# Name of the Course: Python and Deep Learning

# Team ID-06

# Team Members:


### 1.Lakshmana Kumar Mettu (class ID-16)

### 2.Pavan Kumar Chongala (class ID-04)

### 3.Tarun Teja Kasturi (Class ID-11)


---

**Objective: The main objective of lab-2 is to know about machine learning concepts such are supervised and unsupervised approaches using different data sets as well as got insights on natural language processing techniques such are tokenization, lemmetization, stemming,n-gram techniques.**

**Video Link:**Question 1&2

**Video link for 3 and 4 Questions:**Questions 3&4

**Documentation Link:**Documentation

**Source Code:**Source Code

**Task1: To Perform exploratory data analysis on the data set (like Handling null values, removing the features not correlated to the target class, encoding the categorical features, ...)and Applying the three classification algorithms Naïve Bayes', SVM and KNN on the chosen data set and report which classifier gives better result.**

**1.a:Performing EDA on train dataset is shown below in the code snippets and explained each line with comments**

```
import pandas as pd
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
import warnings
warnings.simplefilter("ignore")

# Load the train DataFrames using pandas
train = pd.read_csv('C:/Users/laksh/PycharmProjects/Python_Lesson5/data/train.csv')

print("Train_Set")
print(train.head())
print("\n")

#print("Train_Set description")
#print(train.describe())
#print("\n")

print(train.columns.values)

# For identifying no.of null values in the train set
train.isna().head()

print("NULL values in the train ")
print(train.isna().sum())
print("\n")

# Fill missing or null values with mean column values in the train set
train.fillna(train.mean(), inplace=True)

print(train.isna().sum())

train.info()
#removing the features not correlated to the target class,
train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
```

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
labelEncoder = preprocessing.LabelEncoder()
labelEncoder.fit(train['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])


train.info()
print(train.head())

#caluculating the svm
feature_cols = ['PassengerId' ,'Pclass', 'Age' ,'SibSp' ,'Parch' ,'Fare' , 'Sex']
from sklearn.svm import SVC

X = train[feature_cols]
y = train.Survived
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

svm = SVC()                      # Fitting SVM model to the data
expected = train.Survived

# Training the Model on Testing Set

svm.fit(X_train, y_train)
print(expected)

predicted_label = svm.predict(train[feature_cols])          # Making Prediction based on X, Y
print(predicted_label)

# Cross Validation compare the predicted and expected values
# Matrix which is used to find the accuracy of classification
print(metrics.confusion_matrix(expected, predicted_label))

# Cross Validation compare the predicted and expected values
print(metrics.classification_report(expected, predicted_label))
```

## 1.b:Applying calssification algorithms such are Naive bayes,Knn and SVM model

```
print('Accuracy of SVM classifier on training set: {:.2f}'
     .format(svm.score(X_train, y_train)))
# Evaluating the Model based on Testing Part
print('Accuracy of SVM classifier on test set: {:.2f}'
     .format(svm.score(X_test, y_test)))

#calculating navie bayes model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()       # Fitting gaussian Naive Bayes model to the data
model.fit(train[feature_cols], train.Survived)

expected = train.Survived            # Making Prediction based on X, Y
predicted = model.predict(train[feature_cols])


print(metrics.classification_report(expected, predicted))    #Cross Validation compare the predicted and expected values

print(metrics.confusion_matrix(expected, predicted))         # Matrix which is used to find the accuracy of classification
X_train, X_test, Y_train, Y_test = train_test_split(train[feature_cols], train.Survived, test_size=0.2, random_state=0)

model.fit(X_train, Y_train)                                  #  Model on Training Set

Y_predicted = model.predict(X_test)                           #  Model on Testing Set

print("accuracy using Gaussian navie bayes Model is ", metrics.accuracy_score(Y_test, Y_predicted) * 100)


#caluculating knn
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X, y)
```

```python


#caluculating knn
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X, y)

# predict the response values for the observations in X
logreg.predict(X)


# store the predicted response values
y_pred = logreg.predict(X)

# check how many predictions were generated
len(y_pred)


from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
y_pred = knn.predict(X)
print("accuracy using knn Model is ",metrics.accuracy_score(y, y_pred))
```

## Output for the above task

```
labques-1
C:\Users\laksh\AppData\Local\Programs\Python\Python36\python.exe C:/Users/laksh/PycharmProjects/Python_Lesson6/labques-1.py
Train_Set
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]


['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
NULL values in the train
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Sex            891 non-null int32
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
dtypes: float64(2), int32(1), int64(5)
memory usage: 52.3 KB
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.92       549
           1       0.98      0.75      0.85       342

   micro avg       0.90      0.90      0.90       891
   macro avg       0.92      0.87      0.89       891
weighted avg       0.91      0.90      0.90       891

Accuracy of SVM classifier on training set: 1.00
Accuracy of SVM classifier on test set: 0.61
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       549
           1       0.74      0.71      0.72       342

   micro avg       0.79      0.79      0.79       891
   macro avg       0.78      0.78      0.78       891
weighted avg       0.79      0.79      0.79       891

[[465  84]
 [100 242]]
accuracy using Gaussian navie bayes Model is  79.3296089385475
accuracy using knn Model is  0.7530864197530864
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.92       549
           1       0.98      0.75      0.85       342

   micro avg       0.90      0.90      0.90       891
   macro avg       0.92      0.87      0.89       891
weighted avg       0.91      0.90      0.90       891

Accuracy of SVM classifier : 0.61
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       549
           1       0.74      0.71      0.72       342

   micro avg       0.79      0.79      0.79       891
   macro avg       0.78      0.78      0.78       891
weighted avg       0.79      0.79      0.79       891

