Introduction To
Machine Learning Apps

A Project Report On:

WHAT's COOKING??

Submitted To:
Dr. Lydia Manikonda

Submitted By:
Chirag Sahni

PREFACE

EXECUTIVE SUMMARY.....1

INTRODUCTION.....1
CONTEXT & PROBLEM STATEMENT.....1
BUSINESS APPLICATIONS.....1

DATASET SUMMARY.....2

BENCHMARKING THE THREE SOLUTIONS CHOSEN.....3 – 9

SUMMARY.....3
DETAILED ANALYSIS OF SOLUTION 1.....4 – 5
DETAILED ANALYSIS OF SOLUTION 2.....6– 7
DETAILED ANALYSIS OF SOLUTION 3.....8 – 9

DATA UNDERSTANDING AND VISUALIZATIONS.....10 – 13

STRUCTURE OF THE DATASET.....10
VISUALIZATIONS.....10 – 13

APPLIED OWN MODEL (SOLUTION4).....14 – 15

MODULATING THE SIZES OF TRAINING AND TESTING DATASETS.....16 – 17

CONCLUSIONS.....18

REFERENCES.....18

GitHub Repository URL: https://github.com/chiragsahni/What-is-Cooking

EXECUTIVE SUMMARY / ABSTRACT

INTRODUCTION

Nations or countries are frequently associated with certain foods. In a country such as the USA, we find people of various nationalities who are from all over the world. Immigrants often use food as a means of retaining their cultural identity. As people immigrate, food practices and preferences are imported and exported. And when you find yourself living in other parts of the world, having your traditional meals can be a great way to alleviate your homesickness. [1]

George Bernard Shaw said it well: "There is no love sincerer than the love of food."
People from different cultural backgrounds eat different foods & the ingredients, methods of preparation and types of food eaten at different meals vary among cultures.[1]
Being a food lover myself, I have chosen the 'What's Cooking' *ML* project from the Kaggle portal.

CONTEXT & PROBLEM STATEMENT

Machine learning is a part of computer science focused on computer systems learning to perform a specific task without using explicit instructions, relying on patterns and inference instead. In simpler terms, we can say instead of relying on explicit programming, it is a system through which computers use a massive set of data and apply algorithms to "train" on--to teach themselves--and make predictions.
Business applications of machine learning are numerous, but all boil down to one type of use: Processing, sorting, and finding patterns in huge amounts of data that would be impractical for humans to make sense of. [2]

Goal of the Project – To predict the type of cuisine of a recipe/dish on the basis of its ingredients.

BUSINESS APPLICATIONS / FUTURE SCOPE

This project of What's Cooking has a very wide scope in the machine learning world and has multiple business applications in the real world. A business application of this project is that it can be used by the **online food delivery industry** to automatically divide a new partner restaurant's menu into sub-menus based on different cuisines. Hence, expediting the process of adding a new entity into the existing system which is almost impossible to do by implementing other technological assets like a core programming interface.
If we can get an extra feature / field of 'Calories Contained' in the recipe added to the dataset, this project can turn out to be of great utility to the **dieticians and nutritionists.** For example, a health consultant can define personalized diet charts for its clients which vary depending on the various cuisines the client prefers.

DATASET SUMMARY

- The data is provided to Kaggle by **Yummly**, a renowned multi-utility American website, known for providing personalized recipe recommendations.
- 2 distinct datasets in **JSON format** where individual records combine to form a list of JSON records.
    1. No. Recipes in Train Dataset – 39774
    2. No. Recipes in Test Dataset – 9944
- Can be conveniently converted to Pandas dataframes using third party libraries.
- Adjacent screenshot shows a sample record from the training dataset where each record consists of three fields / features:
    1. ***id*** – Unique identifier for a record / recipe.
    2. ***ingredients*** – JSON array containing the list of ingredients for a particular recipe.
    3. ***cuisine*** – ***Class Label*** – Based on the list of ingredients, gives the type of cuisine that record / recipe belongs to. As, it is a class label, this is not present in the test dataset.


```
{
"id": 24717,
"cuisine": "indian",
"ingredients": [
    "tumeric",
    "vegetable stock",
    "tomatoes",
    "garam masala",
    "naan",
    "red lentils",
    "red chili peppers",
    "onions",
    "spinach",
    "sweet potatoes"
]
},
```


Fig. 1 – Snapshot of Data in JSON format

	id	cuisine	ingredients
0	10259	greek	[romaine lettuce, black olives, grape tomatoes...
1	25693	southern_us	[plain flour, ground pepper, salt, tomatoes, g...
2	20130	filipino	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	22213	indian	[water, vegetable oil, wheat, salt]
4	13162	indian	[black pepper, shallots, cornflour, cayenne pe...
5	6602	jamaican	[plain flour, sugar, butter, eggs, fresh ginge...
6	42779	spanish	[olive oil, salt, medium shrimp, pepper, garli...
7	3735	italian	[sugar, pistachio nuts, white almond bark, flo...
8	16903	mexican	[olive oil, purple onion, fresh pineapple, por...
9	12734	italian	[chopped tomatoes, fresh basil, garlic, extra-...

