**ASSIGNMENT 2**

**Anu Juneja**

**IoT Management (ISTM\_6290\_80)**

**K-Means Clustering with PySpark**

# Importing libraries and Building Spark Session

import os

os.environ['JAVA\_HOME'] = 'C:/Program Files/Java/jre1.8.0\_241'

os.environ['PYSPARK\_SUBMIT\_ARGS'] = "--master local[2] pyspark-shell"

from pyspark.sql import SparkSession

import pyspark.sql.functions as F

spark = SparkSession.builder.appName('k-means clustering').getOrCreate()

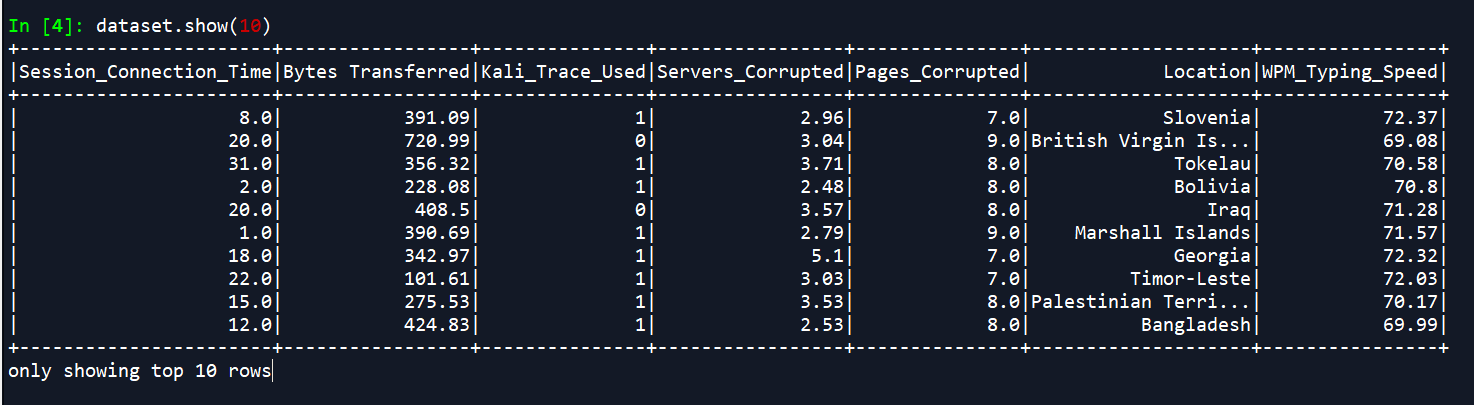
# Importing Data

# Loading datafile "hack\_data.txt"

dataset = spark.read.csv("c:/users/anuju/desktop/Internet\_Of\_Things/Assignments/hack\_data.txt",header=True,inferSchema=True)

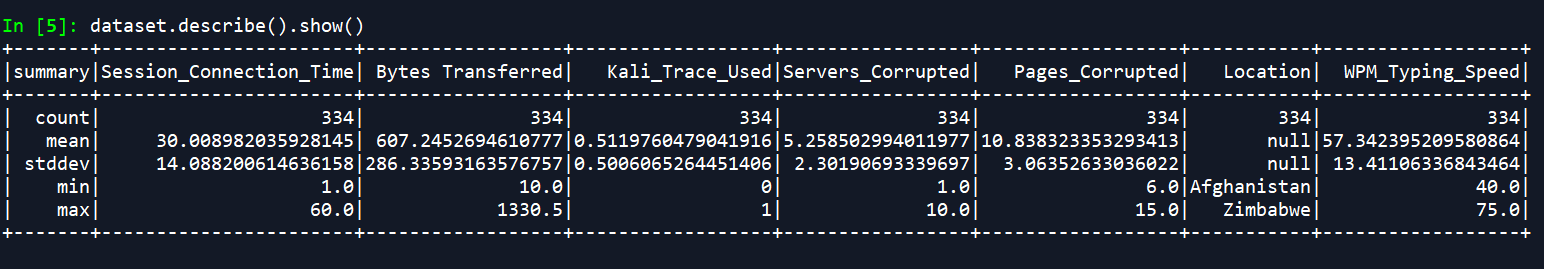
dataset.show(10)

**OUTPUT:**



Dataset.describe().show()

**OUTPUT:**



**#UNDERSTANDING THE DATA**

**#plotting a histogram of the data**

dataset.toPandas().hist(column=dataset.columns, bins=20, figsize=(7,7))

**OUTPUT:**

A screenshot of a video game

Description automatically generated

Looking at the data and various variables (features) of the hackers involved, if we look at the WPM\_Typing\_Speed, We can clearly see that there are two groups, with typing speed of 40-48 words per minute and the other one being 66-75 words per minute, which tells us there might be two hackers.

Similarly, we can even analyse that almost equal number of hacking attempts were made with the use of Kali Linux, and as per the forensics they should have roughly same amount of attacks, again diverging the data towards two hackers.

But we cannot be sure as of now, as the rest of the data is scattered, so we should dive into deeper analysis of the data.

