Data Chefs

Cuisine Classification and Recipe Summarization Final Report

Group 4

Bhavya Melana, Diksha Gulati, Harshdeep Mac, Utkarsh Kekre

Georgia State University, J Mack Robinson College of Business

INDEX

Introduction

Cooking is one of the most central exercises in one's life. It isn't just associated with the delight of eating yet in addition profoundly influences different parts of human life, for example, wellbeing, dietary, culinary craftsmanship, diversion, human correspondence, etc.

Henceforth, the number of websites uploading different kinds of recipes and their users are increasing rapidly, day by day. According to DMR, in the year 2020, Allrecipes have 60.3 million unique visitors and 370 million recipes saved on the website. Allrecipes is available in 24 countries in 12 different languages. Also, one of other famous website Epicurious have 9.50 million users with around 73.5% traffic from the United States. These statistics shows the interest of the users.

All our team members are food enthusiasts which inspired us to come up with an idea for such food websites, to make life of a user easy. User experience has become a preordained part of any company. Hence, saving user's time and providing them a smooth experience yields great value to retain the customers and building trust.

As a user food website user, we thought of few challenges we face, they are as follows:

- Not all the recipes uploaded by users have their corresponding cuisines labeled.
- Not all the recipes have a meaningful title.

When solution is built, users will have the following advantages:

- Cuisine classification based on the ingredients of the recipe.
- Assigning a meaningful title to the recipes with irrelevant titles, using the directions mentioned in the recipes on the websites.

The Solution

We aim to enhance user experience and save their time by improvising their journey on the website in use. Hence, ensuring customer retention by providing a positive experience, reducing the burden, gaining credibility, and giving them productive screen time.

Technology

To achieve our solution, we have jotted down few important concepts which we are going to use to make it possible:

- Stemming
- Term Document Matrix
- Classification Algorithm
- Sequence2Sequence Encoder-Decoder
- Natural Language Processing (NLP)

To implement the above-mentioned concepts, a huge amount of data will be trained on machine learning models and furthermore, tested to check the accuracy to provide a valuable result. Models to be implemented:

- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTMs)

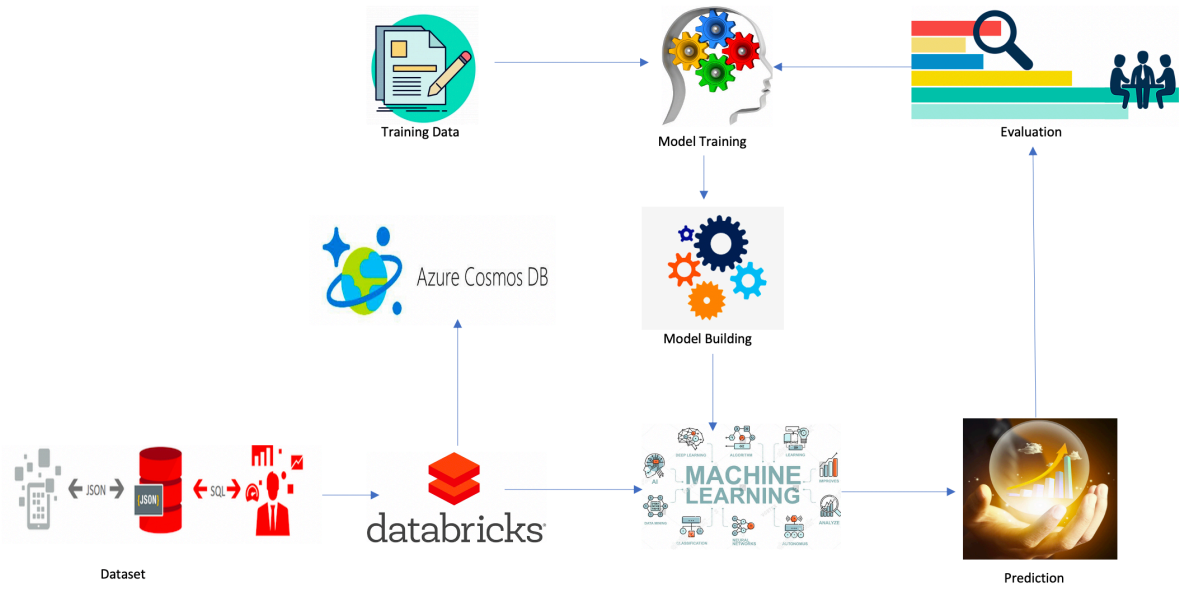
We are going to use Microsoft Azure platform, to carry out the above enumerated tasks, as we need an effective portal. Some of the requirements for the proposed solution are:

- Data storage
- Executing the machine learning models
- Azure Databricks

We shall discuss all the above-mentioned tasks further in detail as part of this report, along with results and the challenges that were faced.

Proposed Architecture

The graphical representation below illustrates the flow of data in a process based on the inputs and outputs. All through our task there are different stages and procedures through which the data will stream and accordingly it is basic that there ought to be a defined flow of data to get the desired result.



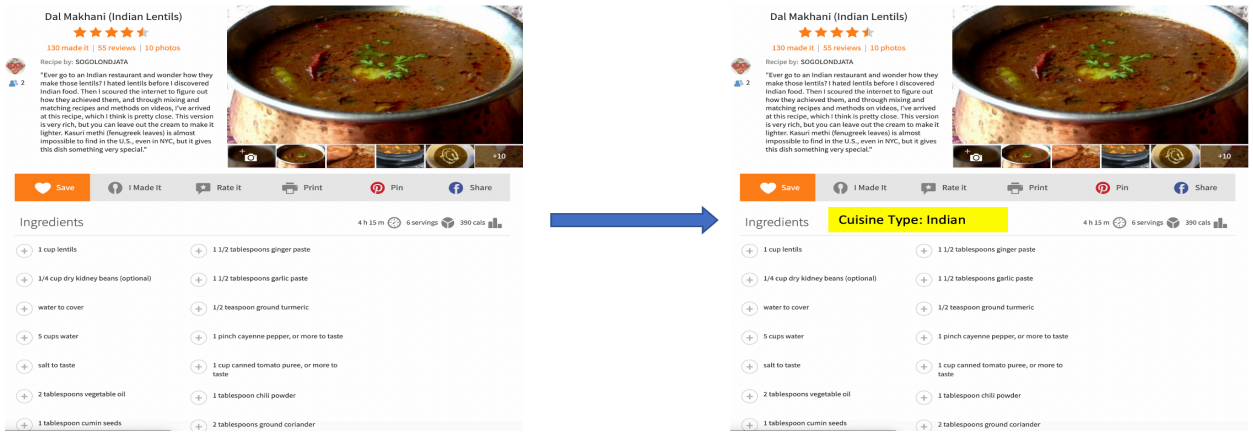
High-Level Architecture

Along with above-mentioned flow of data, we have also covered the platforms and technologies used at distinctive stages to process the data. This architecture defines the execution of tasks that is required for the application to run smoothly and provide desired results.

Business Use Case

Cuisine Classification:

With Cuisine Classification, we intend to provide a solution where-in all the recipes have the appropriate cuisine marked in the recipe, this will help in the smooth filtering process, based on cuisine type. This solution will target in improvising customer's journey on the application, and hence result in better user experience. As you can see below, there are 2 screenshots, the first screenshot (Fig. 1) displays the recipe without the tagged cuisine. Based on the ingredients stated in the recipe, we are going to recommend the cuisine type and designate it with the recipe, as shown in Fig. 2. The mentioned solution will also be a recommendation system used in web searches, and when user jot down the ingredients, it will recommend the cuisine type.



Recipe Summarization:

Here, we aim to achieve a solution where-in all the recipes have meaningful titles, using the directions mentioned in the recipe. As you can see below, there are 2 screenshots, the first screenshot (Fig. 1) displays the recipe with an irrelevant title causing filtering and sorting of recipes difficult for the users, our objective is to provide a meaningful title to before-mentioned recipes to enhance the user experience, as presented in the second screenshot (Fig. 2).

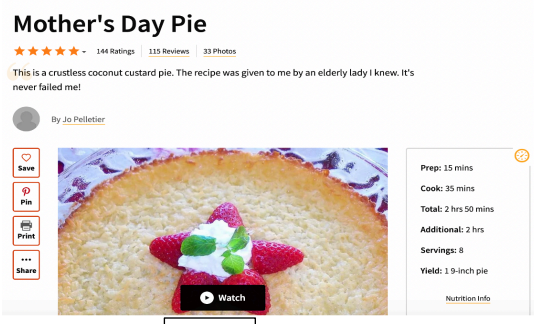


Fig. 1

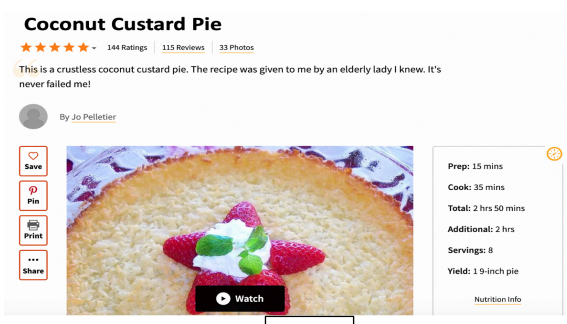


Fig. 2

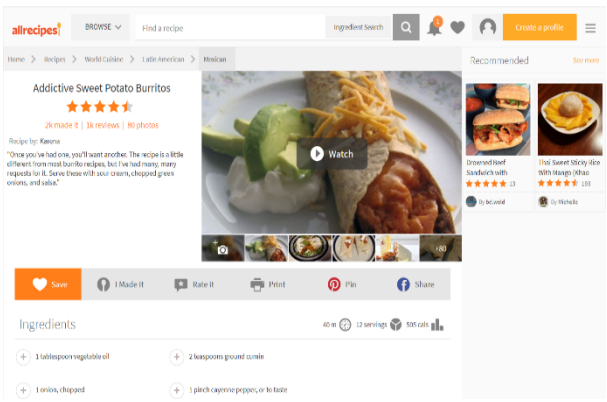
Data Sources

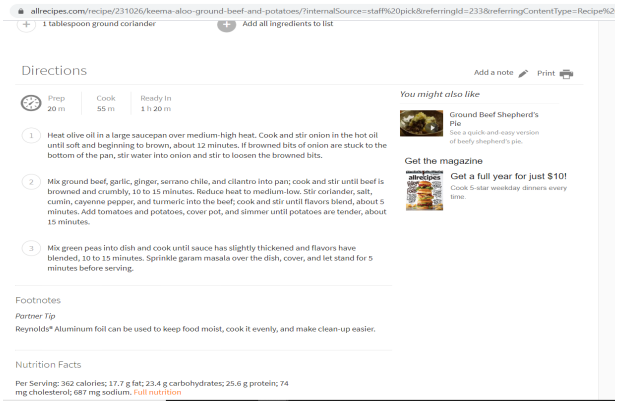
A data source may be the initial location where data is born or where physical information is first digitized, however even the most refined data may serve as a source, as long as another process access utilizes it. We scraped the data from external data sources in html format which is the text file format. The extracted data contains Ingredients, directions and Nutritional Fact.

