INSTITUTO POLITÉCNICO DE COIMBRA
INSTITUTO SUPERIOR DE CONTIBILIDADE E ADMINISTRAÇÃO DE COIMBRA
ANO LETIVO 2024/2025

Previsão do número de crimes na cidade de Los Angeles

Big Data

Mestrado em Análise de Dados e Sistemas de Apoio à Decisão

Trabalho Realizado Por:

Catarina Auxiliar Nº 2021134297

Diogo Machado Nº 2020153309

João Leal Nº 2021130506

Ana Silva Nº 2018074120

março, 2025

Índice

Índice de figuras

Introdução

Com este projeto iremos ter como objetivo prever o número de crimes em Los Angeles por área, aplicando conceitos de Big Data e Machine Learning utilizando o Apache Spark. O dataset escolhido é constituído por mais de dois milhões de registos de crimes ocorridos na cidade de Los Angeles. Este foi o dataset selecionado porque possui um grande volume de dados, permitindo a análise e modelação de padrões criminais ao longo do tempo. Por outro lado, os dados possuem características geográficas e temporais que serão essenciais para a futura análise.

Este projeto foi elaborado em 3 fases (além da presente introdução e da conclusão), com o auxílio do Apache Spark: A primeira fase correspondente à escolha do *dataset* e definição do problema; na segunda foi feita a exploração e transformação dos dados onde foi feita a limpeza, transformação e exploração do dataset para fornecer informação importante acerca dos dados em estudo; e a terceira correspondente ao treino e avaliação dos modelos aplicados.

1. Dados

Descrição do Dataset

O dataset “USA Big City Crime Data” está disponível no Kaggle e contem informações detalhadas sobre crimes registados em Los Angeles desde 2000. O dataset é constituído por 2 ficheiros csv “Chicago Crime Data” e “Crime_Data_from_2020_to_Present”, sendo apenas utilizado para análise o ficheiro “Crime_Data_from_2020_to_Present”. O ficheiro utilizado é constituído por 28 colunas e 944236 linhas.

Objetivo: Prever o número de crimes na cidade de Los Angeles

Tipo de problema: É um problema de regressão pois a variável alvo é um valor contínuo, isto é, o número de crimes por área.

Variável Alvo (Target): Crime_Count- Quantidade de crimes por área

Variáveis Explicativas (Features) selecionadas:

• AREA: Código de 1-21 atribuído a cada posto da polícia;
• DATE OCC: Dia em que ocorreu o crime;
• TIME OCC: Hora em que ocorreu o crime;
• Crm Cd Desc: Tipo de crime;
• LAT e LON- Coordenadas geográficas onde ocorreu o crime.

2. Limpeza dos dados

A limpeza dos dados é uma etapa imprescindível no processo de preparação de dados. Um *dataset* limpo permite uma melhor qualidade e integridade de dados, enquanto reduz o risco de erros na análise, permitindo tirar conclusões mais aproximadas da realidade.

2.1. Necessidades de formatação

Numa primeira análise conseguimos notar logo a necessidade de fazer algumas alterações:

1. Eliminar as horas nas colunas “Date RPTD” e “Date OCC”, deixando apenas a data;
2. Converter as horas da coluna “TIME OCC” para o formato HH:MM;
3. Renomear as colunas para uma melhor leitura.

Assim, o formato das colunas passa a ser:

Tabela 1 – Renomeação das colunas

Denominação Antiga	Denominação Nova
Date Rptd	Date_Reported
DATE OCC	Date_Occurred
TIME OCC	Time_Occurred
AREA	Area_Code
AREA NAME	Area_Name
Rpt Dist No	Report_District_Number
Part 1-2	Part1_2_Indicator
Crm Cd	Crime_Code
Crm Cd Desc	Crime_Description
Vict Age	Victim_Age
Vict Sex	Victim_Sex
Vict Descent	Victim_Descent
Premis Cd	Premise_Code
Status	Status_Code
Status Desc	Status_Description
Crm Cd 1	Crime_Code_1

Denominação Antiga	Denominação Nova
LOCATION	Location
LAT	Latitude
LON	Longitude

2.2. Valores Omissos

De seguida procurámos encontrar valores omissos, onde encontrámos os seguintes resultados:

Tabela 2 - Valores Omissos no dataset

Nome da Coluna	Nº de Valores Omissos
Mocodes	133099
Victim_Sex	126595
Victim_Descent	126605
Premise_Code	10
Premise_Description	567
Weapon_Used_Code	619758
Weapon_Description	619758
Crime_Code_1	11
Crime_Code_2	875977
Crime_Code_3	941954
Crime_Code_4	944171
Cross_Street	796643

Maior parte das colunas que contém valores omissos apresentam um grande volume dos mesmos, mas também informação não relevante, pelo que tomamos a decisão de eliminar essas colunas, com exceção das variáveis: “Victim_Sex” e “Victim_Descent” (ambas relevantes para o estudo); “Premise_Code” e “Crime_Code_1” (por terem poucos valores em falta).

Nas variáveis que mantivemos, tomamos a seguinte abordagem:

1. Para as variáveis “Victim_Sex” e “Victim_Descent”: Os valores nulos foram substituídos por ‘X’;
2. Para as variáveis “Premise_Code” e “Crime_Code_1”: Removemos as linhas com valores nulos.

2.3. Valores únicos

A contagem de valores únicos é uma abordagem essencial para entender a distribuição de uma variável categórica. Esta técnica permite identificar quantas categorias distintas estão presentes numa coluna de um dataset.

Neste contexto, os valores únicos referem-se à quantidade de categorias distintas dentro de uma variável.

A contagem de valores únicos é útil para:

- Entender a diversidade da variável: Permite verificar quantas classes diferentes existem num conjunto de dados.
- Preparar os dados para modelagem: Saber a quantidade de categorias ajuda na codificação de variáveis categóricas, como no uso de one-hot encoding ou label encoding.
- Detetar inconsistências: Um número excessivo de categorias pode indicar dados não limpos, como erros de digitação.
- Analisar a distribuição dos dados: Pode-se verificar se alguma categoria tem representação desproporcional em relação às demais.

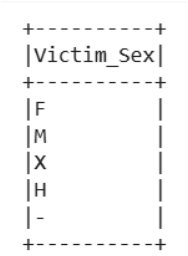


Figura 1 - Categoria de Victim_Sex antes da limpeza

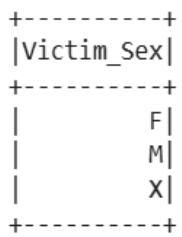


Figura 2 - Categoria de Victim_Sex depois da limpeza

Neste caso, através deste mecanismo, conseguimos detetar uma inconsistência na coluna Victim_Sex, que apresentava 5 categorias diferentes, quando deveria ter apenas 3 (F, M, X). Através do comando *df_new.select("Victim_Sex").distinct().show(truncate=False)* conseguimos encontrar as inconsistências (H e -), que posteriormente substituimos por X, conforme as figuras 1 e 2.

2.4. Duplicados

Valores duplicados podem distorcer análises estatísticas, influenciar negativamente modelos preditivos e comprometer decisões baseadas em dados, afetando a acurácia da análise, levar ao overfitting ou enviesamento de dados, e distorcer os *insights*.

Deste modo, foram detetadas 3954 linhas duplicadas, pelo que decidimos eliminá-las do dataset, conforme se observa na figura 3.

```
# Contar o número de linhas antes de remover duplicados
num_rows_before = df_new.count()

# Contar o número de linhas únicas, eliminando os duplicados
num_rows_unique = df_new.dropDuplicates().count()

# Calcular o número de duplicados
num_duplicated = num_rows_before - num_rows_unique

print(f"O dataset tem {num_rows_before} linhas antes da remoção de duplicados")
print(f"O dataset tem {num_duplicated} linhas duplicadas.")
```

```
O dataset tem 944214 linhas antes da remoção de duplicados
O dataset tem 3954 linhas duplicadas.
```

Figura 3 - Duplicados

3. Transformação dos dados

3.1. Codificação de variáveis categóricas

Uma forma eficiente de tratar de dados categóricos é a codificação das variáveis categóricas, uma vez que a maior parte dos algoritmos de *machine learning* operam com dados numéricos. O One-Hot-Encoding é um dos métodos de codificação mais conhecidos, onde é criada uma coluna binária para cada categoria. Se o atributo corresponder à categoria, recebe o valor 1, caso contrário recebe 0.

Sendo assim, para decidir o método de codificação a implementar, decidimos ver quantas categorias existem em cada variável categórica. Todas, exceto a coluna Location, apresentam uma quantidade razoável de categorias, pelo que decidimos aplicar o One-Hot-Encoding.

Na coluna Location poderíamos ter aplicado outro método de codificação, no entanto, dado que existem as colunas de latitude e longitude, decidimos não codificar a coluna categórica e utilizar as numéricas.

As figuras 4, 5 e 6 apresentam os passos que seguimos para aplicar o método de codificação.

```
#O primeiro passo é filtrar os valores únicos de cada coluna categórica

# Lista de variáveis categóricas
categorical_columns = ['Area_Name', 'Crime_Description', 'Victim_Sex',
                       'Victim_Descent', 'Status_Code', 'Status_Description', 'Location']

# Contar quantas categorias únicas existem em cada variável categórica
category_counts = {col_name: df_new.select(col_name).distinct().count() for col_name in categorical_columns}

# Mostrar os resultados
for col_name, count in category_counts.items():
    print(f"A coluna '{col_name}' tem {count} categorias únicas.")
```

```
A coluna 'Area_Name' tem 21 categorias únicas.
A coluna 'Crime_Description' tem 139 categorias únicas.
A coluna 'Victim_Sex' tem 3 categorias únicas.
A coluna 'Victim_Descent' tem 20 categorias únicas.
A coluna 'Status_Code' tem 6 categorias únicas.
A coluna 'Status_Description' tem 6 categorias únicas.
A coluna 'Location' tem 65694 categorias únicas.
```

Figura 4 - Categorias das variáveis categóricas

Area_Name_index	Crime_Description_index	Victim_Sex_index	Victim_Descent_index	Status_Code_index	Status_Description_index
9.0	0.0	0.0	4.0	2.0	2.0
0.0	2.0	0.0	4.0	0.0	0.0
3.0	21.0	2.0	1.0	0.0	0.0
12.0	25.0	0.0	4.0	0.0	0.0
4.0	3.0	0.0	0.0	0.0	0.0

Figura 5 – Índice das colunas codificadas

Area_Name_ohe	Crime_Description_ohe	Victim_Sex_ohe	Victim_Descent_ohe	Status_Code_ohe	Status_Description_ohe
(21,[9],[1.0])	(139,[0],[1.0])	(3,[0],[1.0])	(20,[4],[1.0])	(6,[2],[1.0])	(6,[2],[1.0])
(21,[0],[1.0])	(139,[2],[1.0])	(3,[0],[1.0])	(20,[4],[1.0])	(6,[0],[1.0])	(6,[0],[1.0])
(21,[3],[1.0])	(139,[21],[1.0])	(3,[2],[1.0])	(20,[1],[1.0])	(6,[0],[1.0])	(6,[0],[1.0])
(21,[12],[1.0])	(139,[25],[1.0])	(3,[0],[1.0])	(20,[4],[1.0])	(6,[0],[1.0])	(6,[0],[1.0])
(21,[4],[1.0])	(139,[3],[1.0])	(3,[0],[1.0])	(20,[0],[1.0])	(6,[0],[1.0])	(6,[0],[1.0])

