

Instituto Politécnico de Setúbal

Escola Superior de Tecnologia do Barreiro

## **Projeto de “Big Data”**

Licenciatura em Bioinformática

# **Relatório do Projeto de Big Data**

Janeiro de 2023

Realizado por:

Pedro Augusto (202000169)

Pedro Brito (202000062)

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Desenvolvimento e Resultados</b>	<b>2</b>
2.1	K Means . . . . .	7
2.2	Regressão Logística . . . . .	10
<b>3</b>	<b>Conclusão</b>	<b>16</b>

# 1 Introdução

Este projeto foi realizado no âmbito da UC de Big Data e tem como referência uma base de dados que está relacionada com anúncios para compra, arrendamento e férias de propriedades situadas em Portugal, tendo em conta 8 variáveis.

Este projeto tem como finalidade o uso de modelos de classificação de Machine Learning para formar diferentes clusters e prever os dados. Neste projeto recorreremos à utilização do software Jupyter Notebook para a realização dos modelos de Machine Learning utilizando pyspark e com a ajuda do Latex Online desenvolvemos toda a nossa estrutura do relatório.

## 2 Desenvolvimento e Resultados

- Como primeiro passo, antes de começarmos a realizar qualquer estudo sobre o dataset, efetuamos um tratamento de erros onde procedemos à eliminação de todas as rows que possuíam nulls.

```
In [5]: #Tratamento de erros, eliminamos todas as rows com nulls
df = df.dropna()
df.printSchema()
```

Figura 1: Tratamento de erros: eliminação de todas as rows com nulls

- De seguida, fomos saber quantas linhas o dataset tinha e quantas localizações diferentes havia. Com a realização do código apresentado abaixo percebemos que existem 93806 linhas no dataset e 2431 localizações diferentes.

```
In [28]: #Contar o número de casas por Location e apresentamos de forma decendente
df.groupBy('Location').agg(count('Location').alias('House_count')).orderBy(desc('House_count')).show()
```

Location	House_count
Cedofeita, Santo...	1875
Matosinhos e Leç...	1635
Paranhos, Porto	1577
Montijo e Afonso...	1396
Albufeira e Olho...	1319
Canidelo, Vila N...	1305
Santa Marinha e ...	1274
Cascais e Estori...	1196
Glória e Vera Cr...	1108
Quarteira, Loulé...	1079
Mafamude e Vilar...	983
Ramalde, Porto	959
Portimão, Faro	896
Castelo (Sesimbr...	755
São Gonçalo de L...	736
Póvoa de Varzim,...	725

Figura 2: Contar o nº de casas por localização de forma decrescente

- Decidimos também fazer a contagem do número de casas por localização e apresentamos de forma decrescente para perceber quais as localizações com mais anúncios de casas. Compreendemos então que a localização com mais anúncios é Cedofeita, Santo Ildefonso, Sé, Miragaia, São Nicolau e Vitória, também conhecida como União de Freguesias do Centro Histórico do Porto com 1875 anúncios.

```
In [8]: #Contamos o número de linhas do dataset
df.count()

Out[8]: 93806

In [9]: #Contamos o número de Localizações diferentes no dataset
df.select('Location').distinct().count()

Out[9]: 2431
```

Figura 3: Contagem do nº de linhas e de localidades diferentes

- Neste passo fizemos a separação da coluna da localização em duas colunas sendo essas as colunas da cidade ("Parish") e freguesia ("County").

```
In [29]: #Split da coluna Location
split_cols = pyspark.sql.functions.split(df['Location'], ',')

#Adiciona as colunas Parish e County, eliminamos a coluna Location
df1 = df.withColumn('Parish', split_cols.getItem(0)) \
        .withColumn('County', split_cols.getItem(1)).drop('Location')

In [30]: #Mostrar o dataset com as alterações
df1.show()
```

Rooms	Price	Area	Bathrooms	Condition	AdsType	ProprietyType	Parish	County
3	1000	125	3	Used	Rent	Apartament	Espinho	Aveiro
3	750	138	2	New	Rent	Apartament	Válega	Ovar
1	450	32	2	New	Rent	Apartament	Gafanha da Nazaré	Ílhavo
3	1300	129	3	Used	Rent	Apartament	Esmoriz	Ovar
3	1300	153	3	Renovated	Rent	Apartament	Esmoriz	Ovar
2	850	88	2	Renovated	Rent	Apartament	Esgueira	Aveiro
2	625	90	2	Used	Rent	Apartament	Santa Maria da F...	Travanca
2	825	77	2	New	Rent	Apartament	Espinho	Aveiro
2	950	79	2	New	Rent	Apartament	Espinho	Aveiro