Fig. 2 – Snapshot of Data After Converted to a Dataframe

BENCHMARKING THE THREE SOLUTIONS CHOSEN

1. Solution1 – Using XGB Classifier [3]
2. Solution2 – Using Random Forest Classifier [4]
3. Solution3 – Using Linear SVC [5]

Parameters	Solution1	Solution2	Solution3
Libraries	Numpy, Pandas, Matplotlib, Seaborn, Sklearn, XGBoost	Numpy, Pandas, OS, Json, IPython, Sklearn	Numpy, Pandas, Sklearn, Time, Json, Seaborn
EDA	Performed the following activities on the training & testing datasets both: • Extracted unique ingredients for different cuisines. • Created a matrix showing the contained ingredients for a recipe from a row of all the unique ingredients. • Found top 15 (overall) ingredients used. • Identified number of recipes per each cuisine. Also, checked proportions for the same.	Performed the following activities on the training & testing datasets both: • As the given data format cannot be used in RFC, first the datasets are converted to matrices with columns those need to be every unique ingredient with a 0 or 1 if the ingredient is in a particular cuisine. • Extracted unique ingredients for different cuisines. • Using above two things, created a dataframe depicting each cuisine and ingredients present in all of its recipes.	Performed the following activities on the training dataset: • Separated out cuisines and their list of ingredients in different lists. • Transformed the list of array of ingredients to a list of strings containing the ingredients separated by a space ‘ ’. • Used TF-IDF algorithm on the **transformed corpus** to get the term frequency for each ingredient in each recipe. • Performed label encoding to normalize and encode the labels for different cuisines in order to fit LSVC model.
Model Used	**XGB Classifier** with the objective set to ‘multi:softmax’ which forces XGBoost to do multiclass classification. Also, sets the number of classes equal to unique number of cuisines in the training dataset.	**Random Forest Classifier** with number of estimators set to 10 and criterion = ‘entropy’ with the objective of computing the information gain. At last, performed cross validation to calculate the accuracy.	**Linear Support Vector Classification** with the cuisines which were label encoded in the previous step. The maximum number of iterations to be run was set to 1000.
Performance	Accuracy Achieved = 74.78%	Accuracy Achieved = 63%	Accuracy Achieved = 78.27%

DETAILED ANALYSIS OF SOLUTION – 1

- The user imports the train and test datasets using 'os' and 'zipfile' python libraries. This way the data is imported in JSON format which is later converted to a dataframe *(Fig. 2)*.

```
import os, zipfile
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        p = os.path.join(dirname, filename)
        with zipfile.ZipFile(p, 'r') as zip_ref:
            zip_ref.extract('.'.join(filename.split('.')[:-1]))
```

Fig. 3 – Solution1: Data Import

- The transformation to the dataframe is followed by the process of identification of the unique ingredients **(7137 unique ingredients)** for different cuisines. Then this useful information is stored in the format shown in *Fig. 5* using a user defined function called 'format_ingredients'. The ingredients present in a recipe would be marked by 1 in the respective columns. Unfortunately, I could not get a screenshot of any columns with value = 1 due to restrictions on data display of a huge dataframe.

- All this processing and transformations have been done on both train and test datasets.

```
lists_of_ingredients = train.ingredients.values.ravel().tolist() + test.ingredients.values.ravel().tolist()

unique_ingredients = sorted(list(set(list(itertools.chain(*lists_of_ingredients)))))
```

```
%%time
train[unique_ingredients] = pd.DataFrame(train.ingredients.apply(lambda x: format_ingredients(unique_ingredients,x)))
train.drop(['ingredients'],axis=1,inplace=True)
```

Fig. 4 – Solution1: Extracting Unique Ingredients

	id	cuisine	(oz.) tomato sauce	(oz.) tomato paste	(10 oz.) frozen chopped spinach	(10 oz.) frozen chopped spinach, thawed and squeezed dry	(14 oz.) sweetened condensed milk	(14.5 oz.) diced tomatoes	(15 oz.) refried beans	1% low-fat buttermilk	...	yuzu	yuzu juice	za'atar	zabaglione	zest	zesty italian dressing
0	10259	greek	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	25693	southern_us	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	20130	filipino	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	22213	indian	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

Fig. 5 – Solution1: Dataframe Created After Extracting Unique Ingredients

- This is followed by finding the top 15 ingredients overall and this together with the extracted unique ingredients prepare the dataset to be used for applying XGBoost model which the user in this case has opted to apply in this project.

- "XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data artificial neural networks tend to outperform all other algorithms or frameworks." [6]

- The split is done using the option *'stratify'* which is responsible for preserving the proportion of target as in original dataset, in the train and test datasets as well. **XGB Classifier** is used with the objective set to *'multi:softmax'* which forces XGBoost to do multiclass classification. Also, sets the number of classes equal to unique number of cuisines in the training dataset.

```
%%time
train_,test_ = train_test_split(train,test_size=0.33,random_state=42,stratify=train.cuisine)

xgc = xgb.XGBClassifier(objective='multi:softmax',num_class=train.cuisine.nunique())
xgc.fit(train_[train_.columns[2:]],train_['cuisine'])
```

```
results = test_[['id','cuisine']].copy()
results['y_pred'] = xgc.predict(test_[test_.columns[2:]])
```

Fig. 6 – Solution1: Splitting the Data, Applying the Model & Predicting the Cuisine for Test Data

- It took more than 6 hours for this model to execute completely.
- This solution achieved the tag of 'Best Submission' and achieved a private / public score of 0.74778 or we can say an accuracy of around 74% – 75%.