Figura 6 - One-Hot-Encoding

3.2. Criação de *features* de datas

As features numéricas de datas são úteis para proporcionar visualizações sobre os padrões temporais. Por isso foram criados os atributos mês, dia da semana e hora. A figura 7 representa os comandos utilizados na criação dessas colunas.

```
# Criar novas colunas "Month", "DayOfWeek" e "Hour" a partir de "Date_Occurred" e "Time_Occurred"
df_encoded = df_encoded.withColumn("Month", month(col("Date_Occurred")))
df_encoded = df_encoded.withColumn("DayOfWeek", dayofweek(col("Date_Occurred")))
df_encoded = df_encoded.withColumn("Hour", hour(col("Time_Occurred")))

# Verificar colunas disponíveis após a adição das novas features
print("Available columns after adding date features:", df_encoded.columns)
```

Figura 7 - Features de datas criadas

3.3. Normalização de atributos numéricos

A normalização de atributos numéricos tem como objetivo padronizar os valores das variáveis, garantindo que todas tenham uma escala comum e facilitando a comparação entre diferentes atributos. Esta garante que variáveis com escalas diferentes não dominem modelos de *machine learning*, melhorando a convergência de algoritmos e tornando as análises mais intuitivas. Além disso, ela reduz a sensibilidade a *outliers*, contribuindo para a estabilidade e precisão dos resultados.

Existem alguns métodos para fazer a normalização, que escolhemos foi o StandardScaler, que padroniza os dados atribuindo uma média de 0 e um desvio-padrão de 1.

A figura 8 refere-se à aplicação do método de normalização, tendo sido selecionadas as colunas “Victim_Age”, “Latitude”, “Longitude”, “Month”, “DayOfWeek” e “Hour”.

```
# Definir as colunas numéricas que vão ser normalizadas
numeric_cols = ["Victim_Age", "Latitude", "Longitude", "Month", "DayOfWeek", "Hour"]

# Criar um vetor com as colunas numéricas
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="features", handleInvalid="skip")
vector_df = assembler.transform(df_encoded)

# Aplicar o StandardScaler para normalização
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
df_scaled = scaler.fit(vector_df).transform(vector_df)

# Mostrar um exemplo das transformações
df_scaled.select("features", "scaled_features").show(5)
```

Figura 8 - Normalização de atributos

3.4. Substituição dos nomes da categoria "Victim_Descent"

Para facilitar a leitura na análise das vítimas, decidimos trocar as categorias da variável "Victim_Descent" pelos seus respetivos nomes, uma vez que as letras que estavam atribuídas podem dificultar a leitura e a compreensão dos resultados obtidos. A figura 9 corresponde às categorias de étnias existentes.

Victim_Descent
Asian Indian
American Indian/A...
Chinese
Hawaiian
Japanese
Hispanic/Latin/Me...
Unknown
Filipino
Vietnamese
Other
Pacific Islander
Samoan
Guamanian
Korean
Laotian
White
Cambodian
Black
Other Asian

Figura 9 - Étnias

4. Análise e exploração dos dados

Esta secção está dividida em 2 partes e tem como objetivo procurar padrões nos dados.

A primeira parte corresponde a uma análise geral dos dados e criação de features de suporte (4.1, 4.2, 4.3 e 4.4); e a segunda parte corresponde à análise através de visualizações, que está subdividida em 3 subcategorias: a 1ª corresponde à vertente dos crimes, a 2ª corresponde à análise temporal dos crimes, e a 3ª à vertente da vítima.

4.1. Estatísticas descritivas

Esta fase tem o com intuito de realizar uma análise exploratória eficiente dos dados e identificar padrões, e variações e possíveis *outliers*.

```
Média:
Area_Code = 10.72
Report_District_Number = 1118.56
Part1_2_Indicator = 1.41
Crime_Code = 500.80
Victim_Age = 29.50
Premise_Code = 306.54
Status_Code = N/A
Crime_Code_1 = 500.54

Desvio Padrão:
Area_Code = 6.10
Report_District_Number = 609.96
Part1_2_Indicator = 0.49
Crime_Code = 207.28
Victim_Age = 21.88
Premise_Code = 217.55
Status_Code = N/A
Crime_Code_1 = 207.08

Mínimo:
Area_Code = 1.00
Report_District_Number = 101.00
Part1_2_Indicator = 1.00
Crime_Code = 110.00
Victim_Age = 0.00
Premise_Code = 101.00
Status_Code = AA
Crime_Code_1 = 110.00

Máximo:
Area_Code = 21.00
Report_District_Number = 2199.00
Part1_2_Indicator = 2.00
Crime_Code = 956.00
Victim_Age = 120.00
Premise_Code = 976.00
Status_Code = JO
Crime_Code_1 = 956.00

Variância:
Area_Code = 37.20
Report_District_Number = 372055.04
Part1_2_Indicator = 0.24
Crime_Code = 42966.06
Victim_Age = 478.77
Premise_Code = 47330.07
Status_Code = N/A
Crime_Code_1 = 42880.63

Moda:
Area_Code = 1.00
Report_District_Number = 162.00
Part1_2_Indicator = 1.00
Crime_Code = 510.00
Victim_Age = 0.00
Premise_Code = 101.00
```

Figura 10 - Estatísticas descritivas

Através da figura 10 conseguimos tirar as seguintes conclusões:

1 Média e Desvio Padrão:

- Victim_Age: Média de 29,5 anos e desvio padrão de 21,88, indicando uma grande variação na idade das vítimas.
- Crime_Code: Média de 500,8, sugerindo um possível padrão nos tipos de crimes registados.
- Report_District_Number: Média de 1118,56 e desvio padrão de 609,96, mostrando que os crimes ocorrem em diferentes distritos com frequências variadas.

2 Valores Mínimos e Máximos:

- Crime_Code: Varia entre 110 e 956, confirmando a existência de diferentes tipos de crimes registados.

3 Moda (Valor mais frequente):

- Crime mais comum: Código 510.
- Distrito policial mais frequente: Número 162. Estes valores indicam que esse crime e esse distrito são os mais recorrentes.

4.2. Outliers

```
from pyspark.sql.functions import col

#  Verificar se a coluna "Victim_Age" existe
if "Victim_Age" not in df_encoded.columns:
    raise ValueError("A coluna 'Victim_Age' não existe no DataFrame!")

#  Remover valores NULL antes de calcular o IQR ( Intervalo Interquartil)
df_encoded = df_encoded.na.drop(subset=["Victim_Age"])

#  Cálculo do IQR (Intervalo Interquartil)
Q1 = df_encoded.approxQuantile("Victim_Age", [0.25], 0)[0]
Q3 = df_encoded.approxQuantile("Victim_Age", [0.75], 0)[0]
IQR = Q3 - Q1

# Definição dos limites para outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtro nas vitimas cuja idade esteja fora dos limites estabelecidos
outliers = df_encoded.filter((col("Victim_Age") < lower_bound) | (col("Victim_Age") > upper_bound))

#  Mostrar resultados
print(f"Outliers encontrados: {outliers.count()}")
outliers.show(5)
```

```
Outliers encontrados: 1
+-------------+-------------+-----------------+-----------+--------------+----------------------+-----------------+----------+--------------------+----------+----------+--------------+-----------+--------------------+----------+--------------------+--------+---------+-----------+
|Date_Reported|Date_Occurred|Time_Occurred|Area_Code|Area_Name|Report_District_Number|Part1_2_Indicator|crime_code|  Crime_Description|Victim_Age|Victim_Sex|Victim_Descent|Premise_Code|Status_Code|Status_Description|Crime_Code_1|            Location|Latitude|Longitude|Crime_Index|
+-------------+-------------+-----------------+-----------+--------------+----------------------+-----------------+----------+--------------------+----------+----------+--------------+-----------+--------------------+----------+--------------------+--------+---------+-----------+
|   2020-04-19|   2020-04-19|            21:45|          8|  West LA|                   889|                1|       230|ASSAULT WITH DEAD...|     120.0|         X|             X|        101|          IC|       Invest Cont|         230|2000 S  BEDFORD  ...| 34.0416|-118.3836|        6.0|
+-------------+-------------+-----------------+-----------+--------------+----------------------+-----------------+----------+--------------------+----------+----------+--------------+-----------+--------------------+----------+--------------------+--------+---------+-----------+
```

Figura 11 - Outliers

Podemos verificar, através da figura 11 que existe um registo *outlier* relativo à variável “Victim_Age” utilizando o intervalo interquartil. Com o código utilizado verificamos que a coluna “Victiom_Age” está presente no Dataset seguindo a eliminação de valores nulos antes de efetuar o cálculo bem como os limites inferiores e superiores na identificação de valores fora do normal.

O resultado mostrou apenas existir um valor *outlier* de 120, que corresponde a uma vítima com 120 anos, onde está associado a um crime de com uma arma mortal (ASSAULT WITH DEADLY WEAPON) na área de West LA. Dada a idade da vítima, que é um caso raro, e através de pesquisas concluímos que esse valor se trata de um

erro do registo, uma vez que não existe histórico de um ataque alguma vítima com essa idade. Deste modo decidimos eliminar o resgisto do dataset.

4.3. Matriz de correlação

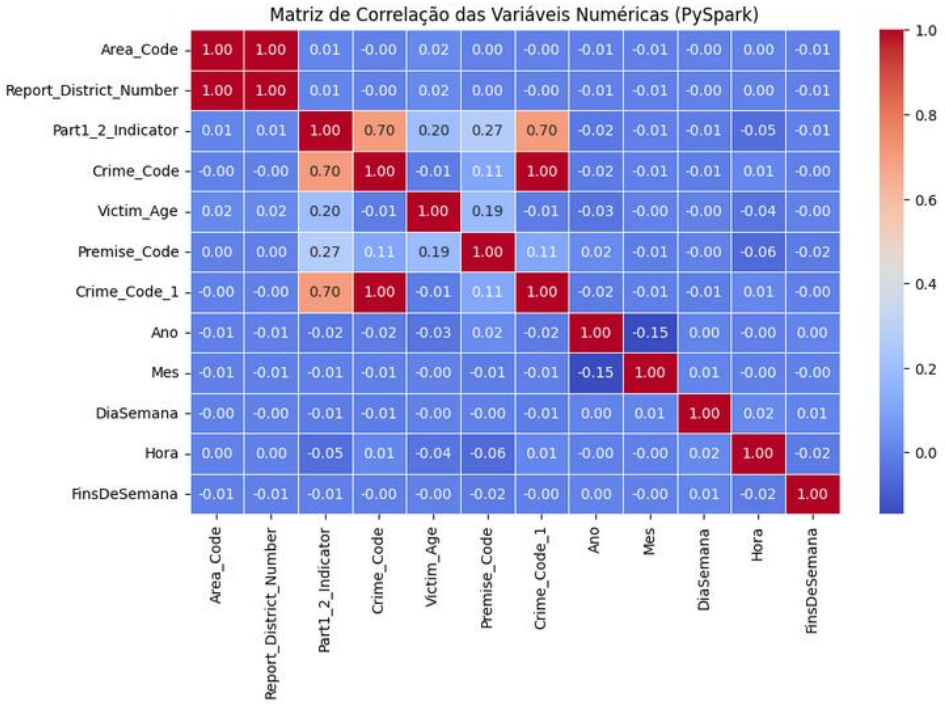


Figura 12 - Matriz de correlação

Para definir as variáveis numéricas usámos as mais relevantes, como Area_Code, Report_District_Number, Part1_2_Indicator, Crime_Code, Victim_Age, Premise_Code, Crime_Code_1, Ano, Mês, DiaSemana, Hora e FinsDeSemana.