There are 2 data sources, basically food websites:

1. All Recipes

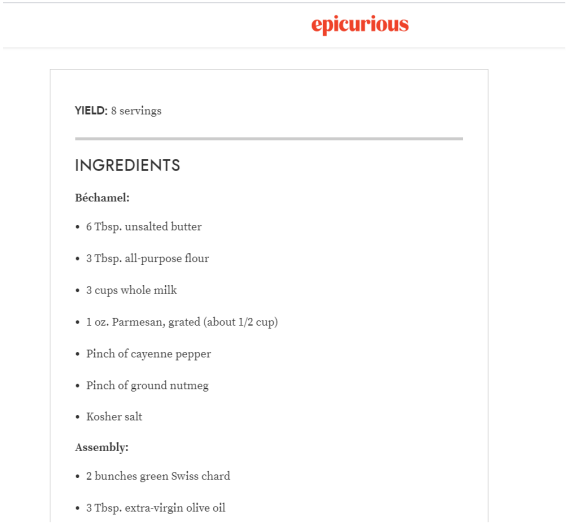
It is a food focused online social networking service. Recipes are categorized by season, type (such as appetizer or dessert), and ingredients. The website allows users to search recipes and include specification in their search (such as type of meal, nutrition, key ingredients, and time needed to prepare the dish).

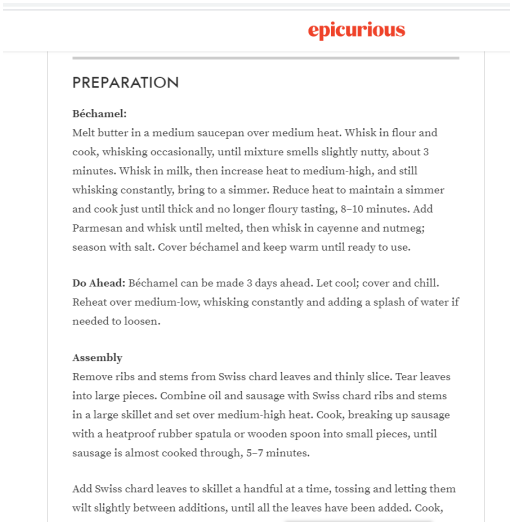




2. Epicurious

It is a digital brand for consumers interested in food and cooking.





Data Scraping

Instead of taking a built-in data set to solve our business problems, we have come up with a solution to build our own dataset by scrapping the food websites. Two scrapper scripts have been written in python using “BeautifulSoup” package to scrape the websites “Epicurious” and “AllRecipes” to fetch ingredients, instructions, and nutrition from the webpage and automatically stores them in azure cosmos database using Azure SQL API. The scrapper codes are shown below.

Epicurious Scrapper Code

```
for cuisine in all_cuisines:
    total_pages = int(BeautifulSoup(requests.get(all_cuisines[cuisine]).text, 'lxml').find('nav', class_="common-pagination")['data-total-pages'])
    for page in range(total_pages):
        current_page = BeautifulSoup(requests.get(f'https://www.epicurious.com/search/?cuisine={cuisine}&content=recipe&page={page+1}').text, 'lxml')
        recipe_index = 0
        while recipe_index <= 17:
            link = current_page.find('article', {"data-index":str(recipe_index)})
            if link != None:
                url = "https://www.epicurious.com" + current_page.find('article', {"data-index":str(recipe_index)}).find('a')['href']
                try:
                    current_recipe = BeautifulSoup(requests.get(url).text, 'lxml')
                    recipe_title = current_recipe.find('h1', {"itemprop":"name"}).text

                    # Fetch the ingredients from the recipe
                    ingredients = []
                    for ingredient in current_recipe.find_all('li', {"class":"ingredient", "itemprop":"ingredients"}):
                        ingredients.append(ingredient.text)

                    # Clean and Parse the ingredients
                    #ingredients = Ing_Cleaner(ingredients)

                    # Putting the ingredients in ingredient corpus
                    for i in ingredients:
                        ingredient_corpus.add(i)

                    ingredients_data_set = {"title":recipe_title, "cuisine": cuisine, "ingredients": ingredients}
                    ingredients_data_set = json.dumps(ingredients_data_set)

                    # Fetch the instructions from the recipe
                    instructions = []
                    for inst in current_recipe.find_all('li', class_="preparation-step"):
                        instructions.append(inst.text)
                    instructions_data_set = {"title":recipe_title, "cuisine": cuisine, "instructions": instructions}
                    instructions_data_set = json.dumps(instructions_data_set)

                    #Fetch the nutritions from the recipe
                    nutritions = {"title":recipe_title, "cuisine": cuisine}
                    nutri_label = current_recipe.find_all('span', class_="nutri-label")
                    nutri_data = current_recipe.find_all('span', class_="nutri-data")
                    length = len(nutri_label)
                    for l in range(length):
                        nutritions[nutri_label[l].text] = nutri_data[l].text
                    nutritions_data_set = json.dumps(nutritions)

                    upsert_document(client, database_name, "Ingredients", ingredients_data_set)
                    upsert_document(client, database_name, "Instructions", instructions_data_set)
                    upsert_document(client, database_name, "Nutritions", nutritions_data_set)
                except:
                    pass
                recipe_index += 1
                dummy += 1
            else:
```

AllRecipes Scrapper Code:

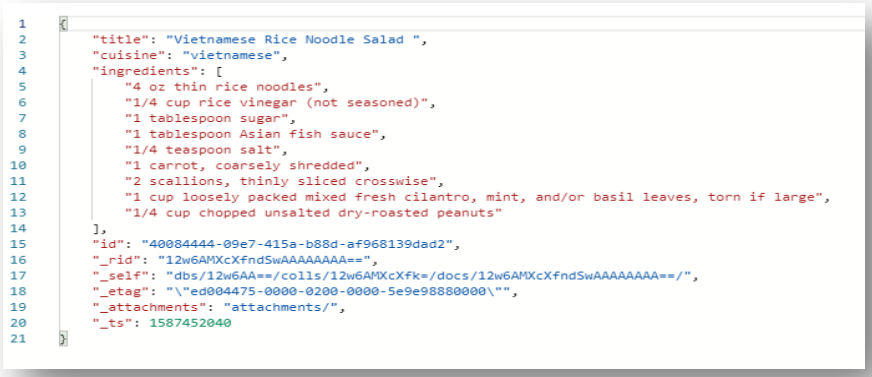
```
for cuisine in world_cuisines.find_all('a', class_="grid-col--subnav"):  # This loop is to iterate through cuisines
    cuisine_url = cuisine['href']  # url of the current cuisine
    cuisine_page = BeautifulSoup(requests.get(cuisine_url).text, "lxml")
    cuisine_name = str.lower(cuisine.find('span').text).replace(" recipes", "")  # name of the current cuisine
    all_cuisines[cuisine_name] = cuisine_url  # Use this dictionary to create the labels dictionary
    for sub_cuisine in cuisine_page.find_all("a", class_="grid-col--subnav"):  # This loop is to iterate thorugh sub cuisines
        sub_cuisine_url = sub_cuisine['href']  # url of the sub-cuisine
        sub_cuisine_page = BeautifulSoup(requests.get(sub_cuisine_url).text, "lxml")
        page_number = 0
        while page_number <= 10000:  # This loop is to iterate through pages
            current_page_url = sub_cuisine_url + f'?
internalSource=hub%20nav&referringId=728&referringContentType=Recipe%20Hub&linkName=hub%20nav%20daughter&clickId=hub%20nav%202&page={page_number+1}'
            page_number += 1
            current_page = BeautifulSoup(requests.get(current_page_url).text, "lxml")
            if current_page.find("title").text == "Allrecipes - File Not Found":
                break
            for recipe in current_page.find_all("article", class_="fixed-recipe-card"):  # This loop is to iterate through all recipies in a page
                try:
                    recipe_url = recipe.find("a", class_="fixed-recipe-card__title-link")['href']
                    recipe_page = BeautifulSoup(requests.get(recipe_url).text, "lxml")
                    recipe_name = recipe_page.find("h1").text

                    ingredients = []
                    for ing in recipe_page.find_all("span", {"class": ["recipe-ingred_txt added", "ingredients-item-name"]}):
                        ingredients.append(ing.text)
                    ingredient_data_set = {"source": "AllRecipes", "cuisine": cuisine_name, "title": recipe_name, "ingredients": ingredients}
                    ingredient_data_set = json.dumps(ingredient_data_set)

                    instructions = []
                    for inst in recipe_page.find_all('li', {"class": ["step", "subcontainer instructions-section-item"]}):
                        instructions.append(inst.text)

                    instruction_data_set = {"source": "AllRecipes", "cuisine": cuisine_name, "title": recipe_name, "instructions": instructions}
                    instruction_data_set = json.dumps(instruction_data_set)

                    upsert_document(client, database_name, "Ingredients", ingredient_data_set)
                    upsert_document(client, database_name, "Instructions", instruction_data_set)
                except:
                    pass
        print(cuisine, " is complete")
```

Data Storage

The meaningful information from the data needs to be derived for the data to be useful. Unstructured data such as text is major input in this project and since the volume is high, so we stored the data in Azure Cosmos DB.

We have primarily 3 types of data i.e. ingredients, directions and nutritional values of all recipes which will be stored in their respective containers. As we have scrapped the data from 2 sources, the scrapping script for the 2 sources is attached below. The raw form of data will be stored in containers named as “Ingredients”, “Instructions” and “Nutritions”. The data in these containers are stored in form of documents and have a JSON schema shown below.

Ingredients JSON Schema:




```
{
    "title": "Vietnamese Rice Noodle Salad ",
    "cuisine": "vietnamese",
    "ingredients": [
        "4 oz thin rice noodles",
        "1/4 cup rice vinegar (not seasoned)",
        "1 tablespoon sugar",
        "1 tablespoon Asian fish sauce",
        "1/4 teaspoon salt",
        "1 carrot, coarsely shredded",
        "2 scallions, thinly sliced crosswise",
        "1 cup loosely packed mixed fresh cilantro, mint, and/or basil leaves, torn if large",
        "1/4 cup chopped unsalted dry-roasted peanuts"
    ],
    "id": "40084444-09e7-415a-b88d-af968139dad2",
    "_rid": "12w6AMXcXfndSwAAAAAAAA==",
    "_self": "dbs/12w6AA==/colls/12w6AMXcXfk=/docs/12w6AMXcXfndSwAAAAAAAA==/",
    "_etag": "\"ed004475-0000-0200-0000-5e9e98880000\"",
    "_attachments": "attachments/",
    "_ts": 1587452040
}
```


Instructions JSON Schema:


```
{
    "source": "AllRecipes",
    "cuisine": "mexican",
    "title": "Tofu Tacos I",
    "instructions": [
        "\nOver a medium heat fry the tofu, oil, garlic and onion in a large skillet for 5 minutes. Add the chili powder, paprika, cayenne,
        "\nSpoon the tofu mixture into taco shells. Top the mixture with lettuce, tomatoes, avocado, Cheddar cheese and salsa.\n
        "\n\n"
    ],
    "id": "3ded8b57-9ae0-4471-b848-50350b8d5cc2",
    "_rid": "12w6AMkGx8xKAAAAAAAAAA==",
    "_self": "dbs/12w6AA==/colls/12w6AMkGx8w=/docs/12w6AMkGx8xKAAAAAAAAAA==/",
    "_etag": "\"e600ce60-0000-0200-0000-5e9e61080000\"",
    "_attachments": "attachments/",
    "_ts": 1587437832
}
```


Nutrition JSON Schema:


```
{
    "title": "Efo Riro With Kale and Whitefish ",
    "cuisine": "african",
    "Calories": "447",
    "Carbohydrates": "11 g(4%)",
    "Fat": "37 g(57%)",
    "Protein": "20 g(40%)",
    "Saturated Fat": "3 g(15%)",
    "Sodium": "129 mg(5%)",
    "Polyunsaturated Fat": "6 g",
    "Fiber": "3 g(14%)",
    "Monounsaturated Fat": "25 g",
    "Cholesterol": "49 mg(16%)",
    "id": "dd4d39a6-2db0-4d0c-9188-59a546a8674b",
    "_rid": "12w6ANdpSMIBAAAAAAAAAA==",
    "_self": "dbs/12w6AA==/colls/12w6ANdpSMI=/docs/12w6ANdpSMIBAAAAAAAAAA==/",
    "_etag": "\"e6008c68-0000-0200-0000-5e9e613e0000\"",
    "_attachments": "attachments/",
    "_ts": 1587437886
}
```

Azure Cosmos DB

Cosmos Db is a database service that is globally distributed. It permits you to deal with your information regardless of whether you keep them in data centers that are dissipated all through the world. It gives the tools you have to scale both worldwide distribution design and computational assets, and these devices are given by Microsoft Azure. The database is used for document, key value, relational, and graph models. We have stored our data in the document format. A document database is a kind of nonrelational database that is intended to store and query data as JSON-like documents. Document databases make it simpler for engineers to store and query data in a database by utilizing a similar document model configuration they use in their application code.