[[465  84]
 [100 242]]
accuracy using Gaussian navie bayes Model is  79.3296089385475
accuracy using knn Model is  0.7530864197530864
```

**Observations On Task1:** 1. In above task EDA is applied on train dataset and did label encoding to convert string to float data type and removed some columns not related to target class and replace the null values with mean which were explained in the code snippets. 2.After applying various classification algorithms ,based on the accuracy we can say that Navie bayes is efficient than KNN and SVM model which are explained in the code snippets and video.

## Task:2 Applying K-means on the data set and visualize the clusters using seaborn and Reporting which K is the best using the elbow method also Evaluation with silhouette score for unsupervised approaches.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.metrics import silhouette_score
import seaborn as sns  # importing seaborn packages for plotting the graph
import warnings

warnings.simplefilter("ignore")

df = pd.read_csv('C:/Users/laksh/PycharmProjects/Python_Lesson5/data/Customers.csv')  # read the customers data

print("customers data set")
print(df.head())
print("\n")

print(df.columns.values)

# For identifying no.of null values in the customer set
df.isna().head()

print("NULL values in the train ")
print(df.isna().sum())
print("\n")

# Fill missing or null values with mean column values in the customer set
df.fillna(df.mean(), inplace=True)

df.info()
print("\n")
```

```python
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing

labelEncoder = preprocessing.LabelEncoder()
labelEncoder.fit(df['Gender'])
df['Gender'] = labelEncoder.transform(df['Gender'])

#removing the features not correlated to the data
df.drop(["CustomerID"], axis=1, inplace=True)

df.info()
print("\n")

#using Age, Annual Income and Spending Score for clustering customers
from mpl_toolkits.mplot3d import Axes3D

sns.set_style("white")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age, df["Annual Income (k$)"], df["Spending Score (1-100)"], c='blue', s=60)
ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()

#appliying k-means and calucuating silhoutte score
from sklearn.cluster import KMeans

wcss = []
```

```python
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(df.iloc[:, 1:])
    wcss.append(kmeans.inertia_)
    score = silhouette_score(df, kmeans.labels_, metric='euclidean')
    print("For n_clusters = {}, silhouette score is {})".format(k, score))

plt.figure(figsize=(12, 6))
plt.grid()
plt.plot(range(2, 11), wcss, linewidth=2, color="red", marker="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1, 11, 1))
plt.ylabel("WCSS")
plt.show()

#kmeans clusturring
km = KMeans(n_clusters=5)
clusters = km.fit_predict(df.iloc[:, 1:])

df["label"] = clusters
#cluturing represntation of the age annual income and spending score
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.label == 0], df["Spending Score (1-100)"][df.label == 0], c='blue', s=60)
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.label == 1], df["Spending Score (1-100)"][df.label == 1], c='red', s=60)
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.label == 2], df["Spending Score (1-100)"][df.label == 2], c='green', s=60)
ax.scatter(df.Age[df.label == 3], df["Annual Income (k$)"][df.label == 3], df["Spending Score (1-100)"][df.label == 3], c='orange', s=60)
ax.scatter(df.Age[df.label == 4], df["Annual Income (k$)"][df.label == 4], df["Spending Score (1-100)"][df.label == 4],  c='purple', s=60)
ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()
```

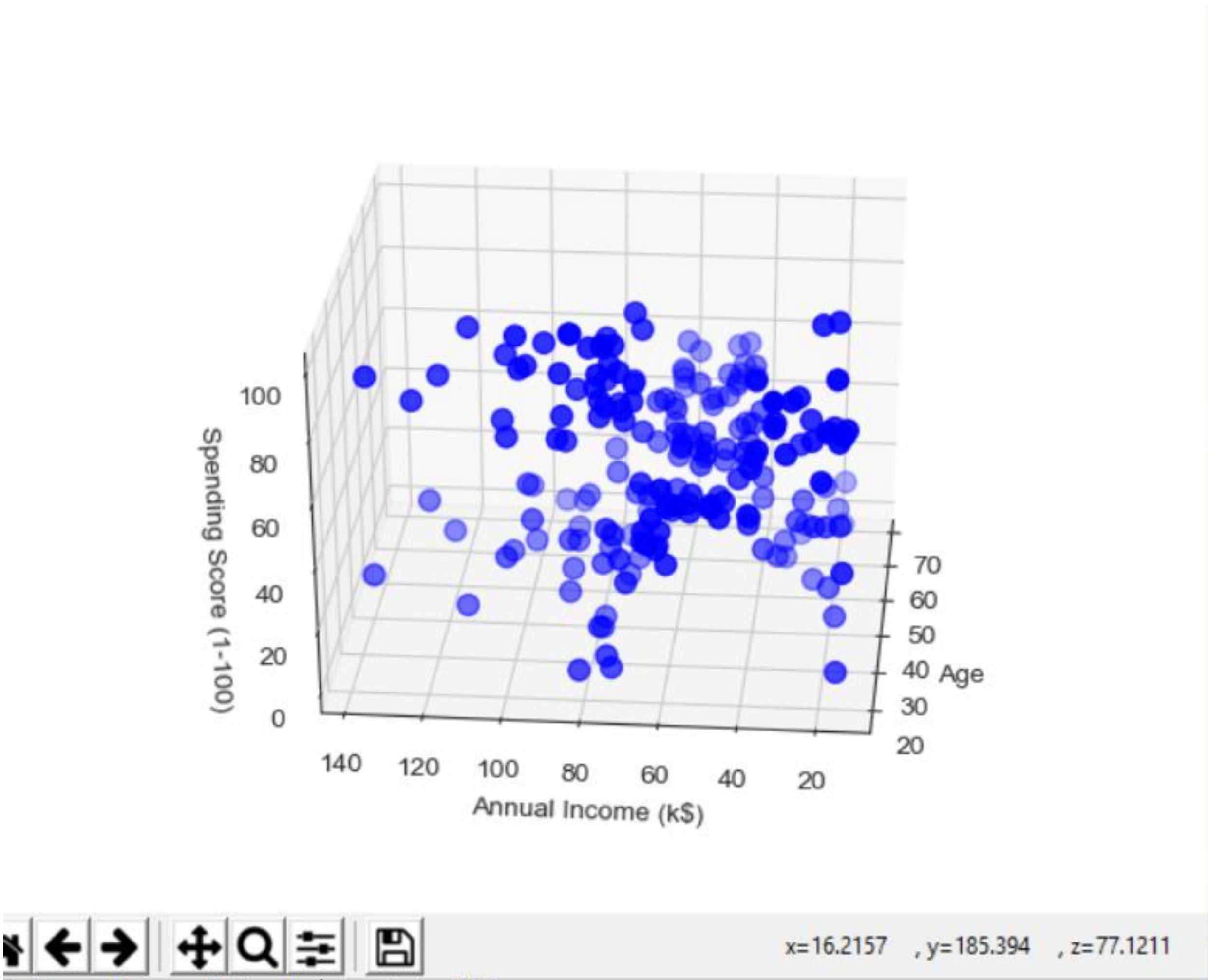
**Output Snippets along with graphs plotted using seaborn and k means**