**# Correlation Matrix plot**

import seaborn as sns

import matplotlib as plt

**# Get Correlations**

corr = dataset.toPandas()[dataset.columns].corr()

**#plotting the heatmap**

sns.heatmap(corr,annot=True)

A close up of a logo

Description automatically generated

If we look at the heatmap above, the variables Servers\_Corrupted, Pages\_Corrupted, Bytes\_Transferred and Session\_Connection\_Time are highly correlated with each other. So, there is an issue of multicollinearity which means that change in value of one variable changes the value of the other.These are not mutually exclusive.

When variables used in clustering are collinear, some variables get a higher weight than others. If two variables are perfectly correlated, they effectively represent the same concept. But that concept is now represented twice in the data and hence gets twice the weight of all the other variables. The final solution is likely to be skewed in the direction of that concept, which could be a problem if it’s not anticipated.

In the case of multiple variables and multicollinearity, the analysis is in effect being conducted on some unknown number of concepts that are a subset of the actual number of variables being used in the analysis.

**# ## Format the Data**

**#Converting Categorical Data**

from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler

from pyspark.ml import Pipeline

import pyspark.sql.functions as F

stage\_string = StringIndexer().setInputCol("Location").setOutputCol("LocationIndex")

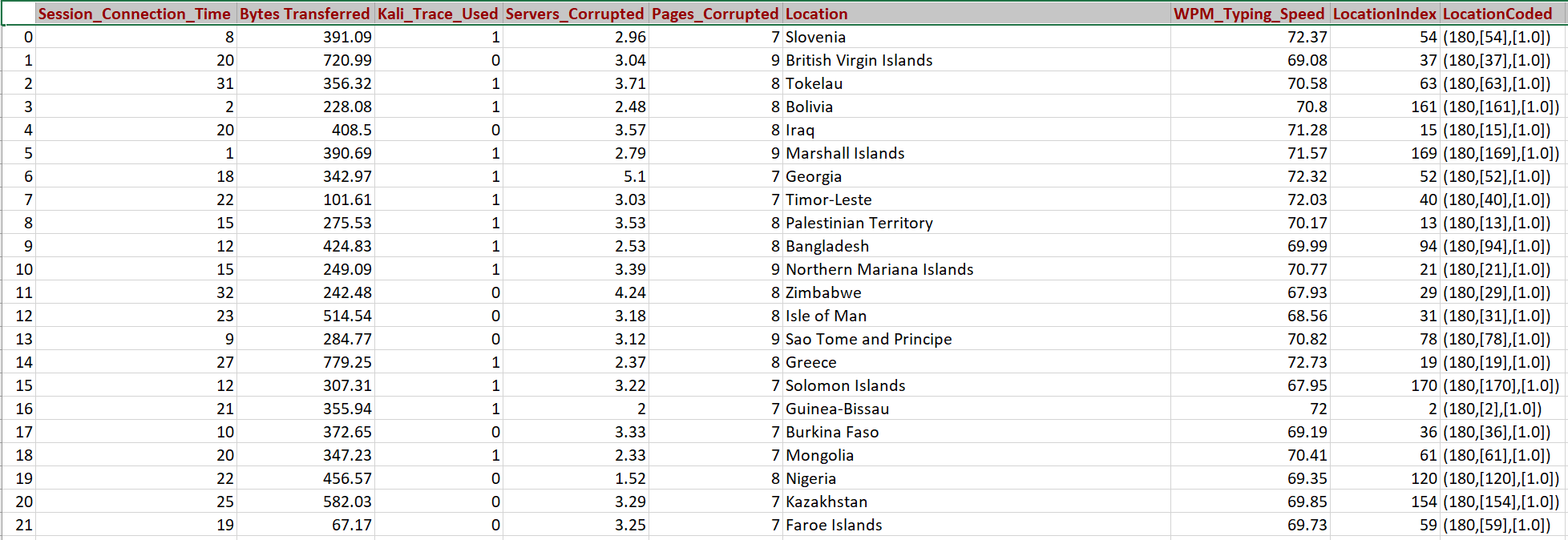
stage\_one\_hot = OneHotEncoder().setInputCol("LocationIndex").setOutputCol("LocationCoded")

ppl = Pipeline(stages = [stage\_string , stage\_one\_hot])

df = ppl.fit(dataset).transform(dataset)

df.toPandas().to\_csv('HackData\_afterTransform.csv')

**#Snippet from the file “HackData\_afterTransform.csv”, showing the coded columns, LocationIndex (after using StringIndexer) and LocationCoded (after using** **OneHotEncoder)**



After encoding the categorical column “Location”, I found out that there are 181 distinct values, so the clustering of data can be affected adversely by that. Hence, I took this decision of preparing three models where:

1. Location column will be eliminated from the features.
2. LocationIndex column (the result after using StringIndexer) will be used in the features along with all the other variables.
3. LocationCoded column (the result after using both StringIndexer and OneHotEncoder) will be used in the features along with all the other variables.