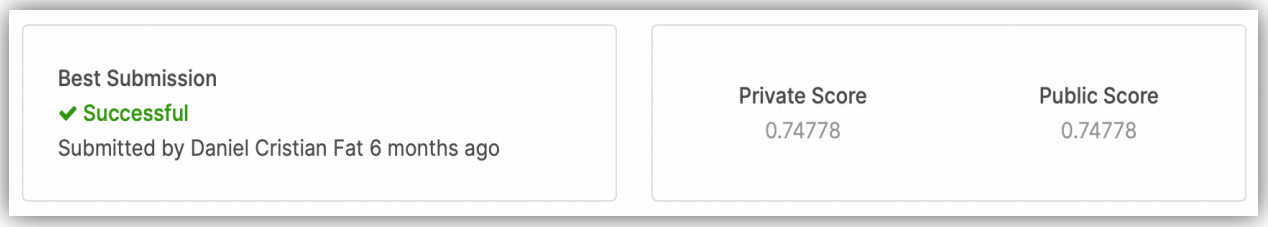

Fig. 7 – Solution1: Scores or Accuracy Achieved

DETAILED ANALYSIS OF SOLUTION – 2

- The user imports the train and test datasets using 'json' python library. This way the data is imported in JSON format which is later converted to a dataframe *(Fig. 2)*.

- For the Random Forest Classifier to be applied efficiently, the datasets should be in a specific format which the user does by importing the data by using 'json.load' functions.

```
with open('/kaggle/input/whats-cooking/train.json') as json_file:
    data = json.load(json_file)

with open('/kaggle/input/whats-cooking/test.json') as json_file:
    test_data = json.load(json_file)
```

Fig. 8 – Solution2: Data Import

- For example, in the train data, the columns need to be every unique ingredient with a 0 or 1 if the ingredient is in a particular cuisine. As the test data would not have the cuisines (class labels) listed, the user uses the random classifier defined to get those. *The same thing which was done in SOLUTION1 but with a different approach altogether.*

```
lst = []
for x in data:
    id_=x["id"]
    ingredients = x['ingredients']
    for ingred in ingredients:
        lst.append(ingred)
```

Fig. 9 – Solution2: Extracting Unique Ingredients

- Every operation performed on the train data by the user is also performed on the test data to make it compatible / similar to the format required by the Random Forest model.

- Following the extraction of the list of unique ingredients, the user creates a dictionary whose primary key is called 'cuisine' and the value is a list of all the cuisines in the dataset. The other keys in the dictionary are all the ingredients in the dataset as keys. The values are 0 or 1 depending on if the particular cuisine mentioned has the ingredient.

```
ingred_value = {ingredient: [] for ingredient in unique_ingredients}
ids = []
ciusine = []
ingredient_lst = unique_ingredients
ingredients_for_id =[]
for x in data:
    id_=x["id"]
    ingredients_in_id = x['ingredients']
    ids.append(id_)
    ciusine.append(x["cuisine"])
    for ingred in ingred_value:
            if ingred in ingredients_in_id:
                ingred_value[ingred].append(1)
            else:
                ingred_value[ingred].append(0)
ingred_value["cuisine"] = ciusine

train_df = pd.DataFrame(ingred_value)
```

	demi-glace	vegan yogurt	pie pastry	ic pop	frozen potatoes	arbol chile	Herdez Salsa	ice	jeera	banana leaves	...	whole wheat flour	mild curry paste	buttercrean frosting
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

Fig. 10 – Solution2: Prepping the Dataset Before Applying the Model & train_df

- "Random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the model's prediction after it is executed." [7]

```
y = train_df["cuisine"]
X = train_df.drop("cuisine", axis = 1)

rf_model = skens.RandomForestClassifier(n_estimators=10,oob_score=True, criterion='entropy')
rf_model.fit(X,y)
```

Fig. 11 – Solution2: Splitting the Data, Applying the Model & Predicting the Cuisine for Test Data

- **Random Forest Classifier** is used with number of estimators set to 10 and criterion = 'entropy' with the objective of computing the information gain.
  This model executed for around 4.5 hours to give an accuracy of around 63%. The user **predicts** using the applied model and performs cross validation to get the accuracy score.

```
#Get an accuracy score using the mean and standard deviation multiplied by 2
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.63 (+/- 0.01)
```

Fig. 12 – Solution2: Scores or Accuracy Achieved

DETAILED ANALYSIS OF SOLUTION – 3

- The user imports the train and test datasets using 'json' python library. This way the data is imported in JSON format which is later converted to a dataframe *(Fig. 2)*.

- The transformation to the dataframe is followed by the process of identification of the list of all ingredients (separated by a ' ') **(39774 unique ingredients)** for different cuisines.

```
docs = []
for ingredient in train.ingredients:
    temp = ""
    for item in ingredient:
        temp = temp + item + " "
    docs.append(temp)
print(len(docs))
print(docs[1])
print(type(docs))
```

```
39774
plain flour ground pepper salt tomatoes ground black pepper thyme eggs green tomatoes yellow corn meal milk vegetab
```

Fig. 13 – Solution3: Creating a list of all ingredients

- Next the user uses the TF-IDF algorithm on the **transformed corpus** to get the term frequency for each ingredient in each recipe.

```
vectorizer = TfidfVectorizer()
X_transformed = vectorizer.fit_transform(docs)
print(X_transformed)
```

```
  (0, 738)        0.3343204746101372
  (0, 522)        0.14568369866765699
  (0, 958)        0.3040361765035925
  (0, 184)        0.20748802168948122
```

Fig. 14 – Solution3: Calculating the term frequency for all ingredients using TF-IDF algo