Using Cosmos DB has following advantages:

1. **Global Distribution**

   It is available in every location where Azure is available by default. So, you can setup instances of your Cosmos DB at any location that you want simply by activating the desired location from the Azure portal. This will ensure that the data is replicated and available for the users in the region with guaranteed low latency. Additionally, it also provides automatic and manual failover that enables high availability and disaster recovery.

2. **Performance**

   It ensures accessing data with faster response time. It also provides guaranteed throughput based on the provisioned output capacity.

3. **Pricing**

   It is dependent on the required throughput and the storage necessary for your data.

4. **Multi-Model and Multi APIs**

   It enables to create containers that can store data in Key-Value, Columnar, Document or Graph data stores. Along with this, it also provides users the flexibility to choose different APIs to access the data such as:
   - SQL API or MongoDB API (for Document databases)
   - Table API (Kay-Value databases)
   - Cassandra API (Columnar databases)
   - Gremlin API (Graph databases)

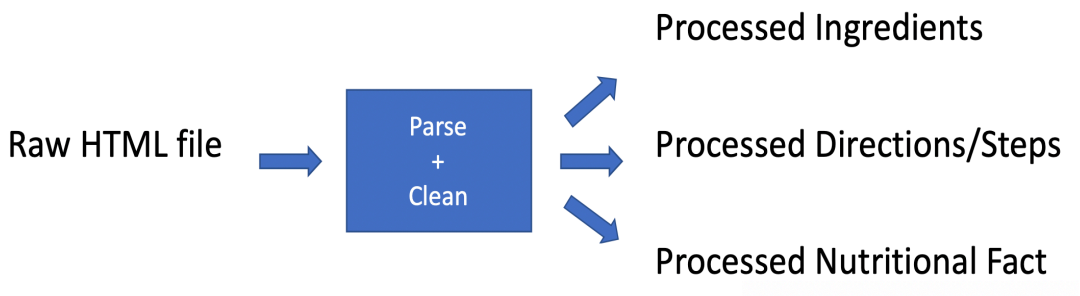
5. **Defined Consistency Levels**

   There are 5 consistency levels that strikes a balance between latency, throughput, and availability.

- **Strong Consistency**: It ensures consistency across all nodes, in all regions, but this comes at the cost of overall performance.
- **Bounded Staleness Consistency**: It provides a mean to set the level of freshness of data.
- **Session Consistency**: It ensures that there is no dirty read for the writer but it's possible to have dirty reads for other users. This is the default consistency level.
- **Consistent Prefix**: It ensures that the read data has been updated to all replicas.
- **Eventual Consistency:** It provides no guarantees on the freshness of the data or on the order. However, this provides the fastest performance.

Data Cleaning

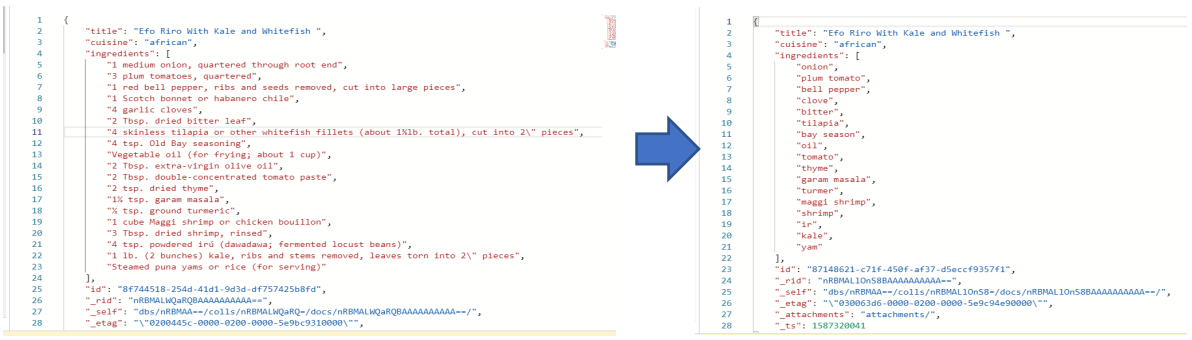
Process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.



After we store our data in the respective containers, we need to clean the data so that it can be used in modeling out the proposed solution. For this we need to clean our ingredients to get only the root ingredient out of the whole line i.e.

"**2 cups drained canned diced tomatoes (from 28-ounce can)"** ➡ "**tomato**".

To obtain this, we have written a script which cleans the ingredients as shown above, cleans the directions to remove any special characters and same for nutritional values. The script includes functions like Stemmer, stop word removal, Part of speech tagger, quantity removal, measurements removal, color removal, additional words removal and regex for special chars and number removal. Below is an example of our cleaning.



Now as the schema for both the sources will be uniform that is the output of the cleaning script will have the same schema for both the sources, the data sources will be automatically stitched together into the respective containers.

Data Science Solution

Different data science solution has been chosen to solve business problems.

Problem 1: Cuisine Classification

Solution: The complete step by step solution to solve the problem of cuisine classification is shown below:

1. **Ingredient Corpus:** A dictionary to save all the ingredients to be later used in creating a final matrix for the classification model.
2. **Label Dictionary:** A dictionary to save all the cuisines to be later used in creating labels for the data matrix.
3. **Term Document Matrix:** Written a python code to covert ingredients into a term document matrix (a binary matrix) to be later used in training and testing of classification model.
4. **Classification Algorithm:** Plan is to implement an auto-ML pipeline in azure to automatically train all the classification models mainly neural-network, support vector machine, logistic regression, random forest, decision tree etc. Auto-ML can pick the best model based on evaluation parameter.
5. **Evaluation:** Confusion matrix works well for most of the classification problems

Sample Outputs:

Example 1:

• Predicted Cuisine: African or Not African (It could be any other cuisine)

• Original Cuisine: African

• Ingredients: onion, plum tomato, bell pepper, clove, butter, tilapia, bay season, oil, tomato, thyme, garam masala, turmeric, Maggi shrimp, shrimp, kale, yam

Problem 2: Recipe Summarization

Solution: The problem that we have in our hand is an abstractive summarization problem. We are using sequence2sequence encoder-decoder which is an advanced deep learning model. Model is a combination of multiple LSTMs. The complete model is divided into encoder, decoder and attention mechanism.

- **Encoder:** Bi-directional LSTM layer that extracts information from the original text. The bidirectional LSTM reads one word at a time and since it is a LSTM, it updates its hidden state based on the current word and the words it has read before.
- **Decoder:** Uni-directional LSTM layer that generates summaries one word at a time. The decoder LSTM starts working once it gets the signal than the full source text has been read. It uses information from the encoder as well as what is has written before to create the probability distribution over the next word.
- **Attention Mechanism:** Through attention mechanism, the decoder can access the intermediate hidden states in the encoder and use all that information to decide which word is next.

Sample Outputs:

Example 1:

- **Generated:** Chicken cake
- **Original:** Chicken French – Rochester, NY Style
- **Recipe:** all-purpose flour; salt; eggs; white sugar; grated parmesan cheese; olive oil; skinless; butter ; minced garlic ; dry sherry ; lemon juice ; low sodium chicken base ; ;Mix together the flour , salt , and pepper in a shallow bowl . In another bowl, whisk beaten eggs, sugar, and Parmesan cheese until the mixture is thoroughly blended and the sugar has dissolved. Heat olive oil in a large skillet over medium heat until the oil shimmers. Dip the chicken breasts into the flour mixture, then into the egg mixture, and gently lay them into the skillet. Pan-fry the chicken breasts until golden brown and no longer pink in the middle, about 6 minutes on each side. Remove from the skillet and set aside. In the same skillet over medium-low heat, melt the butter, and stir in garlic, sherry, lemon juice, and chicken base.

Recipe Summarization

Recipe summarization refers to generating recipe titles from summarizing the recipe instructions and ingredients which relates more to the recipe. This concept is known as text summarization which uses deep learning capabilities to train a recurrent neural network model using multiple LSTM layers with combination of dense and dropout layers, to learn the language context and tries to summarize the recipe.

1. Data Pre-Processing

As we have the cleaned ingredients and instructions in their respective containers stored in Cosmos DB, we need to establish a DB connection with Azure Databricks where all the preprocessing will take place and the model will be trained.

To do this we use the Azure-CosmosDB-Spark connector provided by Azure to do the same. Installing this on the cluster enables us to create a connection link between Azure Databricks and the container where data is stored in CosmosDB.

In the connection configuration, we need to specify the DB endpoint, masterkey, Database name, DB region where it is hosted, container name and the query to fetch the documents. Then this connection config is passed as an argument to the **spark.read.format** function to create a Spark dataframe of the fetched documents from cosmos DB. Repeating this same process for both the containers that is cleaned ingredients and instructions. To play with the data in python, we created a Pandas dataframe from spark dataframe.

Now we clean the instruction which are currently a list, making it a single string by concatenating the list and removing punctuation and other advertisement and steps which are irrelevant to our processing. Same cleaning is done for recipe titles.

We need to join the two Pandas dataframes based on title as it is the common key between 2 DF's. So, left joining the 2 DF's, we obtain the final DF. We create a custom column which has all ingredients first

and then the instructions following that. This is needed to make sure our Vocab contains all possible words which should be present in our Superset of words which will be explained in coming steps of this paper. Dropping nulls and showing the results of our final DF.


TextSummarization (Python)

Detached | File | Edit | View: Code | Permissions | Run All | Clear | Schedule | Comments

Cmd 16

Final DF to model

```
1 final_df
```

Out[344]:

	cuisine_x	Clean_Inst	Clean_titles	joined_ing	concated_data
0	mexican	combine the lime juice and onion slice in a bo...	fresh sangrita	lime paper onion orang style valentina salt	lime paper onion orang style valentina salt co...
3	mexican	combine the tomatillos serrano peppers garlic ...	carne en su jugo meat in its juices	tomatillo chile pepper garlic water bacon flan...	tomatillo chile pepper garlic water bacon flan...
5	mexican	combine the tomatillos serrano peppers garlic ...	carne en su jugo meat in its juices	bacon tomatillo chile pepper garlic water flan...	bacon tomatillo chile pepper garlic water flan...
6	mexican	in a saucepan combine chicken breast with chic...	authentic enchiladas verdes	bone chicken breast chicken broth onion garlic...	bone chicken breast chicken broth onion garlic...

Now selecting only headlines that is “Clean_titles” and mixture of ingredients and instruction which is “concated_data” which will be part of our vocab for our model.

Next, we create the vocab and vocab_count for our model which is all the words in our corpus and its frequency which is vocab_count as a dictionary. Now we create a custom method which takes the vocab as an argument and returns 2 dictionary one being indexes with respect to words and other being vice-versa i.e. words with respect to indexes.

Next, we initialize the **GloVe** dictionary.

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

There are various pre-trained word vectors in GloVe mainly:

- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors).
- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors)
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors).
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors).

In this project, we have used the Wikipedia + Gigaword format of GloVe dictionary with 100 dimensional vectors. This dictionary is a TXT file of 330MB which needs to be installed on our cluster for processing.

