foodbasket

June 19, 2021

1 Web Scraping

[1]:
```
from dotenv import load_dotenv
load_dotenv("dev.env")
import os

db_name = os.getenv("db_name")
db_username = os.getenv("db_username")
db_password = os.getenv("db_password")
db_host = os.getenv("db_host")
db_port = os.getenv("db_port")
website = os.getenv("website")
city_link = os.getenv("city_link")
chrome_path = os.getenv("chrome_path")
selenium_chrome_driver_path = os.getenv("selenium_chrome_driver_path")
```

[2]:
```
import psycopg2
import time
import math
import numpy as np
import os.path
import pandas as pd
import re
import pycld2
import nltk
from datetime import date, timedelta, datetime
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException
from nltk.corpus import stopwords
from TurkishStemmer import TurkishStemmer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score
from sklearn.model_selection import GridSearchCV

nltk.download('stopwords', quiet=True)

stemmer = TurkishStemmer()

conn_string = 'host={pghost} port={pgport} dbname={pgdatabase} user={pguser}␣
 ↪password={pgpassword}'.
 ↪format(pgdatabase=db_name,pguser=db_username,pgpassword=db_password,pghost=db_host,pgport=d
conn=psycopg2.connect(conn_string)
cur=conn.cursor()

options = Options()
options.binary_location = chrome_path

def check_if_table_exists(schema,table):
    cur.execute("select exists(select * from information_schema.tables where␣
 ↪table_schema='{schema}' AND table_name='{table}')".format(schema=schema,␣
 ↪table=table))
    return cur.fetchone()[0]

def check_if_index_exists(index):
    cur.execute("SELECT EXISTS(SELECT * FROM PG_CLASS WHERE relname =␣
 ↪'{index}')".format(index=index))
    return cur.fetchone()[0]

def check_if_file_exists(filename):
    return os.path.isfile(filename)

def execute_mogrify(conn, df, schema, table):
    tuples = [tuple(x) for x in df.to_numpy()]
    cols = '"'+'","'.join(list(df.columns))+'"'
    cursor = conn.cursor()
    try:
        for tup in tuples:
            query  = """INSERT INTO "{schema}"."{table}"({cols}) VALUES␣
 ↪({values}) ON CONFLICT DO NOTHING""".format(schema=schema,table=table,␣
 ↪cols=cols, values=",".join(map(str,tup)))
            cursor.execute(query)
            conn.commit()
    except (Exception, psycopg2.DatabaseError) as error:
        print("Error: %s" % error)
        conn.rollback()
```

```
        cursor.close()
        return 1
    cursor.close()

def df_column_conversation(df, column_name, type):
    if(type == 'timestamp'):
        df[column_name] = df[column_name].apply(lambda x: f"'{x}'::timestamp")
    if(type == 'text'):
        df[column_name] = df[column_name].str.replace("'","").apply(lambda x:␣
 ↪f"'{x}'")
    if(type == 'date'):
        df[column_name] = df[column_name].apply(lambda x: f"'{x}'::date")
    if(type == 'numeric'):
        df[column_name] = df[column_name].apply(str).str.replace(',','.')
    if(type == 'integer'):
        df[column_name] = df[column_name].apply(str).str.replace(',','.').
 ↪apply(float).astype('Int64').apply(str)

def clean_text(text):
    text = text.lower()
    text = re.sub(r"[,.\"\'!@#$%^&*(){}?/;`~:<>+=-\\]", "", text)
    return text
```

[3]:
```
if(check_if_table_exists('ODS','EXT_FB_RESTAURANT')):
    print('Table ODS.EXT_FB_RESTAURANT already exists.')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "ODS"."EXT_FB_RESTAURANT"
    (
    "RESTAURANT_ID" text NOT NULL,
    "RESTAURANT_NAME" text,
    "RESTAURANT_LINK" text,
    "DATE" date,
    CONSTRAINT "RESTAURANT_ID" UNIQUE ("RESTAURANT_ID")
    );
    """)
    cur.execute('COMMIT;')
    end_time = math.trunc(time.time())
    print("Table ODS.EXT_FB_RESTAURANT created in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))

if(check_if_table_exists('ODS','EXT_FB_MENU')):
    print('Table ODS.EXT_FB_MENU already exists.')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
```

```
    CREATE TABLE "ODS"."EXT_FB_MENU"
    (
    "PRODUCT_ID" text NOT NULL,
    "RESTAURANT_ID" text,
    "CATEGORY_NAME" text,
    "PRODUCT_NAME" text,
    "PRODUCT_DESCRIPTION" text,
    "PRODUCT_LISTED_PRICE" text,
    "PRODUCT_PRICE" text,
    "DISCOUNT" boolean,
    "DESIGN_TYPE" text,
    "DATE" date,
    CONSTRAINT "PRODUCT_ID" UNIQUE ("PRODUCT_ID")
    );
    """)
    cur.execute('COMMIT;')
    end_time = math.trunc(time.time())
    print("Table ODS.EXT_FB_MENU created in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))

if(check_if_table_exists('ODS','EXT_FB_COMMENT')):
    print('Table ODS.EXT_FB_COMMENT already exists.')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "ODS"."EXT_FB_COMMENT"
    (
    "RESTAURANT_ID" text,
    "USERNAME" text,
    "COMMENT_TEXT" text,
    "COMMENT_DATE" text,
    "SPEED" text,
    "SERVING" text,
    "FLAVOUR" text,
    "DATE" date,
    CONSTRAINT "UNIQUE_COMMENTS" UNIQUE ("RESTAURANT_ID", "USERNAME",␣
 ↪"COMMENT_TEXT")
    );
    """)
    cur.execute('COMMIT;')
    end_time = math.trunc(time.time())
    print("Table ODS.EXT_FB_COMMENT created in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))

if(check_if_table_exists('EDW','DWH_FB_COMMENT')):
    print('Table EDW.DWH_FB_COMMENT already exists.')
else:
```

```
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_FB_COMMENT"
    (
    "RESTAURANT_ID" text,
    "USERNAME" text,
    "COMMENT_TEXT" text,
    "COMMENT_DATE" date,
    "SPEED" integer,
    "SERVING" integer,
    "FLAVOUR" integer,
    "DATE" date,
    CONSTRAINT "UNIQUE_COMMENTS" UNIQUE ("RESTAURANT_ID", "USERNAME",␣
 ↪"COMMENT_TEXT")
    );
    """)
    cur.execute('COMMIT;')
    end_time = math.trunc(time.time())
    print("Table EDW.DWH_FB_COMMENT created in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))

cur.execute("""
CREATE OR REPLACE FUNCTION public.try_cast(_in text, INOUT _out anyelement)
    LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   EXECUTE format('SELECT %L::%s', $1, pg_typeof(_out))
   INTO  _out;
EXCEPTION WHEN others THEN
   -- do nothing: _out already carries default
END
$BODY$;
""")
cur.execute('COMMIT;')
```

```
Table ODS.EXT_FB_RESTAURANT already exists.
Table ODS.EXT_FB_MENU already exists.
Table ODS.EXT_FB_COMMENT already exists.
Table EDW.DWH_FB_COMMENT already exists.
```

