2440055351, Cindy Amanda Onggirawan, LB01, Final Exam COMP6745001- Machine Learning

Nomor 1: K-Nearest Neighbor

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as stats
sns.set_style('whitegrid')
```

Import Dataset

In [2]:
```
dataset = pd.read_csv('./dataset/knn_datasets.csv')
```

Exploratory Data Analysis (EDA) and Statistical Analysis

Kita melihat gambaran umum dataframe kita terlebih dahulu.

In [3]:
```
dataset.head()
```

Out[3]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9
0	3	126	88.0	41	235	39.3	0.704	27	0
1	8	99	84.0	0	0	35.4	0.388	50	0
2	7	196	90.0	0	0	39.8	0.451	41	1
3	9	119	80.0	35	0	29.0	0.263	29	1
4	11	143	94.0	33	146	36.6	0.254	51	1

Kita mengidentikasi data types, columns names, null value counts, dan memory usage.

In [4]:
```
dataset.shape
```

Out[4]:
```
(480, 9)
```

In [5]:
```
dataset.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   X1      480 non-null    int64  
 1   X2      480 non-null    int64  
 2   X3      479 non-null    float64
 3   X4      480 non-null    int64  
 4   X5      480 non-null    int64  
 5   X6      480 non-null    float64
 6   X7      480 non-null    float64
 7   X8      480 non-null    int64  
 8   X9      480 non-null    int64  
dtypes: float64(3), int64(6)
memory usage: 33.9 KB
```

Kita berusaha melihat detail dari setiap feature.

In [6]:
```
dataset.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
X1	480.0	3.779167	3.335327	0.000	1.000	3.000	6.0000	17.00
X2	480.0	121.087500	32.312033	0.000	99.750	117.000	142.0000	197.00
X3	479.0	69.033403	19.100344	0.000	64.000	70.000	80.0000	122.00
X4	480.0	20.714583	15.632784	0.000	0.000	23.000	33.0000	63.00
X5	480.0	79.135417	114.186313	0.000	0.000	36.500	122.7500	744.00
X6	480.0	32.022083	8.145896	0.000	27.075	32.000	36.6000	67.10
X7	480.0	0.485019	0.337495	0.078	0.254	0.384	0.6455	2.42

	count	mean	std	min	25%	50%	75%	max
X8	480.0	32.916667	11.606336	21.000	24.000	29.000	39.0000	81.00
X9	480.0	0.352083	0.478118	0.000	0.000	0.000	1.0000	1.00

Missing Value Treatment

Dengan menganalisis detail feature di atas, kita menemukan bahwa beberapa feature memiliki nilai nol karena terdapat beberapa feature yang memiliki kisaran nilai yang tidak wajar.

Fitur di bawah ini memiliki nilai nol yang tidak valid:

- X2
- X3
- X4
- X5
- X6

In [7]:
```
dataset.isnull().sum()
```

Out[7]:
```
X1    0
X2    0
X3    1
X4    0
X5    0
X6    0
X7    0
X8    0
X9    0
dtype: int64
```

Jadi, kita akan mengganti angka nol ini dengan NaN agar mudah untuk menghitung nilai yang hilang.

In [8]:
```
dataset[['X2','X3','X4','X5','X6']] = dataset[['X2','X3','X4','X5','X6']].replace(0,np.NaN)
```

In [9]:
```
dataset.head()
```

Out[9]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9

	X1	X2	X3	X4	X5	X6	X7	X8	X9
0	3	126.0	88.0	41.0	235.0	39.3	0.704	27	0
1	8	99.0	84.0	NaN	NaN	35.4	0.388	50	0
2	7	196.0	90.0	NaN	NaN	39.8	0.451	41	1
3	9	119.0	80.0	35.0	NaN	29.0	0.263	29	1
4	11	143.0	94.0	33.0	146.0	36.6	0.254	51	1

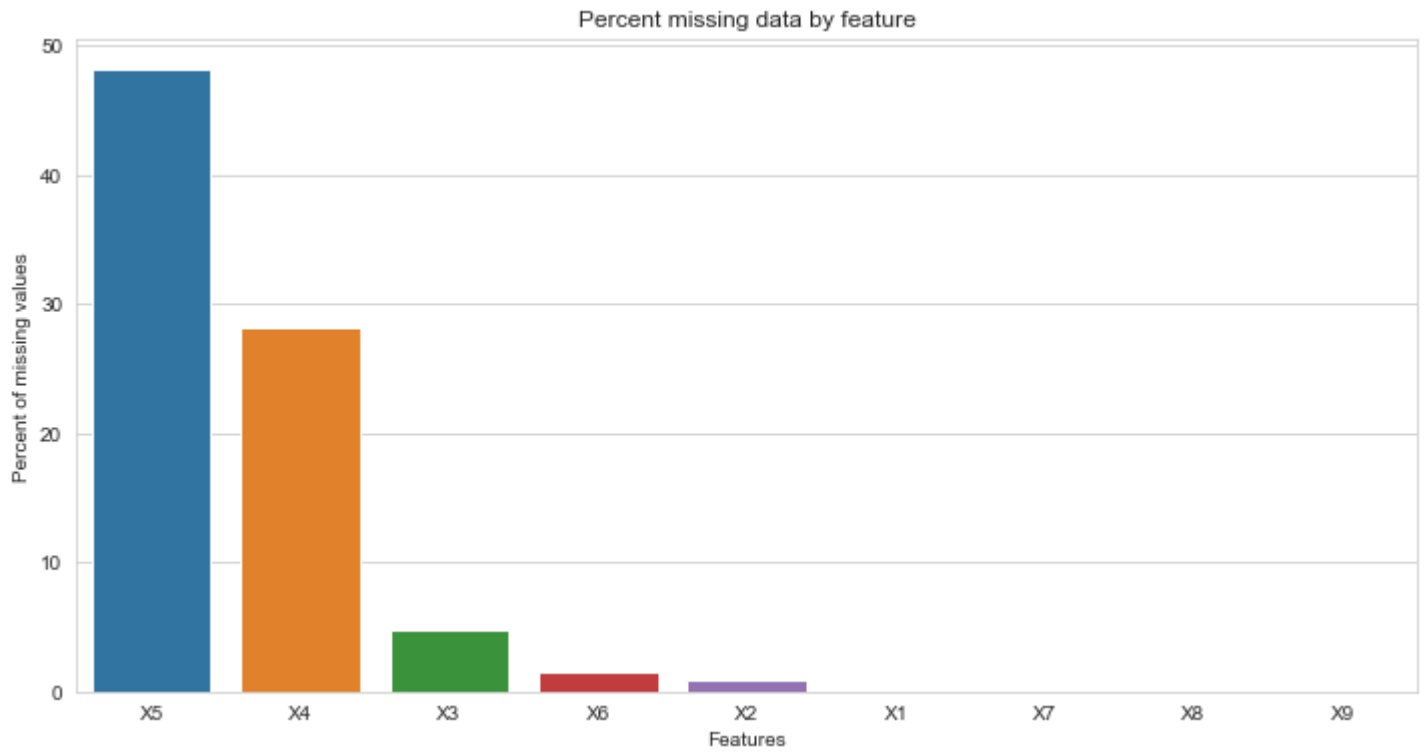
Kita coba melihat persentase nilai NaN untuk setiap feature.

In [10]:
```
total = dataset.isnull().sum().sort_values(ascending=False)
percent = ((dataset.isnull().sum()/dataset.isnull().count())*100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(9)
```

Out[10]:

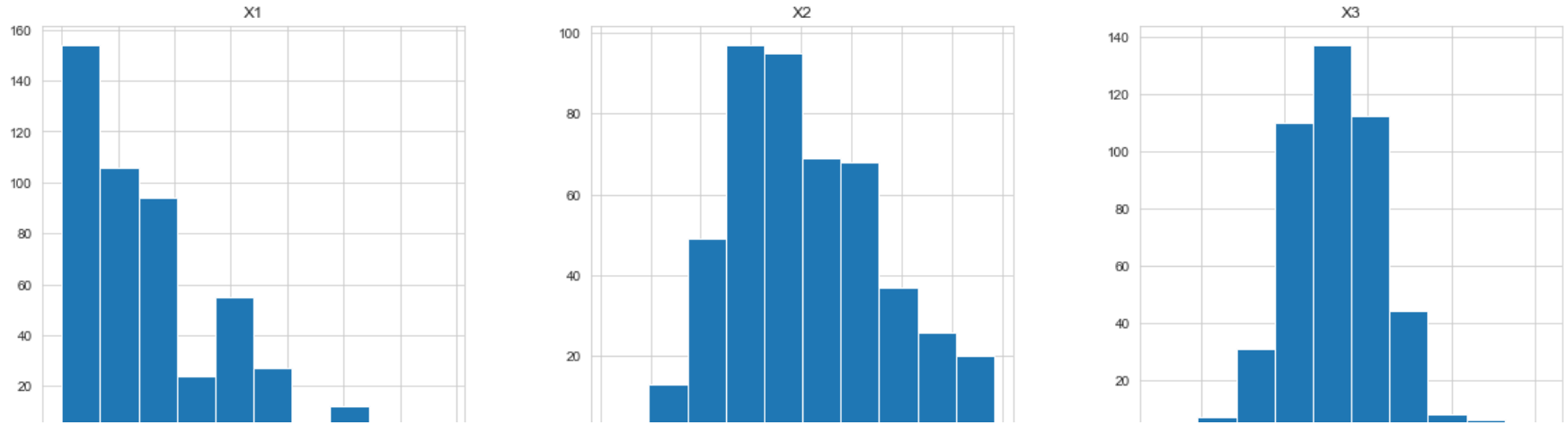
	Total	Percent
X5	231	48.125000
X4	135	28.125000
X3	23	4.791667
X6	7	1.458333
X2	4	0.833333
X1	0	0.000000
X7	0	0.000000
X8	0	0.000000
X9	0	0.000000

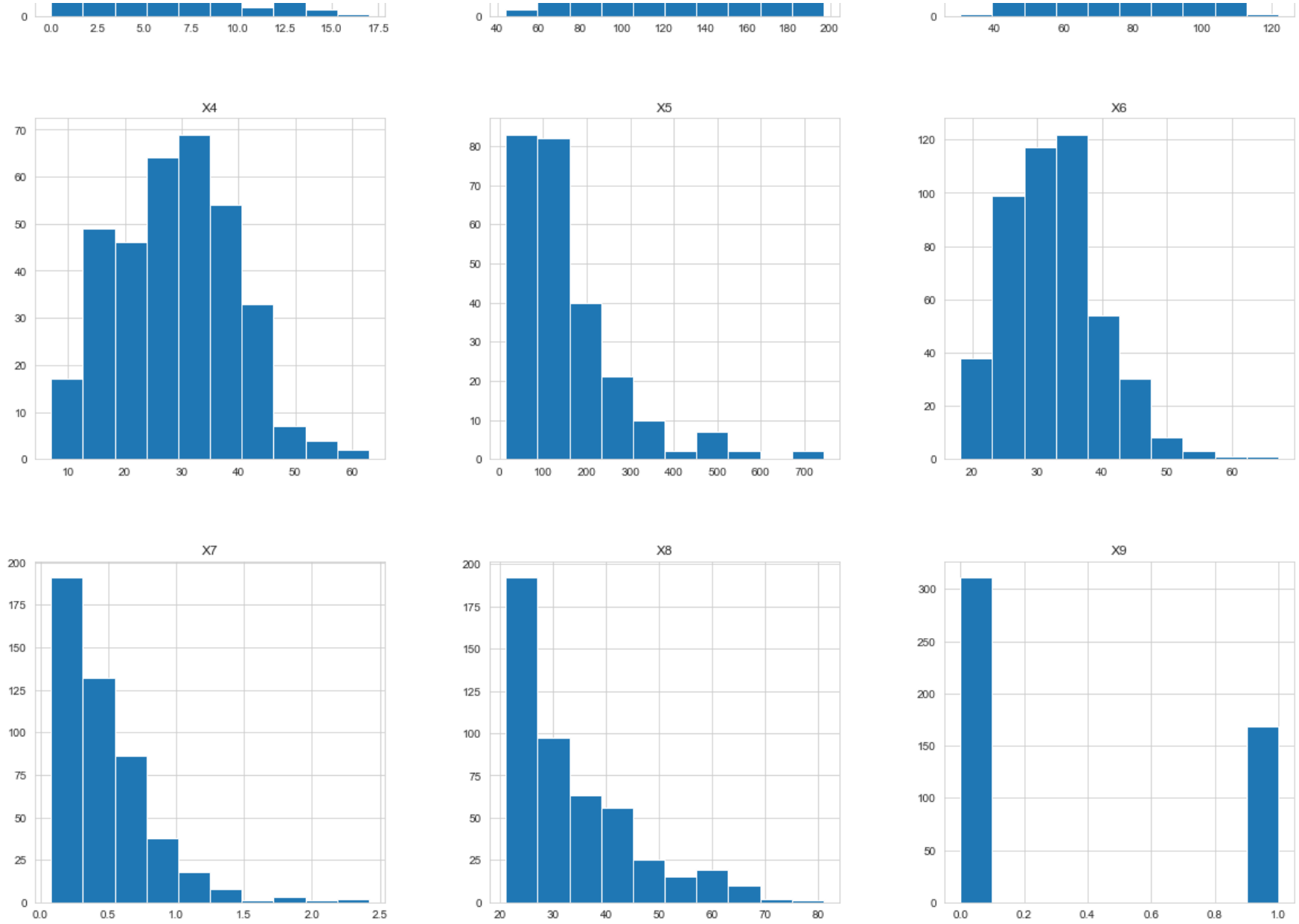
In [11]:
```
f, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x=missing_data.index, y=missing_data['Percent'])
plt.title('Percent missing data by feature')
plt.xlabel('Features')
plt.ylabel('Percent of missing values')
plt.show()
```



Understand Data Distribution

In [12]:
```
dataset.hist(figsize = (20,20))
plt.show()
```





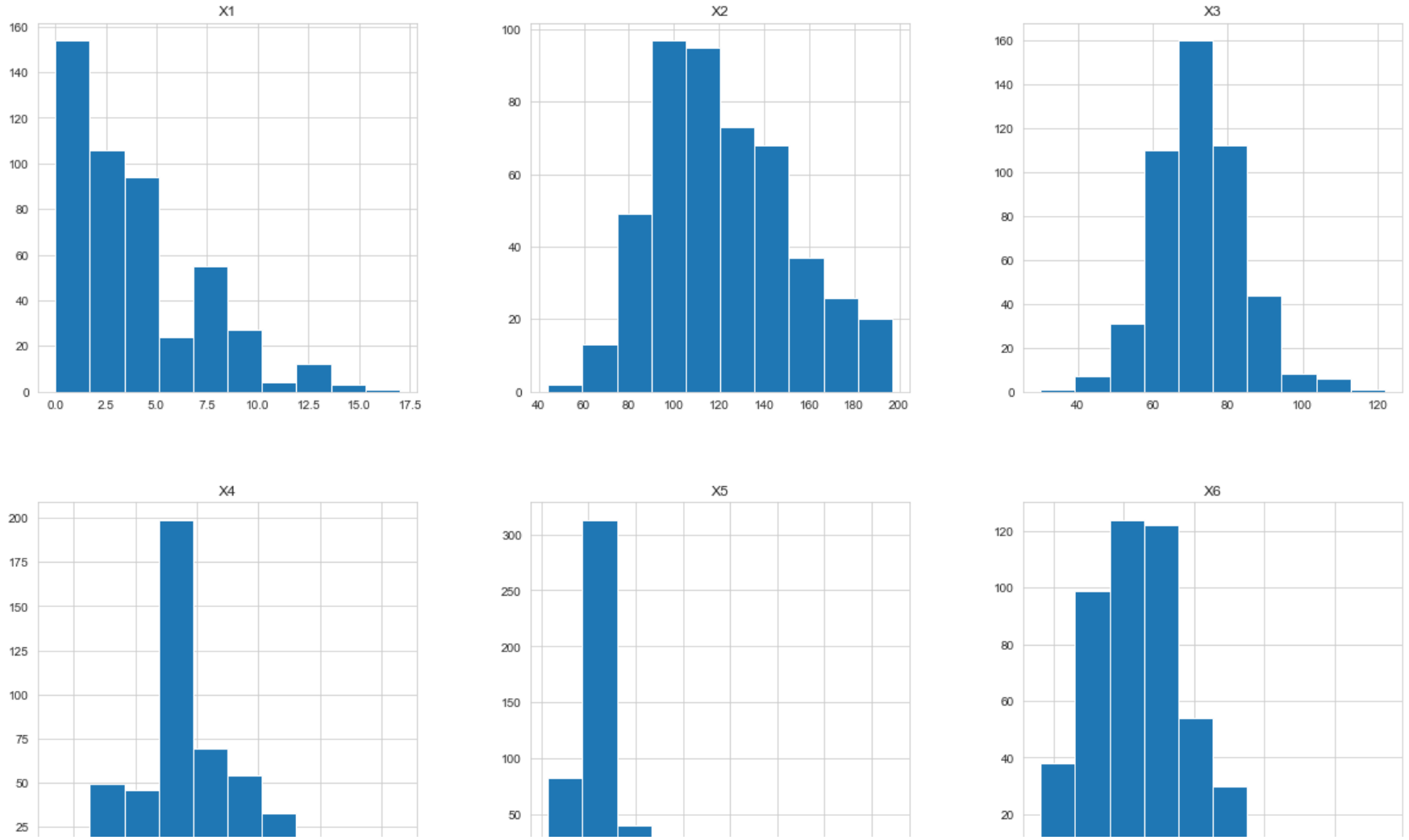
Grafik diatas menunjukkan bahwa feature X2 dan X3 hampir mendekati normal distribution sedangkan X4, X5, dan X6 berbentuk positive skewed. Jadi, kita akan mengganti nilai yang hilang sesuai dengan distribusinya.

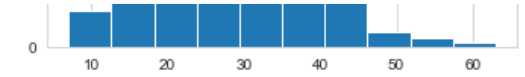
- Jika mendekati normal distribution, maka isi NaN dengan mean-nya.
- Jika berbentuk positive skewed, maka isi NaN dengan median-nya.

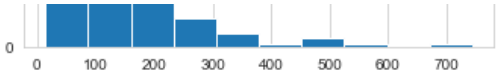
In [13]:
```
dataset['X2'].fillna(dataset['X2'].mean(), inplace = True)
dataset['X3'].fillna(dataset['X3'].mean(), inplace = True)
dataset['X4'].fillna(dataset['X4'].median(), inplace = True)
dataset['X5'].fillna(dataset['X5'].median(), inplace = True)
dataset['X6'].fillna(dataset['X6'].median(), inplace = True)
```

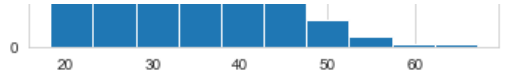
Sekarang, kita coba melihat kembali grafik setelah mengisi NaN.

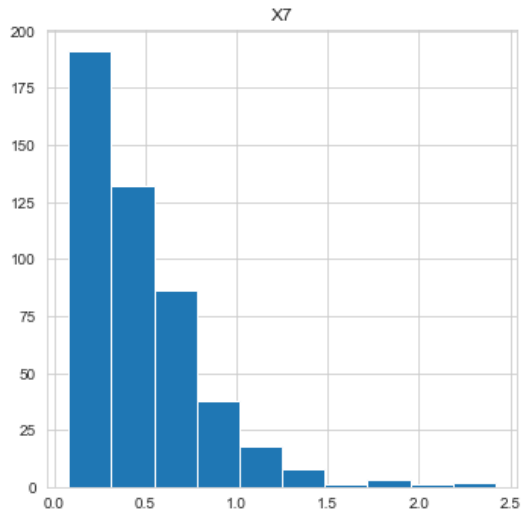
In [14]:
```
dataset.hist(figsize = (20,20))
plt.show()
```

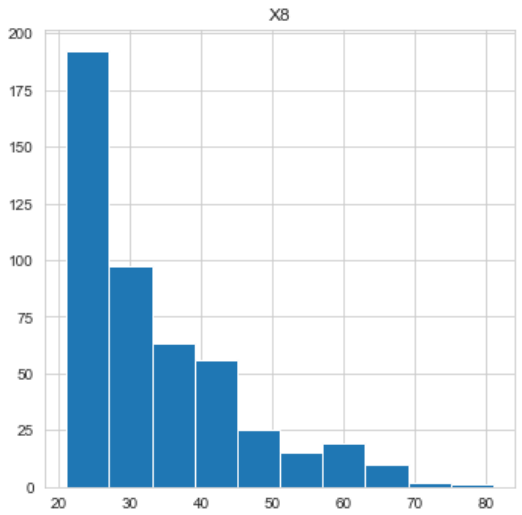


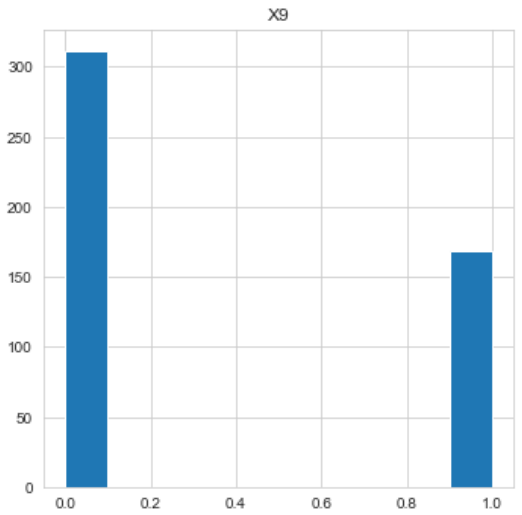












Kita berhasil mengubah feature X2, X3, X4, X5, dan X6 mendekati normal distribution.

