VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

BIG DATA ANALYTICS & BI (CO4033)

Mini-project

Multi-class Text Classification with PySpark

Advisor: Nguyễn Thanh Bình
Class: CC01
Students: Nguyễn Minh Tâm - 1952968
Nguyễn Thanh Ngân - 1911667
Lê Đức Cầm - 1952588

HO CHI MINH CITY, DECEMBER 2022

Contents

1 Import the dataset & set up PySpark

1.1 Creating SparkContext

In this project, we will build a multi-class text classification model. The model can predict the subject category given a course title or text. We will use the Udemy dataset in building our model.

```
# Import a Spark function from library
import pyspark
from pyspark.sql.functions import col
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import *

sc = SparkContext(master="local[2]")
sc
```

SparkContext

Spark UI

Version
v3.3.1
Master
local[2]
AppName
pyspark-shell

Figure 1: Creating SparkContext

SparkContext creates an entry point of our application and creates a connection between the different clusters in our machine allowing communication between them. **SparkContext** will also give a user interface that will show us all the jobs running. The master option specifies the master URL for our distributed cluster which will run locally. We also specify the number of threads to 2. This allows our program to run 2 threads concurrently. It reduces the failure of our program.

1.2 Creating SparkSession

```
[10] spark = SparkSession.builder.appName("Muti-class text classification").getOrCreate()
     spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.3.1
Master
local[2]
AppName
pyspark-shell

Figure 2: Creating SparkSession

By creating **SparkSession**, it enables us to interact with the different Spark functionalities. The functionalities include data analysis and creating our text classification model. A **SparkSession** creates our DataFrame, registers DataFrame as tables, execute SQL over tables, cache tables, and read files.

1.3 Loading the dataset

```
[14] start = datetime.datetime.now()
     df = spark.read.csv("udemy_courses.csv",header=True,inferSchema=True)
     print('total time that need to load the dataset: ', datetime.datetime.now() - start, 's')
     df.show(5)

     total time that need to load the dataset:  0:00:12.637903 s
     +---------+--------------------+--------------------+-------+-----+----------------+-----------+-----------+------------------+----------------+--------------------+
     |course_id|        course_title|                 url|is_paid|price|num_subscribers|num_reviews|num_lectures|             level|content_duration|published_timestamp|
     +---------+--------------------+--------------------+-------+-----+----------------+-----------+-----------+------------------+----------------+--------------------+
     |  1070968|Ultimate Investme...|https://www.udemy...|   true|  200|            2147|         23|         51|       All Levels|             1.5|2017-01-18 20:58:58|
     |  1113822|Complete GST Cour...|https://www.udemy...|   true|   75|            2792|        923|        274|       All Levels|            39.0|2017-03-09 16:34:20|
     |  1006314|Financial Modelin...|https://www.udemy...|   true|   45|            2174|         74|         51|Intermediate Level|             2.5|2016-12-19 19:26:30|
     |  1210588|Beginner to Pro -...|https://www.udemy...|   true|   95|            2451|         11|         36|       All Levels|             3.0|2017-05-30 20:07:24|
     |  1011058|How To Maximize Y...|https://www.udemy...|   true|  200|            1276|         45|         26|Intermediate Level|             2.0|2016-12-13 14:57:18|
     +---------+--------------------+--------------------+-------+-----+----------------+-----------+-----------+------------------+----------------+--------------------+
     only showing top 5 rows
```

Figure 3: Load the dataset

We use the Udemy dataset that contains all the courses offered by Udemy. The dataset contains the course title and subject they belong.

2 Exploratory Data Analysis

2.1 Basic information about the dataset

Before going to the modelling section, we will need to learn about this dataset.

```
[ ] print('Size of dataframe (row, col):  ', (df.count(),  len(df.columns)))

    Size of dataframe (row, col):   (3683, 12)

[ ] df.printSchema()

    root
     |-- course_id: integer (nullable = true)
     |-- course_title: string (nullable = true)
     |-- url: string (nullable = true)
     |-- is_paid: boolean (nullable = true)
     |-- price: integer (nullable = true)
     |-- num_subscribers: integer (nullable = true)
     |-- num_reviews: integer (nullable = true)
     |-- num_lectures: integer (nullable = true)
     |-- level: string (nullable = true)
     |-- content_duration: string (nullable = true)
     |-- published_timestamp: timestamp (nullable = true)
     |-- subject: string (nullable = true)
```

Figure 4: Size of dataframe & data type of each column

2.2 Check NULL value in each column

After that, we check each column if there exist any NULL values.

```
null_list = []
for i in df.columns:
  print('=============== ', i, ' =============== ')
  null_count = df.toPandas()[i].isnull().sum()
  null_percent = (null_count/df.count())*100
  null_list.append(null_percent)
  print('number of null record in "',i,'" column:',null_count)
  print('percentage of null record  in "',i,'" column:',null_percent,'%','\n\n')
```

Figure 5: Function to check NULL values in each column

And we find out that only 1 column contains NULL value: **subject** column.

```
===============  price ===============
number of null record in " price " column: 0
percentage of null record  in " price " column: 0.0 %


===============  num_subscribers ===============
number of null record in " num_subscribers " column: 0
percentage of null record  in " num_subscribers " column: 0.0 %


===============  num_reviews ===============
number of null record in " num_reviews " column: 0
percentage of null record  in " num_reviews " column: 0.0 %


===============  num_lectures ===============
number of null record in " num_lectures " column: 0
percentage of null record  in " num_lectures " column: 0.0 %


===============  level ===============
number of null record in " level " column: 0
percentage of null record  in " level " column: 0.0 %


===============  content_duration ===============
number of null record in " content_duration " column: 0
percentage of null record  in " content_duration " column: 0.0 %


===============  published_timestamp ===============
number of null record in " published_timestamp " column: 0
percentage of null record  in " published_timestamp " column: 0.0 %


===============  subject ===============
number of null record in " subject " column: 5
percentage of null record  in " subject " column: 0.13575889220743959 %
```

Figure 6: Only **subject** column contains NULL values

As you can see, the percentage of NULL values in this column is small so we decide to remove rows containing those NULL values.

2.3 Quick data visualization

After gaining some basic information about the dataset and checking for NULL values, now we perform some visualizations to detect if there is anything worth investigating more.