- Now, as the cuisines are a categorical column, the user uses label encoding to convert it and make it compatible to apply Linear SVC column.

```
X_transformed.shape
target_enc = LabelEncoder()
y = target_enc.fit_transform(train.cuisine)
y = y.reshape(-1)
y
```

```
array([ 6, 16,  4, ...,  8,  3, 13])
```

Fig. 15 – Solution3: Apply Label Encoding to Convert Categorical Features

- "The objective of a **Linear SVC** (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is."

```
data_train, data_test, label_train, label_test = train_test_split(X_transformed, y, test_size=0.33)
lsvc = LinearSVC()
```

Fig. 16 – Solution3: Splitting the Data, Applying the Model & Predicting the Cuisine

- The maximum number of iterations to be run was set to 1000 which is the default value. This model gave an accuracy of around 78.26%. The user **predicts** using the applied model and performs cross validation to get the accuracy score.

```
s = time.time()
print (cval.cross_val_score(lsvc, data_train, label_train, cv=10).mean())
print("Training Time" + str(time.time() -s ))
```

```
0.7826523017647303
Training Time12.196321249008179
```

Fig. 17 – Solution3: Scores or Accuracy Achieved

Solution2 has the least accuracy and the main reason behind this is there is **no significant EDA** or data transformations before the application of model. Yes, there is a transformation to make the **data suitable for RF Classifier,** but it doesn't count as a means to increase the accuracy.

The other two solutions that are **Solution1 and Solution3 implement much more sophisticated machine learning models in the forms of XGB Classifier and Linear SVC respectively.** But this is not the sole reason for their increased accuracy as compared to the Solution2. Their EDA and other data transformations also play a critical role in increasing the accuracy of the models.

DATA UNDERSTANDING & VISUALIZATIONS

STRUCTURE OF THE DATASET

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39774 entries, 0 to 39773
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   id           39774 non-null  int64 
 1   cuisine      39774 non-null  object
 2   ingredients  39774 non-null  object
```

Fig. 18 – Structure of the Dataset

The main points to note down from this are:

- Total number of records: 39773
- There are no null values in any of the columns.
- Only the 'id' column is of integer type.