To install this dictionary, we use the data upload feature of Databricks which uploads the file in FileStore location under tables container. Our code can directly reference this location to read this text file and can be processed.

Next, we read this dictionary in our cluster and extract all words present in it with indexes and the same indexes with the weights of the respective words in another variable in the “get_glove” method.

We need to also create embeddings for our model which needs to be added, so we create in the next step.

First initialize our vocab_size to our total words in vocab i.e. 21374 and our GloVe dimension as embedding dimension to 100. We create a matrix of the same shape as our vocab size and embedding dimension i.e. 21374 X 100 and assign random values in the matrix values.

Next, we copy the GloVe weights of each word found in the corpus to the embedding. With this we see that around 14K has been found in GloVe out of our corpus of 21K which is nearly 60%.

Now, for the remaining words, we try to map the GloVe vectors by finding the similar words using Cosine-Similarity to map these into our embedding. For ex:

Next, we create dense vectors for our X and Y that is X for descriptions and Y for titles. To do this, we just take the indexes of words in titles and description and map it with its index for its place.

To split our data into train and test, we import the **Sklearn package's test_train_split** method to create 2 DF's of 80/20%.

Viewing our data randomly:

2. Modelling our RNN with LSTM's

Now, importing all the required libraries to create our RNN model with multiple LSTM layers. We will be using activation functions and optimizers in our model to train a good performing model.

TextSummarization (Python)

Detached | File | Edit | View: Code | Permissions | Run All | Clear

Cmd 37

Import required Model libraries

```
1  from keras.models import Sequential
2  from keras.layers.core import Dense, Activation, Dropout
3  from keras.layers.wrappers import TimeDistributed
4  from keras.layers.recurrent import LSTM
5  from keras.layers.embeddings import Embedding
6  from keras.layers.core import Lambda
7  import keras.backend as K
8  import numpy as np
9
10 seed = 42
11 optimizer = 'adam'
12 p_W, p_U, p_dense, p_emb, weight_decay = 0, 0, 0, 0, 0
13 regularizer = None
14
```

Command took 0.02 seconds -- by harshmac.786@gmail.com at 4/25/2020, 1:18:40 AM on DataChef_Upgrade

Now, we create our model which will consider the embedding, vocab, corpus, X and Y and try to predict a summary for our recipe aka instructions. To do this, we initialize a Sequential model which will enable us to add multiple layers into our model same way as we append variables into a list.

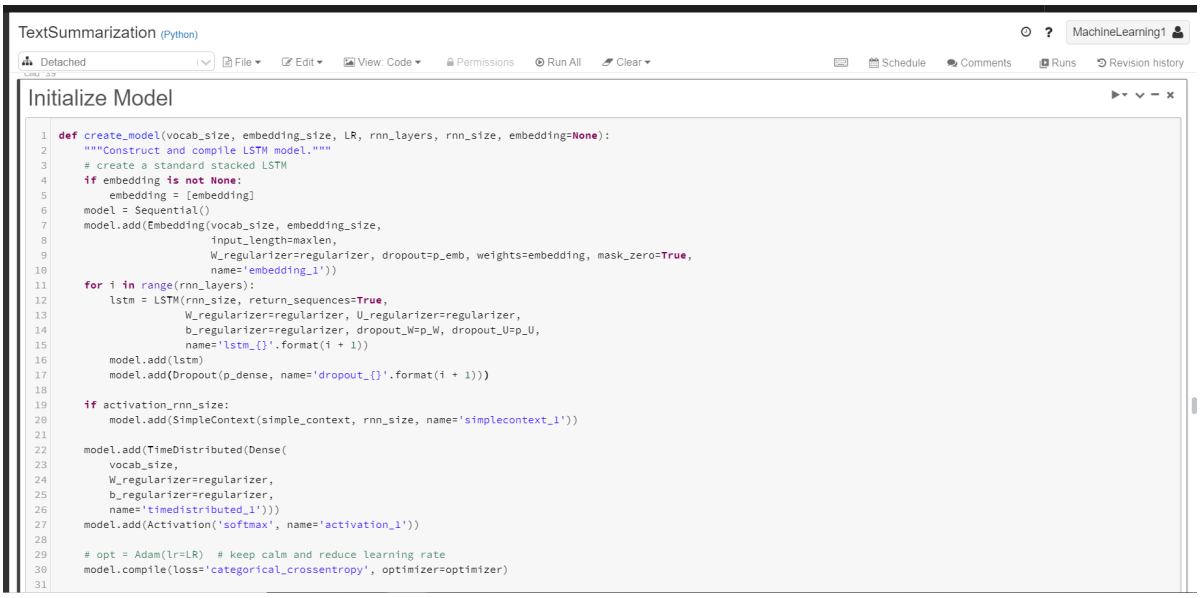
Firstly, we add our embedding layer into our model. The embedding layer is used to create word vectors for incoming words. It sits between the input and the LSTM layer, i.e. the output of the Embedding layer is the input to the LSTM layer. Our embedding should be of same dimension as the vocab size and embedding size.

Secondly, we add our LSTM layer with dropout layer into our model thrice. For our 1 layer of LSTM, we have 512 neurons mapped into it. Following it we add a dropout layer to prevent the LSTM to overfit during training.

Next, we add a Time distributed dense layer, which is actually an important component in our LSTM because our model tries to predict multiple words for our summarization, so as LSTM generates more than 1 output for our final output, our RNN model needs to wait until the LSTM has processed all the possible words and predicted its output for the next layer. This is where Time distributed layer comes into the picture as it waits for the LSTM to complete its processing to feed the consolidated output of LSTM into the next layer that is the dense layer. The dense layer is made up of neurons equal to the number of vocab words that are present in our corpus which is around 21K so that it can activate any word which has a higher probability for output.

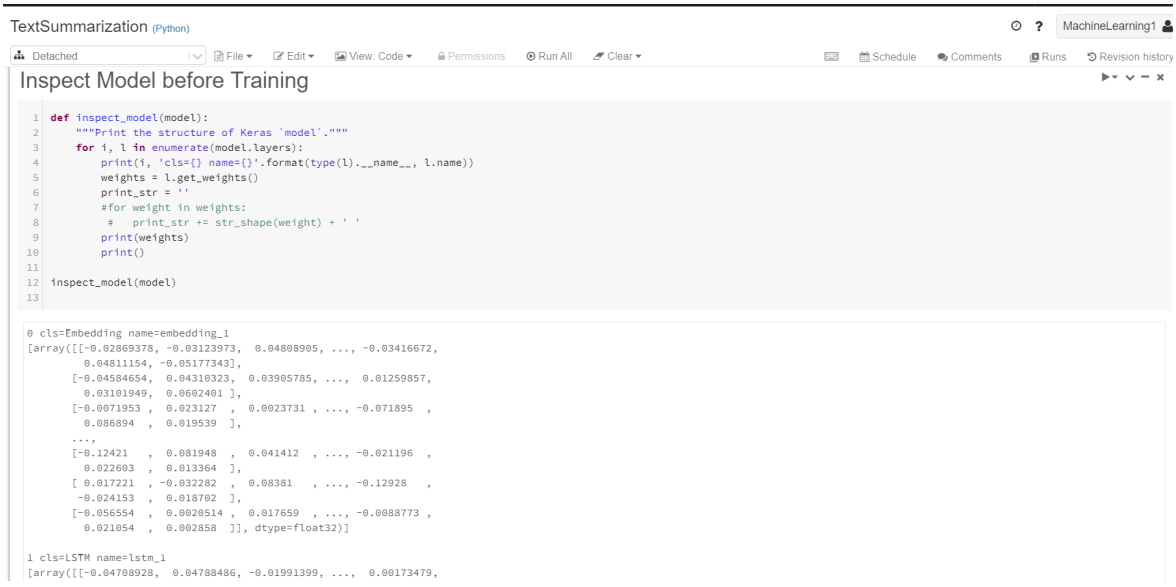
Finally, we add the activation function in our model which is "SoftMax" to predict the output words.

And to compare the correctness of the predicted words, we add a compiler layer with loss function as “categorical_crossentropy” and an optimizer as “Adam”.



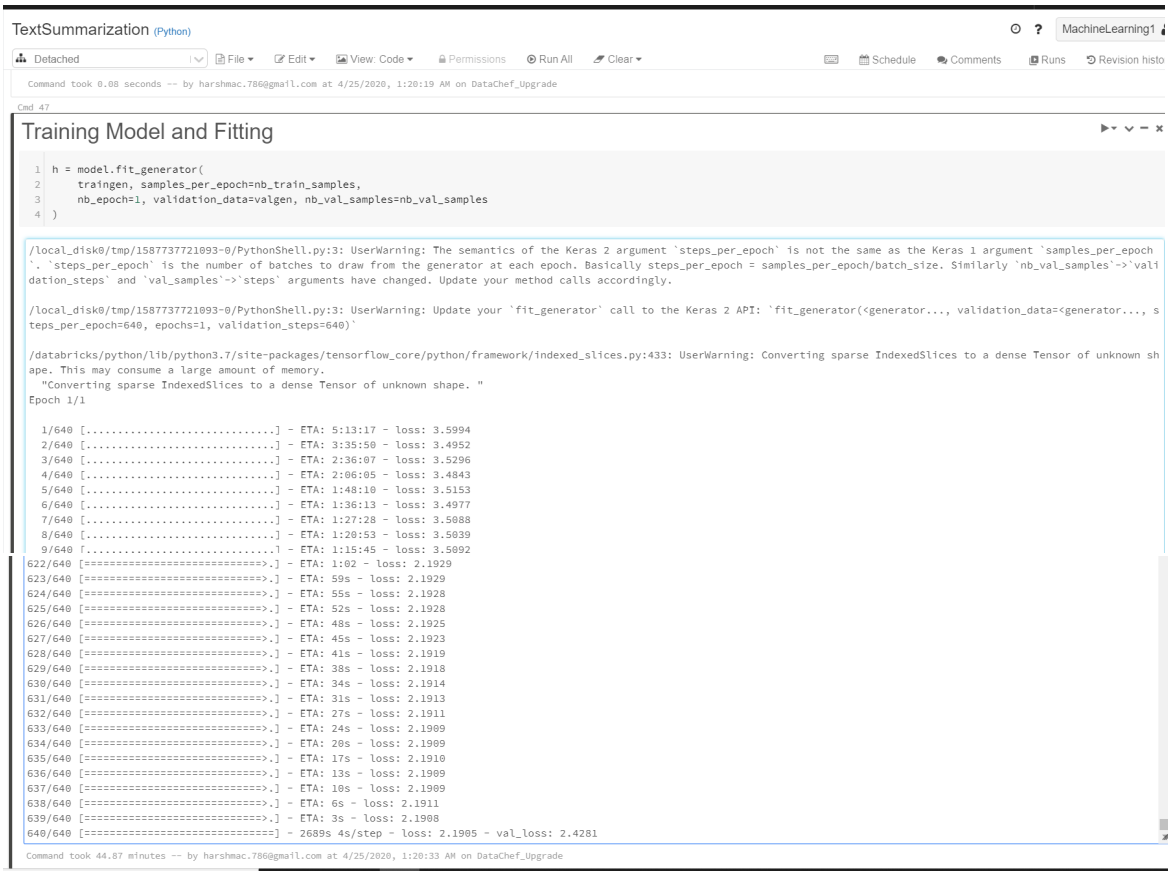
Our model has been initialized; we need to create out model parameters which include the learning rate which will enable our training speed for our model. Here we also initialize the RNN layers to be 3 and RNN size to be 512 for each layer.

Inspect our model to be accurate before training it for our dataset.



Finally, we create a generator matrix for our training dataset and our validation dataset with all the various variables we have created so far like, vocab_size, indexes to words, model, GloVe indexes, our batch size of 32.

