Natural Language Processing

Lecture-2

Extracting Text Data

Analytics

- Analytics is all about using data to make decisions
- Data science is under the umbrella of analytics

Data Science

Data science in particular consists of three main skills

- Programming (knowing how to code)
- Math and Stats (linear algebra, calculus, statistics)
- Communication (communicate your insights)

Steps in Data Science Project

1. Start with a question
2. Get and clean the data
3. Perform (Exploratory Data Analysis)EDA
4. Apply Techniques
5. Share Insights

Start With a Question

- If I study more, will I get higher grades

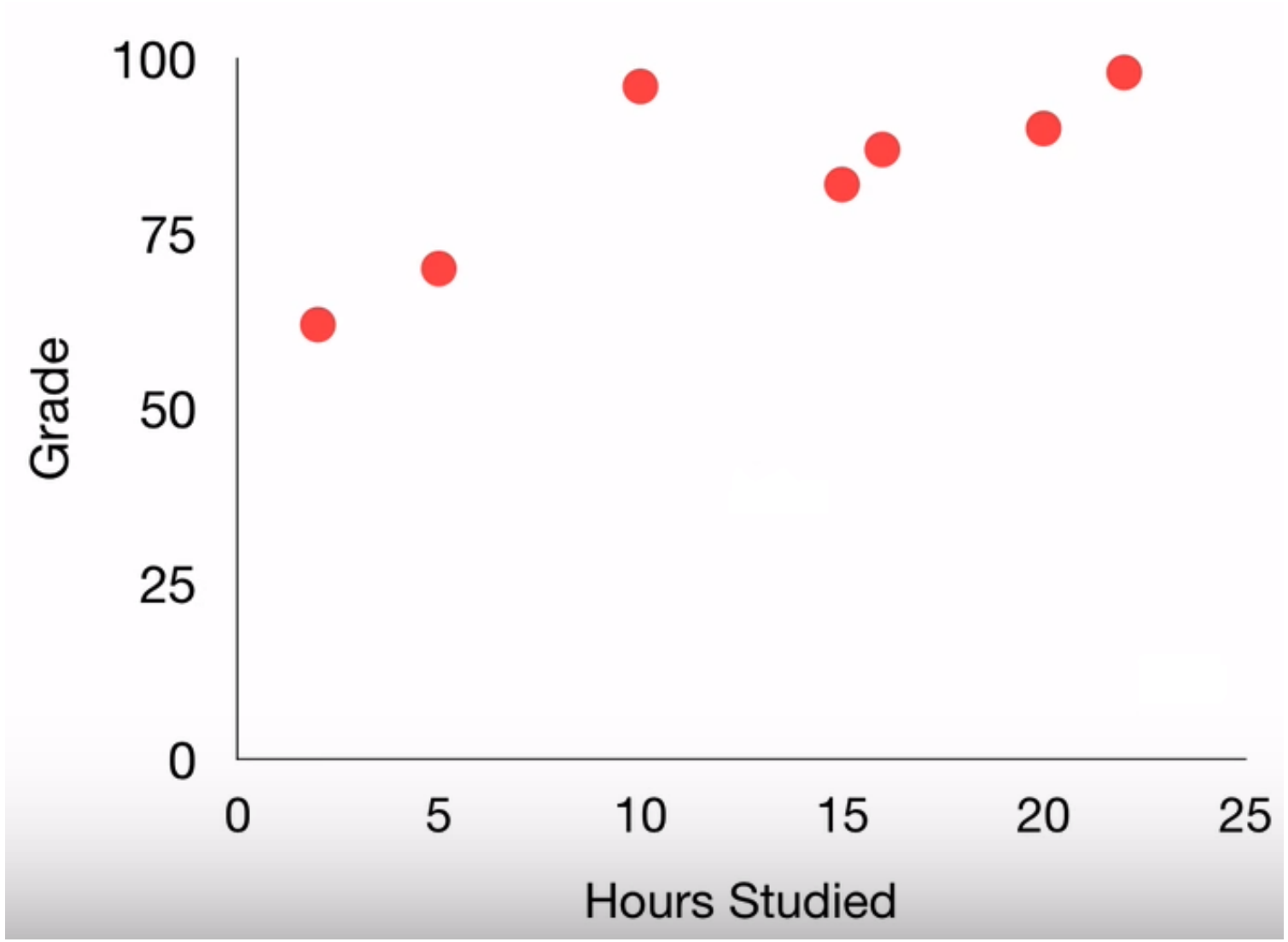
Get and clean the data

Student	Hours Studied	Grade
Alice	20	90
Bob	5	70
Charlie	10	96
David	15	82
Eve	two	62
Frank	16	87
Grace	22	998

