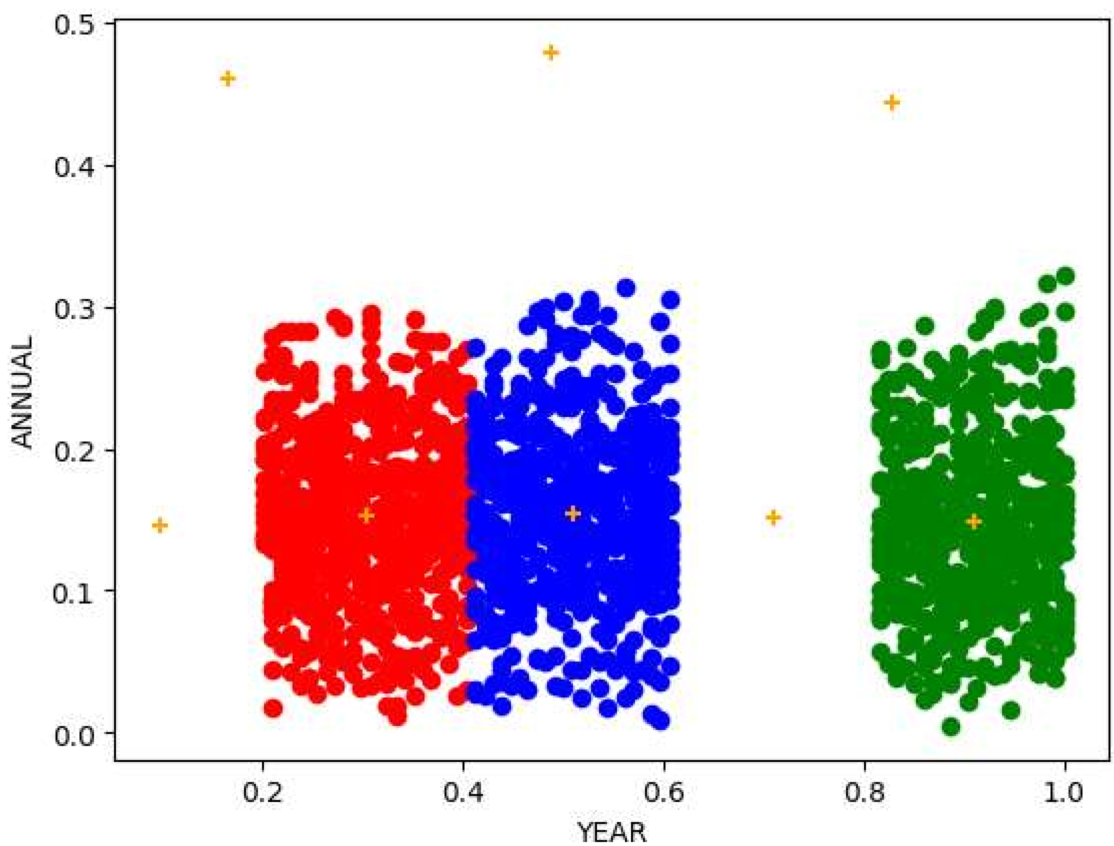



In [67]:
```
km.cluster_centers_
```

Out[67]:
```
array([[0.30426141, 0.15291403],
       [0.90944067, 0.14824557],
       [0.50928396, 0.15377016],
       [0.82795322, 0.44391899],
       [0.48811746, 0.47865383],
       [0.09719624, 0.14511135],
       [0.70936297, 0.1515095 ],
       [0.1664858 , 0.46015296]])
```

In [70]:
```
df1=df[df["New Cluster"]==0]
df2=df[df["New Cluster"]==1]
df3=df[df["New Cluster"]==2]

plt.scatter(df1["YEAR"],df1["ANNUAL"],color="red")
plt.scatter(df2["YEAR"],df2["ANNUAL"],color="green")
plt.scatter(df3["YEAR"],df3["ANNUAL"],color="blue")

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color="orange",ma

plt.xlabel("YEAR")
plt.ylabel("ANNUAL")
```

