Programming Assignment 2

The goal of this programming assignment is to (i) provide hands-on experience with working with unstructured (ii) perform basic data cleaning and integration, and (iii) data exploration and visualization through Pandas. Once you get going, this assignment should not take very long, but we strongly recommend that you do not wait until the last minute to start!

You will be working with the [Yelp (https://www.kaggle.com/yelp-dataset/yelp-dataset)](https://www.kaggle.com/yelp-dataset/yelp-dataset) dataset. The Yelp dataset is a large dataset consisting of reviews of businesses, business, and user information.

Note that for this homework, you will be asked to craft various queries, followed by executing them. Please include as part of your written response ALL of the queries that you have executed corresponding to each step of the instruction, as well as any additional information requested in the specific questions.

The response to this assignment needs to be submitted as one *single pdf* document, named as `Programming Assignment 2_YourName.pdf` . We will be using BCourses for collecting the homework assignments. Please submit your answers via BCourses. If you are submitting code files or notebooks as your responses, please *clearly mark* which lines of code correspond to which question number *in the correct question order*. Exclude any code that is not relevant to the what is asked by the question. The assignment is due on **4/22 midnight**.

Feel free to talk to other members of the class in doing the homework. You should, however, write down your solutions yourself. List the names of everyone you worked with at the top of your submission.

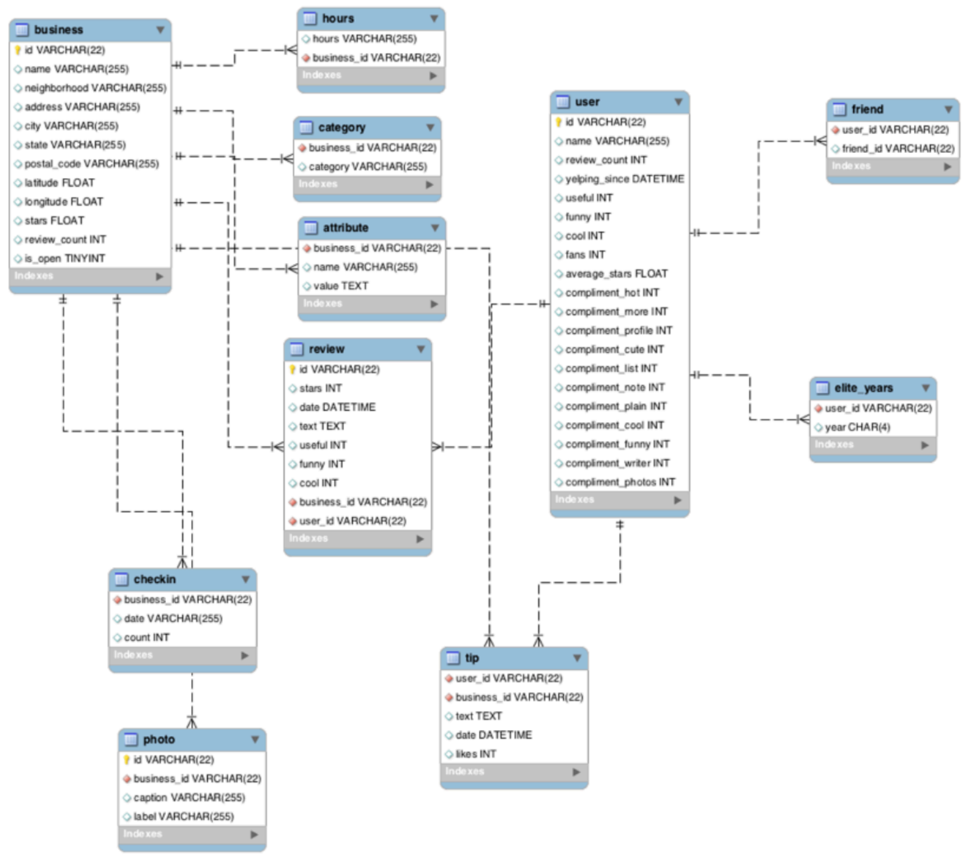
Question 1: Working with Unstructured Data (20pt)

Inspect the three JSON collections `business` , `review` , and `user` . If you were to store this into a relational database, how would you store it? List a sketch of the schema. List two benefits for storing this data in a NoSQL database like Firestore over a relational database management system (RDBMS) such as Postgres. Please point to specific characteristics regarding the Yelp dataset in your explanation. (Hint: Think about what information would be hard to model if this data was stored in a RDBMS.)

Answer:

If I were to store it in a relational database like Postgres, I'd either (1) flatten (denormalize) the data and put all the features into separate columns in 1 table, (2) normalize the data and split it into different tables, or (3) use the json data type and store the nested features there.

1. NoSQL databases model data in a less rigid structure, making it more flexible
2. Removes all the barriers faced from a Relational DB:
    - We have flexible schemas
    - Attributes can take collection data types (json/xml)
    - Attributes can be nested
    - Easier to read
    - No need to use joins



Question 2 : ETL with Pandas (40pt)

In the second part of this assignment, we will be exporting the `business` , `user` , and `sample_review` collections into Pandas.

a) **[10pt]** Export JSON collection to Pandas:

We are now going to import the three JSON collections `business` , `user` , and `sample_review` into three separate Pandas dataframes. Use `json_normalize` command to expand nested fields such as `{"attribute":["DriveThru": true]}` into `attribute.DriveThru`

```
# Load the collections into Pandas. Below you can find how use
r collection is imported into a dataframe
from pandas.io.json import json_normalize
import json

with open('yelp_academic_dataset_user.json','r') as f:
    userdict = json.load(f)

user_df = json_normalize(userdict)
```

Inspect the dataframes `user_df` , `business_df` , `review_df` , and describe how the dataframe representation differs from the document representation in MongoDB? (Hint: What do you notice about the `attribute.*` columns?)

Pandas vs MongoDB: data representation

As we can see, the JSON data in Pandas is represented in tabular format in the dataframes, where all the features in the JSON format are represented as columns of a large data table. On the other hand, MongoDB represents this JSON data as documents with key-value pairs. Each row (Pandas) is represented as a document (MongoDB).

