notebook

January 19, 2023

0.1 Popular Recipes Prediction

0.1.1 Business Problem

A website features new recipes on the homepage every day. On days that they feature a popular recipe, traffic increases by as much as 40%. However, it is difficult to predict in advance which recipes will be popular.

Recipes are considered to be popular if they receive a high score. The data team has collected data from previously published recipes.

0.1.2 Customer Question

The owner wants to know: - Can you use information on previously published recipes to predict whether a recipe will receive a high score?

0.1.3 Success Criteria

The owner estimates that of all low scoring recipes, they currently correctly categorize 75% of them. They want to know how your approach compares to this.

0.1.4 Dataset : recipes dataset

0.2 Data Validation

This dataset has 43092 rows, 10 columns. I have validated all variables All the columns are just as described in the data dictionary, there are 40 missing(null) values in RecipeCategory that need to be removed.

- RecipeId: Numeric, unique identifier of recipe, 43092 values, min 38 - max 540876
- Name: Character, name of recipe, 41240 values
- RecipeCategory: Character, type of recipe (e.g., Dessert, Breakfast, etc), 246 possible values. There are some missing values that need to be removed from dataset.
- Calories: Numeric, number of calories, min 0 - max 30933.4
- CholesterolContent: Numeric, amount of cholesterol in milligrams, min 0 - max 9167.2
- CarbohydrateContent: Numeric, amount of carbohydrates in grams, min 0 - max 3564.4
- SugarContent:Numeric, amount of sugar in grams, min 0 - max 2566.8
- ProteinContent: Numeric, amount of protein in grams, min 0 - max 1420.8
- RecipeServings: Numeric, number of servings, min 1- max 32767
- HighScore:Numeric, where 1 is a high score (i.e., popular), and 0 is a low score (i.e., unpopular)

I have removed all the rows with missing value (RecipeCategory) from dataset after validation. After that, this dataset has 43052rows, 10 columns.

[24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.metrics import␣
↪accuracy_score,f1_score,precision_score,recall_score,roc_auc_score
from sklearn.model_selection import GridSearchCV
```

[25]:
```
recipe = pd.read_csv("data/recipes.csv")
recipe.head()
```

[25]:
```
   RecipeId                                    Name RecipeCategory  \
0     46085          Crock Pot Baked Potato Soup  One Dish Meal
```

```
1     93832  Frittata Di Spaghetti (spaghetti Frittata)      Breakfast
2     36034                  Berries With Italian Cream        Dessert
3    329988  Pork Tenderloin Medallions With Fresh Figs      < 15 Mins
4     59886   Kaseropita (Tiropita Using Kaseri Cheese)    Savory Pies

   Calories  CholesterolContent  CarbohydrateContent  SugarContent  \
0     699.8               137.3                 46.1           1.4
1     297.1               191.8                 11.7           0.7
2     131.9                23.3                 10.3           4.4
3     203.0                74.8                  1.5           0.6
4     261.6               103.6                 20.9           0.2

