Preço ideal da receita 1028

Autor: André Heidemann Iarozinski

1. Descrição do problema e análise de variáveis

Neste case técnico, o objetivo é analisar a base de dados de uma empresa e buscar formas de encontrar o preço ideal para o prato **1028**. Nas imagens abaixo, estão as duas sheets com os dados referentes a todas as encomendas realizadas durante 10 semanas:

_id	areaId	category	createdDate	customerId	mealId	orderDate	orderId	organizationId	price	discountTotal	totalValue	ref	subcategory
60c48a6cc31ac901134	5d13407be54b0000cf	PLATE	2021-06-12T11:20:58.438Z	5f1046153560e72205d05d4d	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c48a6cc31ac9011342ba3f	5f0f364bfef79802010fe5a4	7	1,05	5,95	1121	sb1
60c3a97edf69d5010cc	5d13407be54b0000cf	PLATE	2021-06-11T19:21:11.501Z	60b3f2752b9bfe0119934b54	5eb9687e9ac4fc00fcfa7e6c	2021-06-12T00:00:00Z	60c3a8cec31ac9011342b9f1	60b3f2762b9bfe0119934b55	6,9	4	2,9	1043	sb2
60c48e20ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-12T11:36:28.834Z	6065dd62b7fac1033e96f797	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c48e20ed8b3a011a4ad574	6065dd63b7fac1033e96f798	7	7	0	1121	sb1
60c47c81ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-12T10:24:32.43Z	5e7b96711e73c901081df88c	5eb9687e9ac4fc00fcfa7e6c	2021-06-12T00:00:00Z	60c479bf8c18610125e5402b	60b9e29475908b012a15c663	6,9	0	6,9	1043	sb2
60c4886ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-12T11:12:37.036Z	5e9f6a945cdf6900f5e80e53	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c4887ec31ac9011342ba3c	5ffd7352422a180105d12f40	7	0	7	1121	sb1
60c3bb6e8c18610125	5d13407be54b0000cf	PLATE	2021-06-11T20:38:40.719Z	60b15a0791977f012bf9e419	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c3bb6e8c18610125e53fe9	60b93a9698bb340118a55941	7	7	0	1121	sb1
60c48b408c18610125	5d13407be54b0000cf	PLATE	2021-06-12T11:30:11.789Z	60c48a52df69d5010ccb0696	5f6e0bf8739d89011380b09	2021-06-12T00:00:00Z	60c48b408c18610125e5404c	60c48a53df69d5010ccb0697	7,9	0	7,9	3053	sb3
60c3bb6e8c18610125	5d13407be54b0000cf	PLATE	2021-06-11T20:38:40.719Z	60b15a0791977f012bf9e419	5da5757dd1c60b00db2cb548	2021-06-12T00:00:00Z	60c3bb6e8c18610125e53fe9	60b93a9698bb340118a55941	6,45	6,45	0	1075	sb4
60c48b408c18610125	5d13407be54b0000cf	PLATE	2021-06-12T11:30:11.789Z	60c48a52df69d5010ccb0696	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c48b408c18610125e5404c	60c48a53df69d5010ccb0697	7	4,9	2,1	1121	sb1
60c485fdc31ac901134	5d13407be54b0000cf	PLATE	2021-06-12T11:01:48.449Z	5e6cf256d6616700fb61b56e	5da5757dd1c60b00db2cb548	2021-06-12T00:00:00Z	60c485fdc31ac9011342ba31	5e6cf1ead6616700fb61b56c	6,45	0	5,9033625	1075	sb4
60c3e8d9ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-11T23:51:59.018Z	5f105b619e4d5400ed00b297	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c3e8d9ed8b3a011a4ad4fa	5f0f364bfef79802010fe5a4	7	0	7	1121	sb1
60c47c81ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-12T10:24:32.43Z	5e7b96711e73c901081df88c	605dd4588da91100ffdeba6f	2021-06-12T00:00:00Z	60c479bf8c18610125e5402b	60b9e29475908b012a15c663	7,8	0	7,8	2102	sb5
60c3ce3f8c18610125e	5d13407be54b0000cf	PLATE	2021-06-11T21:57:59.139Z	600df20d22d7d4011187d3dc	5f6e0bf8739d89011380b09	2021-06-12T00:00:00Z	60c3ce3f8c18610125e53ff5	600df20e22d7d4011187d3dd	7,9	0	7,9	3053	sb3
60c488a48c18610125	5d13407be54b0000cf	PLATE	2021-06-12T11:13:26.175Z	606b486d7bfd900127d1d4df	5f6e0bf8739d89011380b09	2021-06-12T00:00:00Z	60c488a48c18610125e54045	606b486e7bfd900127d1d4e0	7,9	0	7,9	3053	sb3
60c48e35c31ac90113	5d13407be54b0000cf	PLATE	2021-06-12T11:36:52.216Z	6065dd62b7fac1033e96f797	5f897da784cce70101161 6d2	2021-06-12T00:00:00Z	60c48e35c31ac9011342ba45	6065dd63b7fac1033e96f798	7	7	0	1121	sb1
60c47017c31ac90113	5d13407be54b0000cf	PLATE	2021-06-12T09:28:25.658Z	5eabdd5ea0618601075a1d11	5da5757dd1c60b00db2cb548	2021-06-12T00:00:00Z	60c47017c31ac9011342ba11	5eabdd5fa0618601075a1d12	6,45	0	6,45	1075	sb4
60c3e8d9ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-11T23:51:59.018Z	5f105b619e4d5400ed00b297	5eb9687e9ac4fc00fcfa7e6c	2021-06-12T00:00:00Z	60c3e8d9ed8b3a011a4ad4fa	5f0f364bfef79802010fe5a4	6,9	1,04	5,86	1043	sb2
60c3d2b38c18610125	5d13407be54b0000cf	PLATE	2021-06-11T22:17:01.268Z	5741f5e1d019e75593354452	5da5757dd1c60b00db2cb548	2021-06-12T00:00:00Z	60c3d2b38c18610125e53ffa	5f1be490b850c7a00fc81558f	12,9	0,9	11,433	1075	sb4
60c32497ed8b3a011a	5d13407be54b0000cf	PLATE	2021-06-11T09:54:23.868Z	604f3ae75f67551947d86b7d	5b9a951ee2b04707b57e096b	2021-06-11T00:00:00Z	60c32497ed8b3a011a4ad3c9	5ea6bb97d50a9500f6de9647	6,6	0	6,6	2029	sb6
60c3285fc31ac901134	5d13407be54b0000cf	PLATE	2021-06-11T10:10:32.108Z	6010aa373f4450122331bc1	5b9a951ee2b04707b57e096b	2021-06-11T00:00:00Z	60c3285fc31ac9011342b8cb	5f215f23da362101333ef31	6,6	0	6,6	2029	sb6
60c3270ac31ac90113	5d13407be54b0000cf	PLATE	2021-06-11T10:04:32.579Z	5e941b543ecd5b00fc82a82f	59f3004eacad7f856cbab2bb	2021-06-11T00:00:00Z	60c3270ac31ac9011342b8bb	5e941aed8cbe2e00f51fd4ef	6,7	0,67	5,5189575	3011	sb6
60c3269cdf69d5010cc	5d13407be54b0000cf	PLATE	2021-06-11T10:02:32.687Z	603f76c51d19a70944bbcad2	6040ef2b2b20d0010b179980	2021-06-11T00:00:00Z	60c3266bed8b3a011a4ad3d4	5e9ca91af7c3201077a4cd4	7,95	0,28	6,9962375	2090	sb7
60c323eded8b3a011a	5d13407be54b0000cf	PLATE	2021-06-11T09:51:20.038Z	5ce534fc08d49a00c17bb3eb	6040ef2b2b20d0010b179980	2021-06-11T00:00:00Z	60c323eded8b3a011a4ad3c0	5e6e472d8a2dd30104ed58f3	7,95	0	7,95	2090	sb7
60c32d8b8c18610125	5d13407be54b0000cf	PLATE	2021-06-11T10:33:12.145Z	608fb7ed9d5944011f327e9b	5f034c8000cfec00ee6c2fb0	2021-06-11T00:00:00Z	60c32d8b8c18610125e53ef4	608fb7ee9d5944011f327e9c	7,9	0	7,9	1104	sb8