1. In MongoDB, the features within the attributes key-value pair are represented as nested data
2. In pandasm through the json_normalize method we are able to flatten those nested key-value pairs (lists of dictionaries) and each feature is represented as a table column.

- Note: there are still a couple of columns that have more nested data within their columns, but that's because the JSON file had quotation marks around them, terning them into long strings, as opposed to dictionaries and where undetected by the "json_normalize" method. In order to flatten these layers, I've demonstrated an extra step (which is commented out)

In [1]:
```
# imported packages

# Q. 1
import json
import pandas as pd
from pandas import json_normalize
# Q. 2
from functools import reduce
import matplotlib.pyplot as plt
%matplotlib inline
# Q. 3
from scipy.stats import ks_2samp
import numpy as np
import seaborn as sns
sns.set_style("white")
# Q. 4
#pip install firebase_admin
import firebase_admin
from firebase_admin import firestore, credentials, auth
```

```
In [2]: # Since we'll repeat the normalization of JSON data for 3 JSON files,
        # it's good practice to create functions that convert JSON files into

        def read_json_as_array(json_file):
            '''
            Read a given Yelp JSON file as string, adding opening / closing
            brackets and commas to convert from separate JSON objects to
            an array of JSON objects, so JSON aware libraries can properly rea

            Parameters: json_file (file of type json)
            -----------
            Returns:    json_data: str (String representation of JSON array)
            -------
            '''

            json_data = ''

            with open(json_file, 'r', encoding='utf-8') as in_file:
                for i, line in enumerate(in_file):
                    if i == 0 and line:
                        json_data += '[' + line
                    elif line:
                        json_data += ',' + line
                    else:
                        pass
                json_data += ']\n'

            return json_data


        def load_json(json_data):
            '''
            Read and normalize a given JSON array into a pandas DataFrame

            Parameters: json_data: str (String representation of JSON array)
            -----------
            Returns:    df: pandas.DataFrame (DataFrame containing the normali
            -------
            '''

            data = json.loads(json_data)
            df = json_normalize(data)

            return df

        user_df = load_json(read_json_as_array("yelp_academic_dataset_user.jso
        business_df = load_json(read_json_as_array("yelp_academic_dataset_busi
        review_df = load_json(read_json_as_array("yelp_academic_dataset_review
```

Data Exploration

In [3]:
```
# Method to flatten nested JSON data that goes undetected from json_nc
# # First remove the json_normalize(data) from the def load_json(json_
# res = [i['attributes'] for i in business_df]

# res[3]

# Result displayed:
#  {'RestaurantsDelivery': 'False',
#  'OutdoorSeating': 'False',
#  'BusinessAcceptsCreditCards': 'False',
#  'BusinessParking': "{'garage': False, 'street': True, 'validated':
#  'BikeParking': 'True',
#  'RestaurantsPriceRange2': '1',
#  'RestaurantsTakeOut': 'True',
#  'ByAppointmentOnly': 'False',
#  'WiFi': "u'free'",
#  'Alcohol': "u'none'",
#  'Caters': 'True'}

# import ast

# # initializing string
# re = res[3]['BusinessParking']

# # printing original string
# print("The original string : " + str(res[3]['BusinessParking']))

# # using ast.literal_eval()
# # convert dictionary string to dictionary
# ast.literal_eval(res[3]['BusinessParking'])

# # print result
# print("The converted dictionary : " + str(ast.literal_eval(res[3]['E

# attr = json_normalize(ast.literal_eval(res[3]['BusinessParking']))
```

In [4]:
```
business_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150346 entries, 0 to 150345
Data columns (total 60 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   business_id                                 150346 non-null  object
 1   name                                        150346 non-null  object
 2   address                                     150346 non-null  object
 3   city                                        150346 non-null  object
 4   state                                       150346 non-null  object
 5   postal_code                                 150346 non-null  object
 6   latitude                                    150346 non-null  float64
 7   longitude                                   150346 non-null  float64
 8   stars                                       150346 non-null  float64
```

```
 9   review_count                           150346 non-null  int64
 10  is_open                                150346 non-null  int64
 11  categories                             150243 non-null  object
 12  hours                                  0 non-null       float64
 13  attributes.ByAppointmentOnly           42339 non-null   object
 14  attributes.BusinessAcceptsCreditCards  119765 non-null  object
 15  hours.Monday                           114474 non-null  object
 16  hours.Tuesday                          120631 non-null  object
 17  hours.Wednesday                        123771 non-null  object
 18  hours.Thursday                         125198 non-null  object
 19  hours.Friday                           124999 non-null  object
 20  hours.Saturday                         110770 non-null  object
 21  attributes.BikeParking                 72638 non-null   object
 22  attributes.RestaurantsPriceRange2      85314 non-null   object
 23  attributes.CoatCheck                   5584 non-null    object
 24  attributes.RestaurantsTakeOut          59857 non-null   object
 25  attributes.RestaurantsDelivery         56282 non-null   object
 26  attributes.Caters                      40127 non-null   object
 27  attributes.WiFi                        56914 non-null   object
 28  attributes.BusinessParking             91085 non-null   object
 29  attributes.WheelchairAccessible        28953 non-null   object
 30  attributes.HappyHour                   15171 non-null   object
 31  attributes.OutdoorSeating              48802 non-null   object
 32  attributes.HasTV                       45084 non-null   object
 33  attributes.RestaurantsReservations     45247 non-null   object
 34  attributes.DogsAllowed                 18284 non-null   object
 35  hours.Sunday                           81172 non-null   object
 36  attributes.Alcohol                     43189 non-null   object
 37  attributes.GoodForKids                 53375 non-null   object
 38  attributes.RestaurantsAttire           39255 non-null   object
 39  attributes.Ambience                    44279 non-null   object
 40  attributes.RestaurantsTableService     19982 non-null   object
 41  attributes.RestaurantsGoodForGroups    44170 non-null   object
 42  attributes.DriveThru                   7760 non-null    object
 43  attributes                             0 non-null       float64
 44  attributes.NoiseLevel                  37993 non-null   object
 45  attributes.GoodForMeal                 29087 non-null   object
 46  attributes.BusinessAcceptsBitcoin      17430 non-null   object
 47  attributes.Smoking                     4567 non-null    object
 48  attributes.Music                       7521 non-null    object
 49  attributes.GoodForDancing              4628 non-null    object
 50  attributes.AcceptsInsurance            5713 non-null    object
 51  attributes.BestNights                  5694 non-null    object
 52  attributes.BYOB                        4451 non-null    object
 53  attributes.Corkage                     3553 non-null    object
 54  attributes.BYOBCorkage                 1444 non-null    object
 55  attributes.HairSpecializesIn           1065 non-null    object
 56  attributes.Open24Hours                 39 non-null      object
 57  attributes.RestaurantsCounterService   19 non-null      object
 58  attributes.AgesAllowed                 129 non-null     object
 59  attributes.DietaryRestrictions         31 non-null      object
dtypes: float64(5), int64(2), object(53)
memory usage: 68.8+ MB
```