Lastly, we train our model on training and validation dataset with 4 epochs with 640 steps in each epoch with 32 samples per step.



This screenshot above is model trained for 1 epoch with the same configuration of 640 steps and 32 samples in each step. This model took 45 minutes to train and had a training loss of 2.1905 and a validation loss of 2.4281.

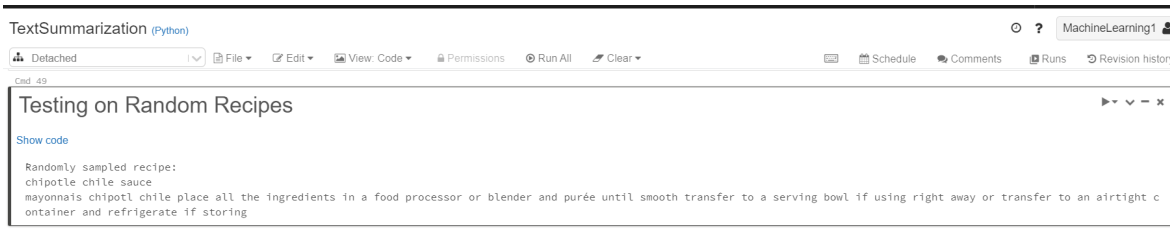
We had trained 2 more models with 4 epochs which took 5 hours to train which had a training loss of 1.3609 and a validation loss of 1.5202. Though the loss for this model was low as compared to the above model, but the predictions were grammatically correct with our 1 epoch model.

Another model was trained on 2 epochs which took 2.25 hours which has a training loss of 1.78 and a validation loss of 1.93.

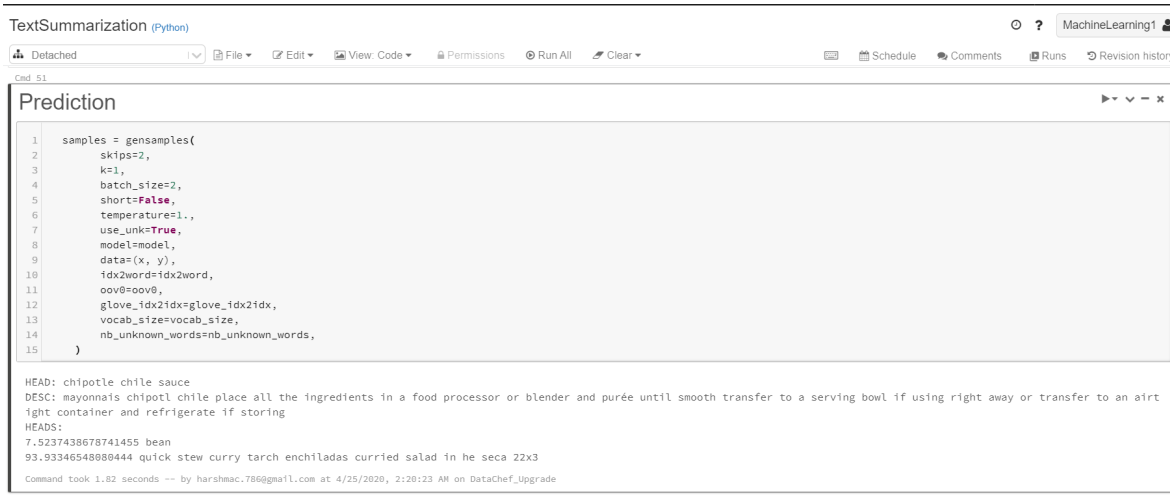
3. Prediction on sample

Now, we test our model to see our summarization of our model.

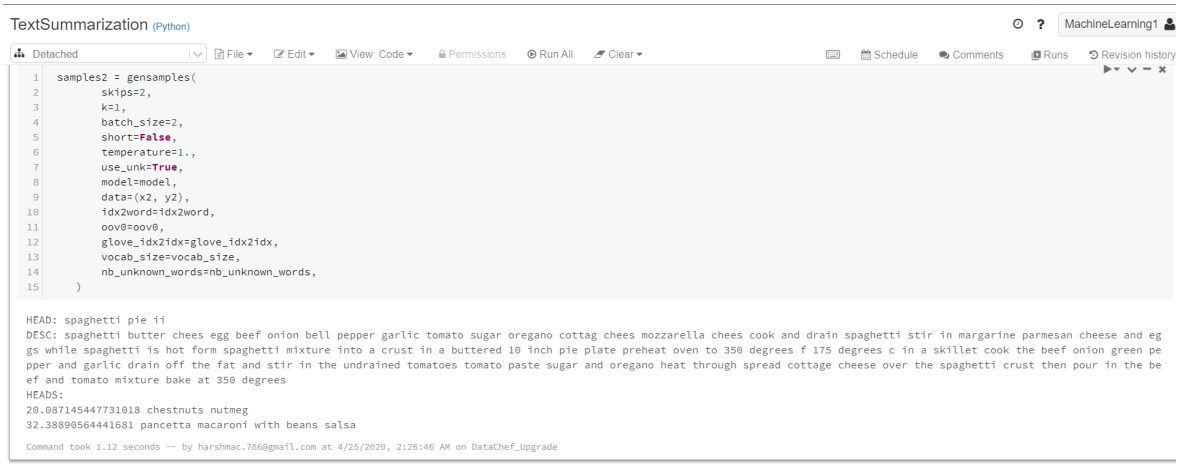
We select a random recipe from our data and try to predict the summary of recipe as a title.



Prediction:



We can see here that the second nearest prediction is grammatically correct but is no where near to our recipe title. Another example below.



So overall, we see that this model needs hyperparameter tuning to a great extent so that it can perform better with a loss value of lower that 1.

Cuisine Classification

Cuisine classification is a technique of classifying cuisine based on the ingredients present in it. The concept or the idea can also be considered as an extension of classification problem. The extra layer of preprocessing converts the ingredients into numbers and generates a final data matrix. Later, different classification models are applied on the data to classify cuisines based on the ingredients. Finally, the best model based on evaluation matrix will be deployed.

1. Data Pre-Processing

As we have the cleaned ingredients and instructions in their respective containers stored in Cosmos DB, we need to establish a DB connection with Azure Databricks where all the preprocessing will take place and the model will be trained.

To do this we use the Azure-CosmosDB-Spark connector provided by Azure to do the same. Installing this on the cluster enables us to create a connection link between Azure Databricks and the container where data is stored in CosmosDB.

In the connection configuration, we need to specify the DB endpoint, master key, Database name, DB region where it is hosted, container name and the query to fetch the documents. Then this connection config is passed as an argument to the **spark.read.format** function to create a Spark data frame of the fetched documents from cosmos DB. Repeating this same process for both the containers that is cleaned ingredients and instructions. To play with the data in python, we created a Pandas data frame from spark data frame.

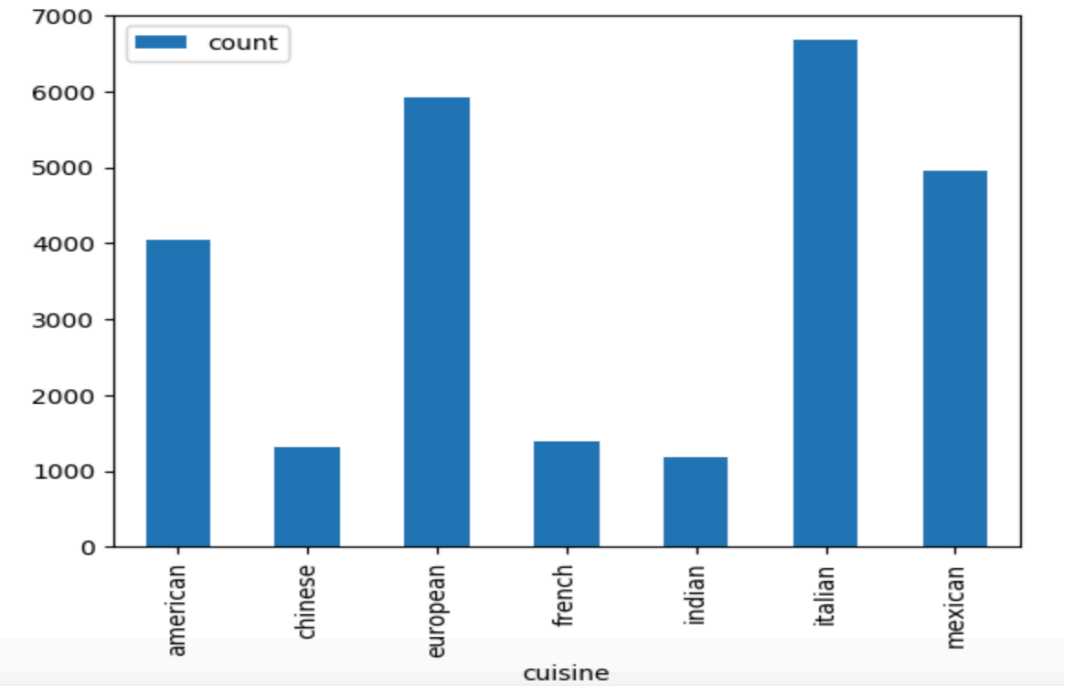
After converting the spark object to python's pandas dataframe, we are originally left with 40K entries. To keep our label count low to make our model makes better predictions, we have slices some of the cuisines from the main data frame. We have only kept seven major cuisines for example **Mexican, Italian, Chinese, Indian, European, American and French.** Alongside, we have also removed the duplicated entries present in our data to kill the bias in our model. Finally, we have left with 18,869 entries for our model to be trained on.

Our final panda's data frame looks as below:

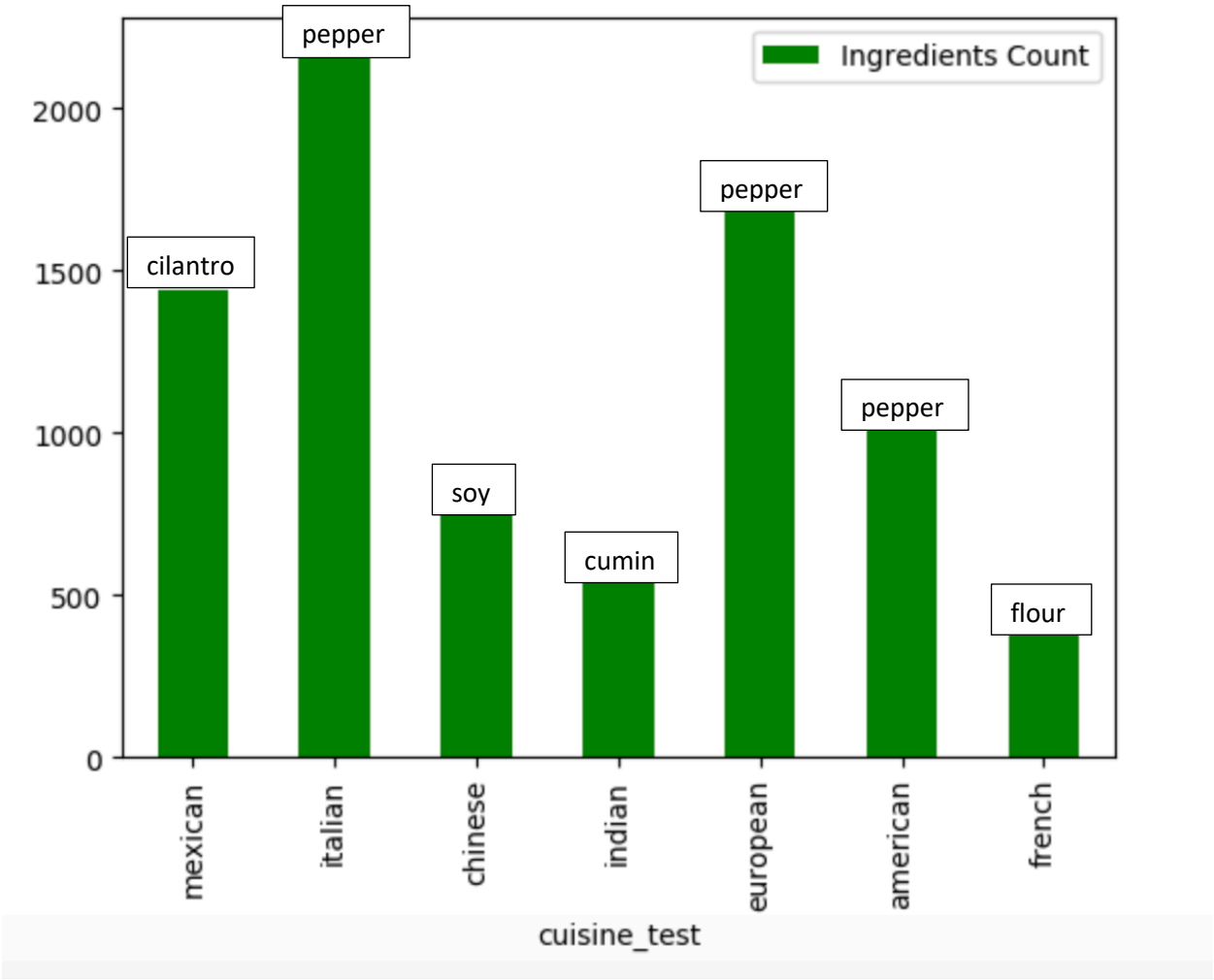
cuisine	id	ingredients	source	title
mexican	a73872f8-eb3e-4a50-9c01-12a7f3d7b9e6	[water, sugar, mint, lime, zest, salt]	AllRecipes	Watermelon-Mint Paletas
mexican	1fd48ed8-9af7-47ed-8e13-65b343a59c1f	[tomatillo, chile pepper, garlic, water, bacon...	AllRecipes	Carne en su Jugo (Meat in its Juices)
mexican	b689840e-cd48-403f-822b-7a1b559d0040	[bone chicken breast, chicken broth, onion, ga...	AllRecipes	Authentic Enchiladas Verdes
mexican	b72e97bb-20c7-49ca-a73e-0affa0a61d20	[water, cinnamon, clove, star anis, milk, tabl...	AllRecipes	Champurrado

Visualization:

1. As We have kept seven major cuisines which are Mexican, Italian, Chinese, Indian, European, American and French. The below graph shows the count of each cuisine type in the data frame.



2. The below graph shows the ingredients count in each cuisine type which is majorly present. For e.g.- Mexican use the cilantro most, Chinese use the soy most and Indian use the cumin.



Label Dictionary: Now we have created a dictionary that will save all the unique cuisines present in our data. We have taken only seven cuisines so the length of the dictionary would be seven. In the dictionary, name of cuisines will be the keys and their numerical encoding like 0,1,2 etc would be the value as per the cuisine. This dictionary will be later used to create labels for the classification model

The code below explains the part of creating and populating dictionary:

```
# a lable dictionary to be used later for creating labels
label_dictionary = {}

unique_cuisine = raw_df['cuisine'].unique()  # picking only unique cuisines
for index, uc in enumerate(unique_cuisine):  # populating label dictionary
  label_dictionary[uc] = index
```

Ingredients Dictionary: Here we have created a dictionary to save all the unique ingredients present in our data. As per the processing, there are approximately 8K unique ingredients in our data set. In the dictionary, name of ingredients is represented as keys and their numerical encoding such as 0,1,2 etc would be the values. This ingredients dictionary will be later used to create the features columns for the final matrix of classification model.

The code below explains the part of creating and populating dictionary:

```
def createIngredientDict(ingredient_corpus):  # function that creates ingredient dictionary
  ingredient_dictionary = {}  # ingredients along with their indexes
  for index, ing in enumerate(ingredient_corpus):
    ingredient_dictionary[ing] = index

  return ingredient_dictionary
```

Ingredient-To-Vector Function: Now as we know that a computer doesn't understand text and hence there was need to convert the entire list of ingredients to number. So, we have created a function called Ingredient_to_vector function that takes list of ingredients as an input and convert it into a binary vector by matching the value with the ingredient dictionary. This binary vector will be of shape (number of ingredients, 1). These different vectors for all the entries will act as a unique row in final data matrix.

The functional code to implement ingredientToVector is as follows:

```
def ingredientToVector(ingredient_list, ingredient_corpus):  # function to convert a list of ingredient to a binary vector
  ingredient_dictionary = createIngredientDict(ingredient_corpus)
  ingredient_vector = np.zeros((len(ingredient_dictionary), 1))
  indices_list = []
  for i in ingredient_list:
    if i in ingredient_dictionary:
      indices_list.append(ingredient_dictionary[i])
  for i in indices_list:
    ingredient_vector[i] = 1

  return ingredient_vector.T
```

Final Matrix: At last we have created final matrix using all the helper code written above. In this final matrix, ingredients will as feature in columns, cuisines will go a label for every entry, and ingredients list will be converted to binary vector which will than be later inserted into the matrix as an entry. The approximate dimension of our final matrix is (18K, 8K).

The code to create final matrix is shown below:

```
# creating a final matrix our from the data frame
final_matrix = np.zeros((len(raw_df), len(ingredient_corpus)+1))
for i in range(len(raw_df)):
  final_matrix[i, 0:len(ingredient_corpus)] = ingredientToVector(raw_df.iloc[i, :]['ingredients'], ingredient_corpus)
  final_matrix[i, len(ingredient_corpus)] = label_dictionary[raw_df.iloc[i, :]['cuisine']]
```

Data Splitting: As a pre-defined practice in machine learning, we always divide the data into train and test split to evaluate the model. In this part we have split our entire data matrix into train and test part. 80% of our data goes in training set and 20% in testing set.

Code to split the data is shown below:

```
features = final_matrix[:, 0:final_matrix.shape[1]-1]  # feature matrix
labels = final_matrix[:, -1]  # label vector

X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size = 0.20)  # splitting the data into training and testing set
```

Dimensionality Reduction: Model with large number of features inversely effects the computation and speed of algorithm. So, we have decided to reduce the number of features to optimize our algorithms. We have used principal component analysis by retaining 95% of the variance in the original data. Once done, total number of features have been reduced to 1.8K from 8K where at the same time keeping 95% of the variance retained.

```
pca = PCA(0.95) # instantiating pca while retaining 95% of variance in the data
pca.fit(X_train)
X_train = pca.transform(X_train)  # transforming traning set
X_test = pca.transform(X_test)  # transforming testing set
```

2. Model Selection:

Once our data is cleaned and pre-processed. We have decided to apply the different classification models on it. We have chosen few classification models such as Support vector machine with both linear and gaussian kernel, Neural network, logistic regression, random forest, decision tree etc.

Let's go through each model one by one and see how they performed on our data:

1. **Support Vector Machine – Linear Kernel:** First we have applied the support vector machine with linear kernel and evaluated the model based on training and testing set accuracy
   Training accuracy: 83%
   Testing accuracy: 72%
   Support vector machine without any regularization and optimization gave us decent result with training and testing accuracy of 83% and 72% respectively. This explains that our data is might be linearly separable and nonlinear functions and algorithms would not do well on it.
   The code is shown below:

Support Vector Machine (Linear Kernel)

```
1  svm = SVC(kernel="linear")
2  svm.fit(X_train, Y_train)
```

Show result

Cmd 13

```
1  train_score = accuracy_score(Y_train, svm.predict(X_train))
2  print("Accuracy on training set: ", train_score)
3  test_score = accuracy_score(Y_test, svm.predict(X_test))
4  print("Accuracy on testing set: ", test_score)
```

```
Accuracy on training set:  0.8306724080821464
Accuracy on testing set:  0.7231054583995761
```

Command took 6.57 minutes -- by utkarshkekre@gmail.com at 4/25/2020, 10:22:08 AM on DataChef_Upgrade

2. **Support Vector Machine – Gaussian Kernel:** As we have sensed in our first model that we might have a linear separation in our data and so the nonlinear functions might not perform well on our data. Let's see, what happened when we applied gaussian kernel SVM on our data.
   Training accuracy: 70%
   Testing accuracy: 47%
   As expected, support vector machine with non-linear kernel didn't perform well on both training and testing accuracy.

Support Vector Machine (Gaussian Kernel)

```
1  svm_rbf = SVC(kernel="rbf")
2  svm_rbf.fit(X_train, Y_train)
```

Show result

Cmd 15

```
1  train_score = accuracy_score(Y_train, svm_rbf.predict(X_train))
2  print("Accuracy of training set: ", train_score)
3  test_score = accuracy_score(Y_test, svm_rbf.predict(X_test))
4  print("Accuracy of testing set: ", test_score)
```

```
Accuracy of training set:  0.4741967538920172
Accuracy of testing set:  0.4785373608903021
```

Command took 10.50 minutes -- by utkarshkekre@gmail.com at 4/25/2020, 10:42:25 AM on DataChef_Upgrade

3. **Neural Network:** Fit the neural network with 3 hidden layers of 100 units each. Activation function we have used is the relu activation. We have used random initialization, but we haven't used any regularization in this. Let's see the evaluation matrix.
   Training accuracy: 99%
   Testing accuracy: 75%
   Neural network without any hyperparameter tuning gave us the satisfying result. We have expected that along with "He" initialization and L2 regularization can dramatically improve our testing accuracy as well.

Neural Network

```
1  nn = MLPClassifier(hidden_layer_sizes = (100, 100, 100))
2  nn.fit(X_train, Y_train)
```

Show result

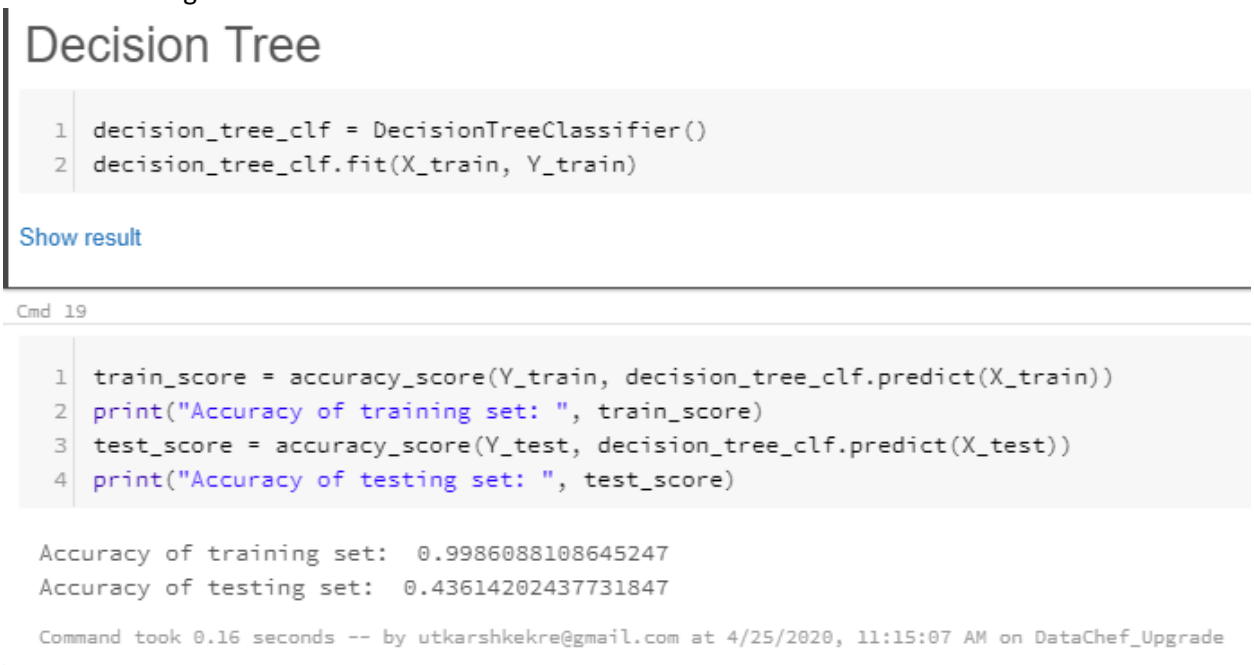
Cmd 17

```
1  train_score = accuracy_score(Y_train, nn.predict(X_train))
2  print("Accuracy of training set: ", train_score)
3  test_score = accuracy_score(Y_test, nn.predict(X_test))
4  print("Accuracy of testing set: ", test_score)
```

```
Accuracy of training set:  0.9942365021530308
Accuracy of testing set:  0.7011128775834659
```

Command took 0.25 seconds -- by utkarshkekre@gmail.com at 4/25/2020, 11:13:05 AM on DataChef_Upgrade

4. **Decision Tree:** Fit the decision tree on our data and got the following results:
   Training accuracy: 99%
   Testing accuracy: 45%
   Decision tree without any regularization and optimization is showing a problem of overfitting. So, even with regularization it is not possible to have a significant rise in testing accuracy. So, we have not decided to go ahead with this model.