Figura 1. Base de dados original com encomendas e vendas

mealId	category							
571c1aa224748e57c7000000	PLATE	I58	I36	I372	I247	I85	I89	I70
57210e0324748e7db2000000	PLATE	I45	I53	I256	I70	I383	I191	I227
572a556624748e6a94000000	PLATE	I45	I31	I70	I225	I227	I298	I313
572cd33d24748e2f35000000	PLATE	I31	I186	I70	I227	I306	I208	I42
574f5daa24748e7c33000000	PLATE	I191	I303	I263	I153	I95	I4	I97
579e1e8d24748e15ed000007	PLATE	I1	I70	I191	I215	I133	I329	I153
581e91c228c9082c04ff1008	PLATE	I31	I186	I231	I70	I191	I227	I36
5854504b28c9082c04ff16ca	PLATE	I186	I70	I225	I191	I36	I271	I215
585450a828c9082c04ff16cb	PLATE	I58	I182	I85	I70	I354	I186	I96
585e72f828c9082c04ff16e9	PLATE	I6	I29	I32	I58	I36	I89	I70
585e774a28c9082c04ff16eb	PLATE	I231	I70	I37	I191	I174	I51	I203
590b517f325a7145ba89228f	PLATE	I186	I70	I280	I225	I191	I215	I326
59997fa0acad7f856cb2f4bf	PLATE	I6	I58	I329	I374	I171	I70	I186
59ea12f7acad7f856c884dd2	PLATE	I186	I56	I191	I20	I214	I203	I162
59f3004eacad7f856cbab2bb	PLATE	I58	I107	I70	I186	I216	I356	I339
59fb601dacad7f856cec4a31	PLATE	I186	I128	I191	I338	I36	I14	I303
5a3bb225ad17d8a721b1f324	PLATE	I273	I106	I215	I275	I185	I186	I203