In [5]:
```python
pd.set_option('display.max_columns', None)
# pd.set_option('display.max_colwidth', None)

user_df.head() # 1000 rows × 22 columns
```

Out[5]:

	user_id	name	review_count	yelping_since	useful	funny	cool	
0	q_QQ5kBBwICcbL1s4NVK3g	Jane	1220	2005-03-14 20:26:35	15038	10030	11291	
1	dIIKEfOgo0KqUfGQvGikPg	Gabi	2136	2007-08-10 19:01:51	21272	10289	18046	200
2	D6ErcUnFALnCQN4b1W_TIA	Jason	119	2007-02-07 15:47:53	188	128	130	
3	JnPIjvC0cmooNDfsa9BmXg	Kat	987	2009-02-09 16:14:29	7234	4722	4035	
4	37Hc8hr3cw0iHLoPzLK6Ow	Christine	495	2008-03-03 04:57:05	1577	727	1124	

In [6]:
```python
review_df.head() # 7500 rows × 9 columns
```

Out[6]:

	review_id	user_id	business_id	stars
0	IWC-xP3rd6obsecCYsGZRg	ak0TdVmGKo4pwqdJSTLwWw	buF9druCkbuXLX526sGELQ	4.0
1	8bFej1QE5LXp4O05qjGqXA	YoVfDbnISlW0f7abNQACIg	RA4V8pr014UyUbDvI-LW2A	4.0
2	NDhkzczKjLshODbqDoNLSg	eC5evKn1TWDyHCyQAwguUw	_sS2LBIGNT5NQb6PD1Vtjw	5.0
3	T5fAqjjFooT4V0OeZyuk1w	SFQ1jcnGguO0LYWnbbftAA	0AzLzHfOJgL7ROwhdww2ew	2.0
4	sjm_uUcQVxab_EeLCqsYLg	0kA0PAJ8QFMeveQWHFqz2A	8zehGz9jnxPqXtOc7KaJxA	4.0

In [7]:
```
business_df.head(10) #158000 x 14 --> 60 columns
```

Out[7]:

	business_id	name	address	city	state	postal_code	latitu
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.4266
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.5511
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.2232
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.9555
4	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA	18054	40.3381
5	CF33F8-E6oudUQ46HnavjQ	Sonic Drive-In	615 S Main St	Ashland City	TN	37015	36.2695
6	n_0UpQx1hsNbnPUSlodU8w	Famous Footwear	8522 Eager Road, Dierbergs Brentwood Point	Brentwood	MO	63144	38.6276
7	qkRM_2X51Yqxk3btIwAQIg	Temple Beth-El	400 Pasadena Ave S	St. Petersburg	FL	33707	27.7665
8	k0hlBqXX-Bt0vf1op7Jr1w	Tsevi's Pub And Grill	8025 Mackenzie Rd	Affton	MO	63123	38.5651
9	bBDDEgkFA1Otx9Lfe7BZUQ	Sonic Drive-In	2312 Dickerson Pike	Nashville	TN	37207	36.2081

b) **[10pt]** Joining Dataframes:

Write a Pandas query that combines `user_df` , `business_df` , `review_df` into one single dataframe called `combined_df` . (Hint: You should first remove the '_id' column in each dataframe, then use the [merge (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html) command to combine dataframes.) This can span multiple statements. State the number of columns in each dataframe, as well as the number of columns in the resulting joined table.

Use the merge command to combine dataframes

- "user_df" --> 1000 rows × 22 columns
- "review_df" --> 7500 rows × 9 columns
- "business_df" --> 150346 rows x 14 columns (unflattened) --> 60 columns (flattened)
- As a result, the "combined_df" dataframe that merges all the dataframes has 180235 rows × 80 columns
- Originally, there would've been 83 columns I wouldn't delete the id columns

In [8]:

```
# remove the '_id' column in each dataframe
user_df2 = user_df.drop('user_id', axis=1) # remove 1 column
review_df2 = review_df.drop(['review_id', 'user_id', 'business_id'], a
business_df2 = business_df.drop(business_df.columns[[0]], axis=1) # re
```

In [9]:

```python
# use the merge command to combine dataframes

combined_df = reduce(lambda left, right:
                     pd.merge(left, right,
                              how = 'outer'),
                     [user_df2, review_df2, business_df2])

combined_df # 180235 rows × 80 columns
```

Out[9]:

	name	review_count	yelping_since	useful	funny	cool	
0	Jane	1220.0	2005-03-14 20:26:35	15038.0	10030.0	11291.0	2006,2007,2008,20
1	Gabi	2136.0	2007-08-10 19:01:51	21272.0	10289.0	18046.0	2007,2008,2009,2010,2
2	Jason	119.0	2007-02-07 15:47:53	188.0	128.0	130.0	
3	Kat	987.0	2009-02-09 16:14:29	7234.0	4722.0	4035.0	20
4	Christine	495.0	2008-03-03 04:57:05	1577.0	727.0	1124.0	
...	...	...	...	...	...	...	
180230	Binh's Nails	13.0	NaN	NaN	NaN	NaN	
180231	Wild Birds Unlimited	5.0	NaN	NaN	NaN	NaN	
180232	Claire's Boutique	8.0	NaN	NaN	NaN	NaN	
180233	Cyclery & Fitness Center	24.0	NaN	NaN	NaN	NaN	
180234	Sic Ink	9.0	NaN	NaN	NaN	NaN	

