PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Ana Paula Salgado Reis

DETECÇÃO DE ANOMALIAS NOS REGISTROS DE CORRIDAS DE TÁXI DOS SERVIDORES E COLABORADORES DA ADMINISTRAÇÃO PÚBLICA FEDERAL

Belo Horizonte
2022

Ana Paula Salgado Reis

DETECÇÃO DE ANOMALIAS NOS REGISTROS DE CORRIDAS DE TÁXI DOS SERVIDORES E COLABORADORES DA ADMINISTRAÇÃO PÚBLICA FEDERAL

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2022

SUMÁRIO

1. Introdução

1.1. Contextualização

Na Administração Pública Federal, há a necessidade de deslocamentos a trabalho por parte dos seus servidores e colaboradores. No entanto, não existia um padrão de serviço entre os órgãos. A frota própria era um modelo frequente, mas que gerava alto grau de imobilização de recursos e elevado esforço de gestão, envolvendo manutenção de veículos, reposição de peças, organização de garagens, administração de seguros e procedimentos de desfazimento.

Tendo em vista o mercado atual, com a disponibilidade de aplicativos e sistemas que facilitam a solicitação de veículos e o monitoramento da utilização dos serviços, o governo federal propôs o desenvolvimento de uma solução para substituir os modelos de carros próprios e alugados por táxis, pagando-se apenas pelo serviço que foi efetivamente utilizado.

Dessa forma, com o objetivo de melhorar a oferta de serviços de transporte com economia, transparência e eficiência, em março de 2017, começou a ser implantado o TáxiGov. Esse serviço é utilizado em deslocamentos a trabalho com o uso de táxis. Ele foi implantando, primeiramente, no Distrito Federal. Hoje já está em operação também em alguns municípios de São Paulo e do Rio de Janeiro. Segundo informações do Ministério de Economia, a previsão é de ser implantado em todas as capitais do Brasil até o final de 2022.

De acordo com o Decreto Federal de nº 9.287, de 15 de fevereiro de 2018, que trata sobre a utilização de veículos oficiais pela administração pública federal, são permitidos os deslocamentos a trabalho, como reuniões, entrega de documentos, visitas técnicas, capacitação, entre outros. Também podem ser realizadas viagens fora do estado no qual o servidor é lotado. No entanto, nesses casos, é recomendado que o servidor entre em contato com o Gestor Setorial da unidade que será realizada a viagem para ajustar o cadastramento do perfil de usuário no TáxiGov correspondente.

O uso do serviço é proibido nos casos de deslocamentos por interesse pessoal e/ou viagens a passeio ou lazer; viagens entre residência e local de trabalho, exceto em casos de áreas de difícil acesso ou que não possuam transporte

público regular; uso aos sábados, domingos e feriados, exceto para eventual desempenho de encargos inerentes ao exercício da função pública ou nas hipóteses previstas nos incisos VIII e IX do caput do art. 5 do Decreto nº 9.287/18; e deslocamentos para aeroportos, se o servidor receber indenização.

Segundo dados do Ministério de Economia, desde a implantação do TáxiGov foram realizadas 918 mil viagens, foram 6,2 milhões de quilômetros rodados e resultou na economia de 33,8 milhões de reais aos cofres públicos.

1.2. O problema proposto

No ano de 2021, o TáxiGov registrou um gasto acima de R$ 3 milhões de reais com os deslocamentos feitos pelos servidores e colaboradores das esferas do governo federal que já implantaram o serviço. Até o momento, a implantação foi feita no Distrito Federal e em alguns municípios dos estados de São Paulo e do Rio de Janeiro, mas há a expectativa de expansão do serviço para todas as capitais do Brasil até o final de 2022. Logo, a tendência é que os valores gastos com corridas de táxis registrados no TáxiGov sejam cada vez maiores. Dessa forma, a análise e controle desses gastos é de grande relevância.

Para isso, serão analisados os registros das corridas disponibilizados no Portal de Dados Abertos do Governo Brasileiro e que serão explicados detalhadamente no item 2.

A análise dos dados buscará identificar registros fora dos padrões, em que os valores das corridas, as distâncias percorridas e as durações dos percursos não estejam em conformidade com o que é esperado para cada caso.

Após essa etapa, uma possível aplicação seria a realização de uma auditoria para avaliar os registros levantados de forma aprofundada, identificando até mesmo possíveis fraudes. Também poderia ser feita a automatização dessa análise, com a identificação de anomalias e a emissão de alertas em tempo real, ou seja, assim que o registro fosse feito no TáxiGov. Essas iniciativas aumentariam a transparência e poderiam prevenir gastos indevidos, que geram prejuízos aos cofres públicos.

1.3. Objetivos

O objetivo do presente trabalho é identificar os registros de corridas de táxi feitos através do sistema TáxiGov que apresentem possíveis inconsistências. Para isso, serão identificadas as anomalias presentes nos registros.

2. Coleta de Dados

Os dados dos registros das corridas de táxi realizadas através do TáxiGov foram obtidos no Portal de Dados Abertos do Governo Brasileiro. No endereço eletrônico abaixo, são mostradas as diversas opções de exportação dos dados:

https://dados.gov.br/dataset/corridas-do-taxigov.

Para a realização do trabalho, foram exportados os registros feitos nos últimos sete dias. A extração dos dados foi feita no dia 09 de abril de 2022, através do endereço eletrônico a seguir, e os dados abarcam as corridas de táxi realizadas entre os dias 31 de março e 06 de abril de 2022.

http://repositorio.dados.gov.br/seges/taxigov/taxigov-corridas-7-dias.zip.

O arquivo foi exportado em formato *.csv* (valores separados por vírgulas) e possui os campos descritos na Tabela a seguir.

Tabela 1 – Campos do arquivo CSV com os registros do TáxiGov.

Coluna	Tipo	Descrição
base_origem	VarChar	Nome atribuído ao vínculo do Contrato. Ex.: TAXIGOV_DF_10, TAXIGOV_DF_21, TAXIGOV_DF_00, TAXIGOV_MG_10, TAXIGOV_MT_10
qru_corrida	int	Código da Corrida
nome_orgao	VarChar	Nome ou Sigla do Órgão Superior / Entidade
data_abertura	DateTime	Data e Hora de abertura do chamado
data_despacho	DateTime	Data e Hora para despacho do veículo
data_local_embarque	DateTime	Data e Hora de chegada do veículo ao local de embarque
data_inicio	DateTime	Data e Hora do Início da Corrida
data_final	DateTime	Date e Hora de Término da Corrida
ateste_setorial_data	DateTime	Data e Hora do Ateste feito pelo responsável Setorial
status_corrida	nvarchar	Status da Corrida
km_total	Decimal	Registro da km total da corrida
valor_corrida	Decimal	Valor da Corrida

origem_endereco	nvarchar	Endereço de Origem da Corrida
origem_bairro	nvarchar	Bairro de Origem da Corrida
origem_cidade	nvarchar	Cidade de Origem da Corrida
origem_uf	nvarchar	Estado ou Região de Origem da Corrida
destino_solicitado_endereco	VarChar	Endereço de Destino Solicitado
destino_efetivo_endereco	VarChar	Endereço de Destino Efetivo
origem_latitude	VarChar	Coordenada de Latitude da Origem
origem_longitude	VarChar	Coordenada de Longitude da Origem
destino_solicitado_latitude	VarChar	Coordenada de Latitude do Destino Solicitado
destino_solicitado_longitude	VarChar	Coordenada de Longitude do Destino Solicitado
destino_efetivo_latitude	VarChar	Coordenada de Latitude do Destino Efetivo
destino_efetivo_longitude	VarChar	Coordenada de Longitude do Destino Efetivo
conteste_info	VarChar	Informações de Conteste

Para comparar algumas informações dos registros do TáxiGov, foi criada uma planilha em Excel, nomeada de *"preco-km-rodado"*, em formato *.xlsx*, com dados pesquisados de preço por quilômetro rodado pelos táxis nas bandeiras 1 e 2 de alguns municípios. Também foram adicionadas as informações do valor da bandeirada e da tarifa horária. Todos os campos podem ser vistos na Tabela abaixo.

Tabela 2 – Campos da planilha "preco-km-rodado".

Coluna	Tipo	Descrição
Cidade	Texto	Município de onde são as informações
R$/km (bandeira 1)	Decimal	Valor cobrado por quilômetro no horário da bandeira 1
R$/km (bandeira 2)	Decimal	Valor cobrado por quilômetro no horário da bandeira 2
Bandeirada	Decimal	Quantia fixa mínima cobrada pela corrida
Tarifa horária	Decimal	Valor cobrado quando o táxi se encontra parado ou circulando com uma velocidade abaixo de 15 km/h

Data da informação	Texto	Data em que foi feita a última alteração dos valores

O relacionamento entre os datasets utilizados foi feito através do campo “origem_cidade” da Tabela 1 e “Cidade” da Tabela 2.

3. Processamento/Tratamento de Dados

Para a análise, foi criado um *Jupyter Notebook*. Primeiramente, foram importadas algumas bibliotecas que seriam necessárias.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import datetime
```

Figura 1 – Importação de bibliotecas.

Em seguida, foi criado um *dataframe* (*df)* a partir da leitura do arquivo *.csv*, no qual estão os registros do TáxiGov que serão analisados, do período entre os dias 31 de março e 06 de abril de 2022.

Para verificar a criação do *dataframe* e analisar sua dimensão e tipos de dados, foram feitos os comandos mostrados na Figura abaixo.

```
df = pd.read_csv('taxigov-corridas-7-dias-vF.csv')
```

```
df.head()
```

	base_origem	qru_corrida	nome_orgao	status_corrida	motivo_corrida	km_total	valor_corrida	data_abertura	data_despacho	data_local_embarc
0	TAXIGOV_DF	514588.0	Ministério da Economia	CONCLUÍDA	3 - VISITA TECNICA	30.62	96.77	2022-03-31 07:16:12.600	2022-03-31 07:16:21.943	2022-03 07:18:46.7
1	TAXIGOV_DF	514592.0	Ministério Cidadania	CONCLUÍDA	1 - REUNIAO EXTERNA	11.98	37.86	2022-03-31 07:27:43.707	2022-03-31 07:29:11.167	2022-03 07:46:46.9
2	TAXIGOV_DF	514612.0	MINISTERIO DO DESENVOLVIMENTO REGIONAL	CONCLUÍDA	1 - REUNIAO EXTERNA	5.84	18.44	2022-03-31 08:07:01.683	2022-03-31 08:07:14.303	2022-03 08:09:47.(
3	TAXIGOV_DF	514616.0	MAPA - Ministério da Agricultura - Pecuária e ...	CONCLUÍDA	1 - REUNIAO EXTERNA	12.46	39.36	2022-03-31 08:10:56.283	2022-03-31 08:11:50.220	2022-03 08:16:10.5
4	TAXIGOV_DF	514623.0	MINISTERIO DO DESENVOLVIMENTO REGIONAL	CONCLUÍDA	1 - REUNIAO EXTERNA	11.15	35.25	2022-03-31 08:16:05.157	2022-03-31 08:16:48.130	2022-03 08:20:47.9

5 rows × 26 columns

```
df.shape
```

```
(2837, 26)
```

```
df.dtypes
```

```
base_origem                  object
qru_corrida                 float64
nome_orgao                   object
status_corrida               object
motivo_corrida               object
km_total                    float64
valor_corrida               float64
data_abertura                object
data_despacho                object
data_local_embarque          object
data_inicio                  object
data_final                   object
origem_endereco              object
origem_bairro                object
origem_cidade                object
origem_uf                    object
destino_solicitado_endereco  object
destino_efetivo_endereco     object
origem_latitude             float64
origem_longitude            float64
destino_solicitado_latitude float64
destino_solicitado_longitude float64
destino_efetivo_latitude    float64
destino_efetivo_longitude   float64
ateste_setorial_data         object
conteste_info               float64
dtype: object
```

Figura 2 – Criação do *dataframe* e algumas análises.

Para analisar as estatísticas descritivas dos dados do *dataframe*, foi dado o comando *describe* e os resultados podem ser vistos na Figura a seguir.

```
df.describe()
```

	qru_corrida	km_total	valor_corrida	origem_latitude	origem_longitude	destino_solicitado_latitude	destino_solicitado_longitude	destino_efetivo_latitude	destino_efetivo_longitude	conteste_info
count	2836.00000	2498.000000	2682.000000	2.837000e+03	2837.000000	2.832000e+03	2.830000e+03	2499.000000	2499.000000	0.0
mean	561140.92842	8.940024	25.427696	-2.244235e+03	-47.075693	-4.480527e+03	-6.813433e+03	-17.102245	-47.059593	NaN
std	104386.27541	22.838921	33.473021	5.928162e+04	1.760276	8.387525e+04	1.798821e+05	2.753874	1.776091	NaN
min	223680.00000	0.010000	0.000000	-1.579626e+06	-48.289545	-1.581422e+06	-4.787247e+06	-23.652500	-48.276390	NaN
25%	516498.50000	2.310000	6.550000	-1.582977e+01	-47.890239	-1.583636e+01	-4.789034e+01	-15.836714	-47.890464	NaN
50%	518542.50000	4.827500	13.320000	-1.579953e+01	-47.872008	-1.579962e+01	-4.787215e+01	-15.799758	-47.872354	NaN
75%	520669.75000	10.967500	31.935000	-1.579482e+01	-47.861970	-1.579500e+01	-4.786324e+01	-15.794817	-47.864609	NaN
max	784629.00000	1000.000000	474.040000	-1.561983e+01	-43.065887	-1.561272e+01	-4.296608e+01	-15.619250	-42.966100	NaN

Figura 3 – Estatística descritiva dos dados.

Como a coluna *conteste_info* está vazia, decidiu-se deletá-la.

```
del df["conteste_info"]
```

Figura 4 – Exclusão da coluna *conteste_info*.

O *dataframe* apresenta 2837 registros. Observa-se pelos valores encontrados no *count* da Figura 3 que existem dados faltando em diversas colunas.

A coluna *qru_corrida* representa o código da corrida. Pode ser visto que existe uma linha sem valor, o que não deveria existir. O valor faltante será então preenchido com o número 784630, que é o valor máximo (784629) somado de um, para não repetir nenhum número.

```
df['qru_corrida'].fillna(784630.0, inplace=True)
```

Figura 5 – Preenchendo o valor vazio da coluna *qru_corrida*.

Os valores vazios das colunas *km_total* e *valor_corrida* serão preenchidos por zero. Uma outra possibilidade seria preencher com os valores médios de cada coluna. Essa opção não foi utilizada, pois observou-se que a maioria dos casos com valores vazios era de corridas canceladas. Dessa forma, o valor mais adequado seria realmente zero.

Em algumas corridas canceladas, o valor da corrida não é zero, mas acredita-se que esses casos se devem à cobrança da bandeirada, pois o valor cobrado é sempre o mesmo.

```
df['km_total'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['km_total'].fillna(0.0, inplace=True)

df['valor_corrida'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['valor_corrida'].fillna(0.0, inplace=True)
```

Figura 6 – Preenchendo os valores vazios das colunas *km_total* e *valor_corrida*.

Devido à grande quantidade de valores vazios que existiam nas colunas *km_total* e *valor_corrida*, observa-se, ao analisar novamente as estatísticas descritivas dessas colunas, que alguns valores abaixaram como, por exemplo, a média (*mean*) e o desvio padrão (*std*).

```
df['km_total'].describe()
```

```
count    2837.000000
mean        7.871759
std        21.625846
min         0.000000
25%         1.770000
50%         4.090000
75%         9.990000
max      1000.000000
Name: km_total, dtype: float64
```

```
df['valor_corrida'].describe()
```

```
count    2837.000000
mean       24.038449
std        33.054698
min         0.000000
25%         6.320000
50%        12.720000
75%        30.640000
max       474.040000
Name: valor_corrida, dtype: float64
```

Figura 7 – Estatística descritiva das colunas *km_total* e *valor_corrida*.

As colunas *origem_latitude* e *origem_longitude* não possuem valores vazios. As colunas *destino_solicitado_latitude* e *destino_solicitado_longitude* possuem apenas cinco e sete valores vazios, respectivamente, que também serão preenchidos por zero para mostrar a falta de registro.

As colunas *destino_efetivo_latitude* e *destino_efetivo_longitude* possuem valores preenchidos praticamente na mesma quantidade da coluna *km_total*. Observou-se que os valores vazios ocorrem principalmente nos casos de corridas canceladas, assim como ocorre na coluna *km_total*, ou seja, quando a corrida de táxi não ocorre, não são preenchidos os dados de quilometragem e as coordenadas do destino efetivo. Os valores vazios também serão preenchidos com zero.

```
df['destino_solicitado_latitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_solicitado_latitude'].fillna(0.0, inplace=True)

df['destino_solicitado_longitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_solicitado_longitude'].fillna(0.0, inplace=True)

df['destino_efetivo_latitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_efetivo_latitude'].fillna(0.0, inplace=True)

df['destino_efetivo_longitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_efetivo_longitude'].fillna(0.0, inplace=True)
```

Figura 8 – Preenchendo os valores vazios das demais colunas.

Em seguida, observou-se que os dados da coluna *motivo_corrida* não estavam padronizados, o que foi corrigido, como pode ser visto na Figura abaixo.

```
df.motivo_corrida.unique()
```

```
array(['3 - VISITA TECNICA', '1 - REUNIAO EXTERNA', '4 - OUTROS',
       '2 - ENTREGA DE DOCUMENTOS', '02 - Entrega de Documentos',
       '01 - Reunião Externa (Ida/Volta)', '10 - Outros',
       '06 - Inspeção/Fiscalização', '07 - Evento',
       '08 - Atendimento Técnico', '03 - Capacitação/Treinamento',
       '05 - Perícia Médica', '06 Fiscalização',
       '01 Reunião Externa (Ida/Volta)', '04 Outros',
       '02 Entrega de Documentos'], dtype=object)
```

```
df.loc[df['motivo_corrida'] == '01 - Reunião Externa (Ida/Volta)', 'motivo_corrida'] = '1 - REUNIAO EXTERNA'
df.loc[df['motivo_corrida'] == '01 Reunião Externa (Ida/Volta)', 'motivo_corrida'] = '1 - REUNIAO EXTERNA'
df.loc[df['motivo_corrida'] == '02 - Entrega de Documentos', 'motivo_corrida'] = '2 - ENTREGA DE DOCUMENTOS'
df.loc[df['motivo_corrida'] == '02 Entrega de Documentos', 'motivo_corrida'] = '2 - ENTREGA DE DOCUMENTOS'
df.loc[df['motivo_corrida'] == '04 Outros', 'motivo_corrida'] = '4 - OUTROS'
df.loc[df['motivo_corrida'] == '05 - Perícia Médica', 'motivo_corrida'] = '5 - PERICIA MEDICA'
df.loc[df['motivo_corrida'] == '06 - Inspeção/Fiscalização', 'motivo_corrida'] = '6 - INSPECAO/FISCALIZACAO'
df.loc[df['motivo_corrida'] == '06 Fiscalização', 'motivo_corrida'] = '6 - INSPECAO/FISCALIZACAO'
df.loc[df['motivo_corrida'] == '07 - Evento', 'motivo_corrida'] = '7 - EVENTO'
df.loc[df['motivo_corrida'] == '08 - Atendimento Técnico', 'motivo_corrida'] = '8 - ATENDIMENTO TECNICO'
df.loc[df['motivo_corrida'] == '03 - Capacitação/Treinamento', 'motivo_corrida'] = '3 - CAPACITACAO/TREINAMENTO'
df.loc[df['motivo_corrida'] == '10 - Outros', 'motivo_corrida'] = '10 - OUTROS'
```

```
df.motivo_corrida.unique()
```

```
array(['3 - VISITA TECNICA', '1 - REUNIAO EXTERNA', '4 - OUTROS',
       '2 - ENTREGA DE DOCUMENTOS', '10 - OUTROS',
       '6 - INSPECAO/FISCALIZACAO', '7 - EVENTO',
       '8 - ATENDIMENTO TECNICO', '3 - CAPACITACAO/TREINAMENTO',
       '5 - PERICIA MEDICA'], dtype=object)
```

Figura 9 – Padronização dos dados da coluna *motivo_corrida*.

4. Análise e Exploração dos Dados

Para a melhor compreensão dos dados disponíveis, foram feitas diversas análises que serão mostradas a seguir.

Em primeiro lugar, foram identificadas as bases de origem. Como pode ser visto na Figura abaixo, as três existentes são do Distrito Federal, de São Paulo e do Rio de Janeiro, que são os três lugares onde o TáxiGov já foi implantado.

```
df.base_origem.unique()
```

```
array(['TAXIGOV_DF', 'TAXIGOV_RJ_10', 'TAXIGOV_SP_10'], dtype=object)
```

Figura 10 – Identificação das bases de origem.

Em seguida, foram analisados os motivos das corridas de táxi para cada base de origem.

```
motivos_df_series = df[df.base_origem == 'TAXIGOV_DF'].motivo_corrida.value_counts()
motivos_df_series
```

```
1 - REUNIAO EXTERNA          1951
2 - ENTREGA DE DOCUMENTOS     195
4 - OUTROS                    121
3 - VISITA TECNICA             61
Name: motivo_corrida, dtype: int64
```

```
plt.bar(motivos_df_series.keys(), motivos_df_series.array)
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, '')])
```

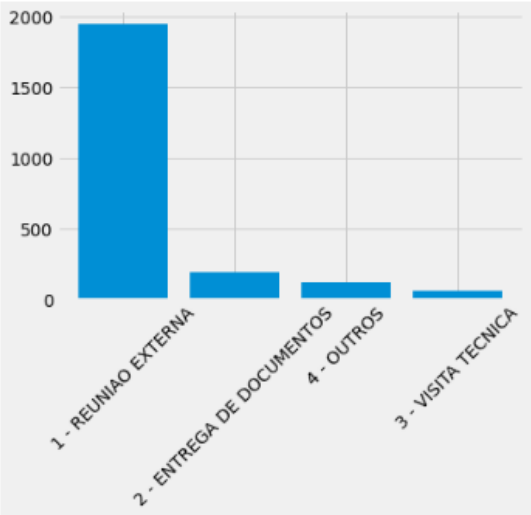


Figura 11 – Análise dos motivos das corridas para o DF.