Feature Selection

Kita perlu menghitung correlation antara independent variable dan dependent variable.

In [15]:
```
dataset.corr()
```

Out[15]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9
X1	1.000000	0.181010	0.167031	0.103035	0.054359	0.039735	-0.041453	0.542114	0.226904
X2	0.181010	1.000000	0.180969	0.197794	0.423308	0.244353	0.151050	0.295736	0.464020
X3	0.167031	0.180969	1.000000	0.191416	0.050763	0.300900	-0.021383	0.316782	0.161057
X4	0.103035	0.197794	0.191416	1.000000	0.223988	0.573104	0.112891	0.102762	0.215599
X5	0.054359	0.423308	0.050763	0.223988	1.000000	0.224863	0.194677	0.073796	0.185890
X6	0.039735	0.244353	0.300900	0.573104	0.224863	1.000000	0.138555	0.016722	0.331095
X7	-0.041453	0.151050	-0.021383	0.112891	0.194677	0.138555	1.000000	0.033701	0.177169

	X1	X2	X3	X4	X5	X6	X7	X8	X9
X8	0.542114	0.295736	0.316782	0.102762	0.073796	0.016722	0.033701	1.000000	0.195662
X9	0.226904	0.464020	0.161057	0.215599	0.185890	0.331095	0.177169	0.195662	1.000000

Untuk mempermudah, kita gunakan heatmap dan membandingkan korelasi dengan setiap kolom dengan warna.

In [16]:
```
plt.figure(figsize=(12,10))
sns.heatmap(dataset.corr(), annot=True, cmap='viridis', linewidths=.1)
plt.show()
```

Semua feature tampak berkorelasi positif dengan target X9.

In [17]:
```
dataset.corr()['X9'].sort_values()
```

Out[17]:
```
X3    0.161057
X7    0.177169
X5    0.185890
X8    0.195662
```

```
X4    0.215599
X1    0.226904
X6    0.331095
X2    0.464020
X9    1.000000
Name: X9, dtype: float64
```

Akibat dari nilai correlation yang kecil (dibawah 0.5) maka kita putuskan untuk menentukan nilai threshold berdasarkan rata-rata nilai correlation, bukan nilai absolut.

In [18]:
```
threshold = dataset.corr()['X9'].sort_values().mean()
```

In [19]:
```
corr_target = abs(dataset.corr()['X9'])
relevant_features = corr_target[corr_target > threshold]
relevant_features
```

Out[19]:
```
X2    0.464020
X6    0.331095
X9    1.000000
Name: X9, dtype: float64
```

Hanya feature X2 dan X6 yang highly correlated dengan target X9. Berarti, kita akan menghapus semua feature lain selain kedua ini. Namun, kita masih harus memastikan bahwa independent variables ini harus tidak berkolerasi satu sama lain. Jika ternyata indepent variables ini berkorelasi satu sama lain, maka kita hanya perlu menyimpan salah satunya dan membuang sisanya.

In [20]:
```
dataset[['X2', 'X6']].corr()
```

Out[20]:

	X2	X6
X2	1.000000	0.244353
X6	0.244353	1.000000

In [21]:
```
dataset[['X2', 'X6']].corr().iloc[1,0] < threshold
```

Out[21]:
```
True
```

Ternyata, X2 dan X6 tidak highly correlated satu sama lain, berarti inilah feature final yang diberikan oleh Pearson correlation. Kita dapat mengambil kedua feature ini untuk dipasangkan dengan target X9.

In [22]:
```
dataset = dataset.drop(['X1', 'X3', 'X4', 'X5', 'X7', 'X8'], axis = 1)
```

In [23]:
```
dataset
```

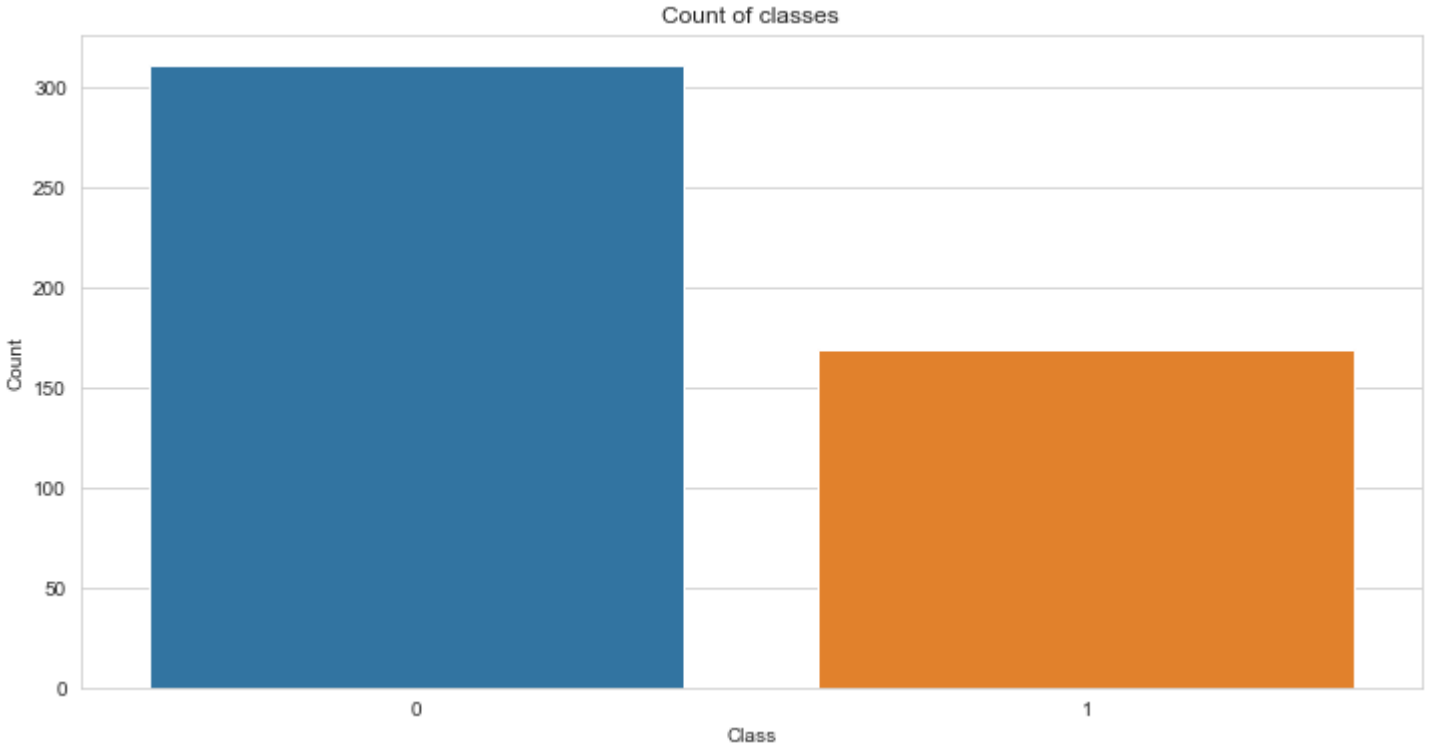
Out[23]:

	X2	X6	X9
0	126.0	39.3	0
1	99.0	35.4	0
2	196.0	39.8	1
3	119.0	29.0	1
4	143.0	36.6	1
...	...	...	...
475	166.0	26.6	0
476	110.0	26.0	0
477	81.0	30.1	0
478	195.0	25.1	1
479	154.0	29.3	0

480 rows × 3 columns

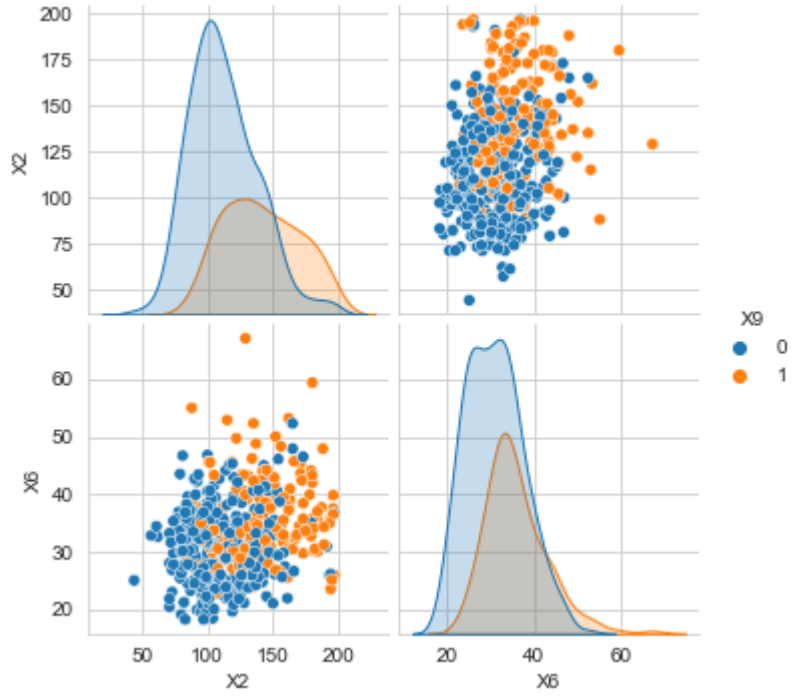
Checking Balance of Data

In [24]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='X9',data=dataset)
plt.title("Count of classes")
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



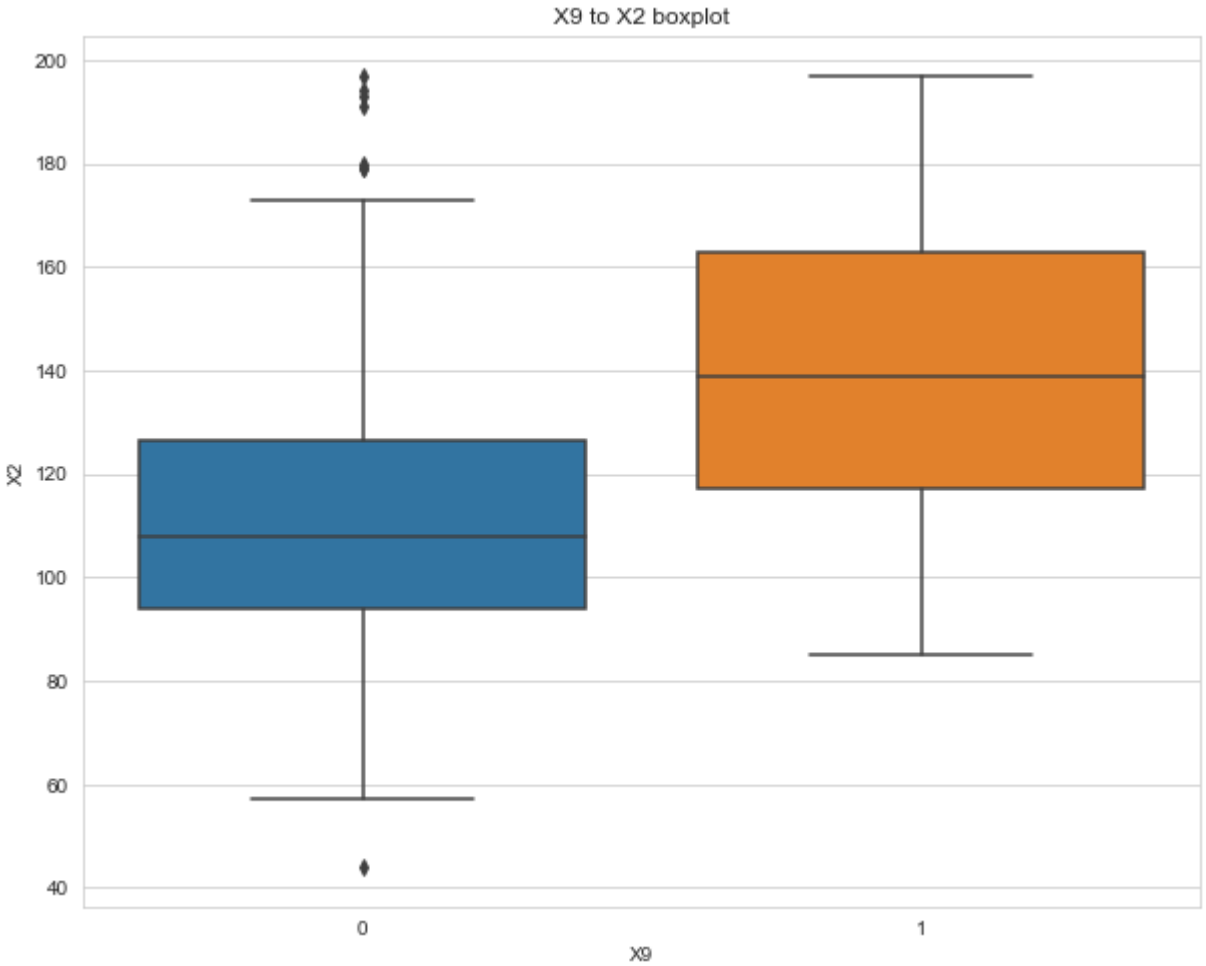
Grafik diatas menunjukkan bahwa dataset bias terhadap class 0. Jumlah data dengan kelas 0 hampir dua kali lipat jumlah data dengan class 1.

In [25]:
```
sns.pairplot(dataset, hue='X9')
plt.show()
```



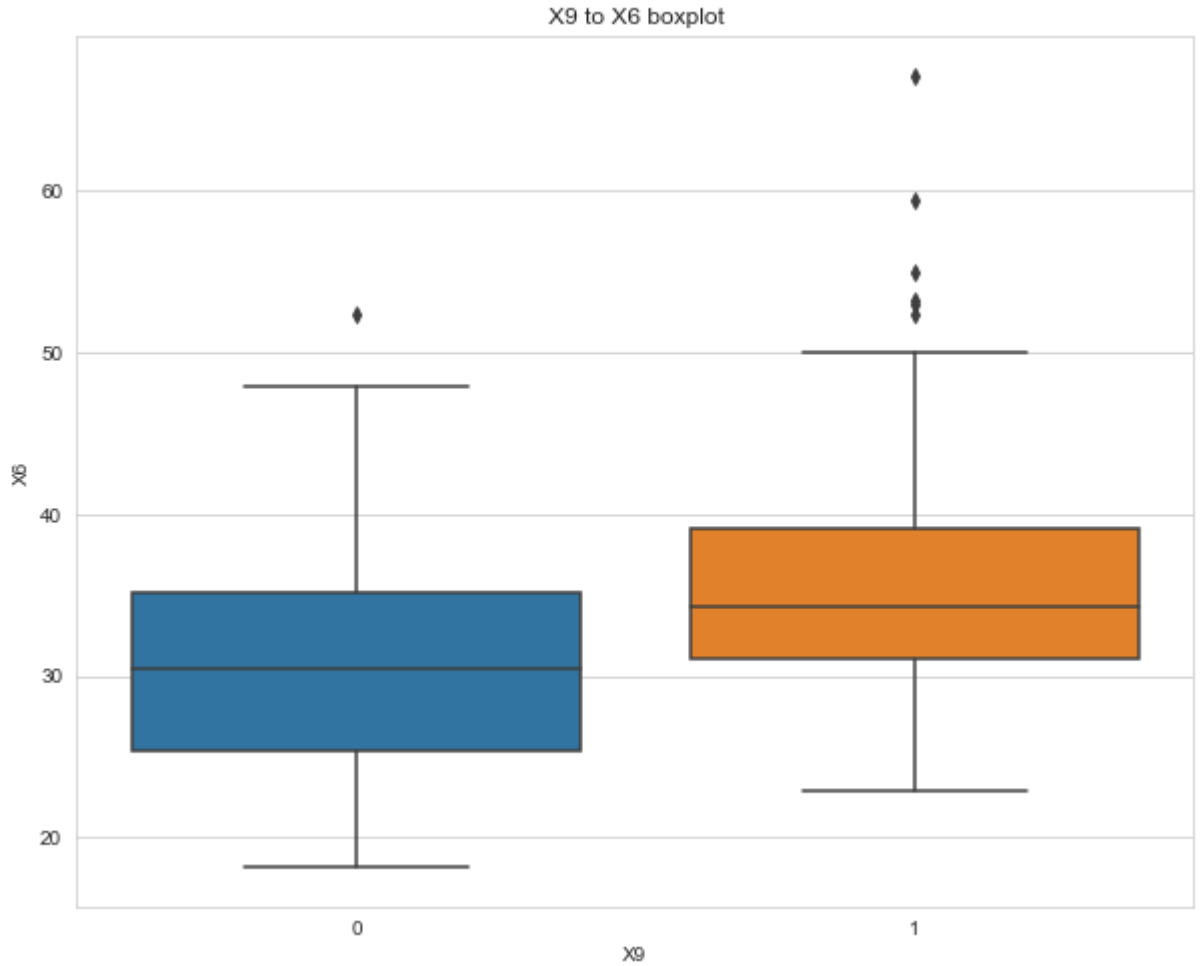
In [26]:

```
plt.figure(figsize=(10,8))
sns.boxplot(data = dataset, x = 'X9', y = 'X2')
plt.title('X9 to X2 boxplot')
plt.show()
```



Kita melihat bahwa class 0 memiliki nilai X2 yang lebih rendah dari class 1, hal ini terlihat dari nilai mediannya yang lebih kecil. Jumlah outliers class 0 juga lebih banyak daripada class 1.

In [27]:
```
plt.figure(figsize=(10,8))
sns.boxplot(data = dataset, x = 'X9', y = 'X6')
plt.title('X9 to X6 boxplot')
plt.show()
```



Kita melihat bahwa class 0 memiliki nilai X6 yang lebih rendah dari class 1, hal ini terlihat dari nilai mediannya yang lebih kecil. Namun, jumlah outliers class 0 lebih sedikit daripada class 1.

Defining Standard Scaler from Scratch

In [28]:
```
class StandardScaler():
    def __init__(self):
        pass

    def fit(self, X):
        self.mean_ = np.mean(X, axis=0)
        self.scale_ = np.std(X - self.mean_, axis=0)
        return self
```

```
    def transform(self, X):
        return (X - self.mean_) / self.scale_

    def fit_transform(self, X):
        return self.fit(X).transform(X)
```

Standardize the Variables

In [29]:
```
scaled_features = StandardScaler().fit_transform(dataset.drop('X9',axis=1))
```

In [30]:
```
scaled_features
```

Out[30]:

	X2	X6
0	0.128505	0.952952
1	-0.762297	0.407075
2	2.437993	1.022936
3	-0.102444	-0.488723
4	0.689381	0.575037
...	...	...
475	1.448213	-0.824648
476	-0.399378	-0.908629
477	-1.356166	-0.334758
478	2.405001	-1.034600
479	1.052300	-0.446733

480 rows × 2 columns

In [31]:
```
target = dataset['X9']
```

In [32]:
```
target
```

Out[32]:
```
0      0
1      0
2      1
3      1
4      1
      ..
475    0
476    0
477    0
478    1
479    0
Name: X9, Length: 480, dtype: int64
```

In [33]:
```
X, y = scaled_features.to_numpy(), target.to_numpy()
```

Train Test Split

In [34]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

In [35]:
```
X_train.shape
```

Out[35]:
```
(360, 2)
```

In [36]:
```
y_train.shape
```

Out[36]:
```
(360,)
```

Defining KNN from Scratch

In [37]:
```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1-x2)**2))
```

In [38]:
```
class KNN:
    def __init__(self, k = 3):
        self.k = k
```

```
    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predicted_labels = [self._predict(x) for x in X]
        return np.array(predicted_labels)

    def _predict(self, x):
        #Hitung jarak dari sebuah titik ke semua titik di x_train
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        #Ambil sejumlah k titik dengan jarak terdekat dan simpan dalam bentuk indeks angka
        k_indices = np.argsort(distances)[:self.k]

        #Tampung label prediksi
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        #Tentukan kelas berdasarkan suara terbanyak
        freq0 = 0
        freq1 = 0
        for target in k_nearest_labels:
            if(target == 0): freq0+=1
            else: freq1+=1
        if(freq0 > freq1): return 0
        else: return 1

    def score(self, X, y):
        pred = knn.predict(X)
        return np.sum(pred == y) / len(y)
```

Choosing a K Value

Kita mencari tahu berapa nilai K yang optimal untuk model KNN kita dengan menggunakan elbow method.