```
num_cuisines = len(df_train['cuisine'].unique())
print(num_cuisines)
```

```
20
```

Fig. 19 – Number of Distinct Cuisines

HOW MANY RECIPES IN EACH CUISINE?

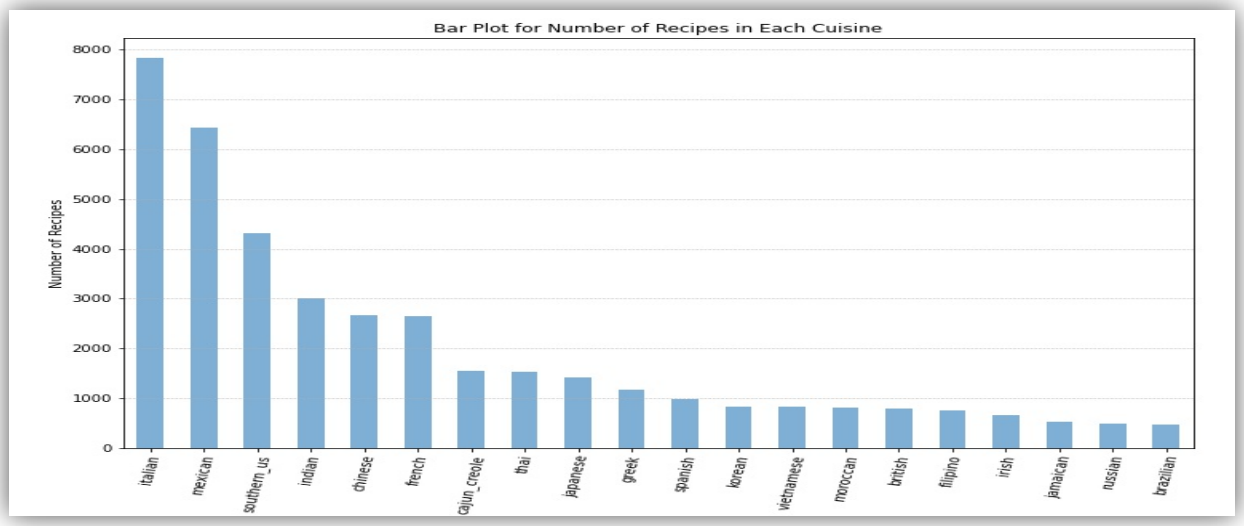

Fig. 20 – Visualization - 1

- The above plot shows the number of recipes in each of the cuisines in the training dataset.
- Out of the 39,774 recipes in the training dataset, almost 8000 recipes (maximum) belong to *'Italian'* cuisine, followed by around 6500 recipes for 'Mexican' cuisine.
- 'Jamaican', 'Russian' and 'Brazilian' are the three cuisines with the least number of datapoints with around 500 recipes each only.
- *More the recipes of a cuisine in the dataset, more distinct combinations of its common ingredients. Hence, more trained the model will be.*

NOTE: The codes for the EDA done, visualizations & the model applied by myself (Solution4) is in the 'Visualizations_And_SelfAppliedModel.ipynb' file on GitHub.

HOW MANY INGREDIENS PER MEAL?

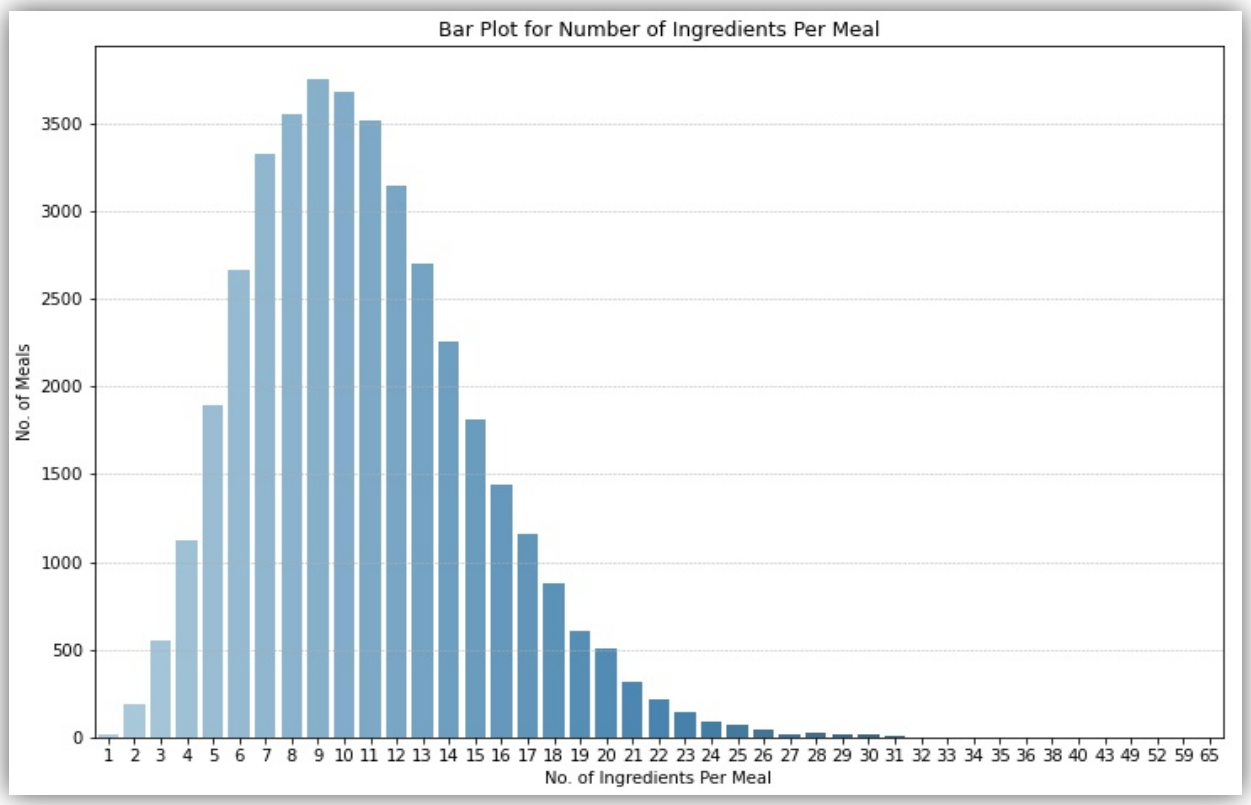

Fig. 21 – Visualization – 2

- The above plot shows the number of meals with different number of ingredients per meal.
- Around 15,000 out of the 39,774 meals in the training dataset are cooked with 8 to 11 distinct ingredients. *It stands out to give a mean of 10 ingredients per meal.*
- However, the plot also shows that a few meals also require up to 30 distinct ingredients.