Out[70]:
```
Text(0, 0.5, 'ANNUAL')
```




In [71]:
```
k_rng=range(1,10)
sse=[]
```

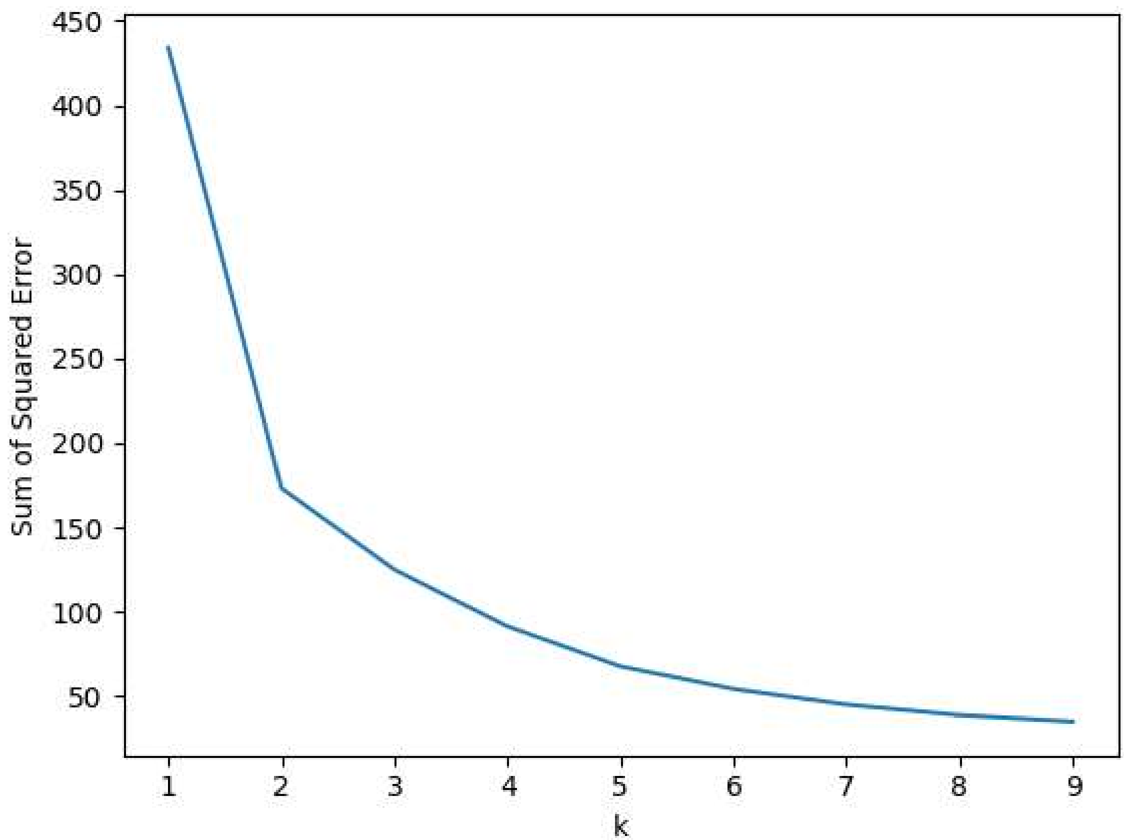
In [73]:
```
for k in k_rng:
    km=KMeans(n_clusters=k)
    km.fit(df[["YEAR","ANNUAL"]])
    sse.append(km.inertia_)
sse
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\skl
earn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to su
ppress the warning
  warnings.warn(
```

Out[73]:
```
[434.0133739262185,
 173.1482983344619,
 124.90114269286946,
 91.16133202288262,
 67.48575762867335,
 54.0886216031728,
 44.862831877855115,
 38.6674115044697,
 34.855906243888334]
```

In [74]:
```
plt.plot(k_rng,sse)
plt.xlabel("k")
plt.ylabel("Sum of Squared Error")
```

Out[74]:
```
Text(0, 0.5, 'Sum of Squared Error')
```




## Here we got the accurate curve in th graph(i.e.Elbow curve) ¶

## CONCLUSION : We can conclude that KMeans is the best model for the given datset