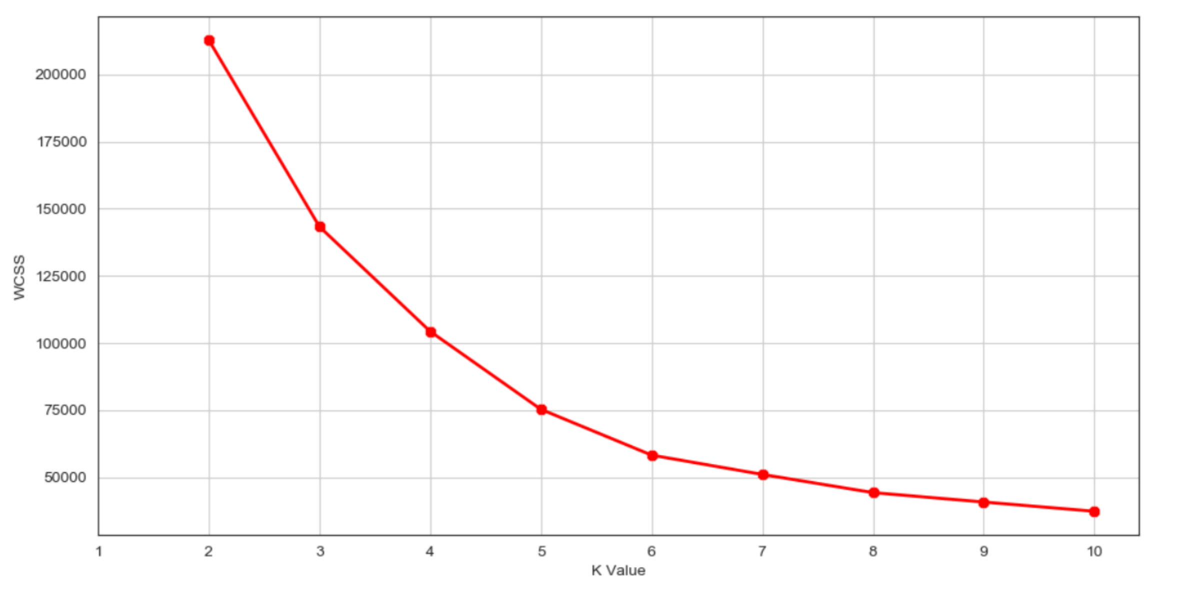
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
CustomerID                200 non-null int64
Gender                    200 non-null object
Age                       200 non-null int64
Annual Income (k$)        200 non-null int64
Spending Score (1-100)    200 non-null int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

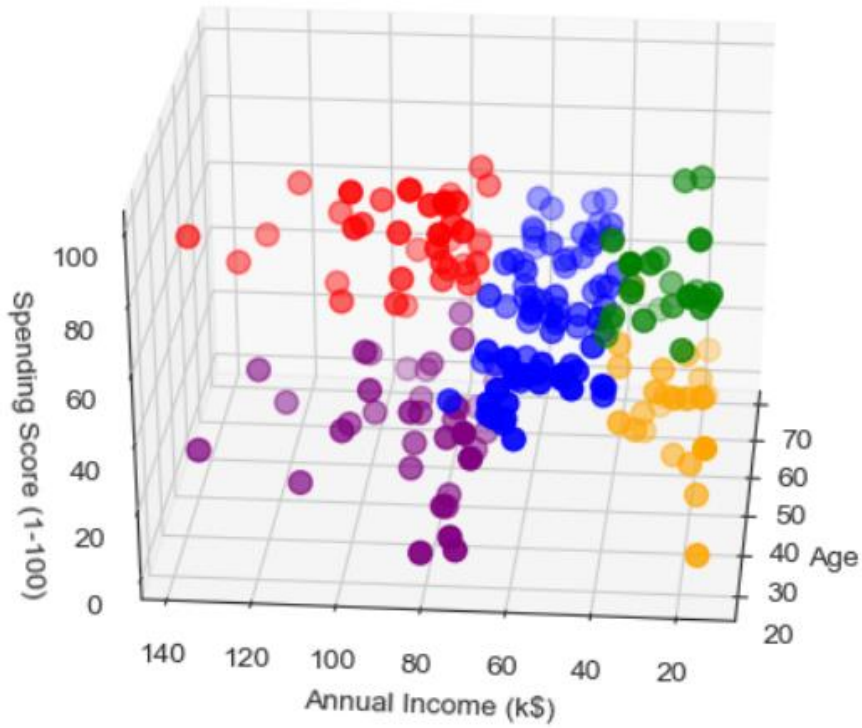
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
Gender                    200 non-null int32
Age                       200 non-null int64
Annual Income (k$)        200 non-null int64
Spending Score (1-100)    200 non-null int64
dtypes: int32(1), int64(3)
memory usage: 5.5 KB
```

**graph**

```
For n_clusters = 2, silhouette score is 0.29307334005502633)
For n_clusters = 3, silhouette score is 0.383798873822341)
For n_clusters = 4, silhouette score is 0.4052954330641215)
For n_clusters = 5, silhouette score is 0.4440669204743008)
For n_clusters = 6, silhouette score is 0.45205475380756527)
For n_clusters = 7, silhouette score is 0.44096462877395787)
For n_clusters = 8, silhouette score is 0.41565411348592207)
For n_clusters = 9, silhouette score is 0.3831792695025229)
For n_clusters = 10, silhouette score is 0.4040290051295149)
```


Spending Score (1-100)
100
80
60
40
20
0
140
120
100
80
60
40
20
Annual Income (k$)
70
60
50
40
30
20
Age
x=16.2157 , y=185.394 , z=77.1211


WCSS
200000
175000
150000
125000
100000
75000
50000
1
2
3
4
5
6
7
8
9
10
K Value




**Observations on task-2:**As we were plotted elbow graph done by elbow method shown in the code snippet also shown the k-means graph.Coming to the best value of 'k' ,As the k value increases the silhouette score increased and at a particular value again it starts decreasing.Hence the best value of K for this customer data set is K=6 , corresponding score is 0.4520 .It is about Task-2.

**Task-3:Saving text file from given url ,performing NLP techniques such are a. Read the data from a file b. Tokenize the text into words and apply lemmatization technique on each word. c. Find all the trigrams for the words. d. Extract the top 10 of the most repeated trigrams based on their count. e. Go through the text in the file f. Find all the sentences with the most repeated tri-gramsg. Extract those sentences and concatenate h. Print the concatenated result.**

## Code snippets along with comments are given below

```
import urllib

from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import ngrams, FreqDist
# Reading Text from the URL
wikiURL = "https://umkc.app.box.com/s/7by0f4540cdbdp3pm60h5fxxffefsvrw"
openURL = urllib.request.urlopen(wikiURL)

# Assigning Parsed Web Page into a Variable
soup = BeautifulSoup(openURL.read(), "lxml")

# Kill all script and style elements
for script in soup(["script", "style"]):
    # Rip it Off
    script.extract()

# get text
text = soup.body.get_text()

# break into lines and remove leading and trailing space on each
lines = (line.strip() for line in text.splitlines())
# break multi-headlines into a line each
chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
# drop blank lines
text = ' '.join(chunk for chunk in chunks if chunk)
```