```
motivos_rj_series = df[df.base_origem == 'TAXIGOV_RJ_10'].motivo_corrida.value_counts()
motivos_rj_series
```

```
10 - OUTROS                    132
1 - REUNIAO EXTERNA            118
2 - ENTREGA DE DOCUMENTOS      100
8 - ATENDIMENTO TECNICO         97
6 - INSPECAO/FISCALIZACAO       23
7 - EVENTO                      12
3 - CAPACITACAO/TREINAMENTO      4
5 - PERICIA MEDICA               1
Name: motivo_corrida, dtype: int64
```

```
plt.bar(motivos_rj_series.keys(), motivos_rj_series.array)
plt.xticks(rotation=90)
```

```
([0, 1, 2, 3, 4, 5, 6, 7],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```

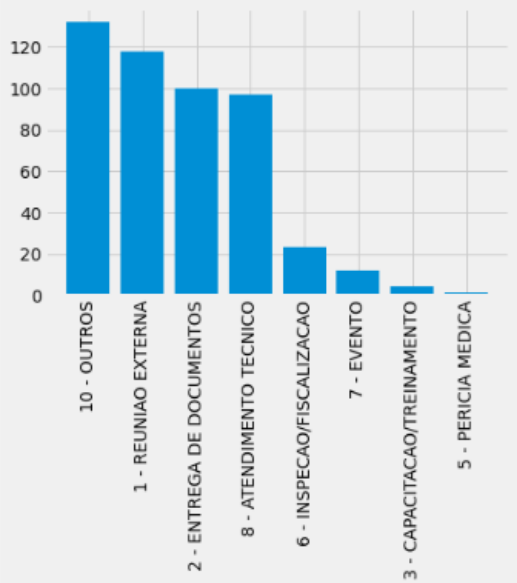


Figura 12 – Análise dos motivos das corridas para o RJ.

```
motivos_sp_series = df[df.base_origem == 'TAXIGOV_SP_10'].motivo_corrida.value_counts()
motivos_sp_series
```

```
6 - INSPECAO/FISCALIZACAO    11
4 - OUTROS                    5
1 - REUNIAO EXTERNA           4
2 - ENTREGA DE DOCUMENTOS     2
Name: motivo_corrida, dtype: int64
```

```
plt.bar(motivos_sp_series.keys(), motivos_sp_series.array)
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, '')])
```

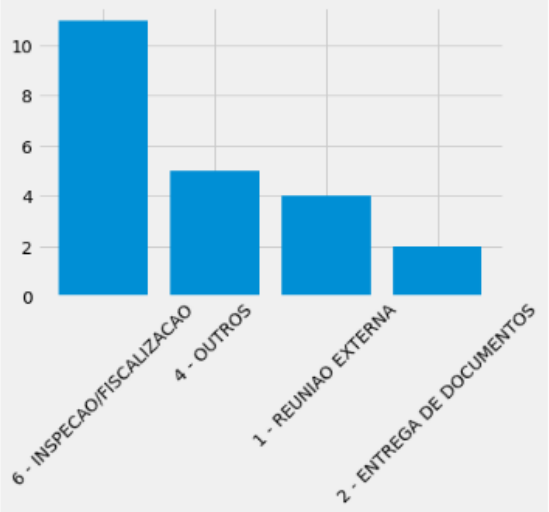


Figura 13 – Análise dos motivos das corridas para SP.

Observa-se que a maioria das corridas de táxi possui como motivo a realização de reuniões externas, principalmente no DF e no RJ. A grande quantidade de corridas por esse motivo poderia ser questionada, avaliando-se a real necessidade desses deslocamentos. É possível que muitas reuniões possam ser substituídas por reuniões remotas. A implantação do teletrabalho no decorrer da pandemia do coronavírus, nos casos em que isso foi possível, mostrou que a realização de reuniões remotas pode proporcionar inúmeros benefícios, como a economia de tempo e de dinheiro.

Outro ponto que chamou a atenção foi a quantidade de corridas cujo motivo selecionado é “outros”. No RJ esse foi o maior motivo. A falta dessa informação prejudica a transparência dos gastos públicos e deveria ser verificada.

Para avaliar se os valores pagos pelas corridas estavam de acordo com os preços praticados no mercado, foi feita uma pesquisa dos valores cobrados por quilômetro pelos táxis em cada município.

Essa pesquisa deu origem à planilha “preco-km-rodado.xlsx” e o *dataframe* *df_preco_km* foi criado a partir da leitura desta planilha.

```
df_preco_km = pd.read_excel('preco-km-rodado.xlsx')
```

```
df_preco_km.head()
```

	Cidade	R$/km (bandeira 1)	R$/km (bandeira 2)	Bandeirada	Tarifa horária	Data da informação
0	AGUAS LINDAS DE GOIAS	2.85	3.66	5.24	31.72	Considerado igual a Brasília, por falta de loc...
1	BRASILIA	2.85	3.66	5.24	31.72	17/03/2016
2	DUQUE DE CAXIAS	2.90	3.48	5.90	36.54	Considerado igual ao Rio de Janeiro, por falta...
3	FUNDAO	2.90	3.48	5.90	36.54	Considerado igual ao Rio de Janeiro, por falta...
4	GUARULHOS	3.42	4.45	5.35	NaN	29/12/2015

Figura 14 – Criação de *dataframe* a partir da leitura da planilha “preco-km-rodado”.

Para unir o *dataframe* principal, *df*, com o *df_preco_km*, foram utilizados os nomes dos municípios, que eram presentes nos dois. Para isso, foi necessário, primeiramente, padronizar os nomes dos municípios do *df*.

```
df.origem_cidade.unique()
```

```
array(['BRASILIA', 'BRASÍLIA', 'NOVO GAMA', 'LUZIANIA',
       'VALPARAISO DE GOIAS', 'AGUAS LINDAS DE GOIAS', 'RIO DE JANEIRO',
       'DUQUE DE CAXIAS', 'RJ', 'Rio de Janeiro', nan, 'SEROPEDICA',
       'NITEROI', 'PETROPOLIS', 'JAPERI', 'SAO JOAO DE MERITI',
       'Duque de Caxias', 'ITAGUAI', 'TRES RIOS', 'FUNDAO', 'Niterói',
       'São Caetano do Sul', 'ITAPECERICA DA SERRA', 'São Paulo',
       'SAO PAULO', 'GUARULHOS'], dtype=object)
```

```
df.loc[df['origem_cidade'] == 'BRASÍLIA', 'origem_cidade'] = 'BRASILIA'
df.loc[df['origem_cidade'] == 'RJ', 'origem_cidade'] = 'RIO DE JANEIRO'
df.loc[df['origem_cidade'] == 'Rio de Janeiro', 'origem_cidade'] = 'RIO DE JANEIRO'
df.loc[df['origem_cidade'] == 'Duque de Caxias', 'origem_cidade'] = 'DUQUE DE CAXIAS'
df.loc[df['origem_cidade'] == 'Niterói', 'origem_cidade'] = 'NITEROI'
df.loc[df['origem_cidade'] == 'São Caetano do Sul', 'origem_cidade'] = 'SAO CAETANO DO SUL'
df.loc[df['origem_cidade'] == 'São Paulo', 'origem_cidade'] = 'SAO PAULO'
```

Figura 15 – Padronização dos nomes dos municípios.

Em seguida, os dois *dataframes* foram unidos, dando origem ao *df_merged*. Para a união, foram utilizadas as colunas *origem_cidade* de *df* e *Cidade* de *df_preco_km*.

```
df_merged = pd.merge(df, df_preco_km, how='left', left_on='origem_cidade', right_on='Cidade')
```

Figura 16 – Criação do *df_merged*.

Após a junção dos dois *dataframes*, observou-se que existiam dados faltando nas colunas “R$/km (bandeira 1)”, “R$/km (bandeira 2)”, “Bandeirada” e “Tarifa horária”. Isso ocorreu, pois alguns registros de corrida não tinham a informação da cidade de origem preenchida.

	R$/km (bandeira 1)	R$/km (bandeira 2)	Bandeirada	Tarifa horária
count	2817.000000	2817.000000	2817.000000	2814.000000
mean	2.867977	3.641700	5.352066	32.600792
std	0.101806	0.151982	0.247818	2.278813
min	2.600000	3.120000	5.240000	29.150000
25%	2.850000	3.660000	5.240000	31.720000
50%	2.850000	3.660000	5.240000	31.720000
75%	2.850000	3.660000	5.240000	31.720000
max	4.000000	5.200000	6.850000	49.000000

Figura 17 – Estatística descritiva das colunas *"R$/km (bandeira 1)", "R$/km (bandeira 2)", "Bandeirada" e "Tarifa horária"*.

Os campos vazios foram então preenchidos pelo valor da média de cada um deles.

```
df_merged['R$/km (bandeira 1)'].fillna(2.867977, inplace=True)
df_merged['R$/km (bandeira 2)'].fillna(3.641700, inplace=True)
df_merged['Bandeirada'].fillna(5.352066, inplace=True)
df_merged['Tarifa horária'].fillna(32.600792, inplace=True)
```

Figura 18 – Preenchendo os campos vazios das colunas *"R$/km (bandeira 1)", "R$/km (bandeira 2)", "Bandeirada" e "Tarifa horária"* com o valor médio.

Para avaliar o valor por quilômetro pago pelas corridas do TáxiGov, foi criada a coluna *valor_por_km_desc_band*, cujo cálculo pode ser visto na Figura abaixo. O valor da bandeirada foi descontado do valor da corrida, pois essa é uma tarifa mínima fixa paga em todas as corridas, independente da quilometragem percorrida. É importante ressaltar que o valor da bandeirada considerado foi o pesquisado, pois na base de dados do TáxiGov não há essa informação.

Como existem muitos registros de *km_total* igual a zero, o resultado do cálculo ficaria igual a infinito nesses casos. Então, nessas situações, adotou-se o valor igual a 110, pois é um valor acima dos demais, que continuará mostrando ser um valor discrepante.

```
for i in range(2837):
    if (df_merged.loc[i, 'km_total'] == 0):
        df_merged.loc[i, 'valor_por_km_desc_band'] = 110
    else:
        df_merged.loc[i, 'valor_por_km_desc_band'] = (df_merged.loc[i, 'valor_corrida'] - df_merged.loc[i, 'Bandeirada'])
        / df_merged.loc[i, 'km_total']
```

Figura 19 – Cálculo do R$/km descontando a bandeirada.

Na Figura a seguir podem ser comparados o valor calculado por quilômetro, que foi pago pela corrida do TáxiGov, com o valor cobrado usualmente pelos táxis. Na Figura abaixo são mostrados apenas os primeiros cinco registros para ilustração.

Mais à frente serão analisados todos os casos e identificadas as situações discrepantes. Observa-se que os valores pagos nesses registros ficaram próximos ao valor cobrado na bandeira 1, que é aplicada na maioria dos municípios nos dias úteis de 6 da manhã às 21 horas da noite. Como os deslocamentos feitos pelo TáxiGov costumam ocorrer nos horários comerciais, a bandeira aplicada seria realmente a 1.

R$/km (bandeira 1)	R$/km (bandeira 2)	valor_por_km_desc_band
2.85	3.66	2.989223
2.85	3.66	2.722871
2.85	3.66	2.260274
2.85	3.66	2.738363
2.85	3.66	2.691480

Figura 20 – Comparação dos valores por quilômetro rodado.

A seguir, foi adicionada uma nova coluna, cujo objetivo foi avaliar a diferença entre o valor por quilômetro calculado, pago através do TáxiGov, com o valor por quilômetro cobrado pelos táxis em cada cidade na bandeira 1.

```
df_merged['dif_valorKM_band1'] = df_merged['valor_por_km_desc_band'] - df_merged['R$/km (bandeira 1)']
```

Figura 21 – Cálculo da diferença de valores por quilômetro rodado.

Para avaliar a diferença entre os valores por quilômetro pago através do TáxiGov e cobrado usualmente pelos táxis, foi feito um gráfico de dispersão desses dois valores. Como os valores cobrados na bandeira 1 variam entre R$2,60 e R$4,00, e os valores pagos deveriam ficar próximo disso, os pontos que ficarem muito fora da reta que vai de (2,60, 2,60) a (4,00, 4,00) deveriam ser analisados.

```
plt.scatter(df_merged['valor_por_km_desc_band'], df_merged['R$/km (bandeira 1)'])
plt.ylabel('R$/km (bandeira 1)')
plt.xlabel('valor_por_km_desc_band')
plt.show()
```

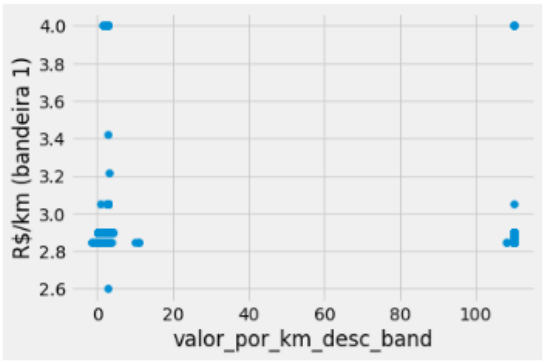


Figura 22 – Gráfico do R$/km calculado e o R$/km cobrado pelos táxis.

Observa-se que existem valores muito discrepantes, cujo valor pago por quilômetro ficou acima de R$100,00. Esses casos devem ser selecionados para serem investigados. Cabe ressaltar que alguns deles se devem aos registros de *km_total* igual a zero, que influenciam no cálculo, como explicado anteriormente.

Também é possível ver através do gráfico a existência de valores abaixo de zero. Esses casos devem ter ocorrido porque no cálculo do valor pago por quilômetro foi descontada a bandeirada e, em alguns registros, o valor da corrida pode estar abaixo do valor da bandeirada, o que não deveria ocorrer. Também existem os casos de valor da corrida igual a zero, que podem ocorrer nas corridas canceladas.

Para melhorar a visualização dos pontos do gráfico, foi plotado um novo gráfico, retirando-se os pontos mais discrepantes, que já serão identificados como anomalias.

```
filter_vKM = df_merged[(df_merged.valor_por_km_desc_band < 20)]
plt.scatter(filter_vKM['valor_por_km_desc_band'], filter_vKM['R$/km (bandeira 1)'])
plt.ylabel('R$/km (bandeira 1)')
plt.xlabel('valor_por_km_desc_band')
plt.show()
```

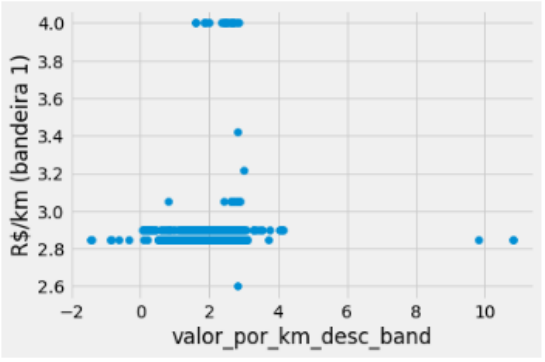


Figura 23 – Gráfico do R$/km calculado e o R$/km cobrado pelos táxis, desconsiderando os pontos mais discrepantes para melhor visualização dos demais.

Outra análise feita foi a comparação da distância percorrida registrada no TáxiGov na coluna *km_total* e a distância calculada utilizando-se as coordenadas. Para o cálculo, foram consideradas as colunas *origem_latitude*, *origem_longitude*, *destino_efetivo_latitude* e *destino_efetivo_longitude*. É importante ressaltar que a distância calculada através das coordenadas é sempre uma reta e, por isso, tende a ser menor do que a distância realmente percorrida. Mesmo assim é possível utilizá-la para fins de comparação.

Foi criada a coluna *dist_coords* e o cálculo da distância foi feito utilizando-se a função *haversine*, obtida através do endereço eletrônico abaixo.

https://stackoverflow.com/questions/40452759/pandas-latitude-longitude-to-distance-between-successive-rows

```
# vectorized haversine function (obtido em:
# https://stackoverflow.com/questions/40452759/pandas-latitude-longitude-to-distance-between-successive-rows)
def haversine(lat1, lon1, lat2, lon2, to_radians=True, earth_radius=6371):
    """
    slightly modified version: of http://stackoverflow.com/a/29546836/2901002

    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees or in radians)

    All (lat, lon) coordinates must have numeric dtypes and be of equal length.

    """
    if to_radians:
        lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    a = np.sin((lat2-lat1)/2.0)**2 + \
        np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return earth_radius * 2 * np.arcsin(np.sqrt(a))
```

```
df_merged['dist_coords'] = haversine(df_merged.origem_latitude, df_merged.origem_longitude,
                                     df_merged.destino_efetivo_latitude, df_merged.destino_efetivo_longitude)
```

Figura 24 – Cálculo da distância percorrida através das coordenadas de origem e destino.

Na Figura abaixo é possível ver a comparação entre o *km_total* e a distância calculada através das coordenadas de origem e destino, *dist_coords*, para os cinco primeiros registros. Como explicado anteriormente, a distância calculada tende a ser menor que a distância efetivamente percorrida (*km_total*).

km_total	dist_coords
30.62	23.516740
11.98	9.345818
5.84	3.871426
12.46	8.873182
11.15	7.864339

Figura 25 – Comparação entre *km_total* e *dist_coords*.

Também foi criada a coluna (*dif_distancias*) para registrar a diferença entre a distância efetivamente percorrida, *km_total*, e a distância calculada pelas coordenadas, *dist_coords*.

```
df_merged['dif_distancias'] = df_merged['km_total'] - df_merged['dist_coords']
```

Figura 26 – Criação da coluna *dif_distancias*.

Em seguida, foi feito um gráfico de dispersão com *dist_coords* por *km_total*.

```
plt.scatter(df_merged['dist_coords'], df_merged['km_total'])
plt.ylabel('km_total')
plt.xlabel('dist_coords')
plt.show()
```

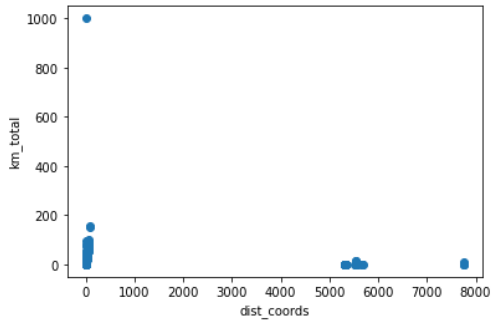


Figura 27 – Gráfico *dist_coord* por *km_total*.

São observados alguns casos em que a quilometragem registrada está próxima de zero, mas a distância calculada através das coordenadas é alta, o que indica uma possível anomalia. Esses casos devem ser analisados posteriormente junto com outros parâmetros, como status, valor e duração da corrida. Nos demais casos, o gráfico se aproxima de uma reta, seguindo um padrão e mostrando que os valores estão próximos.

Eliminando-se os pontos mais discrepantes para melhor visualização dos demais, obtém-se o gráfico a seguir. Ele também mostra a existência de um ponto que chama a atenção, cuja distância calculada através das coordenadas é muito inferior à quilometragem total registrada.

```
filter_dist = df_merged[(df_merged.dist_coords < 1000)]
plt.scatter(filter_dist['dist_coords'], filter_dist['km_total'])
plt.ylabel('km_total')
plt.xlabel('dist_coords')
plt.show()
```

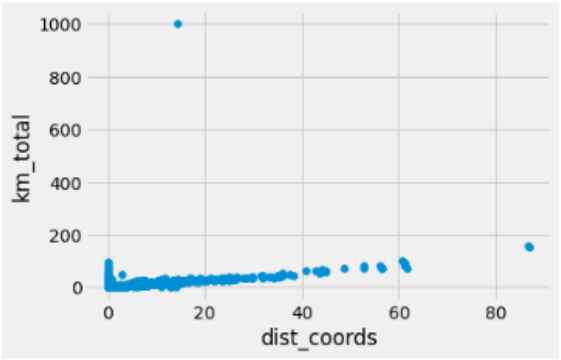


Figura 28 – Gráfico *dist_coord* por *km_total* com a eliminação dos pontos mais discrepantes.

Outro gráfico feito foi das diferenças de distância (*dif_distancias*) e de valor pago por quilômetro na bandeira 1 (*dif_valorKM_band1*). O objetivo desse gráfico é analisar os pontos distantes da coordenada (0,0). Quanto mais próximo de (0,0), as diferenças de distância e de valor por quilômetro calculadas são pequenas. E, ao contrário, quanto mais distante, o caso deve ser investigado.

```
plt.scatter(df_merged['dif_distancias'], df_merged['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')
plt.show()
```

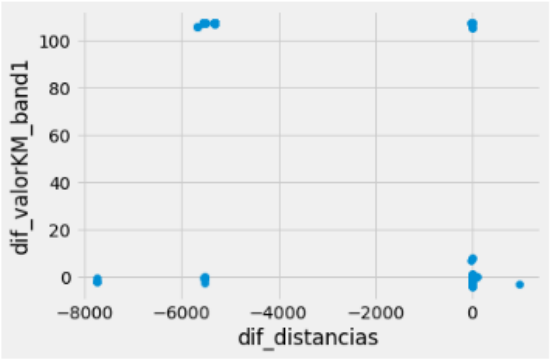


Figura 29 – Gráfico *dif_distancias* por *dif_valorKM_band1*.

No gráfico acima é possível ver que existe uma concentração de pontos próximos de (0,0), como era esperado. No entanto, existem casos com grandes diferenças.

Para visualizar melhor a concentração de pontos próximos de (0,0), os pontos mais distantes foram removidos no gráfico a seguir.

```
filter_difs = df_merged[(df_merged.dif_distancias > -100) & (df_merged.dif_distancias < 200)
                        & (df_merged.dif_valorKM_band1 < 20)]
plt.scatter(filter_difs['dif_distancias'], filter_difs['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')
plt.show()
```

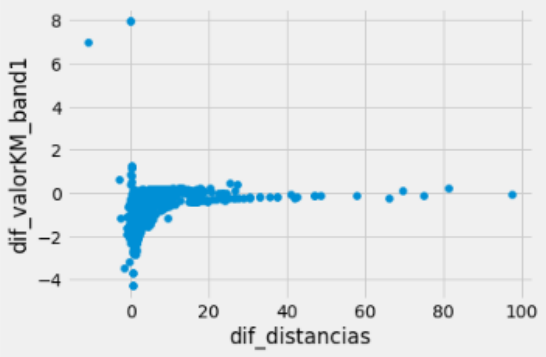


Figura 30 – Gráfico *dif_distancias* por *dif_valorKM_band1* removendo os pontos mais distantes para melhor visualização da concentração em (0,0).

Para avaliar o percentual que a diferença da distância calculada pelas coordenadas representa sobre a quilometragem total registrada, foi criada a coluna *perc_dif_dist*. Nos casos em que a *km_total* é igual a zero e o cálculo ficaria igual a infinito, os valores foram preenchidos com 1000, que é um valor acima dos demais e mostrará ser discrepante.

```
for i in range(2837):
    if (df_merged.loc[i, 'km_total'] == 0):
        df_merged.loc[i, 'perc_dif_dist'] = 1000
    else:
        df_merged.loc[i, 'perc_dif_dist'] = (df_merged.loc[i, 'dif_distancias'] / df_merged.loc[i, 'km_total'])*100
```

Figura 31 – Criação da coluna *perc_dif_dist*.

O gráfico abaixo mostra a relação entre a porcentagem da diferença entre as distâncias e a diferença entre os valores pagos por quilômetro. A seguir, é mostrado o mesmo gráfico, mas com a eliminação dos valores mais discrepantes para facilitar a visualização dos demais.

```
plt.scatter(df_merged['perc_dif_dist'], df_merged['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('perc_dif_dist')
plt.xticks(rotation=90)
plt.show()
```

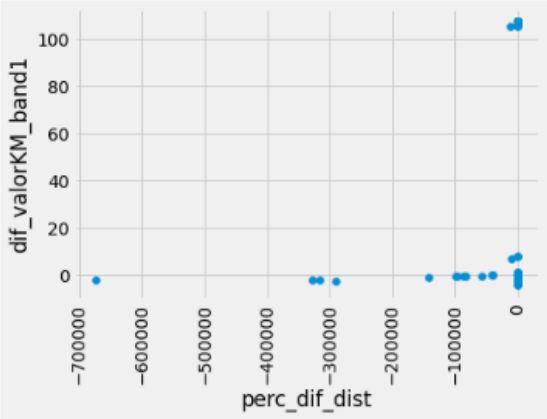


Figura 32 – Gráfico *perc_dif_dist* por *dif_valorKM_band1*

```
filter_difs_perc = df_merged[(df_merged.perc_dif_dist > -100) & (df_merged.dif_valorKM_band1 < 20)]
plt.scatter(filter_difs_perc['perc_dif_dist'], filter_difs_perc['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('perc_dif_dist')
plt.show()
```

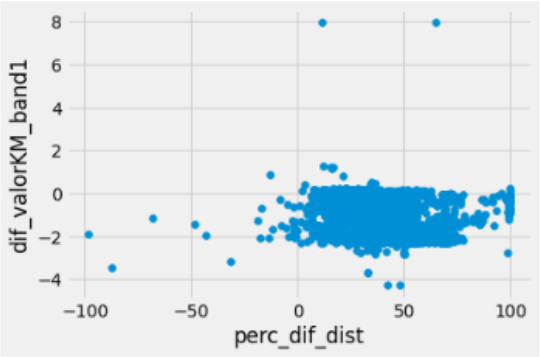


Figura 33 – Gráfico *perc_dif_dist* por *dif_valorKM_band1* removendo os pontos mais distantes para melhor visualização.