180235 rows × 80 columns

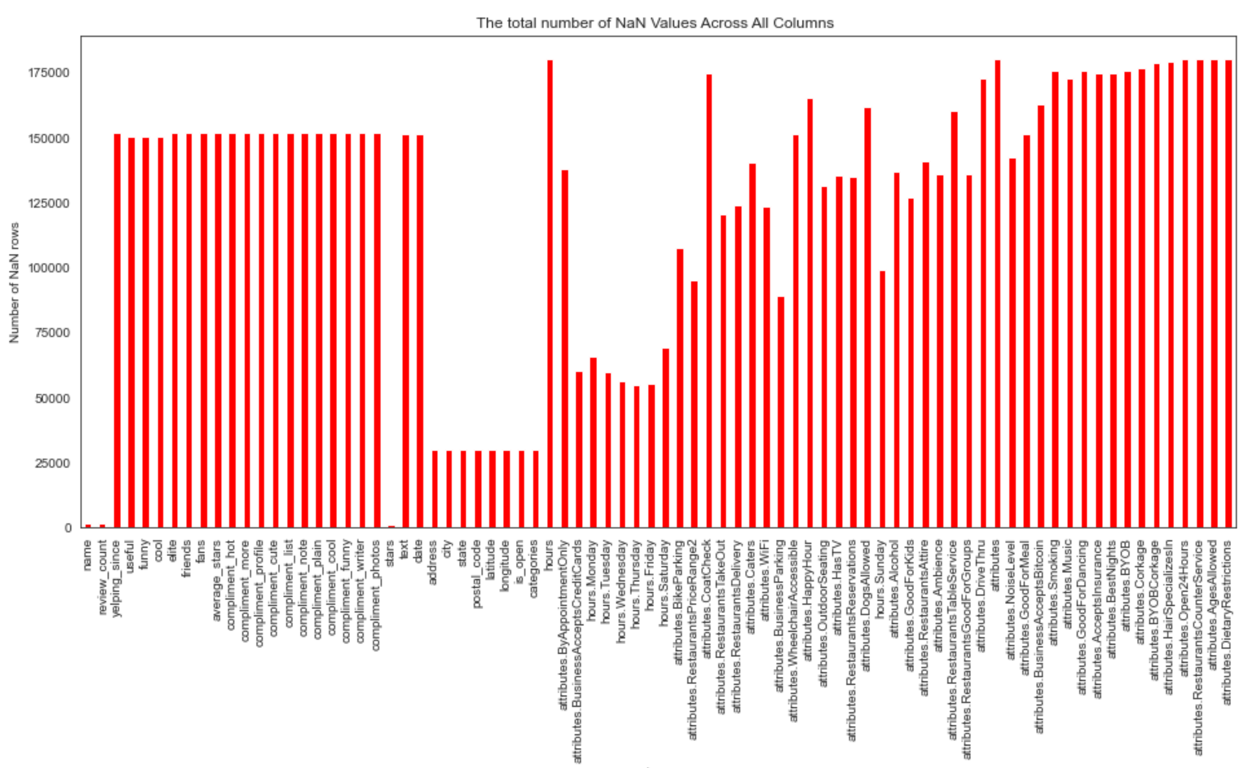
c) **[10pt]** Deriving new fields and dealing with null values:

We observe how due to the nested representation of the data, there is a lot of missing fields with NaN values in the Pandas dataframes. To ease our analysis, we want to get rid of columns that have too many rows with NaN values. First, compute the percentage of NaN values for each column (Hint: You could use the [isnull (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.isnull.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.isnull.html) command to determine what is a NaN value.). Then, plot the histogram distribution the percentage of NaN values across all columns (via `.hist()` function). We will notice that there is a large number of columns that do not have any non-null values. Write a query that keeps only columns with more than 20% non-null values in the `combined_df` dataframe. State the number of columns left in the resulting table.

Plot number of nulls for each column (not %)

In [10]:
```
# number of nulls for each column
vc_nulls = combined_df.apply(lambda x: x.isnull().value_counts()).T[Tr
#vc_nulls.hist() # if you want a histogram of these counts
# or if you wanted to plot the null count of each column as a bar
vc_nulls.plot(kind = 'bar', figsize=(16, 7), color = 'red')
plt.title("The total number of NaN Values Across All Columns ")
plt.xlabel("columns")
plt.ylabel("Number of NaN rows")
```

Out[10]:
```
Text(0, 0.5, 'Number of NaN rows')
```



In [11]:
```
vc_nulls
```
Out[11]:
```
name                                        1575.0
review_count                                1575.0
yelping_since                             151921.0
useful                                    150346.0
funny                                     150346.0
                                            ...
attributes.HairSpecializesIn              179170.0
attributes.Open24Hours                    180196.0
attributes.RestaurantsCounterService      180216.0
attributes.AgesAllowed                    180106.0
attributes.DietaryRestrictions            180204.0
Name: True, Length: 80, dtype: float64
```

Compute percentage of nulls for each column

In [12]:
```
# compute percentage of nulls for each column
percent_missing = combined_df.isnull().sum() * 100 / len(combined_df)
missing_value_df = pd.DataFrame({'column_name': combined_df.columns,
                                 'percent_missing': percent_missing})

missing_value_df
```
Out[12]:

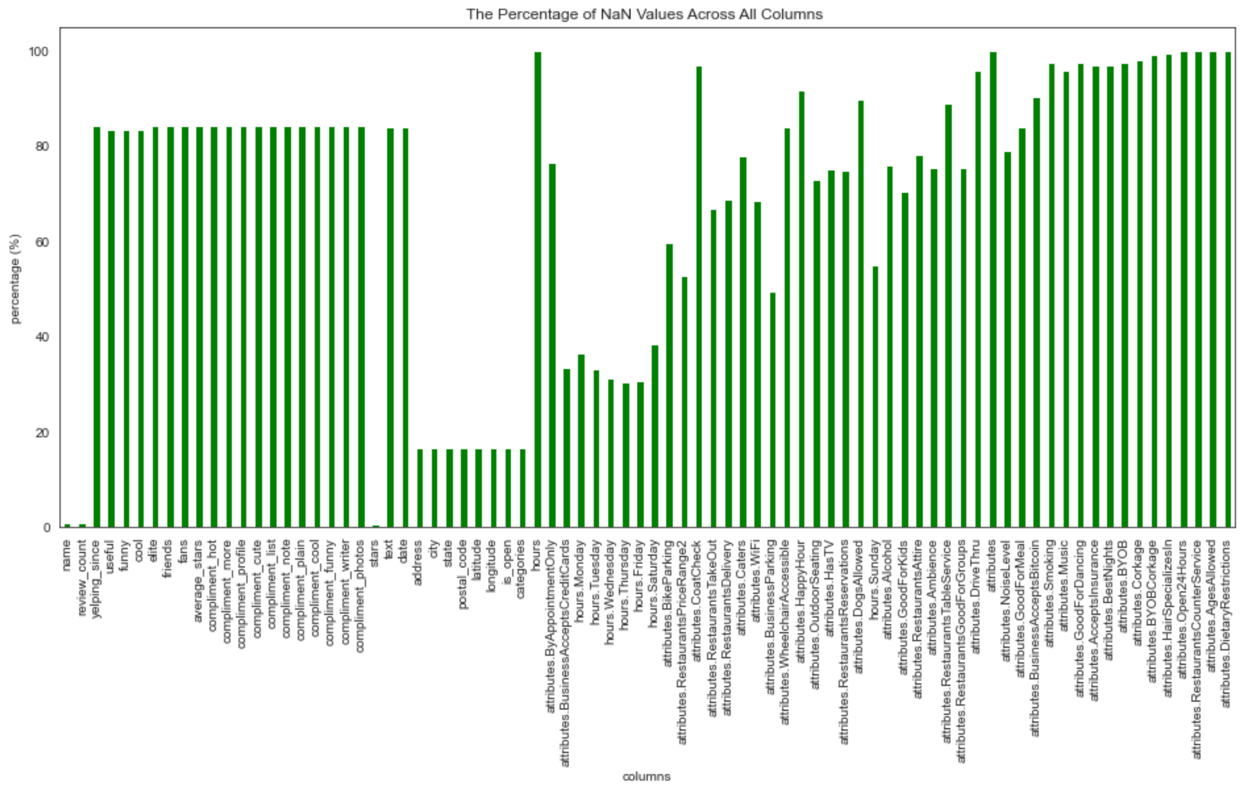
	column_name	percent_missing
name	name	0.873859
review_count	review_count	0.873859
yelping_since	yelping_since	84.290510
useful	useful	83.416650
funny	funny	83.416650
...	...	...
attributes.HairSpecializesIn	attributes.HairSpecializesIn	99.409105
attributes.Open24Hours	attributes.Open24Hours	99.978362
attributes.RestaurantsCounterService	attributes.RestaurantsCounterService	99.989458
attributes.AgesAllowed	attributes.AgesAllowed	99.928427
attributes.DietaryRestrictions	attributes.DietaryRestrictions	99.982800

80 rows × 2 columns

Plot percentage of nulls for each column on a histogram

In [13]:
```
# plot percentage of nulls for each column
percent_missing.plot(kind = 'bar', figsize=(16, 7), color='green')
plt.title("The Percentage of NaN Values Across All Columns ")
plt.xlabel("columns")
plt.ylabel("percentage (%)")
```

Out[13]:
```
Text(0, 0.5, 'percentage (%)')
```



Keep only columns with more than 20% non-null values in the combined_df dataframe

Since the "missing_value_df" contains the "combined_df" columns in its index, I've performed a boolean mask to filter all the columns that have null values above 80%. All those are present in the "del_cols" dataframe, through the "del_cols.index", which I'm utilizing below to remove those columns from our main "combined_df" dataframe

In [14]:
```
# Boolean mask to get columns with NaN above 80%
del_cols = missing_value_df[missing_value_df["percent_missing"]>80]


# remove a list of columns that are included in the dele.index pandas
combined_df = combined_df.drop(del_cols.index, axis=1)
```

In [15]:
```
# Display the filtered dataframe
combined_df # 180235 rows × 36 columns
```

Out[15]:

	name	review_count	stars	address	city	state	postal_code	latitude
0	Jane	1220.0	NaN	NaN	NaN	NaN	NaN	NaN
1	Gabi	2136.0	NaN	NaN	NaN	NaN	NaN	NaN
2	Jason	119.0	NaN	NaN	NaN	NaN	NaN	NaN
3	Kat	987.0	NaN	NaN	NaN	NaN	NaN	NaN
4	Christine	495.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
180230	Binh's Nails	13.0	3.0	3388 Gateway Blvd	Edmonton	AB	T6J 5H2	53.468419
180231	Wild Birds Unlimited	5.0	4.0	2813 Bransford Ave	Nashville	TN	37204	36.115118
180232	Claire's Boutique	8.0	3.5	6020 E 82nd St, Ste 46	Indianapolis	IN	46250	39.908707
180233	Cyclery & Fitness Center	24.0	4.0	2472 Troy Rd	Edwardsville	IL	62025	38.782351
180234	Sic Ink	9.0	4.5	238 Apollo Beach Blvd	Apollo beach	FL	33572	27.771002

180235 rows × 36 columns

d) **[10pt]** Reflect on your experiences with using MongoDB, Pandas, and SQL (from the previous assignment). Comment on the pros/cons of these tools based on their programmability/usability, expected performance, data model/representation, and any additional axes of comparison. In particular, consider what tool(s) you would pick for certain tasks or scenarios that you might encounter in various real-world applications? (e.g., working with nested data, text data, or data with many different types of fields, performing joins on wide tables, processing data that doesn't fit in-memory)

Answer 2d:

pros/cons of these tools based on their:

1. programmability/usability:

    - To use SQL and MongoDB, by definition, means using a database, and a lot of use-cases these days quite simply require bits of data for 'one-and-done' tasks (from .csv, web api, etc.). In these cases loading, storing, manipulating and extracting from a database is not viable and Pandas is a better option.
    - If we have semistructured data MongoDB is a better option since it's designed to work this such data like JSON files in the form of documents, while preserving the power of a database that SQL has.
    - All 3 platforms can work with semi-structured data like JSON, but with MongoDB the workflow is simpler.
2. expected performance:

    - Pandas has limitation. As you can see in this assignment I had to truncate 2 of the 3 JSON files (user and review)in order to work with them on Jupyter Notebooks, because the Kernel kept failing due to the large volumes of data in GB. MongoDB and SQL that work with databases have it easier.
    - In the case of unstructured & semi-structured data, MongoDB performs faster than Postgres, since it's a NoSQL database by design.
3. data model/representation:

    - Pandas works with all kinds of data (structured, semi or unstructured), ranging from tabular, JSON, graphs, images etc.
    - SQL mainly works with tabular data (structured) and semi-structured (JSON).
    - mongoDB works with semi and unstructured data (NoSQL).
4. Additional:

    - As noted from the above points, I'd use each tool in the following way:
    - SQL for large tabular data that requires storage.
    - MongoDB for JSON and other large NoSQL unstructured data.
    - Pandas for most day-to-day data analysis with lower volumes of data (since there is in-memory data-processing limitation), like csv files, text for Natyral Language Processing, Data Visualization, Machine Learning etc.

Question 3: Analysis and Visualization (20pt)

For the following questions, please create a visualization that best address the question. Explain in words the insights conveyed by the visualization, include a justification of why you chose the specific type of visualization or any additional considerations that is not captured by the visualization. We will be using the custom visualization through the `matplotlib` (https://matplotlib.org/) package. Please attach the visualization generated as part of the submission document.

a) **[10pt]** Visualizing Comparisons:

Working with the `review_df`, we want to understand whether reviews marked with a `positiveFlag` tend to have higher average `stars` rating than compared to ones without a `positiveFlag`. Generate the appropriate visualization using `pandas` and `matplotlib`, justify your choice of visualization, and interpret the visualization result in words. (Hint: You may need to fill the NaN value in `positiveFlag` as False.)

Understanding whether "positiveFlag" reviews tend to have higher average stars rating than the ones without

In my review_df I was unable to find any column or attribute that relates to "positiveFlag" and I also checked the very recent Yelp JSON file from Kaggle and they don't appear to be present. Perhaps that is a column/feature associated with an older iteration of the JSON files on the Yelp dataset. However, from this question, I get a sense that we want to see whether other features in the dataset are coorelated with star count.

In [16]:

```
#review_df.positiveFlag #AttributeError: 'DataFrame' object has no att
```

In [17]:

```
# new column to find length of review text
review_df['text_length'] = review_df['text'].apply(len)
```

In [18]:
```
review_df
```

Out[18]:

	review_id	user_id	business_id	s
0	IWC-xP3rd6obsecCYsGZRg	ak0TdVmGKo4pwqdJSTLwWw	buF9druCkbuXLX526sGELQ	
1	8bFej1QE5LXp4O05qjGqXA	YoVfDbnISlW0f7abNQACIg	RA4V8pr014UyUbDvI-LW2A	
2	NDhkzczKjLshODbqDoNLSg	eC5evKn1TWDyHCyQAwguUw	_sS2LBIGNT5NQb6PD1Vtjw	
3	T5fAqjjFooT4V0OeZyuk1w	SFQ1jcnGguO0LYWnbbftAA	0AzLzHfOJgL7ROwhdww2ew	
4	sjm_uUcQVxab_EeLCqsYLg	0kA0PAJ8QFMeveQWHFqz2A	8zehGz9jnxPqXtOc7KaJxA	
...	...	...	...	
7495	mgqBhaEgdqarl9BhnfjlWA	bMYLCx1QiLUoNiZKma__UQ	uC6o9LwG3ejw4RTJjMtFVg	
7496	NfiXHt2F3OWOsBpaGKEkXw	suHRCRzH06l8jjkom0SSyg	ZTT6-SaOmjlY8kkZTHd3SA	
7497	EWztbHWdeXTlVTKZhFPEMA	rnHFkdelIdBCIHBaoICX4g	meznC0oLcwFAeD_3AlyLRw	
7498	IAqL1i68bSc2kInhv-h25w	uZe5h0Oio69Q1W4T41KzGQ	75HV-KqCtn_oHeiLiGIO_w	
7499	AG3W3wpeqNuwRoYU5uuUYg	vec4p1BHgEQVpxoE-70G8Q	XDv29FffNd2dWnDOtZP-wg	

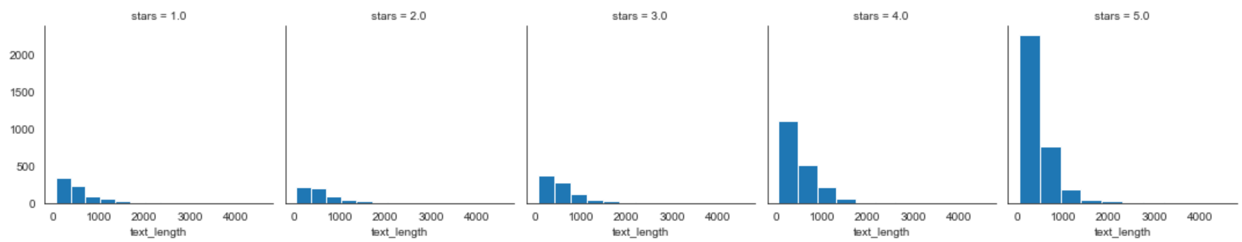
7500 rows × 10 columns

In [19]:
```python
# KS test
# p-value is less than 0.5, meaning that coorelation is insignificant
x = review_df['stars']
y = review_df['text_length']
ks_2samp(x, y, mode = "asymp")
```