Figura 2. Base de dados com pratos e os códigos dos ingredientes

Na figura 1, a base de dados contém diversas variáveis únicas como código do cliente, código do prato, código da empresa, data do pedido, categoria, subcategoria, etc… Na figura 2, temos todos os pratos e seus respectivos ingredientes. Uma solução para o problema, seria utilizar um algoritmo de aprendizagem de máquina para realizar uma regressão múltipla com variáveis independentes relevantes, que pudessem explicar o comportamento do preço de cada prato. Analisando todas as variáveis disponíveis, algumas podem ser utilizadas como características, ou seja, atributos previsores do preço. Dentre elas estão:

- Quais ingredientes são usados em cada prato
- A subcategoria que o prato pertence
- O número de vendas que o prato teve durante o período disponível na base
- A quantidade total de ingredientes que o prato leva

Outras variáveis não possuem correlação e não determinam o preço de cada prato (ex:customer_ID, Order_ID, Category, Area_ID) e não serão utilizadas no modelo de regressão. Para ser possível realizar o treinamento do algoritmo, é preciso formatar os dados de forma que as variáveis independentes/características e a variável *target* fiquem dispostas no seguinte formato:

meal_ID	ref	ingrediente_1	ingrediente_2	ingrediente_3	ingrediente_4	...	ingrediente_134	subcategory	sales	total_ing	price
pjdd8a4tty2	1243	0	1	1	0	...	0	sb3	23	17	8,9
3hh4sd8k24	2410	0	0	1	1	...	0	sb1	41	12	7,5
...	...	...	...	...	...	...	...	...	...	...	...

Figura 3. Formatação ideal dos dados para o treino do algoritmo

Nesta base da figura 3 temos as *features* em azul e a variável *target* em verde. Cada linha é um prato único com seu respectivo código(ref e Meal_ID). Esta base irá possuir todos os pratos com exceção do prato **1028** que será salvo separadamente e inserido no algoritmo na etapa final. O prato **1028** deverá estar formatado desta forma conforme a imagem a seguir:

meal_ID	ref	ingrediente_1	ingrediente_2	ingrediente_3	ingrediente_4	...	ingrediente_134	subcategory	sales	total_ing	price
483ha484jhj23	1028	1	0	1	0	...	0	sb2	35	16	?

Figura 4. Formatação ideal do prato 1028

2. Preparação da base com os ingredientes

Foi utilizado um script em Python para ajustar os dados conforme o planejado. O primeiro passo foi importar a base e remover a coluna “category”, pois era comum a todos os registros. Em seguida foi verificado a quantidade de pratos existentes (134) e a quantidade de ingredientes únicos(334):

```
meal = pd.read_csv('C:/Users/Andre/Desktop/JDS/case eatasty/Meal.csv')

#removendo a coluna "category"
meal.drop(columns=['category'], inplace=True)

#número de pratos únicos
len(meal['mealId'].unique())
134
```

Dentre todos os ingredientes, existiam valores incorretos e nulos que foram removidos:

```
 'I534', 'I207', 'I286', 'I537',
'I541', 'I462', 'I545', 'I544',
 'I16', 'vinho-do-porto', ']',
 'I130', 'I581', 'I551', 'I577',
 'I125', 'I564', 'I590', 'I205',
 'I474', 'I421', 'I44', 'I607',
```

Figura 5. Valores incorretos nos códigos dos ingredientes

```
Ingredientes.dropna(inplace=True)          #removendo valores nulos (nan)
Ingredientes.drop_duplicates(inplace=True)   #removendo valores duplicados

#removendo valores "errados"
Ingredientes.drop(Ingredientes.index[Ingredientes == 'vinho-do-porto'], inplace = True)
Ingredientes.drop(Ingredientes.index[Ingredientes == ']'], inplace = True)

#reiniciando index
Ingredientes = Ingredientes.reset_index()
del Ingredientes['index']
```