```
[ ] ### subject distribution
    sns.set(rc={'figure.figsize':(10,5)})
    tar = df2['subject'].value_counts()
    ax = sns.barplot(x=tar.index, y=tar.values).set(title='Number of courses belong to each subject')
```

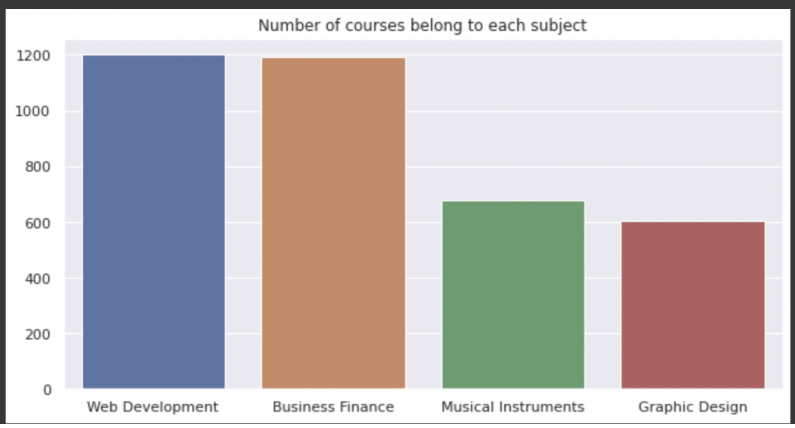


Figure 7: Number of courses belong to each subject

```
[ ] tar = df2['subject'].value_counts()/df2.shape[0]*100
    palette_color = sns.color_palette('bright')
    plt.pie(tar.values, labels=tar.index, colors=palette_color, autopct='%.2f%%')
    plt.title('Percentage of courses related to each subject')
    plt.show()
```

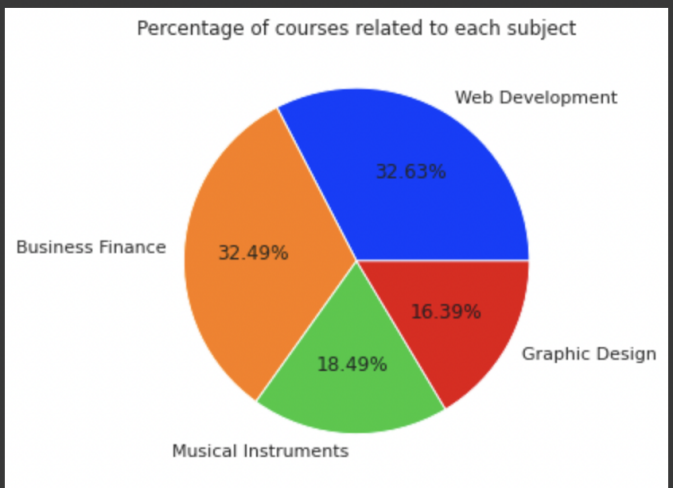


Figure 8: Percentages of courses related to each subject