WHICH ARE THE MOST USED INGREDIENTS OVER?

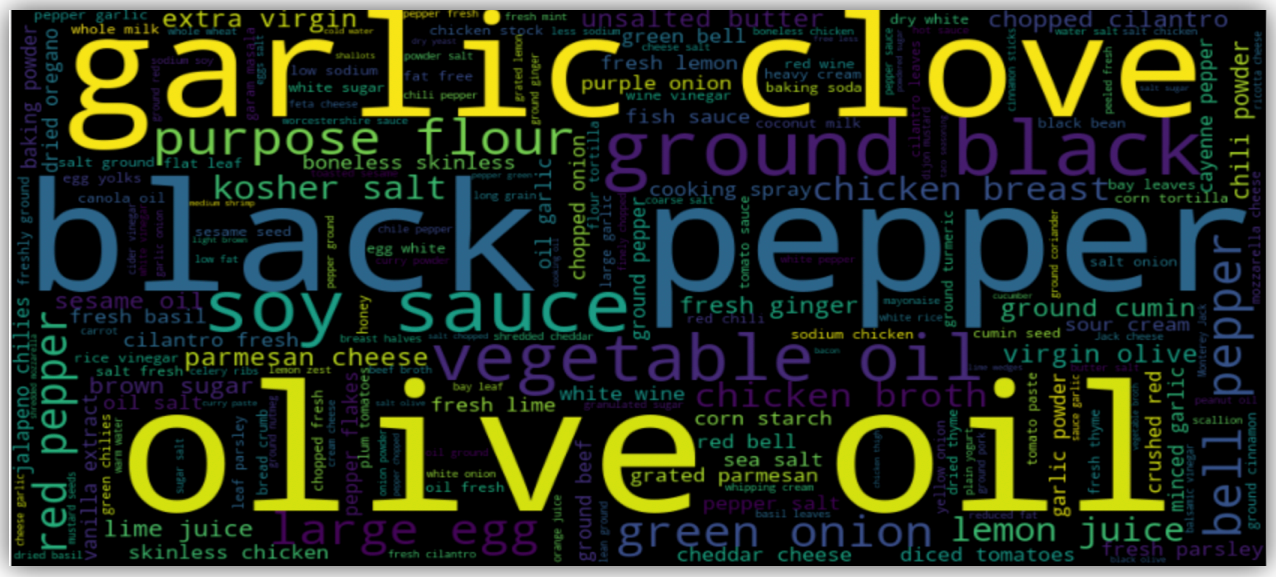

Fig. 22 – Visualization - 3

HOW MANY UNIQUE INGREDIENTS PER CUISINE?

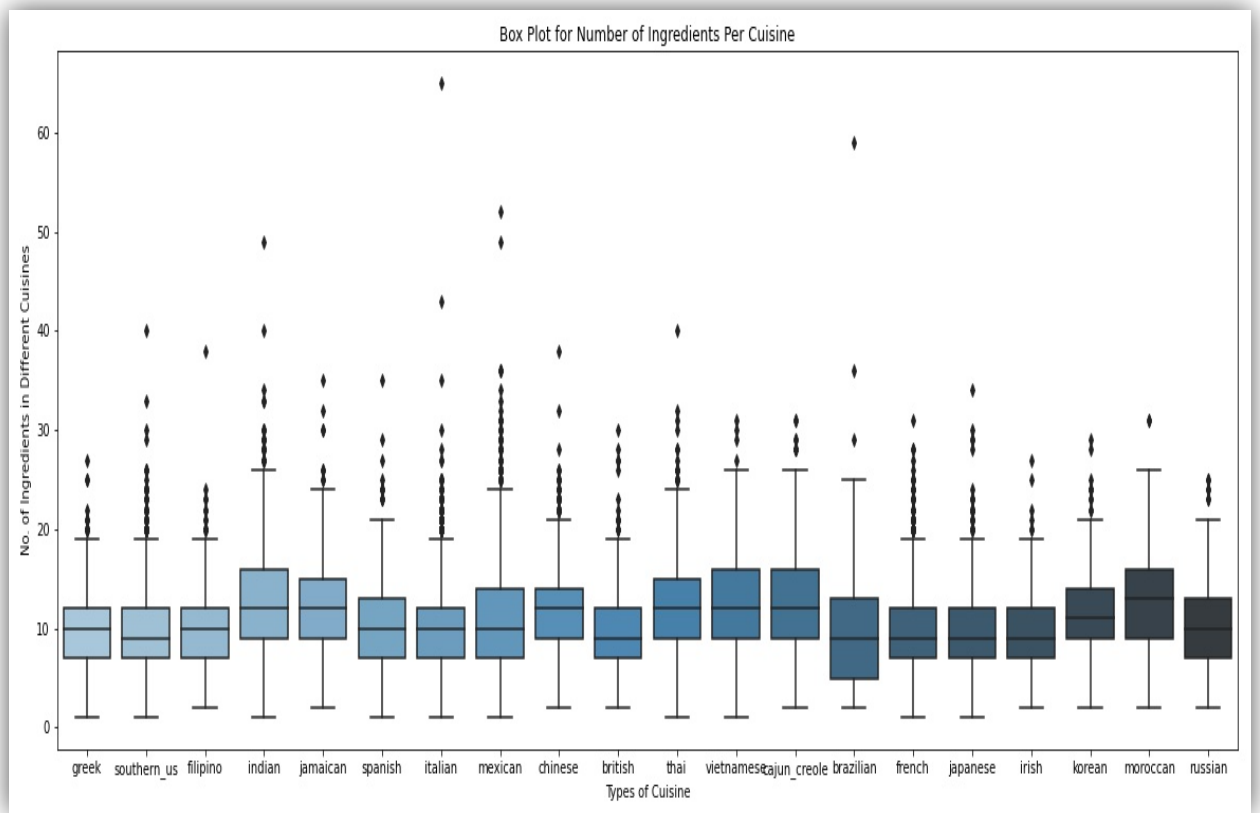

Fig. 23 – Visualization - 4

- The above box plot shows the number of different ingredients per cuisine.
- By looking at the plot, the median number of ingredients for all the cuisines may be approximated to a range of 10 to 15.
- There are also outliers (we can say those above 40 ingredients) which need to be removed from the dataset when we apply any model ourselves and carry out the analysis.

NUMBER OF AVERAGE INGREDIENTS PER CUISINE?

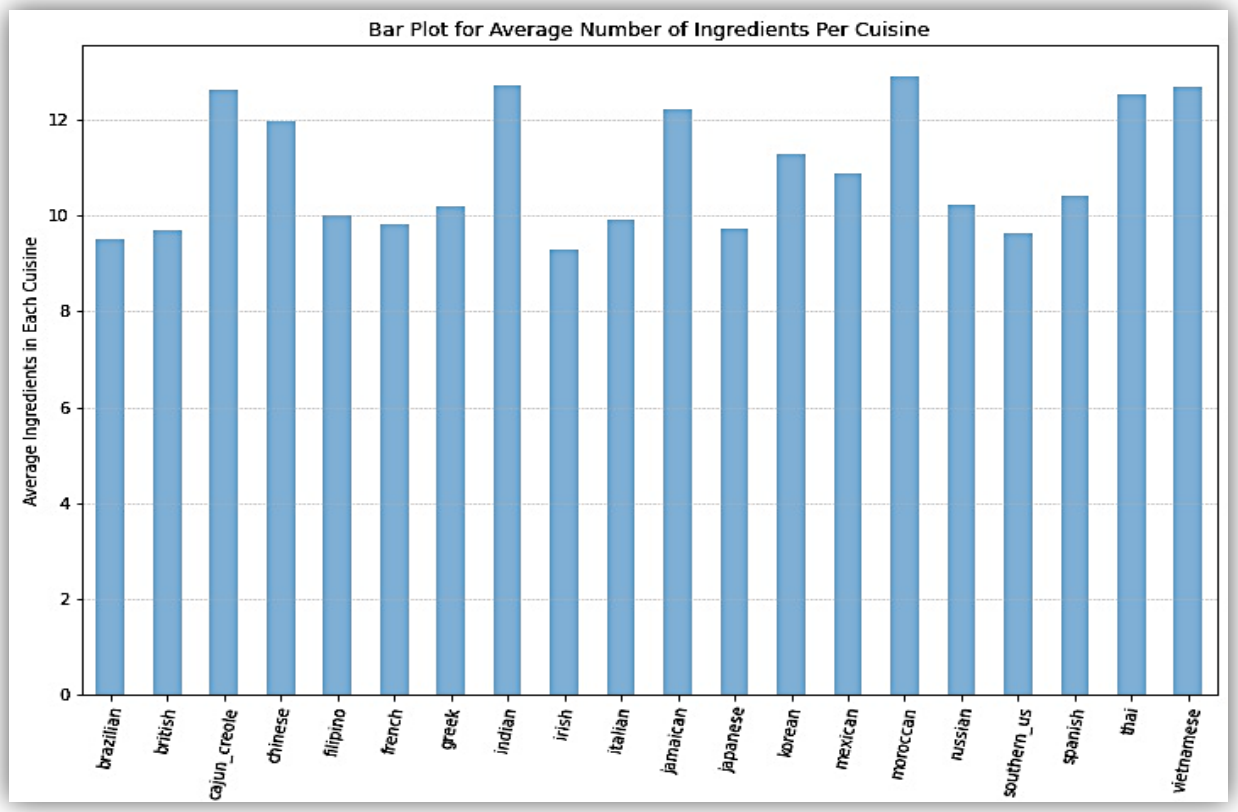

Fig. 24 – Visualization - 5

- The above bar plot shows the average number of different ingredients per cuisine.
- As per the plot the cuisines of Morocco, India and Vietnam require the highest number of ingredients to cook a meal.
- On an average each cuisine at least requires 9 ingredients for its meal to be cooked.

APPLIED OWN MODEL (SOLUTION4)

- The user imports the train and test datasets using 'json' python library. This way the data is imported in JSON format which is later converted to a dataframe *(Fig. 2)*.
- Followed by the visualizations presented in the previous section.
- As part of the **EDA,** created the following python functions to achieve some specific tasks before applying the model:

1.