5. **Random Forrest:** Fitting random forest with an n_estimators value of 100 gave us the following accuracy on training and testing data.
   Training accuracy: 99%
   Testing accuracy: 65%
   Tough this model has performed better than some of the models like decision tree and SVM with gaussian kernel still it is not better than logistic regression or neural network. So, we thought of dropping this model

Random Forest

```
random_forest = RandomForestClassifier(n_estimators = 100)
random_forest.fit(X_train, Y_train)
```

Show result

Cmd 21

```
train_score = accuracy_score(Y_train, random_forest.predict(X_train))
print("Accuracy of training set: ", train_score)
test_score = accuracy_score(Y_test, random_forest.predict(X_test))
print("Accuracy of testing set: ", test_score)
```

```
Accuracy of training set:  0.9986088108645247
Accuracy of testing set:  0.624006359300477
```

Command took 1.42 seconds -- by utkarshkekre@gmail.com at 4/25/2020, 11:17:00 AM on DataChef_Upgrade

6. **Logistic Regression:** At last we have applied logistic regression hoping that it would give us some satisfying result as our data is linearly separable.
   Training accuracy: 80%
   Testing accuracy: 72%
   Most of the model showed overfitting for the data. Whereas logistic regression is the only model that showed nearly good fit without regularization. And, this is a no surprise as our data is linearly separable and logistic regression is best tool for fitting linearly separable data. So, we have decided to include this model.

Logistic Regression

```
log_reg = LogisticRegression(penalty="l2")
log_reg.fit(X_train, Y_train)
```

Show result

Cmd 23

```
train_score = accuracy_score(Y_train, log_reg.predict(X_train))
print("Accuracy of training set: ", train_score)
test_score = accuracy_score(Y_test, log_reg.predict(X_test))
print("Accuracy of testing set: ", test_score)
```

```
Accuracy of training set:  0.7953627028817489
Accuracy of testing set:  0.7286698463169051
```

Command took 0.12 seconds -- by utkarshkekre@gmail.com at 4/25/2020, 11:18:52 AM on DataChef_Upgrade

3. Python Modules:

We have used built-in python modules in this project for cleaning, pre-processing and modelling of data. This list of the modules is here as follows:

1. **BeautifulSoup**: To scrape the data from the website
2. **Pandas**: Python's built in library to handle large data sets
3. **Numpy**: Python's library for scientific computation
4. **Sci-kit Learn**: Python's module for model selection, data modelling and evaluation

Visualizations

Building Connection:

As we have the cleaned nutrition's and the content in the nutrients in the containers stored in Cosmos DB, we need to establish a DB connection with Azure Databricks where all the visualization will take place.

To do this we use the Azure-CosmosDB-Spark connector provided by Azure to do the same. Installing this on the cluster enables us to create a connection link between Azure Databricks and the container where data is stored in CosmosDB.

In the connection configuration, we need to specify the DB endpoint, master key, Database name, DB region where it is hosted, container name and the query to fetch the documents. Then this connection config is passed as an argument to the **spark.read.format** function to create a Spark data frame of the fetched documents from cosmos DB. Repeating this same process for both the containers that is cleaned nutrition's. To play with the data in python, we created a Pandas data frame from spark data frame.

Nutritional Evaluation:

Nutritional Assessment is important to have a complete picture of the food-related health and total body wellness. It is recommended to have information of "why and when to eat" and not just for "what to eat".

According to recent studies, most Americans don't get enough nutrients but are over-the-top on the amount of sodium and fat in their diets. Researchers found that many American adults fell short of consuming enough vitamin A, vitamin C, vitamin D, calcium, and iron. In addition, Americans have a tendency to eat far more saturated fat, cholesterol, and sodium than is normal or recommended.

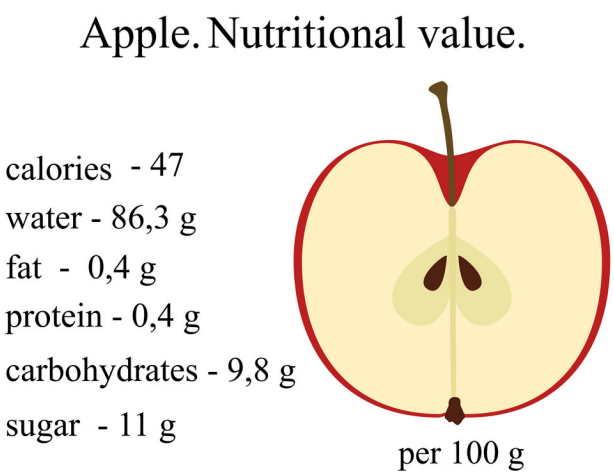
There are few symptoms which depicts that you have a dietary issue:

- Ongoing fatigue that keeps you from feeling rested.
- Constipation or recurrent diarrhea.
- Blood sugar levels that fluctuate greatly throughout the day.
- Low grade aches and pains with no other cause.
- Lowered immune system leaving you susceptible to illnesses.
- Irritability, depression, and other mood issues.
- Obesity and excess weight Dizziness.

- Headaches, migraines, and brain fog.
- Daily heartburn with no other cause

To keep a track on which recipe has what nutrition content, we suggest using the below suggested evaluations to display on web pages, as it will result in a better user experience and users can track the nutritional value of the dish they are going to consume. It will be beneficial for:

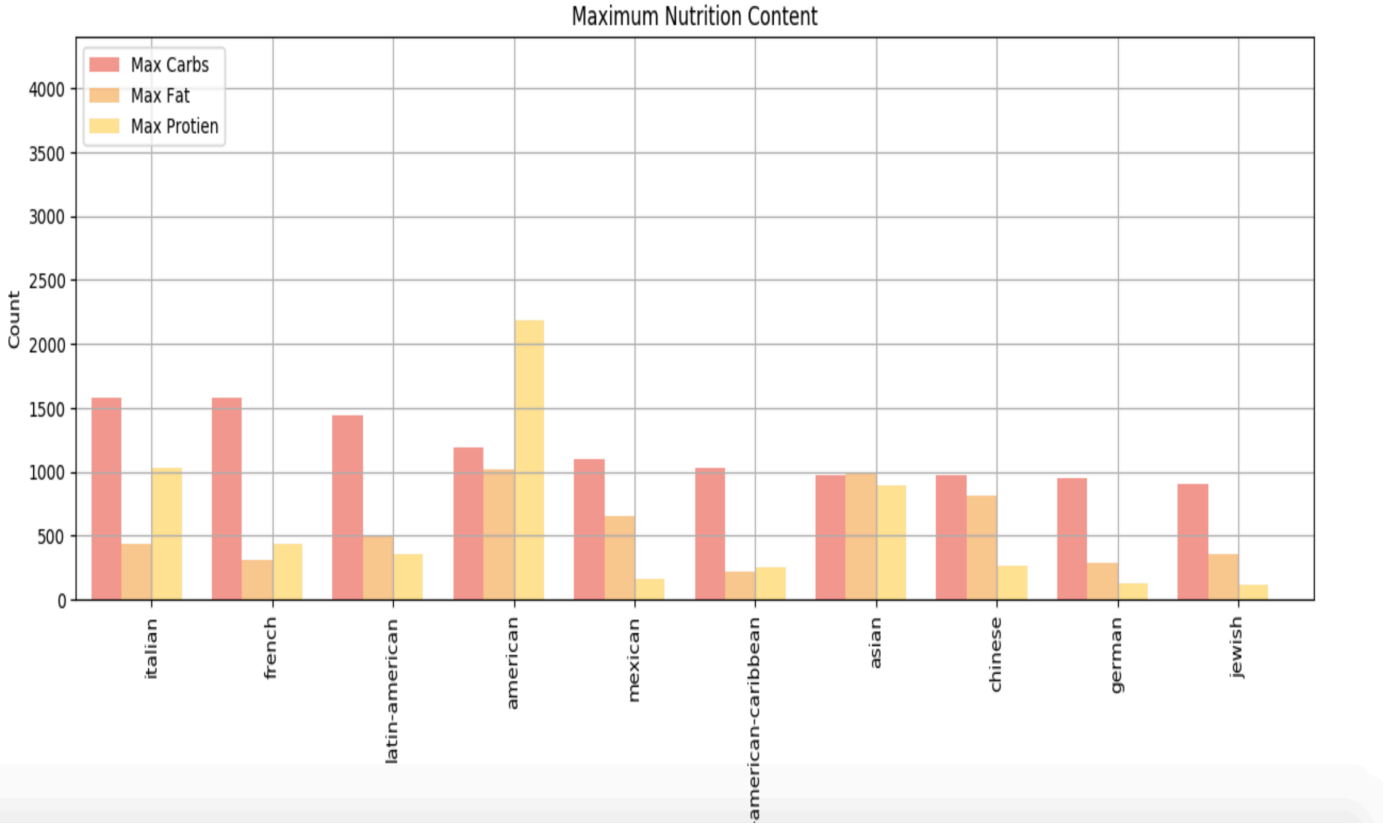
- Health Conscious Customers.
- Calorie Counting Customers.
- Keeping it Legal, as many Canadian food companies have recently been subjected to a healthy ban on trans-fats.
- Professionalism.
- Doing the Right Thing as you are being honest and accountable.