Em seguida foram criadas várias colunas, uma para cada ingrediente. Foi atribuído o valor “1” em casos que o prato contém aquele ingrediente, ou “0” se o prato não contém o ingrediente. Em seguida foram removidas as colunas antigas com os códigos por linha. O código a seguir mostra como foi feito este processo:

```
#criando colunas com os 334 ingredientes
for a in range(0,334):
    meal[Ingredientes.iloc[a,]] = 0

#preenchendo colunas dos ingredientes para cada prato
for h in range(0,133):
    for g in range(25,359):
        for f in range(1,24):
            y = meal.iloc[h,f]
            if y == meal.columns[g]:
                meal.iloc[h,g]=1

#apagando colunas com códigos dos ingredientes
meal.drop(columns = meal.columns[1:25] , inplace=True)

#salvando dataframe
meal.to_excel('meal_encoded.xlsx')
meal.to_csv('meal_encoded2.csv')
```

Ao final, os *dataframes* resultantes foram salvos para a etapa posterior. As figuras 6 e 7 mostram de forma visual como ficou a formatação dos dados após o processamento:

	mealId	category	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 16	Unnamed: 17
0	571c1aa224748e57c7000000	PLATE	I58	I36	I372	I247	I85	I89	I70	I186	...	I196	NaN
1	57210e0324748e7db2000000	PLATE	I45	I53	I256	I70	I383	I191	I227	I199	...	I364	I194
2	572a556624748e6a94000000	PLATE	I45	I31	I70	I225	I227	I298	I313	I315	...	I249	I330
3	572cd33d24748e2f35000000	PLATE	I31	I186	I70	I227	I306	I208	I42	I97	...	I83	I2

Figura 6. Base de dados original

	mealId	I58	I36	I372	I247	I85	I89	I70	I186	I203	...	I193	I668	I467	I692	I535	I695	I696	I73	I670	I700
0	571c1aa224748e57c7000000	1	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
1	57210e0324748e7db2000000	1	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
2	572a556624748e6a94000000	1	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
3	572cd33d24748e2f35000000	1	0	0	0	0	0	1	1	0	...	0	0	0	0	0	0	0	0	0	0

Figura 7. Base de dados processada

3. Junção das bases e Data Cleaning

Em seguida, foi feita a junção da base da figura 1 com a base processada utilizando a variável comum entre as duas bases, ou seja, o número Id de cada prato (mealId). Após a junção, foi feita uma breve análise exploratória dos dados utilizando os *softwares* Orange e Power BI.

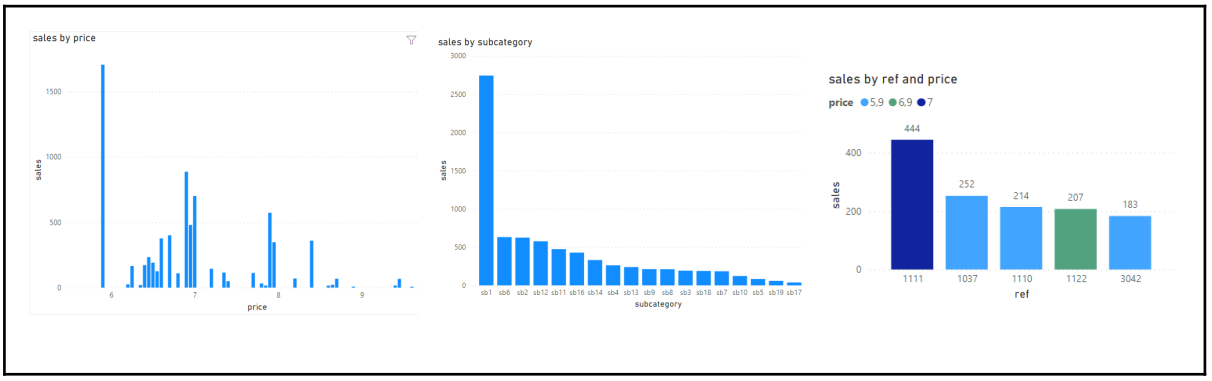

Figura 8. Número de vendas em função do preço / Número de vendas em função da subcategoria / Top 5 pratos em número total de vendas

Em um segundo script em Python, foi feita toda a etapa de pré-processamento dos dados. Verificou-se que havia um registro com ‘price’ igual a zero. Este mesmo registro encontrado, também não possuía nenhum ingrediente na planilha de dados inicial e foi removido da base.

```
#encontrando o registro com preço = 0

full[full['price']==0]
```

	price	discountTotal	totalValue	ref	subcategory	mealId	I58	I36	I372	I247	...	I193	I668	I467	I692	I535	I695	I696	I73	I670
3469	0.0	0.0	0.0	40	sb6	5fae8ba63f546400fbff35e5	0	0	0	0	...	0	0	0	0	0	0	0	0	0