Out[19]:
```
KstestResult(statistic=1.0, pvalue=0.0)
```

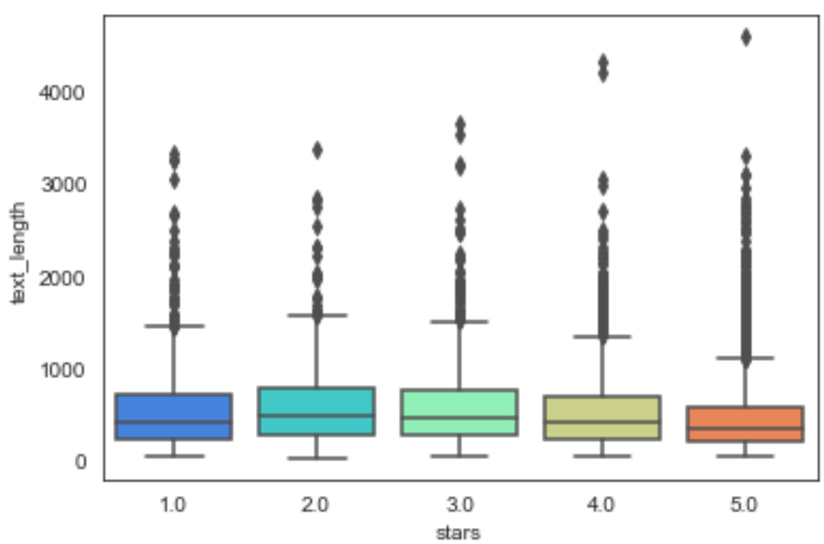
In [20]:
```python
g = sns.FacetGrid(review_df, col="stars")
g.map(plt.hist, "text_length")
```

Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x7fcd5d25d490>
```



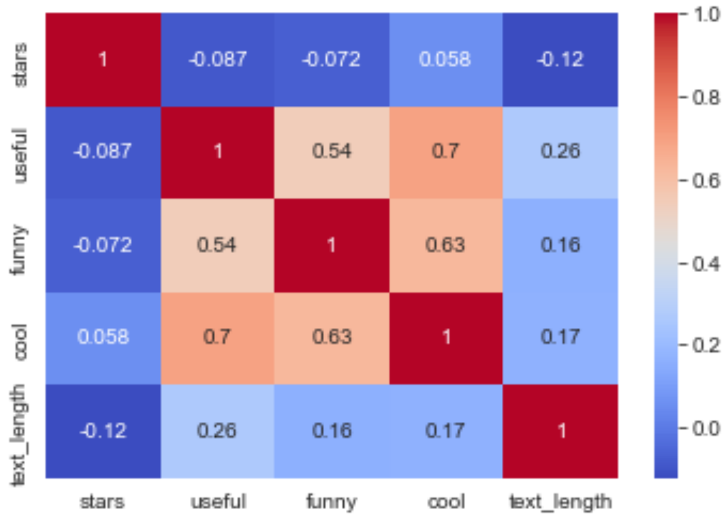
In [21]:
```python
# box plot of text length for each star rating
# shows the skeweness as is shown from the histogram