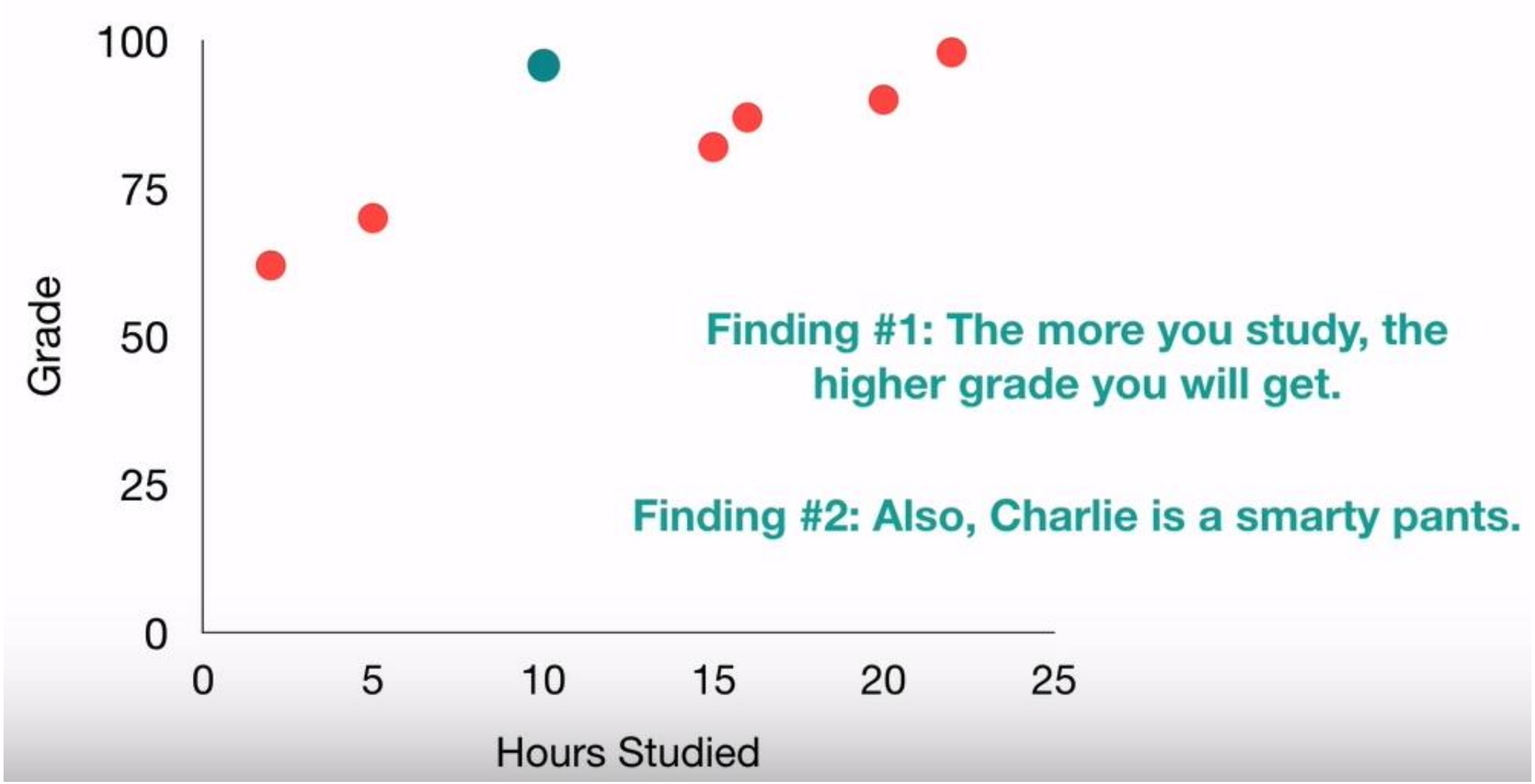
Get and clean the data

Student	Hours Studied	Grade
Alice	20	90
Bob	5	70
Charlie	10	96
David	15	82
Eve	2	62
Frank	16	87
Grace	22	98