Figura 4: Split da coluna da localização em duas, cidade e freguesia

- Uma vez que temos duas novas colunas, visamos contar quantas freguesias diferentes existem no dataset. Com a realização do código abaixo entendemos que há 459 freguesias diferentes no nosso dataset.

```
In [13]: #Contamos quantos County existem
df1.select('County').distinct().count()

Out[13]: 459
```

Figura 5: Contagem do nº de freguesias diferentes

- De seguida, organizamos o dataset por freguesia em relação ao preço.

```
In [14]: #Organizar o dataset por county em relação ao preço
df_price = df1.select('County', 'Price').orderBy(desc('Price')).show()
```

County	Price
Vila Nova de Fam...	285000000
Lisboa	190658610
Lisboa	190658610
Lisboa	168689950
Lisboa	164689950
Lisboa	140645234
Lisboa	138435353
Lisboa	137896713
Bragança	123556750

Figura 6: County em relação ao preço

- Decidimos adicionar também a área.

```
In [15]: #Colocamos a área também
df_price = df1.select('County','Price','Area').orderBy(desc('Price')).show()
```

County	Price	Area
Vila Nova de Fam...	285000000	441
Lisboa	190658610	162
Lisboa	190658610	162
Lisboa	168689950	185
Lisboa	164689950	185
Lisboa	140645234	126
Lisboa	138435353	123
Lisboa	137896713	120
Bragança	123556750	100
Lisboa	122716281	104

Figura 7: County em relação ao preço e à área

- Seguidamente criamos uma coluna designada de "Rooms Totais", que é a soma dos quartos mais as casas de banho e, de seguida, fazemos a média do preço da divisão por freguesia.

```
In [16]: #Criamos uma coluna de Rooms Totais que é a soma dos Rooms mais as Bathrooms e fazemos a média o preço por divisão por Freguesia
df1 = df1.withColumn('TotalRooms', df1.Rooms + df1.Bathrooms)

room_price = df1.withColumn("RoomPrice", df1.Price / df1.TotalRooms)
room_price = room_price.groupBy("County").mean("RoomPrice")

room_price.show()

df2 = df1.join(room_price,on='County',how='inner')
df2.printSchema()
```

County	avg(RoomPrice)
Matosinhos	101673.71423869426
Aguiar da Beira	21070.194003527336
Oeiras	104075.51592945983
Montijo	72795.03349067195
Santa Cruz e São...	53061.33219954648
Vila Velha de Rodão	12694.583333333332
Estarreja	43751.388578869046
Batalha	33195.72072072072
Lourinhã	69144.14912038315
Câmara de Lobos	60568.824404761916
Mealhada	33261.28271128271
Ameal e Arzila	27774.50980392157
Tondela	21977.31922398589

Figura 8: Preço da divisão por freguesia

- Fizemos um cálculo para perceber a média do preço por m2 por freguesia.

```
In [17]: #Preço por média da área por Freguesia
area_price = df2.withColumn('AreaPrice', df2.Price / df2.Area)
area_price = area_price.groupBy('County').mean('AreaPrice')

area_price.show()

df3 = df2.join(area_price,on='County',how='inner')
df3.printSchema()
```

County	avg(AreaPrice)
Matosinhos	3948.7979717840785
Aguiar da Beira	765.8850431199836
Oeiras	5840.673300492195
Montijo	2545.6494372053594
Santa Cruz e São...	2439.5103215384297
Vila Velha de Rodão	617.838030929693
Estarreja	1565.131212321675
Batalha	1072.0271660561484
Lourinhã	3336.1173661567022
Câmara de Lobos	1632.424658912842
Mealhada	907.7479711008058

Figura 9: Média do preço por m2 por freguesia



## 2.1 K Means

Vamos usar o K-Means que é um algoritmo de Machine Learning não-supervisionado para clustering. O objetivo do k-means é particionar um conjunto de dados em k clusters, onde cada observação pertence ao cluster com a média mais próxima.

- Primeiramente, vamos fazer um assemble onde vai resultar numa nova coluna designada de "features". E, de seguida, adaptámos o modelo à parte de treino do scaler e normalizámos os dados.

```
[28] ##### K MEANS #####
      #Treinamos a data com as nossas features
      from pyspark.ml.feature import VectorAssembler
      from pyspark.ml.feature import StandardScaler

      assemble=VectorAssembler(inputCols=[
          'avg(AreaPrice)', 'avg(RoomPrice)', 'avg(BathroomsValue)'
      ], outputCol='features')

      assembled_data=assemble.transform(df_final)

      scale=StandardScaler(inputCol='features',outputCol='standardized')
      data_scale=scale.fit(assembled_data)
      data_scale_output=data_scale.transform(assembled_data)

      data_scale_output.select('standardized').show(2,truncate=False)
```

Figura 10: Treino dos dados com as features

- Segundamente, usamos o algoritmo K-Means para fazer um cluster do dataframe criado anteriormente.