[4]:
```
cur.execute("""
WITH DATES AS(
SELECT
MAX("DATE") AS "DATE"
FROM "ODS"."EXT_FB_MENU" EFM
UNION ALL
SELECT
```

```
MAX("DATE") AS "DATE"
FROM "ODS"."EXT_FB_COMMENT" EFC
)
SELECT
MAX("DATE") AS "LAST_EXECUTION_DATE"
FROM DATES;
""")
last_execution_date = cur.fetchone()[0]
last_execution_date
```

[4]:
```
datetime.date(2021, 6, 11)
```

[5]:
```
restaurant_list = []
end_date = min(date(2021,6,11),(date.today() - timedelta(days=1)))

driver = webdriver.Chrome(options=options,␣
 ↪executable_path=selenium_chrome_driver_path)
if(last_execution_date < end_date):
    driver.get(city_link)
    time.sleep(5)
    for i in range(25):
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(2)
    city_restaurant_groups = driver.
 ↪find_elements_by_class_name("restaurant-main-info")
    for restaurant in city_restaurant_groups:
        restaurant_name = restaurant.
 ↪find_element_by_class_name("restaurant-display-name").text
        restaurant_name = restaurant_name.replace("YENİ ", "")
        restaurant_link = restaurant.
 ↪find_element_by_class_name("restaurant-display-name").
 ↪find_element_by_xpath(".//a").get_attribute('href')
        restaurant_id = restaurant_link.split("/")[-1]
        if(len(restaurant_link) < 2):
            continue
        restaurant_list.append([restaurant_id,restaurant_name,restaurant_link])
    restaurant_df = pd.DataFrame(restaurant_list,␣
 ↪columns=["RESTAURANT_ID","RESTAURANT_NAME","RESTAURANT_LINK"])

    df_column_conversation(restaurant_df, 'RESTAURANT_ID', 'text')
    df_column_conversation(restaurant_df, 'RESTAURANT_NAME', 'text')
    df_column_conversation(restaurant_df, 'RESTAURANT_LINK', 'text')
    restaurant_df['DATE'] = "'"+ datetime.strftime(date.today(), "%Y-%m-%d") +␣
 ↪"'::date"
    execute_mogrify(conn,restaurant_df,"ODS","EXT_FB_RESTAURANT")
```

[6]:
```
sql_command = """
    SELECT
    "RESTAURANT_ID"
    FROM "ODS"."EXT_FB_RESTAURANT" EFR
    WHERE 1=1
    AND NOT EXISTS(SELECT NULL FROM "ODS"."EXT_FB_MENU" EFM WHERE EFR.
↪"RESTAURANT_ID" = EFM."RESTAURANT_ID");
    """
restaurant_df = pd.read_sql(sql_command,conn)
```