**# Extracting Features without Location**

Assembler\_NoLocation = VectorAssembler(

inputCols = ['Session\_Connection\_Time', 'Bytes Transferred',

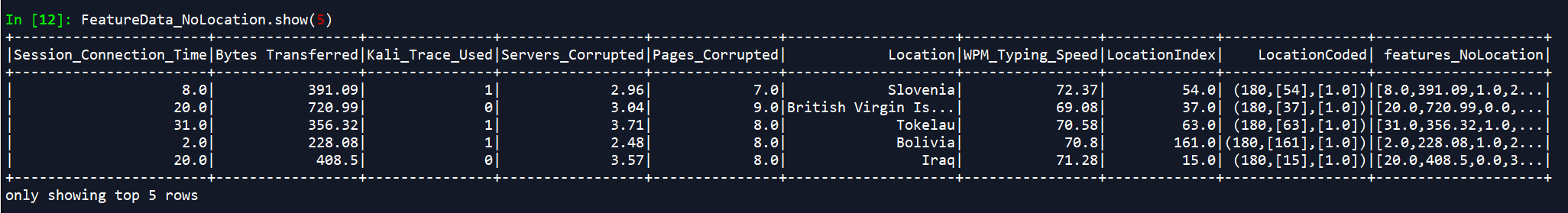
'Kali\_Trace\_Used', 'Servers\_Corrupted', 'Pages\_Corrupted', 'WPM\_Typing\_Speed'], outputCol="features\_NoLocation")

FeatureData\_NoLocation= Assembler\_NoLocation.transform(df)

FeatureData\_NoLocation.toPandas().to\_csv('HackDataFeatures\_NoLocation.csv')

FeatureData\_NoLocation.show(5)

**OUTPUT:**



**# Extracting Features with Location after String Indexer**

Assembler\_LocationIndex = VectorAssembler(

inputCols=['Session\_Connection\_Time', 'Bytes Transferred',

'Kali\_Trace\_Used',

'Servers\_Corrupted',

'Pages\_Corrupted',

'WPM\_Typing\_Speed', 'LocationIndex'], outputCol="features\_LocationIndex")

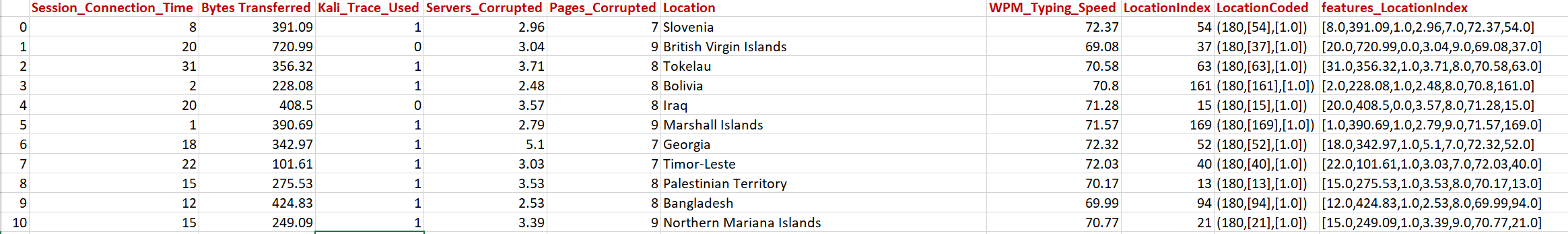
FeatureData\_LocationIndex= Assembler\_LocationIndex.transform(df)

FeatureData\_LocationIndex.toPandas().to\_csv('HackDataFeatures\_LocationIndex.csv')

FeatureData\_LocationIndex.show(5)

**OUTPUT:**

**Snippet from the file “HackDataFeatures\_LocationIndex.csv”**



**# Extracting Features with Location after StringIndexer and OneHotEncoder**

Assembler\_LocationCoded = VectorAssembler(

inputCols=['Session\_Connection\_Time',

'Bytes Transferred',

'Kali\_Trace\_Used', 'Servers\_Corrupted', 'Pages\_Corrupted', 'WPM\_Typing\_Speed', 'LocationCoded'],outputCol="features\_LocationCoded")

FeatureData\_LocationCoded= Assembler\_LocationCoded.transform(df)

FeatureData\_LocationCoded.toPandas().to\_csv('HackDataFeatures\_LocationCoded.csv')

FeatureData\_LocationCoded.show(5)

**OUTPUT:**

**Snippet from the file “HackDataFeatures\_LocationCoded.csv”**



Next step is scaling the data collected under the label features for all the three models.

**#Scaling the data without Location**

from pyspark.ml.feature import StandardScaler

scaler\_NoLocation = StandardScaler(inputCol="features\_NoLocation", outputCol="scaledFeatures\_NoLocation", withStd=True, withMean=False)

**# Compute summary statistics by fitting the StandardScaler**

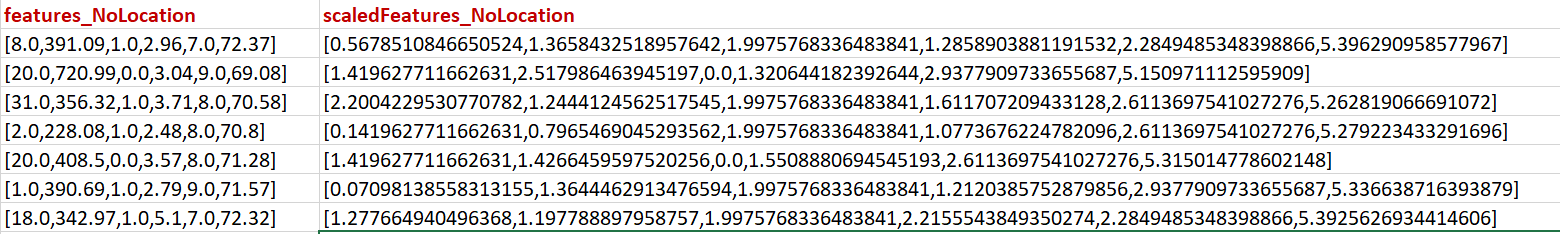
scalerModel\_NoLocation = scaler\_NoLocation.fit(FeatureData\_NoLocation)