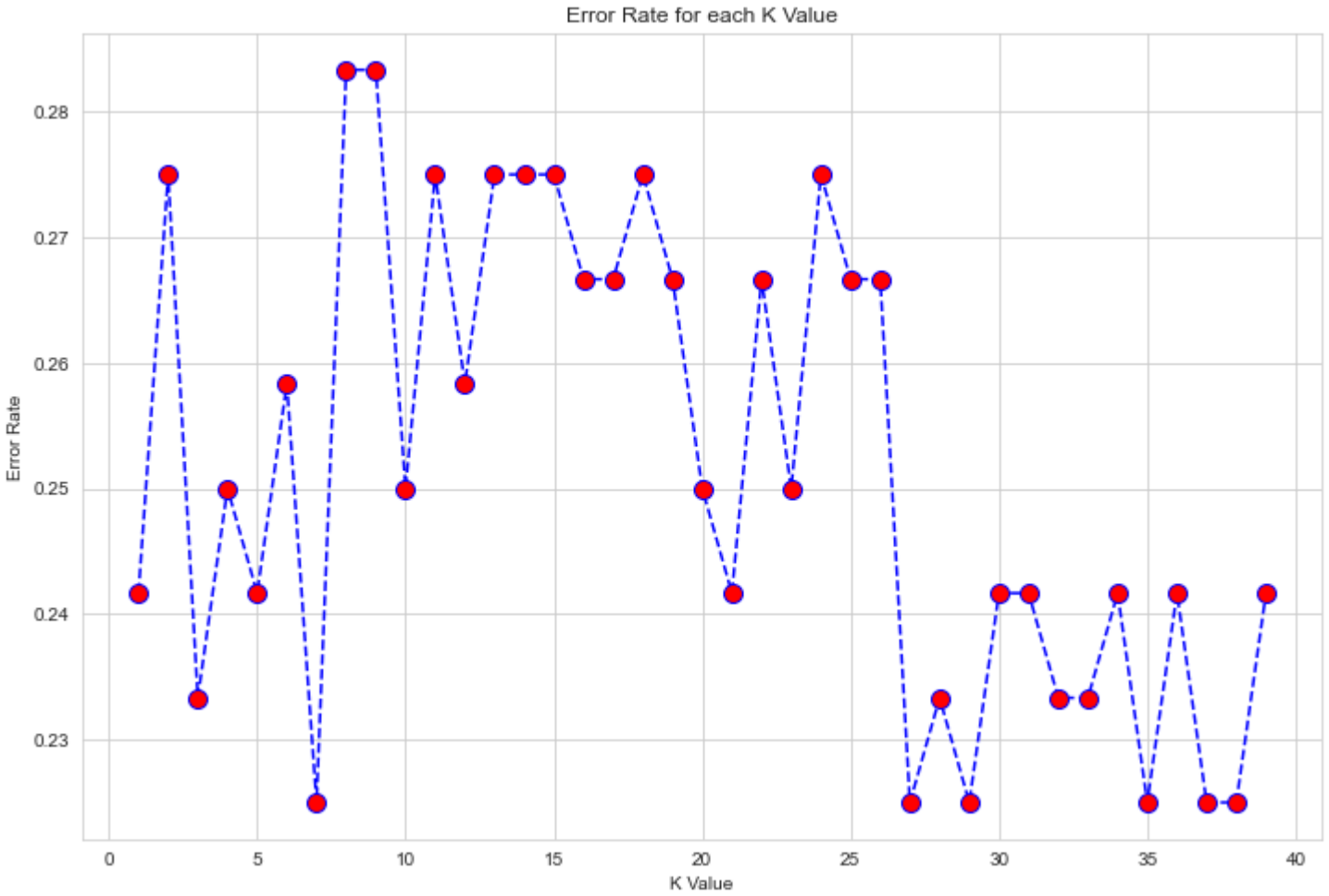
In [39]:

```
error_rate = []
test_scores = []
train_scores = []

for i in range(1,40):
    knn = KNN(k = i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
```

```
    error_rate.append(np.mean(pred_i != y_test))
    train_scores.append(knn.score(X_train, y_train))
    test_scores.append(knn.score(X_test, y_test))
```

In [40]:

```
plt.figure(figsize=(12,8))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('Error Rate for each K Value')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.show()
```



Kita dapat lihat bahwa cekungan terdalam yang menunjukkan error rate terendah terdapat pada K = 3. Oleh karena itu, kita mengambil K =

3 untuk digunakan dalam model KNN.

In [41]:
```
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))
```

```
Max train score 99.72222222222223 % and k = [1]
```

In [42]:
```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

```
Max test score 77.5 % and k = [7, 27, 29, 35, 37, 38]
```

Predictions

In [43]:
```
knn = KNN(k = 7)
```

In [44]:
```
knn.fit(X_train, y_train)
```

In [45]:
```
y_pred = knn.predict(X_test)
```

In [46]:
```
y_pred
```

Out[46]:
```
array([0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1])
```

Evaluations

In [47]:
```
i = 0
TP = 0
TN = 0
FP = 0
FN = 0
```

```
for y_actual in y_test:
    if (y_pred[i] == 1) and (y_actual == 1):
        TP+=1
    elif (y_pred[i] == 0) and (y_actual == 0):
        TN+=1
    elif (y_pred[i] == 1) and (y_actual == 0):
        FP+=1
    elif (y_pred[i] == 0) and (y_actual == 1):
        FN+=1
    i+=1
```

In [48]:
```
accuracy = (TP + TN) / (TP + FP + TN + FN)
precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)
```

In [49]:
```
print(f'Accuracy  : {str(round(accuracy*100, 2))}%')
print(f'Precision : {str(round(precision*100, 2))}%')
print(f'Recall    : {str(round(precision*100, 2))}%')
print(f'F1 Score  : {str(round(f1_score*100, 2))}%')
```

```
Accuracy  : 77.5%
Precision : 66.67%
Recall    : 66.67%
F1 Score  : 61.97%
```

Model Performance Analysis

In [50]:
```
confusion_matrix = np.array([[TP, FP], [FN, TN]])
```

In [51]:
```
confusion_matrix
```

Out[51]:
```
array([[22, 11],
       [16, 71]])
```

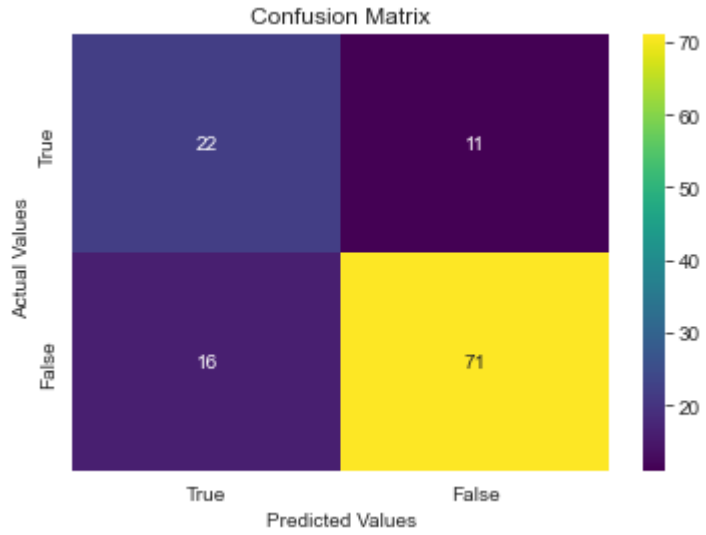
In [52]:
```
matrix_df = pd.DataFrame(confusion_matrix)
matrix_df.columns = ['True', 'False']
matrix_df.index = ['True', 'False']
```

In [53]:
```
matrix_df
```

Out[53]:

	True	False
True	22	11
False	16	71

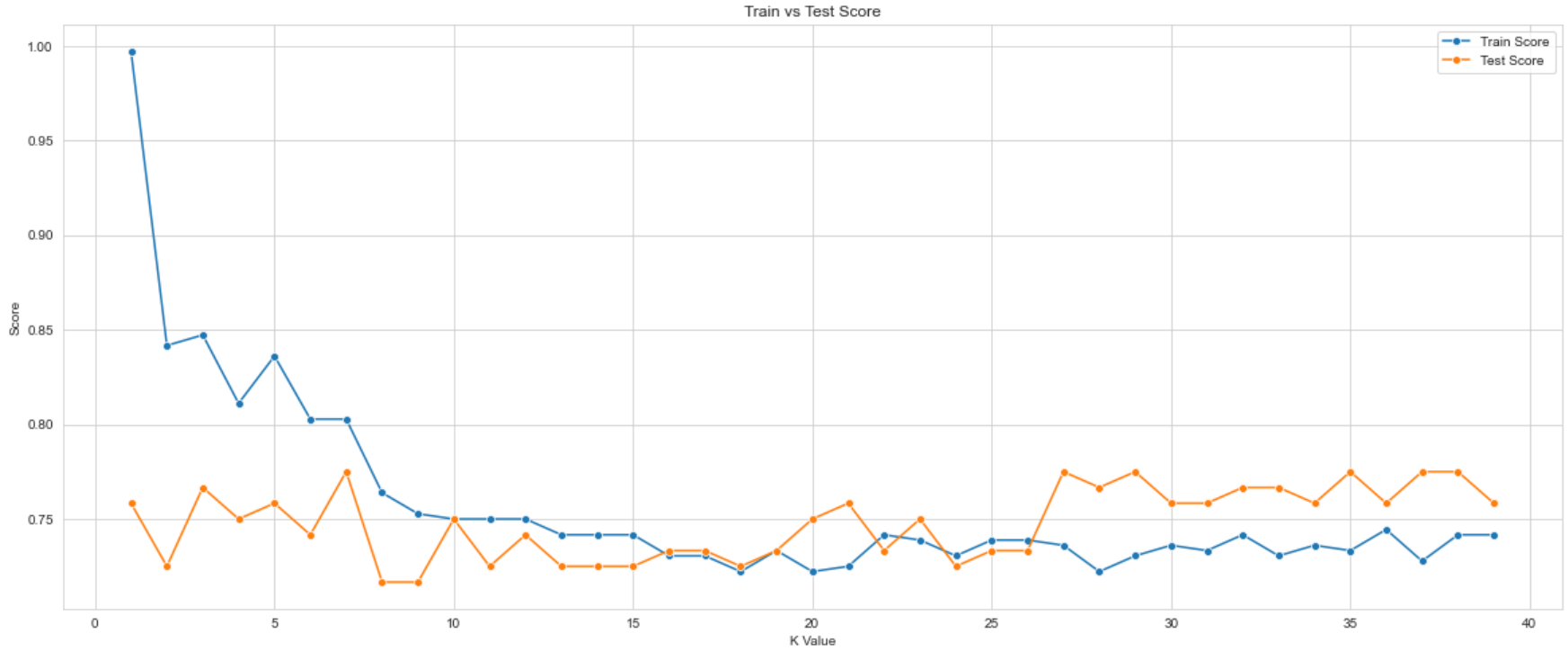
In [54]:
```
cm = sns.heatmap(confusion_matrix, annot=True, cmap='viridis')

cm.set_title('Confusion Matrix');
cm.set_xlabel('Predicted Values')
cm.set_ylabel('Actual Values ');

cm.xaxis.set_ticklabels(['True','False'])
cm.yaxis.set_ticklabels(['True','False'])

plt.show()
```



Result Visualisation

Sama seperti sebelumnya, kita buat visualisasi untuk membandingkan akurasi training data dengan testing data, apabila nilai K-nya berubah-ubah.

In [55]:
```
plt.figure(figsize=(20,8))
sns.lineplot(x = range(1,40), y = train_scores, marker = 'o', label = 'Train Score')
sns.lineplot(x = range(1,40), y = test_scores, marker = 'o', label = 'Test Score')
plt.title('Train vs Test Score')
plt.xlabel('K Value')
plt.ylabel('Score')
plt.show()
```



Jika kita melihat garis oranye, maka tampak bahwa K = 7 menyebabkan kestabilan score sampai K = 9. Artinya, ini sejalan dengan teori elbow method yang menentukan nilai K yang menyebabkan score tidak banyak berubah dan cenderung stabil.

In [56]:
```
d1 = {'X2': X_train[:,0], 'X6': X_train[:,1], 'X9': y_train}
```

In [57]:
```
train_dataset = pd.DataFrame(d1)
```

In [58]:
```
train_dataset.head()
```

Out[58]:

	X2	X6	X9
0	-0.663319	0.057154	1
1	-0.531349	0.575037	0
2	-0.828283	-0.740667	0
3	-0.861276	-1.412515	0
4	-0.696312	-1.160572	0

In [59]:
```
d2 = {'X2': X_test[:,0], 'X6': X_test[:,1], 'X9': y_pred}
```

In [60]:
```
test_dataset = pd.DataFrame(d2)
```

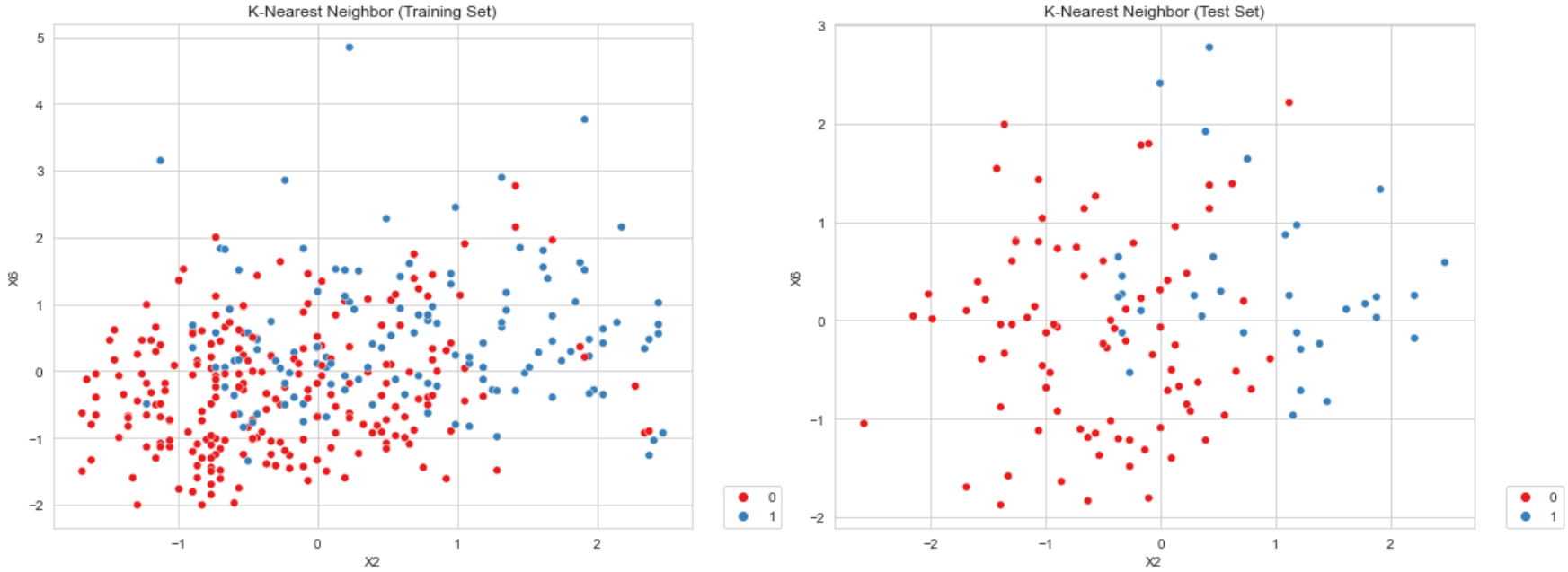
In [61]:
```
test_dataset.head()
```

Out[61]:

	X2	X6	X9
0	0.392447	-1.216559	0
1	-1.059232	-1.118581	0
2	0.524417	0.295100	1
3	0.128505	-0.250777	0
4	0.227483	-0.852641	0

In [62]:
```
fig = plt.figure(figsize=(16,6))

fig.add_subplot(1, 2, 1)
labels = train_dataset['X9'].values
sns.scatterplot(x = X_train[:,0], y = X_train[:,1], hue = labels, palette='Set1')
plt.legend(loc=(1.05,0))
plt.title('K-Nearest Neighbor (Training Set)')
plt.xlabel('X2')
plt.ylabel('X6')

fig.add_subplot(1, 2, 2)
```

```
labels = test_dataset['X9'].values
sns.scatterplot(x = X_test[:,0], y = X_test[:,1], hue = labels, palette='Set1')
plt.legend(loc=(1.05,0))
plt.title('K-Nearest Neighbor (Test Set)')
plt.xlabel('X2')
plt.ylabel('X6')

fig.tight_layout()
plt.show()
```



Terima kasih.

2440055351, Cindy Amanda Onggirawan, LB01, Final Exam COMP6745001- Machine Learning

Nomor 2: K-Means

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as stats
sns.set_style('whitegrid')
```

Import Dataset

In [2]:
```
dataset = pd.read_csv('./dataset/DNA.csv')
```

Exploratory Data Analysis (EDA) and Statistical Analysis

Kita melihat gambaran umum dataframe kita terlebih dahulu.

In [3]:
```
dataset.head()
```

Out[3]:

	num_of_atoms	num_of_heavy_atoms
0	14	6
1	11	5
2	15	6
3	21	7
4	20	7

Kita mengidentikasi data types, columns names, null value counts, dan memory usage.

In [4]:
```
dataset.shape
```

Out[4]:
```
(14610, 2)
```

In [5]:
```
dataset.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14610 entries, 0 to 14609
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   num_of_atoms        14610 non-null  int64
 1   num_of_heavy_atoms  14610 non-null  int64
dtypes: int64(2)
memory usage: 228.4 KB
```

Kita berusaha melihat detail dari setiap feature.

In [6]:
```
dataset.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
num_of_atoms	14610.0	14.412594	4.693705	2.0	11.0	15.0	18.0	33.0
num_of_heavy_atoms	14610.0	5.903285	1.113774	2.0	5.0	6.0	7.0	11.0

Missing Value Treatment

Dengan menganalisis detail feature di atas, kita merasa bahwa kedua feature memiliki kisaran nilai yang masuk akal.

In [7]:
```
dataset.isnull().sum()
```

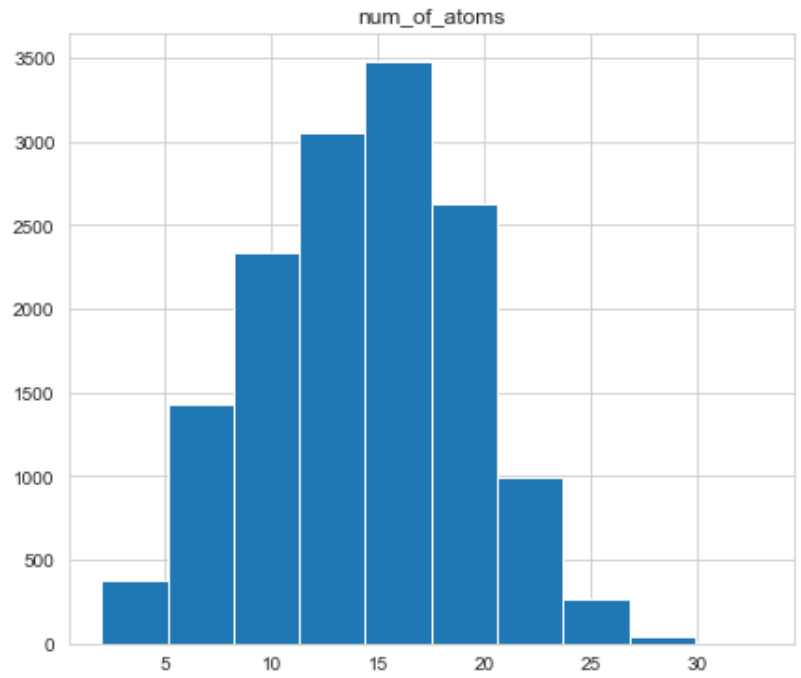
Out[7]:
```
num_of_atoms          0
num_of_heavy_atoms    0
dtype: int64
```

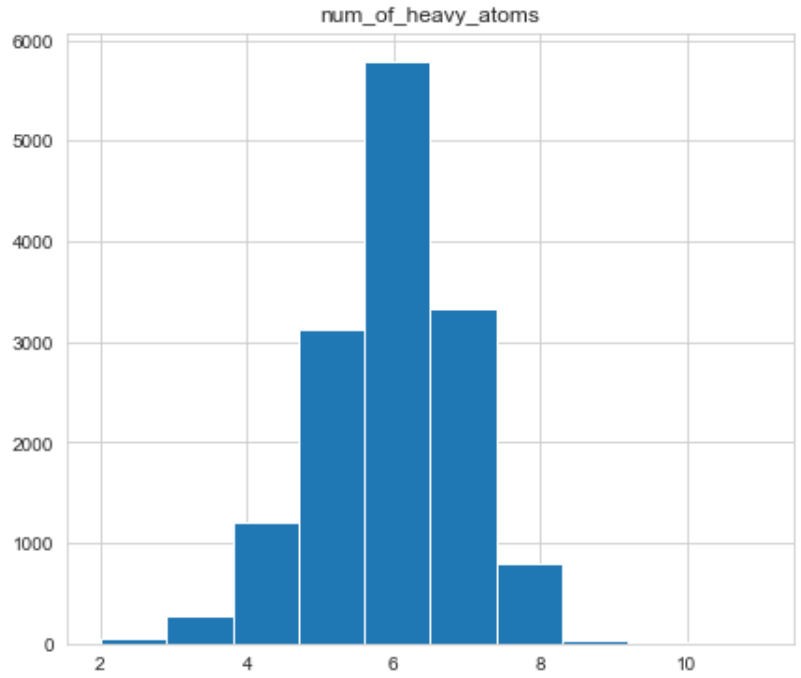
Kita tahu bahwa tidak ada feature yang bernilai kosong, berarti tidak ada yang perlu diisi.

Understand Data Distribution

In [8]:
```
dataset.hist(figsize = (16,6))
```