Podem ser vistos muitos casos de grande variação percentual, que devem ser analisados. Também são observados muitos valores negativos para o percentual da diferença entre as distâncias. Isso ocorre nos casos em que a quilometragem total registrada é inferior à distância calculada pelas coordenadas.

Em seguida, foi feito o cálculo da duração da corrida. Para isso, foram criadas duas colunas. A *tempo_corrida*, que mostra a duração em dias e horas, e a *minutos_corrida*, que mostra a duração em minutos.

```
df_merged['tempo_corrida'] = pd.to_datetime(df_merged['data_final']) - pd.to_datetime(df_merged['data_inicio'])
t1 = pd.to_datetime(df_merged['data_final'])
t2 = pd.to_datetime(df_merged['data_inicio'])
df_merged['minutos_corrida'] = (df_merged['tempo_corrida']).astype('timedelta64[m]')
```

Figura 34 – Cálculo da duração da corrida.

	tempo_corrida	minutos_corrida
count	2778	2778.000000
mean	0 days 00:14:35.009530597	14.086033
std	0 days 00:17:38.944299355	17.642760
min	-1 days +23:52:03.323000	-8.000000
25%	0 days 00:05:36.192750	5.000000
50%	0 days 00:09:45.105500	9.000000
75%	0 days 00:17:39.512250	17.000000
max	0 days 06:26:44.804000	386.000000

Figura 35 – Estatística descritiva das colunas *tempo_corrida* e *minutos_corrida*.

Foram observados campos vazios para *tempo_corrida* e *minutos_corrida* que ocorreram por falta de informação. Eles serão preenchidos por zero para mostrar que o dado está faltando.

```
df_merged['tempo_corrida'].fillna(0.0, inplace=True)
df_merged['minutos_corrida'].fillna(0.0, inplace=True)
```

Figura 36 – Preenchendo os campos vazio das colunas *tempo_corrida* e *minutos_corrida*.

A estatística descritiva da coluna *minutos_corrida* mostra que a duração média das corridas é 14 minutos. Pelo valor mínimo, é possível perceber que existem casos de duração negativa. Provavelmente, isso se deve ao registro da *data_final* anterior à *data_inicio*, o que indica um erro e deve ser verificado.

Para avaliar se as corridas de maior quilometragem, de maior duração e de maior valor possuem R$/km menor, foram traçados os gráficos abaixo.

```
plt.scatter(df_merged['km_total'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('km total')
plt.show()
```

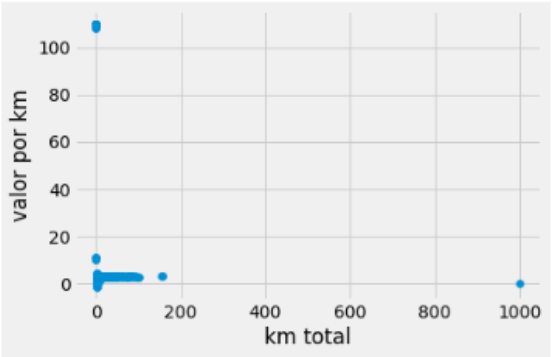


Figura 37 – Gráfico *km total* por *valor por km*.

```
plt.scatter(df_merged['valor_corrida'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('valor corrida')
plt.show()
```

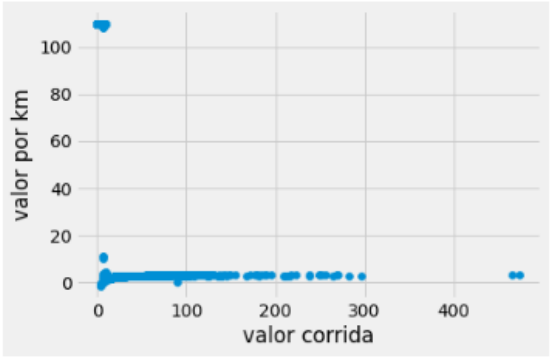


Figura 38 – Gráfico *valor corrida* por *valor por km*.

```
plt.scatter(df_merged['minutos_corrida'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('minutos corrida')
plt.show()
```

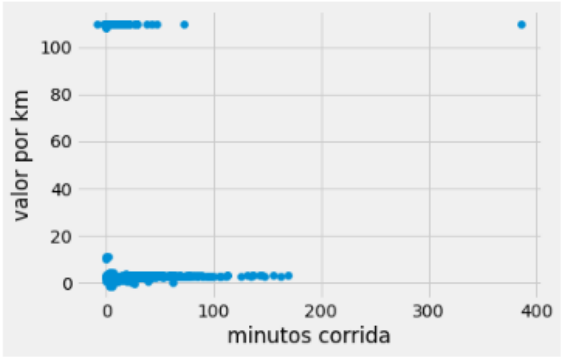


Figura 39 – Gráfico *minutos corrida* por *valor por km*.

De modo geral, percebe-se que o valor cobrado por quilômetro pode diminuir com o aumento da duração da corrida e da sua quilometragem e distância. Mas alguns valores continuam chamando a atenção pela grande discrepância e não podem ser justificados por esse motivo.

Foi feito outro gráfico em três dimensões para avaliar a relação entre os valores de *km_total*, *valor_corrida* e *minutos_corrida*. O resultado pode ser visto na Figura abaixo.

```
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = plt.axes(projection='3d')
xline = df_merged['km_total']
yline = df_merged['valor_corrida']
zline = df_merged['minutos_corrida']
ax.scatter3D(xline, yline, zline)
ax.set_xlabel('km_total')
ax.set_ylabel('valor_corrida')
ax.set_zlabel('minutos_corrida')
```

```
Text(0.5, 0, 'minutos_corrida')
```

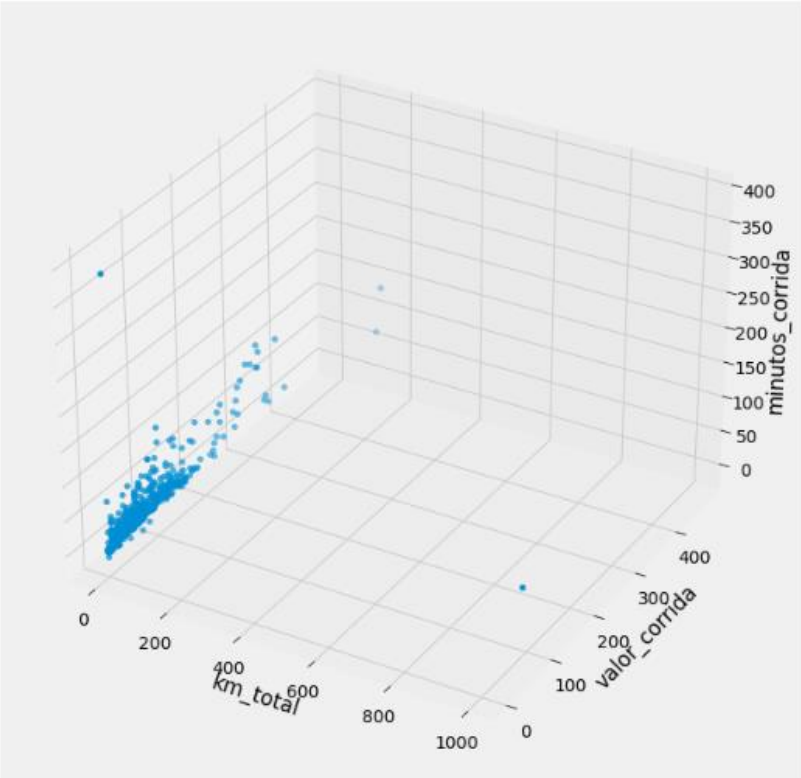


Figura 40 – Gráfico *km_total x valor_corrida x minutos_corrida*.

Pelo gráfico acima, é possível perceber a existência de corridas com baixa duração e valor alto, o que chama atenção. Para melhor visualização, foi feito um gráfico com esses dois eixos.

```
plt.scatter(df_merged['minutos_corrida'], df_merged['valor_corrida'])
plt.ylabel('valor corrida')
plt.xlabel('minutos corrida')
plt.show()
```

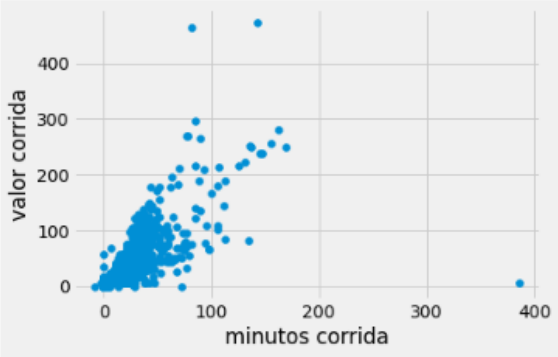


Figura 41 – Gráfico *minutos_corrida x valor_corrida*.

Também foram feitos gráficos de duração pela quilometragem da corrida, e de valor pela quilometragem, conforme Figuras a seguir.

```
plt.scatter(df_merged['minutos_corrida'], df_merged['km_total'])
plt.ylabel('km total')
plt.xlabel('minutos corrida')
plt.show()
```

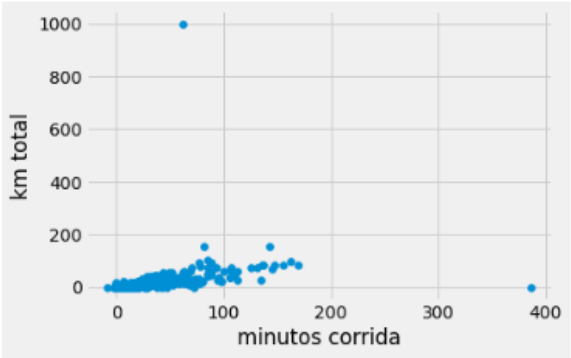


Figura 42 – Gráfico *minutos_corrida x km_total*.

Pelo gráfico acima, é possível observar também corridas com grande duração e baixa quilometragem, assim como corridas de duração relativamente mais baixas, com alta quilometragem. Todas essas situações devem ser investigadas.

```
plt.scatter(df_merged['km_total'], df_merged['valor_corrida'])
plt.ylabel('valor corrida')
plt.xlabel('km total')
plt.show()
```

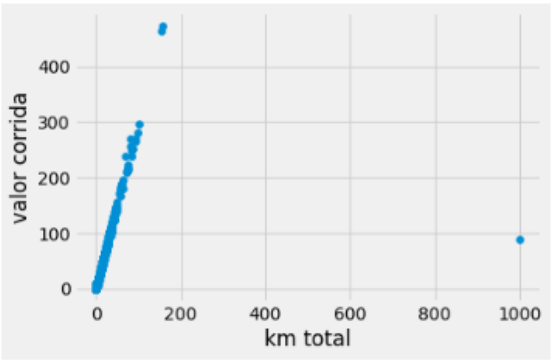


Figura 43 – Gráfico *km_total x valor_corrida*.

O resultado mais esperado do gráfico acima seria quanto maior a quilometragem, maior o valor da corrida. Esse comportamento ocorre na maioria dos casos, formando uma reta. No entanto, é possível visualizar a existência de pontos fora dessa reta, ou seja, que não seguem este padrão e, portanto, devem ser analisados.

Buscou-se analisar os dados disponíveis de inúmeros ângulos, de forma que fosse possível identificar relações entre os diversos parâmetros. Em todas as situações foi possível perceber a existência de pontos discrepantes, que fugiam do padrão. Na próxima seção, serão buscadas formas de se identificar esses pontos discrepantes utilizando-se modelos de *Machine Learning (ML)*.

5. Criação de Modelos de Machine Learning

O aprendizado em *Machine Learning* pode ser dividido em dois grandes tipos: supervisionado e não-supervisionado. No aprendizado supervisionado, existe a informação dos rótulos históricos, ou seja, fala-se que os dados são rotulados, pois as saídas desejadas a serem estimadas são conhecidas. Já no aprendizado não-supervisionado não se tem essa informação. Os dados são, portanto, não-rotulados. Nesses casos, o algoritmo não recebe os resultados esperados durante o treinamento, devendo descobrir os possíveis relacionamentos entre eles através da exploração dos dados. Busca-se padrões entre os dados com o objetivo de agrupá-los em razão de suas similaridades.

No caso do problema em questão, em que se busca identificar registros das corridas do TáxiGov fora dos padrões, não há a informação histórica de quais valores são considerados adequados ou não. Dessa forma, será utilizado o aprendizado não-supervisionado.

O objetivo será agrupar os registros do conjunto de dados em grupos, ou *clusters*, de forma que elementos em um *cluster* compartilhem um conjunto de propriedades comuns que os diferencie dos elementos dos demais *clusters*. Esse é, portanto, um problema de *clusterização*.

Os algoritmos de *clusterização* que serão utilizados são o *K-means*, o *DBSCAN* e o *HDBSCAN*.

Os algoritmos serão aplicados nos dados obtidos pelas colunas *dif_distancias* e *dif_valorKM_band1*. Essa escolha foi feita, pois o gráfico gerado por essas duas colunas (Figuras 29 e 30) permite boa visualização dos pontos discrepantes e são duas medidas que indicam claramente as possíveis anomalias.

Como os pontos de maior discrepância já são claramente *outliers*, para melhor visualização dos dados e para impactar menos no resultado dos algoritmos de *machine learning*, eles serão inicialmente filtrados.

Além disso, as dimensões das colunas *dif_distancias* e *dif_valorKM_band1* são muito diferentes. Logo, os dados deverão ser colocados em escalas mais próximas. Para isso, será utilizado o *RobustScaler*. Diferente de outros métodos, as estatísticas de centralização e dimensionamento do *RobustScaler* são baseadas em percentis e, portanto, não são influenciadas por um pequeno número de valores discre-

pantes marginais muito grandes. Os *outliers* continuam presentes nos dados transformados, que é o objetivo no caso do problema analisado.

```
from sklearn.preprocessing import RobustScaler
```

```
filter_difs_rs = df_merged[(df_merged.dif_distancias > -100) & (df_merged.dif_distancias < 200)
                           & (df_merged.dif_valorKM_band1 < 20)]
```

```
array_difs_rs = filter_difs_rs[['dif_distancias', 'dif_valorKM_band1']].to_numpy()
```

```
transformer = RobustScaler().fit(array_difs_rs)
```

```
transform = transformer.transform(array_difs_rs)
```

```
plt.scatter(transform[:,0], transform[:,1])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')
```

```
Text(0.5, 0, 'dif_distancias')
```

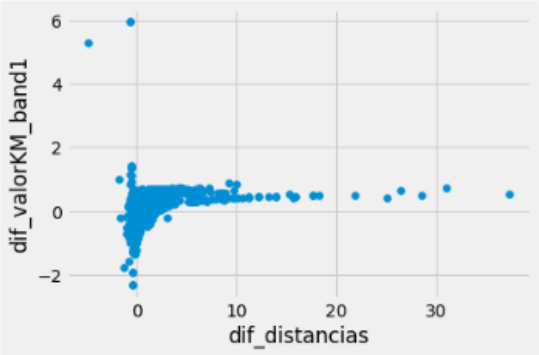


Figura 44 – Aplicação do método *RobustScaler*.

Observa-se que a escala foi alterada. Anteriormente, como pode ser visto na Figura 30, o gráfico variava entre -4 e 8 no eixo *y*, e -20 a 100 no eixo *x*.

5.1 Algoritmo *K-means*

5.1.1 Funcionamento do algoritmo *K-means*

O *K-means* é um algoritmo de clusterização disponível na biblioteca *Scikit-Learn*.

Ele funciona da seguinte forma: primeiramente, deve ser definido um *K*, ou seja, um número de *clusters* (ou agrupamentos); depois, é definido, aleatoriamente, um centroide para cada cluster; o próximo passo é o cálculo, para cada ponto, do centroide mais próximo; em seguida, o centroide é reposicionado, sendo que a nova posição do centroide deve ser a média da posição de todos os pontos do *cluster*; os dois últimos passos são repetidos, iterativamente, até a obtenção da posição ideal dos centroides.

5.1.2 O Método do Cotovelo

A definição do número de *clusters* deve ser feita previamente, pelo usuário do algoritmo. Um dos métodos que auxiliam nessa definição é o Método do Cotovelo (do inglês *Elbow Method*).

Esse Método executa o algoritmo *K-means* no conjunto de dados para um intervalo de valores para *K* (de 1 a 10, por exemplo) e, em seguida, para cada valor de *K*, calcula uma pontuação média para todos os *clusters*. Por padrão, a pontuação de distorção é calculada, a soma das distâncias quadradas de cada ponto ao seu centro atribuído.

Quando as métricas são plotadas em um gráfico, é possível determinar visualmente o melhor valor para *K*. Considerando que o gráfico de linhas se parece com um braço, então o “cotovelo”, ou seja, o ponto de inflexão na curva, é o melhor valor de *K*.

5.1.3 Aplicação do algoritmo *K-means* no conjunto de dados

```
Sum_of_squared_distances = []
K = range(1, 10)
for num_clusters in K:
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(transform)
    Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

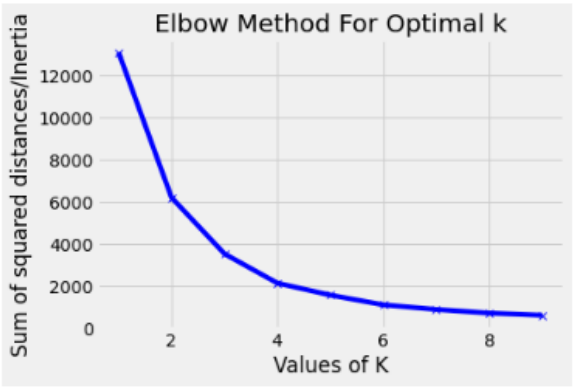


Figura 45 – Definição do número de *clusters* pelo Método do Cotovelo.

Como não fica claro pela Figura acima se o número de *clusters* ideal é quatro ou cinco, foi feito o teste com os dois valores, conforme pode ser visto nas Figuras a seguir.

```
model_k_4 = KMeans(4)
model_k_4.fit(transform)
y_model_k_4 = model_k_difs_4.predict(transform)

plt.scatter(transform[:, 0], transform[:, 1], c=y_model_k_4, s=50, cmap='viridis')

centers = model_k_4.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')
```

```
Text(0, 0.5, 'dif_valorKM_band1')
```

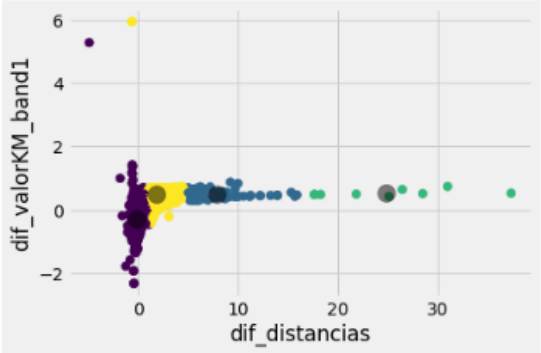


Figura 46 – Aplicação do algoritmo *K-means* com 4 *clusters*.

```
model_k_5 = KMeans(5)
model_k_5.fit(transform)
y_model_k_5 = model_k_5.predict(transform)

plt.scatter(transform[:, 0], transform[:, 1], c=y_model_k_5, s=50, cmap='viridis')

centers = model_k_5.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')
```

```
Text(0, 0.5, 'dif_valorKM_band1')
```

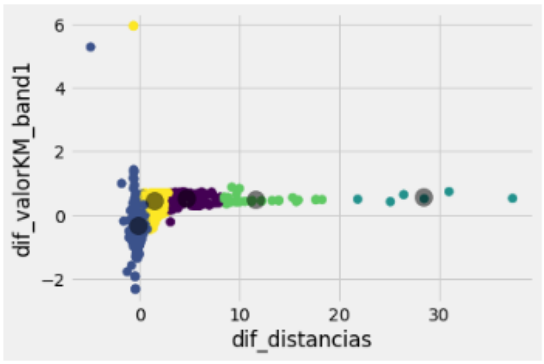


Figura 47 – Aplicação do algoritmo *K-means* com 5 *clusters*.

Com cinco *clusters*, há uma separação maior dos dados, mas em nenhum dos dois casos o resultado se mostrou adequado, pois pontos muito discrepantes fazem parte de *clusters* com valores que não são considerados como anomalias. Percebe-se que a divisão dos pontos entre os *clusters* foi feita tendo-se como base apenas o eixo x.

A seguir, serão aplicados outros algoritmos para verificar se eles apresentam melhor comportamento nesse caso.

5.2 Algoritmo *DBSCAN*

5.2.1 Funcionamento do algoritmo *DBSCAN*

DBSCAN - Density-Based Spatial Clustering of Applications with Noise – também é um algoritmo de clusterização da biblioteca *Scikit-Learn*. Ele visualiza os *clusters* como áreas de alta densidade separadas por áreas de baixa densidade. Devido a essa visão bastante genérica, os *clusters* encontrados pelo *DBSCAN* podem possuir qualquer formato, ao contrário do *K-means*, que assume que os *clusters* possuem formato convexo.

Um dos principais parâmetros para o algoritmo é o *eps*. Ele especifica quão próximos os pontos devem estar uns dos outros para serem considerados parte de um *cluster*. Isso significa que se a distância entre dois pontos for menor ou igual ao valor do *eps*, esses pontos serão considerados vizinhos.

Se o valor de *eps* escolhido for muito pequeno, uma grande parte dos dados não será agrupada. Serão considerados *outliers* por não satisfazerem o número de pontos para criar uma região densa. Por outro lado, se o valor escolhido for muito alto, os *clusters* serão mesclados e a maioria dos objetivos estará no mesmo *cluster*. O *eps* deve ser escolhido com base na distância do conjunto de dados, mas, em geral, valores menores são preferíveis. O valor padrão é 0,5.

Algumas vantagens do *DBSCAN* são que ele não exige que se especifique o número de *clusters* nos dados *a priori*, ao contrário do *K-means*, e ele é robusto a *outliers*.

5.2.2 Determinação do valor ótimo para o *eps*

Para o cálculo do valor do *eps*, pode ser calculada a distância para os *n* pontos mais próximos para cada ponto, classificando e plotando os resultados. Em seguida, verifica-se o ponto onde a mudança de curvatura é mais pronunciada, sendo este o valor selecionado para o *eps*.

A distância de cada ponto ao seu vizinho pode ser calculada usando o *NearestNeighbors*. O próprio ponto está incluído em *n_neighbors*. O método *kneighbors*

retorna dois *arrays*, um que contém a distância até os pontos *n_neighbors* mais próximos e o outro que contém o índice para cada um desses pontos.

Em seguida, os resultados são classificados e plotados. O valor ótimo para *épsilon* será encontrado no ponto de curvatura máxima.

5.2.3 Aplicação do algoritmo *DBSCAN* no conjunto de dados

```
from sklearn.cluster import DBSCAN
```

```
from sklearn.neighbors import NearestNeighbors
```

```
neigh = NearestNeighbors(n_neighbors = 2)
nbrs = neigh.fit(transform)
distances, indices = nbrs.kneighbors(transform)
```

```
distances = np.sort(distances, axis = 0)
distances = distances[:,1]
plt.plot(distances)
plt.ylim(top=0.6)
```

```
(-0.31897752690222864, 0.6)
```

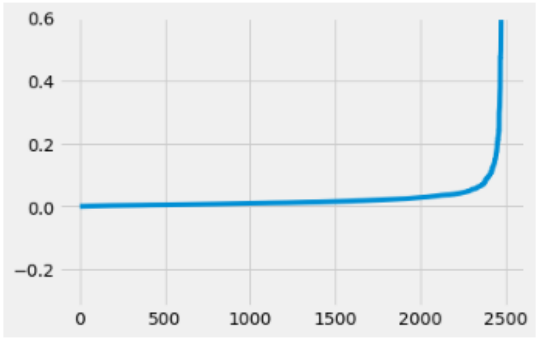


Figura 48 – Definição do valor ótimo para o *eps*.

A partir da análise do ponto de maior curvatura no gráfico da Figura acima, o valor definido para o *epsilon* foi 0,2. Dessa forma, o algoritmo *DBSCAN* foi aplicado considerando-se esse valor.