[7]:
```
if(last_execution_date < end_date):
    for i in range(len(restaurant_df)):
        sublink = restaurant_df.loc[i,"RESTAURANT_ID"]
        restaurant_link = "{website}/{sublink}".
↪format(website=website,sublink=sublink)

        driver.get(restaurant_link)
        time.sleep(5)

        try:
            if("sipariş verebilirsiniz." in driver.find_element_by_xpath('//
↪*[@id="restaurantDetail"]/div/div[2]/h3').text):
                continue
            else:
                pass
        except Exception:
            pass

        menu = driver.find_element_by_xpath('//*[@id="restaurant_menu"]')
        categories = menu.find_elements_by_xpath('//*[contains(@id,"menu_")]')

        menu_list = []

        for category in categories:
            category_name = category.find_element_by_xpath(".//b").text
            for product in category.find_elements_by_xpath(".//div[2]/ul/li"):
                try:
                    design_type = "list"
                    try:
                        product_id = product.
↪find_elements_by_class_name("getProductDetail")[-1].
↪get_attribute('data-product-id')
                        product_name = product.
↪find_elements_by_class_name("getProductDetail")[-1].text
                    except:
                        product_id = product.find_element_by_xpath(".//strong").
↪get_attribute('data-product-id')
```

```
                        product_name = product.find_element_by_xpath(".//
↪strong").text
                        design_type = "card"
                    try:
                        product_description = product.
↪find_element_by_class_name("product-desc").text
                        product_price = product.
↪find_element_by_class_name("price").text
                    except:
                        product_description = product.
↪find_element_by_class_name("productInfo").text
                        product_price = product.
↪find_element_by_class_name("newPrice").text
                        if(not(design_type=="card")):
                            design_type = "box"
                    discount = "TRUE"
                    try:
                        if(design_type=="list"):
                            product_listed_price = product.
↪find_element_by_class_name("listed-price").text
                        if(design_type in ["card","box"]):
                            product_listed_price = product.
↪find_element_by_class_name("listedPrice").text
                    except:
                        product_listed_price = product_price
                        discount = "FALSE"
                    menu_list.
↪append([product_id,sublink,category_name,product_name,product_description,product_listed_pr
                except:
                    continue
        menu_df = pd.DataFrame(menu_list,␣
↪columns=["PRODUCT_ID","RESTAURANT_ID","CATEGORY_NAME","PRODUCT_NAME","PRODUCT_DESCRIPTION",
        menu_df = menu_df[menu_df['PRODUCT_ID'].str.len() > 0]
        menu_df = menu_df[menu_df['PRODUCT_NAME'].str.len() > 0]

        df_column_conversation(menu_df, 'PRODUCT_ID', 'text')
        df_column_conversation(menu_df, 'RESTAURANT_ID', 'text')
        df_column_conversation(menu_df, 'CATEGORY_NAME', 'text')
        df_column_conversation(menu_df, 'PRODUCT_NAME', 'text')
        df_column_conversation(menu_df, 'PRODUCT_DESCRIPTION', 'text')
        df_column_conversation(menu_df, 'PRODUCT_LISTED_PRICE', 'text')
        df_column_conversation(menu_df, 'PRODUCT_PRICE', 'text')
        df_column_conversation(menu_df, 'DESIGN_TYPE', 'text')
        menu_df['DATE'] = "'"+ datetime.strftime(date.today(), "%Y-%m-%d") + "':
↪:date"
        execute_mogrify(conn,menu_df,"ODS","EXT_FB_MENU")
```

```
    menu_df
```

[8]:
```
sql_command = """
    SELECT
    "RESTAURANT_ID"
    FROM "ODS"."EXT_FB_RESTAURANT" EFR
    WHERE 1=1
    AND NOT EXISTS(SELECT NULL FROM "ODS"."EXT_FB_COMMENT" EFC WHERE EFR.
 ↪"RESTAURANT_ID" = EFC."RESTAURANT_ID");
    """
restaurant_df = pd.read_sql(sql_command,conn)
restaurant_df = restaurant_df.sample(frac=1)
restaurant_df
```

[8]:
```
                                        RESTAURANT_ID
41              fitchino-avcilar-ambarli-mah-istanbul
17  dominos-pizza-atasehir-ataturk-mah-girne-cad-i…
56   kurucesme-kahvesi-atasehir-barbaros-mah-istanbul
72           bocek-kadikoy-caferaga-mah-moda-istanbul
14              mondes-besiktas-mecidiye-mah-istanbul
..                                                 …
42  food-hall-istanbul-beyoglu-asmalimescit-mah-is…
61        pizza-hut-umraniye-ihlamurkuyu-mah-istanbul
64                un-po-beyoglu-kemankes-mah-istanbul
43    vera-pizza-pasta-bakirkoy-cevizlik-mah-istanbul
48  kafein-plus-kucukcekmece-tevfikbey-mah-sefakoy…

[79 rows x 1 columns]
```