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import numpy as np

cost = np.zeros(10)

evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='standardized', metricName='silhouette', distanceMeasure='squaredEuclidean')

for i in range(2,10):
    KMeans_algo=KMeans(featuresCol='standardized', k=i)
    KMeans_fit=KMeans_algo.fit(data_scale_output)
    output=KMeans_fit.transform(data_scale_output)
    cost[i] = KMeans_fit.summary.trainingCost
```

Figura 11: Criar um cluster do dataframe

- Usamos a biblioteca Python Pandas (pd) e Matplotlib (pl) para criar e traçar uma "Elbow Curve", que utilizamos para encontrar o número ideal de clusters para o algoritmo k-means.

```
import pandas as pd
import pylab as pl
df_cost = pd.DataFrame(cost[2:])
df_cost.columns = ["cost"]
new_col = range(2,10)
df_cost.insert(0, 'cluster', new_col)
pl.plot(df_cost.cluster, df_cost.cost)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()
```

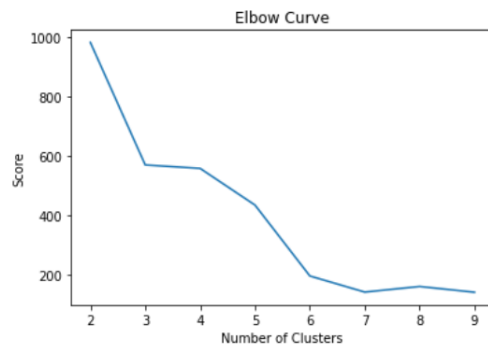


Figura 12: Criação de uma Elbow Curve

- Seguidamente, usamos a biblioteca do PySpark para executar o k-means clustering no dataframe.

```
[31] KMeans_algo=KMeans(featuresCol='standardized', k=4)
      KMeans_fit=KMeans_algo.fit(data_scale_output)
```

Figura 13: Executar o k-means clustering

- De seguida, executamos o clustering no dataframe usando o algoritmo k-means.

```
[32] preds=KMeans_fit.transform(data_scale_output)
      preds.show(5,0)
```

Figura 14: Executar o clustering no dataframe

- Finalmente, usamos a biblioteca do PySpark para executar a manipulação de dados e visualização de tarefas.

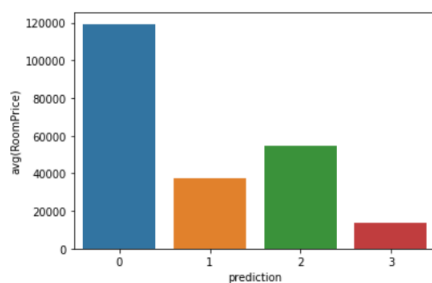


Figura 15: Predição do preço de cada divisão

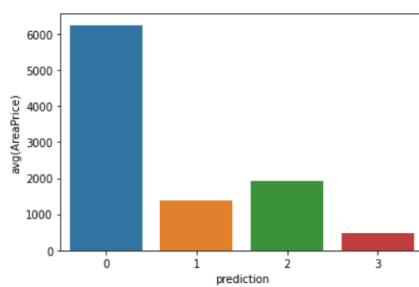


Figura 16: Predição do preço por área

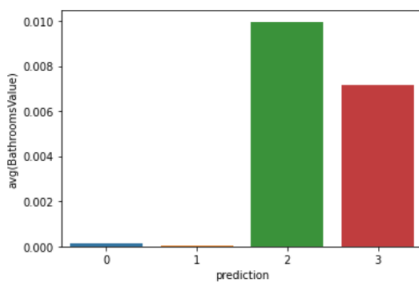


Figura 17: Predição do preço por casa de banho

## 2.2 Regressão Logística

A regressão logística é um método estatístico usado para modelar e resolver problemas de classificação. É um algoritmo de aprendizagem supervisionada usado para prever um resultado binário dado um conjunto de variáveis independentes.

- Primeiro, criamos uma nova sessão Spark.

```
[35] spark = SparkSession.builder.appName("Price").getOrCreate()
```

Figura 18: Criação de sessão Spark

- De seguida, preparámos o dataset para a regressão logística.