```
model_db = DBSCAN(eps=0.2)
model_db.fit(transform)
plt.scatter(transform[:,0], transform[:,1], c=model_db.labels_)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')
```

```
Text(0, 0.5, 'dif_valorKM_band1')
```

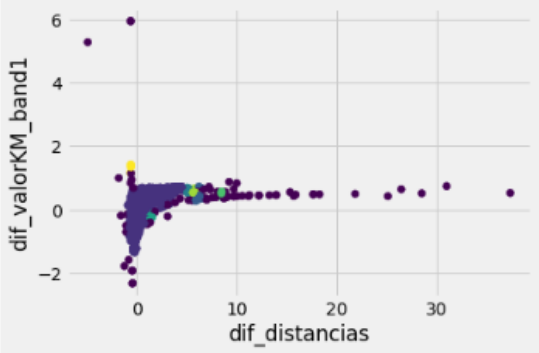


Figura 49 – Aplicação do algoritmo *DBSCAN* com *eps* igual a 0,2.

O resultado obtido com o algoritmo *DBSCAN* foi mais satisfatório do que o obtido com o teste anterior, com o *K-means*. Observa-se que os pontos mais próximos de (0, 0) foram agrupados em um mesmo *cluster* e os mais afastados foram classificados como *outliers* (pontos de cor roxa mais escura). No entanto, ainda existem alguns pontos cuja classificação não está tão clara.

5.3 Algoritmo *HDBSCAN*

5.3.1 Funcionamento do algoritmo *HDBSCAN*

O algoritmo *HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise* – executa o algoritmo *DBSCAN* sobre valores *epsilon* variados e integra o resultado para encontrar um agrupamento que forneça a melhor estabilidade sobre *epsilon*. Isso permite que o *HDBSCAN* encontre *clusters* de densidades variadas (ao contrário do *DBSCAN*) e seja mais robusto na seleção de parâmetros. Além disso, o algoritmo possui como vantagem a sua capacidade de indicar *outliers*.

Embora existam muitos parâmetros que possam ser definidos, na prática poucos têm um efeito significativo no *clustering*. Aqui será avaliada a influência do *min_cluster_size*. Ele é definido para o menor tamanho de agrupamento que se deseja considerar para um *cluster*. O valor padrão do *min_cluster_size* é 5.

5.3.2 Determinação do valor do parâmetro *min_cluster_size*

Para a determinação do valor mais adequado para o *min_cluster_size*, aplicou-se o algoritmo *HDBSCAN* com diversos valores do parâmetro. Dessa forma, foi possível analisar os tamanhos dos *clusters* e a quantidade de *outliers* obtida.

```
import hdbscan
```

```
clusters_sizes = []
num_outliers = []

for k in range (2,50):
 clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
 clusterer.fit(transform)
 clusters_sizes.append(len(set(clusterer.labels_)))
 num_outliers.append(sum(clusterer.labels_ == -1))
```

Figura 50 – Aplicação do algoritmo *HDBSCAN* com diferentes valores de *min_cluster_size*.

```
plt.style.use("fivethirtyeight")
plt.figure(figsize=(12,6))
plt.plot(range(2, 50), clusters_sizes, marker='o')
plt.xticks(range(2, 50))
plt.xlabel("min_cluster_size")
plt.ylabel("Número de Clusters")
plt.xlim(0,20)
plt.show()
```

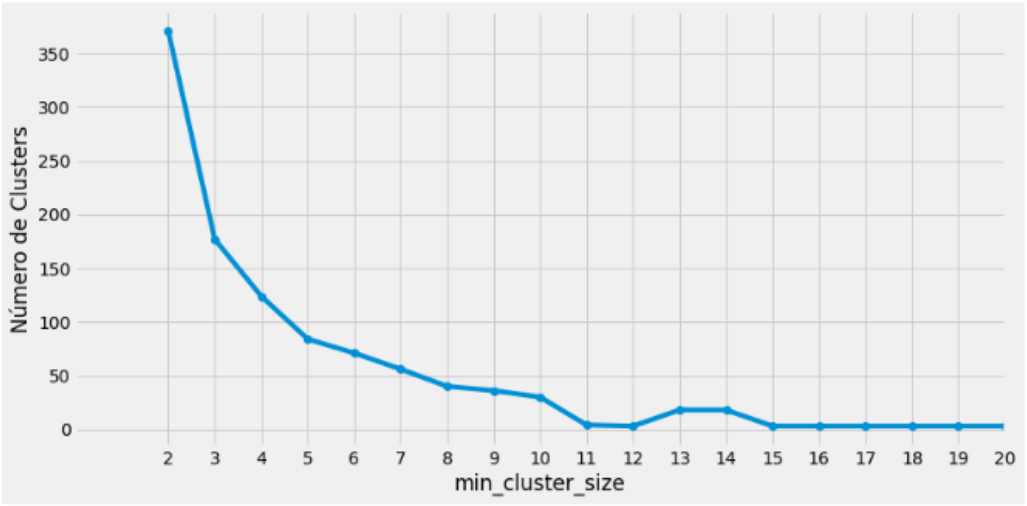


Figura 51 – Gráfico de *min_cluster_size* por número de *clusters*.

```
plt.style.use("fivethirtyeight")
plt.figure(figsize=(12,6))
plt.plot(range(2, 50), num_outliers, marker='o')
plt.xticks(range(2, 50))
plt.xlabel("min_cluster_size")
plt.ylabel("Quantidade de Outliers")
plt.xlim(0,20)
plt.show()
```

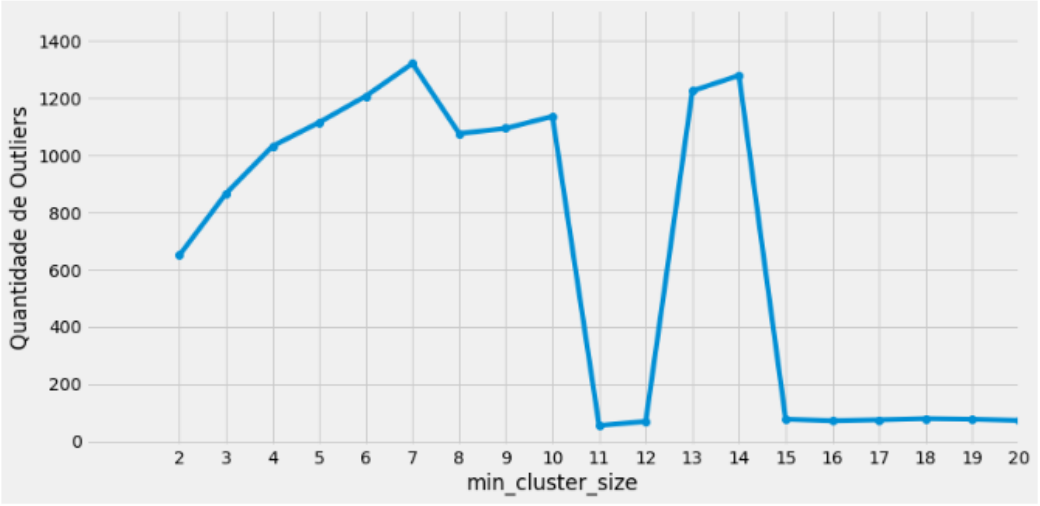


Figura 52 – Gráfico de *min_cluster_size* por quantidade de *outliers*.

A partir da análise dos gráficos acima, optou-se por aplicar o algoritmo *HDBSCAN* com o *min_cluster_size* igual a 16, que indicava uma quantidade menor de *outliers* e uma quantidade estável de *clusters*.

5.3.3 Aplicação do algoritmo *HDBSCAN* no conjunto de dados

A partir da análise feita para a determinação do *min_cluster_size*, o valor definido para o parâmetro foi 16. Dessa forma, o algoritmo *HDBSCAN* foi aplicado considerando-se esse valor.

```
model_hdb = hdbscan.HDBSCAN(min_cluster_size=16)
model_hdb.fit(transform)
plt.scatter(transform[:,0], transform[:,1], c=model_hdb.labels_)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')
```

```
Text(0, 0.5, 'dif_valorKM_band1')
```

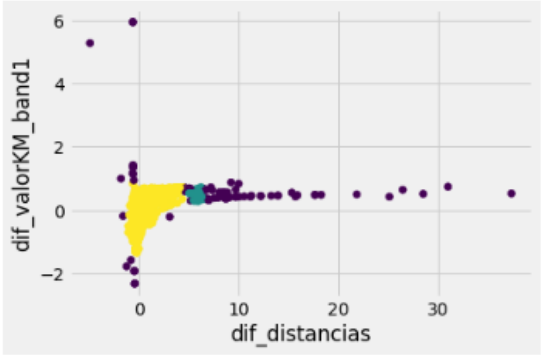


Figura 53 – Aplicação do algoritmo *HDBSCAN* com *min_cluster_size=16*.

O resultado obtido foi o mais satisfatório dos três algoritmos. Formou-se um *cluster* bem definido próximo à coordenada (0,0) do gráfico, que são os pontos cujas diferenças de distância e de valor por quilômetro são menores. À medida que se afasta desse ponto, aparecem os *outliers*, que são as anomalias buscadas no presente trabalho.

6. Interpretação dos Resultados

Através das análises realizadas nas seções anteriores, concluiu-se que o algoritmo que apresentou o resultado mais satisfatório foi o *HDBSCAN*.

Para melhor interpretação dos resultados, foi criada uma coluna com os *labels* obtidos a partir da aplicação do algoritmo. Primeiramente, todos os valores foram preenchidos com zero. Em seguida, os valores identificados como sendo muito discrepantes nas análises iniciais, e que por isso já haviam sido considerados como anomalias, foram preenchidos com -1.

```
df_merged['HDBSCAN_labels'] = 0.0
```

Figura 54 – Criação da coluna *HDBSCAN_labels* e preenchimento dos valores com zero.

```
df_merged['HDBSCAN_labels'] = np.where((df_merged.dif_distancias < -100) | (df_merged.dif_distancias > 200)
                        | (df_merged.dif_valorKM_band1 > 20), -(1.0), df_merged['HDBSCAN_labels'])
```

Figura 55 – Preenchendo com -1 os campos cujos valores já haviam sido identificados como anomalias.

A seguir, os demais campos foram preenchidos com os *labels* obtidos com a aplicação do algoritmo *HDBSCAN* e foi criado o *dataframe df_outliers* com os registros cujos *labels* são iguais a -1, ou seja, com as anomalias buscadas.

```
label_idx = 0
for i in range(2837):
    if (df_merged.loc[i, 'dif_distancias'] > -100) & (df_merged.loc[i, 'dif_distancias'] < 200) &
    (df_merged.loc[i, 'dif_valorKM_band1'] < 20):
        df_merged.loc[i, 'HDBSCAN_labels'] = model_hdb.labels_[label_idx]
        label_idx = label_idx + 1
```

Figura 56 – Preenchimento dos campos com os *labels* obtidos a partir da aplicação do algoritmo *HDBSCAN*.

```
df_outliers = df_merged[(df_merged.HDBSCAN_labels == -(1.0))]
```

Figura 57 – Criação do *dataframe df_outliers*.

```
df_merged.HDBSCAN_labels.value_counts()
```

```
 1.0    2383
-1.0     426
 0.0      28
Name: HDBSCAN_labels, dtype: int64
```

Figura 58 – Contagem dos *labels*.

Pelos resultados mostrados nas Figuras acima, observa-se que foram identificados 426 *outliers*, ou seja, 15% do total de registros.

Alguns gráficos feitos durante a análise inicial dos dados foram refeitos, mas agora plotando-se também os *labels*.

```
plt.scatter(filter_difs_rs['dist_coords'], filter_difs_rs['km_total'], c=model_hdb.labels_)
plt.xlabel('dist_coords')
plt.ylabel('km_total')
```

```
Text(0, 0.5, 'km_total')
```

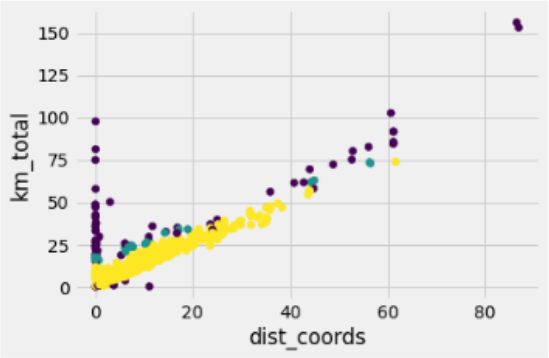


Figura 59 – Gráfico de *dist_coords x km_total*.

O primeiro deles foi o que mostra as distâncias calculadas através das coordenadas de origem e de destino *versus* a quilometragem total registrada. O ideal é que os valores de x e de y sejam o mais próximo possível, formando uma reta. Os pontos muito fora dessa reta mostram possíveis anomalias. Como pode ser visto na Figura acima, o resultado foi satisfatório, pois esses pontos foram realmente identificados como *outliers* pela aplicação do algoritmo.

```
plt.scatter(df_merged['minutos_corrida'], df_merged['valor_corrida'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('minutos_corrida')
plt.ylabel('valor_corrida')
```

```
Text(0, 0.5, 'valor_corrida')
```

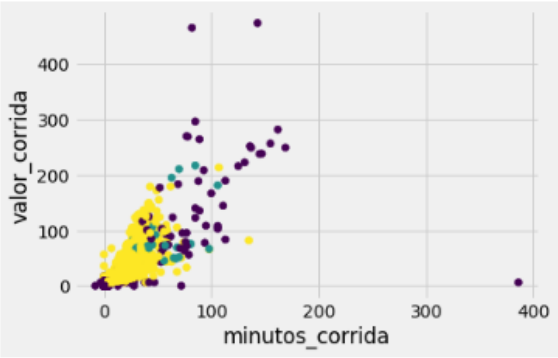


Figura 60 – Gráfico de *minutos_corrida x valor_corrida*.

```
plt.scatter(df_merged['minutos_corrida'], df_merged['km_total'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('minutos_corrida')
plt.ylabel('km_total')
```

```
Text(0, 0.5, 'km_total')
```

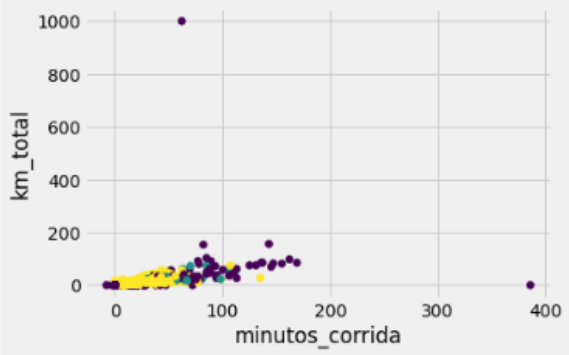


Figura 61 – Gráfico de *minutos_corrida x km_total*.

```
plt.scatter(df_merged['km_total'], df_merged['valor_corrida'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('km_total')
plt.ylabel('valor_corrida')
```

```
Text(0, 0.5, 'valor_corrida')
```

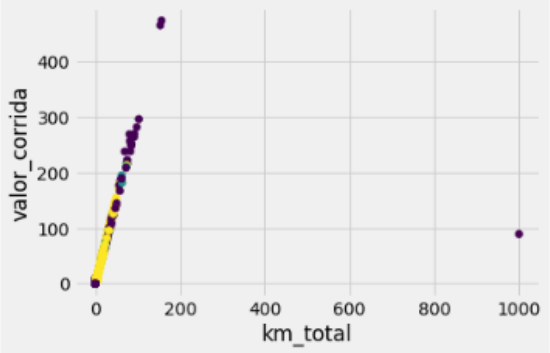


Figura 62 – Gráfico de *km_total x valor_corrida*.

Os gráficos das Figuras 60, 61 e 62, que mostram a relação entre a duração, o valor e a quilometragem total da corrida corroboram que o resultado do algoritmo identificou corretamente os pontos mais discrepantes. Mas eles também mostram pontos que a princípio seriam tratados como pontos conformes pela simples análise dos gráficos e que foram identificados como *outliers*. Isso se deve ao fato de que o modelo não levou em consideração todas as possíveis dimensões, apenas as com maior influência sobre as anomalias. Além disso, mostra que pontos dentro da reta, que à primeira vista não seriam vistos como *outliers*, podem sim apresentar alguma inconformidade e devem ser verificados.

Para enriquecer a análise, os dados do *dataframe df_merged* também foram exportados para uma planilha em Excel e analisados utilizando-se o software Power BI.