```
def get_ingredients(dataset):
    output_dict = {}
    for i in range(dataset.shape[0]):
        for j in dataset["ingredients"][i]:
            if j in output_dict.keys():
                output_dict[j] += 1
            else:
                output_dict[j] = 1
    return output_dict
```

Fig. 25 – Solution4: Function to return a dictionary of unique ingredients with their frequency of occurrence.

2.

```
def find_missing_ingredients(first, second):
    missing = []
    for i in second.keys():
        if i not in first.keys():
            missing.append(i)
    print('Number of Missing Ingredients:', len(missing))
    for i in missing:
        first[i] = 0
    print('Number of Ingredients After Adding From the Other Dataset:', len(first))
```

Fig. 26 – Solution4: Function to add missing ingredients to a dataset and set their freq. of occurrence to 0.

3. This function prepares the dataset in a format required to apply the model. It inputs a dataframe having all ingredients as features and recipes as data points and sets a particular column to 1 if ingredient is present in the recipe, else sets 0.
So, the size of the dataframe created is *no. of recipes * no. of distinct ingredients.*

```
def prep_data(ingredients, dataset):
    for i in ingredients.keys():
        dataset[i] = np.zeros(len(dataset))
    for i in range(len(dataset)):
        for j in dataset['ingredients'][i]:
            dataset[j].iloc[i] = 1
```

Fig. 27 – Solution4: Function to prepare data before applying the models.

```
df_train = df_train.drop('size',axis = 1)
df_train.head()
```

	id	cuisine	ingredients	romaine lettuce	black olives	grape tomatoes	garlic	pepper	purple onion	seasoning	...	bitter orange juice	Franks Wings Sauce	dried ziti	frozen peeled prawns	lovage
0	10259	greek	[romaine lettuce, black olives, grape tomatoes...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0
1	25693	southern_us	[plain flour, ground pepper, salt, tomatoes, g...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

Fig. 28 – Solution4: Format of the Training Dataset Before Applying the Model ***(39774 rows * 7140 columns)***