**# Normalize each feature to have unit standard deviation.**

FinalData\_NoLocation = scalerModel\_NoLocation.transform(FeatureData\_NoLocation)

FinalData\_NoLocation.toPandas().to\_csv('HackDataFinal\_NoLocation.csv')

**Snippet from the file 'HackDataFinal\_NoLocation.csv'**



**#Scaling the data with Location after String Indexer**

scaler\_LocationIndex = StandardScaler(inputCol="features\_LocationIndex", outputCol="scaledFeatures\_LocationIndex", withStd=True, withMean=False)

**# Compute summary statistics by fitting the StandardScaler**

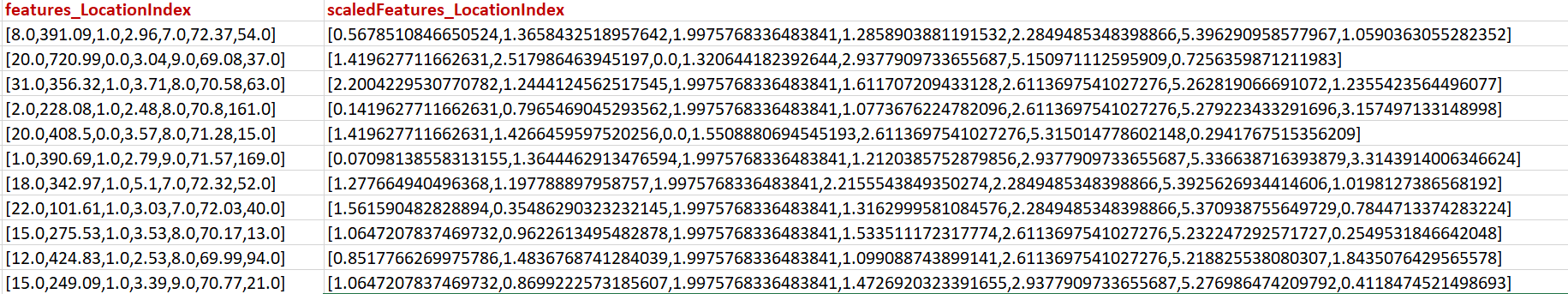
scalerModel\_LocationIndex = scaler\_LocationIndex.fit(FeatureData\_LocationIndex)

**# Normalize each feature to have unit standard deviation.**

FinalData\_LocationIndex = scalerModel\_LocationIndex.transform(FeatureData\_LocationIndex)

FinalData\_LocationIndex.toPandas().to\_csv('HackDataFinal\_LocationIndex.csv')

**Snippet from the file 'HackDataFinal\_LocationIndex.csv'**



**#Scaling the data with Location after StringIndexer and OneHotEncoder**

scaler\_LocationCoded = StandardScaler(inputCol="features\_LocationCoded", outputCol="scaledFeatures\_LocationCoded", withStd=True, withMean=False)

**# Compute summary statistics by fitting the StandardScaler**

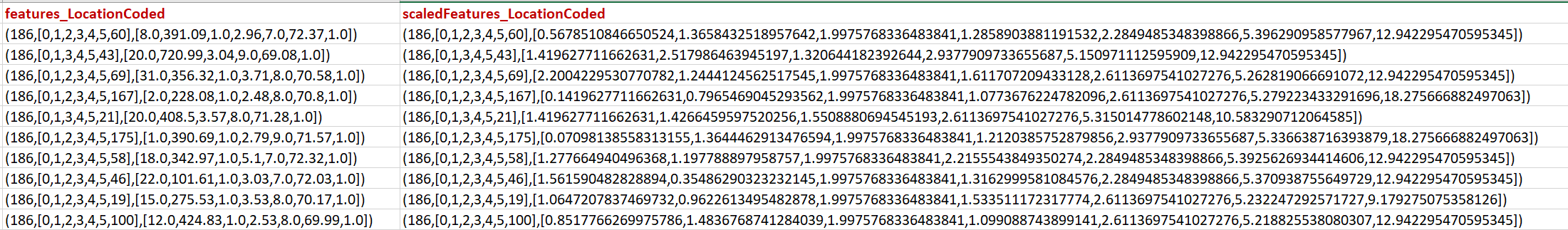
scalerModel\_LocationCoded = scaler\_LocationCoded.fit(FeatureData\_LocationCoded)

**# Normalize each feature to have unit standard deviation.**

FinalData\_LocationCoded = scalerModel\_LocationCoded.transform(FeatureData\_LocationCoded)

FinalData\_LocationCoded.toPandas().to\_csv('HackDataFinal\_LocationCoded.csv')

**Snippet from the file 'HackDataFinal\_LocationCoded.csv'**



Now I will be finding the appropriate K-value for all the three conditions and building clustering models on the same.