```
[ ] tmp = df2['published_timestamp'].apply(lambda x: x.year).value_counts()
    plt.scatter(tmp.index, tmp.values, c=np.array(["red","green","blue","brown","purple","black","orange"]))
    # plt.plot(tmp.index, tmp.values,zorder=10)
    plt.xlabel("Year")
    plt.ylabel("Total courses")
    plt.title("Trend in Number of courses per year from 2011-2017")
    plt.show()  # show first chart
```

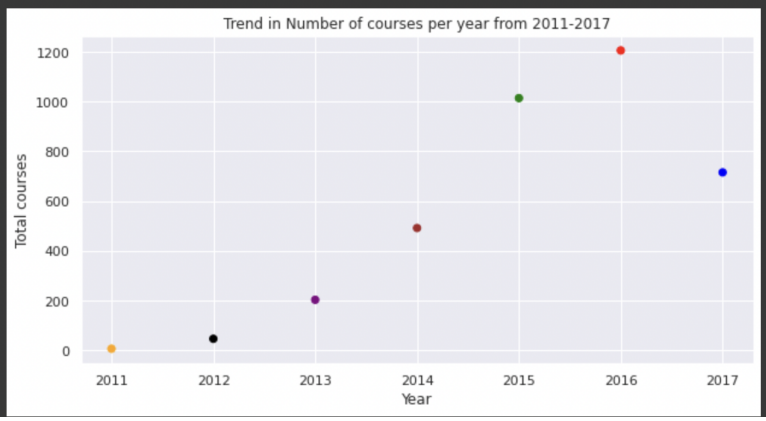


Figure 9: Trend in number of courses from 2011-2017

2.4 Examine numerical features

2.4.1 Visualizing distribution of data

We will examine numerical features by first plotting the distribution of numerical columns.

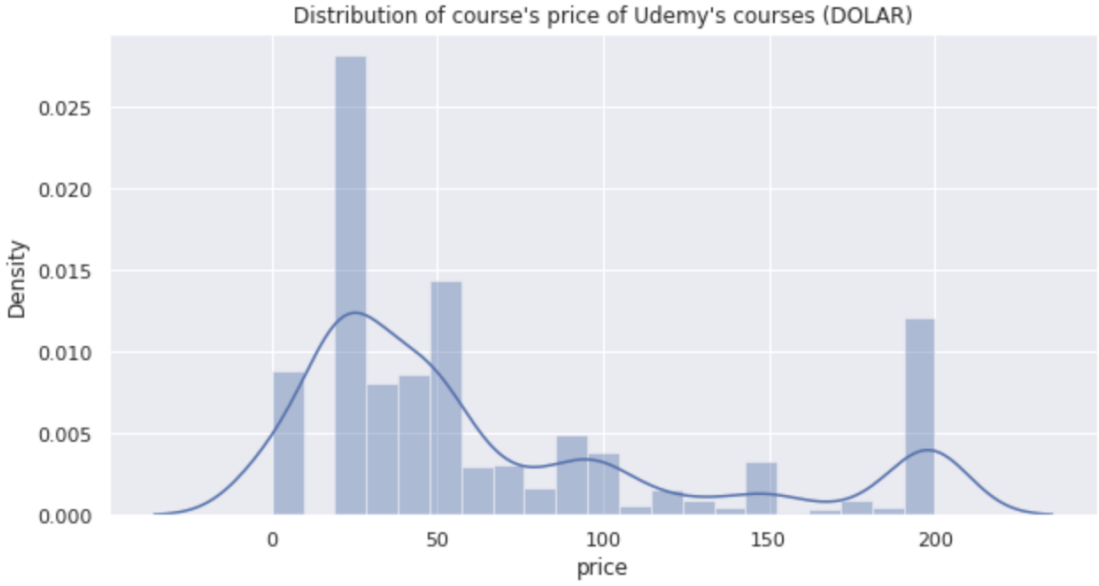
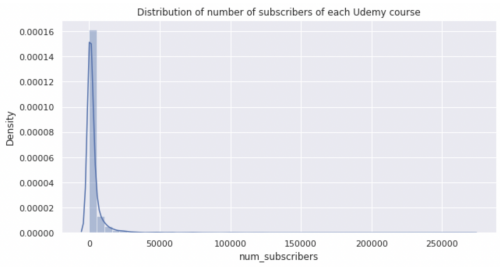
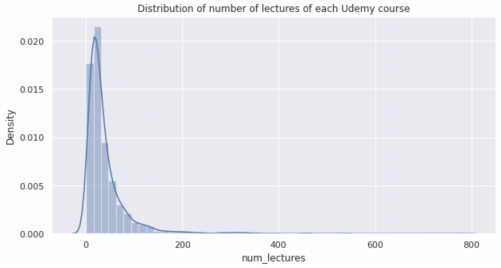


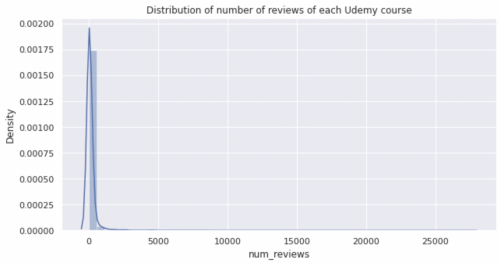
Figure 10: Distribution of courses' price (Dollar)



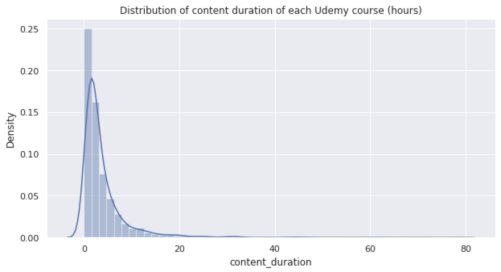
(a) Distribution of the number of subscribers in each course



(b) Distribution of the number of lectures in each course



(c) Distribution of the number of reviews in each course



(d) Distribution of the content duration in each course (Hours)

2.4.2 Box plot visualization

We continue to examine the numerical columns by using box plot for visualization to demonstrate the locality, spread and skewness groups of numerical data through their quartiles.

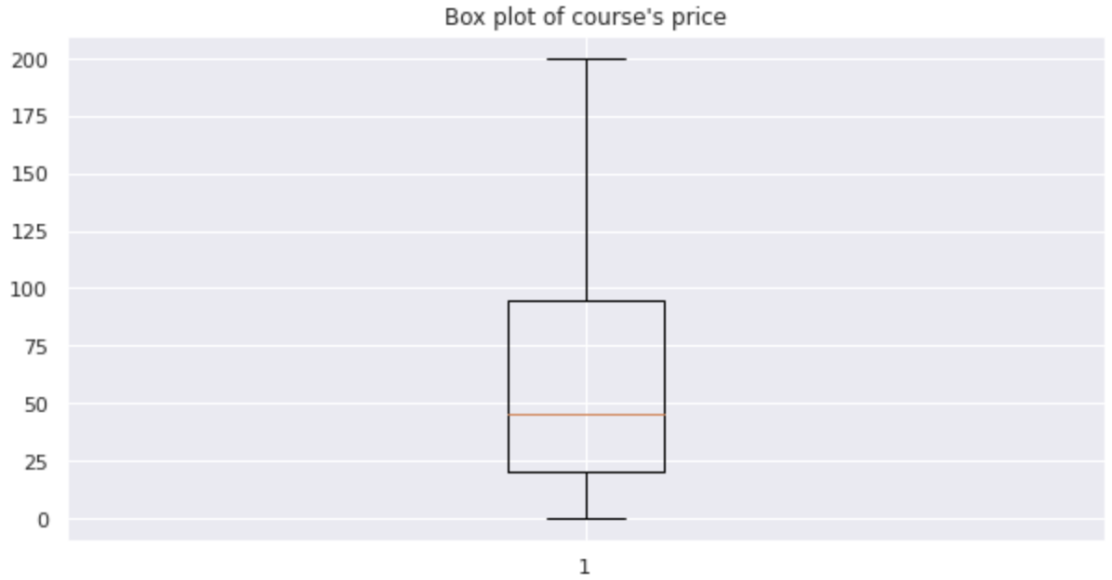
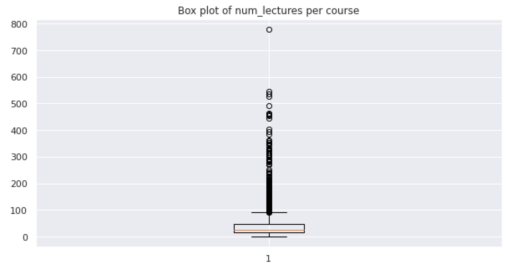
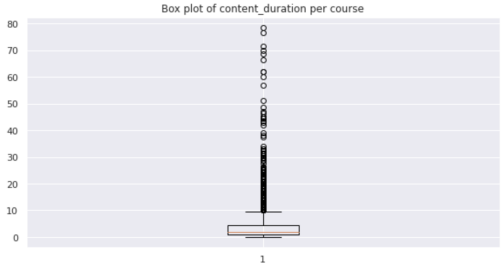


Figure 12: Box plot of courses' price

(a) Box plot of the number of subscribers in each course



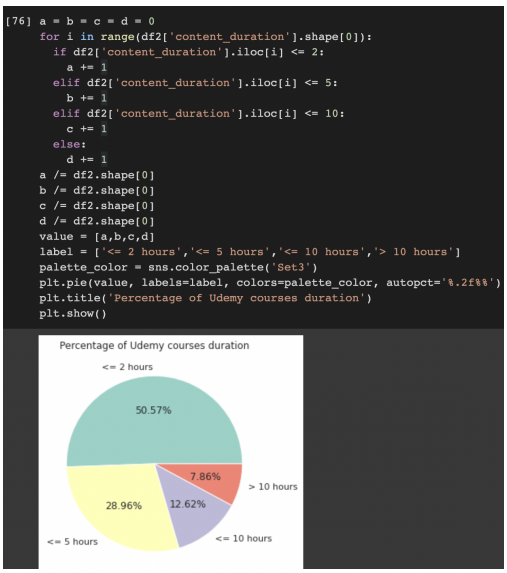
(b) Box plot of the number of lectures in each course



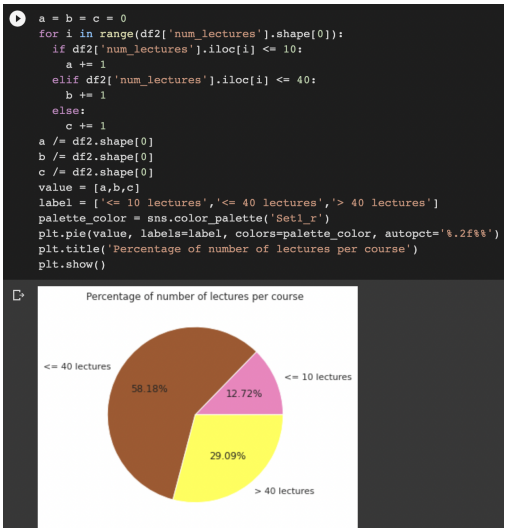
(c) Box plot of content duration in each course

2.4.3 Pie chart visualization

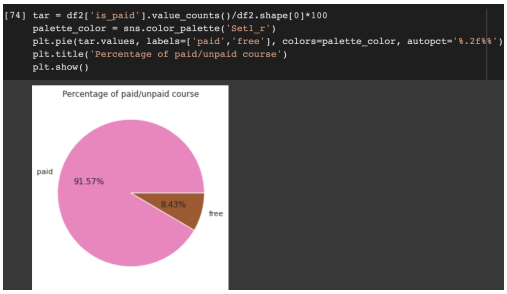
We also use pie chart to illustrate numerical proportions.



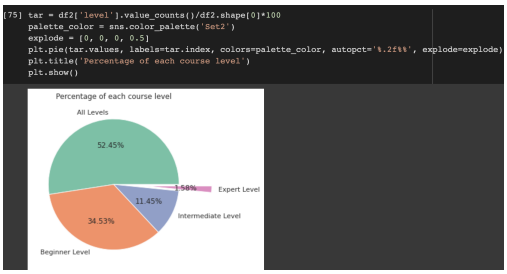
(a) Percentage of each course duration



(b) Percentage of number of lectures in each course



(a) Percentage of paid/unpaid course



(b) Percentage of each course level

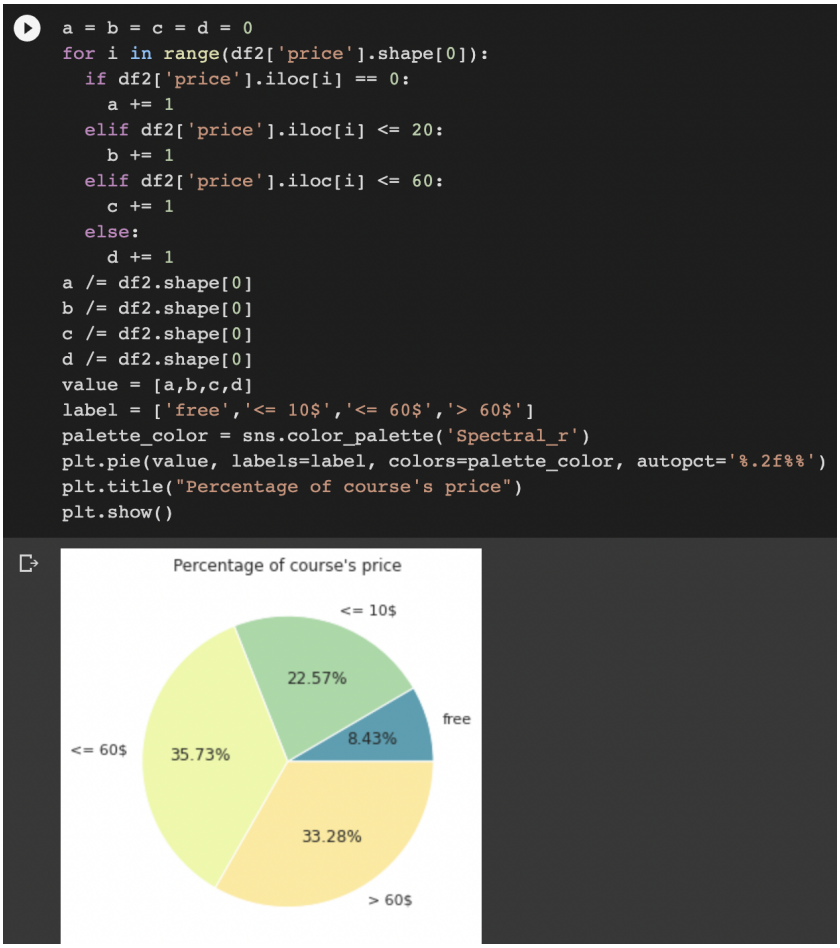


Figure 16: Percentage of course's price