   ProteinContent  RecipeServings  HighScore
0            20.9             6.0        1.0
1            12.2             8.0        1.0
2             9.1             6.0        0.0
3            23.3             4.0        1.0
4             6.7            15.0        1.0
```

[26]:
```
# There are 40 rows with missing value in RecipeCategory.
recipe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43092 entries, 0 to 43091
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   RecipeId             43092 non-null  int64
 1   Name                 43092 non-null  object
 2   RecipeCategory       43052 non-null  object
 3   Calories             43092 non-null  float64
 4   CholesterolContent   43092 non-null  float64
 5   CarbohydrateContent  43092 non-null  float64
 6   SugarContent         43092 non-null  float64
 7   ProteinContent       43092 non-null  float64
 8   RecipeServings       43092 non-null  float64
 9   HighScore            43092 non-null  float64
dtypes: float64(7), int64(1), object(2)
memory usage: 3.3+ MB
```

[27]:
```
# validate name of recipe, 41240 values, categorical variable
recipe.Name.nunique()
```

[27]: 41240

[28]:
```
# validate recipe category, 246 values, categorical variable
recipe.RecipeCategory.nunique()
```

[28]:
```
246
```

[7]:
```
#validate any negative values in numeric variables
recipe.describe()
```

[7]:

	RecipeId	Calories	CholesterolContent	CarbohydrateContent	\
count	43092.000000	43092.000000	43092.000000	43092.000000	
mean	224707.928154	353.297587	69.475757	32.844187	
std	141980.914350	405.065683	112.422309	46.428258	
min	38.000000	0.000000	0.000000	0.000000	
25%	102616.750000	164.800000	4.500000	11.600000	
50%	212333.000000	284.100000	40.700000	25.400000	
75%	336736.250000	446.725000	95.900000	43.400000	
max	540876.000000	30933.400000	9167.200000	3564.400000	

	SugarContent	ProteinContent	RecipeServings	HighScore
count	43092.000000	43092.000000	43092.000000	43092.000000
mean	12.206600	15.773125	9.517521	0.645735
std	28.304371	21.588594	224.022873	0.478296
min	0.000000	0.000000	1.000000	0.000000
25%	2.200000	3.400000	4.000000	0.000000
50%	5.500000	8.800000	6.000000	1.000000
75%	14.100000	24.300000	8.000000	1.000000
max	2566.800000	1420.800000	32767.000000	1.000000

[29]:
```
# remove missing values in RecipeCategory
recipe = recipe.dropna()
recipe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43052 entries, 0 to 43091
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   RecipeId             43052 non-null  int64
 1   Name                 43052 non-null  object
 2   RecipeCategory       43052 non-null  object
 3   Calories             43052 non-null  float64
 4   CholesterolContent   43052 non-null  float64
 5   CarbohydrateContent  43052 non-null  float64
 6   SugarContent         43052 non-null  float64
 7   ProteinContent       43052 non-null  float64
 8   RecipeServings       43052 non-null  float64
 9   HighScore            43052 non-null  float64
dtypes: float64(7), int64(1), object(2)
memory usage: 3.6+ MB
```

0.3 Exploratory Analysis

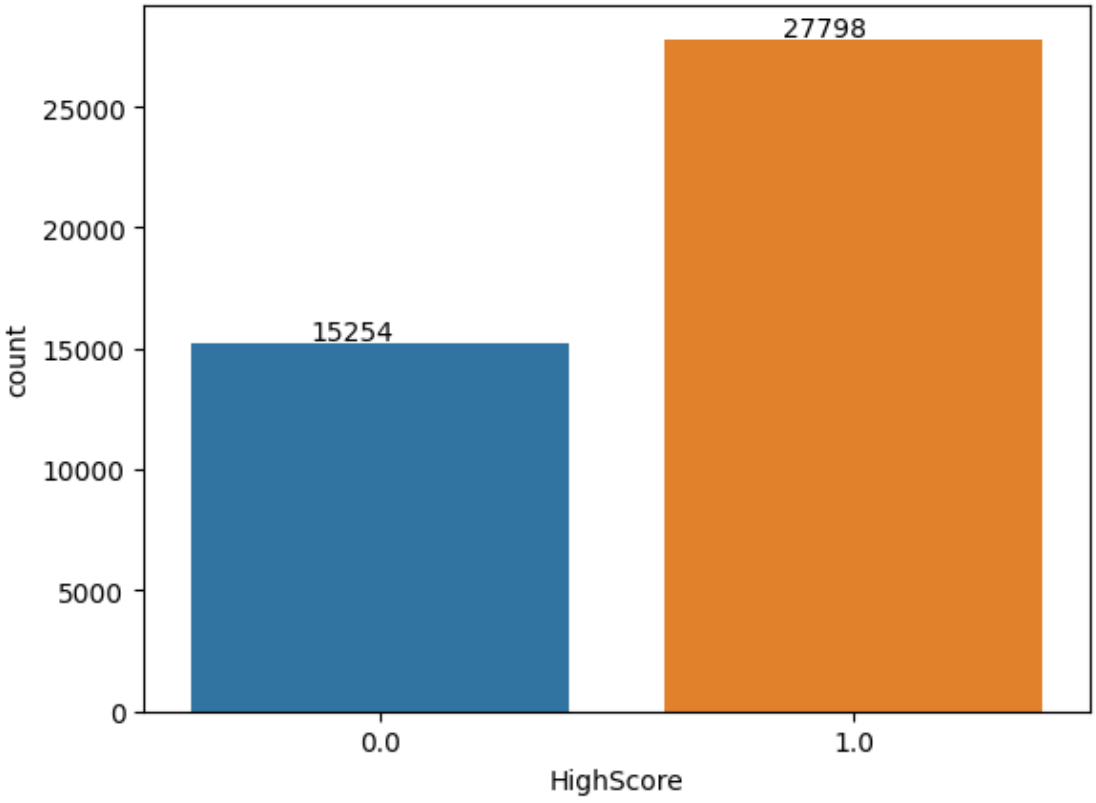
I have investigated the target variable and features of the recipe, and the relationship between target variable and features. After the analysis,I decided to apply the following changes to enable modeling:

- RecipeCategory: remove all the rows with RecipeCategory'value_count < 100.
- Apply a power transform featurewise to make data more Gaussian-like to normalize the numeric features
- Remove some outliers in numerical variablesCalories, CholesterolContent, RecipeServings.

0.3.1 Target Variable: HighScore

Since we need to classify wheather a recipe will receive a high score (1 or 0). HighScore is the target variable. - From the countplot, we can see that there are 27798 of high score(1) and 15254 of low score(0). High score is 64.56% while low score is 35.54%. - It is more important to correctly classify the category of "low score" for the owner of Tasty Bytes, since the owner should try to aviod featuring a low score recipe in the homepage. Our goals is to find correctly predict the low score(positive instances, 35.54%)

[9]:
```
ax = sns.countplot(x= 'HighScore', data = recipe)
for p in ax.patches:
   ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.25, p.
 ↪get_height()+100))
plt.show()
```



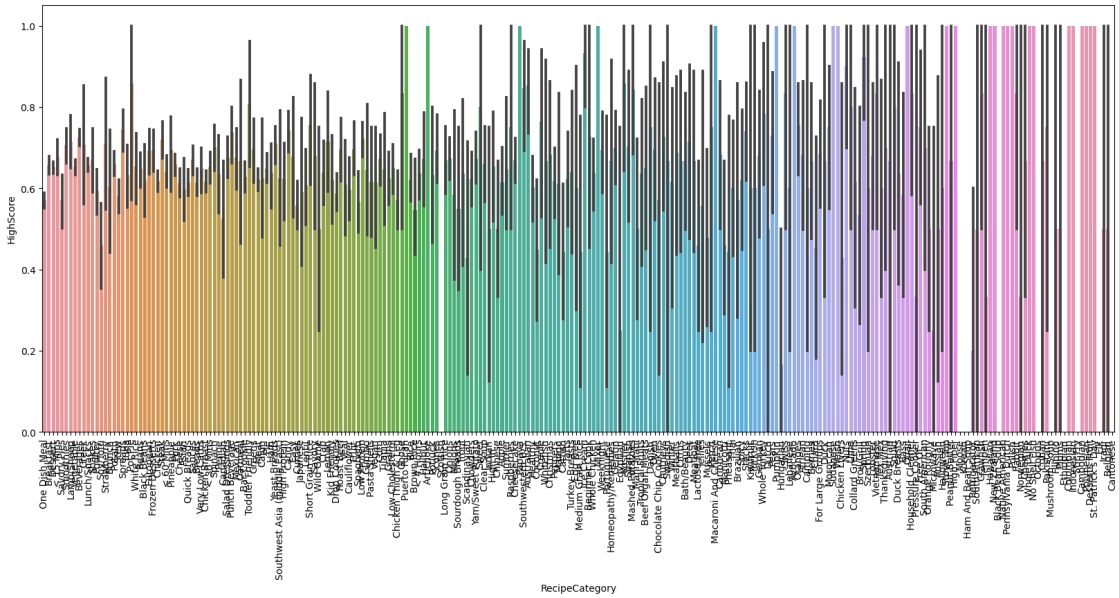