```
plt.show()
```





Grafik diatas menunjukkan bahwa feature num_of_atoms dan num_of_heavy_atoms hampir mendekati normal distribution.

Feature Selection

Kita perlu melihat correlation antara num_of_atoms dan num_of_heavy_atoms. Namun, perlu diperhatikan bahwa feature selection adalah teknik yang terkenal untuk supervised learning, tetapi tidak banyak implementasinya untuk unsupervised learning, seperti clustering ini.

In [9]:
```
dataset.corr()
```

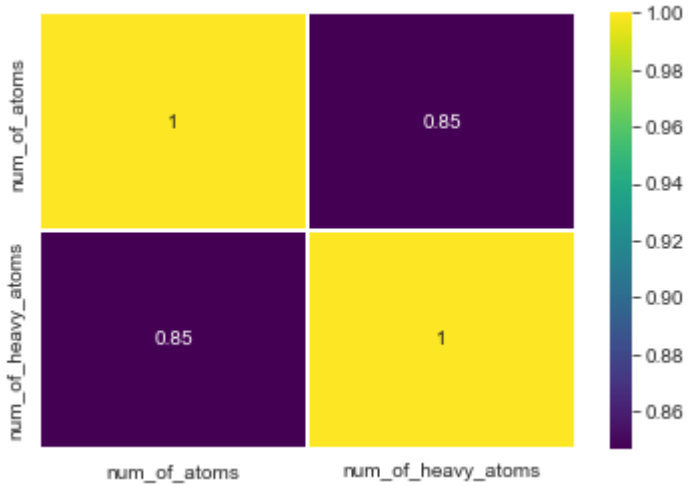
Out[9]:

	num_of_atoms	num_of_heavy_atoms
num_of_atoms	1.000000	0.846791
num_of_heavy_atoms	0.846791	1.000000

Untuk mempermudah, kita gunakan heatmap dan membandingkan korelasi dengan setiap kolom dengan warna.

In [10]:
```
plt.figure(figsize=(6,4))
sns.heatmap(dataset.corr(), annot=True, cmap='viridis', linewidths=.1)
```

```
plt.show()
```



Kedua feature ini berkorelasi positif dan tergolong kuat.

Checking Balance of Data

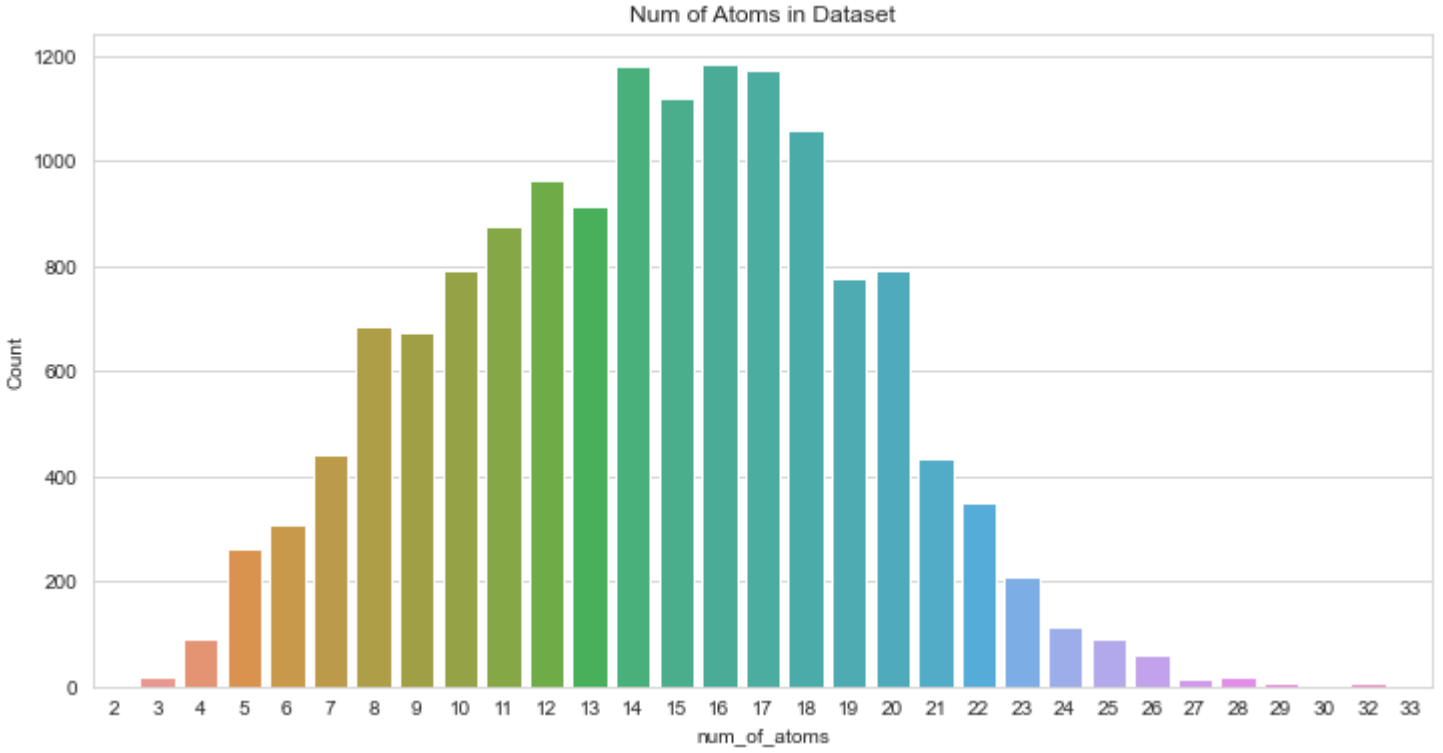
In [11]:
```
dataset['num_of_atoms'].value_counts().sort_values()
```

Out[11]:
```
2       1
33      1
30      3
29      5
32      8
27     14
3      18
28     19
26     59
4      92
25     92
24    112
23    208
5     262
6     308
22    349
21    435
7     442
9     672
8     683
```

```
19     776
10     792
20     792
11     874
13     915
12     961
18    1059
15    1120
17    1174
14    1181
16    1183
Name: num_of_atoms, dtype: int64
```

In [12]:

```
plt.figure(figsize=(12,6))
sns.countplot(x='num_of_atoms',data=dataset)
plt.title("Num of Atoms in Dataset")
plt.xlabel('num_of_atoms')
plt.ylabel('Count')
plt.show()
```

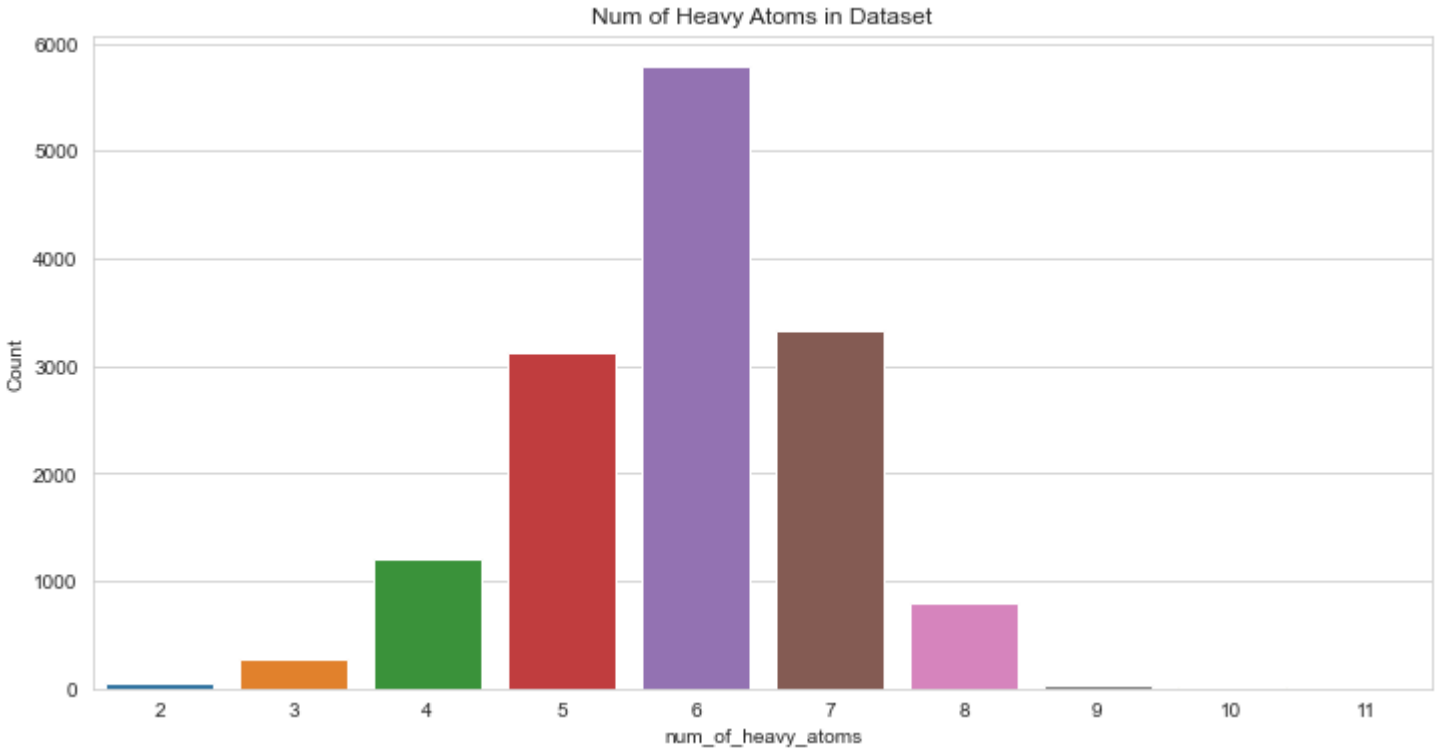


Grafik diatas menunjukkan bahwa data yang memiliki num_of_atoms dari 14-18 masing-masing memiliki count diatas 1000.

In [13]:
```
dataset['num_of_heavy_atoms'].value_counts().sort_values()
```

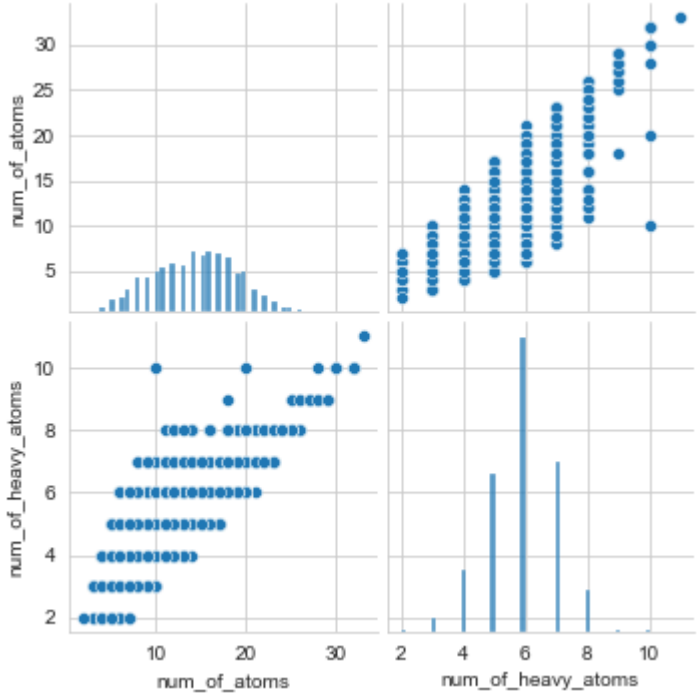
Out[13]:
```
11       1
10      14
9       42
2       43
3      276
8      801
4     1203
5     3120
7     3324
6     5786
Name: num_of_heavy_atoms, dtype: int64
```

In [14]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='num_of_heavy_atoms',data=dataset)
plt.title("Num of Heavy Atoms in Dataset")
plt.xlabel('num_of_heavy_atoms')
plt.ylabel('Count')
plt.show()
```



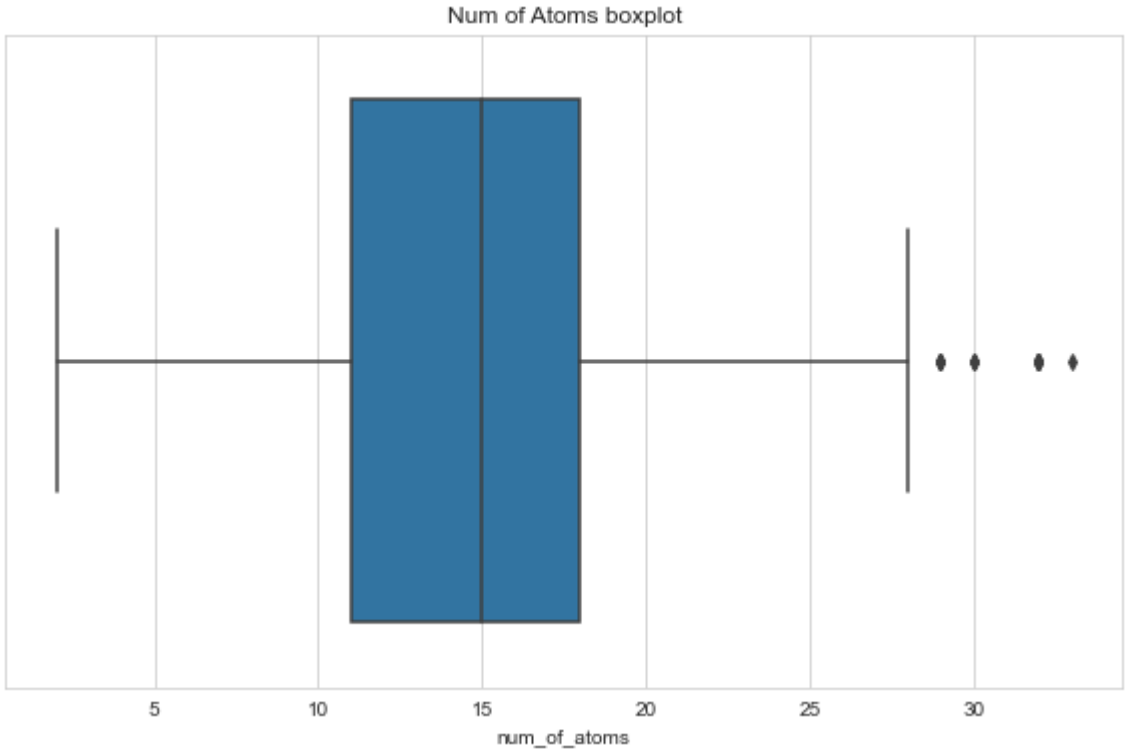
Grafik diatas menunjukkan bahwa num_of_heavy_atoms = 6 yang memiliki jumlah count terbanyak.

In [15]:
```
sns.pairplot(dataset)
plt.show()
```



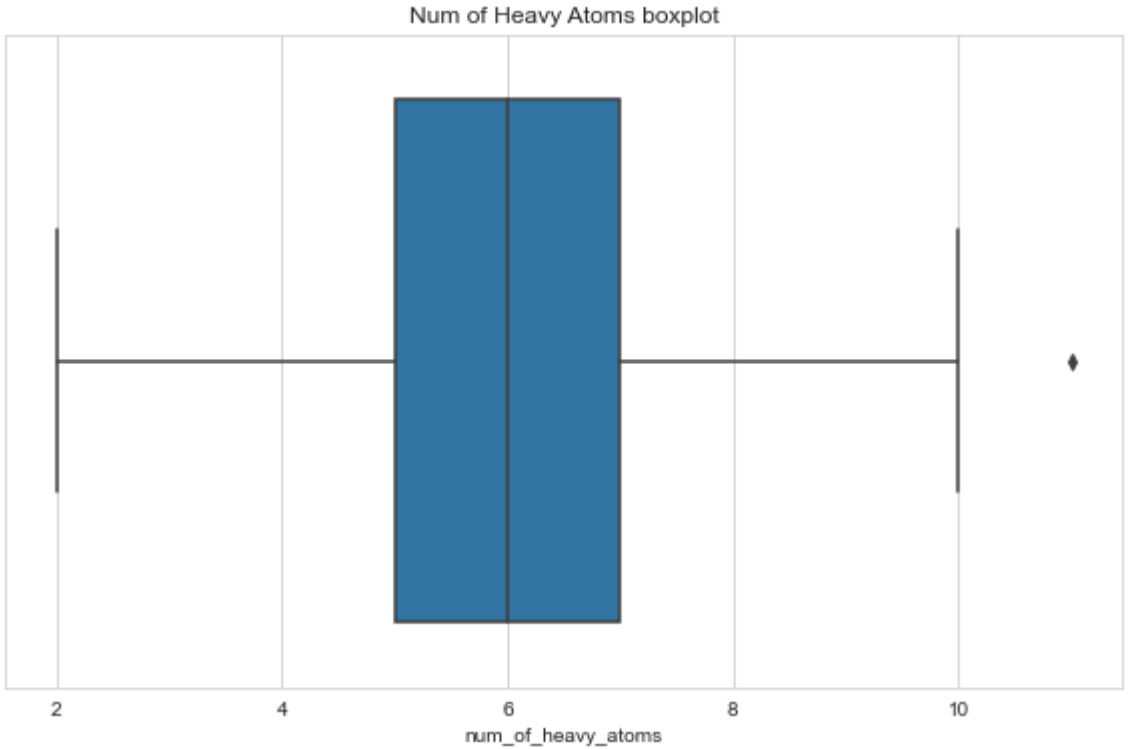
In [16]:

```
plt.figure(figsize=(10,6))
sns.boxplot(data = dataset, x = 'num_of_atoms')
plt.title('Num of Atoms boxplot')
plt.show()
```



In [17]:

```
plt.figure(figsize=(10,6))
sns.boxplot(data = dataset, x = 'num_of_heavy_atoms')
plt.title('Num of Heavy Atoms boxplot')
plt.show()
```



Defining Standard Scaler from Scratch

In [18]:
```
class StandardScaler():
    def __init__(self):
        pass

    def fit(self, X):
        self.mean_ = np.mean(X, axis=0)
        self.scale_ = np.std(X - self.mean_, axis=0)
        return self

    def transform(self, X):
        return (X - self.mean_) / self.scale_

    def fit_transform(self, X):
        return self.fit(X).transform(X)
```

Standardize the Variables

In [19]:
```
scaled_features = StandardScaler().fit_transform(dataset)
```

In [20]:
```
scaled_features
```

Out[20]:

	num_of_atoms	num_of_heavy_atoms
0	-0.087907	0.086838
1	-0.727083	-0.811041
2	0.125152	0.086838
3	1.403504	0.984717
4	1.190445	0.984717
...	...	...
14605	0.125152	0.086838
14606	-0.087907	0.086838
14607	0.764328	0.086838
14608	0.551269	0.086838
14609	-1.579317	-0.811041

14610 rows × 2 columns

In [21]:
```
X = scaled_features.to_numpy()
```

In [22]:
```
colors = 10*['magenta', 'blue', 'green', 'red']
```

Defining K-Means from Scratch

In [23]:
```
class K_Means:
    def __init__(self, k=4, tol=0.001, max_iter=300):
        #Tentukan nilai k
        self.k = k
        self.tol = tol
        self.max_iter = max_iter
```

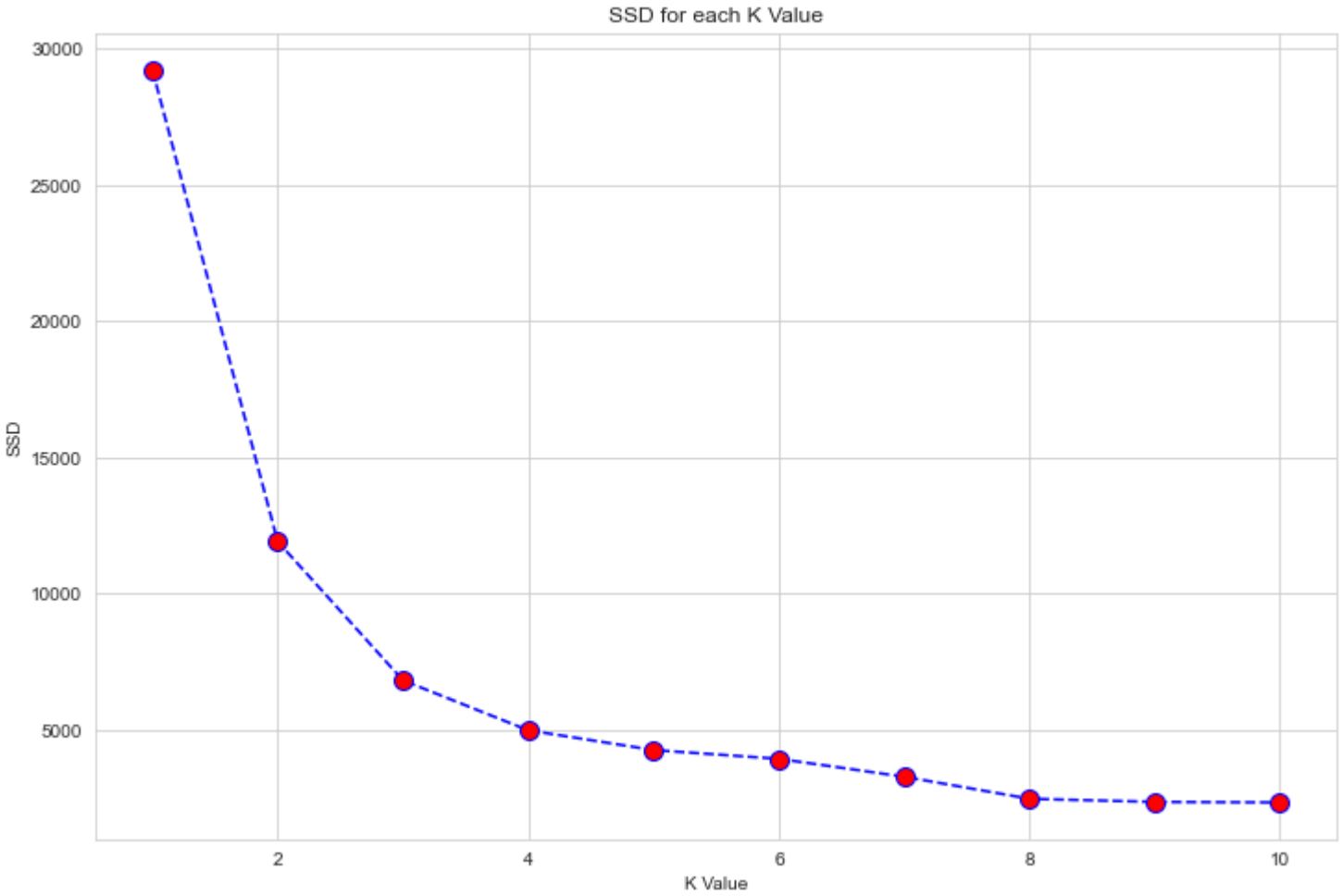
```
    def fit(self,data):
        self.centroids = {}

        #Pilih centroid secara acak
        rand_idx = np.random.randint(low = 0, high = dataset.shape[0]+1, size = self.k)

        for i in range(self.k):
            self.centroids[i] = data[rand_idx[i]]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            #Expectation: Cocokan setiap titik dengan centroid terdekatnya
            for featureset in data:
                distances = [np.linalg.norm(featureset-self.centroids[centroid]) for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classifications[classification].append(featureset)

            prev_centroids = dict(self.centroids)

            #Maximization: Hitung centroid baru (mean) dari setiap cluster
            for classification in self.classifications:
                self.centroids[classification] = np.mean(self.classifications[classification],axis=0)

            optimized = True

            for c in self.centroids:
                original_centroid = prev_centroids[c]
                current_centroid = self.centroids[c]
                if np.sum((current_centroid-original_centroid)/original_centroid*100.0) > self.tol:
                    optimized = False

            #Keluar dari loop jika posisi centroid sudah tidak berubah
            if optimized:
                break

    def predict(self,data):
        distances = [np.linalg.norm(data-self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification
```

Choosing a K Value

Kita mencari tahu berapa nilai K yang optimal untuk model KNN kita dengan menghitung sum of squared distances menggunakan elbow method.