```
[79] tmp = df2['published_timestamp'].apply(lambda x: x.year).value_counts()
     tar = tmp/df2.shape[0]*100
     label = list(tar.index[:-2])
     label = [str(i) for i in label]
     label.append('2011-2013')
     value = list(tar.values[:-2])
     value.append(tar.values[-1]+tar.values[-2])

     palette_color = sns.color_palette('Set3_r')
     plt.pie(value, labels=label, colors=palette_color, autopct='%.2f%%', explode=[0, 0, 0, 0, 0, 0.4])
     plt.title('Percentage of courses per year (2011-2017)')
     plt.show()
```

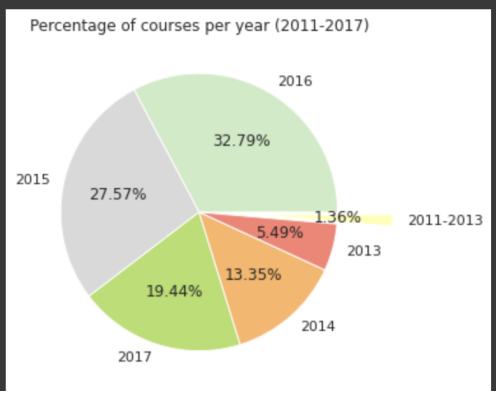


Figure 17: Percentage of courses per year (2011-2017)

2.4.4 Histogram

We try to use the histogram to illustrate the major features of the distribution of the data in a convenient form.

```
df2['word_count'] = df2['course_title'].apply(lambda x: len(str(x).split()))
sns.histplot(data=df2, x="word_count", kde=True,hue="subject").set(title="Course title word count distribution")
```

```
[Text(0.5, 1.0, 'Course title word count distribution')]
```

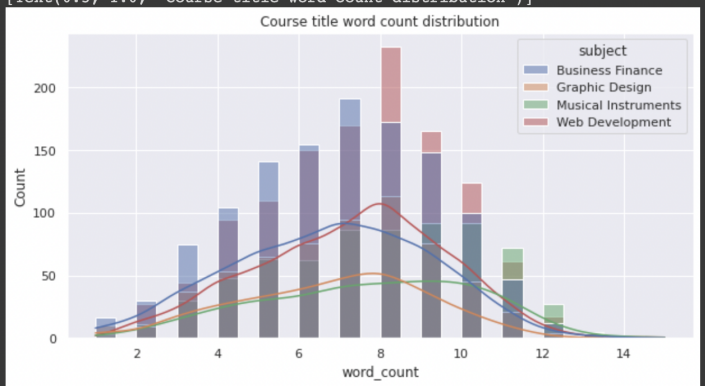


Figure 18: Course title word count distribution

2.4.5 Explore text data in course_title column

Because our target is to classify a course into a suitable subject based on its title so we will investigate more in the **course_title** column.