sns.boxplot(x="stars", y='text_length', data = review_df, palette = "r
```

Out[21]:
```
<AxesSubplot:xlabel='stars', ylabel='text_length'>
```



In [22]:
```
sns.heatmap(review_df.corr(), cmap = 'coolwarm', annot = True)
```

Out[22]: <AxesSubplot:>



Results

- In general, as we can see from the hitmap visualization above, the features included in the review_df don't seem to be strongly coorelated with each other (neither negatively, nor positively).
- From the heatmap, the most coorelated seems to be the text_length at -0.12, which is still an insignificant negative coorelation to star ratings and from the KS test we saw that they have different distributions.
- Looking at the Facet Grid, text_length distribution is very similar accross ratings, but we can observe that most ratings had a text review of lesh than 500 characters and all have a right skeweness were there are less and less reviews from 1000-3000 characters.
- This right skeweness is also reflected in the boxplots as well, were a good amount of observations are more than 1.5 standard deviations away from the text_length mean for each rating.

b) **[10pt]** Understanding Distributions of Data Subsets:

We want to understand what is the distribution of `stars` across businesses that are in the state of Nevada (NV) and are restaurants that allow take out (`attributes.RestaurantsTakeOut`). As in the earlier question, generate the appropriate visualization using `pandas` and `matplotlib` , justify your choice of visualization, and interpret the visualization result in words.

The distribution of stars across take-out restaurants in Nevada

Since we want to find the distribution of stars for businesses, I'll have to work with the "business_df" dataframe.

1. I create a boolean mask to filter our dataframe results for businesses in Nevada --> business_df["state"]=='NV'
2. Filled the NaN values in 'attributes.RestaurantsTakeOut'with False as a string since all existing values are str
3. I filter for reasturants with take out option --> business_df['attributes.RestaurantsTakeOut']=='True'
4. I combined both boolean masks under a "mask" variable and used it in the business_df
5. Visualize the star distribution under the applied boolean mask

The reason for choosing a histogram over other charts, is because histograms are great to visualize distributions and frequencies. Here, I could use a bin = 5 to get 5 buckets of star distributions, but chose to go with 15 to get a more granular representation of the star distributions (they are float numbers between 1-5).

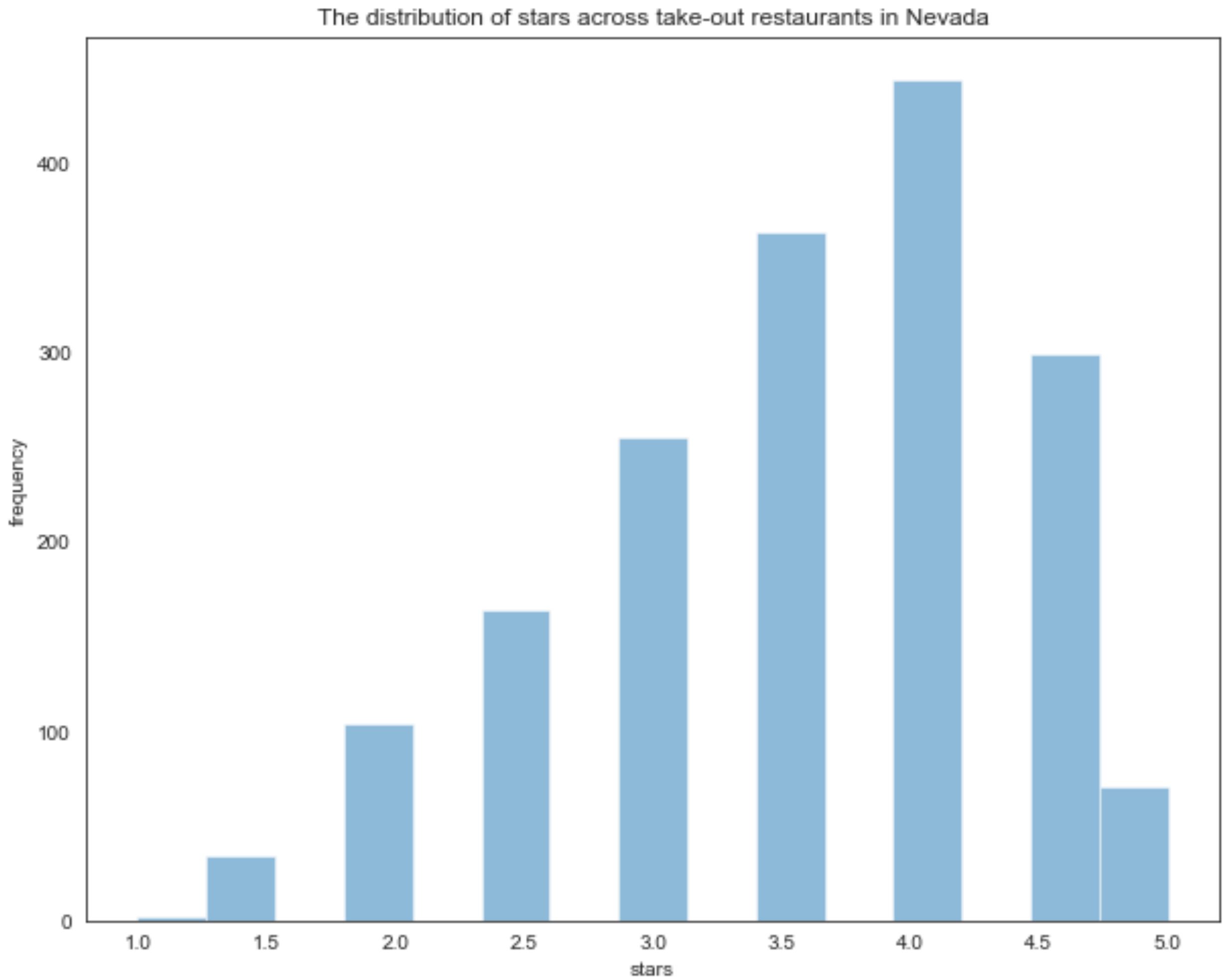
Results

As we can see from the visualization below, the star reviews follow a somewhat left-skewed normal distribution. This means that most of the star reviews in take-out restaurants in Nevada are above average (over 3.0). The majority of the reviewers have given stars that range from 3-4.7 and we see less from 1-2.5 and 4.7-5. Overall, from the star distributions, we can say that take-out restaurants in Nevada are very well recieved by the reviewers.

In [23]:
```
business_df['attributes.RestaurantsTakeOut'].fillna("False", inplace =
```

In [24]:
```
mask = (business_df['state']=='NV') & (business_df['attributes.Restaur
business_df[mask]["stars"].hist(bins=15, alpha=0.5, grid=False, figsiz
plt.title("The distribution of stars across take-out restaurants in Ne
plt.xlabel("stars")
plt.ylabel("frequency")
```

Out[24]: Text(0, 0.5, 'frequency')



Question 4: Working with Document stores (20pt)

Using Firestore modules you have seen in class, create a new document store for your yelp data. Create a collection for each of the three `business` , `review` , and `user` collections. Insert the first record of `user_df` , `business_df` , and `review_df` as a document in their respective Firestore collection.

In [28]:
```python
# start the connection to the firebase database

cred = credentials.Certificate('data-eng-c1173-firebase-adminsdk-bg3qi
firebase_admin.initialize_app(cred)
```

Out[28]: <firebase_admin.App at 0x7fcd79836340>

In [ ]:
```python
# METHOD 1:

db = firestore.client()

review_ref = db.collection(u'review')
user_ref = db.collection(u'user')
business_ref = db.collection(u'business')

doc1 = business_ref.get(u'KgXg6v9LXhaYkrpfPgjQ')
doc2 = review_ref.get(u'LgVvqJOpXJjNxaRvP7Ib')
doc3 = business_ref.get(u'vFfHRCEqL2QqY4Zni7d8')

print('doc1 %s' % (doc1.to_dict(),))
print('doc2 %s' % (doc2.to_dict(),))
print('doc3 %s' % (doc3.to_dict(),))


# METHOD 2:

# ref1 = db.reference("/business/KgXg6v9LXhaYkrpfPgjQ")
# ref2 = db.reference("/user/vFfHRCEqL2QqY4Zni7d8")
# ref3 = db.reference("/review/LgVvqJOpXJjNxaRvP7Ib")

# business_ref = ref1.get()
# user_ref = ref2.get()
# review_ref = ref3.get()

# print(business_ref)
# print(user_ref)
# print(review_ref)
```

In [ ]:

```
# EXAMPLE TO LOAD DATA TO FIREBASE DATABASE

# # pip install firebase_admin

# import firebase_admin
# from firebase_admin import db

# cred_obj = firebase_admin.credentials.Certificate('data-eng-c1173-fi
# if not firebase_admin._apps:
#     default_app = firebase_admin.initialize_app(cred, {
#         'databaseURL': 'https://console.firebase.google.com/project/
#     })

# ref = db.reference("/business/")

# with open("business.json", "r") as f:
#     file_contents = json.load(f)
# ref.set(file_contents)
```