In [24]:
```
ssd = []

for i in range(1, 11):
    kmeans = K_Means(k=i)
    kmeans.fit(X)
    curr_ssd = 0

    for featureset in X:
        distances = []
        for centroid in kmeans.centroids:
            distance = np.linalg.norm(featureset-kmeans.centroids[centroid])
            distances.append(distance)

        nearest_centroid_idx = distances.index(min(distances))

        curr_ssd += (featureset[0] - kmeans.centroids[nearest_centroid_idx][0]) ** 2 + (featureset[1] - kmeans.centroids[

    ssd.append(curr_ssd)
```

In [25]:
```
plt.figure(figsize=(12,8))
plt.plot(range(1, 11), ssd, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('SSD for each K Value')
plt.xlabel('K Value')
plt.ylabel('SSD')
plt.show()
```



Kita dapat melihat disini jika nilai ssd terbesar ketika K = 1 dan dengan bertambahnya jumlah cluster, nilai ssd akan mulai berkurang. Kita juga dapat melihat bahwa grafik akan berubah dengan cepat pada titik K = 4 dan menciptakan bentuk siku. Dari titik ini, grafik mulai bergerak hampir sejajar dengan sumbu X. Nilai K yang sesuai dengan titik ini adalah nilai K optimal atau jumlah cluster yang optimal.

In [26]:
```
abs(pd.Series(ssd).diff())
```

Out[26]:
```
0            NaN
1    17311.649100
2     5096.437846
3     1833.180753
4      724.903724
5      322.750703
```

```
6      656.051606
7      809.116151
8      115.108519
9       20.239199
dtype: float64
```

Apabila secara spesifik dihitung perbedaannya, terlihat bahwa:

- titik 1 ke 2 berjarak 17311 dan kemiringannya masih curam.
- titik 2 ke 3 berjarak 5096 dan kemiringannya kurang curam.
- titik 3 ke 4 berjarak 1833 dan kemiringannya mulai stabil disini.

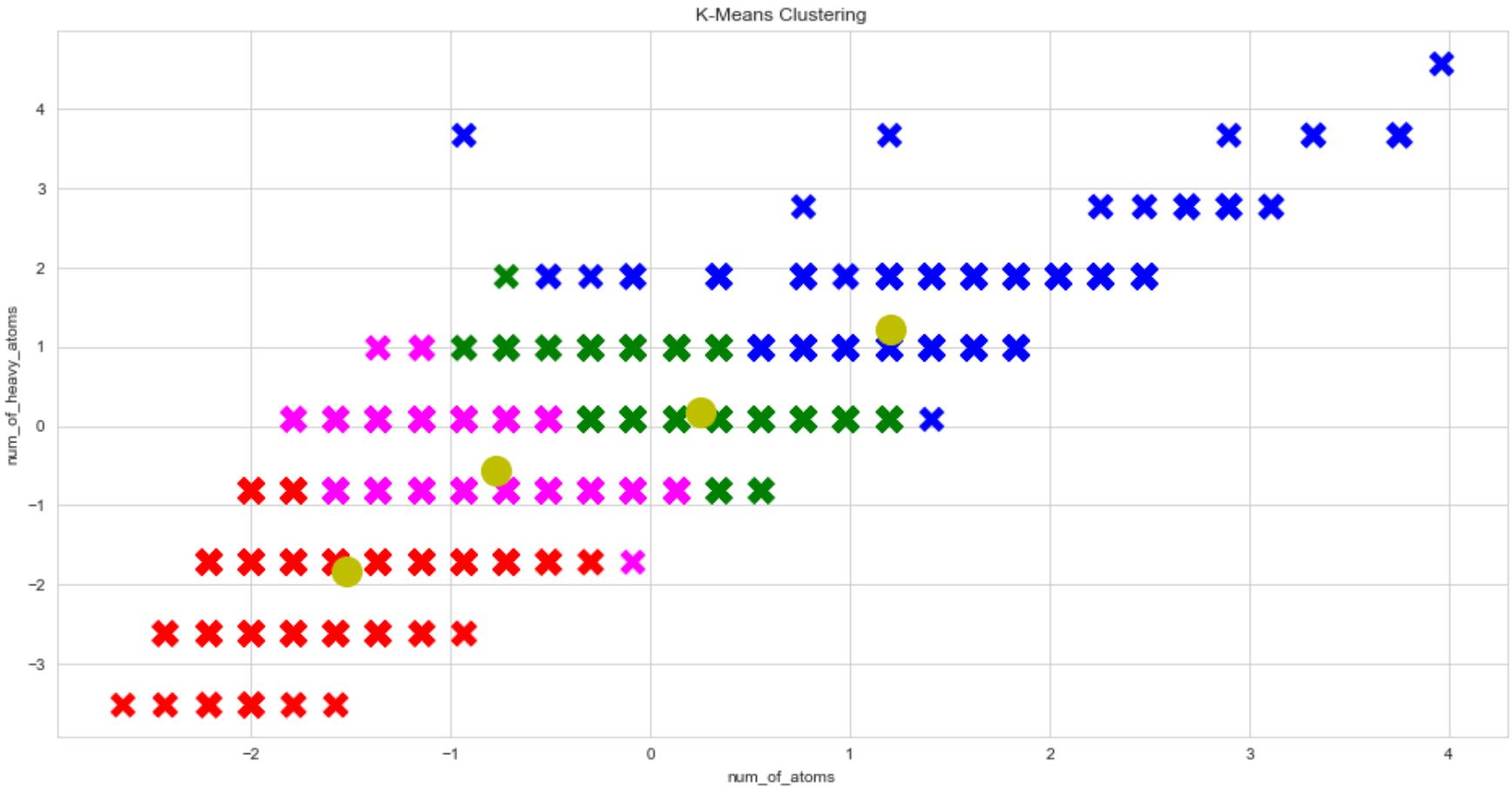
Clustering

In [27]:
```
kmeans = K_Means(k = 4)
kmeans.fit(X)
```

In [28]:
```
kmeans.centroids
```

Out[28]:
```
{0: array([-0.77292865, -0.55696193]),
 1: array([1.20383055, 1.21355949]),
 2: array([0.24855955, 0.16982226]),
 3: array([-1.52298681, -1.82465436])}
```

In [29]:
```
plt.figure(figsize=(16,8))

for classification in kmeans.classifications:
    color = colors[classification]
    for featureset in kmeans.classifications[classification]:
        plt.scatter(featureset[0], featureset[1], marker="x", color=color, s=150, linewidths=5)

for centroid in kmeans.centroids:
    plt.scatter(kmeans.centroids[centroid][0], kmeans.centroids[centroid][1], marker="o", color="y", s=200, linewidths=5)

plt.title('K-Means Clustering')
plt.xlabel('num_of_atoms')
plt.ylabel('num_of_heavy_atoms')
plt.show()
```



Terima kasih.

2440055351, Cindy Amanda Onggirawan, LB01, Final Exam COMP6745001- Machine Learning

Nomor 3: Classification

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as stats
sns.set_style('whitegrid')
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Import Dataset

In [3]:
```
dataset = pd.read_csv('./dataset/experiment.csv')
```

Exploratory Data Analysis (EDA) and Statistical Analysis

Kita melihat gambaran umum dataframe kita terlebih dahulu.

In [4]:
```
dataset.head()
```

Out[4]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F192	F193	F194	F195	F196
0	0.006711	0.0	0.013423	0.006711	0.0	0.006711	0.006711	0.020134	0.020134	0.000000	...	0.013423	0.000000	0.006711	0.013423	0.020134
1	0.000000	0.0	0.000000	0.007246	0.0	0.000000	0.000000	0.021739	0.014493	0.000000	...	0.007246	0.007246	0.000000	0.000000	0.014493
2	0.011696	0.0	0.005848	0.000000	0.0	0.005848	0.000000	0.035088	0.017544	0.017544	...	0.005848	0.000000	0.005848	0.011696	0.035088
3	0.000000	0.0	0.020833	0.000000	0.0	0.000000	0.010417	0.000000	0.020833	0.000000	...	0.010417	0.000000	0.000000	0.041667	0.000000

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F192	F193	F194	F195	F196
4	0.000000	0.0	0.034483	0.000000	0.0	0.000000	0.000000	0.000000	0.034483	0.000000	...	0.017241	0.000000	0.000000	0.068966	0.000000

5 rows × 201 columns

Kita mengidentikasi data types, columns names, null value counts, dan memory usage.

In [5]:
```
dataset.shape
```

Out[5]:
```
(4080, 201)
```

In [6]:
```
dataset.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4080 entries, 0 to 4079
Data columns (total 201 columns):
 #    Column  Dtype  
---   ------  -----  
 0    F1      float64
 1    F2      float64
 2    F3      float64
 3    F4      float64
 4    F5      float64
 5    F6      float64
 6    F7      float64
 7    F8      float64
 8    F9      float64
 9    F10     float64
 10   F11     float64
 11   F12     float64
 12   F13     float64
 13   F14     float64
 14   F15     float64
 15   F16     float64
 16   F17     float64
 17   F18     float64
 18   F19     float64
 19   F20     float64
 20   F21     float64
 21   F22     float64
```

```
22   F23     float64
23   F24     float64
24   F25     float64
25   F26     float64
26   F27     float64
27   F28     float64
28   F29     float64
29   F30     float64
30   F31     float64
31   F32     float64
32   F33     float64
33   F34     float64
34   F35     float64
35   F36     float64
36   F37     float64
37   F38     float64
38   F39     float64
39   F40     float64
40   F41     float64
41   F42     float64
42   F43     float64
43   F44     float64
44   F45     float64
45   F46     float64
46   F47     float64
47   F48     float64
48   F49     float64
49   F50     float64
50   F51     float64
51   F52     float64
52   F53     float64
53   F54     float64
54   F55     float64
55   F56     float64
56   F57     float64
57   F58     float64
58   F59     float64
59   F60     float64
60   F61     float64
61   F62     float64
62   F63     float64
63   F64     float64
64   F65     float64
65   F66     float64
66   F67     float64
```

```
67   F68     float64
68   F69     float64
69   F70     float64
70   F71     float64
71   F72     float64
72   F73     float64
73   F74     float64
74   F75     float64
75   F76     float64
76   F77     float64
77   F78     float64
78   F79     float64
79   F80     float64
80   F81     float64
81   F82     float64
82   F83     float64
83   F84     float64
84   F85     float64
85   F86     float64
86   F87     float64
87   F88     float64
88   F89     float64
89   F90     float64
90   F91     float64
91   F92     float64
92   F93     float64
93   F94     float64
94   F95     float64
95   F96     float64
96   F97     float64
97   F98     float64
98   F99     float64
99   F100    float64
100  F101    float64
101  F102    float64
102  F103    float64
103  F104    float64
104  F105    float64
105  F106    float64
106  F107    float64
107  F108    float64
108  F109    float64
109  F110    float64
110  F111    float64
111  F112    float64
```

```
112  F113    float64
113  F114    float64
114  F115    float64
115  F116    float64
116  F117    float64
117  F118    float64
118  F119    float64
119  F120    float64
120  F121    float64
121  F122    float64
122  F123    float64
123  F124    float64
124  F125    float64
125  F126    float64
126  F127    float64
127  F128    float64
128  F129    float64
129  F130    float64
130  F131    float64
131  F132    float64
132  F133    float64
133  F134    float64
134  F135    float64
135  F136    float64
136  F137    float64
137  F138    float64
138  F139    float64
139  F140    float64
140  F141    float64
141  F142    float64
142  F143    float64
143  F144    float64
144  F145    float64
145  F146    float64
146  F147    float64
147  F148    float64
148  F149    float64
149  F150    float64
150  F151    float64
151  F152    float64
152  F153    float64
153  F154    float64
154  F155    float64
155  F156    float64
156  F157    float64
```

```
157  F158    float64
158  F159    float64
159  F160    float64
160  F161    float64
161  F162    float64
162  F163    float64
163  F164    float64
164  F165    float64
165  F166    float64
166  F167    float64
167  F168    float64
168  F169    float64
169  F170    float64
170  F171    float64
171  F172    float64
172  F173    float64
173  F174    float64
174  F175    float64
175  F176    float64
176  F177    float64
177  F178    float64
178  F179    float64
179  F180    float64
180  F181    float64
181  F182    float64
182  F183    float64
183  F184    float64
184  F185    float64
185  F186    float64
186  F187    float64
187  F188    float64
188  F189    float64
189  F190    float64
190  F191    float64
191  F192    float64
192  F193    float64
193  F194    float64
194  F195    float64
195  F196    float64
196  F197    float64
197  F198    float64
198  F199    float64
199  F200    float64
200  Class   int64
```

```
dtypes: float64(200), int64(1)
memory usage: 6.3 MB
```

Kita berusaha melihat detail dari setiap feature.

In [7]:
```
dataset.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
F1	4080.0	0.007245	0.012460	0.0	0.0	0.0	0.012500	0.136364
F2	4080.0	0.005094	0.008867	0.0	0.0	0.0	0.009901	0.076923
F3	4080.0	0.003749	0.008527	0.0	0.0	0.0	0.000000	0.150000
F4	4080.0	0.004814	0.008642	0.0	0.0	0.0	0.009009	0.100000
F5	4080.0	0.004394	0.008072	0.0	0.0	0.0	0.008282	0.058824
...	...	...	...	...	...	...	...	...
F197	4080.0	0.003831	0.007681	0.0	0.0	0.0	0.005723	0.125000
F198	4080.0	0.003890	0.007581	0.0	0.0	0.0	0.006499	0.075000
F199	4080.0	0.004026	0.007857	0.0	0.0	0.0	0.006909	0.050000
F200	4080.0	0.004319	0.008777	0.0	0.0	0.0	0.005579	0.064103
Class	4080.0	3.000000	1.414387	1.0	2.0	3.0	4.000000	5.000000

201 rows × 8 columns

Missing Value Treatment

Dengan menganalisis detail feature di atas, kita merasa bahwa kedua feature memiliki kisaran nilai yang masuk akal.

In [8]:
```
dataset.isnull().sum()
```

Out[8]:
```
F1       0
F2       0
F3       0
F4       0
F5       0
        ..
F197     0
```

```
F198     0
F199     0
F200     0
Class    0
Length: 201, dtype: int64
```

Kita tahu bahwa tidak ada feature yang bernilai kosong, berarti tidak ada yang perlu diisi.

In [9]:
```
dataset[dataset == 0].count()
```

Out[9]:
```
F1       2567
F2       2805
F3       3126
F4       2855
F5       2919
         ...
F197     3054
F198     3034
F199     3016
F200     3057
Class       0
Length: 201, dtype: int64
```

Namun, terdapat banyak nilai 0 yang tersebar di berbagai feature.

In [10]:
```
def percent_equal_to_zero(dataset):
    percent_zero = 100 * dataset[dataset == 0].count() / len(dataset)
    percent_zero = percent_zero[percent_zero > 0].sort_values()

    return percent_zero
```

In [11]:
```
percent_zero = percent_equal_to_zero(dataset)
percent_zero
```

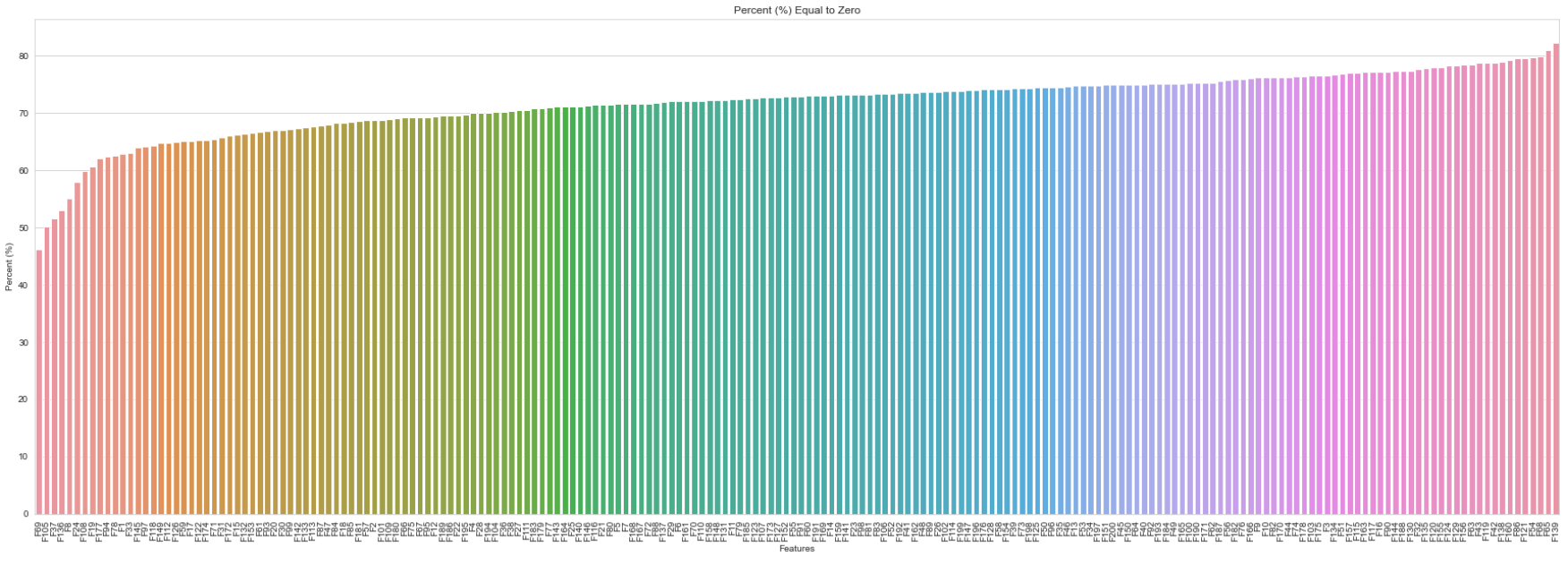
Out[11]:
```
F69     46.127451
F105    50.122549
F37     51.666667
F136    53.112745
F8      55.147059
           ...
F121    79.607843
F54     79.656863
F68     79.901961
```