```
[ ] #preparamos o dataset para a regressão logística
df5 = df1
df5 = df5.drop('Perish', 'TotalRooms')
df5.show(5,0)
df5 = df5.dropna()
df5.printSchema()
```

Figura 19: Preparação do dataset

- Usamos o stringIndexer para alterar os valores de string para int uma vez que para usarmos a regressão logística todos os valores têm que ser do tipo int.

```
#Usamos o stringIndexer para alterar os valores string para int, para os podermos utilizar para a regressão logística
condition_indexer = StringIndexer(inputCol="Condition", outputCol="conditionIndex")
df5 = condition_indexer.fit(df5).transform(df5)
df5 = df5.drop('Condition')

adsType_indexer = StringIndexer(inputCol="AdsType", outputCol="adsTypeIndex")
df5 = adsType_indexer.fit(df5).transform(df5)
df5 = df5.drop('AdsType')

proprietyType_indexer = StringIndexer(inputCol="ProprietyType", outputCol="proprietyTypeIndex")
df5 = proprietyType_indexer.fit(df5).transform(df5)
df5 = df5.drop('ProprietyType')

county_indexer = StringIndexer(inputCol="County", outputCol="countyIndex")
df5 = county_indexer.fit(df5).transform(df5)
df5 = df5.drop('County')
df5.show(5,0)
```

Figura 20: Utilização do stringIndexer

- Seguidamente, dividimos em clusters o preço para a regressão logística poder prevê-lo.

```
#dividimos em clusters o preço para a regressão logística poder prever o preço
df6 = df5
df6 = df6.withColumn("Price", when(col("Price")<4999, 0).when(col("Price")<99999, 1).when(col("Price")<499999, 2).otherwise(3))
df6.show(5,0)
```

Rooms	Price	Area	Bathrooms	conditionIndex	adsTypeIndex	proprietyTypeIndex	countyIndex
3	0	125	3	0.0	1.0	0.0	4.0
3	0	138	2	1.0	1.0	0.0	104.0
1	0	32	2	1.0	1.0	0.0	25.0
3	0	129	3	0.0	1.0	0.0	104.0
3	0	153	3	3.0	1.0	0.0	104.0

only showing top 5 rows

Figura 21: Divisão dos clusters

- Posteriormente, criámos um vetor para a regressão logística.

```
#fazemos o vetor para a regressão logística
assembler = VectorAssembler(inputCols = ['Rooms', 'Area', 'Bathrooms', 'conditionIndex', 'adsTypeIndex', 'proprietyTypeIndex', 'countyIndex'], outputCol='features')
output = assembler.transform(df6)
output.show(5,0)
```

Rooms	Price	Area	Bathrooms	conditionIndex	adsTypeIndex	proprietyTypeIndex	countyIndex	features
3	0	125	3	0.0	1.0	0.0	4.0	[3.0,125.0,3.0,0.0,1.0,0.0,4.0]
3	0	138	2	1.0	1.0	0.0	104.0	[3.0,138.0,2.0,1.0,1.0,0.0,104.0]
1	0	32	2	1.0	1.0	0.0	25.0	[1.0,32.0,2.0,1.0,1.0,0.0,25.0]
3	0	129	3	0.0	1.0	0.0	104.0	[3.0,129.0,3.0,0.0,1.0,0.0,104.0]
3	0	153	3	3.0	1.0	0.0	104.0	[3.0,153.0,3.0,3.0,1.0,0.0,104.0]

only showing top 5 rows

Figura 22: Criação de um vetor

- Neste passo, realizámos a standarização dos dados.

```
[ ] #standarizamos
scale=StandardScaler(inputCol='features',outputCol='standardized')
scale=scale.fit(output)
scale_output= scale.transform(output)

scale_output.select('standardized').show(2,truncate=False)
```

Figura 23: Standarização dos dados

- Depois, fomos obter o dataframe final.

```
#dataframe final
finalised_data = scale_output.select("features", "Price")
finalised_data.show(5,0)
```

features	Price
[3.0,125.0,3.0,0.0,1.0,0.0,4.0]	0
[3.0,138.0,2.0,1.0,1.0,0.0,104.0]	0
[1.0,32.0,2.0,1.0,1.0,0.0,25.0]	0
[3.0,129.0,3.0,0.0,1.0,0.0,104.0]	0
[3.0,153.0,3.0,3.0,1.0,0.0,104.0]	0

only showing top 5 rows

Figura 24: Obtenção do dataframe final

- De seguida, treinámos os dados em 80% e os dados de teste em 20%.

```
#treinamos a data em 80% e a data de teste é 20%
train, test = finalised_data.randomSplit([0.8, 0.2])
```

Figura 25: Treino de dados

- Seguidamente, aplicámos o modelo de regressão logística.