```
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import ngrams, FreqDist

#reading the data from a file
with open('C:/Users/laksh/PycharmProjects/Python_Lesson6/nlp_input.txt','r') as text_file:
    fileData = text_file.read()

# Word Tokenization - to extarct each word from the text
tokens = word_tokenize(fileData)

# Applying Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatizerOutput = []
print("Lemmatization Output : \n")
for tok in tokens:
    #itearting through each word and lematizing and appending it to list
    lemmatizerOutput.append(lemmatizer.lemmatize(str(tok)))
print(lemmatizerOutput)

# performing trigram on the  Lemmatizer Output
print("trigrams :\n")
trigramsOutput = []
for tri in ngrams(tokens, 3):
    # Fetching trigrams using 'ngrams' method and Iterating it and appending it to list
    trigramsOutput.append(tri)
print(trigramsOutput)

# triGram- Word Frequency
# Using trigramOutput fetch the WordFreq Details
wordFreq = FreqDist(trigramsOutput)
# Getting Most Common Words and Printing them - Will get the Counts from top to least
mostCommon = wordFreq.most_common()
print("triGrams Frequency (From Top to Least) : \n", mostCommon)
```

```python
# Fetching the Top 10 trigrams
top10 = wordFreq.most_common(10)
print("Top 10 triGrams : \n", top10)


# Getting Sentences using Sentence Tokenization
sentTokens = sent_tokenize(fileData)
# Creating an Array to append the sentence
concatenatedArray = []
# Iterating the Sentences
for sentence in sentTokens:
    # Iterating the trirams present
    for a, b, c in trigramsOutput:
        # Iterating the Top 10 triGrams
        for ((d, e, f), length) in top10:
            # Comparing the each with each of the Top 10 trigram
            if(a, b, c == d, e, f):
                concatenatedArray.append(sentence)

print("Concatenated Array : ",concatenatedArray)
print("Maximum of Concatenated Array : ", max(concatenatedArray))
```

## Output Snippets:

```
Lemmatization Output :

['Regression', 'analysis', 'is', 'a', 'statistical', 'technique', 'that', 'model', 'and', 'approximates', 'the', 'relationship', 'between', 'a', 'dependent', 'and', 'one', 'or', 'more', 'independent
 'variable', '.', 'This', 'article', 'will', 'quickly', 'introduce', 'three', 'commonly', 'used', 'regression', 'model', 'using', 'R', 'and', 'the', 'Boston', 'housing', 'data-set', ':', 'Ridge', ',
 'Lasso', ',', 'and', 'Elastic', 'Net', '.', 'First', 'we', 'need', 'to', 'understand', 'the', 'basic', 'of', 'regression', 'and', 'what', 'parameter', 'of', 'the', 'equation', 'are', 'changed',
 'when', 'using', 'a', 'specific', 'model', '.', 'Simple', 'linear', 'regression', ',', 'also', 'known', 'a', 'ordinary', 'least', 'square', '(', 'OLS', ')', 'attempt', 'to', 'minimize', 'the', 'sum
 'of', 'error', 'squared', '.', 'The', 'error', 'in', 'this', 'case', 'is', 'the', 'difference', 'between', 'the', 'actual', 'data', 'point', 'and', 'it', 'predicted', 'value', '.', 'Visualization',
 'of', 'the', 'squared', 'error', '(', 'from', 'Setosa.io', ')', 'The', 'equation', 'for', 'this', 'model', 'is', 'referred', 'to', 'a', 'the', 'cost', 'function', 'and', 'is', 'a', 'way', 'to',
 'find', 'the', 'optimal', 'error', 'by', 'minimizing', 'and', 'measuring', 'it', '.', 'The', 'gradient', 'descent', 'algorithm', 'is', 'used', 'to', 'find', 'the', 'optimal', 'cost', 'function', 'b
 'going', 'over', 'a', 'number', 'of', 'iteration', '.', 'But', 'the', 'data', 'we', 'need', 'to', 'define', 'and', 'analyze', 'is', 'not', 'always', 'so', 'easy', 'to', 'characterize', 'with', 'the
 'base', 'OLS', 'model', '.', 'Equation', 'for', 'least', 'ordinary', 'square', 'One', 'situation', 'is', 'the', 'data', 'showing', 'multi-collinearity', ',', 'this', 'is', 'when', 'predictor',
 'variable', 'are', 'correlated', 'to', 'each', 'other', 'and', 'to', 'the', 'response', 'variable', '.', 'To', 'picture', 'this', 'let', "'", 's', 'say', 'we', "'", 're', 'doing', 'a', 'study',
 'that', 'look', 'at', 'a', 'response', 'variable', '?', '—', '?', 'patient', 'weight', ',', 'and', 'our', 'predictor', 'variable', 'would', 'be', 'height', ',', 'sex', ',', 'and', 'diet', '.', 'Th
 'problem', 'here', 'is', 'that', 'height', 'and', 'sex', 'are', 'also', 'correlated', 'and', 'can', 'inflate', 'the', 'standard', 'error', 'of', 'their', 'coefficient', 'which', 'may', 'make', 'the