Seguidamente usámos um VectorAssembler que foi utilizado para transformar as colunas num vetor único, possibilitando uma melhor análise estatística. Para a matriz de correlação foi então calculada utilizando a função Correlation.corr, sendo posteriormente convertida para um array NumPy para a visualização gráfica.

Sendo assim, podemos observar que existe uma correlação forte entre as variáveis “Part1_2_Indicator” e “Crime_Code” (0.70), o que possivelmente significa que os crimes de maior gravidade têm valores de código distintos dos menos graves.

Relativamente à correlação entre as variáveis “Premise_Code” e “Victim_Age” (0.19), podemos verificar que a idade da vítima pode estar associada ao local onde o crime ocorreu, por exemplo, crimes em escolas podem envolver vítimas jovens, enquanto crimes em lar de idosos envolvem vítimas mais velhas.

Concluímos também que, as variáveis temporais não apresentam fortes correlações com outras variáveis.

4.4. Criação de Novas Features no Dataset

```
from pyspark.sql.functions import year, month, dayofweek, hour, when

# Criar novas colunas baseadas nos nomes existentes
df_new = df_new.withColumn("Ano", year(col("Date_Occurred")))
df_new = df_new.withColumn("Mes", month(col("Date_Occurred")))
df_new = df_new.withColumn("DiaSemana", dayofweek(col("Date_Occurred")))
df_new = df_new.withColumn("Hora", hour(col("Time_Occurred")))
df_new = df_new.withColumn("FinsDeSemana", when(col("DiaSemana").isin([1, 7]), 1).otherwise(0))

# Verificar se as colunas foram criadas corretamente
df_new.select("Ano", "Mes", "DiaSemana", "Hora", "FinsDeSemana").show(10)
```

```
+----+---+---------+----+------------+
| Ano|Mes|DiaSemana|Hora|FinsDeSemana|
+----+---+---------+----+------------+
|2020|  3|        1|  21|           1|
|2020|  2|        7|  18|           1|
|2020| 11|        4|  17|           0|
|2020|  3|        3|  20|           0|
|2020|  8|        2|  12|           0|
|2020| 12|        3|  23|           0|
|2020|  7|        6|   9|           0|
|2020|  5|        3|  11|           0|
|2020| 12|        4|  14|           0|
|2020| 12|        5|  12|           0|
+----+---+---------+----+------------+
only showing top 10 rows
```

Figura 13 - Features novas

Neste passo criámos colunas temporais a partir de cada data e hora do crime (“Data_Occurred” e “Time_Ocurred”). Isto permite análises mais detalhadas sobre os padrões temporais dos crimes. Deste modo, conseguimos visualizar que todas as ocorrências registadas e que ocorreram em diferentes dias da semana, horas e meses.

O primeiro dia da semana corresponde ao número 1 (segunda-feira) e o último dia da semana corresponde ao número 7 (domingo).

4.5. Análise dos crimes

Distribuição dos crimes mais frequentes

```
from pyspark.sql.functions import col, count, desc

#  Contar o número de ocorrências por tipo de crime
df_crime_freq = df_new.groupBy("Crime_Description").agg(count("*").alias("Crime_Count"))

#  Ordenar os crimes do mais frequente para o menos frequente
df_crime_freq = df_crime_freq.orderBy(col("Crime_Count").desc())

#  Mostrar os 10 crimes mais comuns
df_crime_freq.show(10)
```

```
+--------------------+-----------+
|   Crime_Description|Crime_Count|
+--------------------+-----------+
|    VEHICLE - STOLEN|     102034|
|BATTERY - SIMPLE ...|      74509|
|BURGLARY FROM VEH...|      58309|
|   THEFT OF IDENTITY|      58240|
|            BURGLARY|      57497|
|VANDALISM - FELON...|      57193|
|ASSAULT WITH DEAD...|      53192|
|THEFT PLAIN - PET...|      48215|
|INTIMATE PARTNER ...|      46631|
|THEFT FROM MOTOR ...|      36611|
+--------------------+-----------+
only showing top 10 rows
```

Figura 14 - Top 10 crimes mais recentes (em tabela)

Através da tabela obtida na figura 14 conseguimos visualizar o top 10 de crimes cometidos na cidade de Los Angeles, com a descrição dos crimes e com o número de registos desses crimes praticados. Deste modo, o crime mais ocorrido são o roubo de veículos. Com base nestes dados, as forças de segurança podem adotar estratégias e medidas mais eficientes nas áreas onde ocorrem mais crimes, promovendo a segurança da população da cidade de Los Angeles.

```
[128] import matplotlib.pyplot as plt
      import pandas as pd

      # Converter para Pandas para visualização
      df_pandas = df_crime_freq.limit(10).toPandas()

      # Criar gráfico de barras
      plt.figure(figsize=(12,6))
      plt.barh(df_pandas["Crime_Description"], df_pandas["Crime_Count"], color="skyblue")
      plt.xlabel("Número de Ocorrências")
      plt.ylabel("Tipo de Crime")
      plt.title("Top 10 Crimes Mais Frequentes")
      plt.gca().invert_yaxis()  # Inverter a ordem para o mais frequente aparecer no topo
      plt.show()
```

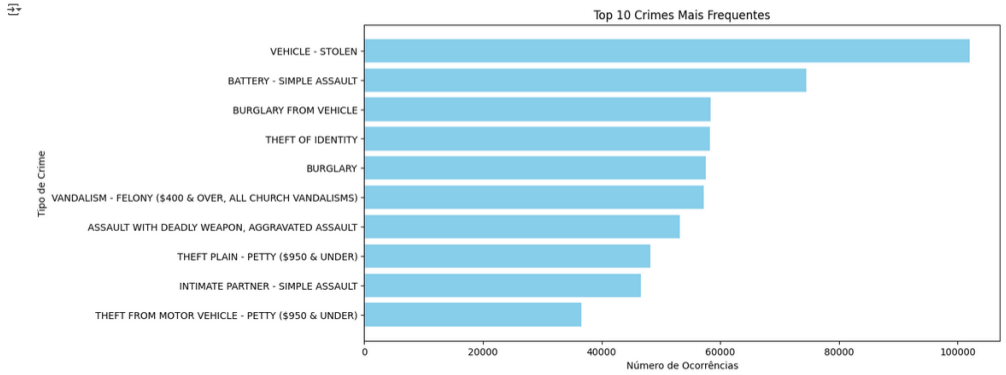


Figura 15 - Top 10 crimes mais recentes (em gráfico)

Nesta etapa do nosso projeto, utilizámos este código com o propósito de explorar graficamente os dados obtidos na figura 15, permitindo identificar rapidamente quais os crimes que ocorrem com mais frequência.

Distribuição de crimes por distrito

```
from pyspark.sql.functions import col, count, desc

#  Contar o número de crimes por distrito policial
df_crime_area = df_new.groupBy("Report_District_Number").agg(count("*").alias("Crime_Count"))
#  Ordenar as áreas com mais crimes
df_crime_area = df_crime_area.orderBy(col("Crime_Count").desc())

# Mostrar os 10 distritos com mais crimes
df_crime_area.show(10)
```

```
+----------------------+-----------+
|Report_District_Number|Crime_Count|
+----------------------+-----------+
|                   162|       4973|
|                   645|       4684|
|                  1494|       4519|
|                   182|       4480|
|                   646|       4156|
|                  2156|       3753|
|                   636|       3749|
|                   111|       3608|
|                  1822|       3101|
|                   152|       2931|
+----------------------+-----------+
only showing top 10 rows
```

Figura 16 - Distribuição de crimes por distrito

Para esta análise em concreto começámos por verificar se a coluna “Area_Name”, existia no conjunto de dados. De seguida procedemos ao agrupamento dos dados por área, bem como a contagem dos dados totais, gerando a coluna “Crime_Count”. O resultado mostra-

nos os dados agrupados por ordem decrescente e onde podemos observar as áreas com maior incidência criminal.

Através da utilização deste código identificamos os distritos policiais com maior número de crimes registados, permitindo quais as áreas com maior incidência criminal. Desta forma, o código realiza três operações principais. Em primeiro lugar, contabiliza os crimes por distrito policial. Em segundo lugar, ordena as áreas pelo número de crimes e, por último, exibe os dez distritos com mais crimes.

Sendo assim, conseguimos visualizar que os distritos 162, 645 e 1494, são os distritos com maiores níveis de criminalidade. Como os 10 distritos com maior criminalidade registam mais de 2900 crimes, é possível concluirmos que o crime está disperso em várias áreas.

Podemos concluir também que os distritos com mais habitantes tendem a registar mais crimes e, as áreas com mais tráfego de pessoas podem ser mais vulneráveis a assaltos.

```
#  Contar crimes por nome da área
if "Area_Name" in df_new.columns:
    df_crime_area_name = df_new.groupBy("Area_Name").agg(count("*").alias("Crime_Count"))
    df_crime_area_name = df_crime_area_name.orderBy(col("Crime_Count").desc())
    df_crime_area_name.show(10)
```

```
+-----------+-----------+
|  Area_Name|Crime_Count|
+-----------+-----------+
|    Central|      63725|
|77th Street|      59228|
|    Pacific|      55019|
|  Southwest|      53464|
|  Hollywood|      49487|
|N Hollywood|      47924|
|  Southeast|      47720|
|    Olympic|      47380|
|     Newton|      46517|
|   Wilshire|      45167|
+-----------+-----------+
only showing top 10 rows
```

Figura 17 - Contagem de crimes por área

Como a tabela anterior apenas nos indicava o número do distrito, realizámos agora uma análise pelo nome de cada área (figura 17) de forma a identificar a localidades mais afetadas.

Assim sendo, visualizamos que a zona central é a que tem mais impacto. Com 67.273 crimes, seguida de 77th Street e Pacific com cerca de 59.228 e 55.019 crimes respetivamente. Outras regiões, como Southwest, Hollywood e Newton também apresentam um elevado número de ocorrências.