[125]:
```
np.mean(recipe.HighScore == 1)
```

[125]:
```
0.6456842887670724
```

0.3.2 Categorical variable: RecipeCatgory

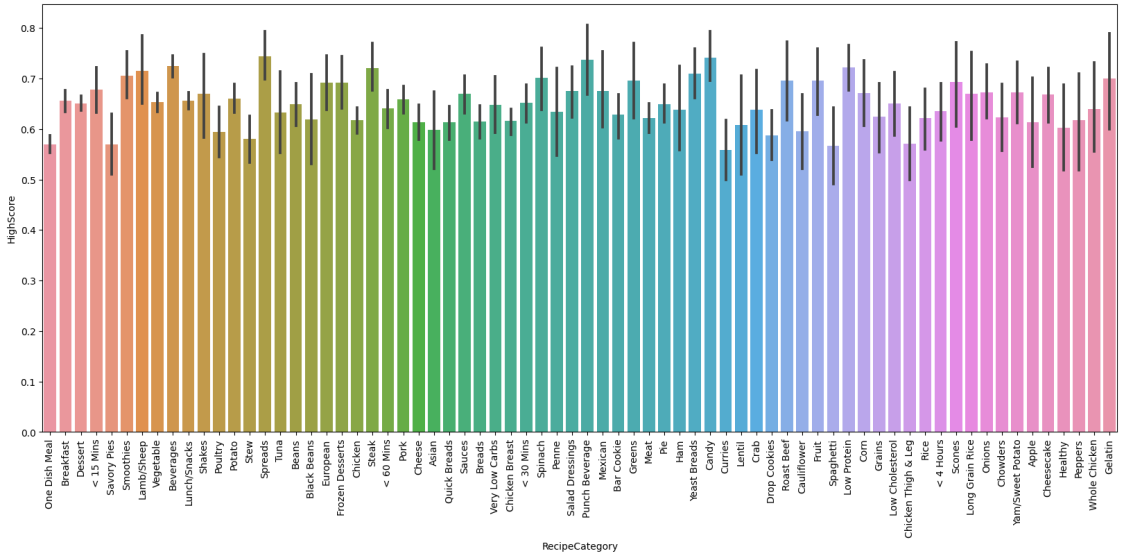
- Since there are some categories only have 1 or only a few count, it is hard to predict with small sample in one category, I removed all the categories with less than 100 count. we can see there is a difference in mean of HighScore among each recipe categories, but not very obviously.
- I did not use another categorical variable "Name"in the modeling. I have tried text mining for Name, and found that for both highscore/lowscore, the top 5 hotest recipes are the same "Chicken,Potato, Salad, Soup,Chocolate", it is hard to use name to predict.

[10]:
```
#Categorical variable:
plt.figure(figsize=(20,8))
sns.barplot(x = 'RecipeCategory', y = 'HighScore', data = recipe )
plt.xticks(rotation = 90)
plt.show()
```



[30]:
```
# remove RecipeCategory with count < 100
counts = recipe.RecipeCategory.value_counts()
recipe = recipe[recipe.RecipeCategory.isin(counts.index[counts >= 100])]
```

[12]:
```
# After removing RecipeCategory with count < 100
plt.figure(figsize=(20,8))
sns.barplot(x = 'RecipeCategory', y = 'HighScore', data = recipe )
plt.xticks(rotation = 90)
plt.show()
```

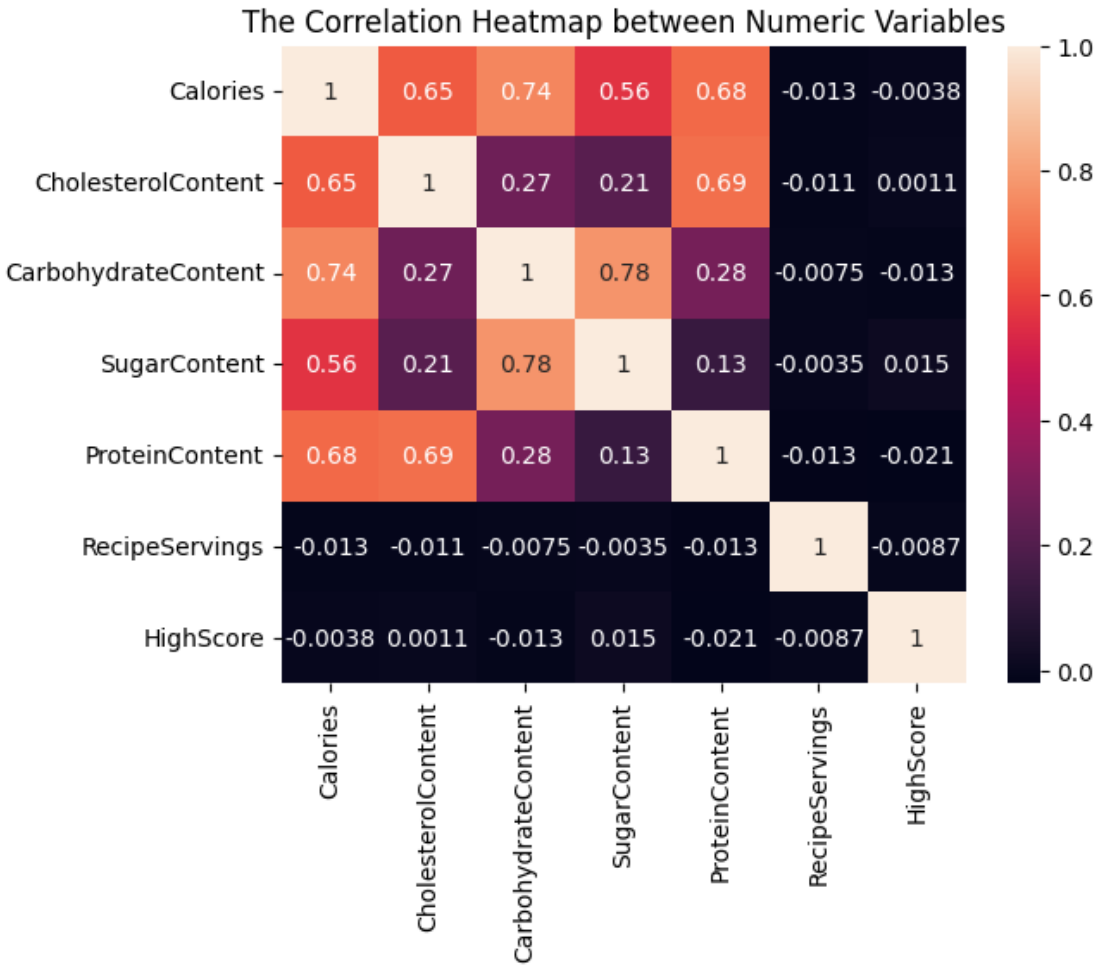


0.3.3 Numeric Variables:Calories, CholesterolContent, CarbohydrateContent, SugarContent,ProteinContent, RecipeServings

From the heatmap below, we could see that the linear relationships between numerical variables and target variable are very week. It is very hard to use these numeric varibles to classify the target value. Among all numerical varibles, 'CarbohydrateContent', SugarContent','ProteinContent' seems to have a little bigger linear relationship with target variable (-0.013, 0.015, 0.021).

[13]:
```
recipe1 =recipe[['Calories', 'CholesterolContent',␣
 ↪'CarbohydrateContent','SugarContent','ProteinContent',␣
 ↪'RecipeServings','HighScore']]
sns.heatmap(recipe1.corr(),annot=True).set(title='The Correlation Heatmap␣
 ↪between Numeric Variables')
```

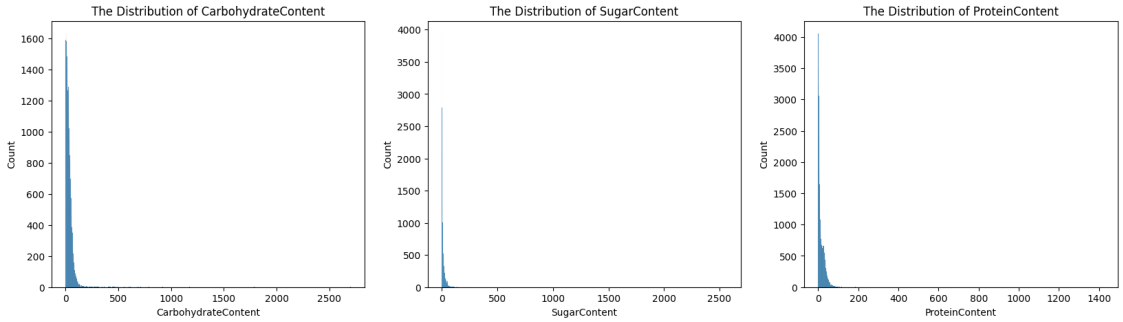
[13]:
```
[Text(0.5, 1.0, 'The Correlation Heatmap between Numeric Variables')]
```



Distribution of CarbohydrateContent, SugarContent, ProteinContent - Since'CarbohydrateContent', SugarContent','ProteinContent' are more related to 'HighScore', I checked their distributions. From the histogram below, their distributions are skewed. - we should apply a power transform featurewise to make data more Gaussian-like.

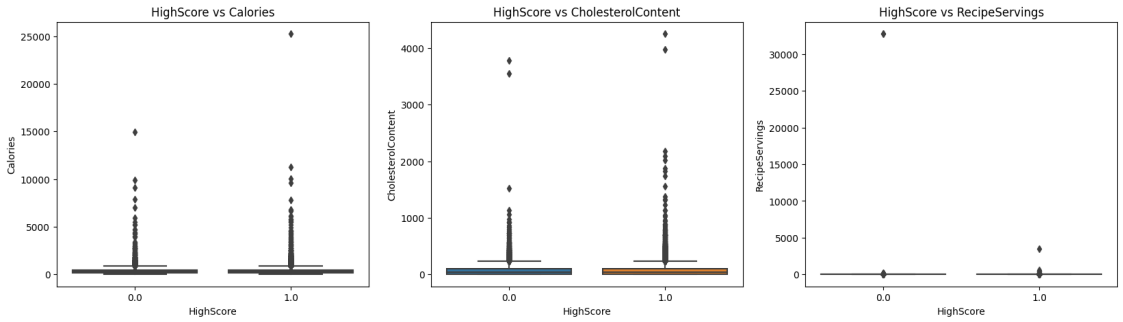
[130]:
```
fig, axes = plt.subplots(1,3,figsize=(20,5))
sns.histplot(recipe.CarbohydrateContent,ax=axes[0]).set(title='The Distribution␣
↪of CarbohydrateContent')
sns.histplot(recipe.SugarContent,ax=axes[1]).set(title='The Distribution of␣
↪SugarContent')
sns.histplot(recipe.ProteinContent,ax=axes[2]).set(title='The Distribution of␣
↪ProteinContent')
```

[130]:
```
[Text(0.5, 1.0, 'The Distribution of ProteinContent')]
```



Relationship between Calories, CholesterolContent, RecipeServings and HighScore - Since the linear relationship between Calories, CholesterolContent, RecipeServings and HighScore from the heatmap above is close to 0, I decided to make boxplot to further investigate their relationship. - From the boxplots below, it is hard to tell the differenc between high score and low score. There are strong overlap between two categoies. - There are many outliers in the dataset, so I decided to remove some outliers from these 3 variable. - From the heatmap, we could see that the relationship has been improved a little bit.

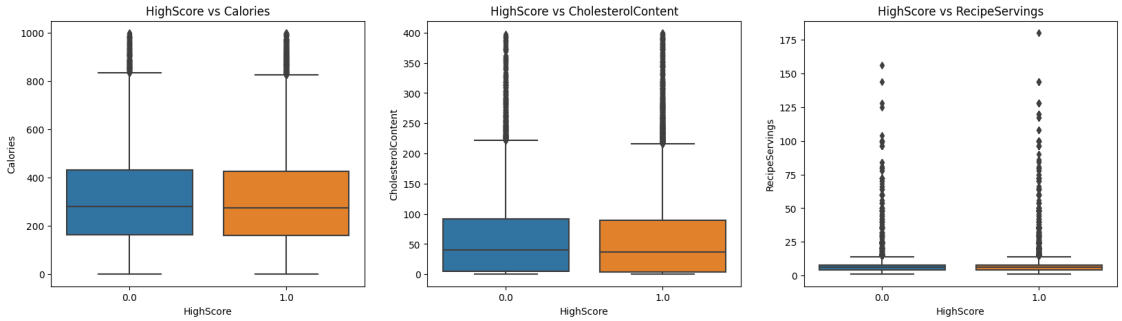
[131]:
```
fig, axes = plt.subplots(1,3,figsize=(20,5))
sns.boxplot(x =recipe.HighScore,y=recipe.Calories,  ax=axes[0]).
 ↪set(title='HighScore vs Calories')
sns.boxplot(x =recipe.HighScore,y=recipe.CholesterolContent, ax=axes[1]).
 ↪set(title='HighScore vs CholesterolContent')
sns.boxplot(x =recipe.HighScore,y=recipe.RecipeServings, ax=axes[2]).
 ↪set(title='HighScore vs RecipeServings');
```



[31]:
```
# remove outliers
recipe = recipe[recipe.Calories < 1000]
recipe = recipe[recipe.CholesterolContent < 400]
recipe = recipe[recipe.RecipeServings < 200]
```

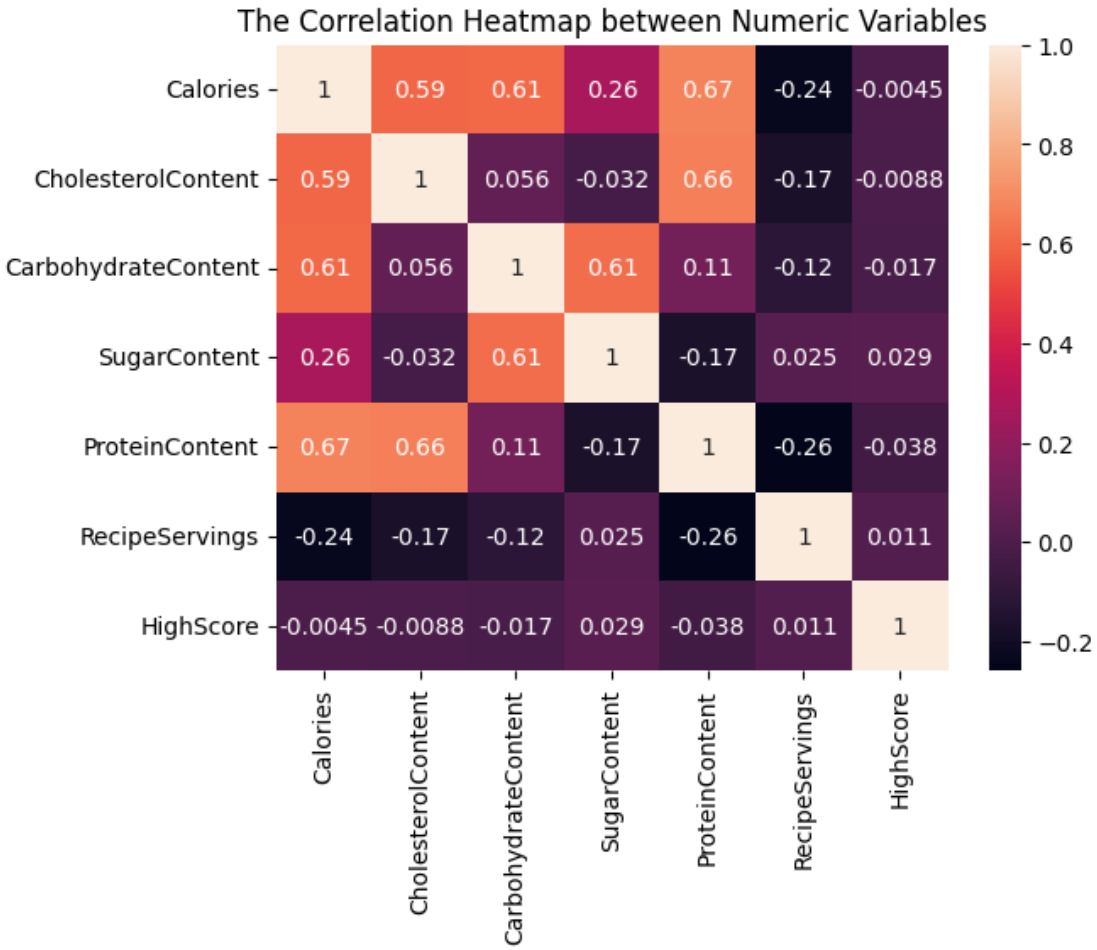
[133]:
```
fig, axes = plt.subplots(1,3,figsize=(20,5))
```