 'seem', 'statistically', 'insignificant', '.', 'To', 'produce', 'a', 'more', 'accurate', 'model', 'of', 'complex', 'data', 'we', 'can', 'add', 'a', 'penalty', 'term', 'to', 'the', 'OLS', 'equation'
 '.', 'A', 'penalty', 'add', 'a', 'bias', 'towards', 'certain', 'value', '.', 'These', 'are', 'known', 'a', 'L1', 'regularization', '(', 'Lasso', 'regression', ')', 'and', 'L2', 'regularization', '(
 'ridge', 'regression', ')', '.The', 'best', 'model', 'we', 'can', 'hope', 'to', 'come', 'up', 'with', 'minimizes', 'both', 'the', 'bias', 'and', 'the', 'variance', ':', 'Ridge', 'regression', 'us',
 'L2', 'regularization', 'which', 'add', 'the', 'following', 'penalty', 'term', 'to', 'the', 'OLS', 'equation', '.', 'L2', 'regularization', 'penalty', 'term', 'The', 'L2', 'term', 'is', 'equal', 't
 'the', 'square', 'of', 'the', 'magnitude', 'of', 'the', 'coefficient', '.', 'In', 'this', 'case', 'if', 'lambda', '(', '?', ')', 'is', 'zero', 'then', 'the', 'equation', 'is', 'the', 'basic', 'OLS'
 'but', 'if', 'it', 'is', 'greater', 'than', 'zero', 'then', 'we', 'add', 'a', 'constraint', 'to', 'the', 'coefficient', '.', 'This', 'constraint', 'result', 'in', 'minimized', 'coefficient', '(',
 'aka', 'shrinkage', ')', 'that', 'trend', 'towards', 'zero', 'the', 'larger', 'the', 'value', 'of', 'lambda', '.', 'Shrinking', 'the', 'coefficient', 'lead', 'to', 'a', 'lower', 'variance', 'and',
 'in', 'turn', 'a', 'lower', 'error', 'value', '.', 'Therefore', 'Ridge', 'regression', 'decrease', 'the', 'complexity', 'of', 'a', 'model', 'but', 'doe', 'not', 'reduce', 'the', 'number', 'of',
 'variable', ',', 'it', 'rather', 'just', 'shrink', 'their', 'effect', '.', 'Lasso', 'regression', 'Lasso', 'regression', 'us', 'the', 'L1', 'penalty', 'term', 'and', 'stand', 'for', 'Least',
 'Absolute', 'Shrinkage', 'and', 'Selection', 'Operator', '.', 'The', 'penalty', 'applied', 'for', 'L2', 'is', 'equal', 'to', 'the', 'absolute', 'value', 'of', 'the', 'magnitude', 'of', 'the',
 'coefficient', ':', 'L1', 'regularization', 'penalty', 'term', 'Similar', 'to', 'ridge', 'regression', ',', 'a', 'lambda', 'value', 'of', 'zero', 'spit', 'out', 'the', 'basic', 'OLS', 'equation', '
 'however', 'given', 'a', 'suitable', 'lambda', 'value', 'lasso', 'regression', 'can', 'drive', 'some', 'coefficient', 'to', 'zero', '.', 'The', 'larger', 'the', 'value', 'of', 'lambda', 'the', 'mor
 'feature', 'are', 'shrunk', 'to', 'zero', '.', 'This', 'can', 'eliminate', 'some', 'feature', 'entirely', 'and', 'give', 'u', 'a', 'subset', 'of', 'predictor', 'that', 'help', 'mitigate',
 'multi-collinearity', 'and', 'model', 'complexity', '.', 'Predictors', 'not', 'shrunk', 'towards', 'zero', 'signify', 'that', 'they', 'are', 'important', 'and', 'thus', 'L1', 'regularization',
 'allows', 'for', 'feature', 'selection', '(', 'sparse', 'selection', ')', '.', 'A', 'third', 'commonly', 'used', 'model', 'of', 'regression', 'is', 'the', 'Elastic', 'Net', 'which', 'incorporates',
 'penalty', 'from', 'both', 'L1', 'and', 'L2', 'regularization', ':', 'In', 'addition', 'to', 'setting', 'and', 'choosing', 'a', 'lambda', 'value', 'elastic', 'net', 'also', 'allows', 'u', 'to',
 'tune', 'the', 'alpha', 'parameter', 'where', '?', '?', '=', '0', 'corresponds', 'to', 'ridge', 'and', '?', '?', '=', '1', 'to', 'lasso', '.', 'Simply', 'put', ',', 'if', 'you', 'plug', 'in', '0',
 'for', 'alpha', ',', 'the', 'penalty', 'function', 'reduces', 'to', 'the', 'L1', '(', 'ridge', ')', 'term', 'and', 'if', 'we', 'set', 'alpha', 'to', '1', 'we', 'get', 'the', 'L2', '(', 'lasso', ')'
 'term', '.', 'Therefore', 'we', 'can', 'choose', 'an', 'alpha', 'value', 'between', '0', 'and', '1', 'to', 'optimize', 'the', 'elastic', 'net', '.', 'Effectively', 'this', 'will', 'shrink', 'some',
 'coefficient', 'and', 'set', 'some', 'to', '0', 'for', 'sparse', 'selection', '.', 'As', 'we', 'mentioned', 'in', 'the', 'previous', 'section', ',', 'lambda', 'value', 'have', 'a', 'large', 'effect
 'on', 'coefficient', 'so', 'now', 'we', 'will', 'compute', 'and', 'chose', 'a', 'suitable', 'one', '.', 'Here', 'we', 'perform', 'a', 'cross', 'validation', 'and', 'take', 'a', 'peek', 'at', 'the',
 'lambda', 'value', 'corresponding', 'to', 'the', 'lowest', 'prediction', 'error', 'before', 'fitting', 'the', 'data', 'to', 'the', 'model', 'and', 'viewing', 'the', 'coefficient', '.', 'We', 'can'
```