Esta análise permite identificar as zonas com maior criminalidade, sendo útil para definir estratégias de policiamento e medidas de segurança.

```python
import pandas as pd
import folium
from folium.plugins import HeatMap

# Converter para Pandas para visualização
df_map = df_new.select("Latitude", "Longitude").dropna().limit(5000).toPandas()

# Criar um mapa centrado em Los Angeles
crime_map = folium.Map(location=[34.0522, -118.2437], zoom_start=10)

# Adicionar Heatmap ao mapa
HeatMap(df_map.values, radius=10).add_to(crime_map)

# Exibir o mapa
crime_map
```

Figura 18 - Código para a criação do heatmap

De seguida, fizemos a conversão do código para python utilizando as bibliotecas Folium e Pandas para gerar um mapa de calor. Apesar de não ser um código utilizado em pyspark mas sim em python achámos que a nível estético facilitava a visualização dos dados, apesar desta não ser um processamento Big Data.

Procedemos à conversão dos dados "Latitude" e "Longitude" para um DataFrame Pandas, removendo valores nulos e limitando a amostra a 5000 registos. De seguida, criou-se um mapa com a coordenadas geográficas de Los Angeles. A função "HeatMap()" foi usada para sobrepor o mapa de calor à visualização, sendo possível destacar as zonas com maior número de crimes. O resultado mostra as áreas mais afetadas que estão representadas com cores mais quentes e as menos afetadas representadas por cores mais frias

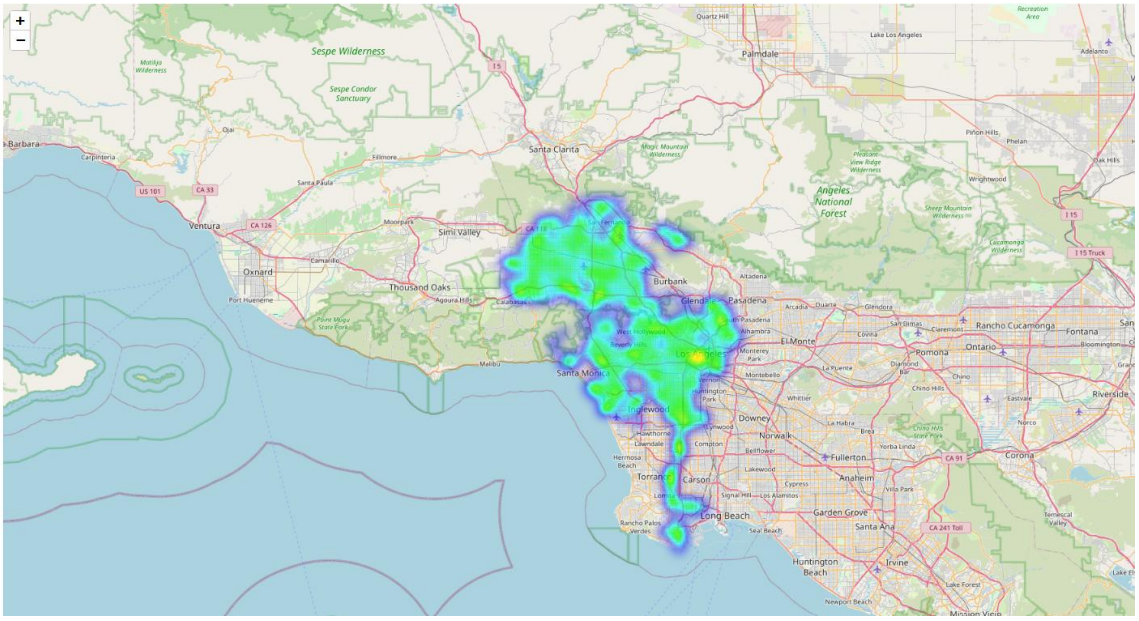


Figura 19 - Mapa de calor da distribuição de crimes por área

4.6. Análise temporal

Criação de Novas Colunas Temporais

```
#Densidade de Crimes por Área

# Contar o número de crimes por cada distrito policial
df_crime_density = df_new.groupBy("Report_District_Number").agg(count("*").alias("Crime_Density"))

# Juntar essa informação ao dataset original
df_new = df_new.join(df_crime_density, on="Report_District_Number", how="left")


# Categorizar Crimes por Hora do Dia

df_new = df_new.withColumn(
    "Hora_Categoria",
    when((col("Hora") >= 5) & (col("Hora") < 12), "Manhã")
    .when((col("Hora") >= 12) & (col("Hora") < 18), "Tarde")
    .when((col("Hora") >= 18) & (col("Hora") < 24), "Noite")
    .otherwise("Madrugada")
)


# . Criar a variável "É_Feriado" (0/1)

# Lista de feriados comuns nos EUA (podes adicionar mais)
feriados = [
    "2024-01-01", "2024-07-04", "2024-12-25",  # Ano Novo, Independência, Natal
    "2024-11-28", "2024-05-27"  # Ação de Graças, Memorial Day
]
feriados = [datetime.datetime.strptime(d, "%Y-%m-%d").date() for d in feriados]

# Criar a variável indicando se o crime ocorreu num feriado (1) ou não (0)
df_new = df_new.withColumn("É_Feriado", when(col("Date_Occurred").isin(feriados), 1).otherwise(0))


#  Verificar as novas features criadas

df_new.select("Report_District_Number", "Crime_Density", "Hora", "Hora_Categoria", "Date_Occurred", "É_Feriado").show(5)
```

```
+----------------------+-------------+----+--------------+-------------+---------+
|Report_District_Number|Crime_Density|Hora|Hora_Categoria|Date_Occurred|É_Feriado|
+----------------------+-------------+----+--------------+-------------+---------+
|                   784|          664|  21|         Noite|   2020-03-01|        0|
|                  1826|         1638|  23|         Noite|   2020-12-01|        0|
|                   182|         4480|  18|         Noite|   2020-02-08|        0|
|                   666|         2750|  12|         Tarde|   2020-08-17|        0|
|                   356|         1493|  17|         Tarde|   2020-11-04|        0|
+----------------------+-------------+----+--------------+-------------+---------+
only showing top 5 rows
```

Figura 20 - Criação de colunas temporais

Distribuição dos crimes por ano

```
# Contar o número de crimes por ano
df_crime_year = df_new.groupBy("Ano").agg(count("*").alias("Crime_Count"))

# Ordenar os anos em ordem crescente
df_crime_year = df_crime_year.orderBy("Ano")

# Converter para Pandas para visualização
df_pandas = df_crime_year.toPandas()

# Criar gráfico de linha para mostrar a tendência ao longo dos anos
plt.figure(figsize=(10, 5))
plt.ylim(180000, 250000)
plt.plot(df_pandas["Ano"], df_pandas["Crime_Count"], marker="o", linestyle="-", color="blue")
plt.xlabel("Ano")
plt.ylabel("Número de Crimes")
plt.title("Variação dos Crimes por Ano")
plt.grid(True)
plt.show()
```

Figura 21 - Distribuição de crimes por ano

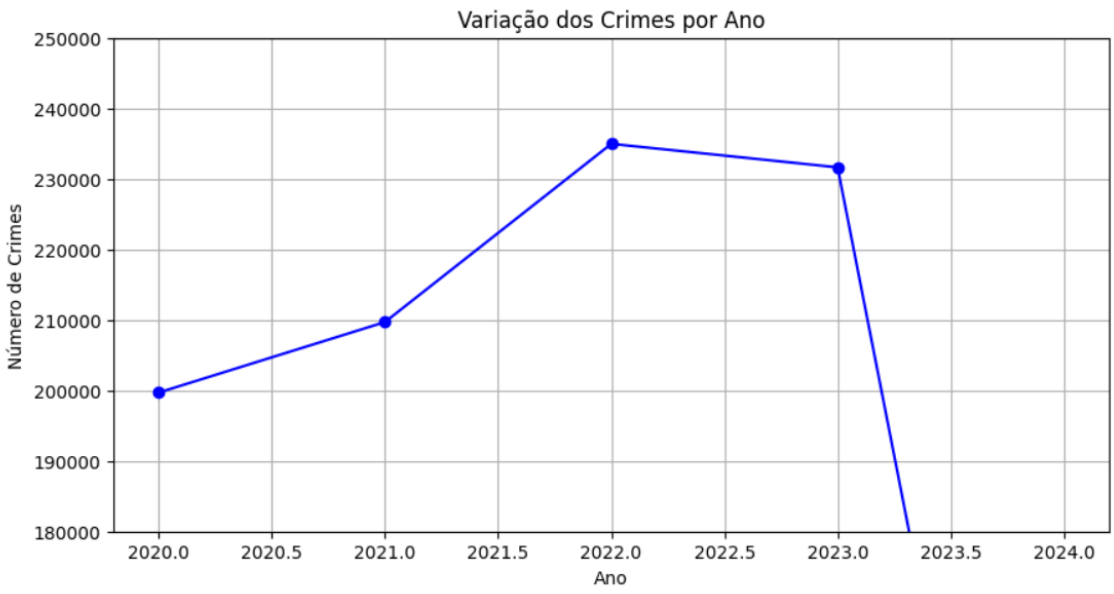


Figura 22 - Cronograma da distribuição de crimes por ano

Seguidamente procedeu-se à contagem dos crimes por ano e a organização dos dados, sendo os resultados convertidos para um DataFrame Pandas para ser a possível a visualização com Matplotlib. A figura 22 mostra um cronograma com a variação relativa ao número de crimes ao longo dos anos, utilizando o Pyspark para associar os dados anuais. observar a partir dos dados representados pela linha azul no gráfico que houve um aumento gradual do número de crimes entre 2020 e 2022, tendo sido registado o maior valor em 2022. Posteriormente nota-se uma ligeira diminuição em 2023 seguida de uma descida repentina, possivelmente sendo causada pela falta de dados relativos a esses anos.

Distribuição de crimes por dia da semana

```
# Converter o DataFrame Pyspark para pandas .
# Apenas a coluna "DayOfweek" é selecionada para a análise
df_pandas = df_encoded.select("DayOfWeek").toPandas()

# Criar um histograma da distribuição de crimes por dia da semana.
# Utilziamos o "bins=7" para garantir um intervalo descreto para os sete dias da semana
sns.histplot(df_pandas, x="DayOfWeek", bins=7, discrete=True)

# Definição do limites do eixo y para melhor visualização dos dados
plt.ylim(127500, 150000)

#Etiquetas dos eixos e título do gráfico
plt.xlabel("Dia da Semana")
plt.ylabel("Número de Crimes")
plt.title("Distribuição de Crimes por Dia da Semana")
plt.show()
```

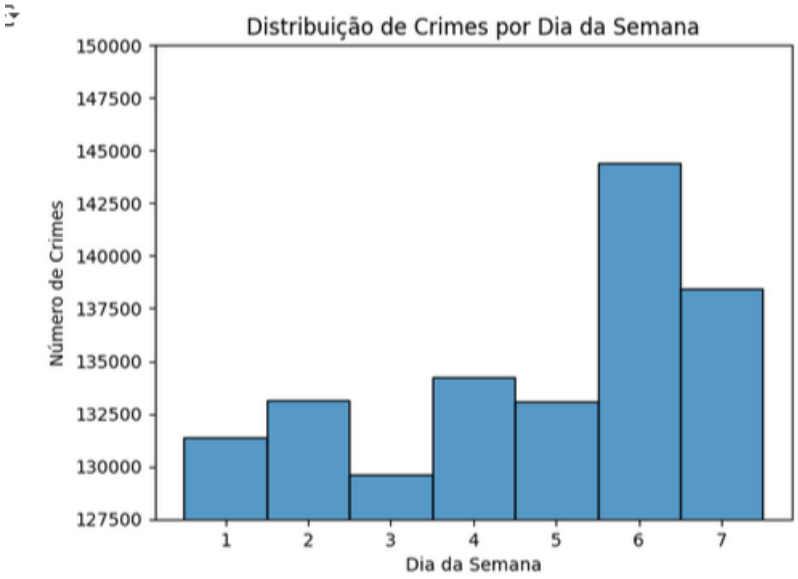


Figura 23 - Distribuição de crimes por dia da semana

Como forma de analisar a distribuição dos crimes ao longo dos dias da semana elaboramos, através do código em baixo explanado, um histograma.