```
sns.boxplot(x =recipe.HighScore,y=recipe.Calories,  ax=axes[0]).
 ↪set(title='HighScore vs Calories')
sns.boxplot(x =recipe.HighScore,y=recipe.CholesterolContent, ax=axes[1]).
 ↪set(title='HighScore vs CholesterolContent')
sns.boxplot(x =recipe.HighScore,y=recipe.RecipeServings, ax=axes[2]).
 ↪set(title='HighScore vs RecipeServings');
```



[134]:
```
recipe1 =recipe[['Calories', 'CholesterolContent',␣
 ↪'CarbohydrateContent','SugarContent','ProteinContent',␣
 ↪'RecipeServings','HighScore']]

sns.heatmap(recipe1.corr(),annot=True).set(title='The Correlation Heatmap␣
 ↪between Numeric Variables')
```

[134]:
```
[Text(0.5, 1.0, 'The Correlation Heatmap between Numeric Variables')]
```



0.4 Model Fitting & Evaluation

- Predicting the HighScore is a classification problem in machine learning. I am choosing the Decision Tree as a base model because it is easy to interpret, and the two comparison models I am choosing are Logistic Regression and Random Forest. Logistic Regression is also easy to interpret. Random Forest is an ensemble model, most of the time it outperforms the Decision Tree model.
- For the evaluation, I am choosing AUC and Recall to evaluate the model. AUC measures the performance of the model at distinguishing between the positive and negative classes. Recall is a measure of the classifier's completeness; the ability of a classifier to correctly find all positive instances (here it is the low score). For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives.

Prepare Data for Modelling

To enable modelling, we chose 'RecipeCategory','Calories', 'CholesterolContent','CarbohydrateContent', 'SugarContent', 'ProteinContent','RecipeServings'as features, 'HighScore' as target variables. I also have made the following changes:

- Normalize the numeric features
- Convert the categorical variables into numeric features
- Split the data into a training set and a test set

[32]:
```
### prepare data for modeling

labelencoder = LabelEncoder()
recipe['RecipeCategory']=labelencoder.fit_transform(recipe['RecipeCategory'])

feature_cols = ['RecipeCategory','Calories',␣
 ↪'CholesterolContent','CarbohydrateContent', 'SugarContent',␣
 ↪'ProteinContent','RecipeServings']
X = recipe[feature_cols] # Features
y = recipe['HighScore'] # Target variable

# define the scaler
scaler = PowerTransformer()
scaler = MinMaxScaler()
# fit and transform the train set
X[['Calories', 'CholesterolContent','CarbohydrateContent', 'SugarContent',␣
 ↪'ProteinContent','RecipeServings']] = scaler.fit_transform(X[['Calories',␣
 ↪'CholesterolContent','CarbohydrateContent', 'SugarContent',␣
 ↪'ProteinContent','RecipeServings']])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[136]:
```
X.describe()
```

[136]:

	RecipeCategory	Calories	CholesterolContent	CarbohydrateContent	\
count	38467.000000	38467.000000	38467.000000	38467.000000	
mean	32.570177	0.314266	0.148923	0.128673	
std	19.222704	0.201808	0.169943	0.103715	
min	0.000000	0.000000	0.000000	0.000000	
25%	16.000000	0.161164	0.010511	0.049766	
50%	37.000000	0.276911	0.094344	0.107614	
75%	46.000000	0.428171	0.224975	0.180774	
max	69.000000	1.000000	1.000000	1.000000	

	SugarContent	ProteinContent	RecipeServings
count	38467.000000	38467.000000	38467.000000
mean	0.049386	0.119941	0.039476
std	0.065417	0.122530	0.049246
min	0.000000	0.000000	0.000000
25%	0.009726	0.027211	0.016760
50%	0.024315	0.068878	0.027933
75%	0.062776	0.193027	0.039106

```
max         1.000000         1.000000         1.000000
```

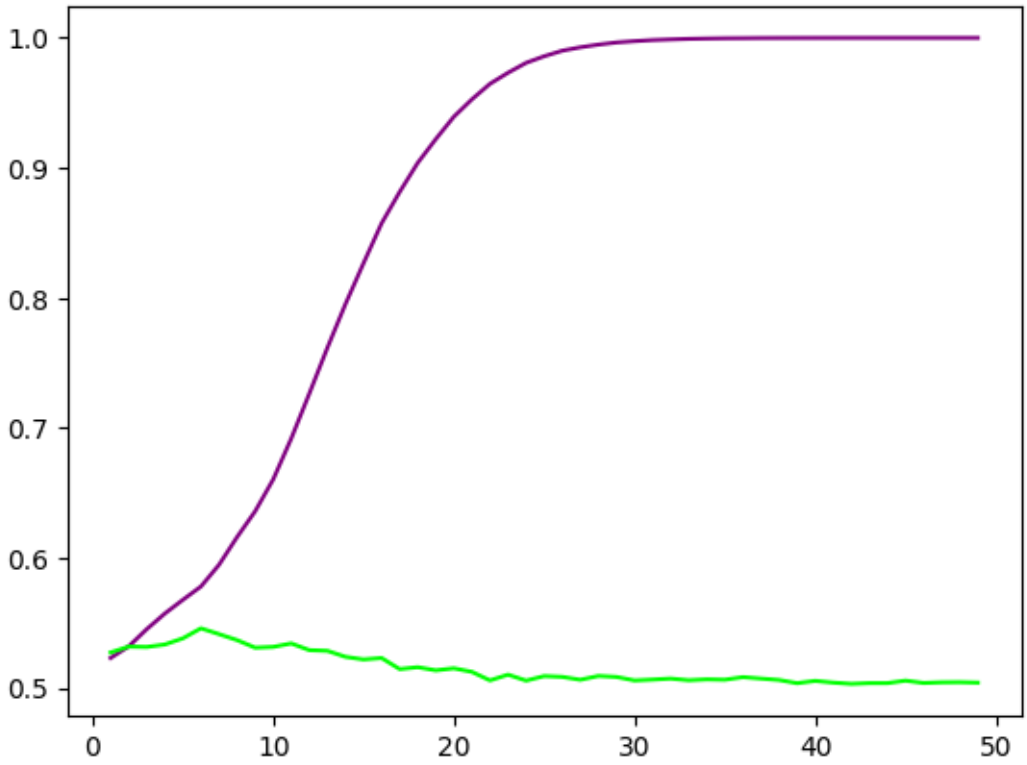
0.4.1 Decision Tree Model

[33]:
```
# Decision Tree
# Create Decision Tree classifer object
dt = DecisionTreeClassifier(criterion="entropy", max_depth=6)

# Train Decision Tree Classifer
dt = dt.fit(X_train,y_train)
```