```
trigrams :

[('Regression', 'analysis', 'is'), ('analysis', 'is', 'a'), ('is', 'a', 'statistical'), ('a', 'statistical', 'technique'), ('statistical', 'technique', 'that'), ('technique', 'that', 'models'),
 ('that', 'models', 'and'), ('models', 'and', 'approximates'), ('and', 'approximates', 'the'), ('approximates', 'the', 'relationship'), ('the', 'relationship', 'between'), ('relationship', 'between',
 'a'), ('between', 'a', 'dependent'), ('a', 'dependent', 'and'), ('dependent', 'and', 'one'), ('and', 'one', 'or'), ('one', 'or', 'more'), ('or', 'more', 'independent'), ('more', 'independent',
 'variables'), ('independent', 'variables', '.'), ('variables', '.', 'This'), ('.', 'This', 'article'), ('This', 'article', 'will'), ('article', 'will', 'quickly'), ('will', 'quickly', 'introduce'),
 ('quickly', 'introduce', 'three'), ('introduce', 'three', 'commonly'), ('three', 'commonly', 'used'), ('commonly', 'used', 'regression'), ('used', 'regression', 'models'), ('regression', 'models',
 'using'), ('models', 'using', 'R'), ('using', 'R', 'and'), ('R', 'and', 'the'), ('and', 'the', 'Boston'), ('the', 'Boston', 'housing'), ('Boston', 'housing', 'data-set'), ('housing', 'data-set', ':'
 ('data-set', ':', 'Ridge'), (':', 'Ridge', ','), ('Ridge', ',', 'Lasso'), (',', 'Lasso', ','), ('Lasso', ',', 'and'), (',', 'and', 'Elastic'), ('and', 'Elastic', 'Net'), ('Elastic', 'Net', '.'),
 ('Net', '.', 'First'), ('.', 'First', 'we'), ('First', 'we', 'need'), ('we', 'need', 'to'), ('need', 'to', 'understand'), ('to', 'understand', 'the'), ('understand', 'the', 'basics'), ('the',
 'basics', 'of'), ('basics', 'of', 'regression'), ('of', 'regression', 'and'), ('regression', 'and', 'what'), ('and', 'what', 'parameters'), ('what', 'parameters', 'of'), ('parameters', 'of', 'the'),
 ('of', 'the', 'equation'), ('the', 'equation', 'are'), ('equation', 'are', 'changed'), ('are', 'changed', 'when'), ('changed', 'when', 'using'), ('when', 'using', 'a'), ('using', 'a', 'specific'),
 ('a', 'specific', 'model'), ('specific', 'model', '.'), ('model', '.', 'Simple'), ('.', 'Simple', 'linear'), ('Simple', 'linear', 'regression'), ('linear', 'regression', ','), ('regression', ',',
 'also'), (',', 'also', 'known'), ('also', 'known', 'as'), ('known', 'as', 'ordinary'), ('as', 'ordinary', 'least'), ('ordinary', 'least', 'squares'), ('least', 'squares', '('), ('squares', '(',
 'OLS'), ('(', 'OLS', ')'), ('OLS', ')', 'attempts'), (')', 'attempts', 'to'), ('attempts', 'to', 'minimize'), ('to', 'minimize', 'the'), ('minimize', 'the', 'sum'), ('the', 'sum', 'of'), ('sum', 'of
 'error'), ('of', 'error', 'squared'), ('error', 'squared', '.'), ('squared', '.', 'The'), ('.', 'The', 'error'), ('The', 'error', 'in'), ('error', 'in', 'this'), ('in', 'this', 'case'), ('this',
 'case', 'is'), ('case', 'is', 'the'), ('is', 'the', 'difference'), ('the', 'difference', 'between'), ('difference', 'between', 'the'), ('between', 'the', 'actual'), ('the', 'actual', 'data'),
 ('actual', 'data', 'point'), ('data', 'point', 'and'), ('point', 'and', 'its'), ('and', 'its', 'predicted'), ('its', 'predicted', 'value'), ('predicted', 'value', '.'), ('value', '.', 'Visualization
  ('.', 'Visualization', 'of'), ('Visualization', 'of', 'the'), ('of', 'the', 'squared'), ('the', 'squared', 'error'), ('squared', 'error', '('), ('error', '(', 'from'), ('(', 'from', 'Setosa.io'),
 ('from', 'Setosa.io', ')'), ('Setosa.io', ')', 'The'), (')', 'The', 'equation'), ('The', 'equation', 'for'), ('equation', 'for', 'this'), ('for', 'this', 'model'), ('this', 'model', 'is'), ('model',
 'is', 'referred'), ('is', 'referred', 'to'), ('referred', 'to', 'as'), ('to', 'as', 'the'), ('as', 'the', 'cost'), ('the', 'cost', 'function'), ('cost', 'function', 'and'), ('function', 'and', 'is')
 ('and', 'is', 'a'), ('is', 'a', 'way'), ('a', 'way', 'to'), ('way', 'to', 'find'), ('to', 'find', 'the'), ('find', 'the', 'optimal'), ('the', 'optimal', 'error'), ('optimal', 'error', 'by'), ('error
 'by', 'minimizing'), ('by', 'minimizing', 'and'), ('minimizing', 'and', 'measuring'), ('and', 'measuring', 'it'), ('measuring', 'it', '.'), ('it', '.', 'The'), ('.', 'The', 'gradient'), ('The',
 'gradient', 'descent'), ('gradient', 'descent', 'algorithm'), ('descent', 'algorithm', 'is'), ('algorithm', 'is', 'used'), ('is', 'used', 'to'), ('used', 'to', 'find'), ('to', 'find', 'the'), ('find
 'the', 'optimal'), ('the', 'optimal', 'cost'), ('optimal', 'cost', 'function'), ('cost', 'function', 'by'), ('function', 'by', 'going'), ('by', 'going', 'over'), ('going', 'over', 'a'), ('over', 'a'
 'number'), ('a', 'number', 'of'), ('number', 'of', 'iterations'), ('of', 'iterations', '.'), ('iterations', '.', 'But'), ('.', 'But', 'the'), ('But', 'the', 'data'), ('the', 'data', 'we'), ('data',
 'we', 'need'), ('we', 'need', 'to'), ('need', 'to', 'define'), ('to', 'define', 'and'), ('define', 'and', 'analyze'), ('and', 'analyze', 'is'), ('analyze', 'is', 'not'), ('is', 'not', 'always'),
 ('not', 'always', 'so'), ('always', 'so', 'easy'), ('so', 'easy', 'to'), ('easy', 'to', 'characterize'), ('to', 'characterize', 'with'), ('characterize', 'with', 'the'), ('with', 'the', 'base'),
 ('the', 'base', 'OLS'), ('base', 'OLS', 'model'), ('OLS', 'model', '.'), ('model', '.', 'Equation'), ('.', 'Equation', 'for'), ('Equation', 'for', 'least'), ('for', 'least', 'ordinary'), ('least',
 'ordinary', 'squares'), ('ordinary', 'squares', 'One'), ('squares', 'One', 'situation'), ('One', 'situation', 'is'), ('situation', 'is', 'the'), ('is', 'the', 'data'), ('the', 'data', 'showing'),
 ('data', 'showing', 'multi-collinearity'), ('showing', 'multi-collinearity', ','), ('multi-collinearity', ',', 'this'), (',', 'this', 'is'), ('this', 'is', 'when'), ('is', 'when', 'predictor'),
 ('when', 'predictor', 'variables'), ('predictor', 'variables', 'are'), ('variables', 'are', 'correlated'), ('are', 'correlated', 'to'), ('correlated', 'to', 'each'), ('to', 'each', 'other'), ('each'
 'other', 'and'), ('other', 'and', 'to'), ('and', 'to', 'the'), ('to', 'the', 'response'), ('the', 'response', 'variable'), ('response', 'variable', '.'), ('variable', '.', 'To'), ('.', 'To', 'pictur
 , ('To', 'picture', 'this'), ('picture', 'this', 'let'), ('this', 'let', '''), ('let', ''', 's'), (''', 's', 'say'), ('s', 'say', 'we'), ('say', 'we', '''), ('we', ''', 're'), (''', 're', 'doing'),
 ('re', 'doing', 'a'), ('doing', 'a', 'study'), ('a', 'study', 'that'), ('study', 'that', 'looks'), ('that', 'looks', 'at'), ('looks', 'at', 'a'), ('at', 'a', 'response'), ('a', 'response', 'variable
  ('response', 'variable', '?'), ('variable', '?', '—'), ('?', '—', '?'), ('—', '?', 'patient'), ('?', 'patient', 'weight'), ('patient', 'weight', ','), ('weight', ',', 'and'), (',', 'and', 'our'),
```