Recommended Evaluations to Display on Website:

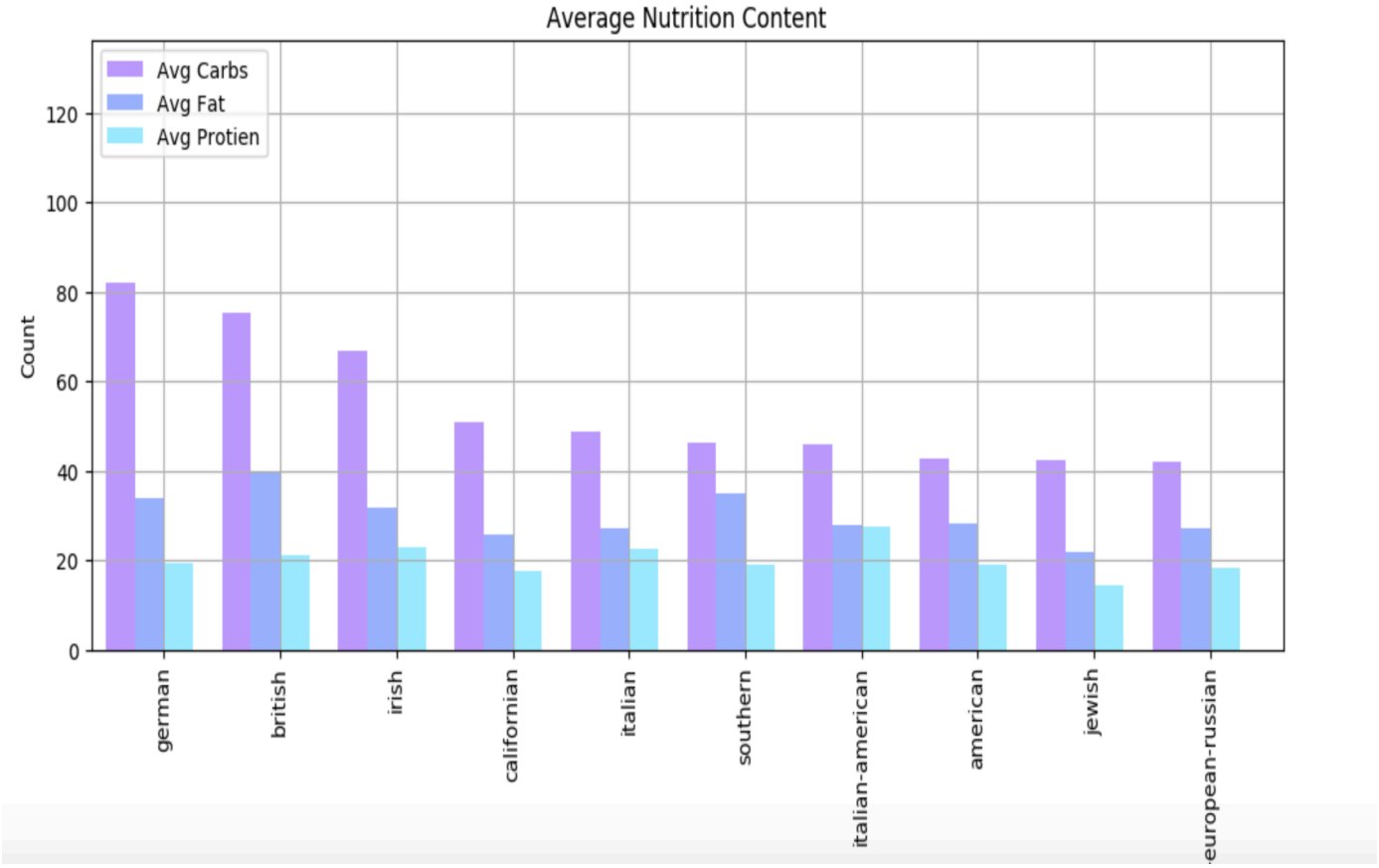
1. **Maximum Nutrition Content**

   The graph shows the top 10 cuisine type with maximum carbs, maximum fat and maximum Protein. This will help customer to calculate their nutrition content and decide what to eat, whether they want to eat something which has maximum carbs (cuisine type like- Italian, French) or which has maximum fat (cuisine type like- American, Asian) or which has maximum protein (cuisine type like- American, Italian).



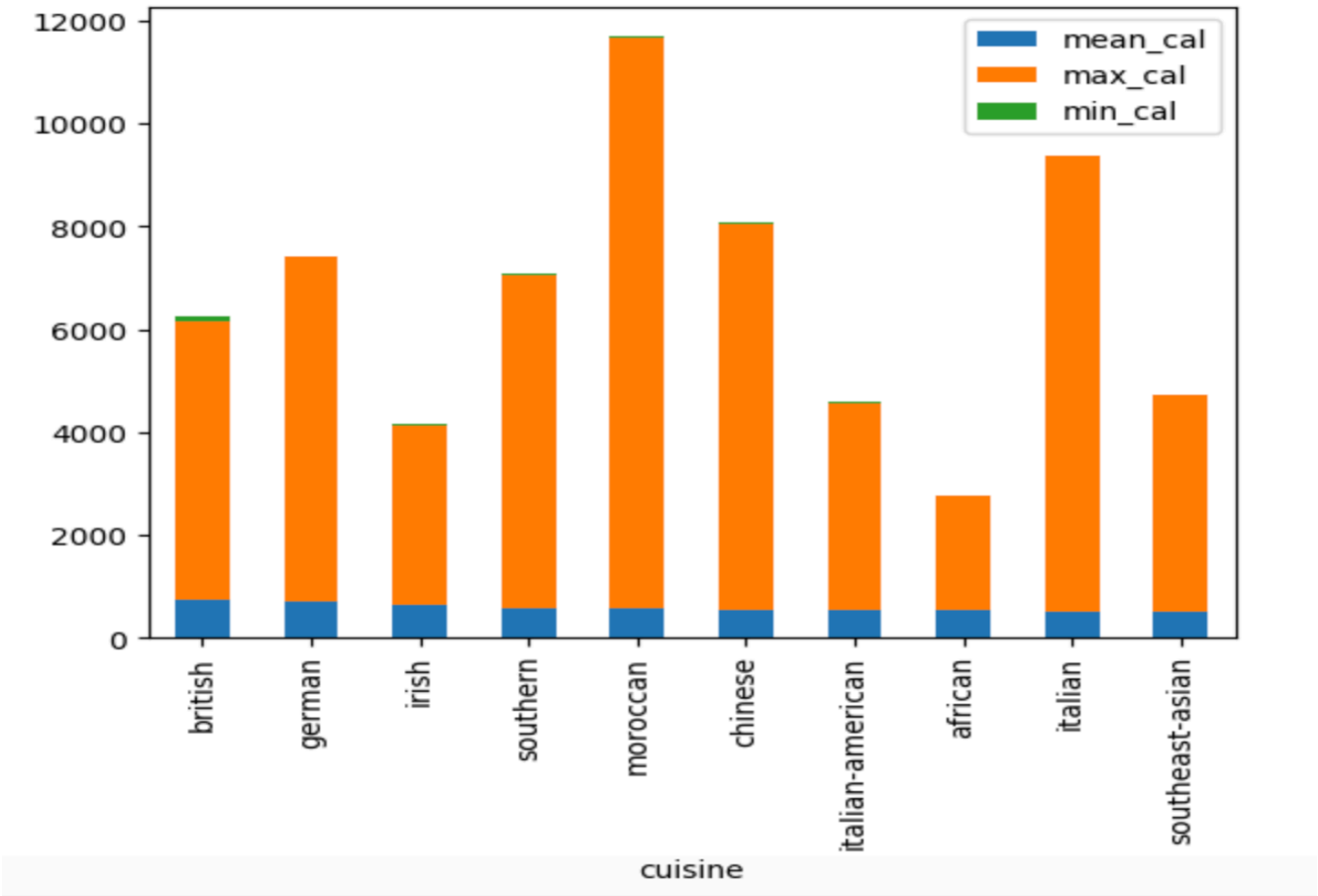
2. **Average Nutrition Content**

The graph shows the top 10 cuisine type with average carbs, average fat, and average Protein. It is strange to know that cuisine type like German and British has the most stable average content of all of the three nutrient content which are carbs, fat and protein.



3. **Calories Content**

This graph shows the maximum, minimum and average calories content in the top 10 cuisine type. Having a calorie count is extremely important as this will help one to keep track of calorie in and calorie out concept. Cuisine type like Moroccan, Italian and Chinese has the maximum calorie count.



Business Values:

We are targeting to deliver the best to our customers and this solution will generate business values in 5 ways:

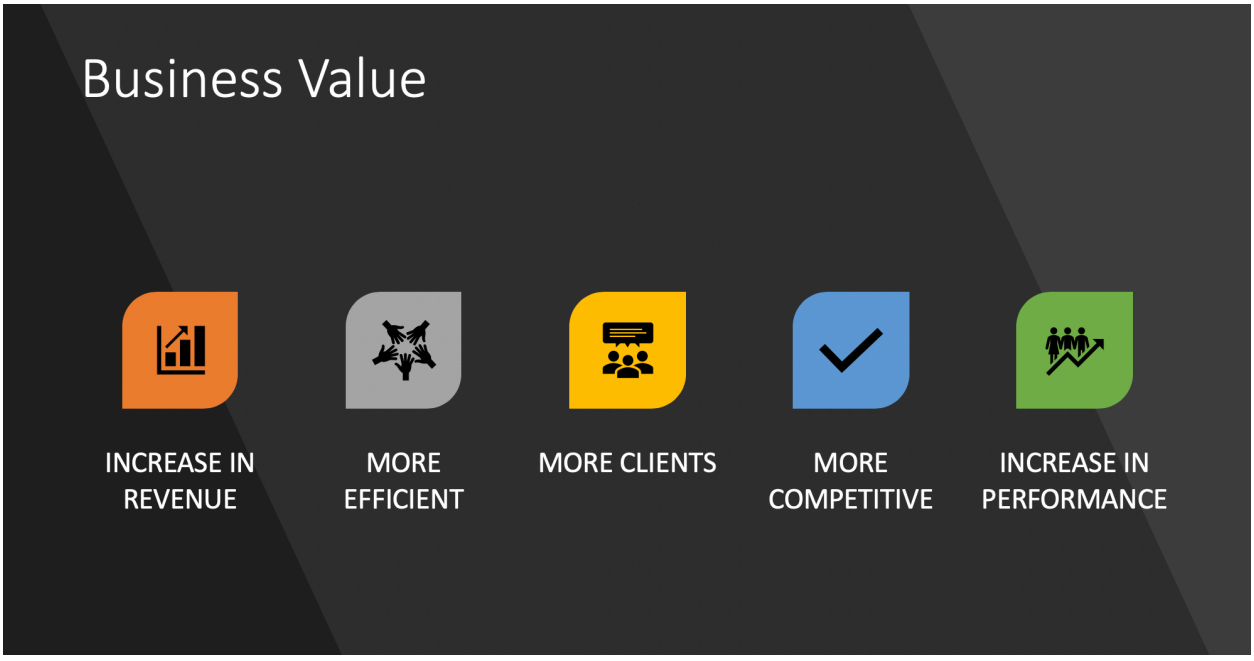
Increase in Revenue: As we provide more facilities to the users, it will attract more users due to mouth to mouth publicity and also, marketing from online platforms. The increase in customer base will result in increase of the revenue.

More Efficient: We are targeting to give a better user experience when anyone uses the website, we are giving a meaningful title to the recipe, which will reduce the screen time for the user and user can be more productive in the time they spent on the website. This will earn trust for the company.

More Clients: Increasing the user satisfaction will result in attracting more clients and increase the customer base of the company.

More Competitive: As we are also going to provide the nutritional values evaluation for cuisines and recipes, users can use it as their calorie counter and can make use of it for planning their diet. This feature will give competition to the companies like HealthifyMe, Fitocracy etc., henceforth, it will be a strong competitor in the market.

Increase in Performance: As the above-mentioned values will contribute in the success of the food website company, shortly the reduced screen time, less clicks, and more information will result in alleviating the performance of the website, hence increasing the customer satisfaction.



References:

https://expandedramblings.com/index.php/allrecipes-facts-statistics/

https://www.similarweb.com/website/epicurious.com

https://en.wikipedia.org/wiki/Allrecipes.com

https://en.wikipedia.org/wiki/Epicurious

https://www.netwoven.com/2018/12/04/a-complete-walkthrough-of-azure-cosmos-db-and-why-should-you-use-it/

https://milehighspine.com/the-importance-of-a-nutritional-assessment/

https://www.menusano.com/nutrition-blog/6-ways-providing-nutritional-value-information-increases-sales-and-grows-your-food-service-business/

https://nlp.stanford.edu/projects/glove/