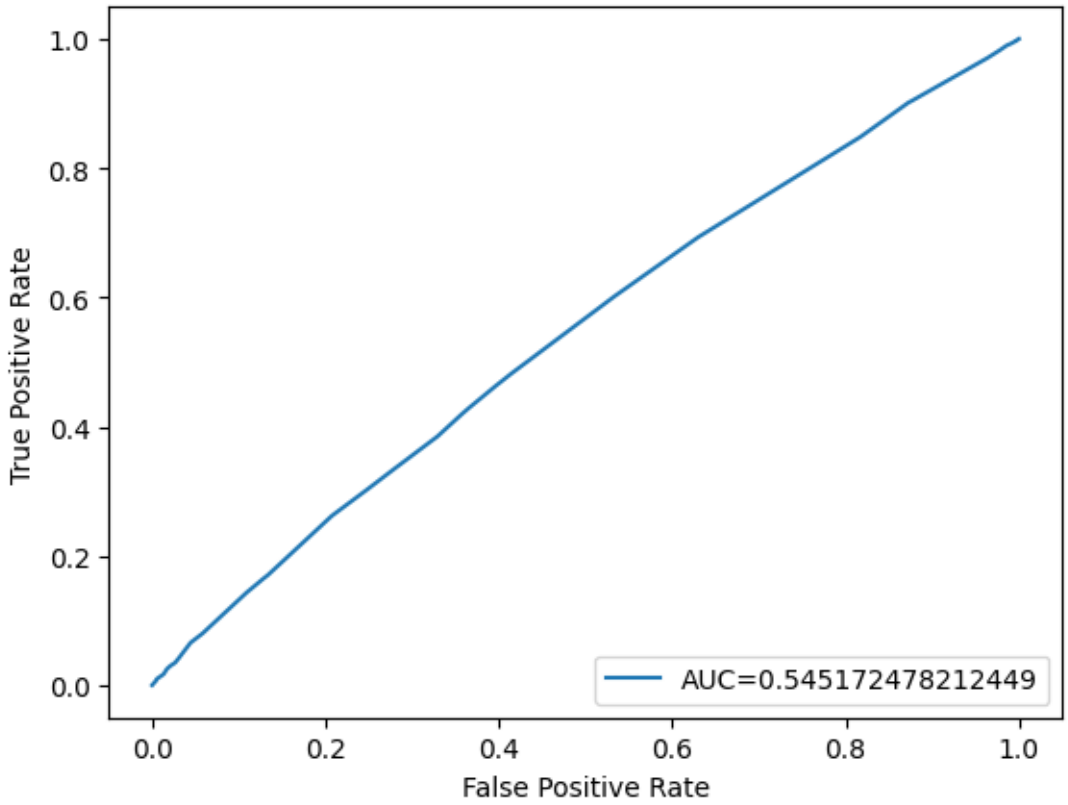
Finding the best parameter for Desicion Tree Model

[34]:
```
train_score = []
test_score = []
max_score = 0
max_pair = (0,0)

for i in range(1,50):
    tree = DecisionTreeClassifier(max_depth=i,random_state=42)
    tree.fit(X_train,y_train)
    y_pred = tree.predict_proba(X_train)[:,1]
    y_pred_t = tree.predict_proba(X_test)[:,1]
    train_score.append(metrics.roc_auc_score(y_train,y_pred))
    test_score.append(metrics.roc_auc_score(y_test, y_pred_t))
    test_pair = (i,metrics.roc_auc_score(y_test,y_pred_t))
    if test_pair[1] > max_pair[1]:
        max_pair = test_pair

fig, ax = plt.subplots()

ax.plot(np.arange(1,50), train_score, label = "roc_auc_score",color='purple')
ax.plot(np.arange(1,50), test_score, label = "roc_auc_score",color='lime')
print(f'Best max_depth is: {max_pair[0]} \nroc_auc_score is: {max_pair[1]}')
```

```
Best max_depth is: 6
roc_auc_score is: 0.5458047637121048
```



[35]:
```
#define metrics
y_pred_proba = dt.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

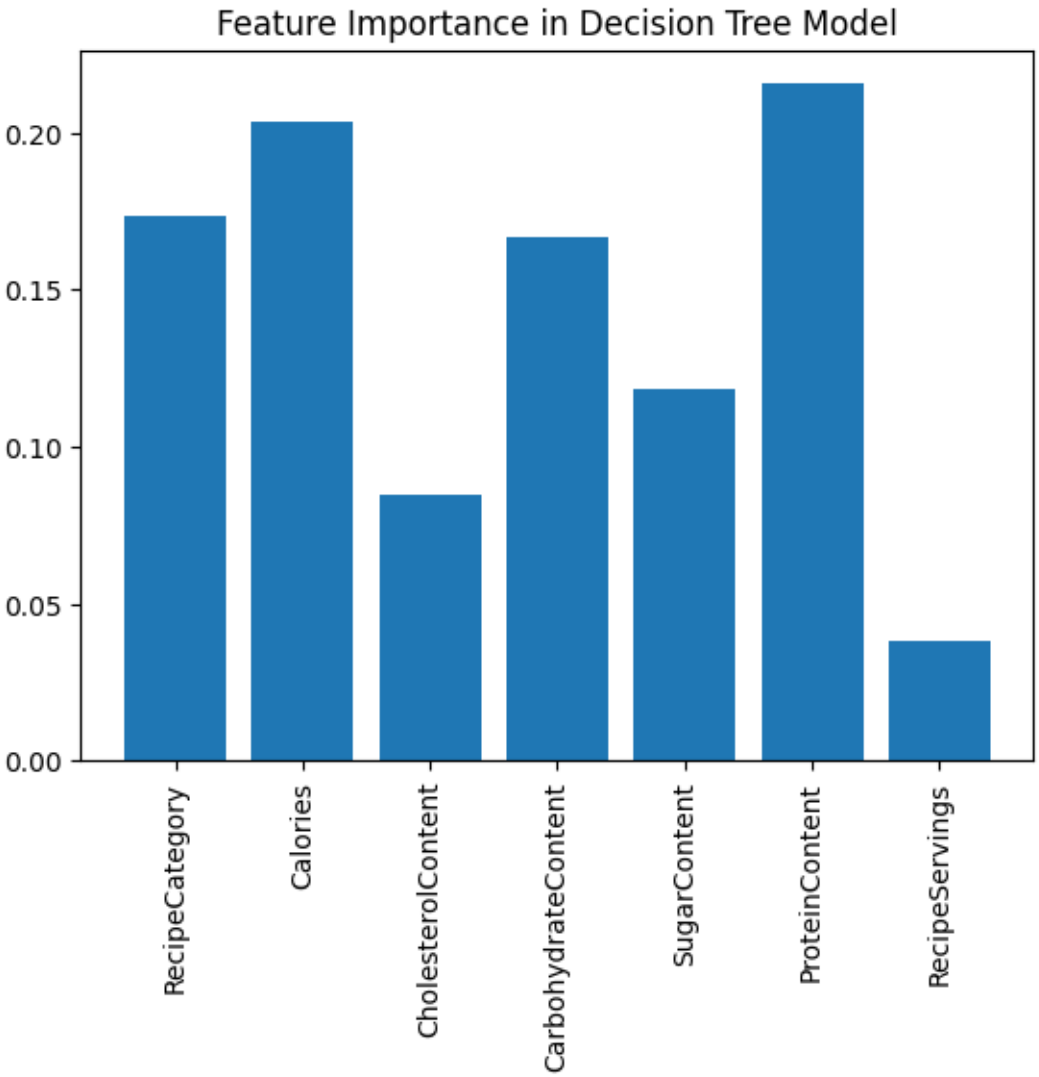


[36]:
```
#Predict the response for test dataset
# select the right threshold to make sure the recall of "0" category is around␣
 ↪80 and accuaracy is 0.45.
threshold = 0.688
y_pred = (dt.predict_proba(X_test)[:, 1] > threshold).astype('float')

dt_matrix = metrics.confusion_matrix(y_test, y_pred)
print(dt_matrix)
dt_report = metrics.classification_report(y_test,y_pred)
print(dt_report)
```

```
[[3238  847]
 [5502 1954]]
              precision    recall  f1-score   support