```
triGrams Frequency (From Top to Least) :
 [(('we', 'need', 'to'), 3), (('the', 'coefficients', '.'), 3), (('?', '?', '='), 3), (('to', 'find', 'the'), 2), (('find', 'the', 'optimal'), 2), (('over', 'a', 'number'), 2), (('a', 'number', 'of')
 2), (('to', 'each', 'other'), 2), (('penalty', 'term', 'to'), 2), (('term', 'to', 'the'), 2), (('to', 'the', 'OLS'), 2), (('the', 'OLS', 'equation'), 2), (('OLS', 'equation', '.'), 2), (('and', 'L2'
 'regularization'), 2), (('regularization', 'penalty', 'term'), 2), (('is', 'equal', 'to'), 2), (('equal', 'to', 'the'), 2), (('of', 'the', 'magnitude'), 2), (('the', 'magnitude', 'of'), 2), (
 ('magnitude', 'of', 'the'), 2), (('of', 'the', 'coefficients'), 2), (('the', 'basic', 'OLS'), 2), (('larger', 'the', 'value'), 2), (('the', 'value', 'of'), 2), (('value', 'of', 'lambda'), 2), (('.',
 'Lasso', 'regression'), 2), (('a', 'lambda', 'value'), 2), (('to', 'zero', '.'), 2), (('us', 'to', 'tune'), 2), (('?', '=', '1'), 2), (('.', 'We', 'can'), 2), (('We', 'can', 'see'), 2), (('Performin
  'Elastic', 'Net'), 2), (('alpha', 'and', 'lambda'), 2), (('Regression', 'analysis', 'is'), 1), (('analysis', 'is', 'a'), 1), (('is', 'a', 'statistical'), 1), (('a', 'statistical', 'technique'), 1),
 (('statistical', 'technique', 'that'), 1), (('technique', 'that', 'models'), 1), (('that', 'models', 'and'), 1), (('models', 'and', 'approximates'), 1), (('and', 'approximates', 'the'), 1), (
 ('approximates', 'the', 'relationship'), 1), (('the', 'relationship', 'between'), 1), (('relationship', 'between', 'a'), 1), (('between', 'a', 'dependent'), 1), (('a', 'dependent', 'and'), 1),
 (('dependent', 'and', 'one'), 1), (('and', 'one', 'or'), 1), (('one', 'or', 'more'), 1), (('or', 'more', 'independent'), 1), (('more', 'independent', 'variables'), 1), (('independent', 'variables',
 '.'), 1), (('variables', '.', 'This'), 1), (('.', 'This', 'article'), 1), (('This', 'article', 'will'), 1), (('article', 'will', 'quickly'), 1), (('will', 'quickly', 'introduce'), 1), (('quickly',
 'introduce', 'three'), 1), (('introduce', 'three', 'commonly'), 1), (('three', 'commonly', 'used'), 1), (('commonly', 'used', 'regression'), 1), (('used', 'regression', 'models'), 1), (('regression'
 'models', 'using'), 1), (('models', 'using', 'R'), 1), (('using', 'R', 'and'), 1), (('R', 'and', 'the'), 1), (('and', 'the', 'Boston'), 1), (('the', 'Boston', 'housing'), 1), (('Boston', 'housing',
 'data-set'), 1), (('housing', 'data-set', ':'), 1), (('data-set', ':', 'Ridge'), 1), ((':', 'Ridge', ','), 1), (('Ridge', ',', 'Lasso'), 1), ((',', 'Lasso', ','), 1), (('Lasso', ',', 'and'), 1),
 ((',', 'and', 'Elastic'), 1), (('and', 'Elastic', 'Net'), 1), (('Elastic', 'Net', '.'), 1), (('Net', '.', 'First'), 1), (('.', 'First', 'we'), 1), (('First', 'we', 'need'), 1), (('need', 'to',
 'understand'), 1), (('to', 'understand', 'the'), 1), (('understand', 'the', 'basics'), 1), (('the', 'basics', 'of'), 1), (('basics', 'of', 'regression'), 1), (('of', 'regression', 'and'), 1),
 (('regression', 'and', 'what'), 1), (('and', 'what', 'parameters'), 1), (('what', 'parameters', 'of'), 1), (('parameters', 'of', 'the'), 1), (('of', 'the', 'equation'), 1), (('the', 'equation',
 'are'), 1), (('equation', 'are', 'changed'), 1), (('are', 'changed', 'when'), 1), (('changed', 'when', 'using'), 1), (('when', 'using', 'a'), 1), (('using', 'a', 'specific'), 1), (('a', 'specific',
 'model'), 1), (('specific', 'model', '.'), 1), (('model', '.', 'Simple'), 1), (('.', 'Simple', 'linear'), 1), (('Simple', 'linear', 'regression'), 1), (('linear', 'regression', ','), 1), (
 ('regression', ',', 'also'), 1), ((',', 'also', 'known'), 1), (('also', 'known', 'as'), 1), (('known', 'as', 'ordinary'), 1), (('as', 'ordinary', 'least'), 1), (('ordinary', 'least', 'squares'), 1),
 (('least', 'squares', '('), 1), (('squares', '(', 'OLS'), 1), (('(', 'OLS', ')'), 1), (('OLS', ')', 'attempts'), 1), ((')', 'attempts', 'to'), 1), (('attempts', 'to', 'minimize'), 1), (('to',
 'minimize', 'the'), 1), (('minimize', 'the', 'sum'), 1), (('the', 'sum', 'of'), 1), (('sum', 'of', 'error'), 1), (('of', 'error', 'squared'), 1), (('error', 'squared', '.'), 1), (('squared', '.',
 'The'), 1), (('.', 'The', 'error'), 1), (('The', 'error', 'in'), 1), (('error', 'in', 'this'), 1), (('in', 'this', 'case'), 1), (('this', 'case', 'is'), 1), (('case', 'is', 'the'), 1), (('is', 'the'
 'difference'), 1), (('the', 'difference', 'between'), 1), (('difference', 'between', 'the'), 1), (('between', 'the', 'actual'), 1), (('the', 'actual', 'data'), 1), (('actual', 'data', 'point'), 1),
 (('data', 'point', 'and'), 1), (('point', 'and', 'its'), 1), (('and', 'its', 'predicted'), 1), (('its', 'predicted', 'value'), 1), (('predicted', 'value', '.'), 1), (('value', '.', 'Visualization'),
 1), (('.', 'Visualization', 'of'), 1), (('Visualization', 'of', 'the'), 1), (('of', 'the', 'squared'), 1), (('the', 'squared', 'error'), 1), (('squared', 'error', '('), 1), (('error', '(', 'from'),
 1), (('(', 'from', 'Setosa.io'), 1), (('from', 'Setosa.io', ')'), 1), (('Setosa.io', ')', 'The'), 1), ((')', 'The', 'equation'), 1), (('The', 'equation', 'for'), 1), (('equation', 'for', 'this'), 1)
 (('for', 'this', 'model'), 1), (('this', 'model', 'is'), 1), (('model', 'is', 'referred'), 1), (('is', 'referred', 'to'), 1), (('referred', 'to', 'as'), 1), (('to', 'as', 'the'), 1), (('as', 'the',
 'cost'), 1), (('the', 'cost', 'function'), 1), (('cost', 'function', 'and'), 1), (('function', 'and', 'is'), 1), (('and', 'is', 'a'), 1), (('is', 'a', 'way'), 1), (('a', 'way', 'to'), 1), (('way',
 'to', 'find'), 1), (('the', 'optimal', 'error'), 1), (('optimal', 'error', 'by'), 1), (('error', 'by', 'minimizing'), 1), (('by', 'minimizing', 'and'), 1), (('minimizing', 'and', 'measuring'), 1),
 (('and', 'measuring', 'it'), 1), (('measuring', 'it', '.'), 1), (('it', '.', 'The'), 1), (('.', 'The', 'gradient'), 1), (('The', 'gradient', 'descent'), 1), (('gradient', 'descent', 'algorithm'), 1)
 (('descent', 'algorithm', 'is'), 1), (('algorithm', 'is', 'used'), 1), (('is', 'used', 'to'), 1), (('used', 'to', 'find'), 1), (('the', 'optimal', 'cost'), 1), (('optimal', 'cost', 'function'), 1),
 (('cost', 'function', 'by'), 1), (('function', 'by', 'going'), 1), (('by', 'going', 'over'), 1), (('going', 'over', 'a'), 1), (('number', 'of', 'iterations'), 1), (('of', 'iterations', '.'), 1),
 (('iterations', '.', 'But'), 1), (('.', 'But', 'the'), 1), (('But', 'the', 'data'), 1), (('the', 'data', 'we'), 1), (('data', 'we', 'need'), 1), (('need', 'to', 'define'), 1), (('to', 'define',
 'and'), 1), (('define', 'and', 'analyze'), 1), (('and', 'analyze', 'is'), 1), (('analyze', 'is', 'not'), 1), (('is', 'not', 'always'), 1), (('not', 'always', 'so'), 1), (('always', 'so', 'easy'), 1)
```