```
X = df_train.drop(['id', 'ingredients', 'cuisine'], axis = 1)
y = df_train['cuisine']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)
logistic_regressor = LogisticRegression()
logistic_regressor.fit(X_train, y_train)
```

Fig. 29 – Solution4: Splitting the Data, Applying the Model & Predicting the Cuisine

- On applying the **Logistic Regressor**, I achieved a testing accuracy of around **78.2%.**

- ## WHY IS MY MODEL PERFORMING BETTER THAN THE OTHER THREE SOLUTIONS?

  - **My model overcame the main shortcoming which was to provide the model with the data in a format which best fits it (as shown in Fig. 28),** a dataframe having **all ingredients (from both train & test datasets)** as features and recipes as data points with a column set to 1 if that ingredient is in the recipe, else set to 0.

  - Initially, **when I applied the logistic regressor without converting the data to the above-mentioned format, I built a far less accurate model.** So, through good EDA we achieved the increase in accuracy of prediction.

  - One of the three solutions described above, use a TF-IDF vectorizer to prepare the dataset for applying the model. This step took around 25 minutes to execute on my machine but **with the format I am converting the dataset to it is needless to apply TF-IDF algo which reduces the execution time and gives better accuracy.**

MODULATING THE SIZES OF TRAINING AND TESTING DATASETS

- **Training Dataset Size = 80%, Accuracy Achieved = 78.19%**

```
X = df_train.drop(['id', 'ingredients', 'cuisine'], axis = 1)
y = df_train['cuisine']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)
logistic_regressor = LogisticRegression()
logistic_regressor.fit(X_train, y_train)

print('Training Accuracy:', logistic_regressor.score(X_train, y_train))
print('Testing Accuracy 1:', logistic_regressor.score(X_test, y_test))
print('Testing Accuracy 2:', accuracy_score(y_test, logistic_regressor.predict(X_test)))
```

```
Training Accuracy: 0.893711304566454
Testing Accuracy 1: 0.7818981772470145
Testing Accuracy 2: 0.7818981772470145
```

Fig. 30 – Solution4: Scores or Accuracy Achieved

- **Training Dataset Size = 75%, Accuracy Achieved = 77.98%**

```
X = df_train.drop(['id', 'ingredients', 'cuisine'], axis = 1)
y = df_train['cuisine']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 100)
logistic_regressor = LogisticRegression()
logistic_regressor.fit(X_train, y_train)

print('Training Accuracy:', logistic_regressor.score(X_train, y_train))
print('Testing Accuracy 1:', logistic_regressor.score(X_test, y_test))
print('Testing Accuracy 2:', accuracy_score(y_test, logistic_regressor.predict(X_test)))
```

```
Training Accuracy: 0.8943345625209521
Testing Accuracy 1: 0.7797666934835077
Testing Accuracy 2: 0.7797666934835077
```

Fig. 31 – Solution4: Scores or Accuracy Achieved

- **Training Dataset Size = 70%, Accuracy Achieved = 77.74%**

```
X = df_train.drop(['id', 'ingredients', 'cuisine'], axis = 1)
y = df_train['cuisine']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 100)
logistic_regressor = LogisticRegression()
logistic_regressor.fit(X_train, y_train)

print('Training Accuracy:', logistic_regressor.score(X_train, y_train))
print('Testing Accuracy 1:', logistic_regressor.score(X_test, y_test))
print('Testing Accuracy 2:', accuracy_score(y_test, logistic_regressor.predict(X_test)))
```

```
Training Accuracy: 0.8949750368162063
Testing Accuracy 1: 0.7774239503896757
Testing Accuracy 2: 0.7774239503896757
```

Fig. 32 – Solution4: Scores or Accuracy Achieved

- **Training Dataset Size = 65%, Accuracy Achieved = 77.3%**

```
X = df_train.drop(['id', 'ingredients', 'cuisine'], axis = 1)
y = df_train['cuisine']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.35, random_state = 100)
logistic_regressor = LogisticRegression()
logistic_regressor.fit(X_train, y_train)

print('Training Accuracy:', logistic_regressor.score(X_train, y_train))
print('Testing Accuracy 1:', logistic_regressor.score(X_test, y_test))
print('Testing Accuracy 2:', accuracy_score(y_test, logistic_regressor.predict(X_test)))
```

```
Training Accuracy: 0.89861911576993
Testing Accuracy 1: 0.7730048128726384
Testing Accuracy 2: 0.7730048128726384
```

Fig. 33 – Solution4: Scores or Accuracy Achieved

As expected, we see that with a decrease in the size of the training dataset, there is also a decrease in the accuracy achieved by the model. Below is a graph plotted to depict the same.

Fig. 34 – Solution4: Variation of Accuracy with respect to the Size of the Training Dataset

CONCLUSIONS

- On comparing the three other solutions and the model applied by myself, it is evident that good EDA plays an important part in increasing the accuracy of any model.

- As shown in the description of the model applied by me, it is recommended that a complete analysis should include multiple iterations of a model with different sets of data.

- The Logistic Regressor applied by myself was giving an accuracy of around 48% when I tried applying it without converting the data to the format to which I converted later i.e., a dataframe having all ingredients as features and recipes as data points and having the value of a particular column as 1 if that ingredient is present in the recipe, else the value is 0.

- My work on this project gives a lot of information to analyze and paves a way to do some enhancement work on this project in the future.

    - For example, a different type of classification model such as an optimized NLP process can be implemented to achieve better accuracy.

    - Another enhancement possible is the application of our use-case / project on an online recipe database rather than using static data.

REFERENCES

1. https://family.jrank.org/pages/639/Food-Food-Culture.html
2. https://www.techrepublic.com/article/machine-learning-the-smart-persons-guide/
3. https://www.kaggle.com/cristianfat/what-do-we-have-for-dinner
4. https://www.kaggle.com/samcoh223/random-forest-classifier
5. https://www.kaggle.com/ranjan1701/what-scooking-using-tf-idf-linearsvc
6. https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d
7. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
8. https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
9. https://www.kaggle.com/asmaaeltaher/what-s-cooking-ml-workshop-data-understanding