Começámos por converter os dados para um DataFrame Pandas (toPandas()), para nos permitir visualizar a biblioteca Seaborn. O Eixo X, representa os dias da semana (numericamente de 1 a 7); Eixo Y, número de crimes por cada dia da semana.

Deste modo, podemos observar o número de crimes registado por cada dia da semana e, através dos dados fornecidos, verificamos que os dias em que se regista um maior número de crimes são no dia 6 e 7, o que provavelmente são sexta-feira e sábado, respetivamente. Ao longo da semana verificamos uma redução dos crimes registados em comparação com os dias 6 e 7. Isto talvez se deve ao facto de nos dias 6 e 7 existir maior população a passear pelas ruas e, maior frequência em bares noturnos, visto ser no fim de semana. Com base nesta análise, concluímos que nos dias de maior criminalidade, deve-se reforçar a presença das forças de segurança na cidade de Los Angeles, com o objetivo de reduzir a criminalidade nestes dois dias da semana.

Distribuição de crimes a diferentes horas do dia

```
from pyspark.sql.functions import col, count, desc

#  Contar crimes por hora do dia
df_crime_hour = df_new.groupBy("Hora").agg(count("*").alias("Crime_Count"))

#  Ordenar do horário com mais crimes para o menor
df_crime_hour = df_crime_hour.orderBy(col("Crime_Count").desc())

#  Mostrar os horários com mais crimes
df_crime_hour.show(10)
```

```
+----+-----------+
|Hora|Crime_Count|
+----+-----------+
|  12|      63876|
|  18|      56170|
|  17|      55099|
|  20|      52742|
|  19|      52080|
|  16|      49777|
|  15|      49464|
|  21|      47810|
|  14|      46245|
|  22|      46079|
+----+-----------+
only showing top 10 rows
```

Figura 24 - Distribuição de crimes por hora do dia

Podemos observar a distribuição dos crimes ao longo do dia estando agrupados pela hora em que ocorreu o crime. Para a codificação utilizámos a função groupBy(“Hora”) para contar o número total de crimes registados em cada hora do dia, criando a coluna Crime_Count. De seguida, os resultados foram ordenados por ordem decrescente, sendo possível observar os horários que têm mais ocorrências.

Ao analisar a tabela observamos que o horário em que ocorrem mais crimes é ao meio-dia (12 horas) e fim da tarde, podendo sugerir que os crimes aumentam durante a tarde e inicio da noite.

```
[43] from pyspark.sql.functions import when

     # Criar uma nova coluna categorizando os crimes por horário do dia
     df_new = df_new.withColumn(
         "Hora_Categoria",
         when((col("Hora") >= 5) & (col("Hora") < 12), "Manhã")
         .when((col("Hora") >= 12) & (col("Hora") < 18), "Tarde")
         .when((col("Hora") >= 18) & (col("Hora") < 24), "Noite")
         .otherwise("Madrugada")
     )

     # Contar o número de crimes por período do dia
     df_crime_period = df_new.groupBy("Hora_Categoria").agg(count("*").alias("Crime_Count"))

     # Ordenar os períodos pelo número de crimes
     df_crime_period = df_crime_period.orderBy(col("Crime_Count").desc())

     # Mostrar os períodos do dia com mais crimes
     df_crime_period.show()
```

```
+--------------+-----------+
|Hora_Categoria|Crime_Count|
+--------------+-----------+
|         Tarde|     307179|
|         Noite|     294619|
|         Manhã|     212738|
|     Madrugada|     129678|
+--------------+-----------+
```

Figura 25 - Categorização das horas do dia

Para esta análise, decidimos subdividir o número de crimes, de acordo com o período do dia em que ocorreram, utilizámos a função when() para criar uma nova coluna chamada Hora_Categoria dividindo as horas do dia em 4 tempos, isto é, Manhã (5h -12h), Tarde (12h-18h), Noite (18h-24h) e Madrugada (0h-5h). Depois agrupámos por período do dia (groupBy(“Hora_Categoria”) e o resultado na coluna “Crime_Count” .

Tal como observado na imagem anterior, aqui também é possível observar que as horas de maior incidência são a tarde e noite.

```
# Ordenar as horas corretamente de 0 a 23
df_crime_hour = df_crime_hour.orderBy(col("Hora").asc())

# Converter para Pandas
df_pandas = df_crime_hour.toPandas()

# Criar gráfico
plt.figure(figsize=(12,6))
plt.plot(df_pandas["Hora"], df_pandas["Crime_Count"], marker="o", linestyle="-", color="red")
plt.xlabel("Hora do Dia")
plt.ylabel("Número de Crimes")
plt.title("Distribuição de Crimes por Hora do Dia")

# Ajustar os rótulos do eixo x (para garantir que as horas aparecem corretamente)
plt.xticks(range(0, 24))

# Adicionar grade para melhor visualização
plt.grid()

# Exibir gráfico
plt.show()
```

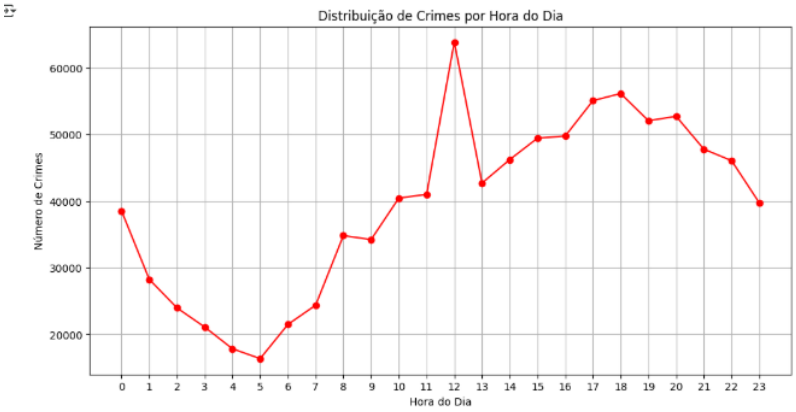


Figura 26 - Distribuição dos crimes por hora do dia

A figura 26 representa um gráfico de linhas que demonstra a distribuição dos crimes ao longo 24 horas. Para elaborar este gráfico foi necessário converter os dados originais para um DataFrame Pandas utilizando toPandas(), uma vez que a biblioteca Matplotlib não é compatível com o DataFrame do PySpark.

O gráfico gerado tem as seguintes características:

Eixo X: Representa as horas do dia, num intervalo de 0 a 23 horas

Eixo Y: Incide sobre o número total de crimes registados em cada hora

Verificamos assim que existe um pico que ronda a 12 horas, sendo que durante a tarde e início da noite também apresenta valores elevados. Durante a madrugada parece apresentar os valores mais baixos.

4.7. Características das vítimas

Por género

```
#Contar o número de crimes por género da vítima
df_crime_sex = df_new.groupBy("Victim_Sex").agg(count("*").alias("Crime_Count"))
df_crime_sex = df_crime_sex.orderBy("Crime_Count", ascending=False)

df_crime_sex.show()

#Converter para pandas
df_pandas_sex = df_crime_sex.toPandas()
```

```
+----------+-----------+
|Victim_Sex|Crime_Count|
+----------+-----------+
|         M|     385764|
|         F|     344144|
|         X|     214306|
+----------+-----------+
```

Figura 27 - Contagem de vítima por género

```
plt.figure(figsize=(8, 5))
sns.barplot(x="Victim_Sex", y="Crime_Count", data=df_pandas_sex, palette="Blues_r")

plt.xlabel("Sexo da Vítima")
plt.ylabel("Número de Crimes")
plt.title("Distribuição de Crimes por Sexo da Vítima")
plt.ylim(150000, df_pandas_sex["Crime_Count"].max() + 10000)  # Ajuste do eixo Y

plt.xticks(rotation=0)  # Garante que os rótulos fiquem visíveis
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

```
<ipython-input-73-112ca959b22e>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x="Victim_Sex", y="Crime_Count", data=df_pandas_sex, palette="Blues_r")
```

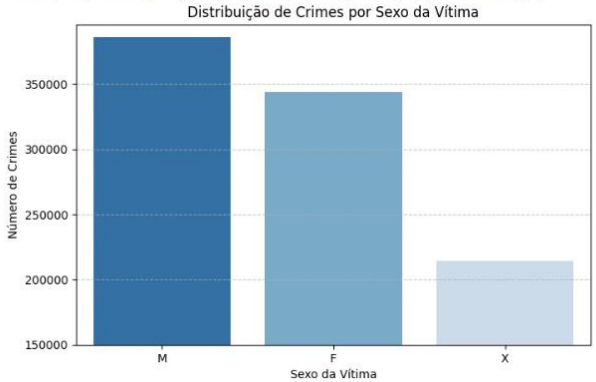


Figura 28 - Visualização do género das vítimas

Nesta figura, temos um gráfico de barras que ilustra a distribuição de crimes por sexo da vítima. Para gerar este gráfico recorremos à biblioteca Seaborn e ao Matplotlib. Com esta representação gráfica observamos os crimes são mais frequentes em vítimas do sexo masculino (M), seguidos pelas do sexo feminino (F). Existe uma outra categoria que representa as vítimas cujo sexo não foi identificado estando representadas pela letra (X).

Por etnia

```
# Agrupar crimes por etnia da vítima
df_crime_ethnicity = df_new.groupBy("Victim_Descent").agg(count("*").alias("Crime_Count"))

#Ordenar os dados do maior para o menor
df_pandas_ethnicity = df_pandas_ethnicity.sort_values(by="Crime_Count", ascending=False)

# Criar o gráfico de barras
plt.figure(figsize=(12, 6))
sns.barplot(x="Victim_Descent", y="Crime_Count", data=df_pandas_ethnicity, palette="Blues_r")

plt.xlabel("Etnia da Vítima")
plt.ylabel("Número de Crimes")
plt.title("Distribuição de Crimes por Etnia da Vítima")

plt.xticks(rotation=30, ha="right")  # Ajusta rotação e alinhamento
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()  # Evita sobreposição de elementos
plt.show()
```