[9]:
```
if(last_execution_date < end_date):
    for i in range(len(restaurant_df)):
        sublink = restaurant_df.loc[i,"RESTAURANT_ID"]
        last_comment_page_url = "{website}/{sublink}?
 ↪section=comments&page=9999".format(website=website,sublink=sublink)

        driver.get(last_comment_page_url)
        time.sleep(0.1)

        comments_list = []

        if(sublink not in driver.current_url):
            continue

        last_comment_page_redirect_url = driver.current_url
        last_comment_page_number = int(last_comment_page_redirect_url.
 ↪replace("&status=closed","").replace("{website}/{sublink}?
 ↪section=comments&page=".format(website=website,sublink=sublink),""))
```

```
        for page_number in range(1, last_comment_page_number+1):
            current_comment_page_url = "{website}/{sublink}?
↪section=comments&page={page_number}".
↪format(website=website,sublink=sublink,page_number=page_number)
            driver.get(current_comment_page_url)
            time.sleep(1)

            try:
                if("sipariş verebilirsiniz." in driver.find_element_by_xpath('//
↪*[@id="restaurantDetail"]/div/div[2]/h3').text):
                    continue
                else:
                    pass
            except Exception:
                pass

            try:
                driver.find_element(By.XPATH, '//
↪*[@id="alternative-restaurant-popup"]/div[1]/div[2]/img').click(); #Closing␣
↪pop-up
            except Exception:
                pass

            #driver.find_element(By.XPATH, '//*[@id="restaurantDetail"]/div[2]/
↪div[1]/ul/li[4]/a').click(); #Clicking comments

            comment_list = driver.find_elements_by_class_name("comments-body")

            for comment in comment_list:
                try:
                    username = comment.find_element_by_class_name("userName").
↪text
                    comment_text = comment.find_element_by_xpath('.//p').text
                    comment_date = comment.
↪find_element_by_class_name("commentDate").text
                except NoSuchElementException:
                    continue
                try:
                    speed = comment.find_element_by_class_name("speed").text
                except NoSuchElementException:
                    speed = ""
                try:
                    serving = comment.find_element_by_class_name("serving").text
                except NoSuchElementException:
                    serving = ""
```

```
                try:
                    flavour = comment.find_element_by_class_name("flavour").text
                except NoSuchElementException:
                    flavour = ""
                comments_list.append([sublink, username, comment_text,␣
↪comment_date, speed, serving, flavour])
        comment_df = pd.DataFrame(comments_list,␣
↪columns=["RESTAURANT_ID","USERNAME","COMMENT_TEXT","COMMENT_DATE","SPEED","SERVING","FLAVOU
        df_column_conversation(comment_df, 'RESTAURANT_ID', 'text')
        df_column_conversation(comment_df, 'USERNAME', 'text')
        df_column_conversation(comment_df, 'COMMENT_TEXT', 'text')
        df_column_conversation(comment_df, 'COMMENT_DATE', 'text')
        df_column_conversation(comment_df, 'SPEED', 'text')
        df_column_conversation(comment_df, 'SERVING', 'text')
        df_column_conversation(comment_df, 'FLAVOUR', 'text')
        comment_df['DATE'] = "'"+ datetime.strftime(date.today(), "%Y-%m-%d") +␣
↪"'::date"
        execute_mogrify(conn,comment_df,"ODS","EXT_FB_COMMENT")
    comment_df
```

[10]:
```
driver.quit()
```

2 Data Preprocessing

[11]:
```
if(last_execution_date < end_date):
    sql_command = """
        WITH CLEAN_DATA AS(
        SELECT
        EFC."RESTAURANT_ID",
        EFC."USERNAME",
        LOWER(EFC."COMMENT_TEXT") AS "COMMENT_TEXT",
        EFC."COMMENT_DATE",
        TRY_CAST(REGEXP_REPLACE(EFC."SPEED", '\D','','g'),NULL::INTEGER) AS␣
↪"SPEED",
        TRY_CAST(REGEXP_REPLACE(EFC."SERVING", '\D','','g'),NULL::INTEGER) AS␣
↪"SERVING",
        TRY_CAST(REGEXP_REPLACE(EFC."FLAVOUR", '\D','','g'),NULL::INTEGER) AS␣
↪"FLAVOUR",
        EFC."DATE",
        REGEXP_REPLACE(EFC."COMMENT_DATE", '\D','','g')||'␣
↪'||REPLACE(REPLACE(REPLACE(REGEXP_REPLACE(REPLACE(EFC."COMMENT_DATE",'␣
↪önce',''), '[^[:alpha:]]', '',␣
↪'g'),'ay','month'),'bugün','today'),'gün','day') AS "COMMENT_DATE_INTERVAL"
        FROM "ODS"."EXT_FB_COMMENT" EFC
        WHERE 1=1
        AND EFC."USERNAME" <> 'Yemeksepeti'
```

```
        )
        SELECT
        CD."RESTAURANT_ID",
        CD."USERNAME",
        CD."COMMENT_TEXT",
        CASE WHEN CD."COMMENT_DATE_INTERVAL" = ' today' THEN CD."DATE" ELSE CD.
↪"DATE" - CAST(CD."COMMENT_DATE_INTERVAL" AS INTERVAL) END::date AS␣
↪"COMMENT_DATE",
        CD."SPEED",
        CD."SERVING",
        CD."FLAVOUR",
        CD."DATE"
        FROM CLEAN_DATA CD;
        """
    comment_df = pd.read_sql(sql_command,conn)
    comment_df['COMMENT_TEXT'] = comment_df['COMMENT_TEXT'].apply(clean_text)
    comment_df
    df_column_conversation(comment_df, 'RESTAURANT_ID', 'text')
    df_column_conversation(comment_df, 'USERNAME', 'text')
    df_column_conversation(comment_df, 'COMMENT_TEXT', 'text')
    df_column_conversation(comment_df, 'COMMENT_DATE', 'date')
    df_column_conversation(comment_df, 'SPEED', 'integer')
    df_column_conversation(comment_df, 'SERVING', 'integer')
    df_column_conversation(comment_df, 'FLAVOUR', 'integer')
    df_column_conversation(comment_df, 'DATE', 'date')
    comment_df.replace('<NA>', 'NULL', inplace=True)
    execute_mogrify(conn,comment_df,"EDW","DWH_FB_COMMENT")
```

[12]:
```
sql_command = """
    SELECT
    *
    FROM "EDW"."DWH_FB_COMMENT" EFR;
    """
comment_df = pd.read_sql(sql_command,conn)
comment_df
```

[12]:

	RESTAURANT_ID	USERNAME	\
0	magic-akademi-kartal-esentepe-mah-istanbul	…i	
1	magic-akademi-kartal-esentepe-mah-istanbul	…3	
2	magic-akademi-kartal-esentepe-mah-istanbul	…0	
3	magic-akademi-kartal-esentepe-mah-istanbul	…t	
4	magic-akademi-kartal-esentepe-mah-istanbul	…e	
…	…	…	
271838	magic-akademi-kartal-esentepe-mah-istanbul	…1	
271839	magic-akademi-kartal-esentepe-mah-istanbul	…d	
271840	magic-akademi-kartal-esentepe-mah-istanbul	…k	
271841	magic-akademi-kartal-esentepe-mah-istanbul	…7	

```
271842  magic-akademi-kartal-esentepe-mah-istanbul     …l

                                                COMMENT_TEXT COMMENT_DATE  SPEED  \
0       ilk kez sipariş verdim bu restauranttan patat…   2021-01-10   10.0
1       her söyledigimde somon daha da kötüleşiyor bu …   2021-01-10    9.0
2                                                  itu   2021-01-10    8.0
3          18 dkda kapıdaydi sıcak ve lezzetli süpersiniz   2021-01-10   10.0
4       dağ kekikli tavuğun yanında gelen makarna bild…   2021-01-10   10.0
…                                                     …          …    …
271838       anladık ki hamburgerler kötü pizzadan devam   2021-01-10    5.0
271839  tiramisu malesef rezaletti kreması ekşimişti y…   2021-01-10    1.0
271840    dağ kekikli tavuk inanılmaz güzel bağımlısıyım   2021-01-10   10.0
271841  waffle söyledik kalın bir hamur üzerine 6 7 kü…   2021-01-10    6.0
271842  pizza adana börek ve pasta nefisti teşekkür ed…   2021-01-10   10.0

        SERVING  FLAVOUR        DATE
0             5        8  2021-06-10
1             6        4  2021-06-10
2             8        8  2021-06-10
3            10       10  2021-06-10
4             6        3  2021-06-10
…           …        …          …
271838        5        5  2021-06-10
271839        1        1  2021-06-10
271840       10       10  2021-06-10
271841        4        3  2021-06-10
271842       10        9  2021-06-10

[271843 rows x 8 columns]
```

[13]:
```
stop_words = [element for element in stopwords.words('turkish') if element not␣
 ↪in␣
 ↪['çok','eğer','gibi','hiç','niçin','niye','sanki','yani','en','az','birkaç','bazı','aslında
comment_df['COMMENT_TEXT'] = comment_df['COMMENT_TEXT'].apply(lambda x: ' '.
 ↪join([word for word in x.lower().split() if word not in (stop_words)]))
#comment_df['COMMENT_TEXT'] = comment_df['COMMENT_TEXT'].apply(lambda x: ' '.
 ↪join([stemmer.stem(word) for word in x.split()]))
comment_df
```

[13]:
```
                                    RESTAURANT_ID USERNAME  \
0       magic-akademi-kartal-esentepe-mah-istanbul     …i
1       magic-akademi-kartal-esentepe-mah-istanbul     …3
2       magic-akademi-kartal-esentepe-mah-istanbul     …0
3       magic-akademi-kartal-esentepe-mah-istanbul     …t
4       magic-akademi-kartal-esentepe-mah-istanbul     …e
…                                                 …      …
271838  magic-akademi-kartal-esentepe-mah-istanbul     …1
```

```
271839  magic-akademi-kartal-esentepe-mah-istanbul     …d
271840  magic-akademi-kartal-esentepe-mah-istanbul     …k
271841  magic-akademi-kartal-esentepe-mah-istanbul     …7
271842  magic-akademi-kartal-esentepe-mah-istanbul     …l

                                              COMMENT_TEXT COMMENT_DATE  SPEED  \
0       ilk sipariş verdim restauranttan patates sıca…   2021-01-10   10.0
1       söyledigimde somon kötüleşiyor sefer fettucini…   2021-01-10    9.0
2                                                  itu   2021-01-10    8.0
3            18 dkda kapıdaydi sıcak lezzetli süpersiniz   2021-01-10   10.0
4       dağ kekikli tavuğun yanında gelen makarna bild…   2021-01-10   10.0
…                                                   …          …    …
271838          anladık hamburgerler kötü pizzadan devam   2021-01-10    5.0
271839  tiramisu malesef rezaletti kreması ekşimişti y…   2021-01-10    1.0
271840    dağ kekikli tavuk inanılmaz güzel bağımlısıyım   2021-01-10   10.0
271841  waffle söyledik kalın bir hamur üzerine 6 7 kü…   2021-01-10    6.0
271842   pizza adana börek pasta nefisti teşekkür ederiz   2021-01-10   10.0

        SERVING  FLAVOUR        DATE
0             5        8  2021-06-10
1             6        4  2021-06-10
2             8        8  2021-06-10
3            10       10  2021-06-10
4             6        3  2021-06-10
…           …        …          …
271838        5        5  2021-06-10
271839        1        1  2021-06-10
271840       10       10  2021-06-10
271841        4        3  2021-06-10
271842       10        9  2021-06-10

[271843 rows x 8 columns]
```

3 Machine Learning

3.1 Bag of Words

3.1.1 Gaussian Naive Bayes

[14]:
```
model = CountVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))
#print(np.shape(labels))
```

```
features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = GaussianNB()
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
<ipython-input-14-57e8c84ed6cb>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
Total Features after vectorizing: 2733
Accuracy Score: 31.33%
F1 Score: 31.33%
Precision Score: 31.33%
```