         0.0       0.37      0.79      0.50      4085
         1.0       0.70      0.26      0.38      7456

    accuracy                           0.45     11541
   macro avg       0.53      0.53      0.44     11541
weighted avg       0.58      0.45      0.42     11541
```

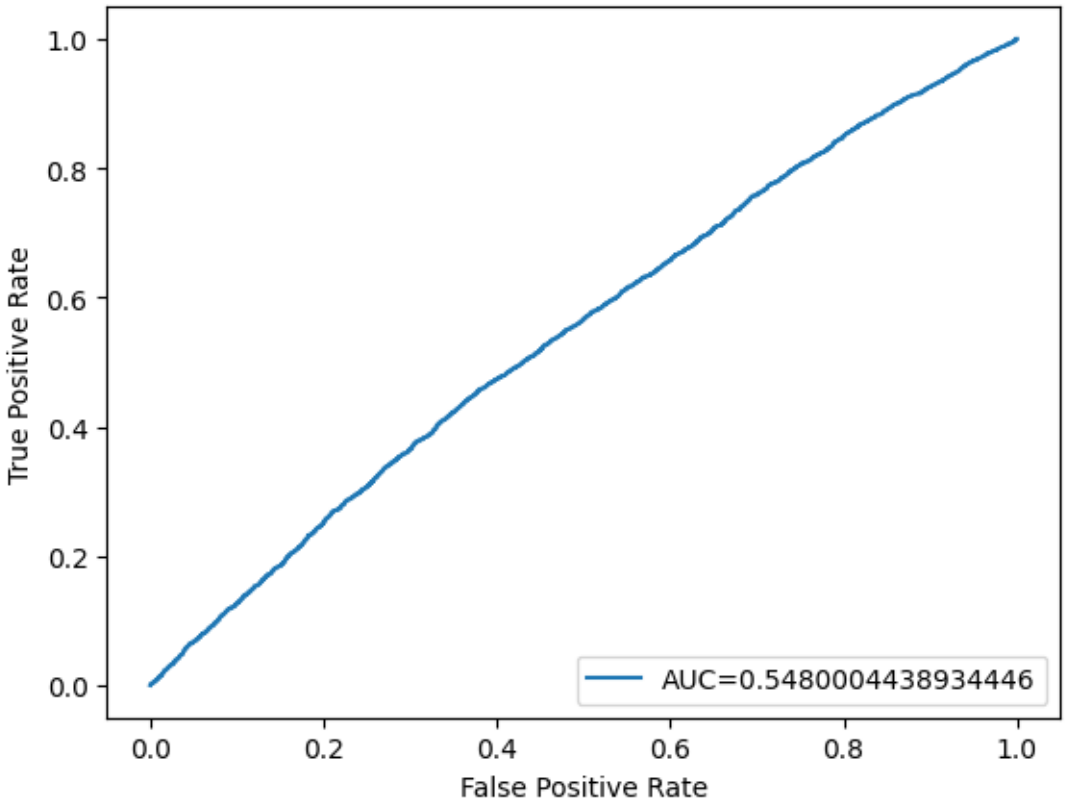
[37]:
```
resultdict = {}
for i in range(len(feature_cols)):
    resultdict[feature_cols[i]] = dt.feature_importances_[i]

plt.bar(resultdict.keys(),resultdict.values())
plt.xticks(rotation='vertical')
plt.title('Feature Importance in Decision Tree Model')
```

[37]:
```
Text(0.5, 1.0, 'Feature Importance in Decision Tree Model')
```



0.4.2 Logistic Regression Model

[38]:
```
lg = LogisticRegression(random_state= 42)
lg.fit(X_train, y_train)

#define metrics
y_pred_proba = lg.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



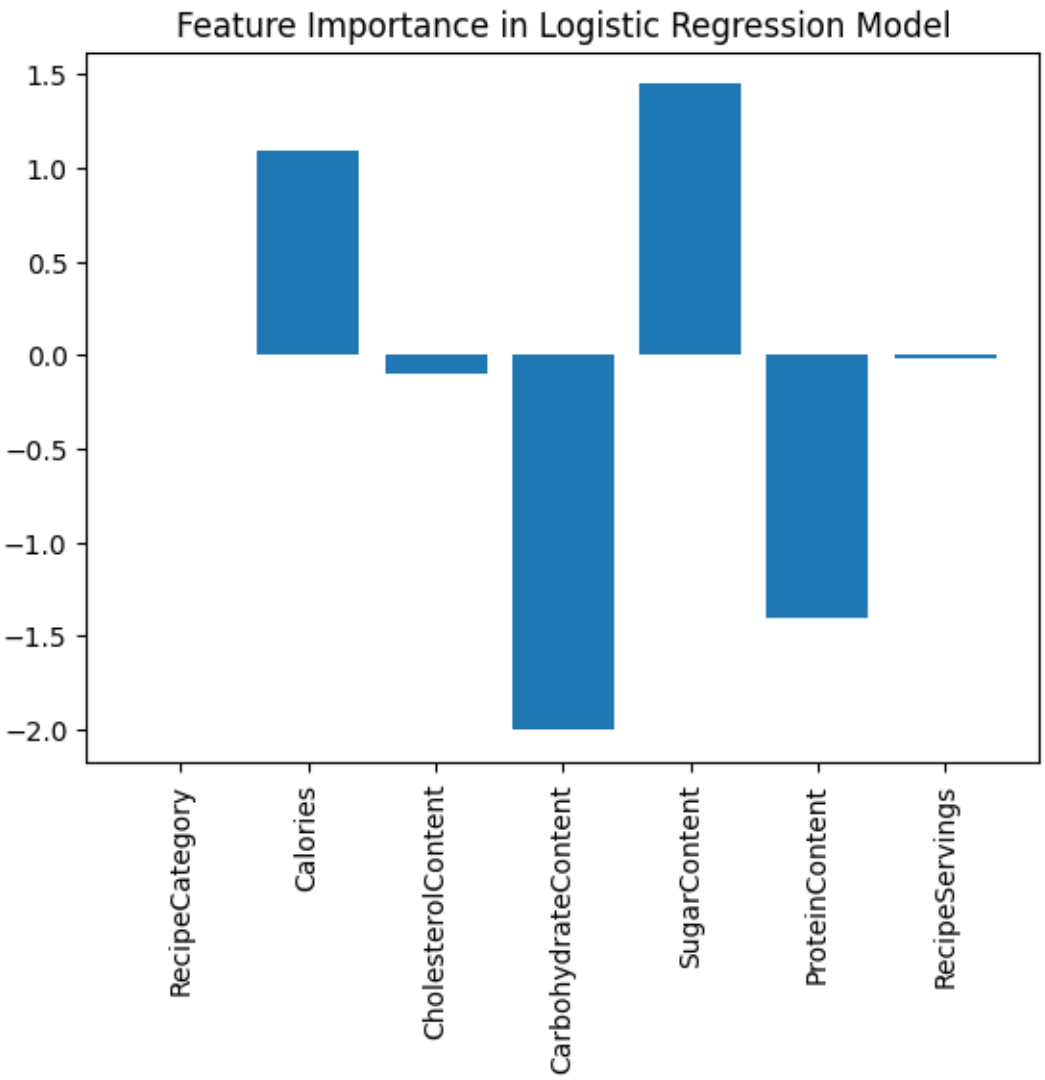
[39]:
```
# select the right threshold to make sure the recall of "0" category is around␣
 ↪80 and accuaracy is 0.45.
threshold = 0.666
y_pred2 = (lg.predict_proba(X_test)[:, 1] > threshold).astype('float')
```

```
lg_matrix = metrics.confusion_matrix(y_test, y_pred2)
print(lg_matrix)

lg_report = metrics.classification_report(y_test,y_pred2)
print(lg_report)
```

```
[[3262  823]
 [5560 1896]]
              precision    recall  f1-score   support

         0.0       0.37      0.80      0.51      4085
         1.0       0.70      0.25      0.37      7456

    accuracy                           0.45     11541
   macro avg       0.53      0.53      0.44     11541
weighted avg       0.58      0.45      0.42     11541
```

[40]:
```
resultdict = {}
for i in range(len(feature_cols)):
    resultdict[feature_cols[i]] = lg.coef_[0][i]
plt.bar(resultdict.keys(),resultdict.values())
plt.xticks(rotation='vertical')
plt.title('Feature Importance in Logistic Regression Model')
```