```
Top 10 triGrams :
 [(('we', 'need', 'to'), 3), (('the', 'coefficients', '.'), 3), (('?', '?', '='), 3), (('to', 'find', 'the'), 2), (('find', 'the', 'optimal'), 2), (('over', 'a', 'number'), 2), (('a', 'number', 'of')
 2), (('to', 'each', 'other'), 2), (('penalty', 'term', 'to'), 2), (('term', 'to', 'the'), 2)]
```

```
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the
highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regres:
also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.'
'Lasso regression also showed the highest R² value.', 'Lasso regression also showed the highest R² value.']
4aximum of Concatenated Array :  is zero then the equation is the basic OLS but if it is greater than zero then we add a constraint to the coefficients.
```

## Task-4:To Create Multiple Regression by Diabetes dataset also (we did on train dataset) also Evaluate the model using RMSE and R2 and provide report on improvement before and after the EDA.

## Code snippets along with comments:

```
import pandas as pd
import warnings
warnings.simplefilter("ignore")
# Importing the dataset
dataset = pd.read_csv('diabetes.csv')
dataset.describe()
dataset["Insulin"].value_counts()
dataset.groupby(['Insulin', 'BMI']).mean()

dataset = dataset.fillna(dataset.mean())



#X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 2].values
X = dataset.drop(['Pregnancies','Insulin','SkinThickness','BMI'],axis=1)
#dataset = dataset.fillna(dataset.mean())



#df = df_train.drop(['Summary','Daily Summary'],axis=1)

#X = pd.get_dummies(X, columns=["Precip Type"])
#before EDA
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 = 0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
model = regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score
print("Variance score: %.2f" % r2_score(y_test,y_pred))
print("Mean squared error: %.2f" % mean_squared_error(y_test,y_pred))
```

## Code for train dataset before and after EDA

```
import pandas as pd
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
import warnings
warnings.simplefilter("ignore")

# Load the train DataFrames using pandas
train = pd.read_csv('C:/Users/laksh/PycharmProjects/Python_Lesson5/data/train.csv')

print("Train_Set")
print(train.head())
print("\n")

#print("Train_Set description")
#print(train.describe())
#print("\n")

print(train.columns.values)

# For identifying no.of null values in the train set
train.isna().head()

print("NULL values in the train ")
print(train.isna().sum())
print("\n")

# Fill missing or null values with mean column values in the train set
train.fillna(train.mean(), inplace=True)

print(train.isna().sum())

train.info()
#removing the features not correlated to the target class,
train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
```

```python
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
labelEncoder = preprocessing.LabelEncoder()
labelEncoder.fit(train['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])


train.info()
print(train.head())


feature_cols = ['PassengerId' ,'Pclass', 'Age' ,'SibSp' ,'Parch' ,'Fare' , 'Sex']
from sklearn.svm import SVC

X = train[feature_cols]
y = train.Survived
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
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 = 0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
model = regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score
print("Variance score: %.2f" % r2_score(y_test,y_pred))
print("Mean squared error: %.2f" % mean_squared_error(y_test,y_pred))
```

## Output Snippets:

```
Name: Insulin, dtype: float64
Variance score: 1.00
Mean squared error: 0.00
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch     Fare
0            1         0       3    1  22.0      1      0   7.2500
1            2         1       1    0  38.0      1      0  71.2833
2            3         1       3    0  26.0      0      0   7.9250
3            4         1       1    0  35.0      1      0  53.1000
4            5         0       3    1  35.0      0      0   8.0500
Variance score: 0.40
Mean squared error: 0.14
```

```
   PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch     Fare
0            1         0       3    1  22.0      1      0   7.2500
1            2         1       1    0  38.0      1      0  71.2833
2            3         1       3    0  26.0      0      0   7.9250
3            4         1       1    0  35.0      1      0  53.1000
4            5         0       3    1  35.0      0      0   8.0500
Variance score: 0.19
Mean squared error: 0.19
```

**Observations on Task-4** In Task-4 we did multiple regression on diabetes dataset as well as train dataset .AS diabetes data does not contain any null values so we get same RMSE and R2 score before and after EDA,where as coming to train data we obtained Variance as 0.40 and R2 score as O.19 before EDA and after EDA these are changed to 0.14 and 0.14 which are shown in above snippets and explained in the video.

## References:


https://www.google.com/search?q=stackoverflow+python

https://www.google.com/search?q=kaggle+stack+overflow&oq=kagg&aqs=chrome.2.69i57j0l5.3004j0j7&sourceid=chrome&ie=UTF-8