3.1.2 Support Vector Machines

C-Support Vector Classification

[15]:
```
model = CountVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = SVC()
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2733
<ipython-input-15-31bcc9e0fb71>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 64.19%
F1 Score: 64.19%
Precision Score: 64.19%
```

Linear Support Vector Classification

[16]:
```
model = CountVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = LinearSVC()
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2733
<ipython-input-16-abecfc9eee10>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 60.05%
F1 Score: 60.05%
Precision Score: 60.05%
```

3.1.3 Decision Tree Classifier

[17]:
```
model = CountVectorizer(min_df=3)
not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))
```

```
features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

params = {
    'max_depth': [10,13,14,15],
    'min_samples_split': [2,3,4]
    }

gscv = GridSearchCV(DecisionTreeClassifier(), params, cv=5)
gscv.fit(features_train, labels_train)
print("Best parameters: {best_parameters}".format(best_parameters = gscv.
 ↪best_params_))

model = gscv.best_estimator_
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2619
<ipython-input-17-de71e106f8b8>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
Best parameters: {'max_depth': 13, 'min_samples_split': 2}
Accuracy Score: 63.09%
F1 Score: 63.09%
Precision Score: 63.09%
```

3.1.4 Logistic Regression

[18]:
```
model = CountVectorizer(min_df=3)
not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)
```

```
model = LogisticRegression(penalty='l2', C=1.2, n_jobs=-1)
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2619
<ipython-input-18-eccb7253df41>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 63.19%
F1 Score: 63.19%
Precision Score: 63.19%
```

3.2 Bag of Words with TF-IDF Vectorizer

3.2.1 Gaussian Naive Bayes

[19]:
```
model = TfidfVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))
#print(np.shape(labels))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = GaussianNB()
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2733
```

```
<ipython-input-19-6f54f08c5ef9>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 31.43%
F1 Score: 31.43%
Precision Score: 31.43%
```

3.2.2 Support Vector Machines

C-Support Vector Classification

[20]:
```
model = TfidfVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

params = {
    'C': [1.5],
    #'C': [1.4,1.5,1.6],
    'kernel': ['rbf']
    #'kernel': ['linear','rbf','sigmoid']
    }

gscv = GridSearchCV(SVC(), params, cv=5)
gscv.fit(features_train, labels_train)
print("Best parameters: {best_parameters}".format(best_parameters = gscv.
 ↪best_params_))

model = gscv.best_estimator_
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2733
<ipython-input-20-ed41f9384cd3>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
```

```
Best parameters: {'C': 1.5, 'kernel': 'rbf'}
Accuracy Score: 65.76%
F1 Score: 65.76%
Precision Score: 65.76%
```

Linear Support Vector Classification

[21]:
```
model = TfidfVectorizer(min_df=3)
not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = LinearSVC()
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2733
<ipython-input-21-61109543768c>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:10000][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 63.05%
F1 Score: 63.05%
Precision Score: 63.05%
```

3.2.3 Decision Tree Classifier

[22]:
```
model = TfidfVectorizer(min_df=3)
not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)
```

```
params = {
    'max_depth': [9,10,11],
    'min_samples_split': [2,3]
    }

gscv = GridSearchCV(DecisionTreeClassifier(), params, cv=5)
gscv.fit(features_train, labels_train)
print("Best parameters: {best_parameters}".format(best_parameters = gscv.
 ↪best_params_))

model = gscv.best_estimator_
model.fit(features_train, labels_train)

label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2619
<ipython-input-22-b6a83c6a06e2>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
Best parameters: {'max_depth': 10, 'min_samples_split': 2}
Accuracy Score: 62.77%
F1 Score: 62.77%
Precision Score: 62.77%
```

3.2.4 Logistic Regression

[23]:
```
model = TfidfVectorizer(min_df=3)
not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
features = model.fit_transform(not_null_df['COMMENT_TEXT'].values).todense()
labels = not_null_df['SPEED'].values
print("Total Features after vectorizing: {total_features}".
 ↪format(total_features = np.shape(features)[1]))

features_train, features_test, labels_train, labels_test =␣
 ↪train_test_split(features, labels, test_size = 0.3, random_state = 9)

model = LogisticRegression(penalty='l2', C=1.2, n_jobs=-1)
model.fit(features_train, labels_train)
```

```
label_prediction = model.predict(features_test)

print("Accuracy Score: {accuracy_score:0.2f}%".format(accuracy_score =␣
 ↪accuracy_score(labels_test, label_prediction)*100))
print("F1 Score: {f1_score:0.2f}%".format(f1_score = f1_score(labels_test,␣
 ↪label_prediction, average='micro')*100))
print("Precision Score: {precision_score:0.2f}%".format(precision_score  =␣
 ↪precision_score(labels_test, label_prediction, average='micro')*100))
```

```
Total Features after vectorizing: 2619
<ipython-input-23-3703865826cd>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  not_null_df = comment_df[0:9600][pd.notnull(comment_df['SPEED'])]
Accuracy Score: 65.71%
F1 Score: 65.71%
Precision Score: 65.71%
```