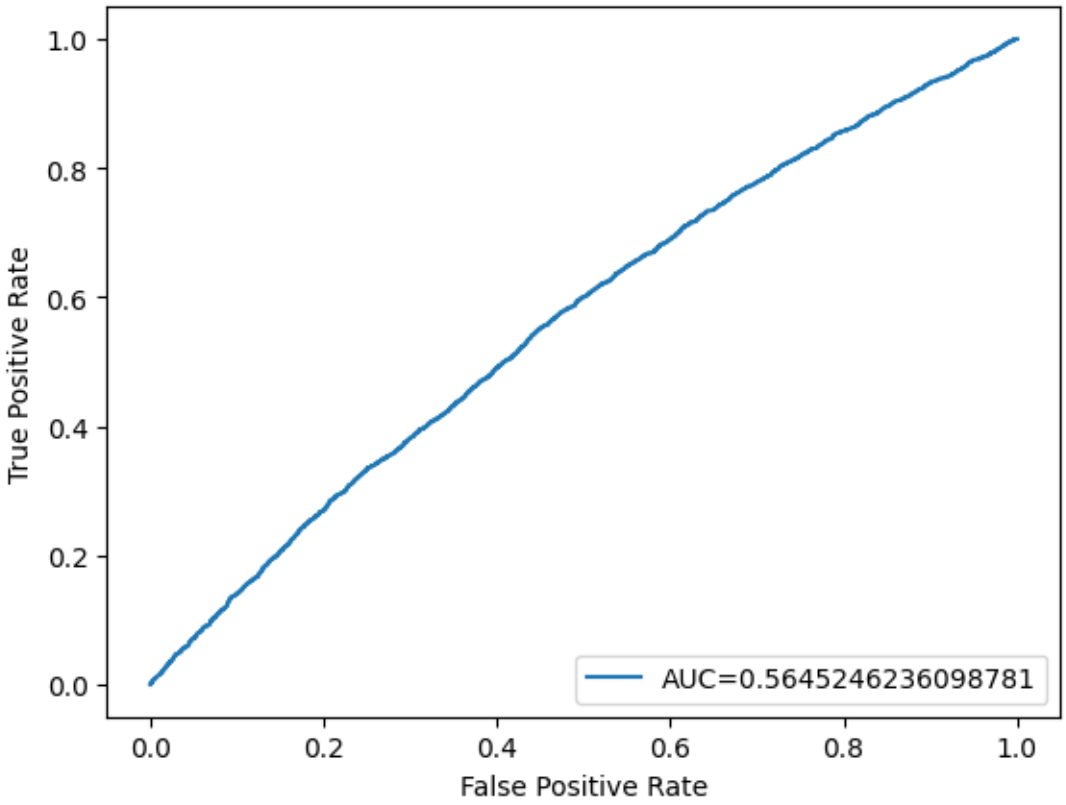
[40]: Text(0.5, 1.0, 'Feature Importance in Logistic Regression Model')



0.4.3 Random Forest Model

[41]:
```
rf = RandomForestClassifier(random_state = 42, n_estimators=1000,max_features =␣
↪2, bootstrap = True, max_depth=10,criterion='entropy')
rf.fit(X_train, y_train)
#define metrics
y_pred_proba = rf.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Hyperparameters Tuning for Random Forest Model

[199]:
```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [10, 20],
    'max_features': [2, 3],
    'n_estimators': [500, 1000]
}
# Create a based model
rf_t = RandomForestClassifier(random_state = 42)
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf_t, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2, scoring = 'roc_auc')
```

[200]:
```
# Fit the grid search to the data.
grid_search.fit(X_train, y_train)
```

```
grid_search.best_params_
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=500; total
time=   9.3s
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=500; total
time=   9.3s
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=1000; total
time=  18.7s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=500; total
time=  12.7s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=500; total
time=  12.7s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=1000; total
time=  25.4s
[CV] END bootstrap=True, max_depth=20, max_features=2, n_estimators=500; total
time=  15.5s
[CV] END bootstrap=True, max_depth=20, max_features=2, n_estimators=500; total
time=  15.2s
[CV] END bootstrap=True, max_depth=20, max_features=2, n_estimators=1000; total
time=  29.8s
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=500; total
time=   9.3s
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=1000; total
time=  18.7s
[CV] END bootstrap=True, max_depth=10, max_features=2, n_estimators=1000; total
time=  18.6s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=500; total
time=  12.7s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=1000; total
time=  25.3s
[CV] END bootstrap=True, max_depth=10, max_features=3, n_estimators=1000; total
time=  25.6s
[CV] END bootstrap=True, max_depth=20, max_features=2, n_estimators=500; total
time=  15.2s
[CV] END bootstrap=True, max_depth=20, max_features=2, n_estimators=1000; total
time=  29.6s
```

[200]:
```
{'bootstrap': True, 'max_depth': 10, 'max_features': 2, 'n_estimators': 1000}
```

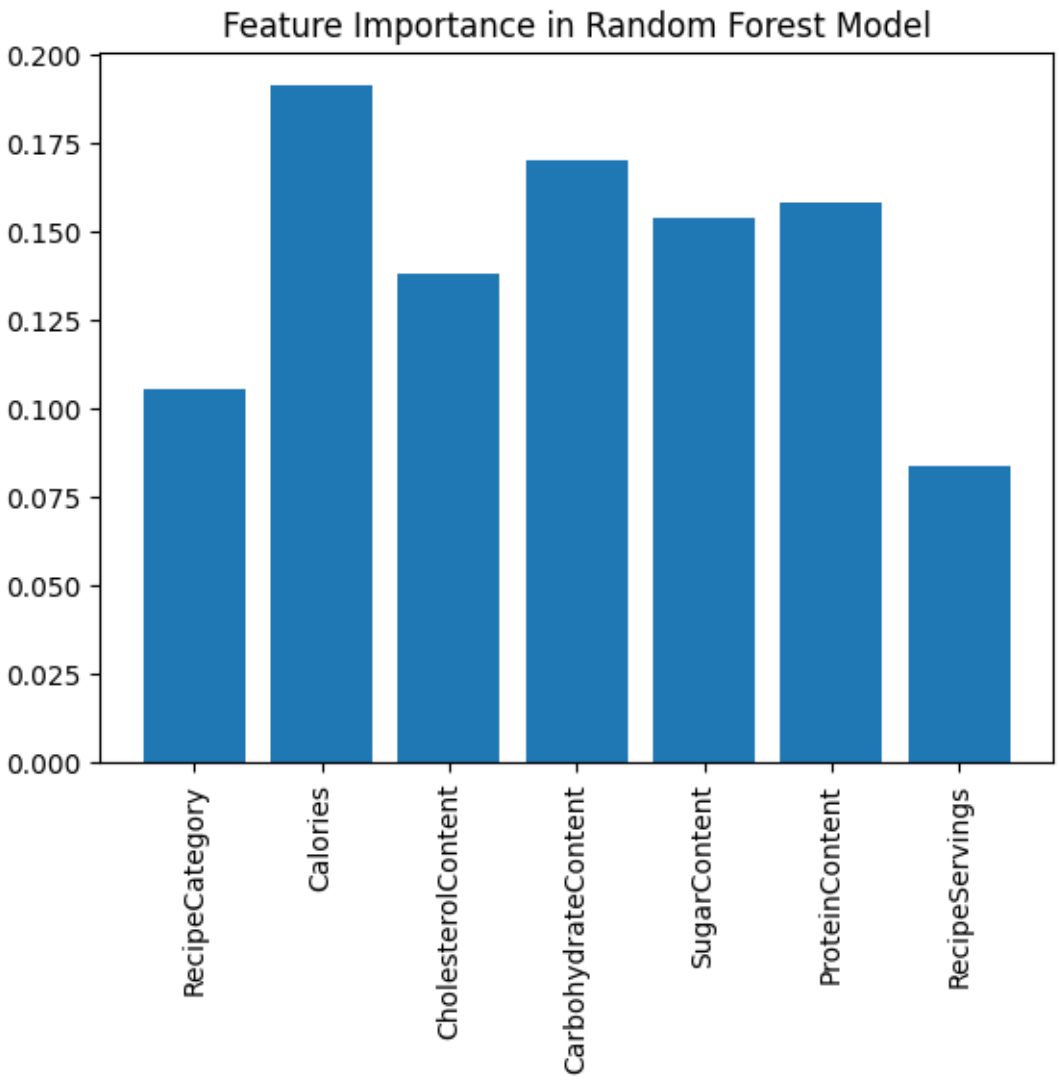
[42]:
```
# select the right threshold to make sure the recall of "0" category is around␣
 ↪80 and accuaracy is 0.45.
threshold = 0.674
y_pred3 = (rf.predict_proba(X_test)[:, 1] > threshold).astype('int')
rf_matrix = metrics.confusion_matrix(y_test, y_pred3)
print(rf_matrix)
```

```
rf_report = metrics.classification_report(y_test,y_pred3)
print(rf_report)
```

```
[[3377  708]
 [5661 1795]]
              precision    recall  f1-score   support