**# ## Train the Model and Evaluate**

**#Importing necessary libraries**

from pyspark.ml.clustering import KMeans

import matplotlib.ticker as ticker

import math

**# Finding the Appropriate K Value by Elbow Method**

**# Without Location**

wssse= [ ]

KMin = 2

KMax = 34

KStep = 1

index = 0

for iLoop in range(KMin,KMax,KStep):

kmeans = KMeans(featuresCol='scaledFeatures\_NoLocation', k=iLoop)

model = kmeans.fit(FinalData\_NoLocation)

wssse.append(model.computeCost(FinalData\_NoLocation))

print("When k=" + str(iLoop) +" Within Set Sum of Squared Errors = " + str(wssse[index]))

index += 1

fig, ax = plt.subplots(1,1, figsize =(8,8))

ax.plot(range(KMin,KMax, KStep),wssse[0:math.ceil(KMax/KStep)],'--bo')

ax.xaxis.set\_major\_locator(ticker.MultipleLocator(KStep))

ax.set\_xlabel('k-Value',fontsize = 20)

ax.set\_ylabel('WSSSE',fontsize = 20)

ax.grid()

ax.set\_title("The Elbow Method to find the k-value (NoLocation)",fontsize = 24)

A close up of a map

Description automatically generated

As Clear from the graph above, the k-Value for the model without the Location Column is 6.

**# With LocationIndex**

wssse= []

KMin = 2

KMax = 40

KStep = 1

index = 0

for iLoop in range(KMin,KMax,KStep):

kmeans = KMeans(featuresCol='scaledFeatures\_LocationIndex', k=iLoop)

model = kmeans.fit(FinalData\_LocationIndex)

wssse.append(model.computeCost(FinalData\_LocationIndex))

print("When k=" + str(iLoop) +" Within Set Sum of Squared Errors = " + str(wssse[index]))

index += 1

fig, ax = plt.subplots(1,1, figsize =(8,8))

ax.plot(range(KMin,KMax, KStep),wssse[0:math.ceil(KMax/KStep)],'--bo')

ax.xaxis.set\_major\_locator(ticker.MultipleLocator(KStep))

ax.set\_xlabel('k-Value',fontsize = 20)

ax.set\_ylabel('WSSSE',fontsize = 20)

ax.grid()

ax.set\_title("The Elbow Method to find the k-value (With LocationIndex)",fontsize = 24)

A close up of a mans face

Description automatically generated

As Clear from the graph above, the k-Value for the model with the LocationIndex Column is 9. After 9 it is reducing but there is a sudden rise in the WSSSE at k=11, which is an exception, but otherwise after k=9, it falls down.

**# With LocationCoded**

wssse= []

KMin = 2

KMax = 400

KStep = 10

index = 0

for iLoop in range(KMin,KMax,KStep):

kmeans = KMeans(featuresCol='scaledFeatures\_LocationCoded', k=iLoop)

model = kmeans.fit(FinalData\_LocationCoded)

wssse.append(model.computeCost(FinalData\_LocationCoded))

print("When k=" + str(iLoop) +" Within Set Sum of Squared Errors = " + str(wssse[index]))

index += 1

fig, ax = plt.subplots(1,1, figsize =(8,8))

ax.plot(range(KMin,KMax, KStep),wssse[0:math.ceil(KMax/KStep)],'--bo')

ax.xaxis.set\_major\_locator(ticker.MultipleLocator(KStep))

ax.set\_xlabel('k-Value',fontsize = 20)

ax.set\_ylabel('WSSSE',fontsize = 20)

ax.grid()

ax.set\_title("The Elbow Method to find the k-value (With LocationCoded)",fontsize = 24)

A close up of a mans face

Description automatically generated

As Clear from the graph above, the k-Value for the model with the LocationCoded Column is 192.

Let us Build the models now with these k-Values.

**#Training the K-Means Model**

**#Without Location**

Final\_KMeans\_NoLocation = KMeans(featuresCol='scaledFeatures\_NoLocation', k=6)

Final\_Model\_NoLocation = Final\_KMeans\_NoLocation.fit(FinalData\_NoLocation)

**#Evaluation of Model**

FinalModelWSSSE\_NoLocation = Final\_Model\_NoLocation.computeCost(FinalData\_NoLocation)

print ("Within Set Sum of Suared Errors = " + str(FinalModelWSSSE\_NoLocation))

**OUTPUT:**

Within Set Sum of Suared Errors = 232.62054194167905

**#With LocationIndex**

Final\_KMeans\_LocationIndex = KMeans(featuresCol='scaledFeatures\_LocationIndex', k=9)

Final\_Model\_LocationIndex = Final\_KMeans\_LocationIndex.fit(FinalData\_LocationIndex)

**#Evaluation of Model**

FinalModelWSSSE\_LocationIndex = Final\_Model\_LocationIndex.computeCost(FinalData\_LocationIndex)

print ("Within Set Sum of Suared Errors = " + str(FinalModelWSSSE\_LocationIndex))

**OUTPUT:**

Within Set Sum of Suared Errors = 335.55699366206727

**#With LocationCoded**

Final\_KMeans\_LocationCoded = KMeans(featuresCol='scaledFeatures\_LocationCoded', k=192)

Final\_Model\_LocationCoded = Final\_KMeans\_LocationCoded.fit(FinalData\_LocationCoded)