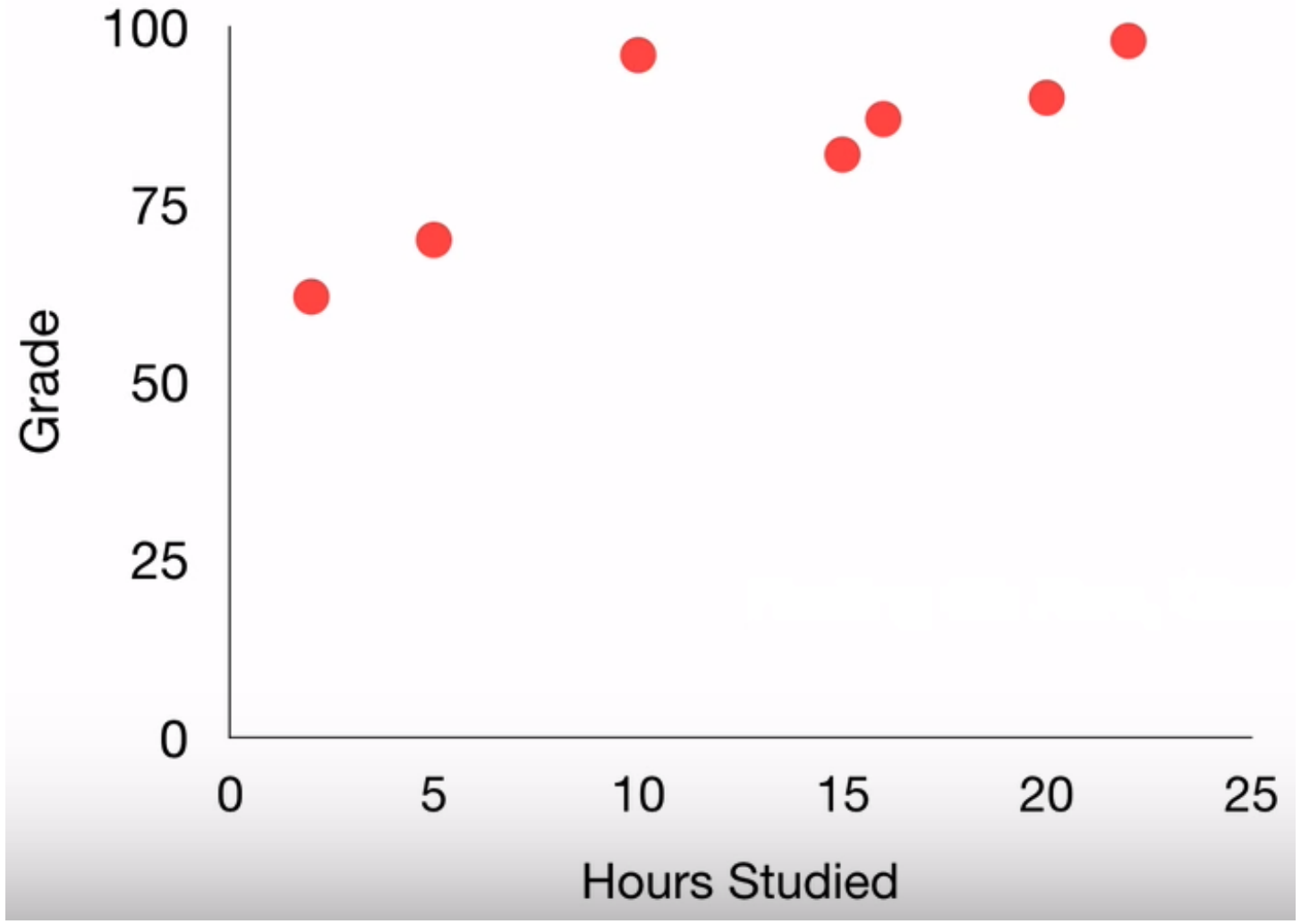
Perform EDA



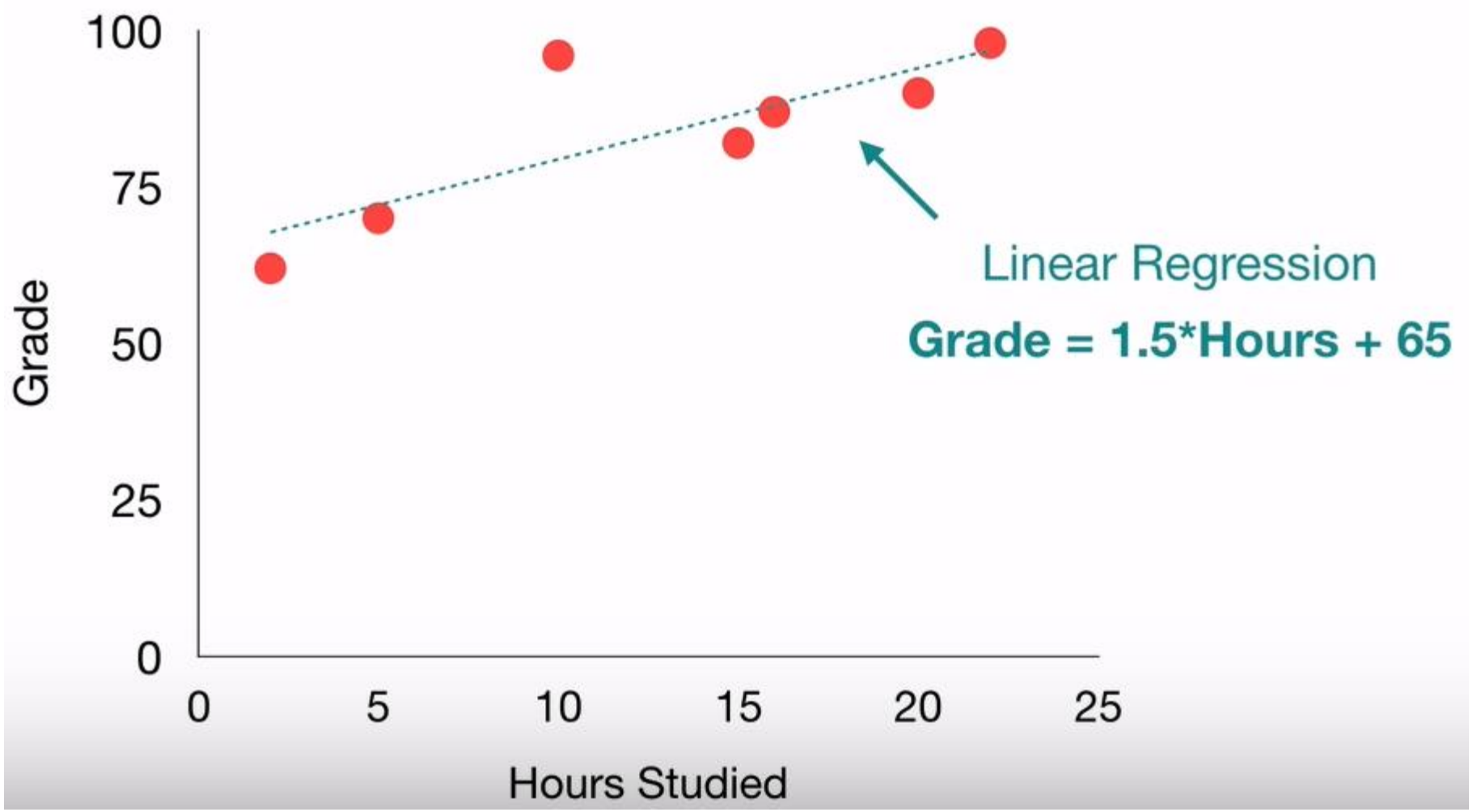
Perform EDA


100
75
50
25
0
Grade
0
5
10
15
20
25
Hours Studied
Finding #1: The more you study, the higher grade you will get.
Finding #2: Also, Charlie is a smarty pants.