```
F65     81.004902
F139    82.254902
Length: 200, dtype: float64
```

Semua features memiliki data yang bernilai 0, bahkan ada yang mencapai 80% dari keseluruhan data dalam kolom tersebut

In [12]:

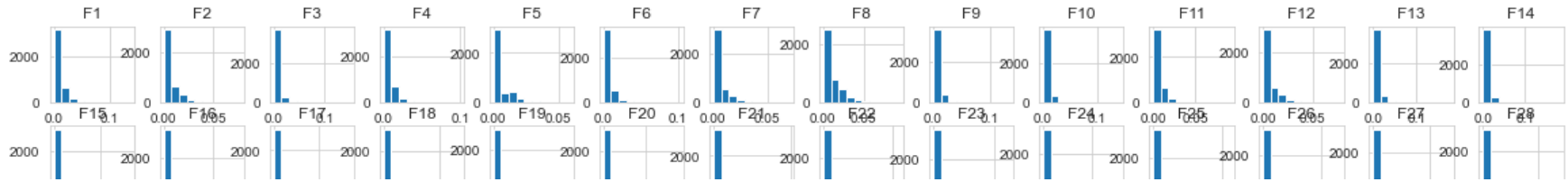
```
plt.figure(figsize=(30, 10))
sns.barplot(x=percent_zero.index,y=percent_zero)
plt.title('Percent (%) Equal to Zero')
plt.xlabel('Features')
plt.ylabel('Percent (%)')
plt.xticks(rotation=90);
```

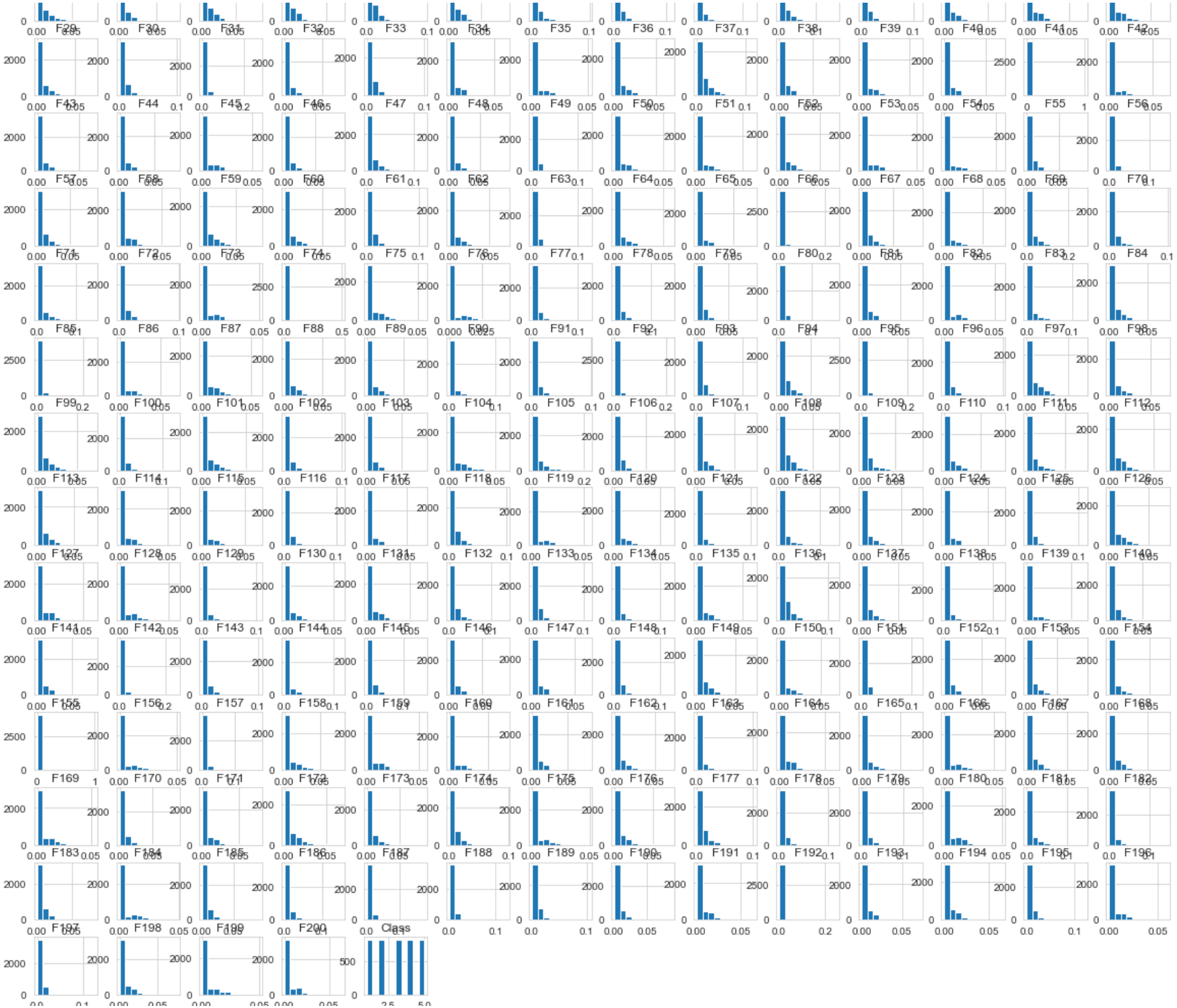


Understand Data Distribution

In [13]:

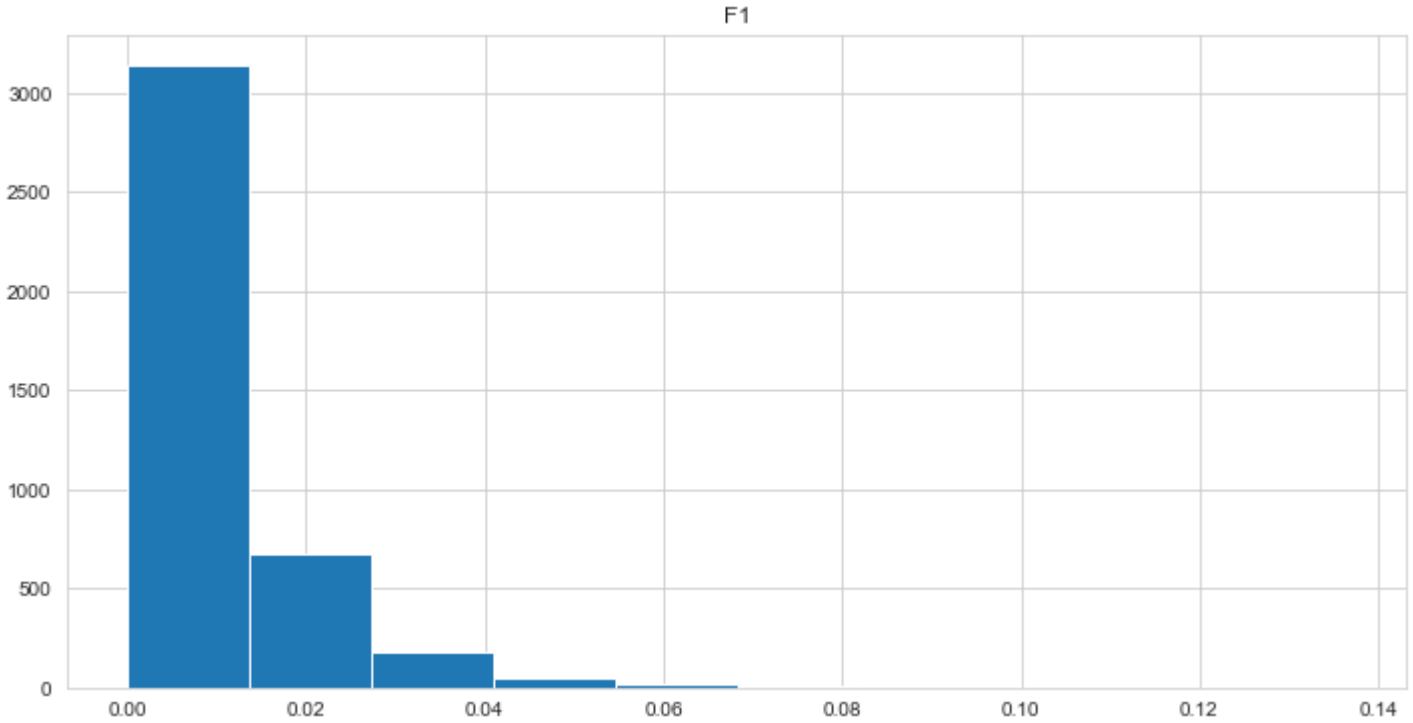
```
dataset.hist(figsize = (20,20))
plt.show()
```




F29
F30
F31
F32
F33
F34
F35
F36
F37
F38
F39
F40
F41
F42
F43
F44
F45
F46
F47
F48
F49
F50
F51
F52
F53
F54
F55
F56
F57
F58
F59
F60
F61
F62
F63
F64
F65
F66
F67
F68
F69
F70
F71
F72
F73
F74
F75
F76
F77
F78
F79
F80
F81
F82
F83
F84
F85
F86
F87
F88
F89
F90
F91
F92
F93
F94
F95
F96
F97
F98
F99
F100
F101
F102
F103
F104
F105
F106
F107
F108
F109
F110
F111
F112
F113
F114
F115
F116
F117
F118
F119
F120
F121
F122
F123
F124
F125
F126
F127
F128
F129
F130
F131
F132
F133
F134
F135
F136
F137
F138
F139
F140
F141
F142
F143
F144
F145
F146
F147
F148
F149
F150
F151
F152
F153
F154
F155
F156
F157
F158
F159
F160
F161
F162
F163
F164
F165
F166
F167
F168
F169
F170
F171
F172
F173
F174
F175
F176
F177
F178
F179
F180
F181
F182
F183
F184
F185
F186
F187
F188
F189
F190
F191
F192
F193
F194
F195
F196
F197
F198
F199
F200
Class

Karena 200 fitur memiliki pola data yang sama, maka kita cukup melakukan generalisasi dengan memvisualisasikan fitur pertama, yaitu F1.

In [14]:
```
dataset['F1'].hist(figsize = (12,6))
plt.title('F1')
plt.show()
```



Grafik diatas menunjukkan bahwa sebagian besar data F1 hampir mendekati 0 dan berbentuk positive skewed.

Setelah mencoba untuk menampilkan visualisasi feature, sekarang kita coba membuat grafik untuk 1 baris data tertentu dan mengeneralisasi 4079 baris lainnya.

In [15]:
```
single_data = dataset.drop('Class', axis=1).iloc[0]
```

In [16]:
```
single_data
```

Out[16]:
```
F1      0.006711
F2      0.000000
```

```
F3      0.013423
F4      0.006711
F5      0.000000
          ...
F196    0.020134
F197    0.006711
F198    0.000000
F199    0.006711
F200    0.000000
Name: 0, Length: 200, dtype: float64
```

In [17]:
```
single_data.to_numpy().shape
```

Out[17]:
```
(200,)
```

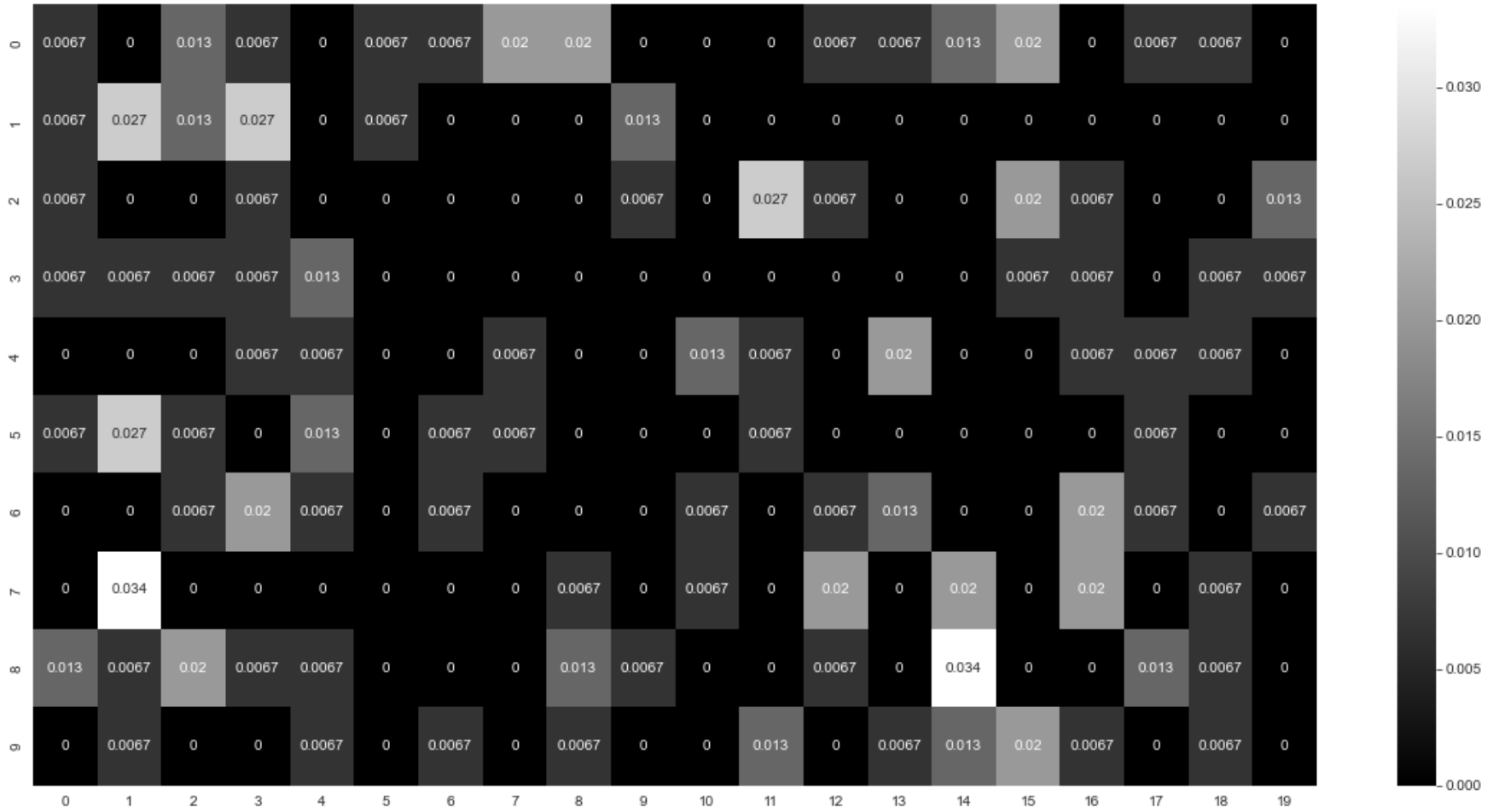
In [18]:
```
single_data.to_numpy().reshape(10,20)
```

Out[18]:
```
array([[0.00671141, 0.        , 0.01342282, 0.00671141, 0.        ,
        0.00671141, 0.00671141, 0.02013423, 0.02013423, 0.        ,
        0.        , 0.        , 0.00671141, 0.00671141, 0.01342282,
        0.02013423, 0.        , 0.00671141, 0.00671141, 0.        ],
       [0.00671141, 0.02684564, 0.01342282, 0.02684564, 0.        ,
        0.00671141, 0.        , 0.        , 0.        , 0.01342282,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.00671141, 0.        , 0.        , 0.00671141, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.00671141,
        0.        , 0.02684564, 0.00671141, 0.        , 0.        ,
        0.02013423, 0.00671141, 0.        , 0.        , 0.01342282],
       [0.00671141, 0.00671141, 0.00671141, 0.00671141, 0.01342282,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.00671141, 0.00671141, 0.        , 0.00671141, 0.00671141],
       [0.        , 0.        , 0.        , 0.00671141, 0.00671141,
        0.        , 0.        , 0.00671141, 0.        , 0.        ,
        0.01342282, 0.00671141, 0.        , 0.02013423, 0.        ,
        0.        , 0.00671141, 0.00671141, 0.00671141, 0.        ],
       [0.00671141, 0.02684564, 0.00671141, 0.        , 0.01342282,
        0.        , 0.00671141, 0.00671141, 0.        , 0.        ,
        0.        , 0.00671141, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.00671141, 0.        , 0.        ],
       [0.        , 0.        , 0.00671141, 0.02013423, 0.00671141,
        0.        , 0.00671141, 0.        , 0.        , 0.        ,
```

```
 0.00671141, 0.        , 0.00671141, 0.01342282, 0.        ,
 0.        , 0.02013423, 0.00671141, 0.        , 0.00671141],
[0.        , 0.03355705, 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.00671141, 0.        ,
 0.00671141, 0.        , 0.02013423, 0.        , 0.02013423,
 0.        , 0.02013423, 0.        , 0.00671141, 0.        ],
[0.01342282, 0.00671141, 0.02013423, 0.00671141, 0.00671141,
 0.        , 0.        , 0.        , 0.01342282, 0.00671141,
 0.        , 0.        , 0.00671141, 0.        , 0.03355705,
 0.        , 0.        , 0.01342282, 0.00671141, 0.        ],
[0.        , 0.00671141, 0.        , 0.        , 0.00671141,
 0.        , 0.00671141, 0.        , 0.00671141, 0.        ,
 0.        , 0.01342282, 0.        , 0.00671141, 0.01342282,
 0.02013423, 0.00671141, 0.        , 0.00671141, 0.        ]])
```

In [19]:

```
plt.figure(figsize=(20,10))
sns.heatmap(single_data.to_numpy().reshape(10,20),annot=True,cmap='gray')
plt.show()
```



Informasi yang kita peroleh sama dengan grafik batang sebelumnya, yaitu setiap data memuat banyak feature yang bernilai 0, yang ditandai dengan warna hitam.

Correlation

Kita perlu menghitung correlation antara independent variable dan dependent variable.

In [20]:
```
dataset.corr()
```

Out[20]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F192	F193	
F1	1.000000	0.062642	-0.033671	0.009647	0.057143	-0.064946	-0.034114	-0.045268	-0.029826	-4.264591e-02	...	-0.075487	-0.029067	0.(

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F192	F193	
F2	0.062642	1.000000	-0.068749	0.021577	0.076734	0.025676	-0.017032	-0.006521	-0.030620	-2.652480e-02	...	-0.024747	-0.009527	0.(
F3	-0.033671	-0.068749	1.000000	-0.050458	-0.035755	-0.034025	-0.049696	-0.088605	0.085910	-1.317400e-02	...	0.177315	-0.007673	-0.(
F4	0.009647	0.021577	-0.050458	1.000000	0.025955	0.024615	0.011098	-0.003480	-0.045377	3.094654e-02	...	-0.058337	0.001371	0.(
F5	0.057143	0.076734	-0.035755	0.025955	1.000000	0.161424	0.058923	-0.006961	0.055993	-4.006644e-02	...	-0.027651	-0.057030	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
F197	-0.027931	-0.010725	0.086427	0.021453	-0.055358	-0.053639	-0.005306	-0.061777	0.014667	1.479263e-02	...	0.082560	-0.022100	-0.(
F198	-0.028397	0.083870	-0.053308	-0.014131	0.019856	0.099709	-0.006233	-0.084724	-0.053033	-3.131133e-02	...	-0.020317	0.050680	-0.(
F199	-0.007867	-0.033720	0.096942	-0.004262	-0.017213	0.007877	-0.044835	-0.081779	0.128999	2.835366e-07	...	0.100440	-0.052859	-0.(
F200	0.021292	0.051830	-0.035988	0.062791	0.004151	0.015164	-0.028651	-0.022038	-0.053442	-5.626099e-02	...	0.001951	-0.074896	0.(
Class	0.026725	0.054789	-0.037185	0.111382	0.046696	-0.121490	0.025376	0.011915	-0.035052	-9.831682e-02	...	0.015134	-0.011522	-0.(

201 rows × 201 columns

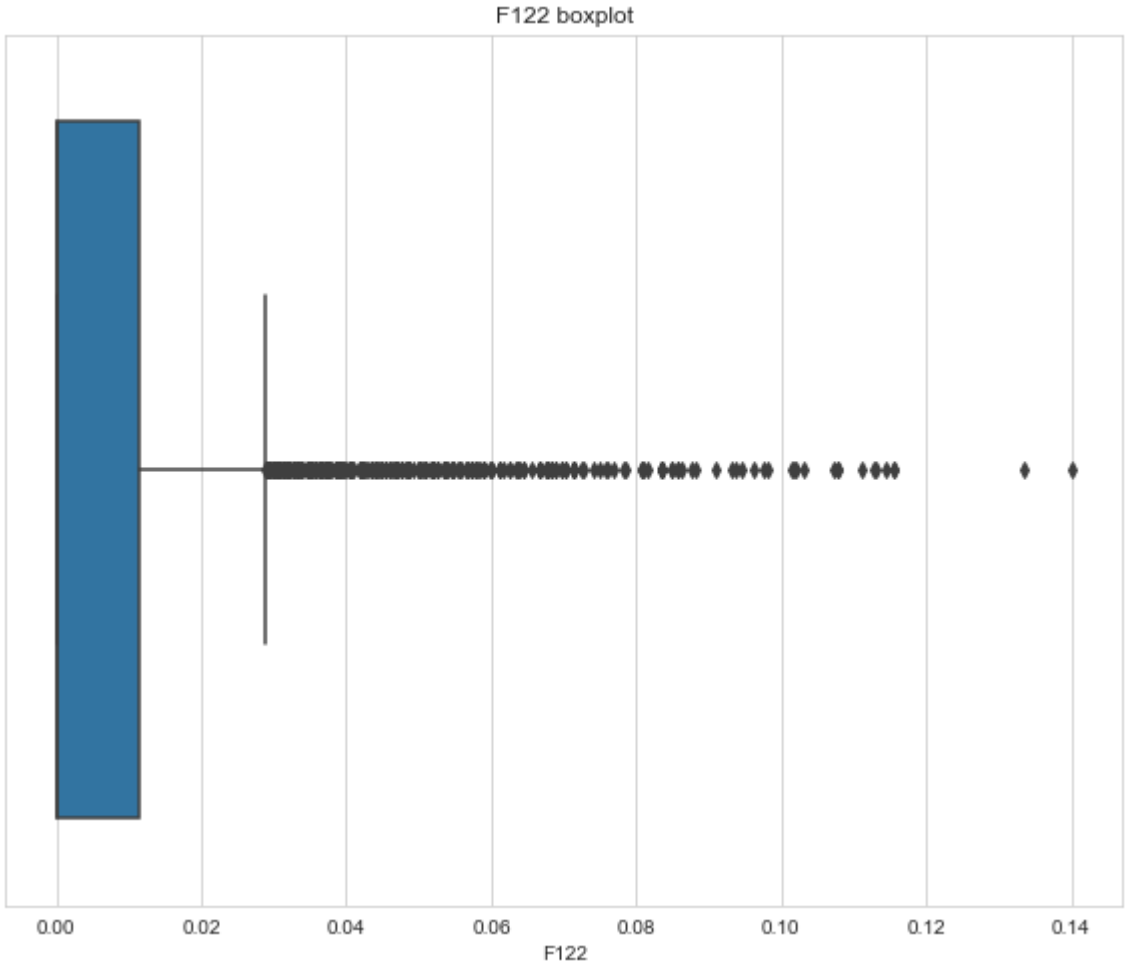
In [21]:
```
dataset.corr()['Class'].sort_values()
```

Out[21]:
```
F157    -0.214910
F65     -0.185885
F80     -0.164355
F13     -0.160201
F15     -0.152490
           ...
F114     0.143675
F12      0.164925
F200     0.188194
F122     0.203490
```