```
import openpyxl
```

```
df_merged.to_excel('df_merged_final.xlsx', index=False)
```

Figura 63 – Exportação do *datafram df_merged* para Excel.

Na Figura abaixo, pode-se ver que quase a totalidade das corridas canceladas foi identificada como *outlier* (barra laranja, representando o *label* igual a “-1”). Isso era o esperado, pois nesses casos os dados registrados apresentam diversas inconsistências, com muitos campos sem preenchimento.

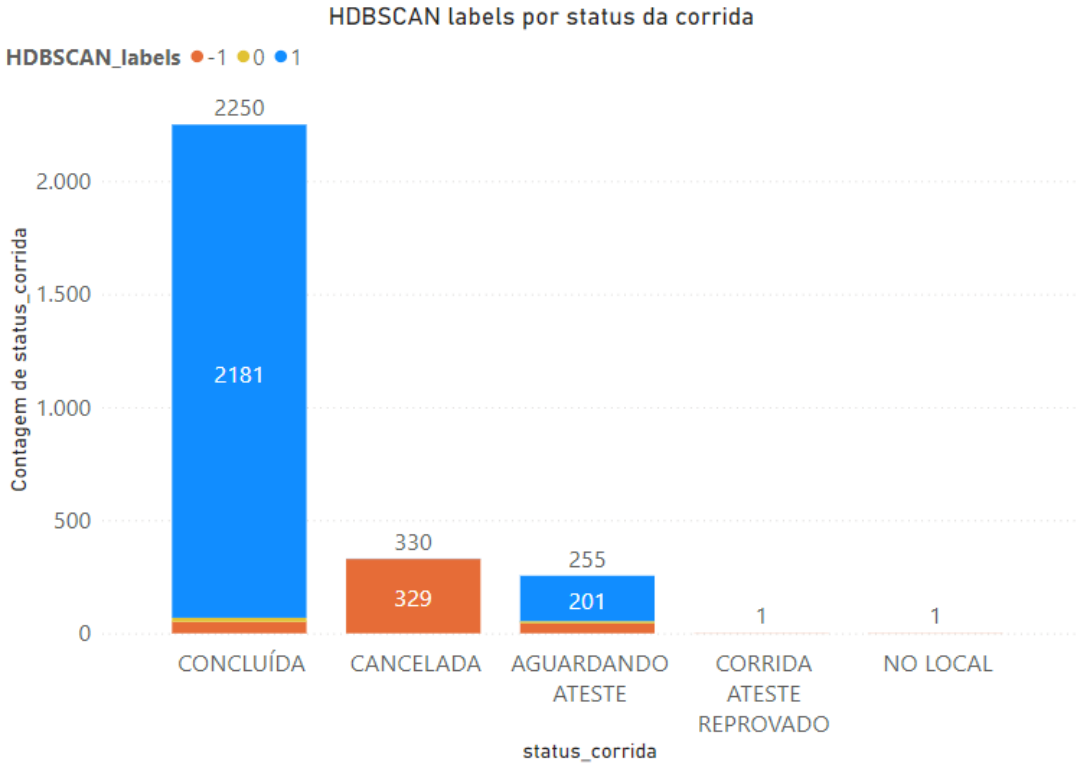


Figura 64 – Verificação do status da corrida dos *outliers*.

A corrida com status “ateste reprovado” também foi considerada *outlier*, o que está de acordo com a análise já feita anteriormente pelo governo, que reprovou o registro desta corrida.

Cerca de 21% das corridas com status “aguardando ateste” foram identificadas como *outliers*, ou seja, é possível que seus registros sejam reprovados e o status mude para “ateste reprovado”.

Em relação às bases de origem, observa-se maior quantidade de *outliers* à medida em que aumenta a quantidade de registros, o que é esperado, conforme pode ser visto na Figura abaixo.

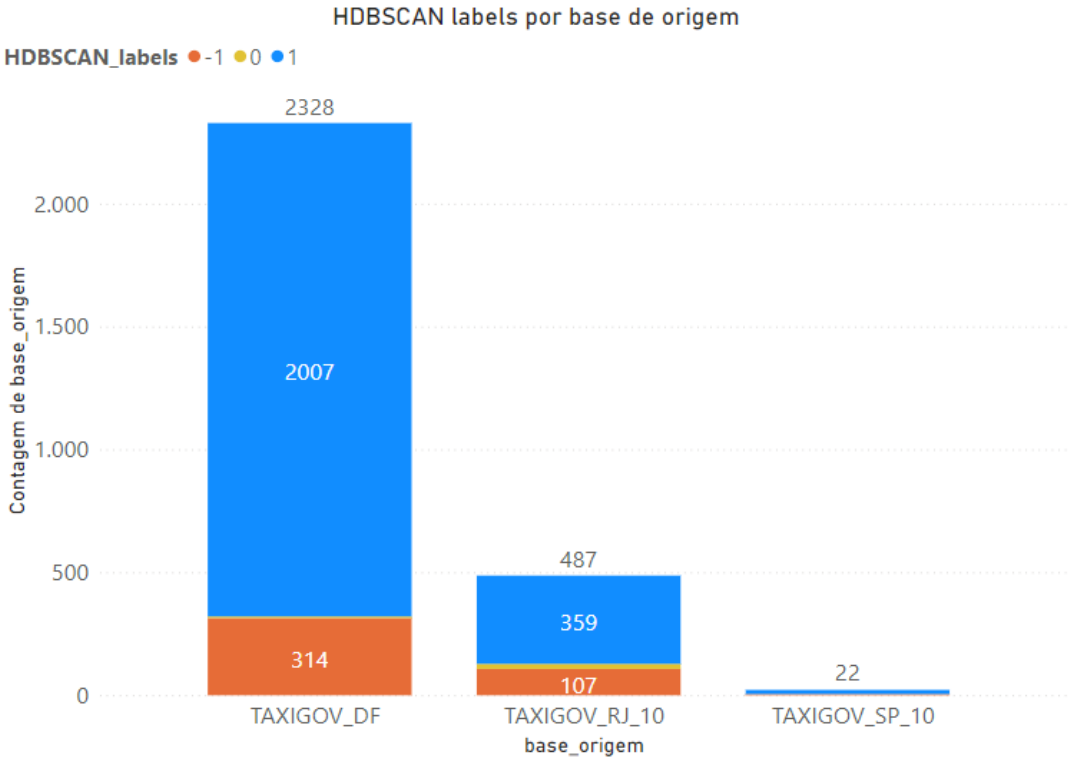


Figura 65 – Verificação dos *outliers* por base de origem.

A seguir, foi analisada a quantidade de *outliers* por órgão de cada base de origem. Para isso, foram desconsideradas as corridas canceladas.

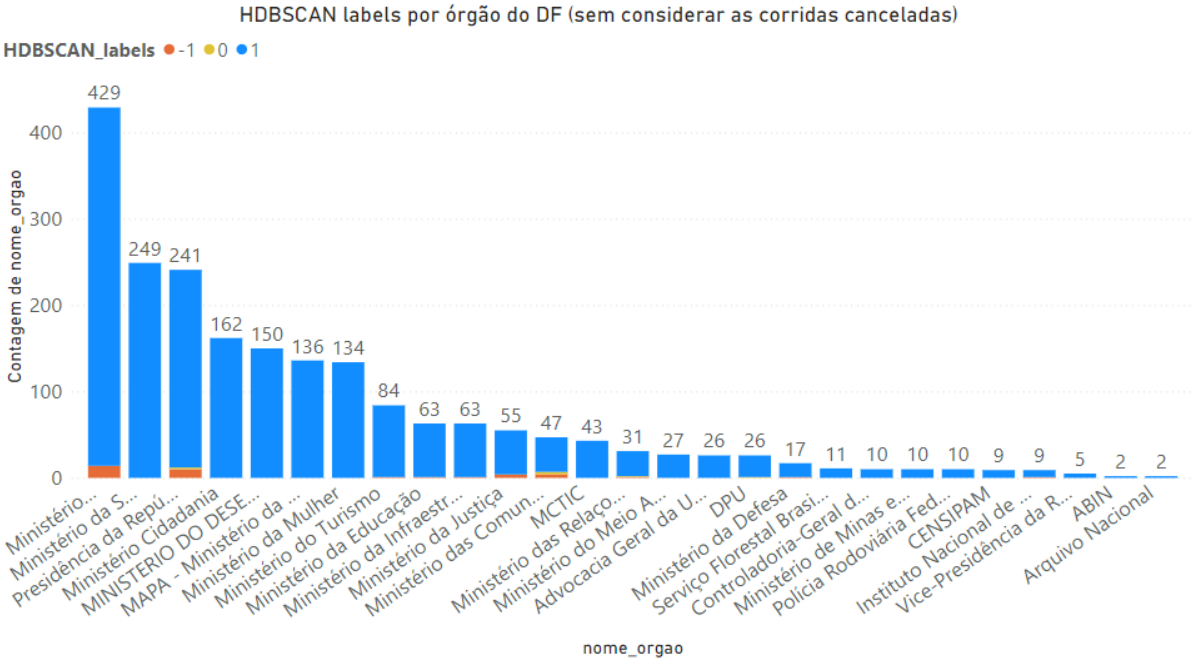


Figura 66 – Verificação dos *outliers* por órgão do DF.

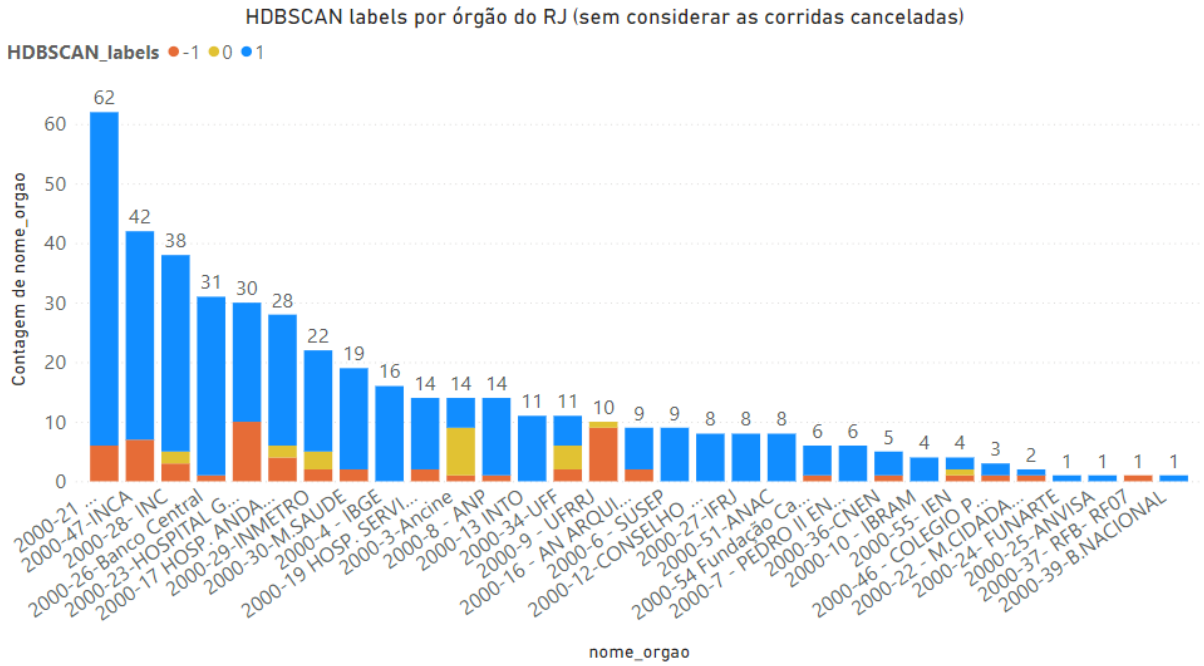


Figura 67 – Verificação dos *outliers* por órgão do RJ.

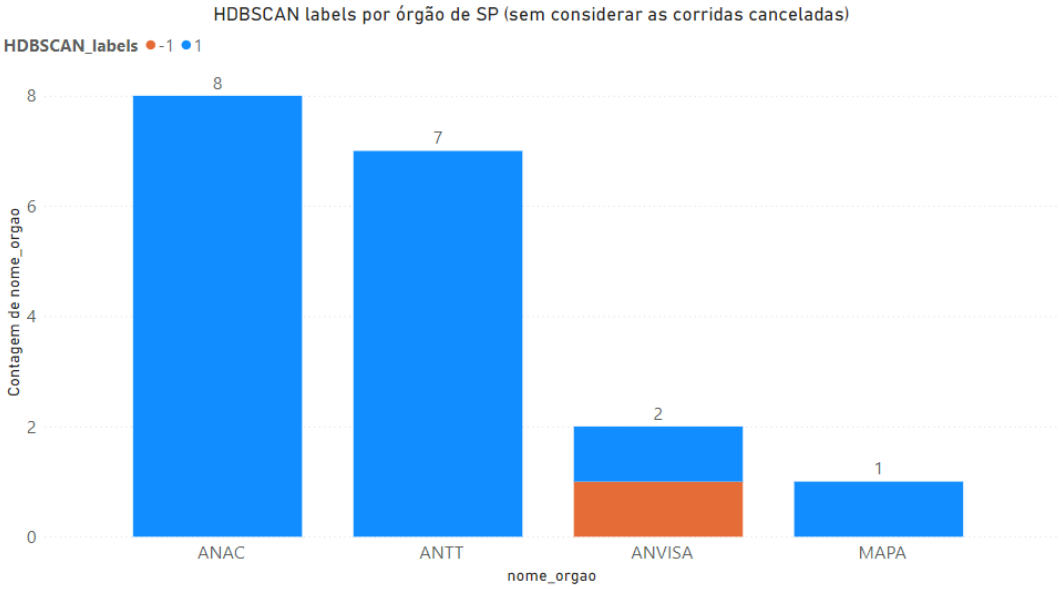


Figura 68 – Verificação dos *outliers* por órgão de SP.

Observa-se que os gráficos do DF e de SP não apresentam valores que chamam a atenção. No entanto, no RJ, existem órgãos cujos registros das corridas de táxi apresentam grande quantidade de *outliers*, como é o caso da UFRRJ e do Hospital Gaffree e Guinle. A maioria desses casos ocorre pela elevada diferença entre a quilometragem calculada pelas coordenadas de origem e destino (*dist_coords*) e a quilometragem total registrada (*km_total*).

Também foi analisada a quantidade de *outliers* de acordo com o município de origem da corrida de táxi. No estado do RJ, principalmente, existem algumas cidades com grande concentração de *outliers*, como Petrópolis e São João de Meriti. A identificação dessas cidades pode ajudar a identificar e a sanar as possíveis causas dessas anomalias.

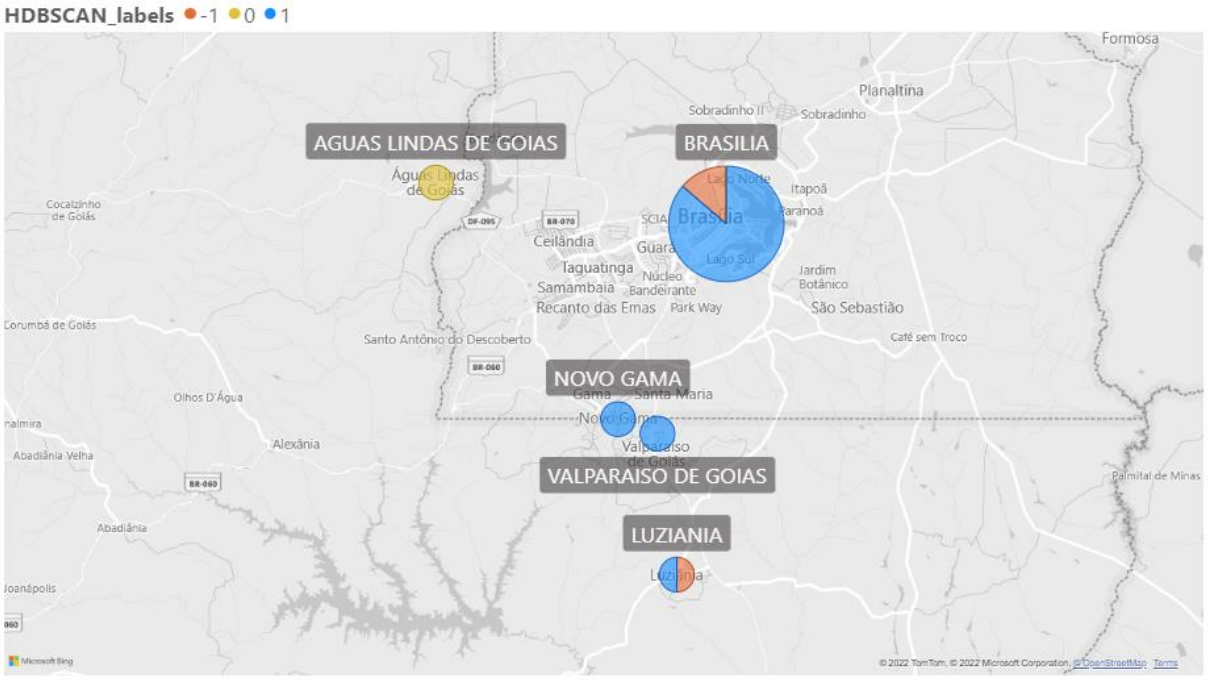


Figura 69 – Verificação dos *outliers* por cidade de origem no DF.

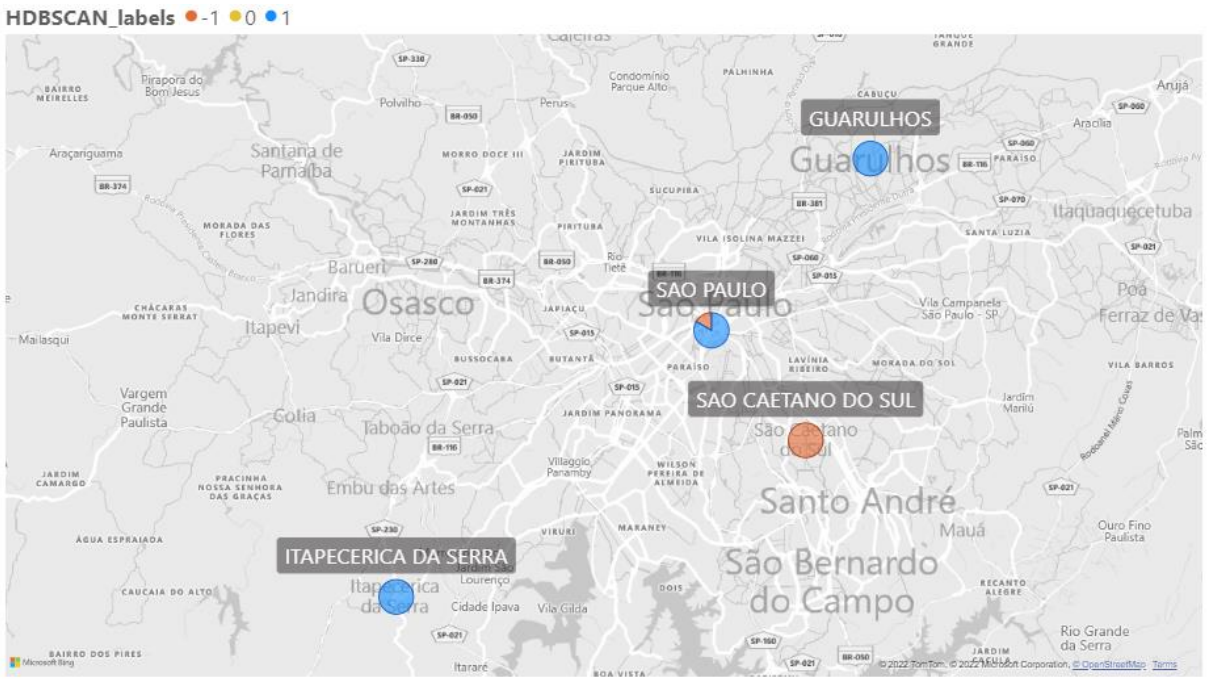


Figura 70 – Verificação dos *outliers* por cidade de origem em SP.

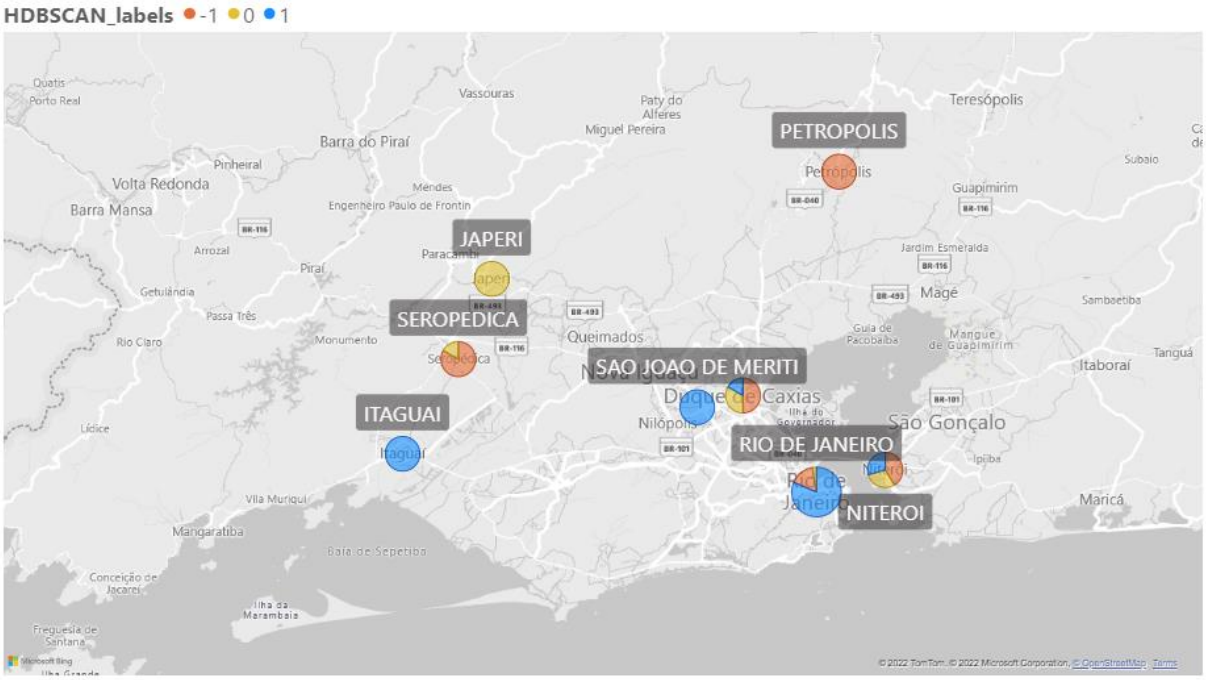


Figura 71 – Verificação dos *outliers* por cidade de origem no RJ.

Ao analisar a totalidade dos dados, observou-se que os principais grupos de *outliers* são formados por:

- corridas canceladas;
- grandes diferenças de valor pago por quilômetro ao se comparar o valor calculado através dos dados registrados e o valor cobrado usualmente pelos táxis na bandeira 1;
- grandes diferenças de distâncias quando são comparadas a quilometragem total registrada e a quilometragem calculada através das coordenadas de origem e de destino;
- coordenadas do destino efetivo em branco e coordenadas de origem preenchidas sem vírgula, fazendo com que a diferença de distâncias ficasse muito grande. Isso mostra erros de cadastro que devem ser corrigidos pelos servidores;
- quilometragem total muito pequena, o que deixa o valor por quilômetro bastante elevado, já que a bandeirada é cobrada de toda forma, independentemente da quilometragem rodada.

7. Apresentação dos Resultados

O governo federal implantou, a partir de março de 2017, o TáxiGov, que é um serviço utilizado pelos servidores e colaboradores em deslocamentos a trabalho por meio de táxis. Até o momento, ele já está em operação no Distrito Federal e em alguns municípios de São Paulo e do Rio de Janeiro.

De acordo com o Ministério da Economia, desde a implantação do TáxiGov foram realizadas 918 mil viagens, foram 6,2 milhões de quilômetros rodados e resultou na economia de 33,8 milhões de reais aos cofres públicos.

Em 2021, foi registrado um gasto acima de R$ 3 milhões com os deslocamentos feitos pelo TáxiGov nos locais onde o serviço já está em operação. Como a expectativa é a expansão do serviço para todas as capitais do Brasil até o final de 2022, os valores gastos com corridas de táxis registrados no TáxiGov serão cada vez maiores. Logo, a análise e o controle desses gastos públicos são de extrema relevância.

Para isso, foram analisados os registros das corridas feitas entre os dias 31 de março e 06 de abril de 2022, disponíveis no Portal de Dados Abertos do Governo Brasileiro.

A análise dos dados buscou identificar registros fora dos padrões, em que os valores das corridas, as distâncias percorridas e as durações dos percursos não estavam em conformidade com o que era esperado para cada caso.

Para complementar a análise, os dados obtidos foram comparados com o preço por quilômetro rodado cobrado pelos táxis na bandeira 1 dos municípios abrangidos.

Durante a exploração inicial dos dados, foram calculadas algumas novas medidas para auxiliar na identificação dos registros fora dos padrões, entre elas:

- *dist_coords*: a distância em quilômetros das coordenadas de origem e de destino efetivo;

- *dif_distancias*: a diferença entre a quilometragem total registrada (*km_total*) e a distância calculada através das coordenadas (*dist_coords*);

- *valor_por_km_desc_band*: o valor por quilômetro pago pelas corridas registradas. O cálculo foi feito da seguinte forma: (*valor_corrida – bandeirada*) / *km_total*;

- *dif_valorKM_band1*: a diferença entre o valor pago por quilômetro calculado através dos registros do TáxiGov (*valor_por_km_desc_band*) e o valor cobrado usualmente pelos táxis nos municípios em análise obtido em pesquisas realizadas.

O gráfico de *dif_distancias* x *dif_valorKM_band1* deveria mostrar uma concentração de pontos próxima ao eixo (0,0), significando que as diferenças calculadas de quilometragem e de valor cobrado por quilômetro foram pequenas ou próximas de zero. O gráfico obtido pode ser visto na Figura abaixo.

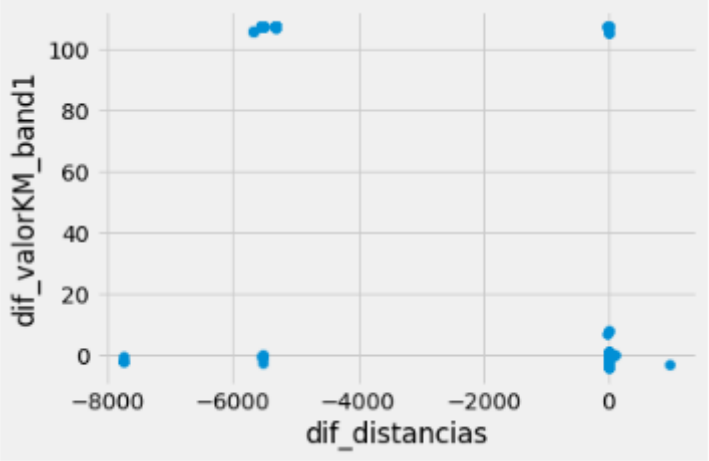


Figura 72 - Gráfico *dif_distancias* por *dif_valorKM_band1*.

Observa-se que realmente ocorreu uma concentração próxima ao ponto (0,0), mas também são vistos diversos pontos muito distantes, que representam casos que devem ser investigados.

Para visualizar melhor os pontos próximos de (0,0), foi feito novo gráfico eliminando-se os casos mais discrepantes, que já foram considerados como anomalias.