```
<ipython-input-74-a033150de6af>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x="Victim_Descent", y="Crime_Count", data=df_pandas_ethnicity, palette="Blues_r")
```

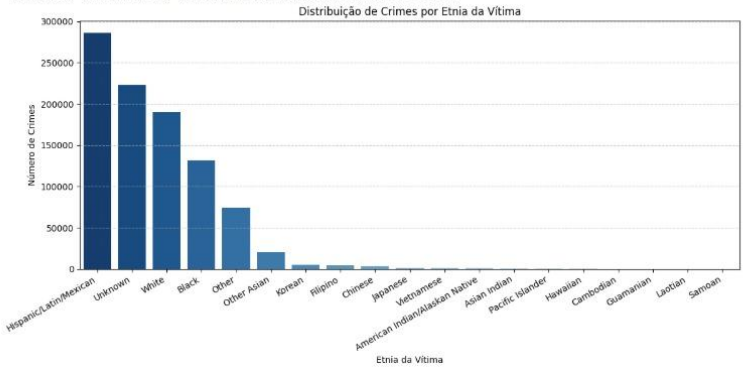


Figura 29 - Contagem de crimes por étnia

Na figura 29 temos a representação gráfica da distribuição dos crimes de acordo com a etnia das vítimas. Agrupou-se os dados pela coluna “Victim_Descent”, sendo contado o número total de acontecimentos para cada etnia. Em seguida, os dados foram ordenados do maior para o mais pequeno e gerou-se um gráfico de barras utilizando Seaborn e Matplotlib.

O gráfico indica que as etnias que mostram uma maior relevância são a Hispanic, Latino e Mexican, seguida por pessoas cuja etnia é desconhecida (Unknown) e pelas etnias Branca (White), Negra (Black) e Outras (Other). Também são apresentadas outras, como Filipino, Chinês, Japonês e Nativo Americano, que apresentam um número menor de ocorrências.

5. Treino do modelo

O treino é o processo no qual o modelo aprende com os dados conhecidos. Este permite exibir o desempenho do modelo, determinando se há necessidade de fazer melhoria do modelo. Esta etapa corresponde à quarta do *Machine Learning*.

Deste modo, nesta fase do projeto o foco é fazer a preparação do dataset para ser modelado.

De forma a melhorar a correlação e facilitar a análise de dados, foi criada uma variável chamada "Crime_Zone_Multi". Utilizámos esta abordagem para melhorar a correlação com a variável alvo, e para facilitar a identificação de padrões de forma a auxiliar a tomada de decisão.

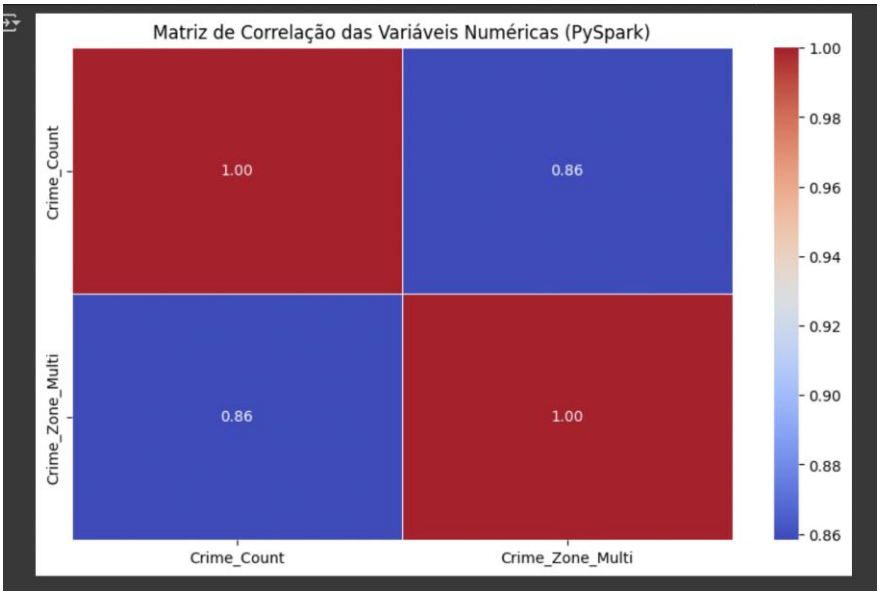


Figura 30 - Correlação entre a nova variável e a variável alvo

Como conseguimos verificar na figura 30, existe correlação forte entre a variável alvo e a variável nova (86%) faz sentido utilizarmos esta variável no modelo.

Depois foram selecionadas as *features* a utilizar no modelo, onde optámos por escolher "Area_Code", "Victim_Age", "Crime_Code" e "Crime_Zone_Multi".

Posteriormente, foi feita a vetorização das *features* e estabelecida variável-alvo – Crime_Count – através do Vector Assembler.

Por fim, foi feita a separação do *dataset* em treino e teste, pelo que foi escolhida a divisão de 70% para o treino e 30% para o teste.

Desta forma, o modelo está pronto para ser modelado.

6. Modelação

Uma vez que o objetivo do estudo é fazer uma previsão da criminalidade na cidade de Los Angeles, optámos por escolher modelos de regressão, entre os existentes escolhemos: Random Forest; Gradient Boosting; e Regressão Linear.

6.1. Random Forest Regressor

O *Random Forest* é um algoritmo de aprendizagem supervisionado que utiliza um método de aprendizagem em conjunto para a regressão.

Este método opera através da construção várias árvores de decisão, que não interagem entre si, fazendo a previsão para cada uma das árvores, individualmente.

A lógica deste algoritmo permite combinar o resultado de várias predições, que no final fornecem informação importante, permitindo fazer previsões futuras.

Resultados

```
R² no treino: 0.8107294101865609
RMSE no treino: 377.28321026964073
MAE no treino: 226.22072411072665
```

Figura 31 - Resultado no treino

```
R² no teste: 0.8105589432079287
RMSE no teste: 376.17934769151805
MAE no teste: 225.5487334774827
```

Figura 32 - Resultado no teste

Com base nas figuras 30 e 31, o desempenho do modelo quer no treino, quer no teste é praticamente igual, indicando que o modelo consegue generalizar bem.

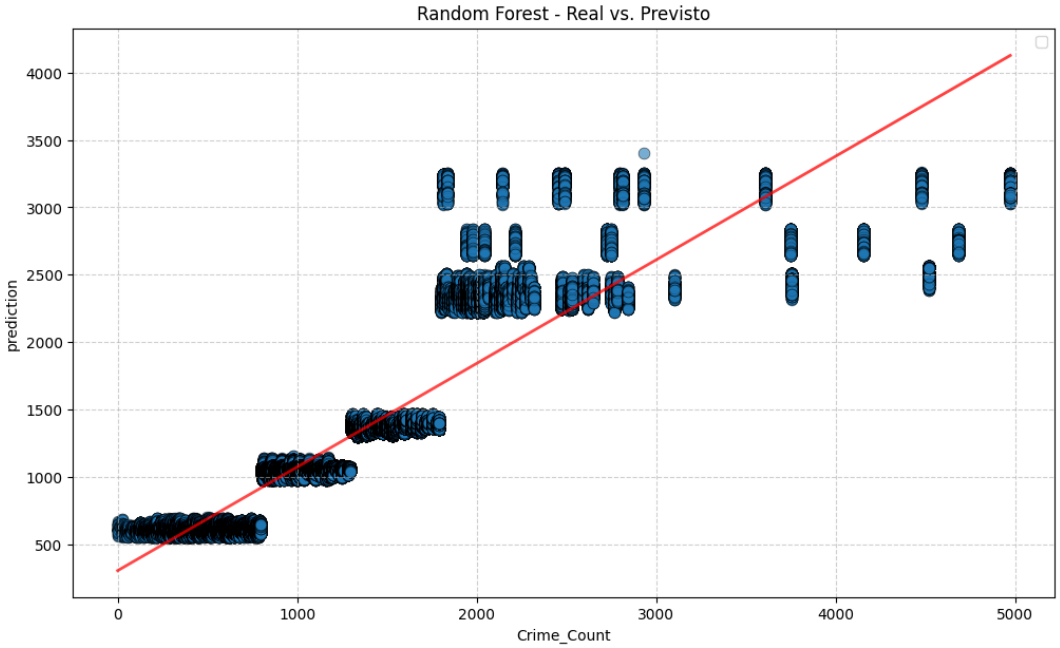


Figura 33 - Desempenho do Random Forest

Com base no gráfico de dispersão na figura 32, referente ao desempenho do modelo, concluímos que existe uma tendência crescente, indicando que o modelo é capaz de captar padrões dos dados, apesar dos desvios.

Algumas faixas revelam agrupamento de previsões, sobretudo nos valores médios e altos, o que sugere que o modelo não é capaz de lidar com a granularidade dos dados.

Os valores altos apresentam uma variação grande, o que pode indicar que o modelo está subajustado ou possui dados ruidosos ou insuficientes para esses intervalos (de 3000 crimes para cima).

Muitos pontos ficam abaixo da linha de tendência, o que indica que o modelo tende a subestimar os valores reais.

6.2. Gradiend Boosting para a regressão

O Gradient Boosting (GBT Regressor) é um estimador que constrói um modelo aditivo de forma progressiva. Deste modo, ele consegue otimizar as funções de perda que sejam diferenciáveis. Em cada etapa, uma árvore de regressão é ajustada ao gradiente negativo da função de perda fornecida.

Resultados

📌 Comparação dos Modelos (Treino vs. Teste):

	R^2	RMSE	MAE
GBTRegressor	0.848897	335.965019	200.506735
GBTRegressor_Treino	0.850147	335.705945	200.647461
GBTRegressor_Teste	0.848897	335.965019	200.506735

Figura 34 - Comparação dos resultados no treino e no teste

O modelo GBT Regressor apresentou um desempenho consistente entre treino e teste, indicando que ele generaliza bem sem sinais evidentes de *overfitting* ou *underfitting*. O coeficiente de determinação no treino e no teste são semelhantes, mostrando que a capacidade explicativa do modelo é praticamente a mesma em ambos os conjuntos de dados. Além disso, os erros médios também são muito próximos, com o erro quadrático médio (RMSE) de 335.71 no treino e 335.97 no teste, e o MAE (erro absoluto médio) de 200.65 no treino e 200.51 no teste, sugerindo que o modelo erra cerca de 200 crimes, em média, tanto no treino quanto no teste. No geral, o modelo está bem equilibrado e generaliza bem, mas ainda apresenta um erro médio considerável.

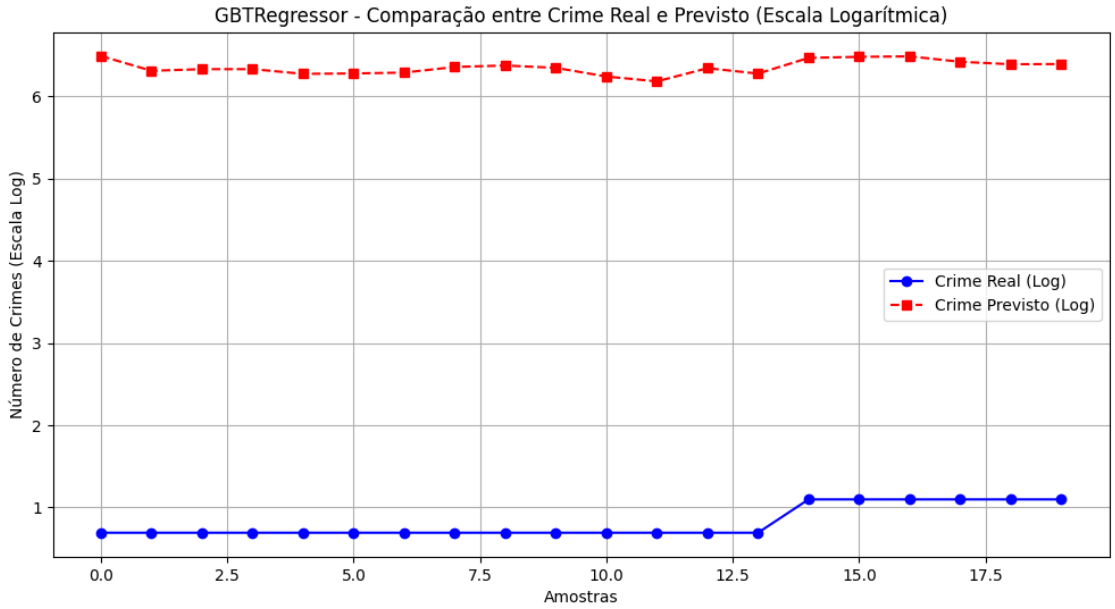


Figura 35 – Desempenho do GBT Regressor

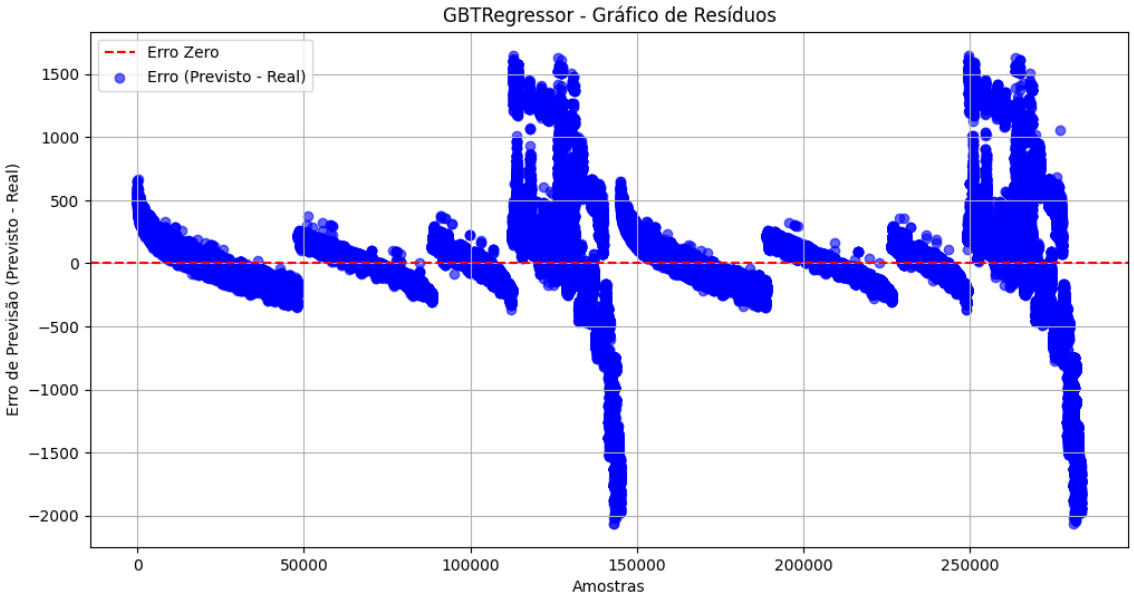


Figura 36 - Gráfico de resíduos do GBT Regressor

A avaliação do modelo na figura 34 mostrou que há uma discrepância significativa entre os valores reais (linha azul) e previstos (linha vermelha), indicando uma tendência do modelo de superestimar os valores de crime previsto.

O gráfico de resíduos na figura 35 indica que os erros do modelo não estão distribuídos de forma aleatória, o que sugere que ele não capturou completamente a variabilidade dos dados, uma vez que os resíduos deveriam estar dispersos aleatoriamente em torno da linha vermelha de erro zero, o que não se observa na figura. A estrutura do gráfico indica que o modelo cometeu erros sistemáticos em certas faixas de dados, possivelmente por não aprender corretamente determinados padrões. Além disso, há valores extremos de erro, com resíduos que ultrapassam 1500 tanto para cima quanto para baixo, o que sugere que o modelo está superestimou e subestimou previsões em diferentes situações. A presença de regiões onde os resíduos oscilam e depois voltam para perto da linha zero pode indicar que o modelo está tentando ajustar padrões não lineares, mas ainda enfrenta dificuldades em certas amostras.

6.3. Regressão Linear

A análise de regressão linear é utilizada para prever o valor de uma variável com base no valor de outra variável. Este modelo estima coeficientes da equação linear, envolvendo a(s) variável/variáveis independente(s) de forma a prever a variável dependente.

Resultados