```
Class    1.000000
Name: Class, Length: 201, dtype: float64
```

Kita tidak dapat menggunakan heatmap karena terlalu banyak feature menyebabkan grafik juga sulit dipahami. Tetapi disini, kita memperoleh informasi bahwa correlation tertinggi antara independent variable dan dependent variable dipegang oleh feature F122.

In [22]:
```
plt.figure(figsize=(10,8))
sns.boxplot(data = dataset, x = 'F122')
plt.title('F122 boxplot')
plt.show()
```



In [23]:
```
dataset['F122'].describe().T
```

Out[23]:
```
count    4080.000000
mean        0.008230
std         0.016096
min         0.000000
25%         0.000000
50%         0.000000
75%         0.011528
max         0.140000
Name: F122, dtype: float64
```

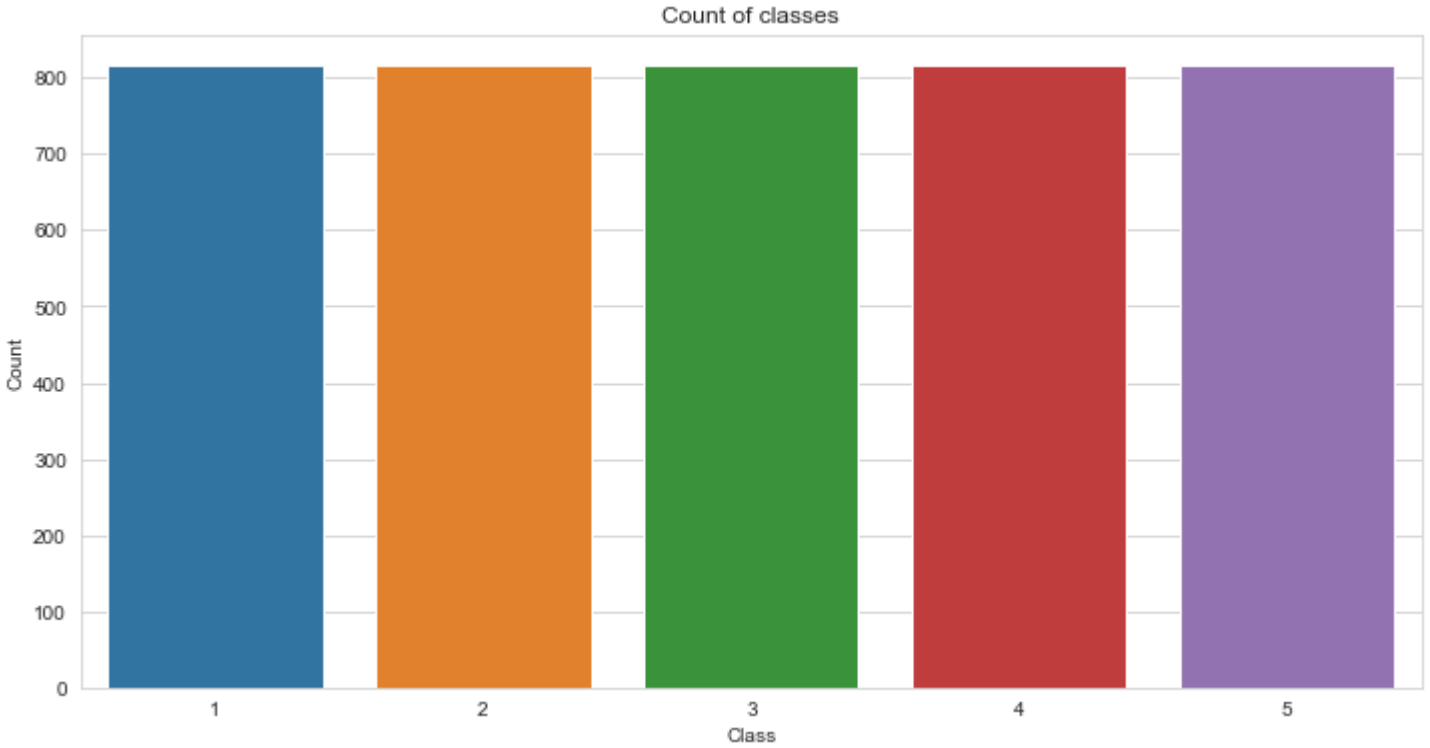
Kita melihat banyak sekali outlier. Ini disebabkan oleh sebagian besar data di feature F122 bernilai 0. Sehingga, Q1, Q2, dan Q3-nya pun mendekati 0. Hal ini juga menyebabkan nilai IQR yang kecil.

Checking Balance of Data

In [24]:
```
dataset['Class'].value_counts()
```

Out[24]:
```
1    816
2    816
3    816
4    816
5    816
Name: Class, dtype: int64
```

In [25]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='Class',data=dataset)
plt.title("Count of classes")
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Grafik diatas menunjukkan bahwa datanya terbagi dengan merata karena untuk setiap class, ada sebanyak 816 data.

Standarization using StandardScaler

Disini, kita menggunakan standarisasi, bukan normalisasi karena:

- Normalisasi digunakan ketika feature memiliki skala yang berbeda, sedangkan dataset kita memiliki skala yang sama yaitu mendekati 0 dan tidak jauh berbeda satu sama lain.
- Standardisasi tidak terpengaruh oleh outlier karena tidak ada rentang feature yang diubah sebelumnya, sedangkan normalisasi sangat dipengaruhi oleh outlier karena nilai minimum dan maksimum feature digunakan untuk penskalaan.

In [26]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [27]:
```
X = dataset.drop('Class', axis=1)
```

In [28]:
```
```

```
X.head()
```

Out[28]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F191	F192	F193	F194	F195
0	0.006711	0.0	0.013423	0.006711	0.0	0.006711	0.006711	0.020134	0.020134	0.000000	...	0.000000	0.013423	0.000000	0.006711	0.013423
1	0.000000	0.0	0.000000	0.007246	0.0	0.000000	0.000000	0.021739	0.014493	0.000000	...	0.000000	0.007246	0.007246	0.000000	0.000000
2	0.011696	0.0	0.005848	0.000000	0.0	0.005848	0.000000	0.035088	0.017544	0.017544	...	0.005848	0.005848	0.000000	0.005848	0.011696
3	0.000000	0.0	0.020833	0.000000	0.0	0.000000	0.010417	0.000000	0.020833	0.000000	...	0.000000	0.010417	0.000000	0.000000	0.041667
4	0.000000	0.0	0.034483	0.000000	0.0	0.000000	0.000000	0.000000	0.034483	0.000000	...	0.017241	0.017241	0.000000	0.000000	0.068966

5 rows × 200 columns

In [29]:
```
y = dataset['Class']
```

In [30]:
```
y.head()
```

Out[30]:
```
0    1
1    1
2    1
3    1
4    1
Name: Class, dtype: int64
```

In [31]:
```
scaled_X = scaler.fit_transform(X)
```

In [32]:
```
scaled_X
```

Out[32]:
```
array([[-0.0427896 , -0.57454216,  1.13461489, ..., -0.51316937,
         0.34179386, -0.49211189],
       [-0.58148916, -0.57454216, -0.43974027, ..., -0.51316937,
         0.40989316, -0.49211189],
       [ 0.35729731, -0.57454216,  0.24616297, ..., -0.51316937,
        -0.51254179, -0.49211189],
       ...,
       [ 0.6165144 , -0.57454216, -0.43974027, ...,  1.45594986,
```

```
 -0.51254179,  2.90943485],
[-0.58148916,  1.40425033,  1.61796956, ...,  1.80140944,
 -0.51254179,  3.50619755],
[ 0.80241157, -0.57454216, -0.43974027, ...,  1.76150295,
 -0.51254179,  3.43726126]])
```

Feature Selection using Principal Component Analysis (PCA)

Disini, kita menggunakan PCA karena:

- PCA dapat digunakan ketika dimensi dari feature itu tinggi, contohnya dataset kita terdiri dari 200 independent variables dan 1 dependent variables.
- PCA sangat berguna dalam memproses data di mana terdapat multikolinieritas antara features. Hal ini ditunjukkan dengan korelasi positif dan negatif yang sudah kita hitung sebelumnya.
- PCA juga dapat digunakan untuk denoising dan kompresi data.

Kita menggunakan 2 komponen saja agar mudah divisualisasikan.

In [33]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
```

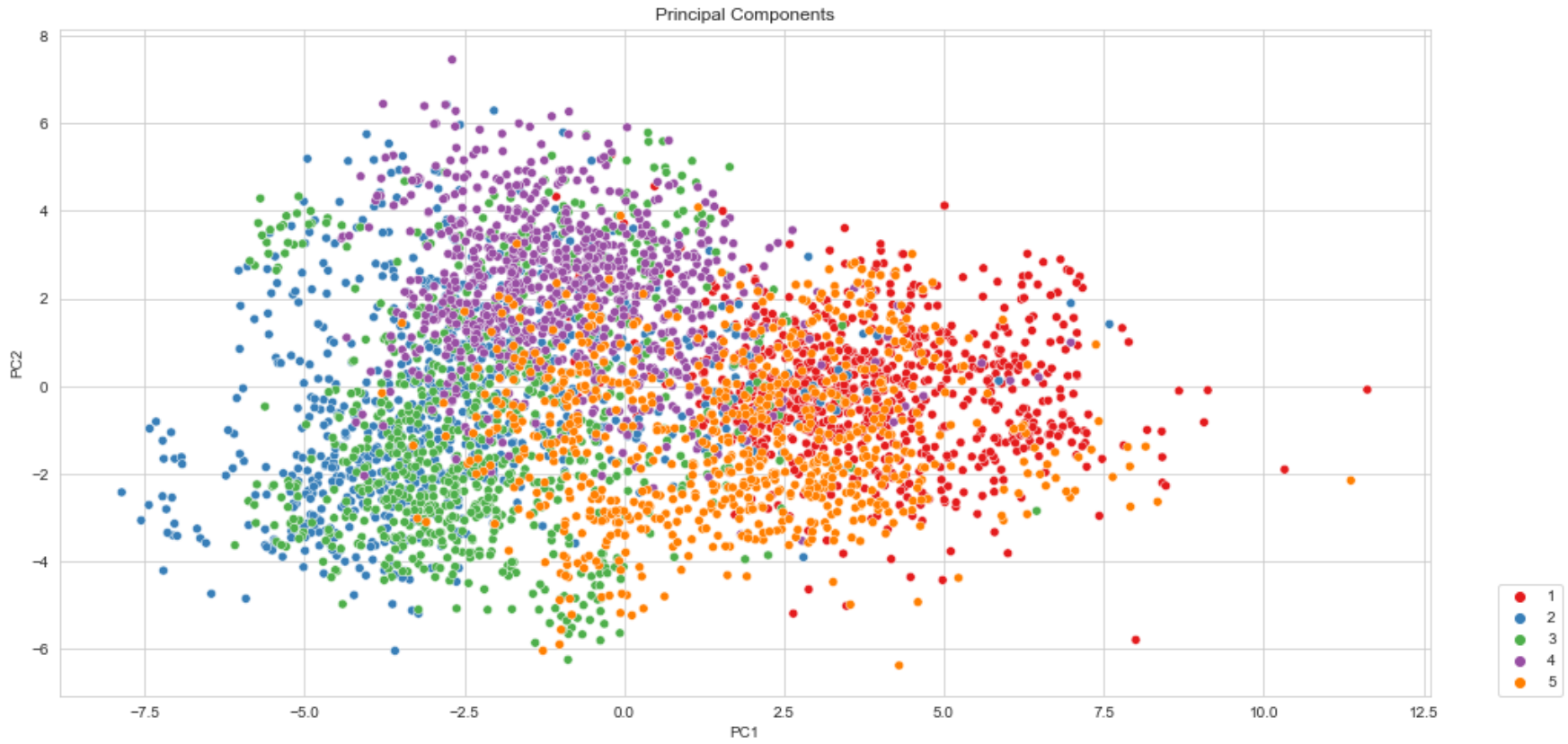
In [34]:
```
principal_components = pca.fit_transform(scaled_X)
```

In [35]:
```
principal_components
```

Out[35]:
```
array([[4.59802335, 0.9515533 ],
       [1.57196201, 2.01716509],
       [2.83304128, 0.81440737],
       ...,
       [2.79146941, 1.97687851],
       [3.69278394, 1.9135366 ],
       [3.81918682, 2.33544962]])
```

In [36]:
```
plt.figure(figsize=(16,8))
labels = dataset['Class'].values
sns.scatterplot(x = principal_components[:,0], y = principal_components[:,1], hue = labels, palette = 'Set1')
plt.legend(loc=(1.05,0))
plt.title('Principal Components')
plt.xlabel('PC1')
```

```
plt.ylabel('PC2')
plt.show()
```



Dibawah ini adalah dataframe yang menjelaskan arah variance maksimum pada dataset dan diurutkan berdasarkan explained variance.

In [37]:
```
dataset_comp = pd.DataFrame(pca.components_,index=['PC1','PC2'],columns=X.columns)
```

In [38]:
```
dataset_comp
```

Out[38]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F191	F192	F1
PC1	-0.039987	-0.052769	0.127041	-0.048936	-0.034778	-0.005335	-0.050461	-0.048827	0.074149	0.048702	...	-0.061166	0.117490	-0.0163
PC2	-0.001938	-0.110866	-0.040590	-0.057303	-0.050473	-0.063588	0.003232	0.052532	-0.021379	0.010466	...	-0.057339	-0.098545	0.0056

2 rows × 200 columns

In [39]:
```
pca.explained_variance_ratio_
```

Out[39]:
```
array([0.0497773 , 0.02675805])
```

Kita dapat melihat bahwa di ruang PCA, PC1 menjelaskan 4,9% variance dari dataset dan PC2 menjelaskan 2,6% variance dari dataset.

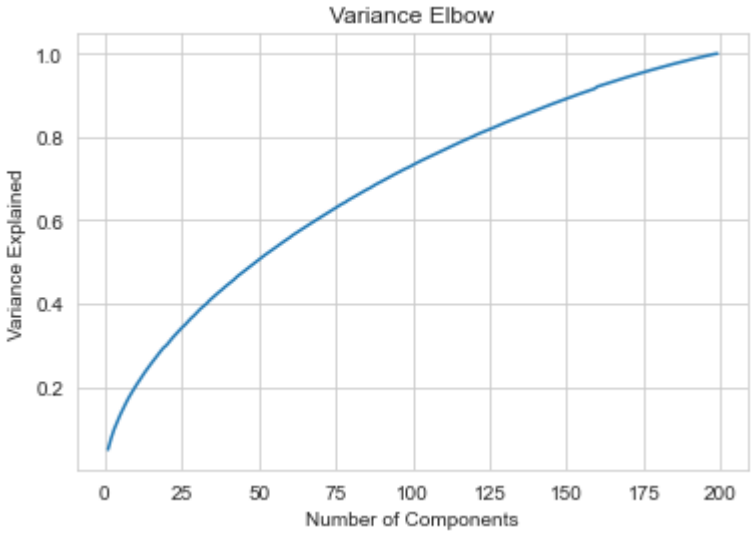
In [40]:
```
np.sum(pca.explained_variance_ratio_)
```

Out[40]:
```
0.07653534727307632
```

Bersama-sama, mereka menjelaskan 7,6% variance dari dataset. Ini memang tergolong kecil karena kita berusaha mengubah dari 200 dimensi menjadi 2 dimensi dan kemungkinan ada banyak loss.

In [41]:
```
explained_variance = []

for n in range(1,200):
    pca = PCA(n_components=n)
    pca.fit(scaled_X)

    explained_variance.append(np.sum(pca.explained_variance_ratio_))
```

In [42]:
```
plt.plot(range(1,200),explained_variance)
plt.title('Variance Elbow')
plt.xlabel("Number of Components")
plt.ylabel("Variance Explained");
plt.show()
```



Data Visualization

In [43]:
```
d = {'PC1': principal_components[:,0], 'PC2': principal_components[:,1], 'Class': y}
```

In [44]:
```
new_dataset = pd.DataFrame(d)
```

In [45]:
```
new_dataset
```

Out[45]:

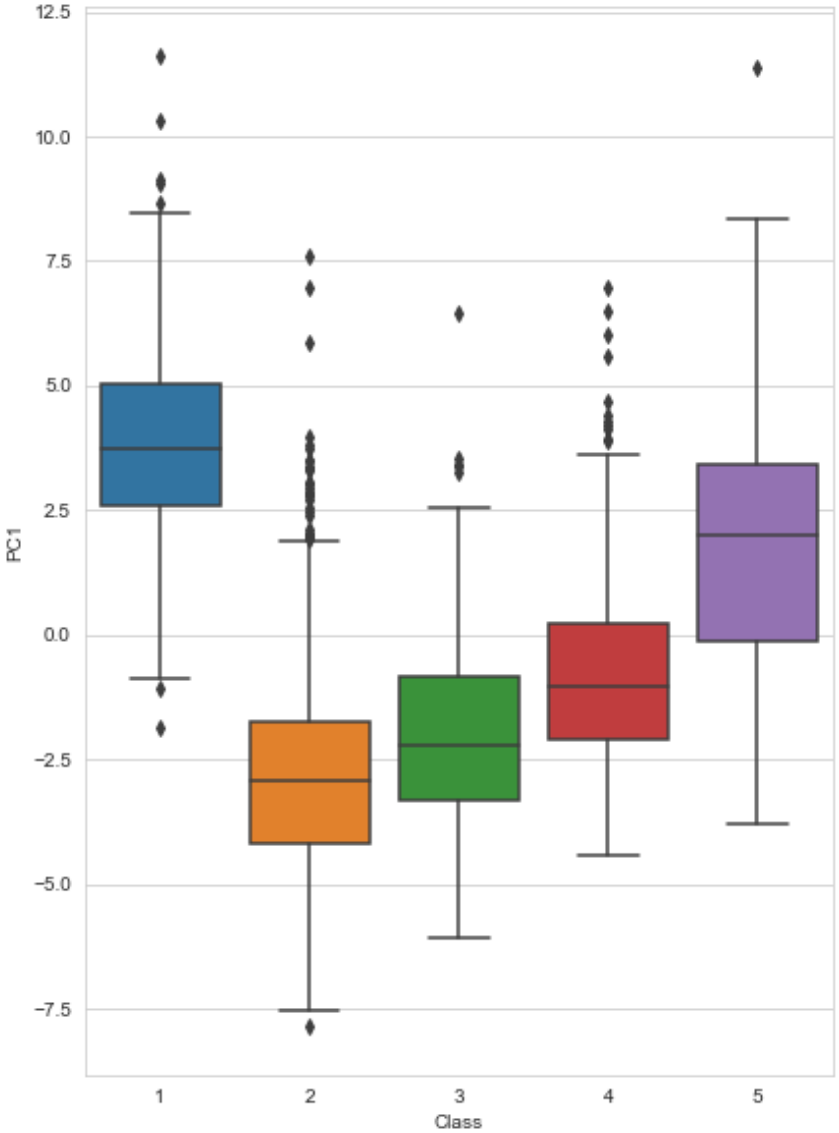
	PC1	PC2	Class
0	4.598023	0.951553	1
1	1.571962	2.017165	1
2	2.833041	0.814407	1
3	7.025828	-0.176614	1
4	7.209296	-1.291620	1
...	...	...	...
4075	3.856432	2.068715	5
4076	3.654091	2.361181	5

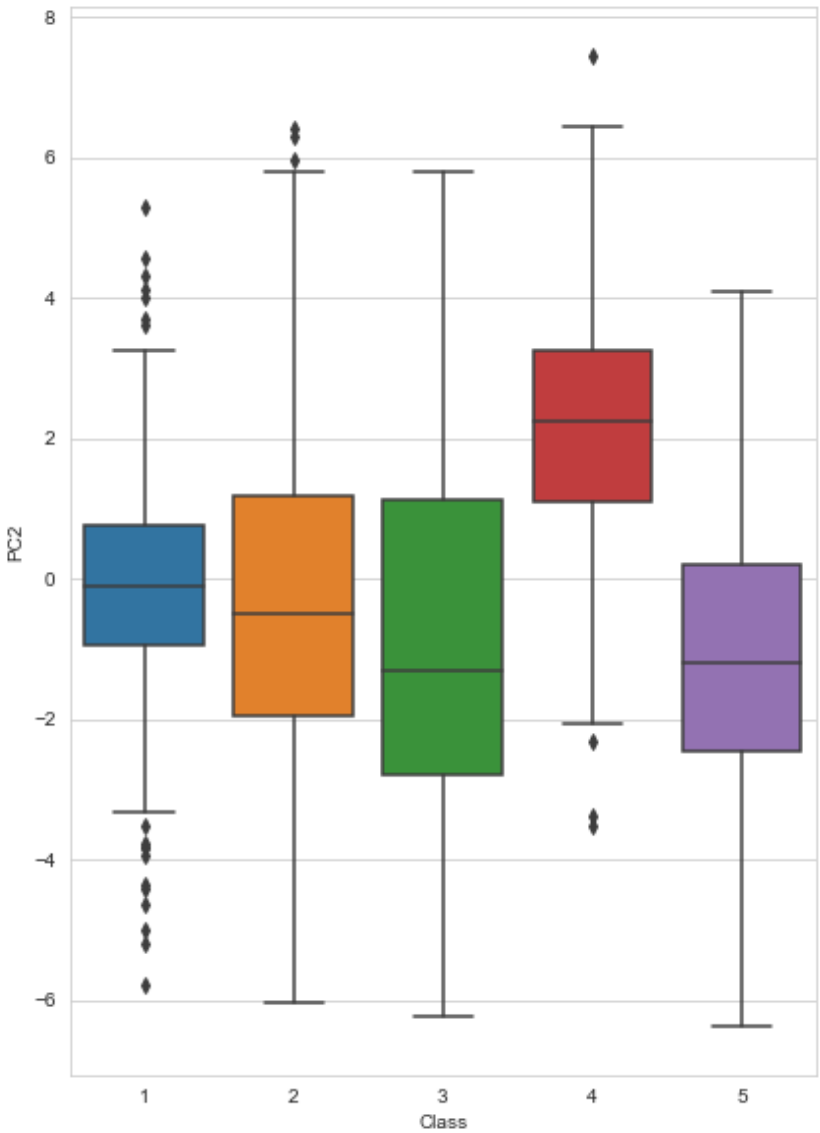
	PC1	PC2	Class
4077	2.791469	1.976879	5
4078	3.692784	1.913537	5
4079	3.819187	2.335450	5

4080 rows × 3 columns

In [46]:
```
num_list = ['PC1', 'PC2']
```

In [47]:
```
fig = plt.figure(figsize = (15,10))

for i in range(len(num_list)):
    column = num_list[i]
    sub = fig.add_subplot(1, 2, i+1)
    sns.boxplot(x = 'Class', y = column, data = new_dataset)
```





Ada banyak outlier di setiap kelas. Namun, kita tidak perlu menghapus outlier kecuali jika ada kesalahan dalam data. Outlier tidak selalu merupakan titik data yang buruk. Terkadang mereka adalah data yang paling penting dari semuanya.