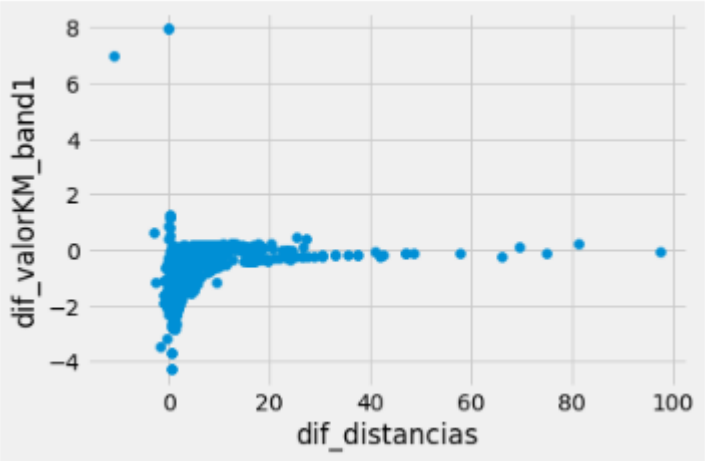


Figura 73 – Gráfico *dif_distancias* por *dif_valorKM_band1* removendo os pontos mais distantes para melhor visualização da concentração em (0,0).

O gráfico da Figura acima mostra realmente que a maioria dos pontos está concentrada próxima de (0,0), mas ainda é possível identificar altas diferenças de quilometragem e de R$/km.

Foram feitas diversas outras análises considerando os demais parâmetros registrados pelo TáxiGov. No entanto, considerou-se que as diferenças de distância (*dif_distancias*) e de valor pago por quilômetro na bandeira 1 (*dif_valorKM_band1*) foram as que mostraram melhor os casos de anomalias.

Como os pontos de maior discrepância já são claramente *outliers*, para melhor visualização dos dados e para impactar menos no resultado dos algoritmos de *machine learning*, esses pontos foram inicialmente retirados.

No problema em questão, em que se busca identificar registros das corridas do TáxiGov fora dos padrões, não há a informação histórica de quais valores são considerados adequados ou não. Dessa forma, foi utilizado o aprendizado não-supervisionado.

Além disso, o objetivo era agrupar os registros do conjunto de dados em grupos, ou *clusters*, de forma que elementos em um *cluster* compartilhassem um conjunto de propriedades comuns que os diferenciasse dos elementos dos demais *clusters*. Sendo assim, foram utilizados os algoritmos de *clusterização K-means*, *DBSCAN* e *HDBSCAN*.

Para cada um deles, foram determinados os parâmetros mais adequados. Para o *K-means*, foi utilizado o Método do Cotovelo para a definição do número de *clusters*. No *DBSCAN*, foi determinado o valor ótimo para o *eps*. Já no *HDBSCAN*, determinou-se o valor mais adequado para o *min_cluster_size*.

Nas Figuras abaixo, foram mostrados os resultados obtidos com cada algoritmo.

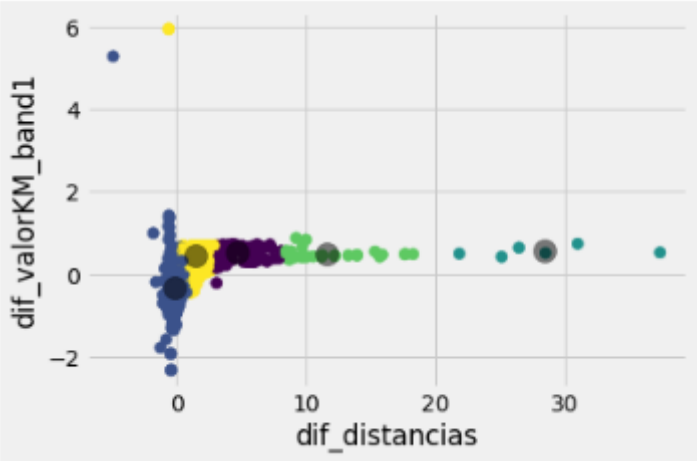


Figura 74 – Aplicação do algoritmo *K-means* com 5 *clusters*.

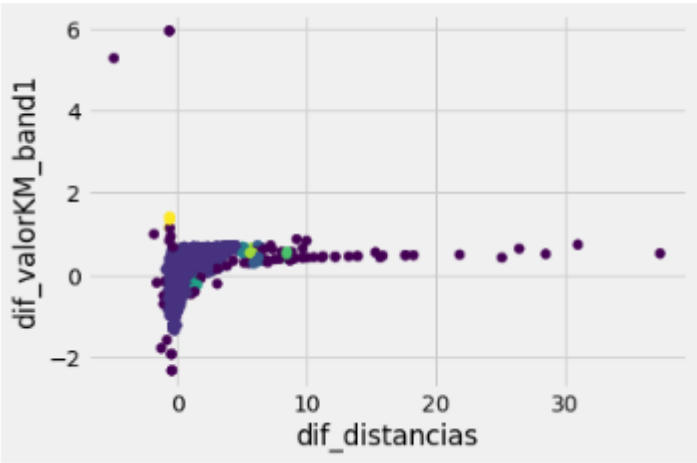


Figura 75 – Aplicação do algoritmo DBSCAN com eps igual a 0,2.

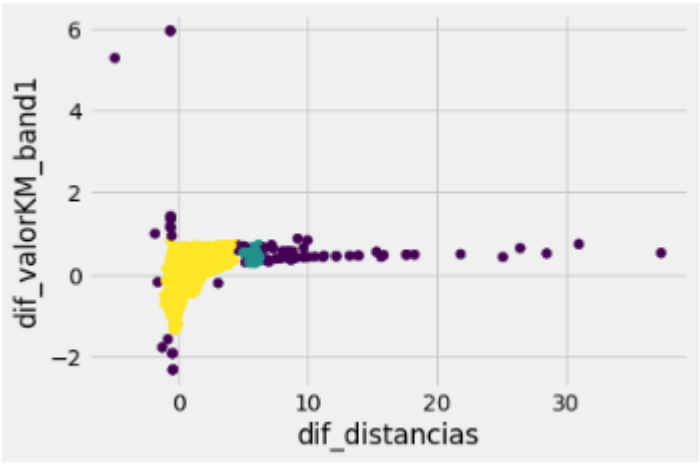


Figura 76 – Aplicação do algoritmo *HDBSCAN* com *min_cluster_size=16*.

O resultado obtido com o algoritmo *DBSCAN* foi mais satisfatório do que o obtido com o *K-means*. Observou-se que os pontos mais próximos de (0, 0) foram agrupados em um mesmo *cluster* e os mais afastados foram classificados como *outliers* (pontos de cor roxa mais escura). No entanto, ainda existiam alguns pontos cuja classificação não estava tão clara.

O resultado mais satisfatório foi obtido com o algoritmo *HDBSCAN*. Formou-se um *cluster* bem definido próximo à coordenada (0,0) do gráfico, que são os pontos cujas diferenças de distância e de valor por quilômetro são menores. À medida que se afasta desse ponto, apareceram os *outliers*, ou seja, as anomalias buscadas. Sendo assim, esse foi o algoritmo selecionado para a análise.

Observando todos os casos de *outliers*, verificou-se que eles somavam 426 registros de um total de 2837, ou seja, 15,0% dos casos. Dentre os *outliers*, 329 eram de corridas canceladas (77,2%).

Quase a totalidade das corridas canceladas (99,7%) foi classificada como *outlier*, o que era um resultado esperado, já que as corridas canceladas costumam apresentar registros diferentes das demais, com muitas inconsistências.

A corrida com status “ateste reprovado” também foi considerada *outlier*, o que está de acordo com a análise já feita anteriormente pelo governo, que reprovou o registro desta corrida.

Cerca de 21% das corridas com status “aguardando ateste” foram identificadas como *outliers*, ou seja, é possível que seus registros sejam reprovados e o status mude para “ateste reprovado”.

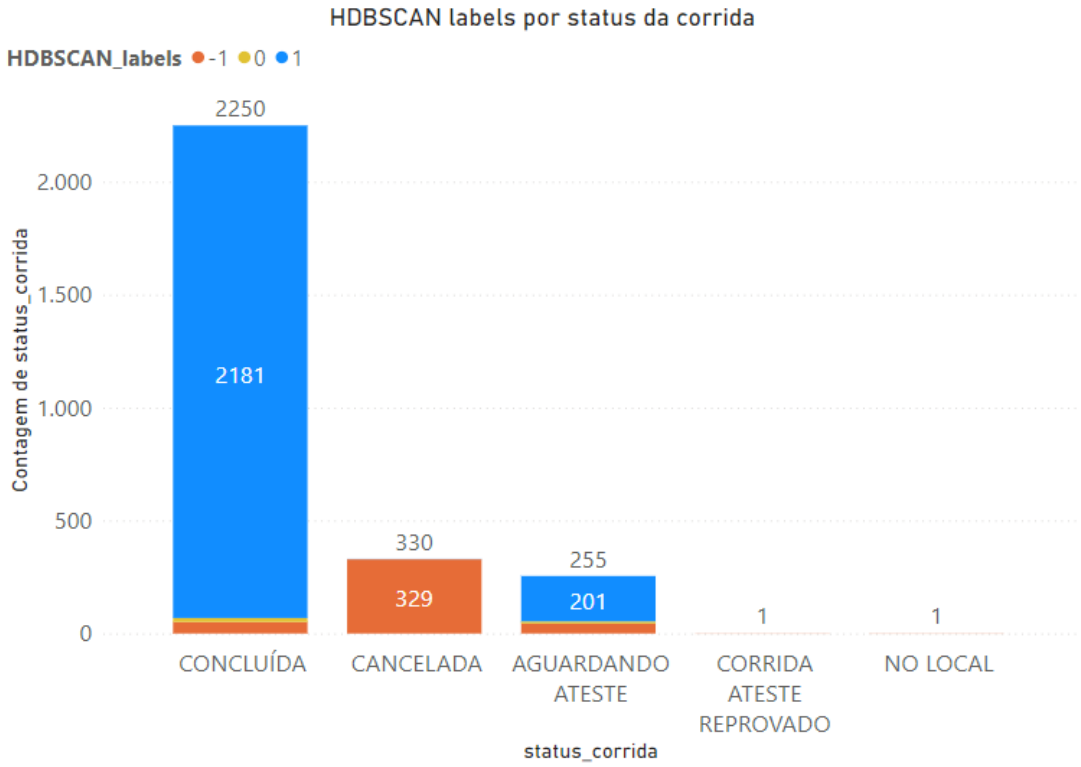


Figura 77 – Verificação do status da corrida dos *outliers*.

Também foram plotados novamente alguns gráficos para análise, mas dessa vez considerando o resultado obtido pela aplicação do algoritmo HDBSCAN. O primeiro deles foi o que mostra as distâncias calculadas através das coordenadas de origem e destino *versus* a quilometragem total registrada. O ideal seria que os valores de x e de y fossem o mais próximo possível, formando uma reta. Os pontos muito fora dessa reta mostram possíveis anomalias. Pela Figura abaixo é possível ver que o resultado obtido foi satisfatório, pois esses pontos foram realmente identificados como *outliers* pela aplicação do algoritmo.

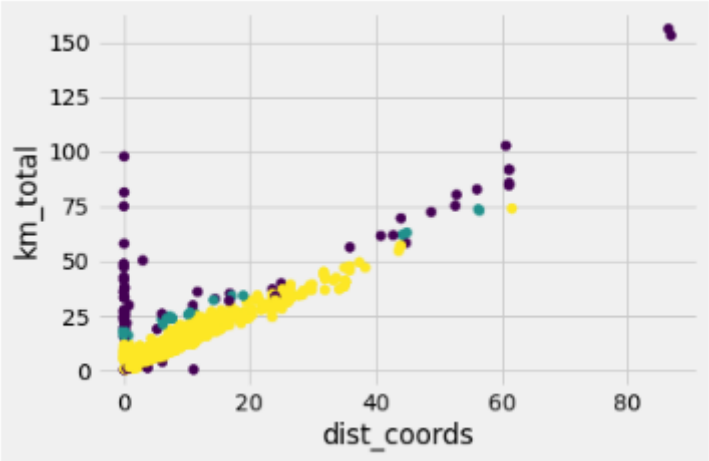


Figura 78 – Gráfico de *dist_coords x km_total*.

Em seguida, foram plotados os gráficos das Figuras abaixo. Eles mostraram que a relação entre a duração, o valor e a quilometragem total da corrida corroboraram que o resultado do algoritmo identificou corretamente os pontos mais discrepantes. Mas eles também mostraram pontos que a princípio seriam tratados como pontos conformes pela simples análise dos gráficos e que foram identificados como *outliers*. Isso se deve ao fato de que o modelo não levou em consideração todas as possíveis dimensões, apenas as com maior influência sobre as anomalias. Além disso, mostrou que pontos dentro da reta, que à primeira vista não seriam vistos como *outliers*, poderiam sim apresentar alguma inconformidade e deveriam ser verificados.

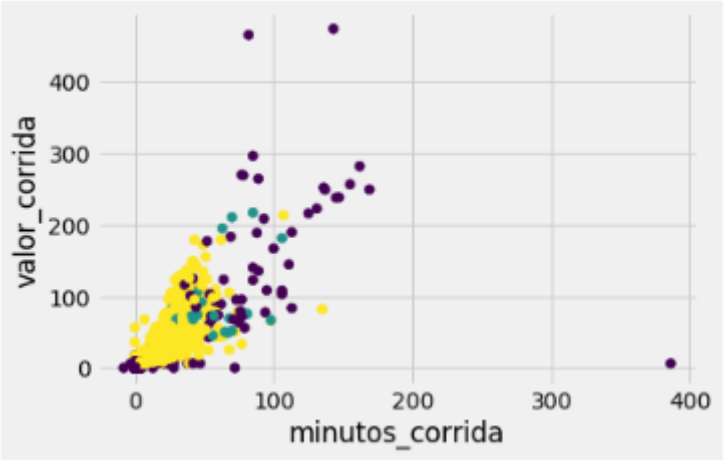


Figura 79 – Gráfico de *minutos_corrida x valor_corrida*.

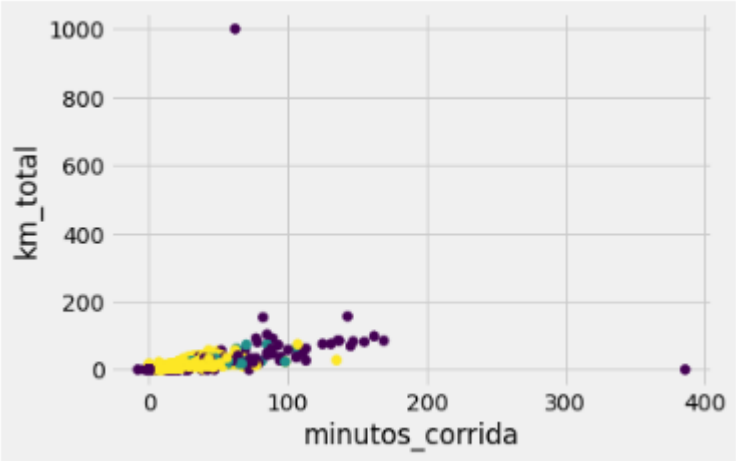


Figura 80 – Gráfico de *minutos_corrida x km_total*.

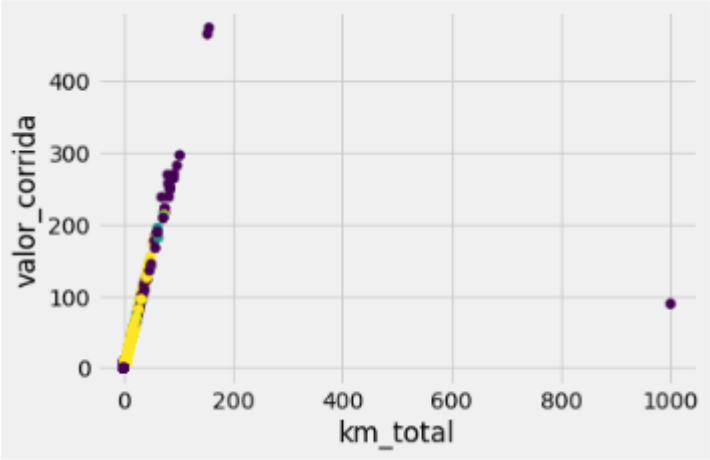


Figura 81 – Gráfico de *km_total x valor_corrida*.

Também foi analisada a quantidade de *outliers* por órgão de cada base de origem. Para isso, foram desconsideradas as corridas canceladas. Pode ser visto pelas Figuras abaixo que os gráficos do DF e de SP não apresentaram valores que chamaram a atenção. No entanto, no RJ, foram identificados órgãos cujos registros das corridas de táxi apresentaram grande quantidade de *outliers*, como é o caso da UFRRJ e do Hospital Gaffree e Guinle. A maioria desses casos ocorre pela elevada diferença entre a quilometragem calculada pelas coordenadas de origem e de destino (*dist_coords*) e a quilometragem total registrada (*km_total*).

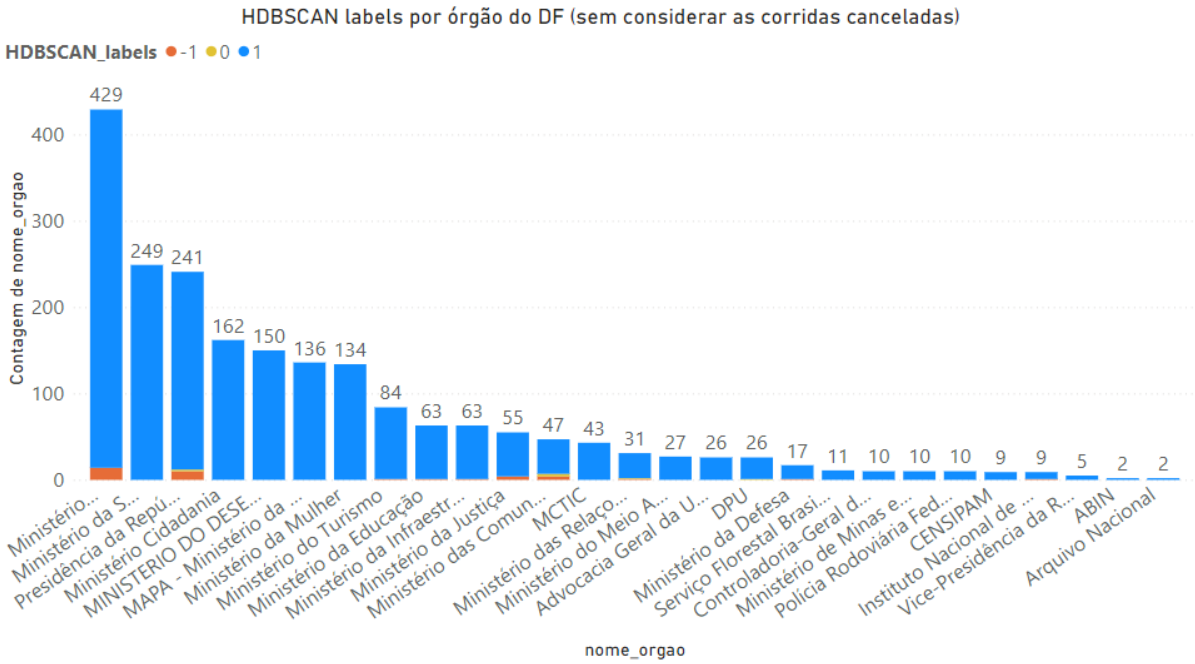


Figura 82 – Verificação dos *outliers* por órgão do DF.

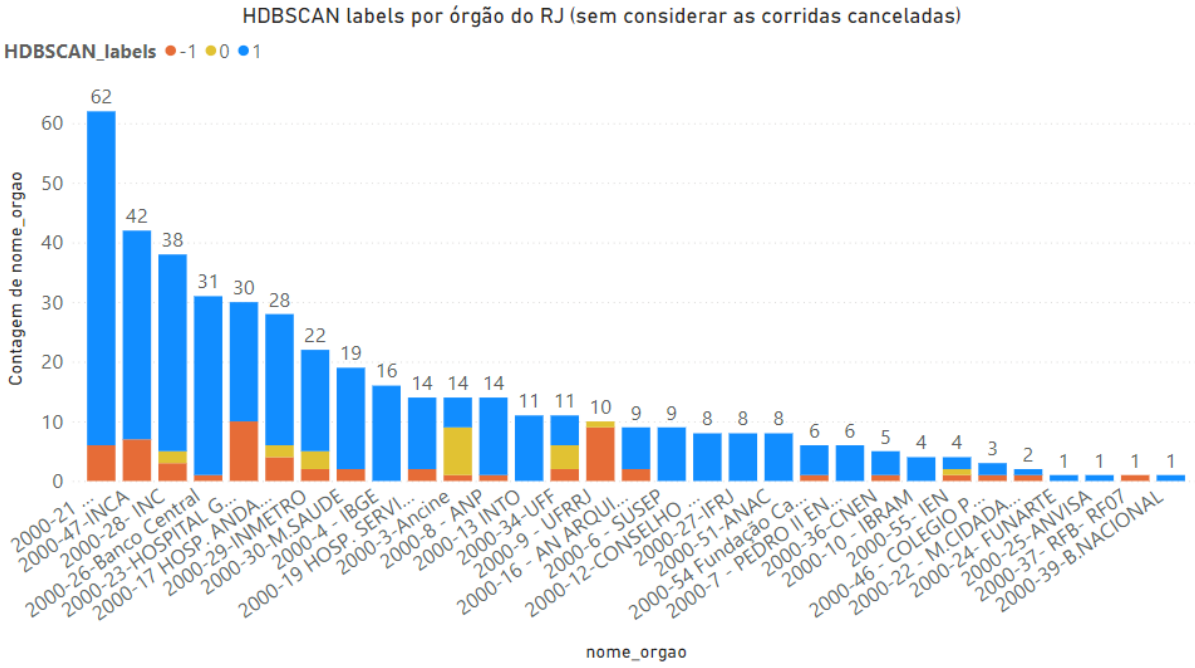


Figura 83 – Verificação dos *outliers* por órgão do RJ.

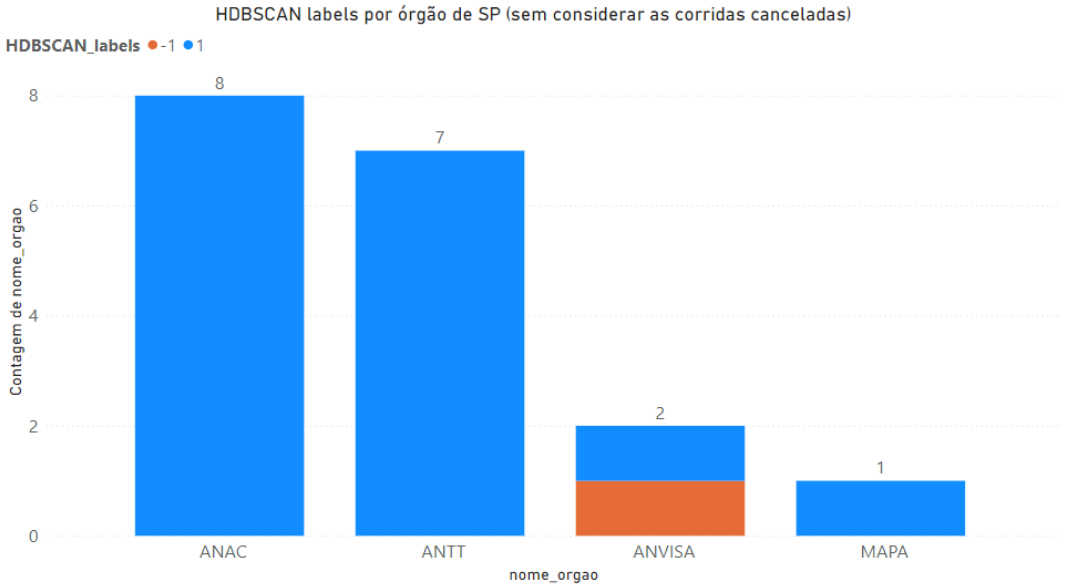


Figura 84 – Verificação dos *outliers* por órgão de SP.