```
#aplicamos o modelo
lr = LogisticRegression(featuresCol='features',labelCol='Price')
lrn = lr.fit(train)
predictions=lrn.transform(test)
```

Figura 26: Aplicação do modelo

- Posteriormente, mostrámos os resultados.

```
#mostramos os resultados
lrn_summary = lrn.summary
lrn_summary.predictions.show()
```

features	Price	rawPrediction	probability	prediction
(7,[0,1,2],[1.0,2...]	2.0	[-2.5503370024186...	[0.00356567904022...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5501976911649...	[0.00356573621341...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5501976911649...	[0.00356573621341...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5501976911649...	[0.00356573621341...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5501976911649...	[0.00356573621341...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5500583799111...	[0.00356579335935...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5496404461497...	[0.00356596463388...	2.0
(7,[0,1,2],[1.0,2...]	2.0	[-2.5496404461497...	[0.00356596463388...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5495011348959...	[0.00356602167100...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5495011348959...	[0.00356602167100...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5495011348959...	[0.00356602167100...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5495011348959...	[0.00356602167100...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5493618236422...	[0.00356607868095...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5492225123884...	[0.00356613566374...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5492225123884...	[0.00356613566374...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5492225123884...	[0.00356613566374...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5492225123884...	[0.00356613566374...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5490832011346...	[0.00356619261939...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5488045786270...	[0.00356630644929...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5488045786270...	[0.00356630644929...	2.0
(7,[0,1,2],[1.0,3...]	2.0	[-2.5488045786270...	[0.00356630644929...	2.0

only showing top 20 rows

Figura 27: Resultados

- Depois, decidimos apresentar um sumário dos dados.

```
lrn_summary.predictions.describe().show(5)
```

summary	Price	prediction
count	148816	148816
mean	2.1236358993656594	2.0480996667025053
stddev	0.5924420839237861	0.48055074985701934
min	0.0	0.0
max	3.0	3.0

Figura 28: Sumário de dados

- De seguida, fomos tentar obter a taxa sucesso de previsão do modelo onde obtemos um valor de 0.97.

```
#taxa de previsão do modelo
eval = BinaryClassificationEvaluator(rawPredictionCol = "prediction", labelCol = "Price")
auc = eval.evaluate(predictions)
print(auc)
```

0.972612659246318

Figura 29: Taxa de sucesso de previsão



- E, por fim, realizámos a matrix de confusão.

```
#fazemos a matrix confusão
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

y_test = predictions.select("Price")
y_test = y_test.toPandas()

y_pred = predictions.select("prediction")
y_pred = y_pred.toPandas()

cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.set_xlabel('Predicted outputs', color='black')
ax.set_ylabel('Actual outputs', color='black')
ax.xaxis.set(ticks=range(4))
ax.yaxis.set(ticks=range(4))
ax.set_ylim(3.5, -0.5)
for i in range(4):
    for j in range(4):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='cyan')
plt.show()
```

Figura 30: Código de matriz de confusão

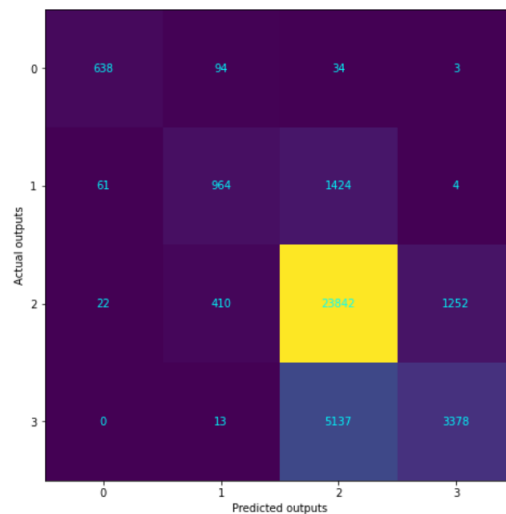


Figura 31: Apresentação da matriz de confusão

### 3 Conclusão

Concluimos que o modelo de regressão logística fez uma predição dos clusters muito positiva embora o resultado do K-Means não tenha sido o melhor uma vez que o dataset possuía valores irregulares.

Em geral fizemos uma boa análise dos dados.

Acerca dos desafios que enfrentámos foi a aplicação dos dois modelos ao dataset e podíamos ter melhorado a qualidade dos dados, por exemplo, eliminando rows que não faziam sentido, como casas que tinham 0 divisões embora possuissem 3 casas de banho, não fazendo qualquer sentido pois induzem os modelos aplicados em erro.