Kita hanya membuat new_X karena tidak ada pengubahan pada y-nya.

In [48]:
```
new_X = new_dataset.drop(['Class'], axis=1)
```

In [49]:
```
new_X.head()
```

Out[49]:

	PC1	PC2
0	4.598023	0.951553
1	1.571962	2.017165
2	2.833041	0.814407
3	7.025828	-0.176614
4	7.209296	-1.291620

Dengan menggunakan PCA

Train Test Split

In [50]:
```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
X_train, X_test, y_train, y_test = train_test_split(new_X, y, test_size=0.25, random_state=101)
```

In [51]:
```
X_train.shape
```

Out[51]:
```
(3060, 2)
```

In [52]:
```
y_train.shape
```

Out[52]:
```
(3060,)
```

Pipeline

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
model_pipeline = []
model_pipeline.append(KNeighborsClassifier())
model_pipeline.append(DecisionTreeClassifier())
model_pipeline.append(RandomForestClassifier())
model_pipeline.append(GaussianNB())
```

Kita tidak memasukkan SVM karena pada umumnya, SVM tidak mendukung classification multiclass secara native. SVM memang mendukung binary classification dan memisahkan titik data menjadi dua kelas. Namun, untuk klasifikasi multiclass, kita butuh tenaga ekstra untuk mengimplementasikan prinsip yang sama, yaitu memecah multiclass classification menjadi beberapa masalah binary classification.

In [54]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

model_list = ['KNN', 'Decision Tree', 'Random Forest', 'Naive Bayes']
acc_list = []
prec_list = []
rec_list = []
f1_list = []
cm_list = []

for model in model_pipeline:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc_list.append(accuracy_score(y_test, y_pred))
    prec_list.append(precision_score(y_test, y_pred, average='weighted'))
    rec_list.append(recall_score(y_test, y_pred, average='weighted'))
    f1_list.append(f1_score(y_test, y_pred, average='weighted'))
    cm_list.append(confusion_matrix(y_test, y_pred))
```

In [55]:
```
d = {'Model': model_list, 'Accuracy': acc_list, 'Precision': prec_list, 'Recall': rec_list, 'F1 Score': f1_list}
```

In [56]:
```
model_evaluation = pd.DataFrame(d)
```

In [57]:
```

```

```
model_evaluation
```

Out[57]:

	Model	Accuracy	Precision	Recall	F1 Score
0	KNN	0.530392	0.529720	0.530392	0.527318
1	Decision Tree	0.495098	0.497582	0.495098	0.495517
2	Random Forest	0.529412	0.522813	0.529412	0.524707
3	Naive Bayes	0.576471	0.569653	0.576471	0.564802

Evaluation Metrics

Jika kita menggunakan PCA, kita akan menggunakan kedua model dengan akurasi tertinggi yaitu Naive Bayes dan Random Forest.

In [58]:
```
model_evaluation.nlargest(n=2, columns=['Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

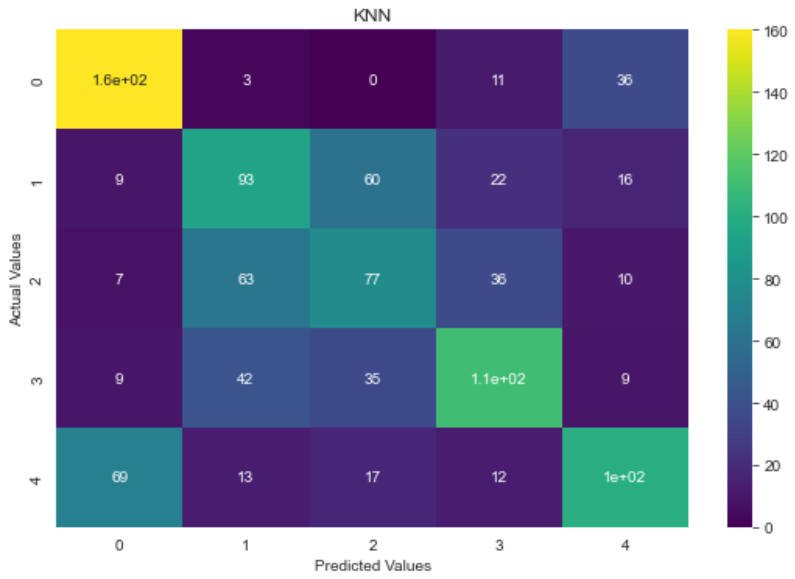
Out[58]:

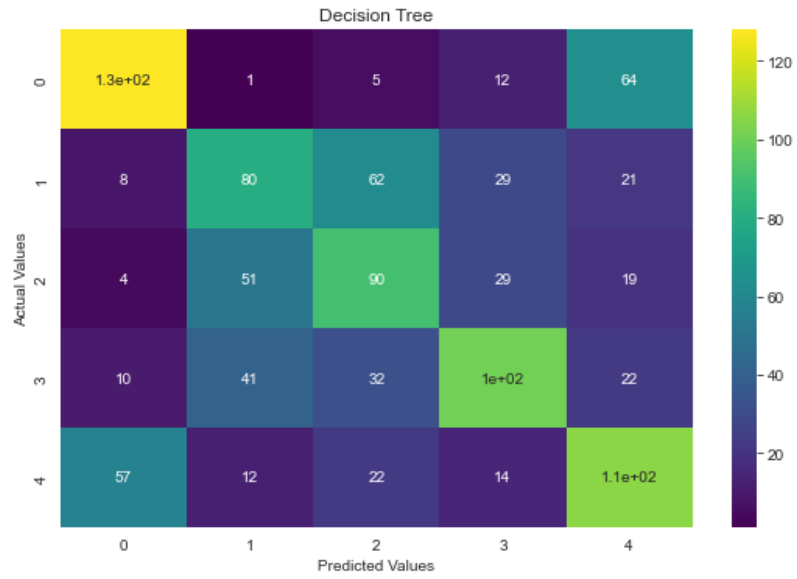
	Model	Accuracy	Precision	Recall	F1 Score
3	Naive Bayes	0.576471	0.569653	0.576471	0.564802
0	KNN	0.530392	0.529720	0.530392	0.527318

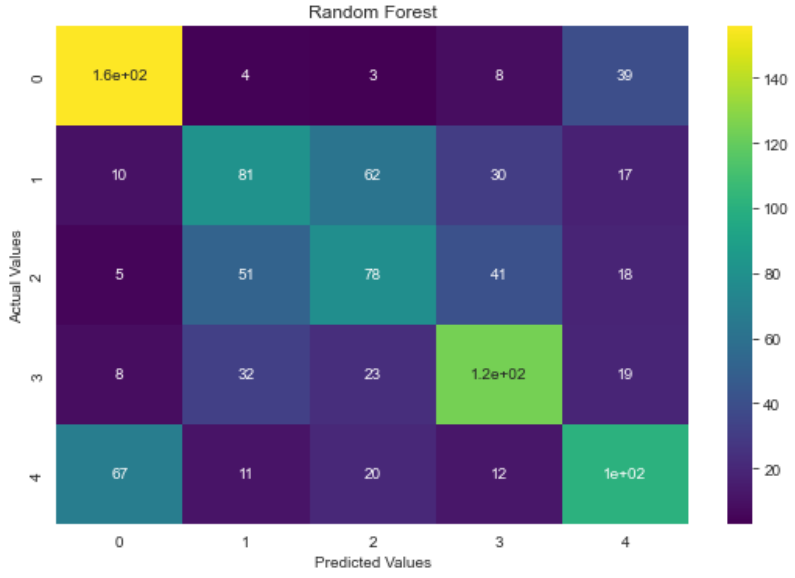
Kita menggunakan Naive Bayes sebab algoritma ini mudah dan cepat untuk memprediksi kelas dari kumpulan data testing. Algoritma ini juga bekerja dengan baik dalam prediksi multiclass dengan jumlah feature yang tidak terlalu banyak. Disebut naif karena mengasumsikan independensi antara setiap pasangan fitur dalam data.

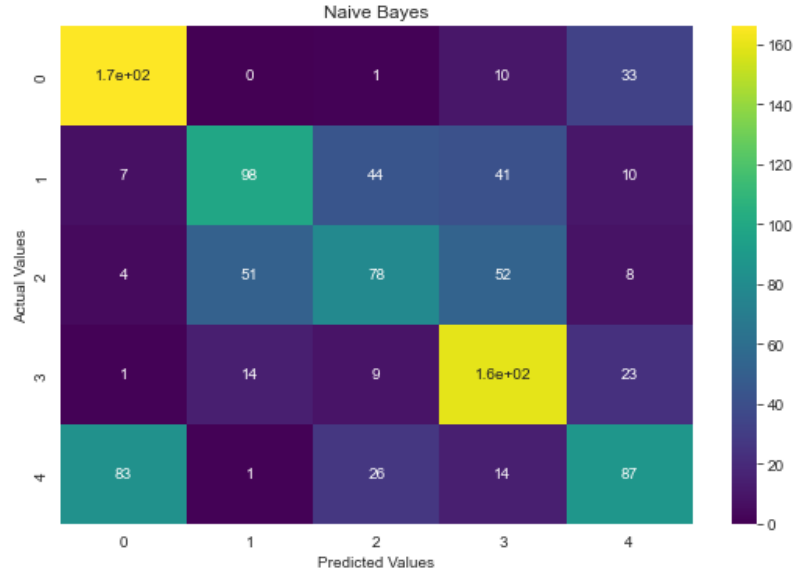
Random Forest juga tidak kalah menarik dengan Naive Bayes karena algoritma ini juga mudah diinterpretasikan. Tetapi, perbedaannya ada Naive Bayes bekerja dengan baik pada kumpulan data yang kecil. Sedangkan, Random Forest bekerja dengan baik pada kumpulan data yang besar dan tidak sensitif terhadap outlier.

In [59]:
```
fig = plt.figure(figsize=(20,20))
for i in range(len(cm_list)):
    cm = cm_list[i]
    model = model_list[i]
    sub = fig.add_subplot(3, 2, i+1).set_title(model)
    cm_plot = sns.heatmap(cm, annot=True, cmap='viridis')
    cm_plot.set_xlabel('Predicted Values')
    cm_plot.set_ylabel('Actual Values')
```









Tanpa menggunakan PCA

Train Test Split

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101)
```

In [61]:
```
X_train.shape
```

Out[61]:
```
(3060, 200)
```

In [62]:
```
y_train.shape
```

Out[62]:
```
(3060,)
```

Pipeline

In [63]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

model_pipeline = []
model_pipeline.append(KNeighborsClassifier())
model_pipeline.append(DecisionTreeClassifier())
model_pipeline.append(RandomForestClassifier())
model_pipeline.append(GaussianNB())
```

In [64]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

model_list = ['KNN', 'Decision Tree', 'Random Forest', 'Naive Bayes']
acc_list = []
prec_list = []
rec_list = []
f1_list = []
cm_list = []
```

```
for model in model_pipeline:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc_list.append(accuracy_score(y_test, y_pred))
    prec_list.append(precision_score(y_test, y_pred, average='weighted'))
    rec_list.append(recall_score(y_test, y_pred, average='weighted'))
    f1_list.append(f1_score(y_test, y_pred, average='weighted'))
    cm_list.append(confusion_matrix(y_test, y_pred))
```

In [65]:
```
d = {'Model': model_list, 'Accuracy': acc_list, 'Precision': prec_list, 'Recall': rec_list, 'F1 Score': f1_list}
```

In [66]:
```
model_evaluation = pd.DataFrame(d)
```

In [67]:
```
model_evaluation
```

Out[67]:

	Model	Accuracy	Precision	Recall	F1 Score
0	KNN	0.914706	0.916135	0.914706	0.914446
1	Decision Tree	0.682353	0.684833	0.682353	0.682586
2	Random Forest	0.919608	0.920329	0.919608	0.919337
3	Naive Bayes	0.786275	0.785520	0.786275	0.785171

Evaluation Metrics

Jika kita tidak menggunakan PCA, kita akan menggunakan kedua model dengan akurasi tertinggi yaitu KNN dan Random Forest.

In [68]:
```
model_evaluation.nlargest(n=2, columns=['Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

Out[68]:

	Model	Accuracy	Precision	Recall	F1 Score
2	Random Forest	0.919608	0.920329	0.919608	0.919337
0	KNN	0.914706	0.916135	0.914706	0.914446

Seperti yang dapat dilihat, KNN mengandalkan kesamaan data yang dapat diamati dan metrik jarak yang canggih untuk menghasilkan prediksi yang akurat. Teknik ini mungkin tampak sedikit berlawanan dengan intuisi dan kita mungkin tidak dapat dipercaya pada awalnya,

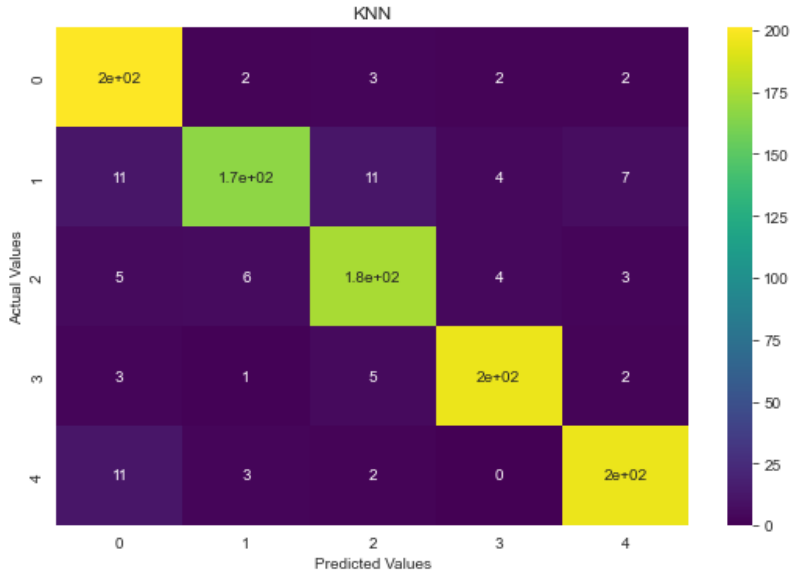
tetapi sebenarnya sangat dapat diandalkan. KNN paling berguna ketika feature data berlabel terlalu banyak, dan dapat mencapai akurasi tinggi dalam berbagai macam masalah tipe prediksi.

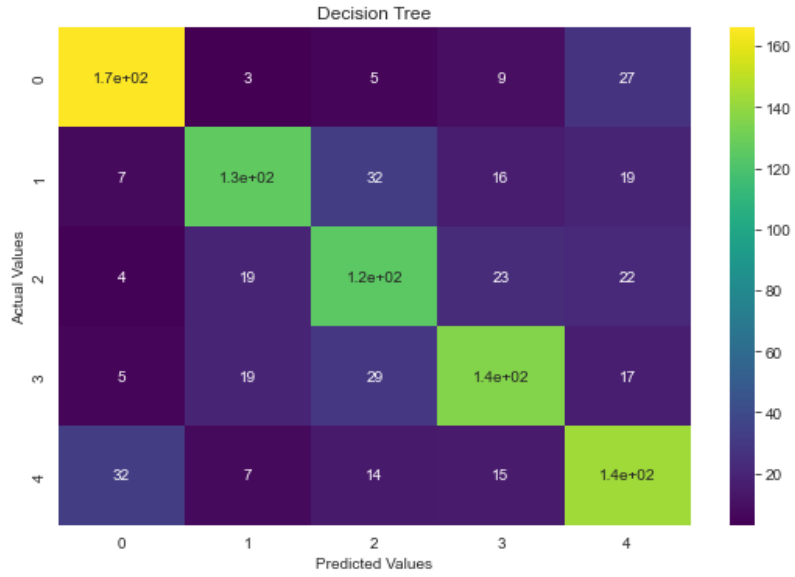
Sama seperti model dengan menggunakan PCA, disini, kita juga menggunakan model Random Forest membangun beberapa Decision Tree dan menggabungkannya untuk mendapatkan prediksi yang lebih akurat dan stabil. Kualitas hebat lainnya dari algoritma Random Forest adalah sangat mudah untuk mengukur kepentingan relatif dari setiap feature yang diprediksi. Kita dapat mengetahui pentingnya fitur dengan menghitung skor secara otomatis untuk setiap feature setelah training dan scale hasilnya sehingga jumlah semua kepentingan sama dengan satu.

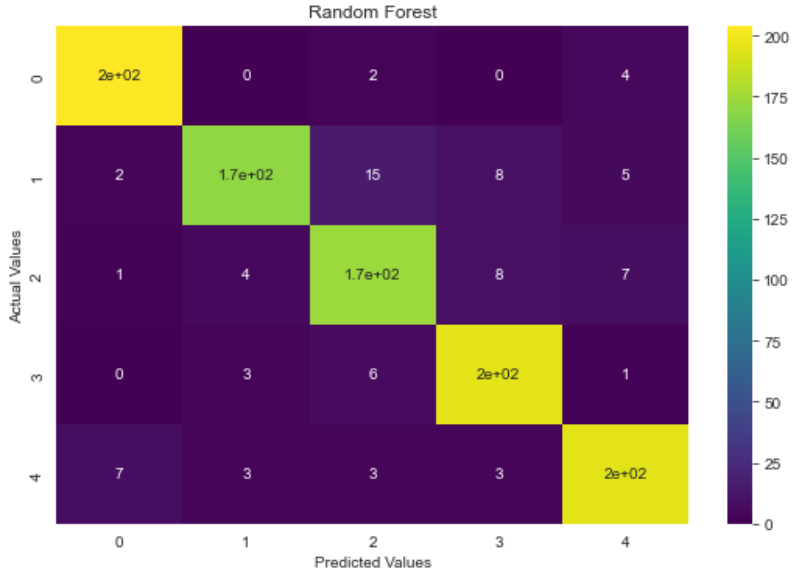
Mengapa akurasi tanpa menggunakan PCA lebih tinggi daripada menggunakan PCA?

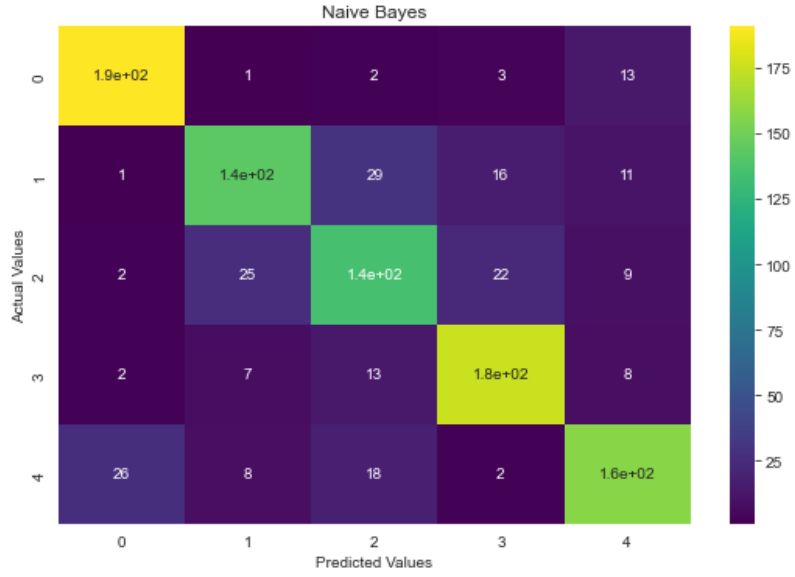
Tentu saja lebih tinggi karena mengurangi feature juga menyebabkan penurunan variance yang menjelaskan data kita secara lebih spesifik. Dalam kasus ini, datanya terbagi secara merata, dimana setiap kelas memiliki 816 data. Sebenarnya, tanpa melakukan preprocessing juga, kita dapat langsung membuat model karena keuntungan ini. Alangkah lebih baik agar setiap feature dilabeli dengan keterangan yang jelas, tidak hanya F1, F2, dst., agar kita tidak semata-mata memilih jumlah komponen PCA, dan memikirkan kepentingan tiap feature.

In [69]:
```
fig = plt.figure(figsize=(20,20))
for i in range(len(cm_list)):
    cm = cm_list[i]
    model = model_list[i]
    sub = fig.add_subplot(3, 2, i+1).set_title(model)
    cm_plot = sns.heatmap(cm, annot=True, cmap='viridis')
    cm_plot.set_xlabel('Predicted Values')
    cm_plot.set_ylabel('Actual Values')
```









Terima kasih.