**#Evaluation of Model**

FinalModelWSSSE\_LocationCoded = Final\_Model\_LocationCoded.computeCost(FinalData\_LocationCoded)

print ("Within Set Sum of Suared Errors = " + str(FinalModelWSSSE\_LocationCoded))

**OUTPUT:**

Within Set Sum of Suared Errors = 848.7928430757306

**# Shows the result.**

**#Without Location**

centers = Final\_Model\_NoLocation.clusterCenters()

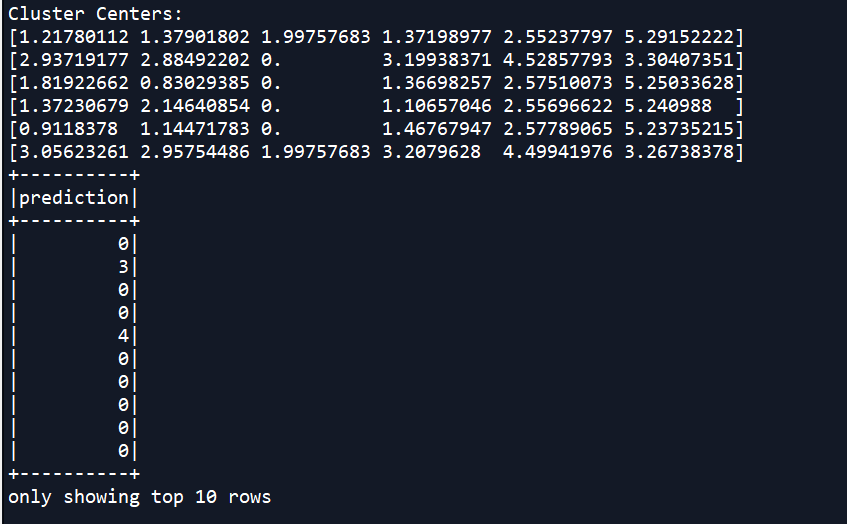
print("Cluster Centers: ")

for center in centers:

print(center)

**# Predict the label of each hacking attempt**

Final\_Model\_NoLocation.transform(FinalData\_NoLocation).select('prediction').show(10)

**OUTPUT:** 

**#With LocationIndex**

centers = Final\_Model\_LocationIndex.clusterCenters()

print("Cluster Centers: ")

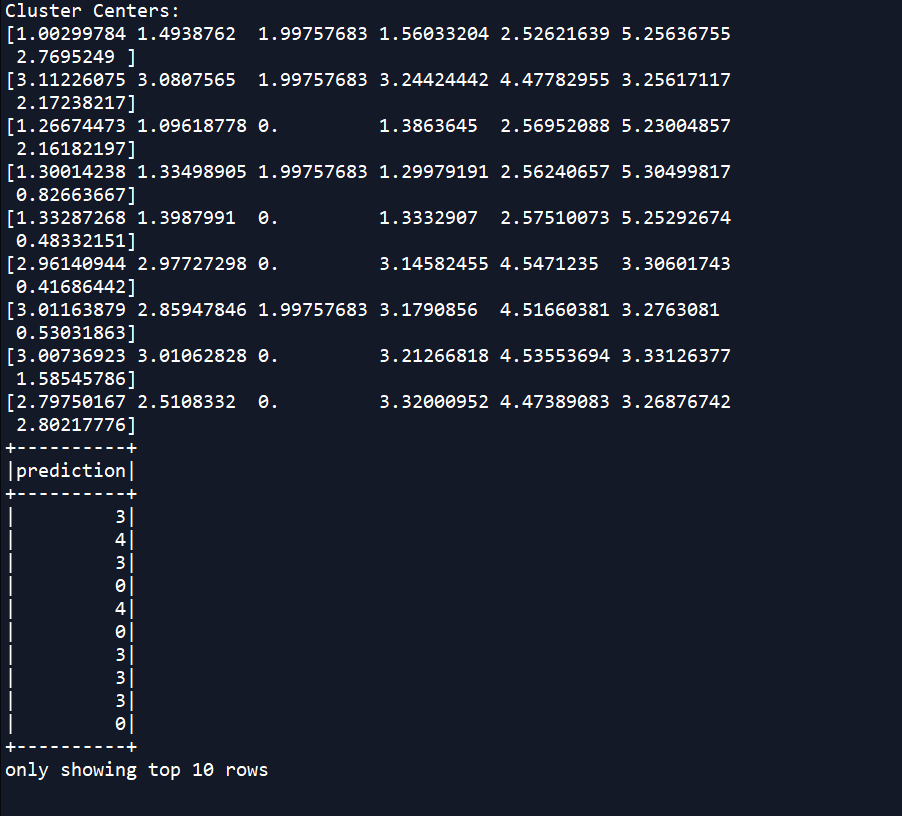
for center in centers:

print(center)

**# Predict the label of each hacking attempt**

Final\_Model\_LocationIndex.transform(FinalData\_LocationIndex).select('prediction').show(10)

**OUTPUT:**



**#With LocationCoded**

centers = Final\_Model\_LocationCoded.clusterCenters()

print("Cluster Centers: ")