1 rows × 340 columns

Figura 9. Registro com preço e ingredientes nulos

Após este registro removido, a base total ficou com **7610** registros e **340** colunas:

	price	discountTotal	totalValue	ref	subcategory	mealId	I58	I36	I372	I247	...	I193	I668	I467	I692	I535	I695	I696	I73	I670
0	7.0	1.05	5.950000	1121	sb1	5f897da784cce701011616d2	1	1	1	0	...	0	0	0	0	0	0	0	0	0
1	6.9	4.00	2.900000	1043	sb2	5eb9687e9ac4fc00fcfa7e6c	1	0	0	0	...	0	0	0	0	0	0	0	0	0
2	7.0	7.00	0.000000	1121	sb1	5f897da784cce701011616d2	1	1	1	0	...	0	0	0	0	0	0	0	0	0
3	6.9	0.00	6.900000	1043	sb2	5eb9687e9ac4fc00fcfa7e6c	1	0	0	0	...	0	0	0	0	0	0	0	0	0
4	7.0	0.00	7.000000	1121	sb1	5f897da784cce701011616d2	1	1	1	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7606	6.6	0.55	5.490650	1051	sb6	5c62a620dd8a61abd1f77713	1	0	0	0	...	0	0	0	0	0	0	0	0	0
7607	5.9	0.55	4.849975	1037	sb1	5a3bb225ad17d8a721b1f324	0	0	0	0	...	0	0	0	0	0	0	0	0	0
7608	7.9	0.00	7.900000	3065	sb12	5fdb6e050d93dd00f9fc1a66	0	1	0	0	...	0	0	0	0	0	0	0	0	0
7609	8.4	1.68	6.720000	2072	sb11	5f89870726361f00fa97a62b	1	0	0	0	...	0	0	0	0	0	0	0	0	0
7610	5.9	0.00	5.900000	1037	sb1	5a3bb225ad17d8a721b1f324	0	0	0	0	...	0	0	0	0	0	0	0	0	0

7610 rows × 340 columns

Figura 10.Base completa a ser processada

Foi então criada uma lista com todos os 133 pratos únicos. Note que antes eram 134 pratos únicos, mas o prato 40 foi removido e possuía apenas uma ocorrência em todos os registros. Analisando a base completa, foi observado que a coluna ‘price’ possuía valores duplicados ou triplicados em vários casos. Dado este fenômeno, considerei que estes valores duplicados/triplicados foram erros que ocorreram por algum motivo desconhecido. As imagens a abaixo ilustram este problema:

```
full.loc[full['ref']==pratos[3]]['price']
```

```
7       6.45
9       6.45
15      6.45
17     12.90
4115    6.45
4136    6.45
4137    6.45
4157    6.45
6082    6.45
6084    6.45
6099    6.45
6115    6.45
6117    6.45
6119    6.45
6121    6.45
6127    6.45
6130    6.45
6133   12.90
6135    6.45
6137    6.45
6139    6.45
6145    6.45
6149    6.45
6154    6.45
6159    6.45
6160    6.45
6161    6.45
6168    6.45
6188    6.45
6196    6.45
6199    6.45
6202    6.45
6203   12.90
6204    6.45
6210    6.45
```

Figura 11.Valores inconsistentes do prato 1075

```
full.loc[full['ref']==pratos[7]]['price']
```

```
21      7.95
22      7.95
24      7.95
26      7.95
27      7.95
30      7.95
32      7.95
36      7.95
39      7.95
40      7.95
41      7.95
42      7.95
45      7.95
62      7.95
65      7.95
69      7.95
76      7.95
82      7.95
6791   15.90
6793    7.95
6796    7.95
6797    7.95
6802    7.95
6816    7.95
6820   15.90
6822    7.95
6823    7.95
6829    7.95
6830    7.95
6846    7.95
6864   15.90
6865    7.95
6884    7.95
6888    7.95
6890    7.95
6897    7.95
6909    7.95
6916   15.90
6917    7.95
```

Figura 12.Valores inconsistentes do prato 2090

A hipótese da venda desses pratos terem sido registrados de forma duplicada também foi descartada pois cada amostra possui um ‘ID’, ‘order_ID’ e ‘Meal_ID’ associados. Além disso, a base de dados não possui nenhuma variável informando a quantidade de itens vendidos, ou seja, para cada amostra a quantidade de venda é única.