```
from sklearn.feature_extraction.text import CountVectorizer
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

common_words = get_top_n_words(df2['course_title'], 20)

tmp = pd.DataFrame(common_words, columns = ['comon_word', 'count'])
plt.figure(figsize=(21,7))
ax = sns.barplot(data=tmp, x="comon_word", y="count").set(title="Top 20 most common words in course title")
```

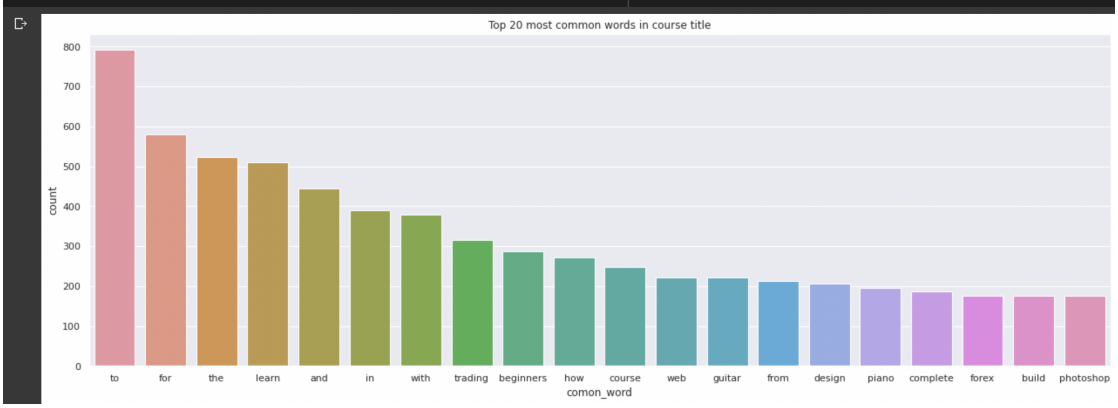


Figure 19: Top 20 most common words in **course_title** column

```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_bigram(df2['course_title'], 15)

tmp = pd.DataFrame(common_words, columns = ['comon_word', 'count'])
plt.figure(figsize=(30,7))
ax = sns.barplot(data=tmp, x="comon_word", y="count").set(title="Top 15 most common bigrams in course title")
```

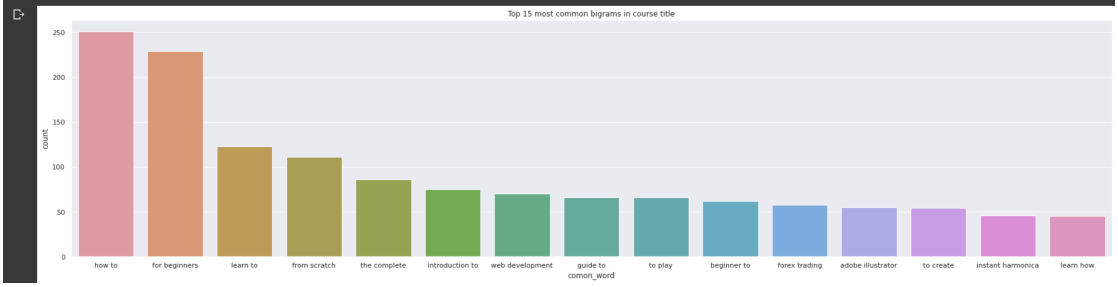
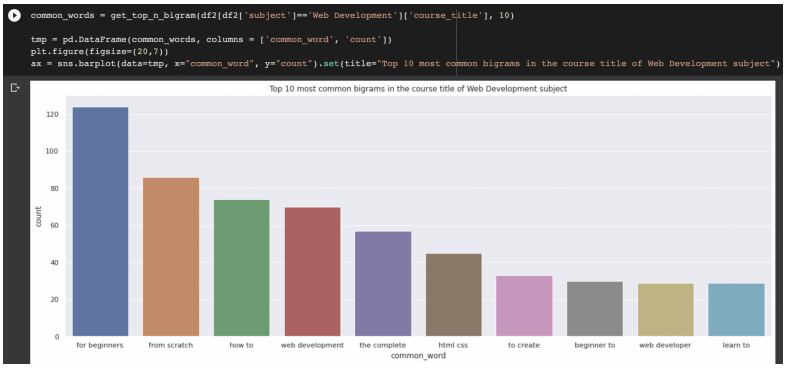
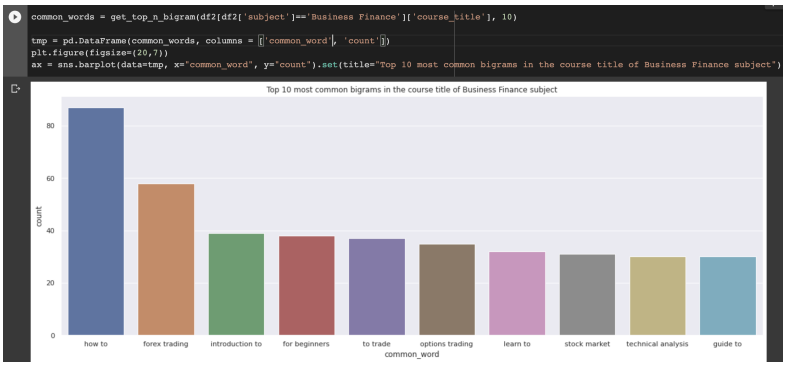


Figure 20: Top 15 most common bigrams in **course_title** column

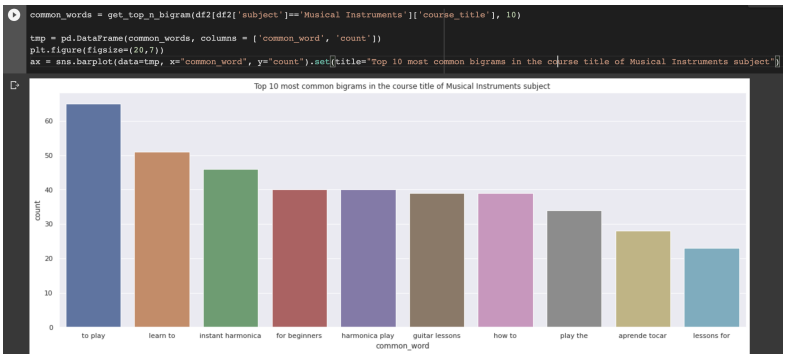
We dive into the investigation in each specific course.



(a) Top 10 most common bigrams in Web Development subject



(b) Top 10 most common bigrams in Business Finance subject



(c) Top 10 most common bigrams in Musical Instruments subject

(d) Top 10 most common bigrams in Graphic Design subject

3 Feature engineering

3.1 Feature selection

As our target is automatically classifying a course to a subject using its title, at the first step of the feature engineering procedure, we select the **course_title** and **subject** columns which will be used to build our model. Let's see the way we do it in Spark and the relative output (shown in Figure 22)

```
[ ]   1 df = df.select('course_title','subject')
      2 df.show(5)

+--------------------+----------------+
|        course_title|         subject|
+--------------------+----------------+
|Ultimate Investme...|Business Finance|
|Complete GST Cour...|Business Finance|
|Financial Modelin...|Business Finance|
|Beginner to Pro -...|Business Finance|
|How To Maximize Y...|Business Finance|
+--------------------+----------------+
only showing top 5 rows
```

Figure 22: Feature selection

3.2 Processing NULL value

After choosing suitable columns, we check for any missing values in our dataset. This ensures that we have a well-formatted dataset that can be used to build our model. According to the figure 23, we can observe that there are 5 records (0.13575% of total record) not containing any value.