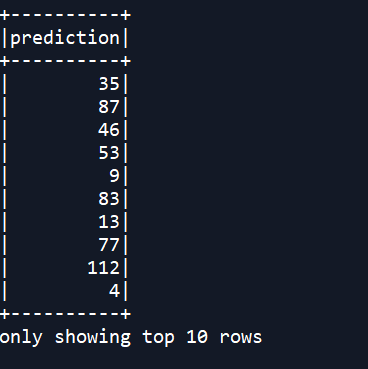
for center in centers:

print(center)

**# Predict the label of each hacking attempt**

Final\_Model\_LocationCoded.transform(FinalData\_LocationCoded).select('prediction').show(10)

**OUTPUT:**



**#formingClusters**

**#Without Location**

clusters\_NoLocation = Final\_Model\_NoLocation.transform(FinalData\_NoLocation).select('\*')

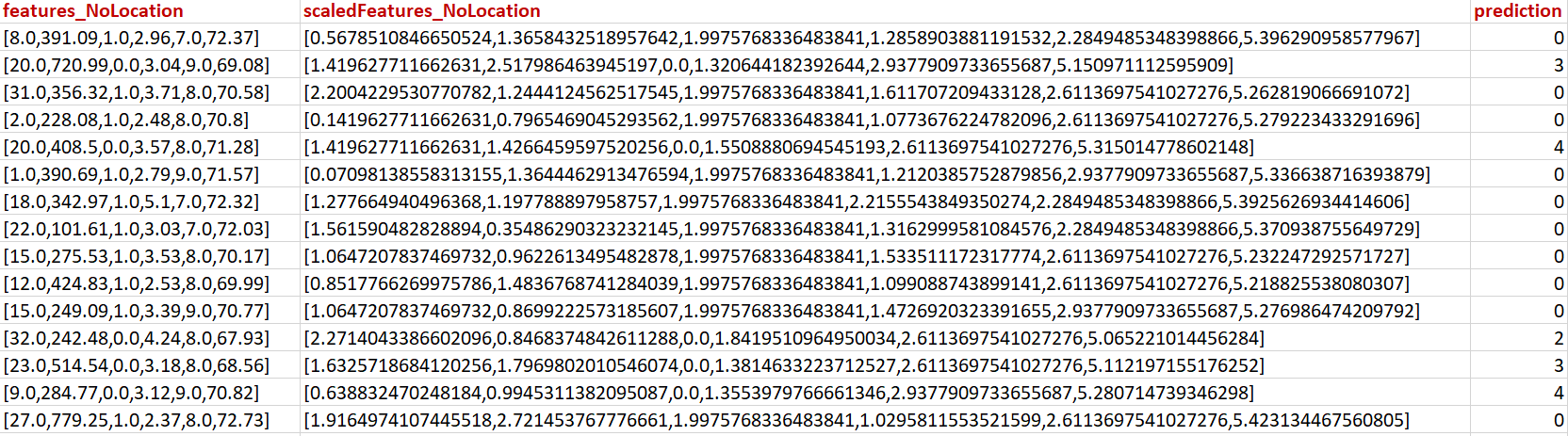
clusters\_NoLocation.groupBy("prediction").count().orderBy(F.desc("count")).show()

clusters\_NoLocation.show()

clusters\_NoLocation\_pd = clusters\_NoLocation.toPandas()

clusters\_NoLocation\_pd.to\_csv("Clusters\_NoLocation.csv")

**Snippet from the file 'Clusters\_NoLocation.csv'**



**#With LocationIndex**

clusters\_LocationIndex = Final\_Model\_LocationIndex.transform(FinalData\_LocationIndex).select('\*')

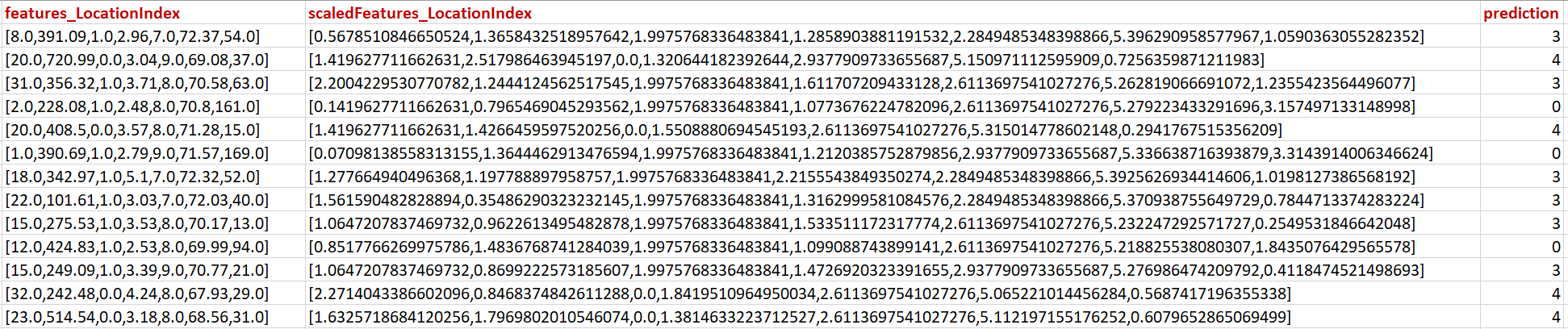
clusters\_LocationIndex.groupBy("prediction").count().orderBy(F.desc("count")).show()

clusters\_LocationIndex.show()

clusters\_LocationIndex\_pd = clusters\_LocationIndex.toPandas()

clusters\_LocationIndex\_pd.to\_csv("clusters\_LocationIndex.csv")