Apply Techniques



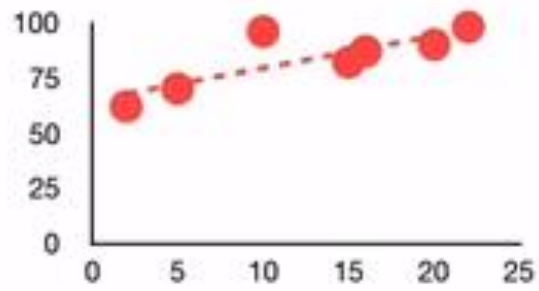
Apply Techniques



Share Insights

"If I study more, will I get a higher grade?

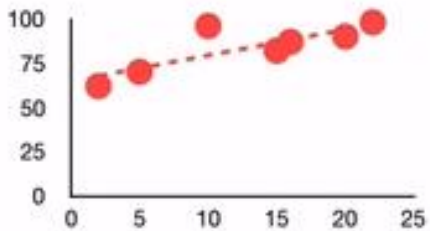
Yes, there is a **positive correlation** between the number of hours you study and the grade you will get.



Share Insights

"If I study more, will I get a higher grade?

Yes, there is a **positive correlation** between the number of hours you study and the grade you will get.

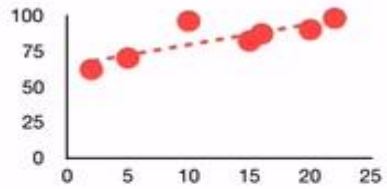


Specifically, the relationship is: **Grade = 1.5 * Hours + 65**
So if you study **10 hours**, you can expect to get an **80**.

Share Insights

"If I study more, will I get a higher grade?

Yes, there is a **positive correlation** between the number of hours you study and the grade you will get.



Specifically, the relationship is: **Grade = 1.5 * Hours + 65**
So if you study **10 hours**, you can expect to get an **80**.

However, Charlie is a smarty pants and is inflating the grade estimate. You'll probably get **slightly less than 80**.

Steps for NLP Project

1. Start with a question
2. Get and clean the text data
3. Perform EDA
4. Apply Techniques
5. Share Insights

Overall steps are same for natural language processing, except step two. Here, instead of using numerical data for data analysis like we're typically used to. Now we are using text data for the analysis

Data Gathering

We start to think that

- Is there already any corpus related to our problem which could be used?

Data Gathering

Let suppose we don’t find any corpus. Then we should start to think

From where we should get the data for this problem?

- what type of data I should analyze for this problem?
- from where we should get the data for this problem?
- how much data should we get?
- what should be the time range

How we can code Data Gathering?

Data Gathering

Depending upon the problem, we may need to gather data from different sources e.g.

- API
- PDFs files
- Word files
- JSON files
- Scraping Web

Data Gathering and Extracting Sources

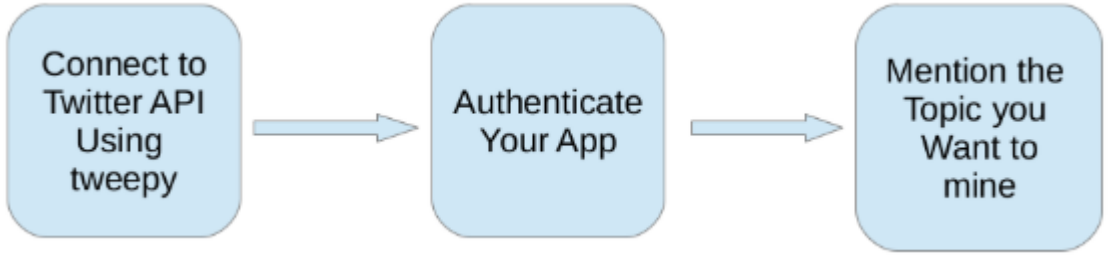
Various sources and ways to extract text data

- *Text data collection using APIs*
- *Reading PDF file in Python*
- *Reading word document*
- *Reading JSON object*
- *Reading HTML page and HTML parsing*
- *Regular expressions*
- *String handling*
- *Web scraping*

Problem 1 (*Text data collection using APIs*)

You want to collect text data using Twitter APIs.