```
[ ]   1 null_count = df.toPandas()['subject'].isnull().sum()
      2 null_percent = (null_count/df.count())*100
      3 print('number of null record in subject column:',null_count)
      4 print('percentage of null record  in subject column:',null_percent,'%')

number of null record in subject column: 5
percentage of null record  in subject column: 0.13575889220743959 %

      1 null_count = df.toPandas()['course_title'].isnull().sum()
      2 null_percent = (null_count/df.count())*100
      3 print('number of null record in course_title column:',null_count)
      4 print('percentage of null record  in course_title column:',null_percent,'%')

number of null record in course_title column: 0
percentage of null record  in course_title column: 0.0 %
```

Figure 23: Number of record containing null values and corresponding percentage

As the number of missing values is relatively small compared with the total record, that's why we decided to drop those invalid rows.

```
# too small percentage -> drop
df = df.dropna(subset=('subject'))
```

Figure 24: Drop NULL values

3.3 Encoding labels

As our label is the subject that courses belong to, it is in text format, that's why we must change it to a number which can fit to ML algorithms. According to the dictionary of labels, we can see the subject name with the corresponding number. All of that stuff can be done via **StringIndexer** function of PySpark MLlib, see the figure below:

```
labelEncoder = StringIndexer(inputCol='subject',outputCol='label').fit(df)
```

```
label_dict = {'Web Development':0.0,
 'Business Finance':1.0,
 'Musical Instruments':2.0,
 'Graphic Design':3.0}
```

```
df = labelEncoder.transform(df)
df.show(5)
```

```
+--------------------+----------------+-----+
|        course_title|         subject|label|
+--------------------+----------------+-----+
|Ultimate Investme...|Business Finance|  1.0|
|Complete GST Cour...|Business Finance|  1.0|
|Financial Modelin...|Business Finance|  1.0|
|Beginner to Pro -...|Business Finance|  1.0|
|How To Maximize Y...|Business Finance|  1.0|
+--------------------+----------------+-----+
only showing top 5 rows
```

Figure 25: Encoding label

3.4 Feature engineering pipeline

As feature engineering is a process of getting the relevant features and characteristics from raw data so we extract various characteristics from our Udemy dataset that will act as inputs into our classification algorithm. The features will be used in making predictions.

Machine learning algorithms do not understand texts so we have to convert them into numeric values during this stage. Thus, we import all the packages required for feature engineering from PySpark MLlib and list all the available methods to have a glance at it using the following code lines:

```
import pyspark.ml.feature
dir(pyspark.ml.feature)
```

To transform data from raw text to numeric values, we do the following steps with some MLlib features:

1. **Tokenizer**: It converts the input text and converts it into word tokens. These word tokens are short phrases that act as inputs into our model.
2. **StopWordsRemover**: It extracts all the stop words available in our dataset. Stop words are a set of words that are used in a given sentence frequently. These words may be biased when building the classifier.
3. **CountVectorizer**: It converts from text to vectors of numbers. Numbers are understood by the machine more easily rather than text.
4. **Inverse Document Frequency (IDF)**: It's a statistical measure that indicates how important a word is relative to other documents in a collection of documents. This creates a relation between different words in a document. If a word appears frequently in a given document and also appears frequently in other documents, it shows that it has little predictive power towards classification. The more the word is rare in given documents, the more it has value in predictive analysis.

This is a sequential process starting from the **Tokenizer** stage to the **IDF** stage as shown below:

```
1 from pyspark.ml.feature import Tokenizer,StopWordsRemover,CountVectorizer,IDF
2 from pyspark.ml.feature import StringIndexer

[ ] 1 tokenizer = Tokenizer(inputCol='course_title',outputCol='mytokens')
    2 stopwords_remover = StopWordsRemover(inputCol='mytokens',outputCol='filtered_tokens')
    3 vectorizer = CountVectorizer(inputCol='filtered_tokens',outputCol='rawFeatures')
    4 idf = IDF(inputCol='rawFeatures',outputCol='vectorizedFeatures')
```

Figure 26: Transform raw text to numeric values

4 Modeling

4.1 Splitting into training and testing set

As we want to examine the strength of each ML algorithm with different settings for this text-classification problem so we split our data into train/test set with 2 different ratios:

- 75% for training and 25% for testing
- 50% for training and 50% for testing

Python code:

```
(trainDF, testDF) = df.randomSplit((0.75, 0.25), seed=27)
(trainDF2, testDF2) = df.randomSplit((0.5, 0.5), seed=72)
```

4.2 Choosing algorithms

There are various kinds of ML algorithms and many of them are provided by Spark, but for multi-label problems, just some of which are implemented. To extensively do the experiment, we use 5 ML algorithms for our training and testing process including Logistic Regression, Decision Tree, Naive Bayes, Multi-layer Perceptron (3 layers) and Random Forest (10 trees). Let's see how to define classifier for our pipeline with figure 27:

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.classification import RandomForestClassifier


nb = NaiveBayes(smoothing=1.0, modelType="multinomial",featuresCol='vectorizedFeatures',labelCol='label')
lr = LogisticRegression(featuresCol='vectorizedFeatures',labelCol='label')
dt = DecisionTreeClassifier(featuresCol='vectorizedFeatures',labelCol='label',maxDepth=30)
mlp = MultilayerPerceptronClassifier(maxIter=50, layers=[3765,15,4], blockSize=128, seed=1234, featuresCol='vectorizedFeatures',labelCol='label',)
rfc = RandomForestClassifier(featuresCol='vectorizedFeatures',labelCol='label', numTrees=10,maxDepth=30)
classifiers = [nb,lr,dt,mlp,rfc] #
```

Figure 27: Classifiers used in our training/testing pipeline

4.3 Training process

As we have defined feature engineering functions, ml algorithms and also have well-formatted data so now it is time for training!

For each ML algorithm (classifiers[i]), we create a pipeline consisting of

$$tokenizer, stopwords\_remover, vectorizer, IDF, classifiers[i]$$

then trained it and measure how long it finishes. See the figure below:

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
pipeline = []
model = []
for i in range(len(classifiers)):
  start = datetime.datetime.now()
  pipeline.append(Pipeline(stages=[tokenizer,stopwords_remover,vectorizer,idf,classifiers[i]]))
  model.append(pipeline[i].fit(trainDF))
  end = datetime.datetime.now()
  if i == 0:
    print('total time need to train Naive Bayes: ', end - start, ' (s)\n')
  elif i == 1:
    print('total time need to train Logistic Regression: ', end - start, ' (s)\n')
  elif i == 2:
    print('total time need to train Decision Tree: ', end - start, ' (s)\n')
  elif i==3:
    print('total time need to train Multilayer (3 layers) Perceptron: ', end - start, ' (s)\n')
  else:
    print('total time need to train Random Forest (72 trees): ', end - start, ' (s)\n')
```

Figure 28: 5 stages training pipeline

4.4 Making predictions

After training to measure the test result, we should have predictions first. Overall it will look like this:

```
predictions = []
for i in range(len(model)):
  if i == 0:
    print('======= Naive Bayes =========')
  elif i == 1:
    print('======= Logistic Regression =========')
  elif i == 2:
    print('======= Decision Tree =========')
  elif i==3:
    print('======= Multilayer (3 layers) Perceptron ========')
  else:
    print('======= Random Forest (72 trees) =========')
  predictions.append(model[i].transform(testDF))
  predictions[i].show(3)
  print('\n')
```

```
======= Naive Bayes =========
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|        course_title|             subject|label|            mytokens|     filtered_tokens|         rawFeatures| vectorizedFeatures|       rawPrediction|         probability|prediction|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|#1 Piano Hand Coo...|Musical Instruments|  2.0|[#1, piano, hand,...|[#1, piano, hand,...|(3765,[10,14,190,...|(3765,[10,14,190,...|[-493.18545479425...|[1.11975930901261...|       2.0|
|#10 Hand Coordina...|Musical Instruments|  2.0|[#10, hand, coord...|[#10, hand, coord...|(3765,[0,4,277,33...|(3765,[0,4,277,33...|[-489.55198479289...|[9.58331470409147...|       2.0|
|#7 Piano Hand Coo...|Musical Instruments|  2.0|[#7, piano, hand,...|[#7, piano, hand,...|(3765,[10,14,63,3...|(3765,[10,14,63,3...|[-453.03747931629...|[1.58661358548346...|       2.0|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
only showing top 3 rows


======= Logistic Regression =========
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|        course_title|             subject|label|            mytokens|     filtered_tokens|         rawFeatures| vectorizedFeatures|       rawPrediction|         probability|prediction|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|#1 Piano Hand Coo...|Musical Instruments|  2.0|[#1, piano, hand,...|[#1, piano, hand,...|(3765,[10,14,190,...|(3765,[10,14,190,...|[-12.043548148172...|[4.54047946158202...|       2.0|
|#10 Hand Coordina...|Musical Instruments|  2.0|[#10, hand, coord...|[#10, hand, coord...|(3765,[0,4,277,33...|(3765,[0,4,277,33...|[-10.759314606227...|[1.22043000989810...|       2.0|
|#7 Piano Hand Coo...|Musical Instruments|  2.0|[#7, piano, hand,...|[#7, piano, hand,...|(3765,[10,14,63,3...|(3765,[10,14,63,3...|[-11.675844215689...|[2.02609958653588...|       2.0|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
only showing top 3 rows


======= Decision Tree =========
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|        course_title|             subject|label|            mytokens|     filtered_tokens|         rawFeatures| vectorizedFeatures|       rawPrediction|         probability|prediction|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
|#1 Piano Hand Coo...|Musical Instruments|  2.0|[#1, piano, hand,...|[#1, piano, hand,...|(3765,[10,14,190,...|(3765,[10,14,190,...| [0.0,0.0,107.0,0.0]|   [0.0,0.0,1.0,0.0]|       2.0|
|#10 Hand Coordina...|Musical Instruments|  2.0|[#10, hand, coord...|[#10, hand, coord...|(3765,[0,4,277,33...|(3765,[0,4,277,33...|[225.0,461.0,132....|[0.23267838676318...|       1.0|
|#7 Piano Hand Coo...|Musical Instruments|  2.0|[#7, piano, hand,...|[#7, piano, hand,...|(3765,[10,14,63,3...|(3765,[10,14,63,3...| [0.0,0.0,107.0,0.0]|   [0.0,0.0,1.0,0.0]|       2.0|
+--------------------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+
```

Figure 29: Predictions of each classifier for the test set

4.5 Evaluation

To evaluate the performance of ML algorithms, there are many metrics that have been developed. To have the fairest experiment we use accuracy, precision, recall and f1. For PySpark, it provides **MulticlassClassificationEvaluator**, which uses the label, column and prediction columns to calculate all metrics that we need. You can see how did we do it.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
accuracy = []
f1 = []
precision = []
recall = []
for i in range(len(model)):
  if i == 0:
    print('======= Naive Bayes =========')
  elif i == 1:
    print('======= Logistic Regression =========')
  elif i == 2:
    print('======= Decision Tree =========')
  elif i == 3:
    print('======= Multilayer (3 layers) Perceptron ========')
  else:
    print('======= Random Forest (72 trees) =========')
  evaluator = MulticlassClassificationEvaluator(labelCol='label',predictionCol='prediction',metricName='accuracy')
  evaluator2 = MulticlassClassificationEvaluator(labelCol='label',predictionCol='prediction',metricName='precisionByLabel
  evaluator3 = MulticlassClassificationEvaluator(labelCol='label',predictionCol='prediction',metricName='recallByLabel')
  evaluator4 = MulticlassClassificationEvaluator(labelCol='label',predictionCol='prediction',metricName='f1')
  accuracy.append(evaluator.evaluate(predictions[i]))
  precision.append(evaluator2.evaluate(predictions[i]))
  recall.append(evaluator3.evaluate(predictions[i]))
  f1.append(evaluator4.evaluate(predictions[i]))
  print('testing accuracy: ',accuracy[i])
  print('testing precision: ',precision[i])
  print('testing recall: ',recall[i])
  print('testing f1-score: ',f1[i])
  print('\n')

```

Figure 30: Evaluation using MulticlassClassificationEvaluator function

4.6 Result

We conduct experiments for both settings with many algorithms to have a clear view of which one is a good choice for this topic.

4.6.1 General Result

Notation:

- Logistic Regression: LR
- Decision Tree: DT
- Naive Bayes: NB
- Multi-layer Perceptron: MLP
- Random Forest: RF

Table 1: Result of our experiment with 75% for training, 25% for testing.

	NB	LR	DT	MLP	RF
Accuracy (%)	92.99	92.57	80.13	**93.28**	86.00
Precision (%)	91.03	96.48	**96.88**	95.90	91.29
Recall (%)	**96.27**	92.88	73.89	94.92	88.81
F1-score (%)	92.98	92.56	80.52	**93.77**	85.92
Training time (s)	**5.17**	7.06	11.10	10.97	7.59

Table 2: Result of our experiment with 50% for training, 50% for testing.

	NB	LR	DT	MLP	RF
Accuracy (%)	92.95	91.92	79.35	**94.36**	82.43
Precision (%)	89.46	**95.16**	89.31	94.54	89.44
Recall (%)	96.76	93.71	75.34	**97.11**	86.40
F1-score (%)	92.95	91.87	79.51	**94.31**	82.04
Training time (s)	**1.33**	3.37	5.41	5.18	5.21

$\longrightarrow$ After all, Naive Bayes is the algorithm which takes the least time to be fully trained, while for general performance, Multi-layer Perceptron has seen amazing results in both of 5 metrics. No doubt, MLP is one of the strongest algorithms for machine learning right now.