**Snippet from the file 'Clusters\_LocationIndex.csv'**



**#With LocationCoded**

clusters\_LocationCoded = Final\_Model\_LocationCoded.transform(FinalData\_LocationCoded).select('\*')

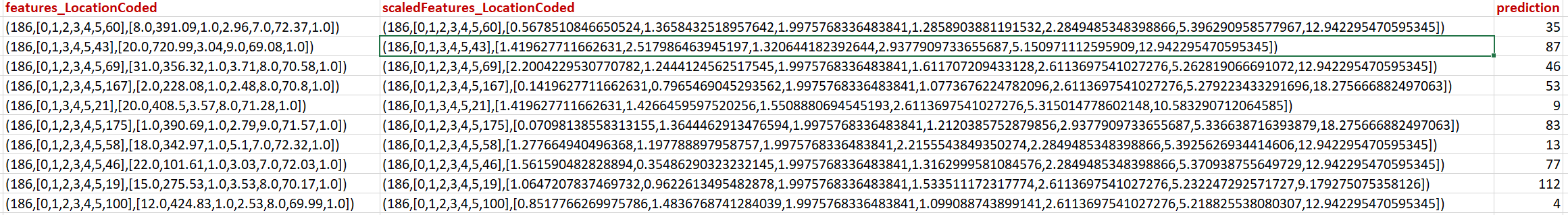
clusters\_LocationCoded.groupBy("prediction").count().orderBy(F.desc("count")).show()

clusters\_LocationCoded.show()

clusters\_LocationCoded\_pd = clusters\_LocationCoded.toPandas()

clusters\_LocationCoded\_pd.to\_csv("clusters\_LocationCoded.csv")

**Snippet from the file 'Clusters\_LocationCoded.csv'**



**#Plotting Clusters**

**# Pairwise Scatterplot**

**# Without Location**

dataScatterPlot\_NoLocation = clusters\_NoLocation\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed','prediction']]

Variables\_NoLocation = clusters\_NoLocation\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed']]

g\_NoLocation = sns.pairplot( dataScatterPlot\_NoLocation, vars = Variables\_NoLocation, hue = "prediction", diag\_kind = 'kde',palette= "husl", plot\_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'})

A picture containing text

Description automatically generated

If we take a look at all the pairplots between the different variables of the data, we see that there are two different clusters formed, hence confirming the presence of two hackers, not three.

Though Bytes\_transferred and Session\_Connection\_Time pairplots are a little distorted and they do not form clear clusters( because they are highly correlated, see HEATMAP)

**# With LocationIndex**

dataScatterPlot\_LocationIndex = clusters\_LocationIndex\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed','LocationIndex','prediction']]

Variables\_LocationIndex = clusters\_LocationIndex\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed','LocationIndex']]

g\_LocationIndex = sns.pairplot( dataScatterPlot\_LocationIndex, vars = Variables\_LocationIndex, hue = "prediction", diag\_kind = 'kde',palette= "husl", plot\_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'})

A screenshot of a cell phone

Description automatically generated

Using the LocationIndex in the features to form the clustering, there is a distortion in the clusters and clear clusters are not formed, with LocationIndex and Bytes\_Transferred, LocationIndex and Session\_connection\_Time, and LocationIndex and Servers\_Corrupted. This might be the result of Multicollinearity of these three variables nmely, Session\_Connection\_Time, Servers\_Corrupted and Bytes\_Transferred.

Also, The pairplot between Bytes\_Transferred and Session\_Connection\_Time is also quite distorted, this is again due to high correlation among these two variables.

But, elsewhere, two segregated clusters are formed, again confirming the presence of two hackers and not three.

**# With LocationCoded**

dataScatterPlot\_LocationCoded = clusters\_LocationCoded\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed','LocationCoded','prediction']]

Variables\_LocationCoded = clusters\_LocationCoded\_pd[['Session\_Connection\_Time', 'Bytes Transferred', 'Kali\_Trace\_Used','Servers\_Corrupted', 'Pages\_Corrupted','WPM\_Typing\_Speed']]

g\_LocationCoded = sns.pairplot( dataScatterPlot\_LocationCoded, hue = "prediction", diag\_kind = 'kde',palette= "husl", plot\_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'})

Look at the plot on the next page

A close up of a map

Description automatically generated

If we look at the pairplots above, we see that due to LocationCoded( using OneHotEncoder) the k-value was too high (k=192), which is not an optimal way of working, as when the data has high cardinality, OneHotEncoder should not be used.

Still, tried to experiment with the features containing Location OneHotEncoded.

We can see the pairplots of highly correlated variables, the clusters are a little distorted, Bytes\_Transferred, Session\_Connection\_Time, Servers\_Corrupted, these three variables show multicollinearity hence distorted clusters.

Elsewhere, two clusters confirming two hackers, and not three.

**RESULT:**

From the K-Means Clustering of the Hacking Data, I found out that there were **TWO HACKERS** and not three. And the tird suspect had nothing to do with the hacking attempts.