- Twitter has a large amount of data with a lot of value in it. Social media marketers are making their living from it
- There is an enormous amount of tweets every day
- Every tweet has some story to tell
- When all of this data is collected and analyzed, it gives a tremendous amount of insights to a business about their company, product and service



Problem 1 (*Text data collection using APIs*)

How It Works

- Log in to the Twitter developer portal
- Create your own app in the Twitter developer portal, and get the keys
- Consumer key: Key associated with the application (Twitter, Facebook, etc.).
- Consumer secret: Password used to authenticate with the authentication server (Twitter, Facebook, etc.)
- Access token: Key given to the client after successful authentication
- Access token secret: Password for the access key

Code for Twitter(*Text data collection using APIs*)

```
# Install tweepy
#!pip install tweepy
# Import the libraries
import numpy as np
import tweepy
import pandas as pd
from tweepy import OAuthHandler
# credentials
consumer_key = "adjbiejfaaoeh"
consumer_secret = "had73haf78af"
access_token = "jnsfby5u4yuawhafjeh"
access_token_secret = "jhdfgay768476r"
# calling API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
# Provide the query you want to pull the data. For example, pulling data for the mobile phone ABC
query ="ABC"
# Fetching tweets
Tweets = api.search(query, count = 10,lang='en', exclude='retweets',tweet_mode='extended')
```

Gathering Data From PDFs

- Most of the time your data stored as PDF files. We need to extract text from these files and store it for further analysis.

```
import PyPDF2
from PyPDF2 import PdfFileReader

#Creating a pdf file object
pdf = open("mypdf.pdf","rb")
#creating pdf reader object
pdf_reader = PyPDF2.PdfFileReader(pdf)
#checking number of pages in a pdf file
print(pdf_reader.numPages)
#creating a page object
page = pdf_reader.getPage(1)
#finally extracting text from the page
print(page.extractText())
#closing the pdf file
pdf.close()
```

Gathering Data From Word Docx

- You want to read Word files by using the docx library

```
#Import library
# pip install python-docx
#pip install docx
from docx import Document

document = Document('ReadWord.docx')
for para in document.paragraphs:
    print(para.text +"\n")
'''# create an empty string and call this document. This document  variable store each paragraph in the Word document. We then create a for loop that goes through each paragraph in the Word document and appends the paragraph.'''
print("\n\n")
docu=""
for para in document.paragraphs:
      docu += para.text

#to see the output call docu
print(docu)
```

Gathering Data From Word Docx

- You want to read Word files by using different libraries

Docx2txt : allows you to scrape text and images from Word Documents
Docx2python :
python-docx :
docx:

Gathering Data From CSV

- You want to read CSV files by using csv library

```
import csv
with open('SalesRecord.csv', 'r') as file:
    reader = csv.reader(file, delimiter = '\t', skipinitialspace=True)
    for row in reader:
        print(row)
```

Collecting Data from HTML

let us look at reading HTML pages OR read parse/read HTML pages

By using the **bs4** library

```
#pip install bs4
import urllib.request as urllib2
from bs4 import BeautifulSoup

#Fetch the HTML file
response = urllib2.urlopen('https://en.wikipedia.org/wiki/Natural_language_processing')
html_doc = response.read()
#Parse the HTML file
soup = BeautifulSoup(html_doc, 'html.parser')
# Formating the parsed html file
strhtm = soup.prettify()
# Print few lines
print (strhtm[:1000])
```

Parsing Text Using Regular Expressions

- A **regular expression** is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

- Regular expressions are helpful when dealing with text data.
- very much required when dealing with raw data from the web, which would contain HTML tags, long text, and repeated text
- We can do all sort of basic and advanced data cleaning using regular expressions

Parsing Text Using Regular Expressions

- by using the “**re**” library in Python
- we can use regular expressions for our tasks. Basic flags: the basic flags are I, L, M, S, U, X
- re.I: This flag is used for ignoring casing.
- re.L: This flag is used to find a local dependent.
- re.M: This flag is useful if you want to find patterns throughout multiple lines.
- re.S: This flag is used to find dot matches.
- re.U: This flag is used to work for unicode data.
- re.X: This flag is used for writing regex in a more readable format.