_id	createddate	customerId	mealId	orderId	price	scountTot	otalValue	ref
6076b7bd5e4394010ef2a52a	2021-04-14T10:37:51.127Z	5e4a647204504e010233b608	5c05345e1462e56e4eb3b5cb	6076b7bd5e4394010ef2a529	6.5	0	6.5	1053
6076b7cf5d5c80011c3e5fa5	2021-04-14T10:37:23.957Z	5afc4eea1ac63500f42bb753	5ac3d6bb1d983805a37d73c7	6076b7cf5d5c80011c3e5fa4	5.9	0	5.39998	2027
6076b7e45e4394010ef2a530	2021-04-14T10:38:14.639Z	5e61025db00c0e00fb41b29d	5c05345e1462e56e4eb3b5cb	6076b7e45e4394010ef2a52f	6.5	0	6.5	1053
6076b7f25e4394010ef2a532	2021-04-14T10:38:38.473Z	6064ad7df4aa42011447026c	59ea12f7acad7f856c884dd2	6076b7f25e4394010ef2a531	6.95	0	6.95	1028
6076b80a3613e30115feeab0	2021-04-14T10:41:53.34Z	5f16c2797845810102bcf713	5c05345e1462e56e4eb3b5cb	6076b80a3613e30115feeaaf	6.5	0	6.5	1053
6076b8945d5c80011c3e5fb4	2021-04-14T10:40:49.597Z	5c8f7c172cd14f00c3fdfcc6	59ea12f7acad7f856c884dd2	6076b8945d5c80011c3e5fb3	6.95	0	6.36099	1028
6076b8945d5c80011c3e5fb5	2021-04-14T10:40:49.597Z	5c8f7c172cd14f00c3fdfcc6	5ac3d6bb1d983805a37d73c7	6076b8945d5c80011c3e5fb3	5.9	0	5.39998	2027
6076b8d45e4394010ef2a540	2021-04-14T10:42:37.566Z	5e4d1c0446096c00fb72e8d0	59ea12f7acad7f856c884dd2	6076b8d45e4394010ef2a53f	6.95	0	6.95	1028
6076b8d84055030127fb4465	2021-04-14T10:43:23.87Z	5dc2ace0ab99ae00f42a3c7d	5fc0f53dbbc54a01007b18e2	6076b8d84055030127fb4464	7.2	1.44	5.76	3064
6076c2b53613e30115feeb8d	2021-04-14T11:24:07.614Z	606332d4c1eef7011bb07f60	59ea12f7acad7f856c884dd2	6076b8dd4055030127fb4466	6.95	0.7	6.25	1028
6076c2a64055030127fb4534	2021-04-14T11:24:07.614Z	606332d4c1eef7011bb07f60	59ea12f7acad7f856c884dd2	6076b8dd4055030127fb4466	6.95	0	6.95	1028
6076b9305d5c80011c3e5fbb	2021-04-14T10:43:37.908Z	5f44412527149c010c160812	59ea12f7acad7f856c884dd2	6076b9305d5c80011c3e5fba	6.95	0	6.95	1028
6076b9374055030127fb4471	2021-04-14T10:45:12.738Z	5e6ffd342d776f00eff966c3	5fc0f53dbbc54a01007b18e2	6076b9374055030127fb4470	7.2	0	7.2	3064
6076b9a25e4394010ef2a54e	2021-04-14T10:47:48.842Z	6036b245bf65ba011064d14e	59ea12f7acad7f856c884dd2	6076b9a25e4394010ef2a54d	13.9	0	13.9	1028
6076b9a25e4394010ef2a54f	2021-04-14T10:47:48.842Z	6036b245bf65ba011064d14e	5de80409b727e500e899df40	6076b9a25e4394010ef2a54d	6.4	0	6.4	1082
6076ba484055030127fb4480	2021-04-14T10:48:29.695Z	600ea3c0316cb60524cbde89	5de80409b727e500e899df40	6076ba484055030127fb447f	6.4	0	6.4	1082
6076baf55e4394010ef2a56f	2021-04-14T10:51:16.25Z	6061a7bd8c62d501267a8e2e	5c05345e1462e56e4eb3b5cb	6076baf55e4394010ef2a56e	6.5	0	6.5	1053

Figura 13. Valor duplicado do prato 1028 encontrado na base de dados original

Para corrigir este problema, para todas as amostras de determinado prato, considerei que o preço correto é o preço mínimo encontrado neste conjunto de valores. Além do preço mínimo, a moda ou mediana também poderiam ter sido usadas para realizar este filtro.

Após definida esta estratégia de adoção de um preço “base” de cada prato, foi criado um novo *data frame* com os 133 pratos existentes com uma coluna adicional chamada ‘sales’. Esta coluna foi utilizada para representar o total de vendas realizadas por prato. Todo esse processo foi feito através do código abaixo:

```
#criando coluna para armazenar o total de vendas por prato
df['sales']= 0

#adicionando o total de vendas por prato
k=0
for k in range(0,133):
    df['sales'][k]  = len(full.loc[full['ref']== pratos[k]])

#adicionando preço base por prato (filtrando valores incorretos)
k=0
for k in range(0,133):
    df['price'][k]  = full.loc[full['ref']==pratos[k]]['price'].min()
```

É importante compreender que no código acima o *data frame* ‘df’ está buscando os valores no *data frame* ‘full’. Este *data frame* ‘full’ (representado na figura 10), não será mais utilizado daqui em diante e os dados a serem preparados para o algoritmo de regressão estarão todos no *data frame* ‘df’.