A quantidade de *outliers* de acordo com o município de origem da corrida de táxi também foi avaliada. No estado do RJ, principalmente, foram identificadas algumas cidades com grande concentração de *outliers*, como Petrópolis e São João de Meriti. A identificação dessas cidades pode ajudar a identificar e a sanar as possíveis causas.

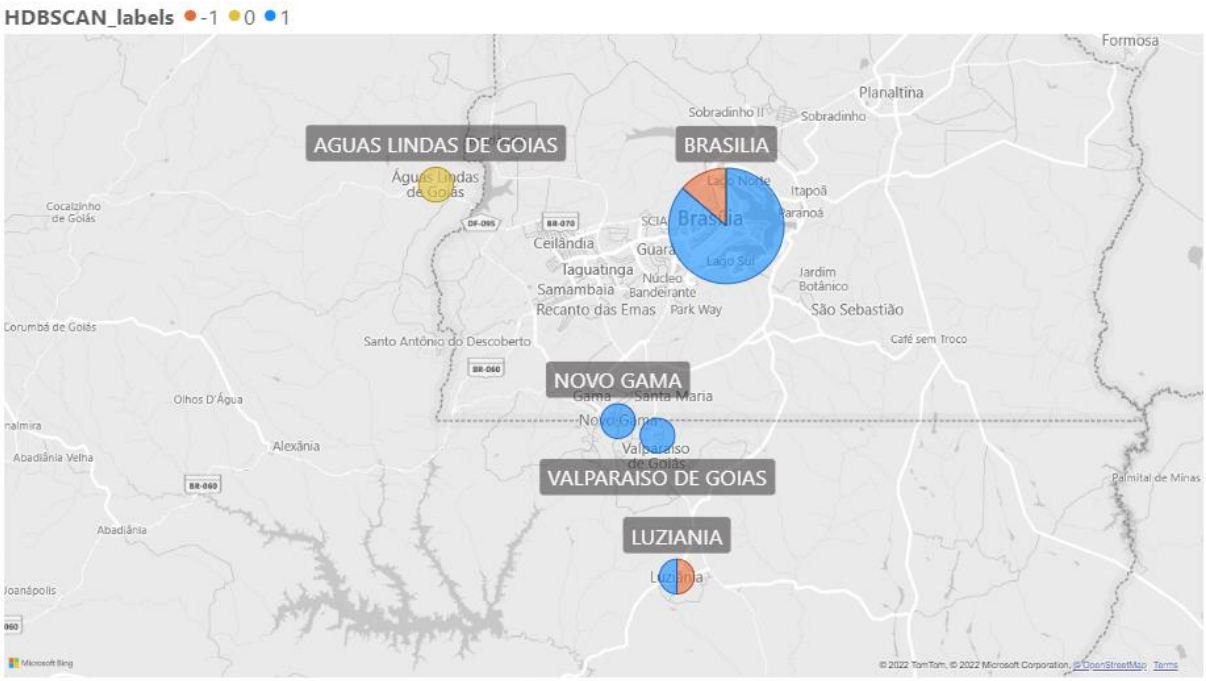


Figura 85 – Verificação dos *outliers* por cidade de origem no DF.

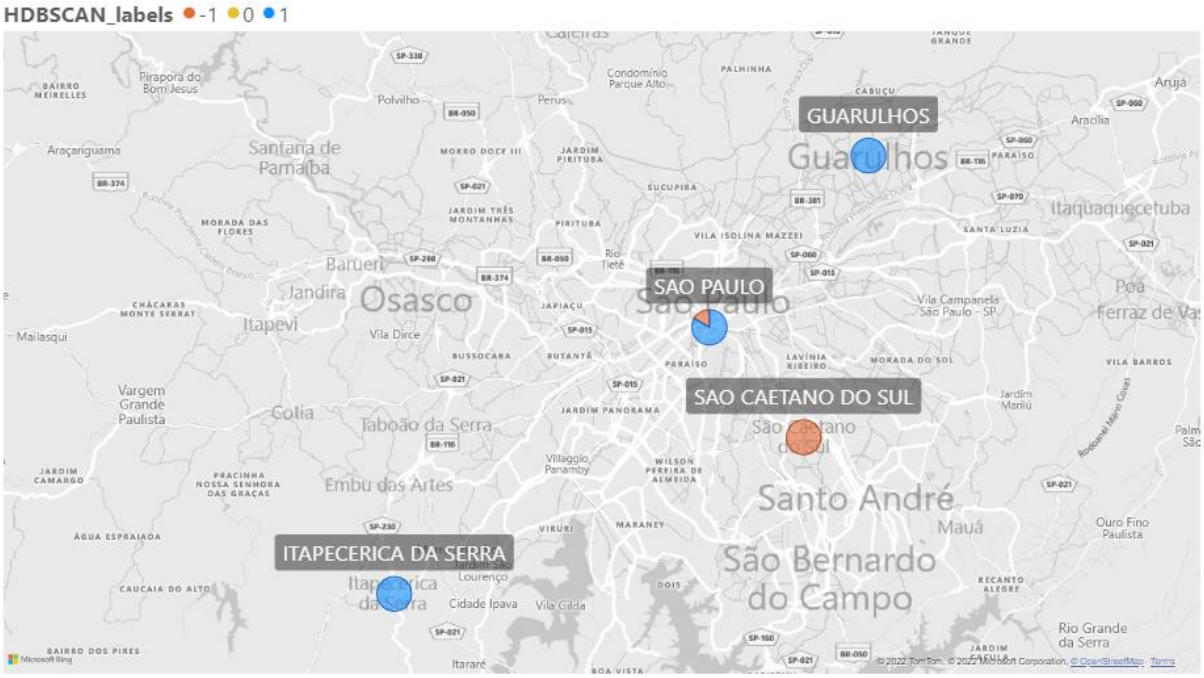


Figura 86 – Verificação dos *outliers* por cidade de origem em SP.

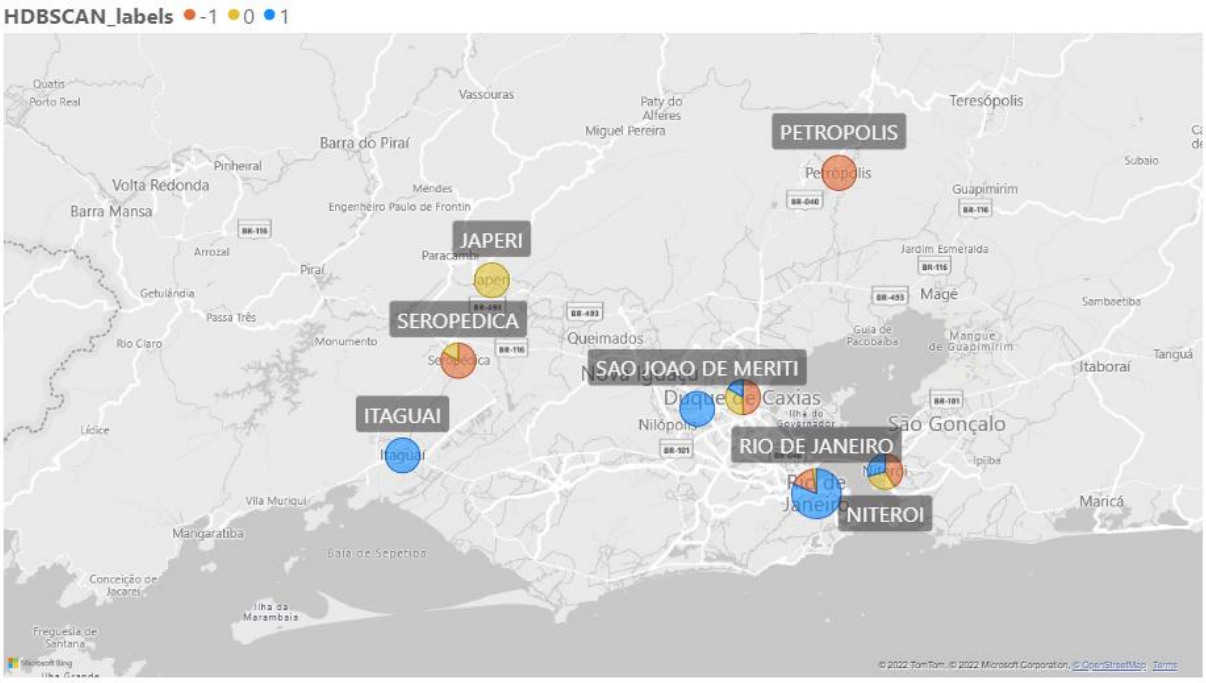


Figura 87 – Verificação dos *outliers* por cidade de origem no RJ.

Ao analisar a totalidade dos dados, observou-se que os principais grupos de *outliers* foram formados por:

- corridas canceladas;

- grandes diferenças de valor pago por quilômetro ao se comparar o valor calculado através dos dados registrados e o valor cobrado usualmente pelos táxis na bandeira 1;
- grandes diferenças de distâncias quando são comparadas a quilometragem total registrada e a quilometragem calculada através das coordenadas de origem e de destino;
- coordenadas do destino efetivo em branco e coordenadas de origem preenchidas sem vírgula, fazendo com que a diferença de distâncias ficasse muito grande. Isso mostra erros de cadastro que devem ser corrigidos pelos servidores;
- quilometragem total muito pequena, o que deixa o valor por quilômetro bastante elevado, já que a bandeirada é cobrada de toda forma, independentemente da quilometragem rodada.

A seguir, foram mostrados alguns registros classificados como *outliers*.

Caso 1:

Dados registrados no TáxiGov:

QRU: 518059

Km total: 81,2km

Valor da corrida: R$256,58

Minutos da corrida: 155 min

Coordenadas de origem: -15.79757, -47.865944

Coordenadas do destino efetivo: -15.797906, -47.866346

Coordenadas do destino solicitado: -16.034727, -48.058332

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino efetivo:

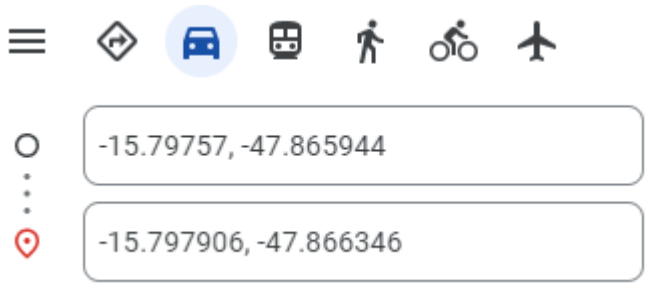
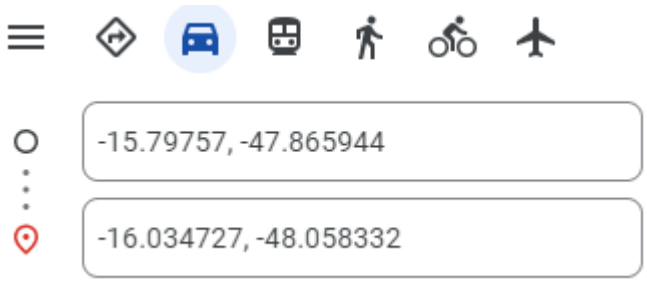


Figura 88 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino efetivo.

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino solicitado:



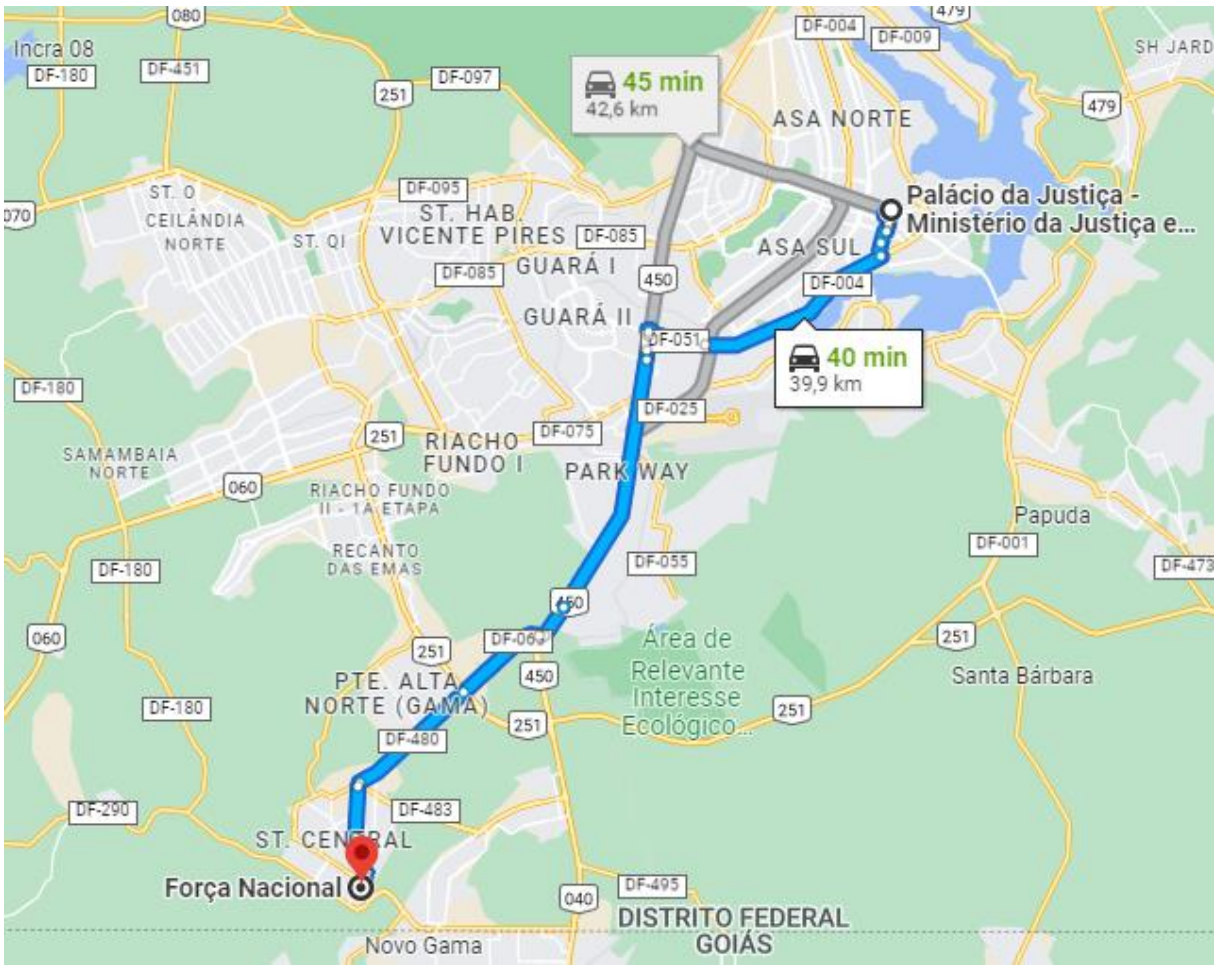


Figura 89 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino solicitado.

Esse é um caso de *outlier* em que a quilometragem total registrada no Táxi-Gov é muito diferente da quilometragem calculada através das coordenadas de origem e de destino.

Se fossem consideradas somente as coordenadas de origem e do destino efetivo, o trajeto teria apenas 150m de distância e duraria um minuto de carro. Já considerando as coordenadas do destino solicitado, o percurso deveria ter quilometragem aproximada de 40km e duração de cerca de 40 minutos.

No entanto, pelos dados registrados no TáxiGov, a corrida teve duração de 155 minutos, com uma quilometragem total registrada de 81,2km. Os dados registrados indicam que, provavelmente, a corrida teve início e fim no mesmo lugar, ou em locais bem próximos. Além disso, o tempo de permanência no local foi pequeno,

de aproximadamente 30 minutos, tempo no qual o motorista permaneceu esperando com o taxímetro rodando, ou seja, cujo valor da corrida foi aumentando.

O motivo registrado para a corrida foi de reunião externa. Isso levanta o questionamento da real necessidade desse deslocamento de mais de 80km, que totalizou R$256,58, para a provável realização de uma reunião tão curta, com o motorista esperando e cobrando pelo tempo parado.

A identificação de casos como esses pelo algoritmo seria o passo inicial para a emissão de alertas e para futuros questionamentos das unidades em que eles estão ocorrendo. Casos assim poderiam dar origem a recomendações de realização de reuniões remotas nas situações em que isso é possível, o que geraria grande economia de tempo e de dinheiro.

Caso 2:

Dados registrados no TáxiGov:

QRU: 519609

Km total: 0,1km

Valor da corrida: R$6,32

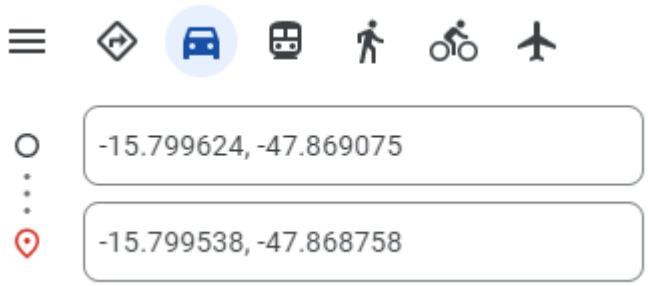
Minutos da corrida: 2 min

Coordenadas de origem: -15.799624, -47.869075

Coordenadas do destino efetivo: -15.799538, -47.868758

Coordenadas do destino solicitado: -15.794738, -47.873794

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino efetivo:



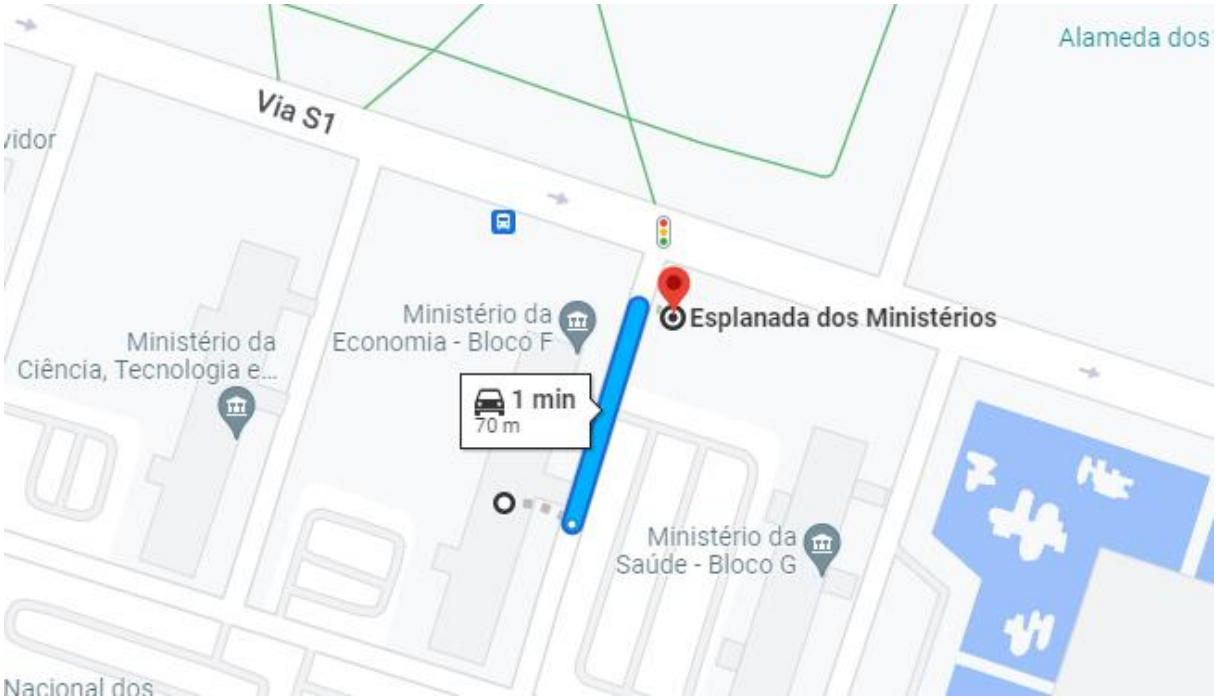
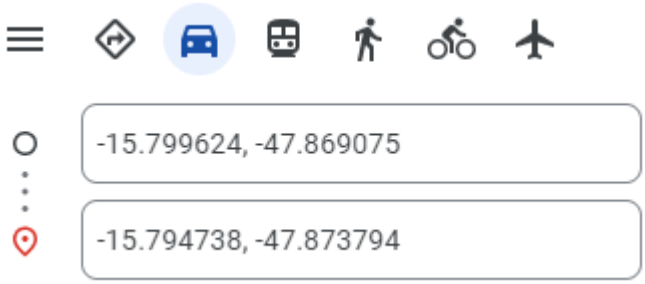


Figura 90 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino efetivo.

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino solicitado:



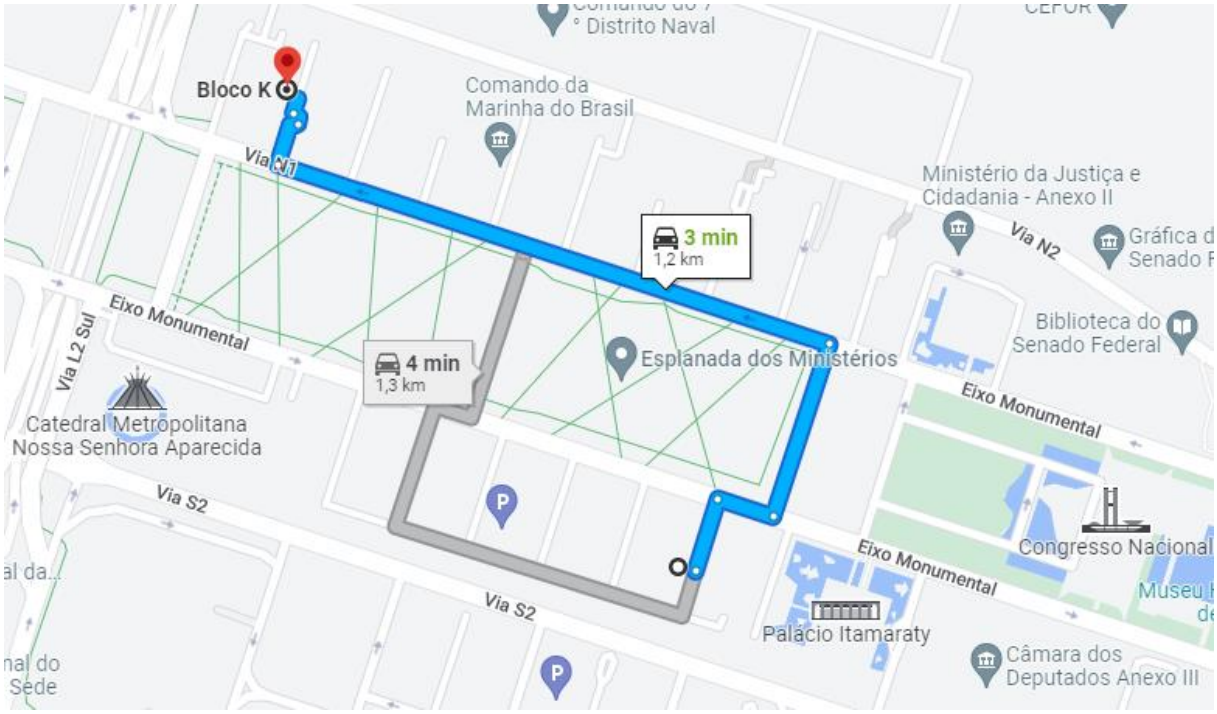


Figura 91 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino solicitado.

Esse é um caso de *outlier* identificado em que a diferença de valor cobrado por quilômetro calculado através dos dados registrados é grande em relação ao valor cobrado usualmente pelos taxistas do município. Isso ocorre devido à baixa quilometragem registrada.

A quilometragem e a duração da corrida são condizentes com o destino efetivo, que está localizado muito mais próximo do que o destino solicitado. Como a quilometragem percorrida foi muito baixa, o valor por quilômetro calculado ficou alto, dando grande diferença em relação ao valor por quilômetro praticado usualmente. O maior questionamento que deve ser feito nesse caso é a necessidade de se solicitar um táxi para percorrer somente 100 metros. Além disso, deve ser verificado o motivo de o destino solicitado ser diferente do destino efetivo.

Caso 3:

Dados registrados no TáxiGov:

QRU: 518542

Km total: 1000km

Valor da corrida: R$89,00

Minutos da corrida: 62 min

Coordenadas de origem: -15.838558, -48.037657

Coordenadas do destino efetivo: -15.873431, -47.909103

Coordenadas do destino solicitado: -15.871267, -47.908699

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino efetivo:

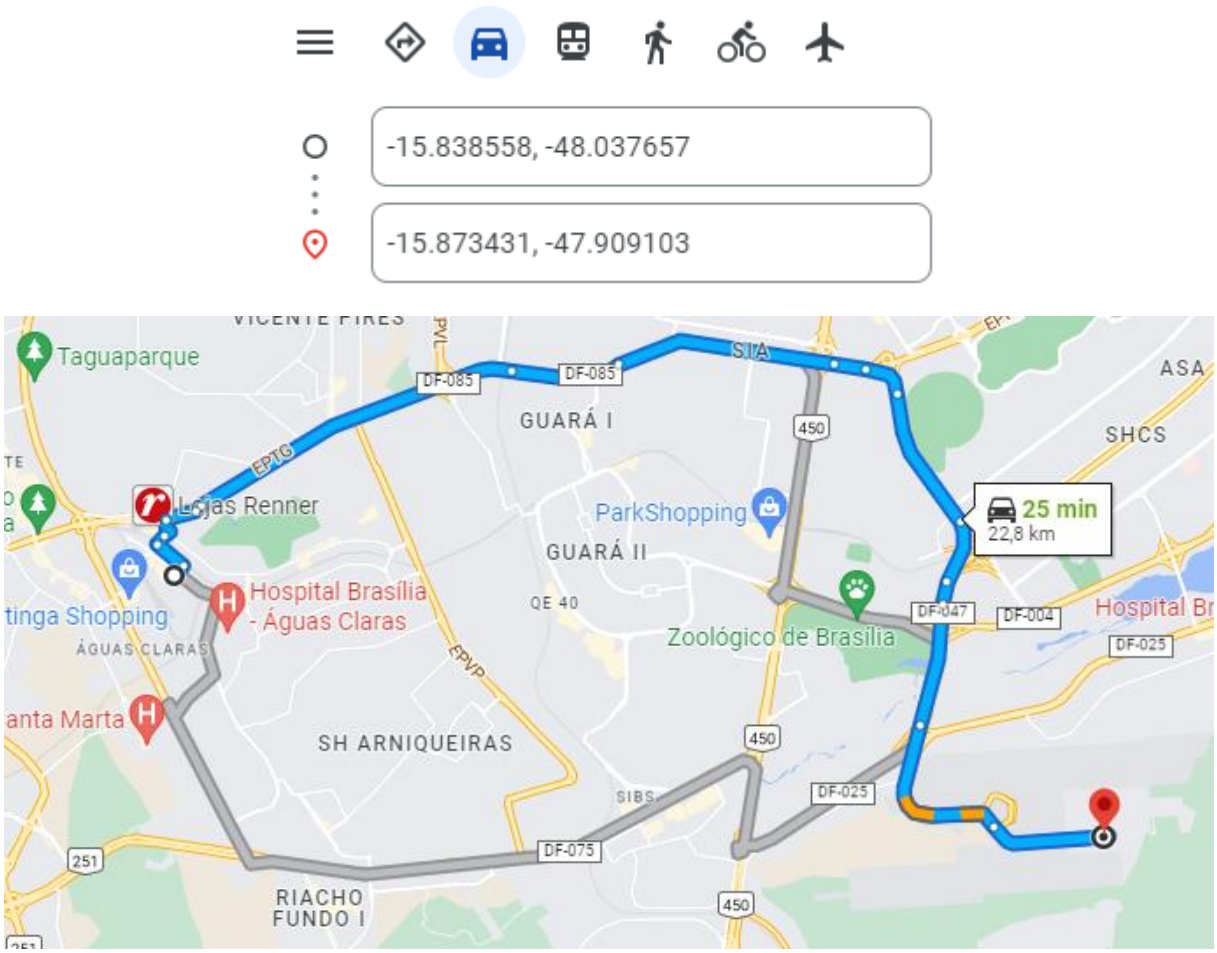


Figura 92 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino efetivo.

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino solicitado:

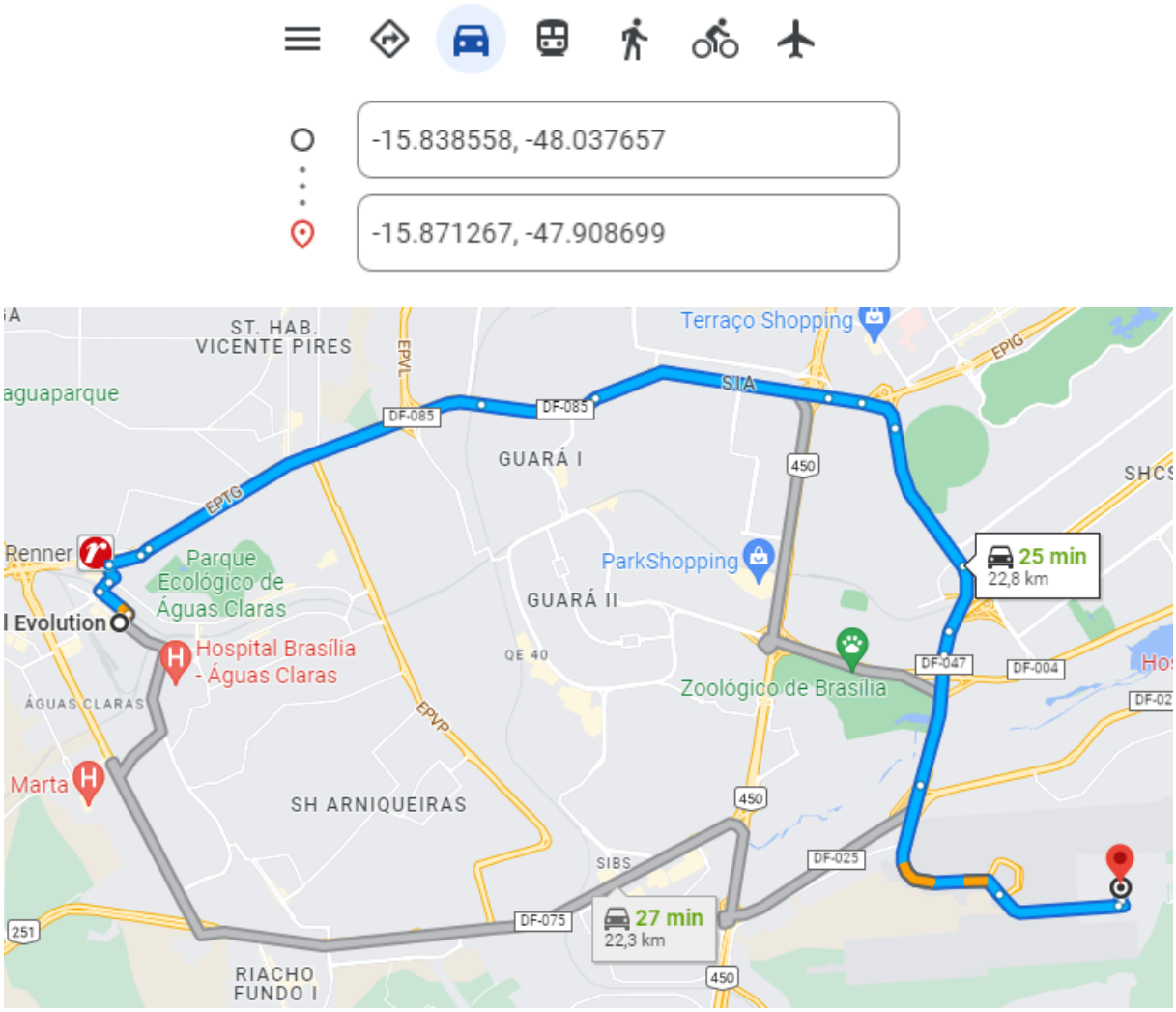


Figura 93 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino solicitado.

Esse é um caso de *outlier* cuja diferença da quilometragem total registrada e da quilometragem calculada através das coordenadas de origem e de destino é muito grande.

Os destinos solicitado e efetivo estão iguais. O que difere muito é o registro de quilometragem total igual a 1000km, sendo que a distância da origem ao destino é de cerca de 23km. Além disso, a corrida teve duração de 62 minutos, sendo que o percurso leva cerca de 25 minutos para ser percorrido. Mesmo considerando uma corrida de ida e volta, a quilometragem registrada continuaria com uma diferença acima de 900km.