```
Comparação dos Modelos (Treino vs. Teste):
                              R²        RMSE         MAE
LinearRegression_Treino  0.740022  442.175191  269.386501
LinearRegression_Teste   0.740912  439.927521  268.126983
```

Figura 37 - Comparação dos modelos de treino vs teste

Os resultados da Regressão Linear da figura 36 revelam um desempenho consistente entre treino e teste, com R^2 de 0.74 em ambos, indicando que o modelo explica 74% da variabilidade dos dados sem sinais de *overfitting*. O RMSE (440 crimes) e o MAE (270 crimes) são relativamente altos, sugerindo que o modelo ainda tem uma margem de erro significativa. A regressão linear pode não estar a capturar relações não lineares, o que pode ser melhorado com modelos mais complexos.

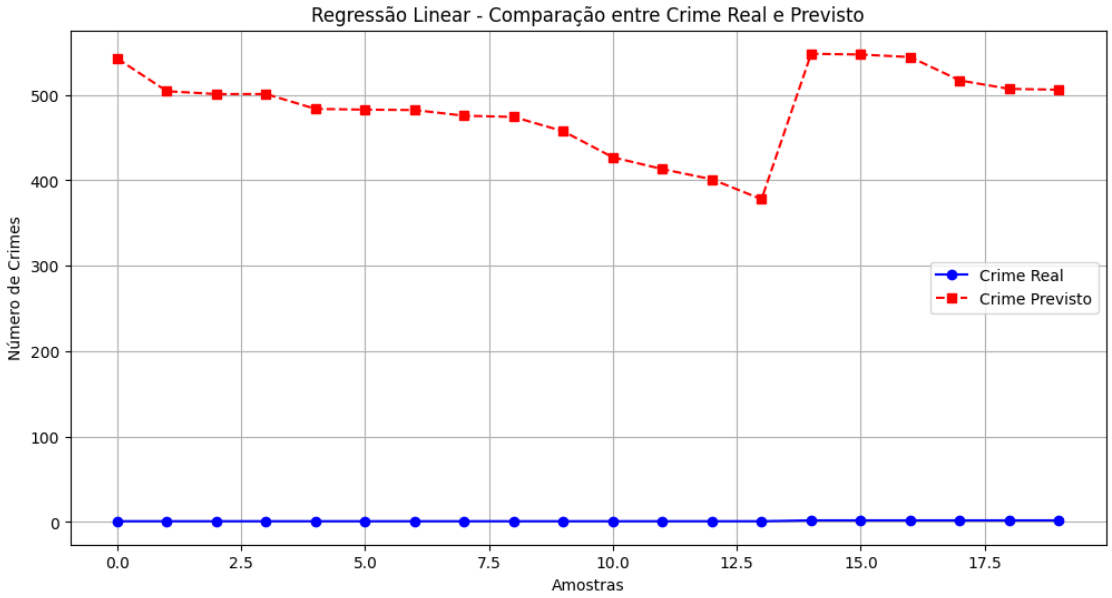


Figura 38 - Desempenho da Regressão Linear

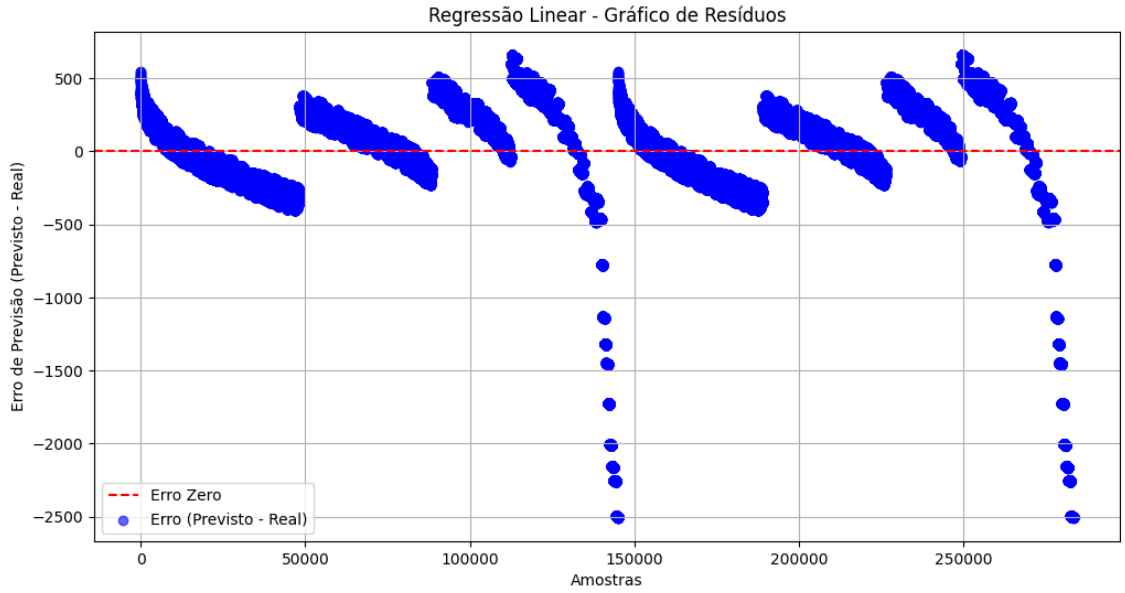


Figura 39 - Gráfico de resíduos da regressão linear

O gráfico da regressão linear mostra que os valores previstos são significativamente maiores que os valores reais, indicando que o modelo superestimou os crimes. A linha azul, que representa os valores reais, está quase fixa perto de zero, enquanto a linha vermelha, correspondente aos valores previstos, apresenta valores muito mais elevados. Isso sugere que a regressão linear pode não ter capturado os padrões dos dados corretamente.

O gráfico de resíduos da Regressão Linear apresenta um padrão bem definido, indicando que o modelo não está ajustado corretamente aos dados. Idealmente, os resíduos deveriam estar distribuídos aleatoriamente ao redor da linha de erro zero, mas o que se observa são agrupamentos organizados em diferentes faixas de valores. Existe uma predominância de resíduos negativos em algumas regiões, o que significa que o modelo subestimou o

número de crimes previstos nessas amostras. Há também valores extremamente negativos, que ultrapassam -2500, indicando erros muito grandes em certas previsões. Esses padrões reforçam que a Regressão Linear pode não ser a abordagem ideal para esse problema, já que o modelo tentou ajustar uma relação linear a um fenómeno que pode ter comportamentos mais complexos e não lineares.

Conclusões

Ao longo da análise, foram testados três modelos principais: Regressão Linear, GBTRegressor e Random Forest. Cada um teve um desempenho diferente, com pontos fortes e fracos.

1. Random Forest: Este modelo conseguiu reduzir os erros em relação à regressão linear, mas apresentou alta dispersão nas previsões para crimes de maior magnitude, sugerindo dificuldades na generalização. A proximidade entre os desempenhos no treino e teste indica que não houve *overfitting* severo, mas o modelo pode estar limitado pelo hiper parâmetros e memorização de padrões do treino. Comparado ao GBT Regressor, o desempenho foi inferior, especialmente para valores altos, sugerindo a necessidade de ajustes como otimização dos hiper parâmetros e *feature engineering* para melhorar a precisão das previsões.

2. GBTRegressor: Este modelo teve o melhor desempenho entre os três, conseguindo capturar mais padrões nos dados do que a regressão linear e o Random Forest. No entanto, os resíduos mostraram que o modelo ainda superestima alguns valores, sugerindo que pode ter aprendido padrões incorretos.

3. Regressão Linear: Este modelo apresentou um desempenho estável entre treino e teste, sem sinais de *overfitting*, mas teve dificuldades em capturar padrões mais complexos. Os erros foram elevados e os resíduos mostraram padrões estruturados, indicando que o modelo não está ajustado corretamente ao comportamento real dos crimes.

No geral, o GBTRegressor apresentou os melhores resultados, mas ainda há margem para melhorias. Os erros médios ainda são elevados, e todos os modelos apresentaram padrões estruturados nos resíduos, o que indica que nenhum dos modelos conseguiu capturar perfeitamente as relações nos dados.

Futuras etapas do projeto

Uma vez que houve uma certa dificuldade em fazer previsões, devido à falta de correlação entre as variáveis existentes no dataset, há ainda capacidade para melhorar o projeto, nomeadamente:

- Explorar Novas Variáveis[1]: Testar novas *features* baseadas em padrões temporais, geográficos e sociais pode melhorar a precisão dos modelos. Por exemplo, considerar a densidade populacional da região ou distribuição de recursos por área;
- Otimizar os Hiperparâmetros dos Modelos de forma a fornecer melhores resultados;
- Testar Modelos Mais Avançados como o XGBoost ou o LightGBM podem capturar padrões mais complexos e podem superar o desempenho do GBTRegressor e Random Forest.

[1] Ao longo do projeto houve uma tentativa de agregar novas variáveis ao estudo, porém houve dificuldade em conseguir fazê-lo de forma segura, uma vez que os dados tinham outra proveniência e não possuíam uma variável em comum. Pelo que decidimos descartar este plano. Em anexo encontram-se os códigos utilizados.

Webgrafia

Afroz Chakure. (2023, April 17). Random Forest Regression in Python Explained. https://builtin.com/data-science/random-forest-python

Etapas do Machine Learning. (n.d.). Retrieved March 20, 2025, from https://didatica.tech/etapas-do-machine-learning/

GradientBoostingRegressor. (n.d.). Retrieved March 20, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

MiddleHigh. (2024). USA Big City Crime Data. https://www.kaggle.com/datasets/middlehigh/los-angeles-crime-data-from-2000?resource=download&select=Crime_Data_from_2020_to_Present.csv

What is linear regression? (n.d.). Retrieved March 20, 2025, from https://www.ibm.com/think/topics/linear-regression

ANEXOS

Nota: As figuras incluídas nos anexos correspondem apenas ao *notebook* que foi criado como teste para tentar agregar duas variáveis de outro *dataset*, não fazendo parte dos *notebooks* principais.