Em seguida foi criada uma última coluna chamada ‘total_ing’ com o total de ingredientes por prato. O último passo antes de salvar o *data frame* processado, foi fazer a conversão da variável ‘subcategory’ de *string* para *int* utilizando o LabelEncoder() e normalização das variáveis numéricas utilizando o MinMaxScaler(). A normalização dos valores é necessária, caso o algoritmo de regressão trabalhe com cálculos de distâncias. Como não havia definido previamente qual algoritmo seria utilizado, optei por já realizar a normalização.

```
#convertendo o tipo da variável "subcategory" de string->numérica utilizando o LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['subcategory'] = le.fit_transform(df['subcategory'])

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['subcategory'] = scaler.fit_transform(df[['subcategory']])

#Realizando a normalização da coluna "total_ing"
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['total_ing'] = scaler.fit_transform(df[['total_ing']])

#Realizando a normalização da coluna "sales"
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['sales'] = scaler.fit_transform(df[['sales']])
```

Finalmente a base de dados está pronta para ser utilizada na etapa final. Antes de salvar os arquivos, é necessário separar e apagar todos os registros do prato 1028, pois ele será usado somente no final após o treinamento e avaliação do regressor. Foram salvos no formato .csv, um arquivo contendo apenas o registro do prato 1028 chamado 'b_1028.csv' e outro contendo todos os 132 pratos restantes chamado 'b_treino.csv'.

4. Etapa final

No último script, após os arquivos serem carregados, os *data frames* finais obtidos foram estes:

	price	subcategory	I58	I36	I372	I247	I85	I89	I70	I186	...	I467	I692	I535	I695	I696	I73	I670	I700	sales	total_ing
0	7.00	0.000000	1	1	1	0	0	0	1	1	...	0	0	0	0	0	0	0	0	0.031603	0.727273
1	6.90	0.578947	1	0	0	0	0	0	1	1	...	0	0	0	0	0	0	0	0	0.067720	0.545455
2	7.90	0.684211	1	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0.090293	0.590909
3	6.45	0.736842	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0.103837	0.590909
4	7.80	0.789474	1	1	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0.040632	0.590909
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
127	6.40	0.000000	1	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0.090293	0.727273
128	7.90	0.157895	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0.033860	0.454545
129	6.60	0.473684	1	1	0	0	0	1	1	1	...	0	0	0	0	0	0	0	0	0.063205	0.636364
130	8.40	0.631579	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0.000000	0.636364
131	6.90	0.000000	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0.065463	0.772727

132 rows × 338 columns

Figura 14. DataFrame final usado na regressão

	price	subcategory	I58	I36	I372	I247	I85	I89	I70	I186	...	I467	I692	I535	I695	I696	I73	I670	I700	sales	total_ing
0	6.95	0.0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0.1693	0.636364

1 rows × 338 columns

Figura 15. DataFrame com o registro do prato 1028

Realizei vários testes usando diferentes tipos de algoritmos (SRV, Decision Trees, etc..) e os que obtiveram os melhores resultados foram o **Random Forest** e o **XGBoost Regressor**. Os dados foram separados entre atributos previsores e o atributo target('price'). O registro único do prato 1028 foi formatado da mesma forma.

```
#atribuindo as colunas "subcategory", 'sales', 'total_ing' e as colunas dos ingredientes como features
X = base_r.iloc[:, 1:].values

#atribuindo o preço como variável dependente da regressão
y = base_r.iloc[:, 0].values

#amostra do prato 1028 é formatada como as amostras de treino (sem a coluna price)
Z = Q.iloc[0, 1:].values
```

A proporção entre dados de treinamento e dados de teste foi a mesma nos dois algoritmos, 3 amostras de teste e 129 amostras de treinamento. Cada algoritmo foi ajustado de forma empírica variando cada um de seus parâmetros, com o objetivo de se obter:

- o melhor score de regressão (R^2),
- o menor MAE (Mean Absolute Error)
- o menor RMSE (Root Mean Square Error)

Os melhores parâmetros obtidos para cada algoritmo foram os seguintes:

```
xgboost.XGBRegressor(n_estimators=60, max_depth=5, learning_rate=0.09)

RandomForestRegressor(n_estimators = 220, max_depth=17,  random_state = 0)
```

Os resultados da regressão de cada algoritmo estão representados a seguir:

#RandomForest	#XGBoost
MAE: 0.49730 RMSE: 0.55644 score_tr: 0.89322 score_te: 0.53555 **rf_price: 6.70344**	MAE: 0.37739 RMSE: 0.38418 score_tr: 0.91415 score_te: 0.77859 **xgb_price: 6.65804**

Dentre os dois algoritmos, o XGBoost teve a melhor performance obtendo um R^2 de 0.91, ou seja, indicando que 91% da variância do preço pode ser explicada pelas variáveis independentes. Foi realizado um teste iterativo variando o parâmetro 'random_state' e o parâmetro 'n_estimators' de ambos os algoritmos, de modo a observar o comportamento em função do número de estimadores e variação das amostras utilizadas no treinamento.