Regular Expressions functionality

- Find the single occurrence of character a and b: Regex: [ab]
- Find characters except for a and b: Regex: [^ab]
- Find the character range of a to z: Regex: [a-z]
- Find a range except to z: Regex: [^a-z]
- Find all the characters a to z as well as A to Z: Regex: [a-zA-Z]
- Any single character: Regex:
- Any whitespace character: Regex: \s
- Any non-whitespace character: Regex: \S
- Any digit: Regex: \d
- Any non-digit: Regex: \D

Regular Expressions functionality

- Any non-words: Regex: \W
- Any words: Regex: \w
- Either match a or b: Regex: (a|b)
- Matches zero or one occurrence but not more than one occurrence

  Regex: a? ; ?
- The occurrence of a is zero times or more than that: Regex: a* ; * matches zero or more than that
- The occurrence of a is one time or more than that: Regex: a+ ; + matches occurrences one or more that one time

Regular Expressions functionality

- Exactly match three occurrences of a: Regex: a{3}
- Match simultaneous occurrences of a with 3 or more than 3: Regex: a{3,}
- Match simultaneous occurrences of a between 3 to 6: Regex: a{3,6}
- Starting of the string: Regex: ^
- Ending of the string: Regex: $
- Match word boundary: Regex: \b
- Non-word boundary: Regex: \B

Regular Expressions functionality

- Exactly match three occurrences of a: Regex: a{3}
- Match simultaneous occurrences of a with 3 or more than 3: Regex: a{3,}
- Match simultaneous occurrences of a between 3 to 6: Regex: a{3,6}
- Starting of the string: Regex: ^
- Ending of the string: Regex: $
- Match word boundary: Regex: \b
- Non-word boundary: Regex: \B

Regular Expressions functions

- re.match() and re.search() functions are used to find the patterns

- re.match(): This checks for a match of the string only at the beginning of the string. So, if it finds the pattern at the beginning of the input string, then it returns the matched pattern; otherwise; it returns a noun.
- re.search(): This checks for a match of the string anywhere in the string. It finds all the occurrences of the pattern in the given input string or data

Tokenizing using Regular Expressions Function

- split the sentence into words – tokenize by using re.split.

Example for Tokenize

```
# Import library
import re
#run the split query
re.split('\s+','I like this book.')
['I', 'like', 'this', 'book.']
```

Example using Regular Expressions Function

- split the sentence into words – tokenize by using re.split.

Example for Tokenize

```
# Import library
import re
#run the split query
re.split('\s+','I like this book.')
['I', 'like', 'this', 'book.']
```

Example using Regular Expressions Function

- by using re.findall

Example Extracing email IDs

```
# Import library
import re
Read/create the document or sentences
doc = "For more details please mail us at: xyz@abc.com, pqr@mno.com"
#Execute the re.findall function
addresses = re.findall(r'[\w\.-]+@[\w\.-]+', doc)
for address in addresses:
        print(address)
```

Example using Regular Expressions Function

- by using re.sub

Example Replacing email IDs

```
# Import library
import re
Read/create the document or sentences
doc = "For more details please mail us at xyz@abc.com"
#Execute the re.sub function
new_email_address = re.sub(r'([\w\.-]+)@([\w\.-]+)', r'pqr@mno.com', doc)
        print(new_email_address)
```

Handling Strings

string functionality.

- s.find(t) index of first instance of string t inside s (-1 if not found)
- s.rfind(t) index of last instance of string t inside s (-1 if not found)
- s.index(t) like s.find(t) except it raises ValueError if not found
- s.rindex(t) like s.rfind(t) except it raises ValueError if not found
- s.join(text) combine the words of the text into a string using s as the glue
- s.split(t) split s into a list wherever a t is found (whitespace by default)
- s.splitlines() split s into a list of strings, one per line
- s.lower() a lowercased version of the string s
- s.upper() an uppercased version of the string s
- s.title() a titlecased version of the string s
- s.strip() a copy of s without leading or trailing whitespace
- s.replace(t, u) replace instances of t with u inside s

Next Lecture: **Exploring and Processing Text Data**

1. Lowercasing
2. Punctuation removal
3. Stop words removal
4. Text standardization
5. Spelling correction
6. Tokenization
7. Stemming
8. Lemmatization
9. Exploratory data analysis
10. End-to-end processing pipeline