```
# Verificar os nomes exatos das colunas
print("Colunas do Dataset de Pobreza:")
print(df_poverty.columns)

print("\nColunas do Dataset de Renda:")
print(df_income.columns)
```

```
[ ] # Renomear as colunas para facilitar o uso
df_poverty.rename(columns={"Estimate!!Total:!!Income in the past 12 months below poverty level:": "Poverty_Count"}, inplace=True)
df_income.rename(columns={"Estimate!!Median household income in the past 12 months (in 2020 inflation-adjusted dollars)": "Median_Income"}, inplace=True)

# Visualizar os datasets limpos
print("Dados de Pobreza:")
display(df_poverty.head())

print("Dados de Renda:")
display(df_income.head())
```

Dados de Pobreza:

	Geography	Geographic Area Name	Poverty_Count
0	1400000US06037101110	Census Tract 1011.10, Los Angeles County, Cali...	195
1	1400000US06037101122	Census Tract 1011.22, Los Angeles County, Cali...	93
2	1400000US06037101220	Census Tract 1012.20, Los Angeles County, Cali...	234
3	1400000US06037101221	Census Tract 1012.21, Los Angeles County, Cali...	283
4	1400000US06037101222	Census Tract 1012.22, Los Angeles County, Cali...	393

Dados de Renda:

	Geography	Geographic Area Name	Median_Income
0	1400000US06037101110	Census Tract 1011.10, Los Angeles County, Cali...	74625
1	1400000US06037101122	Census Tract 1011.22, Los Angeles County, Cali...	93125
2	1400000US06037101220	Census Tract 1012.20, Los Angeles County, Cali...	55682
3	1400000US06037101221	Census Tract 1012.21, Los Angeles County, Cali...	46274
4	1400000US06037101222	Census Tract 1012.22, Los Angeles County, Cali...	30016

```
# Mesclar os dados de renda e pobreza com base no identificador geográfico
df_combined = df_income.merge(df_poverty, on=["Geography", "Geographic Area Name"], how="left")

# Exibir os primeiros registros do dataframe combinado
print("Dados Combinados de Renda e Pobreza:")
display(df_combined.head())
```

Dados Combinados de Renda e Pobreza:

	Geography	Geographic Area Name	Median_Income	Poverty_Count
0	1400000US06037101110	Census Tract 1011.10, Los Angeles County, Cali...	74625	195
1	1400000US06037101122	Census Tract 1011.22, Los Angeles County, Cali...	93125	93
2	1400000US06037101220	Census Tract 1012.20, Los Angeles County, Cali...	55682	234
3	1400000US06037101221	Census Tract 1012.21, Los Angeles County, Cali...	46274	283
4	1400000US06037101222	Census Tract 1012.22, Los Angeles County, Cali...	30016	393

```
df_combined.count()
```

```
0
```

0	
Geography	2498
Geographic Area Name	2498
Median_Income	2498
Poverty_Count	2498

dtype: int64

```
print("Tipos de dados no df_combined:")
print(df_combined.dtypes)
```

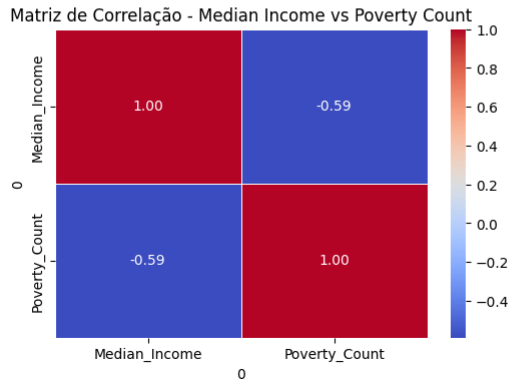
```
Tipos de dados no df_combined:
0
Geography               object
Geographic Area Name    object
Median_Income           object
Poverty_Count           object
dtype: object
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Selecionar apenas as colunas numéricas
df_combined_numeric = df_combined[["Median_Income", "Poverty_Count"]].astype(float)

# Calcular a matriz de correlação
correlation_matrix = df_combined_numeric.corr()

# Plotar o heatmap da matriz de correlação
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Matriz de Correlação - Median Income vs Poverty Count")
plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Criar cópias para normalização
df_combined_scaled = df_combined.copy()

# Standard Scaling (Z-Score)
standard_scaler = StandardScaler()
df_combined_scaled[["Median_Income_Std", "Poverty_Count_Std"]] = standard_scaler.fit_transform(df_combined[["Median_Income", "Poverty_Count"]])

# Ver os dados normalizados
print(df_combined_scaled.head())
```

```
from pyspark.sql import SparkSession

# Criar sessão Spark (caso ainda não tenha sido criada)
spark = SparkSession.builder.appName("CrimeData").getOrCreate()

# Caminho correto do arquivo no Google Drive
crime_csv_path = "/content/drive/My Drive/df_new_spark.csv"

# Carregar o CSV no PySpark
df_crimes = spark.read.csv(crime_csv_path, header=True, inferSchema=True)

# Verificar se carregou corretamente
df_crimes.printSchema()
df_crimes.show(5)
```

```python
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

# Converter df_crimes para Pandas
df_crimes_pd = df_crimes.toPandas()

# Criar uma coluna de geometria com Latitude e Longitude
df_crimes_pd["geometry"] = df_crimes_pd.apply(lambda row: Point(row["Longitude"], row["Latitude"]), axis=1)

# Criar um GeoDataFrame
gdf_crimes = gpd.GeoDataFrame(df_crimes_pd, geometry="geometry", crs="EPSG:4326")

# Verificar se os crimes estão corretamente convertidos
print(gdf_crimes.head())
```

```python
# Fazer um Spatial Join entre os crimes e os Census Tracts
gdf_crimes = gpd.sjoin(gdf_crimes, census_tracts, how="left", predicate="within")

# Verificar se cada crime recebeu um Census Tract
print(gdf_crimes.head())
```

```python
[ ] # 1 Unir os crimes com os dados socioeconômicos usando o Census Tract
df_final = gdf_crimes.merge(df_combined, left_on="TRACTCE", right_on="Geography", how="left")

# 2 Remover a coluna `geometry` para evitar erro no PySpark
df_final = df_final.drop(columns=["geometry"])

# 3 Converter para PySpark sem erro
df_final_spark = spark.createDataFrame(df_final)

# 4 Verificar os primeiros resultados
df_final_spark.show(5)
```

```python
from pyspark.sql.functions import col, mean

# 1 Criar um DataFrame com a média de renda e pobreza por TRACTCE
df_avg = df_final_spark.groupBy("TRACTCE").agg(
    mean("Median_Income").alias("avg_income"),
    mean("Poverty_Count").alias("avg_poverty")
)

# 2 Aplicar `fillna()` diretamente sem precisar de `when()`
df_final_spark = df_final_spark.join(df_avg, on="TRACTCE", how="left").fillna({
    "Median_Income": df_avg.select("avg_income").first()[0],
    "Poverty_Count": df_avg.select("avg_poverty").first()[0]
})

# 3 Remover colunas auxiliares
df_final_spark = df_final_spark.drop("avg_income", "avg_poverty")

# 4 Mostrar resultado final
df_final_spark.show(5)
```