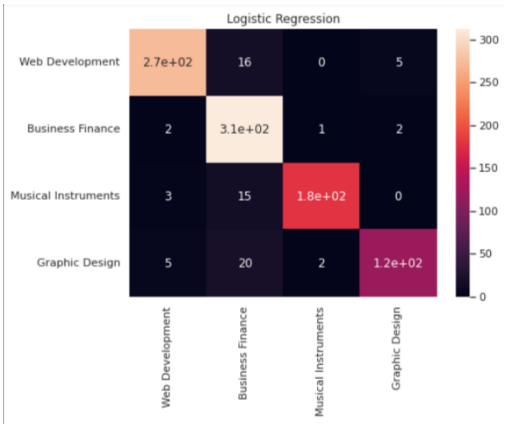
4.6.2 Confusion matrices

To have an alternative view about each classifier, we use Sklearn library to draw the confusion matrix to check for a specific model, in which case it is usually misclassified.

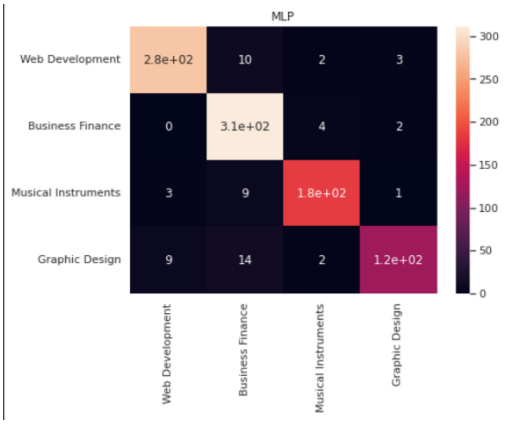
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

for i in range(len(predictions)):
  y_pred=predictions[i].select("prediction").collect()
  y_orig=predictions[i].select("label").collect()
  cm = confusion_matrix(y_orig, y_pred)
  df_cm = pd.DataFrame(cm, index = [i for i in ['Web Development','Business Finance', 'Musical Instruments', 'Graphic Design']],
                  columns = [i for i in ['Web Development','Business Finance', 'Musical Instruments', 'Graphic Design']])
  plt.figure(figsize = (7,5))
  if i == 0:
    sns.heatmap(df_cm, annot=True).set(title='Naive Bayes')
  elif i == 1:
    sns.heatmap(df_cm, annot=True).set(title='Logistic Regression')
  elif i == 2:
    sns.heatmap(df_cm, annot=True).set(title='Decision Tree')
  elif i == 3:
    sns.heatmap(df_cm, annot=True).set(title='MLP')
  else:
    sns.heatmap(df_cm, annot=True).set(title='Random Forest')
  # print(plot_confusion_matrix(cm),'\n\n')
```

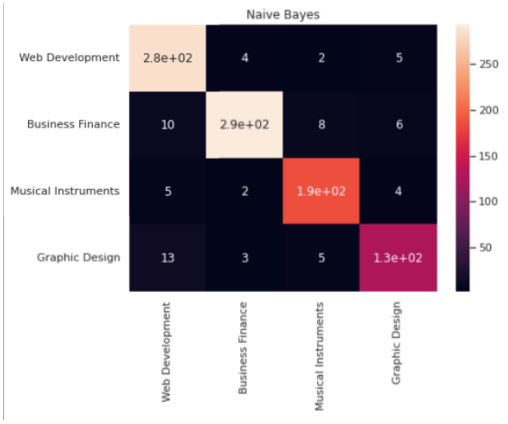
Figure 31: Code for drawing confusion matrix



(a) Confusion matrix of Logistic Regression Model

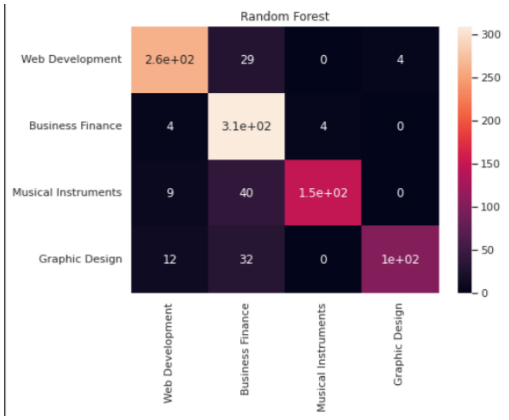


(b) Confusion matrix of Multi-layer Perceptron Model



(c) Confusion matrix of Naive Bayes Model

(d) Confusion matrix of Decision Tree Model



(e) Confusion matrix of Random Forest Model

5 Custom test

We use our trained model to make custom predictions. We input a text into our model and see if our model can classify the right subject. Custom predictions expose our model to a new set of data that is not available in the training set or the testing set. This makes sure that our model makes new predictions on its own under a new environment. To perform a single prediction, we prepare our sample input as a string (as shown in the figure below).

```
[ ]  1 from pyspark.sql.types import StringType
     2
     3 ex = spark.createDataFrame([
     4     ("Building Machine Learning Apps with Python and PySpark",StringType()),
     5     ("Principle of Programming language",StringType()),
     6     ("Microservices Application development",StringType()),
     7     ("Introduction to Banking system",StringType()),
     8     ("International Monetery system",StringType()),
     9     ("Piano or Violin ?",StringType()),
    10     ("Logo Design Fundamentals",StringType()),
    11 ], ["course_title"] )
    12 ex.show(truncate=False)
```

```
+-----------------------------------------------------+---+
|course_title                                         |_2 |
+-----------------------------------------------------+---+
|Building Machine Learning Apps with Python and PySpark|{} |
|Principle of Programming language                    |{} |
|Microservices Application development                |{} |
|Introduction to Banking system                       |{} |
|International Monetery system                       |{} |
|Piano or Violin ?                                    |{} |
|Logo Design Fundamentals                             |{} |
+-----------------------------------------------------+---+
```

Figure 33: Create a custom test with StringType function in Spark

The table below is the prediction of each trained model where

- 0: Web Development
- 1: Business Finance
- 2: Musical Instruments
- 3: Graphic Design

Table 3: Result of our custom test

	NB	LR	DT	MLP	RF
Building Machine Learning Apps with Python and PySpark	0	0	0	0	0
Principle of Programming language	0	0	1	0	0
Microservices Application development	0	0	1	0	0
Introduction to Banking system	1	1	1	1	1
International Monetary system	1	1	1	1	1
Piano or Violin ?	2	2	2	2	2
Logo Design Fundamentals	3	3	3	3	3

6 Conclusion

In this mini-project, we solved the task of multi-class text classification with PySpark. We started with PySpark basics and used some core components of PySpark used for Big Data processing, then started preparing our dataset by removing missing values. We used the Udemy dataset (from Kaggle) to build our model.

We then followed the stages in the machine learning workflow. We started with feature engineering and then applied the pipeline approach to automate certain workflows. Pipeline makes the process of building a machine learning model easier. After following all the pipeline stages, we ended up with 5 machine learning models.

Finally, we used these 5 models to make predictions, this is the goal of any machine learning model. The more accurate a model predicts, the better a model is.