         0.0       0.37      0.83      0.51      4085
         1.0       0.72      0.24      0.36      7456

    accuracy                           0.45     11541
   macro avg       0.55      0.53      0.44     11541
weighted avg       0.60      0.45      0.42     11541
```

[44]:
```
resultdict = {}
for i in range(len(feature_cols)):
    resultdict[feature_cols[i]] = rf.feature_importances_[i]

plt.bar(resultdict.keys(),resultdict.values())
plt.xticks(rotation='vertical')
plt.title('Feature Importance in Random Forest Model')
```

[44]: Text(0.5, 1.0, 'Feature Importance in Random Forest Model')



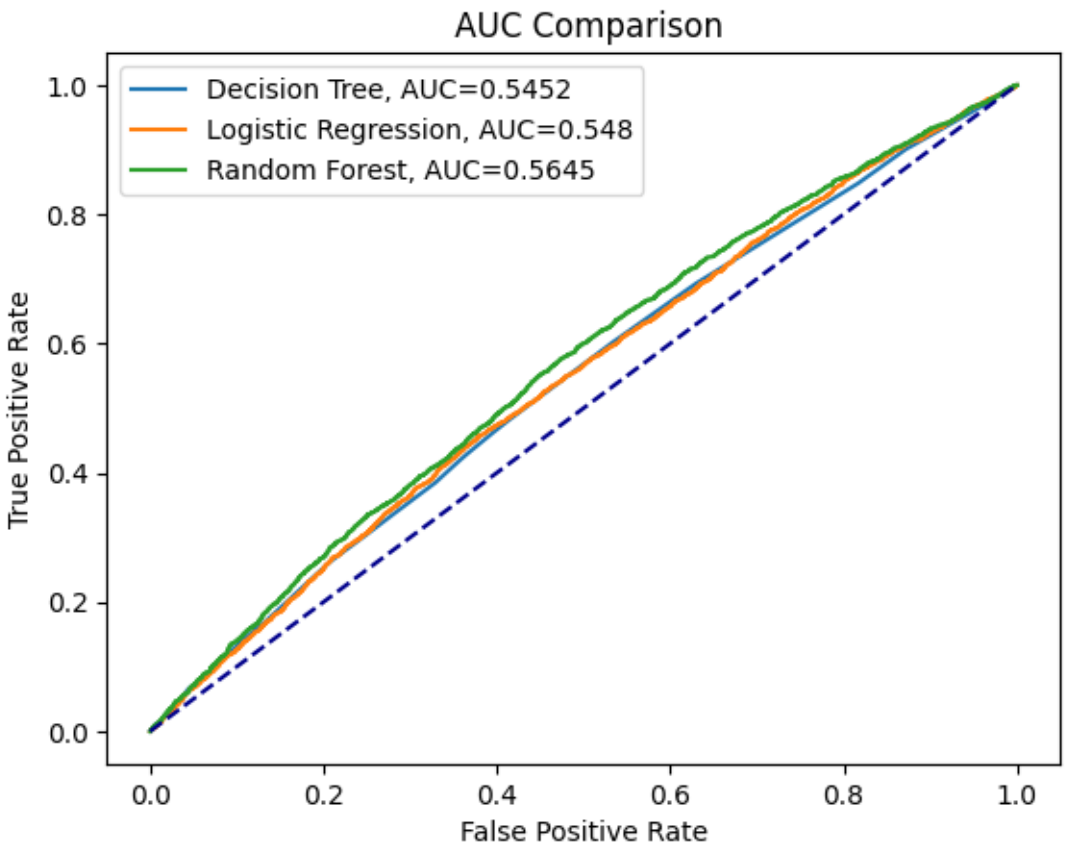
0.5 Results

- The AUC of Decision Tree, Logisitc Regression, and Random Forest are 0.5453, 0.548 and 0.5645, meaning that **Random Forest model fits the features best**, while three models are not very good to do the classification.
- The Recall of Decision Tree, Logisitc Regression, and Random Forest are 0.79, 0.80, 0.83 meaning that **Random Forest model has the best ability to classify low score correctly with 83% while keeping the accuary as the same of 0.45.** Recall is the business KPI in this case.

0.5.1 Model Comparison

[43]:
```
#set up plotting area
plt.figure(0).clf()
#fit decisiom tree model and plot ROC curve
y_pred = dt.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Decision Tree, AUC="+str(auc))

#fit logistic regression model and plot ROC curve
y_pred = lg.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))

#fit random forest model and plot ROC curve

y_pred = rf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.title(" AUC Comparison")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
#add legend
plt.legend()
```

[43]:
```
<matplotlib.legend.Legend at 0x7f9cac048190>
```

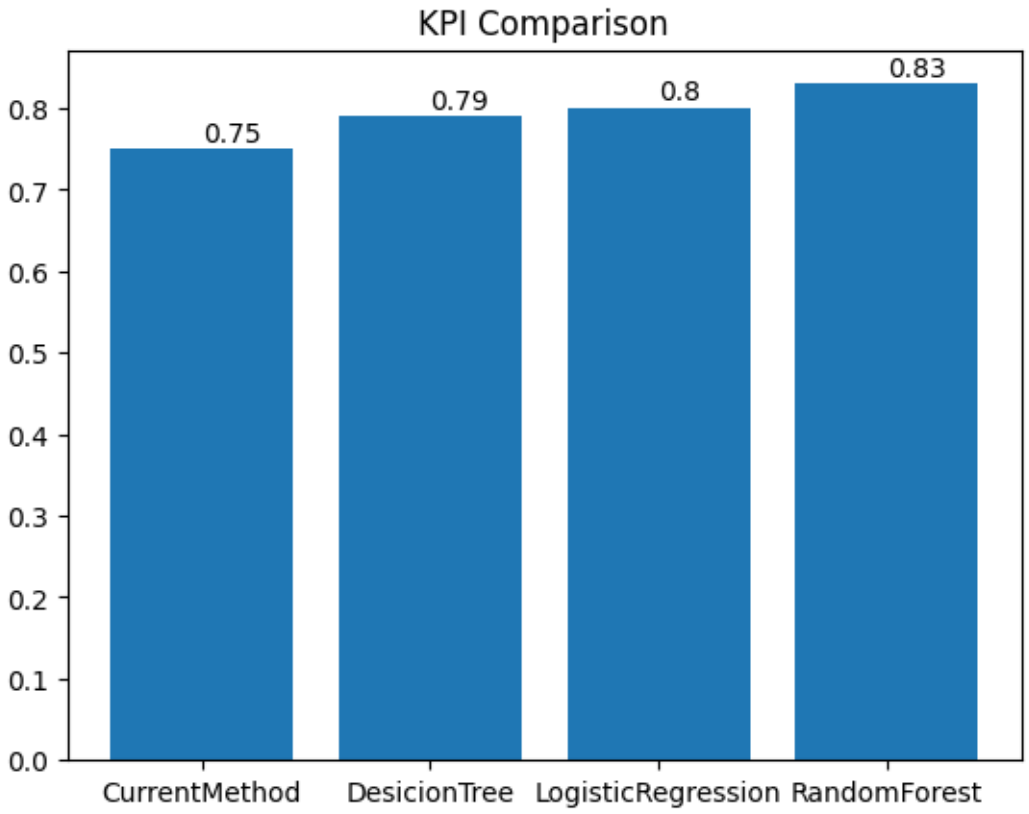


0.5.2 Evaluation by Business Criteria

- The KPI is the model metric of recall.
- The owner estimates that of all low scoring recipes, they currently correctly categorize 75% of them.
- In the selected model: Random Forest, for the low scoring recipes, it could correctly categize 83% of them while Decision Tree is with 79% and logistic Regression is with 80%.
- There is a tradeoff between recall and precision. We also need to consider the precision. In this project, the recall (the ability of a classifier correctly all the low score category) is more important for owner.

[257]:
```
x = ["CurrentMethod","DesicionTree","LogisticRegression","RandomForest" ]
y = [0.75, 0.79,0.80,0.83]
ax = plt.bar(x, y)
plt.title("KPI Comparison")
for i in range(len(x)):
    plt.text(i,y[i]+0.01,y[i])
```

```
plt.show()
```



0.6 Recommendation

- To help Tasty Bytes to better classify low scoring recipes, we can deploy this **Random Forest Model** into production. By implementing this model, about 83% of the low scoring recipe will be correctly categorized, which has been improved from 75% compared with the current method. This will help Tasty Bytes to avoid featuring low scoring recipes in the homepage more correctly.
- The recipe score not only depends on internal features of a recipe like nutritional values, such as calories, cholesterol content, carbohydrate content, sugar content, protein content, and recipe category, recipe servings, but also depends on its image, its taste, customer review text, time taken to prepare the recipe, number of reviews and number of ingredients used etc. The classification ability of this model is limited. I recommend improving the model with more high-quality data.

However, the classification ability of this model is limited, to implement and improve the model, I will consider the following steps:

- Collecting more data, e.g. recipe image, reviews, no. of ingredients, no. of reviews
- Feature Engineering, e.g reduce the categories in recipe category, create more meaningful

features from the variable, such as time taken to prepare the recipe, consistant category,such as: (“American food”, “Chinese food”, “Japanese food”…) , (“Breakfast”, “Lunch”, “Snack”…)

- Try some deep learning models to deal with recipe images, text mining with reviews.
- Fine tune hyperparameters: it could take a lot time (such as 1 hour in this platform) to tune the hyperparamters, I just tried a small subset of hyperparameters in this case. The company could try more hypermeters to select the best one to improve model performance.