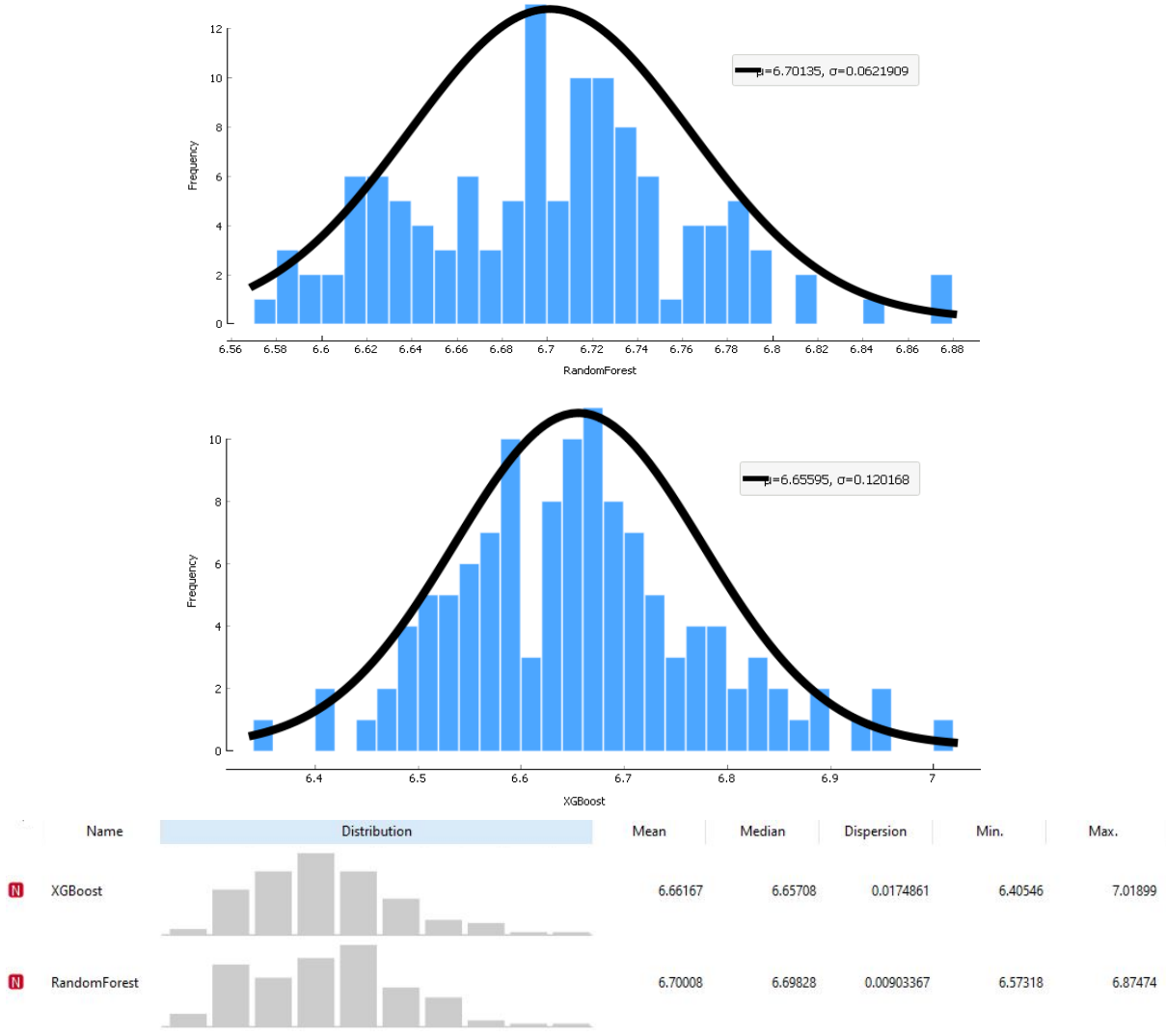

Figura 16. Distribuições das predições para cada algoritmo

Analisando a distribuição das previsões do algoritmo **RandomForest**, o desvio padrão foi menor, porém o erro nas amostras de testes foi maior. Por este motivo, sua predição é relativamente menos confiável em relação ao XGBoost. A partir dos resultados dos experimentos deste estudo, é possível então concluir que de acordo com as variáveis independentes utilizadas neste modelo, o preço ideal para o prato com o código **1028** está compreendido em torno de **6,65 (€)** de acordo com o algoritmo **XGBoost**. Outras variáveis independentes (como o custo) poderiam também ser adicionadas para eventualmente melhorar a performance do modelo.