3.3 Markov Chains

[24]:
```
sql_command = """
    WITH COMMENTS AS(
    SELECT "RESTAURANT_ID",
    STRING_AGG(LOWER("COMMENT_TEXT"), ' ') AS ALL_COMMENT,
    COUNT("COMMENT_TEXT") AS COMMENT_COUNT
    FROM "ODS"."EXT_FB_COMMENT" EFC
    GROUP BY "RESTAURANT_ID"
    )
    SELECT *
    FROM COMMENTS
    ORDER BY COMMENT_COUNT DESC;
    """
restaurant_df = pd.read_sql(sql_command,conn)
restaurant_df.head()
```

[24]:

	RESTAURANT_ID	all_comment	comment_count
0	pizza-bulls-uskudar-altunizade-mah-istanbul	lezzeti eskisi gibi gelmedi bize. fark ödeyip …	2293
1	pizza-bulls-uskudar-ferah-mah-istanbul	her şey çok güzeldi. elinize sağlık. hizli ve …	1764
2	pizza-bulls-kartal-soganlik-yeni-mah-istanbul	cok pahali olmasi disinda bir problem yok gibi…	1743
3	pizza-bulls-umraniye-cakmak-mah-istanbul	çok lezzetli güvenle çoçuguma yedirebiliyorum …	1582
4	pizza-bulls-atasehir-fetih-mah-istanbul	servis ve hız 10 üzerinden 20.. bu siparişi is…	1277

[25]:
```
import random

def make_markov_model(cleaned_stories, n_gram=2):
    markov_model = {}
    for i in range(len(cleaned_stories)-n_gram-1):
        curr_state, next_state = "", ""
        for j in range(n_gram):
            curr_state += cleaned_stories[i+j] + " "
            next_state += cleaned_stories[i+j+n_gram] + " "
        curr_state = curr_state[:-1]
        next_state = next_state[:-1]
        if curr_state not in markov_model:
            markov_model[curr_state] = {}
            markov_model[curr_state][next_state] = 1
        else:
            if next_state in markov_model[curr_state]:
                markov_model[curr_state][next_state] += 1
            else:
                markov_model[curr_state][next_state] = 1

    # calculating transition probabilities
    for curr_state, transition in markov_model.items():
        total = sum(transition.values())
        for state, count in transition.items():
            markov_model[curr_state][state] = count/total

    return markov_model

def generate_story(markov_model, limit=100, start='my god'):
    n = 0
    curr_state = start
    next_state = None
    story = ""
    story+=curr_state+" "
    while n<limit:
        next_state = random.choices(list(markov_model[curr_state].keys()),
                                    list(markov_model[curr_state].values()))

        curr_state = next_state[0]
        story+=curr_state+" "
        n+=1
    return story
```

[26]:
```
restaurant_df['clean_comment'] = restaurant_df['all_comment'].apply(clean_text)
text = restaurant_df.loc[0]['clean_comment'].split()

markov_model = make_markov_model(text)
```

```
for i in range(20):
    print(str(i)+". ", generate_story(markov_model, start="yemek çok",␣
 ↪limit=20))
```

```
0.  yemek çok güzeldi semih beye ilgisinden dolayı tesekkurler atakan kardeşim
çok ilgili teşekkür ederim guzel güzel güzeldi güzlel harika harikaaaaa tavsiye
ederim efsane lezzetin sıcacık ve çok lezzetliydi ayrıca ikram sufle için de
teşekkürler sıcak olarak hızlı bir şekilde getirdiği için cumali
1.  yemek çok güzeldi semih beyin özel ilgisi için ayrıca teşekkür ederim
oğlumun doğum günü sebebiyle sufle hediyesiyle bizi oldukça mutlu etti çok
lezzetliydi çok hızlı geldi çalışanlara arkadaşlara çok teşekkür ediyorum her
zamanki gibi müthiş hizli servisleri sahane pizzalari ve kadinlar gunu
2.  yemek çok lezzetliydi ve yanında 2 adet ikram sufle geldi 2 tane multinetten
çektiği için surat yaptı ve maskesizdi bir daha asla suflenize bayılıyoruz
gerçekten 30 liraya sufle satan yerlere taş çıkartırsınız hep böyle devam eder
pizzalar gerçekten güzel ve lezzetliydi pizzamız
3.  yemek çok güzeldi semih beye çok teşekkür ederiz çok lezzetliydi teşekkürler
tatlı için teşekkür ederim özellikle aramanız inceliktir pizza çok küçüktü diğer
restoranlarda bu fiyata büyük bi pizza getirdiği için teşekkür ederim pizza
mükemmeldi mükemmeldi pizza da tek gecerim doyurucu lezzetli temiz
4.  yemek çok lezzetliydi ve sıcak geldi suffle ikramı içinde teşekkürler bilal
beyin ilgisine teşekkür ederim tatlı ikramınız için çok teşekkür ederim sipariş
gelmeden önce arayıp bilgi verdi ikramlarınız için ayrıca teşekkür ederiz
yardımcı oldu siparişim neredeyse 1 saatte ve soğuk geldi notum
5.  yemek çok lezzetliydi ve restoran bana çok yakın olmamasına rağmen sıcaktı
da soğan halkaları da tam tersi az pişmişti ama pizzanın lezzeti malzeme
kalitesi ve malzeme kalitesi iyiydi sufle ikramı için semih bey e de ikram
gönderdi şiddetle tavsiye edilir muazzam adeta
6.  yemek çok lezzetliydi ve kuriyeniz atakan bey çok nazik teşekkürler küçük
boy new jersey siparişi vermiştim gayet güzeldi çok hızlı sıcak güzel çok iyi
nazik bi insan dı kendisine ayrıca teşekkür ederiz hızlı sıcak lezzetli çok
lezzetliydi sicak geldi ikraminiz içinde teşekkür
7.  yemek çok lezzetliydi ve hızlıydı tesekkurler çok hızlı ve sıcak geldi
getiren çalışan arkadaşımız güler yüzle teslim etti sufle hediye edeceğiz diyip
göndermeyerek beni üzseniz de seviyorum sufle ikramı için bilal beye
ilgilerinden dolayı teşekkür ediyorum kuryenize de geçmiş olsun her zaman
8.  yemek çok lezzetliydi ve hızlı ulaştı he zamanki gibi cumali beye ayrıca
teşekkür ederim hızlı sıcak ve lezzetli geldi atakan beye ayrıca teşekkür ederim
bize hediye sufle gönderen atakan beye çok teşekkür ederiz çok hızlı ve sıcak
geldi ellerinize sağlık pizza bullsdan
9.  yemek çok lezzetliydi ve kurye de çok hızlıydı semih bey sağolsun hızlı ve
sıcak geldi bilal beyin göndermiş olduğu süpriz tatlı çocuklarımı ve beni çok
mutlu etti tek kelimeyle mukemmel pizzalar sicacik geliyor ve citir bir hamuru
var teslimat cok hizli yedigimiz
10.  yemek çok güzeldi semih beye teşekkürler ürünler gayet sıcak geldi sıcak ve
lezzetli ürünler ilgilenen arkadşlarada teşekür ederim hızlı sıcak ve lezzetli
pizzaları için pizza bulls tatlı ikramları için çok çok teşekkür her şey çok
```

güzeldi ürün sipariş ettiğimiz gibi ortalama bir
11. yemek çok lezzetliydi ve sıcak geldi cumali beye çok teşekkürler kuryeniz cumali beye kibarlığından dolayı teşekkür ederim atakan beye ikramlar icin teşekkür ederim fakat 1 saatte geldi çok lezzetliydi ayrıca ikramları için çok teşekkür ederiz selim beyin gonderdigi sufle icin cok teşekkür
12. yemek çok güzeldi semih beye çok teşekkürler ilgisi için keep up the good work d muhteşem hızlı taze ve sıcak geldi ayrıca teslimatı yapan ibrahim beye ayrıca teşekkürler pizzaları sıcacık ve kurye çok nazik bir beyefendiydi 2 senedir benim pizzamın adresi burasıayrıca
13. yemek çok lezzetliydi ve kurye de çok hızlıydı atakan beye ilgisinden dolayı teşekkürler bilal beye inceligi ve hediyeleri içn çok teşekkür ederim harika hatalarını telafi etmek için elinden geleni yapan bir firma tatlı ikramı için ayrıca teşekkürler soguk ve geç saat olduğu
14. yemek çok güzeldi semih beyin özel ilgisi için teşekkürler yarım saat olmadan sipariş sıcacık şekilde geldi bilal beye teşekkür ederiz sağlıkla kalın her zamanki gibi harikaydi sufle ikrami icin tesekkur ediyorum pizza dışındaki yan ürünler lezzet açısından sınıfta kaldı pizzanın dilim kesimleri
15. yemek çok lezzetliydi ve getiren arkadasta cok hizli ve saygiliydi kuryeniz mustafa beye teşekkür ederiz hiç beğenmedim hiçbir siparişimde beni mağdur etmenize rağmen bir kere bile pişman olmadım şaşırtmadı her zaman harika tatlı hediye göndermişler çok lezzetliydi teşekkürler tatlı ikramı için ayrıca
16. yemek çok güzeldi semih beye ilgisinden dolayı teşekkür ederiz atakan beye teşekkürler sıcacıktı kurye mustafa pizzaları çok hızlı ve sıcak geldi başarılarının devamını dilerim ridvan beydi sanirim tesekkurler semih beye çok teşekkürler böylesi karlı bir havada pizzalar sıcacık ve çok ilgiliydi pizza
17. yemek çok lezzetliydi ve hızlıydı tesekkurler çok hızlı teslimat ettiği için teşekkürler bilal bey olmak üzere tüm altunizade şubesi pizza bulls çalışanlarına özellikle duygu hanıma ilgi ve alakalarından dolayı bilal beye çok teşekkür ederim cumali beye ayrıca teşekkürler zincir pizzacılar arasındaki en
18. yemek çok güzeldi semih beye teşekkürler bilal beye tatlı ikramı ve ilgisinden dolayı tekrar teşekkür ediyorum cumaliye teşekkürler çıtır tavukların tadı hariç güzeldi pizzaları bol malzemeli geldi harikaydı e güzeldi baya elinize saglik teşekkürler fasfasf gayet başarılı ayrıca bilal beye de jestinden
19. yemek çok güzeldi semih beye ilgisinden dolayı teşekkür ederim bilal bey teşekkürler bilal bey in ilgi ve alakasina cok tesekkur ederiz semih beye ikramı yardımı ve ilgisi için servis hızlıydı kurye de çok nazikti kendileri pizzamız hızlı ve lezzetli kuryeler hep güler

4 Sources

1. Selenium Documentation
2. Selenium choose element by partial id
3. Selenium scroll down to end of the page
4. Selenium click button
5. Markov Chains
6. Bag of Words
7. Turkish Porter Stemmer