Em um contexto de auditoria, por exemplo, esse caso também seria um alerta a ser investigado.

Caso 4:

Dados registrados no TáxiGov:

QRU: 782557

Km total: 41km

Valor da corrida: R$123,39

Minutos da corrida: 64 min

Coordenadas de origem: -22.917268, -43.220079

Coordenadas do destino efetivo: -22.9175, -43.2203

Coordenadas do destino solicitado: -22.917476, -43.220255

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino efetivo:

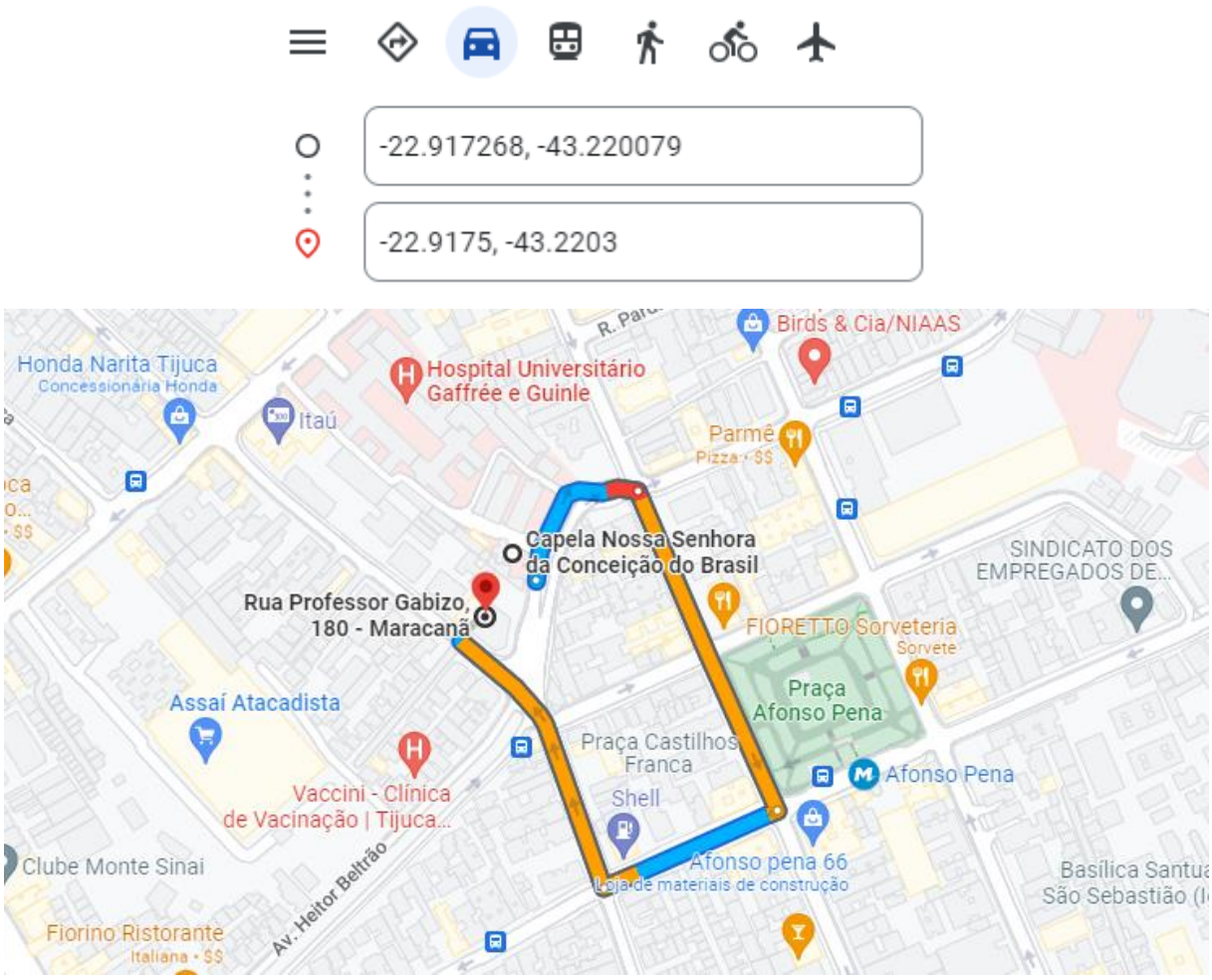


Figura 94 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino efetivo.

Traçado do percurso no Google Maps considerando as coordenadas de origem e de destino solicitado:

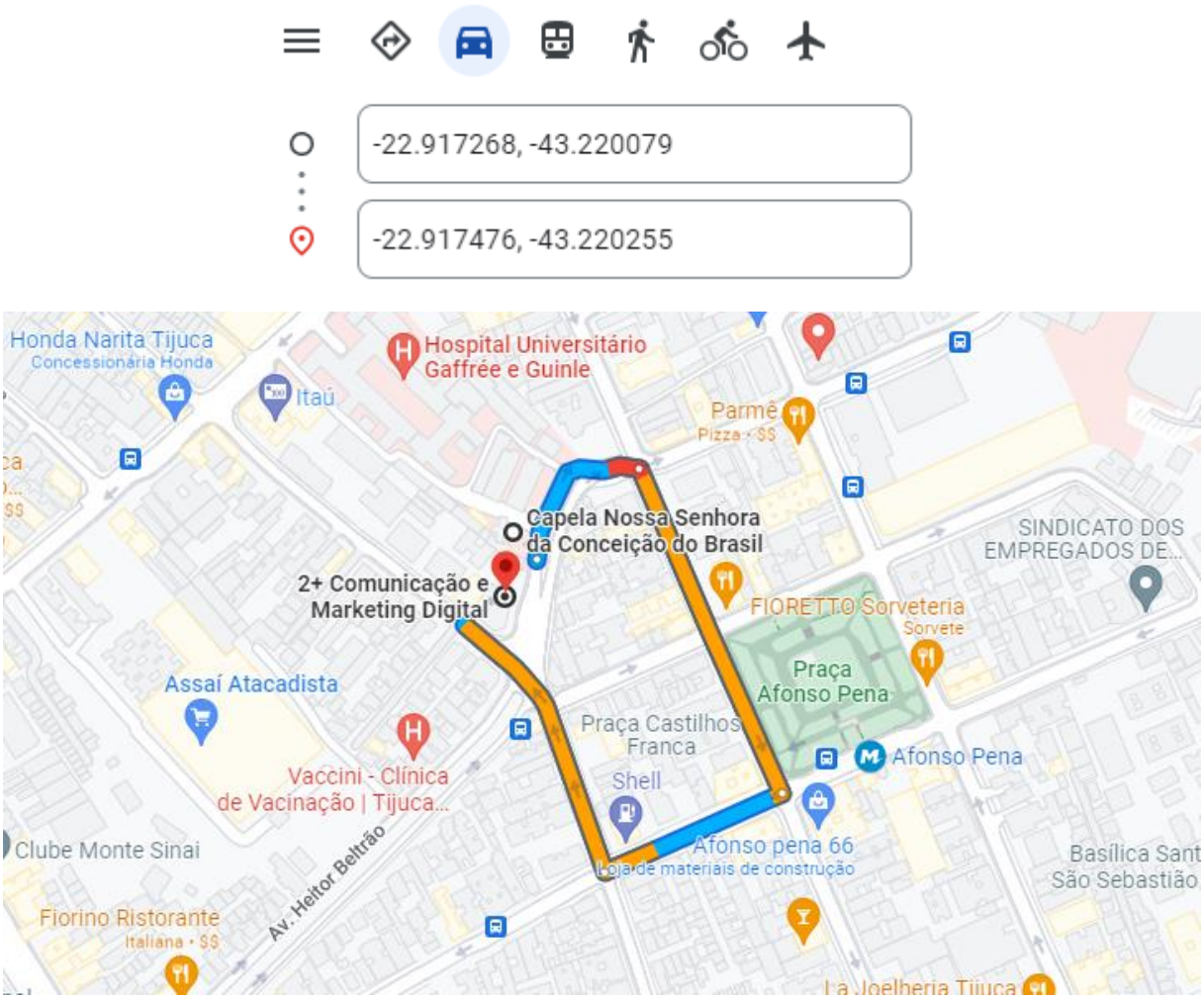


Figura 95 – Percurso traçado no Google Maps considerando as coordenadas de origem e de destino solicitado.

Esse é um caso de *outlier* do Hospital Gaffree e Guinle, no Rio de Janeiro, que foi um órgão que se destacou pela grande quantidade de *outliers* identificados.

As coordenadas de destino efetivo e solicitado são iguais, mas são muito próximas das coordenadas da origem da corrida, o que não justificaria percorrer esse trajeto de táxi.

No entanto, a quilometragem total registrada da corrida foi de 41km, com duração de 64 minutos, dados que não condizem com as coordenadas.

Verificando-se os endereços registrados, observa-se que o endereço de origem (Rua Silva Ramos, Tijuca, Rio de Janeiro, RJ) está de acordo com a coordena-

da de origem. Já o endereço solicitado é diferente (Rua Dr. Luiz Palmier, 762, Barreto, Niteroi, RJ).

Ao traçar o trajeto pelo Google Maps dos endereços cadastrados, obtém-se o seguinte:

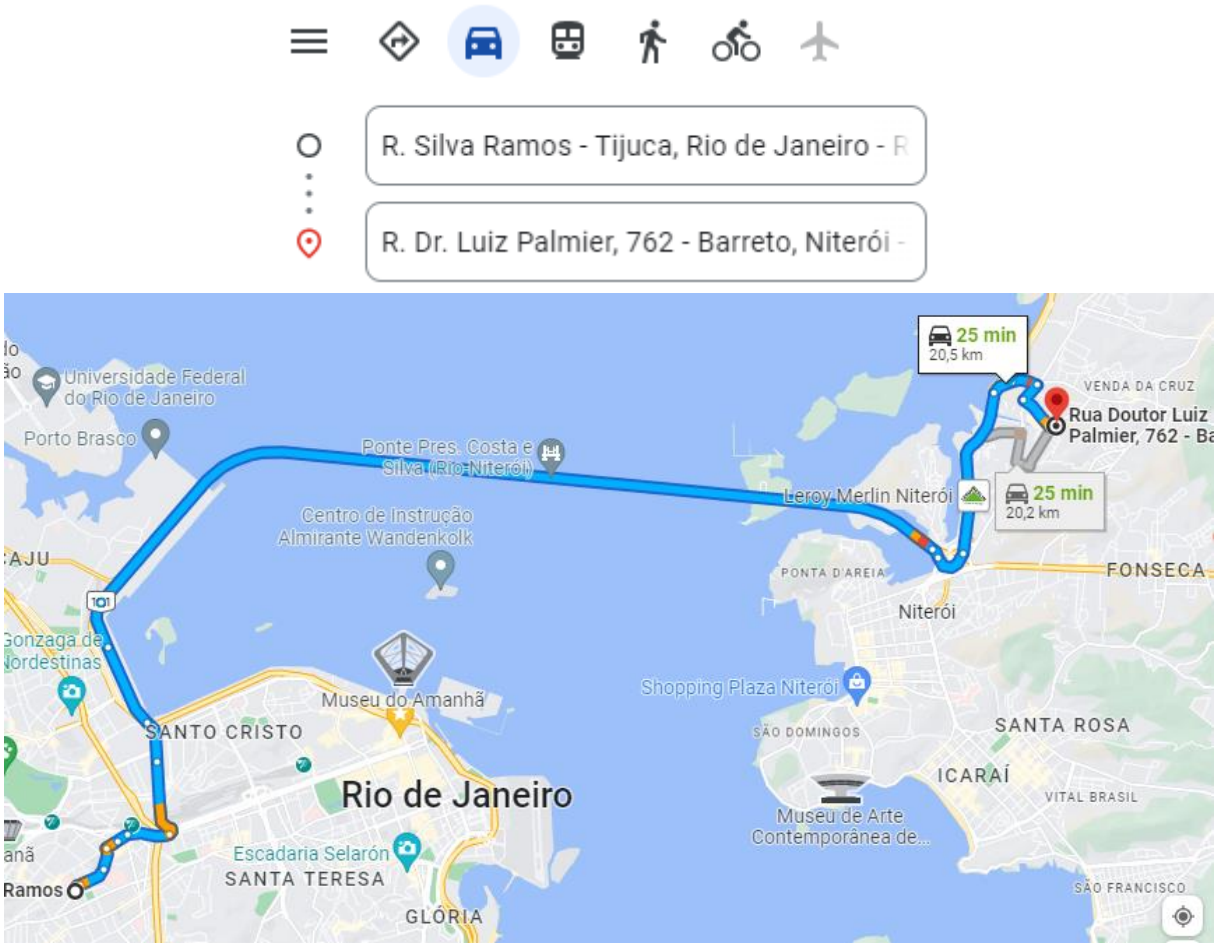


Figura 96 – Percurso traçado no Google Maps considerando o endereço de origem e o endereço solicitado cadastrados.

Pelos dados obtidos pelo Google Maps e pelas informações registradas no TáxiGov, pode-se concluir que foi uma corrida de ida e volta, com retorno para endereço próximo ao de origem.

O motivo cadastrado para a corrida foi de “atendimento técnico”, o que, a princípio, estaria de acordo com as atividades do Hospital.

Nesse caso, a grande diferença de quilometragem pode ter ocorrido por erros no registro das coordenadas. Deve-se verificar, portanto, como esse registro é feito. Se é feito automaticamente através de um sistema, ou se é cadastrado manualmen-

te por um servidor. Em ambos os casos, o problema deve ser corrigido, pois foram identificados muitos registros com o mesmo erro.

Pelas análises feitas anteriormente, concluiu-se que a aplicação dos modelos de machine learning possibilita a visualização de aspectos que seriam de difícil entendimento sem eles.

Através deles, foi possível identificar os casos que devem ser investigados mais detidamente em uma possível auditoria, sendo os principais: registros de quilometragem total e de valor da corrida com valores que dão origem a R$/km muito diferentes dos praticados pelos taxistas em cada município e registros de pontos de partida e de destinos que não condizem com a quilometragem total cadastrada.

Também foram identificados possíveis erros de cadastros, que são passíveis de ocorrer. Mas nos casos em que esses erros se repetem com frequência, podem ser recomendados treinamentos nas unidades de maior problema.

Alguns órgãos e estados também se destacaram pela grande quantidade de *outliers*, como foi o caso da UFRRJ e do Hospital Gaffree e Guinle no Rio de Janeiro. Identificar os locais onde há essa concentração de anomalias é importante para facilitar a sua correção.

Uma possível aplicação futura seria a automatização da identificação das anomalias com a emissão de alertas para a análise dos casos levantados. Esse alerta poderia até mesmo ser emitido em tempo real, ou seja, assim que o registro da corrida fosse feito, o que poderia ajudar na correção de erros de cadastros, quando for o caso. Essa automatização facilitaria a aprovação dos registros das corridas, tornando o processo mais rápido e efetivo.

8. Links

A seguir estão os links para o vídeo com a apresentação resumida do trabalho e para o repositório contendo todos os arquivos utilizados.

Link para o vídeo: https://youtu.be/PY3XGJ6d8Co
Link para o repositório: https://github.com/anapsr/tcc-puc-ciencia-de-dados

REFERÊNCIAS

Clustering. **Scikit-learn**, 2022. Disponível em: <https://scikit-learn.org/stable/modules/clustering.html>. Acesso em: 16 de abril de 2022.

Compare the effect of diferente scalers on data with outliers. **Scikit-learn**, 2022. Disponível em: <https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py>. Acesso em: 16 de abril de 2022.

Conjunto de dados do TáxiGov. **Portal Brasileiro de Dados Abertos**, 2022. Disponível em: <http://dados.gov.br/dataset>. Acesso em 09 de abril de 2022.

DBSCAN Python Example: The Optimal Value For Epsilon (EPS). **Towards Data Science**, 2019. Disponível em: <https://towardsdatascience.com/machine-learning-clustering-dbscan-determine-the-optimal-value-for-epsilon-eps-python-example-3100091cfbc#:~:text=In%20layman's%20terms%2C%20we%20find,and%20select%20that%20as%20epsilon>. Acesso em 29 de maio de 2022.

Decreto nº 9.287, de 15 de fevereiro de 2018. **Planalto**, 2022. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/decreto/D9287.htm>. Acesso em: 14 de abril de 2022.

Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra. **IOP Science**, 2016. Disponível em: <https://iopscience.iop.org/article/10.1088/1755-1315/31/1/012012/pdf>. Acesso em 29 de maio de 2022.

Elbow Method. **Scikit-yb**, 2022. Disponível em: <https://www.scikit-yb.org/en/latest/api/cluster/elbow.html#:~:text=The%20elbow%20method%20runs%20k,point%20to%20its%20assigned%20center>. Acesso em: 16 de abril em 2022.

HDBSCAN. Github, 2022. Disponível em: <https://github.com/scikit-learn-contrib/hdbscan>. Acesso em: 16 de abril de 2022.

How DBSCAN works and why should we use it?. **Towards Data Science**, 2017. Disponível em: <https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80>. Acesso em: 16 de abril de 2022.

K-means: o que é, como funciona, aplicações e exemplo em Python. **Medium**, 2020. Disponível em: <https://medium.com/programadores-ajudando-programadores/k-means-o-que-%C3%A9-como-funciona-aplica%C3%A7%C3%B5es-e-exemplo-em-python-6021df6e2572>. Acesso em: 16 de abril de 2022.

Machine Learning: Conceitos e Modelos – Parte II: Aprendizado Não-Supervisionado. **Medium**, 2020. Disponível em: <https://tatianaesc.medium.com/machine-learning-conceitos-e-modelos-parte-ii-aprendizado-n%C3%A3o-supervisionado-fb6d83e4a520>. Acesso em: 15 de abril de 2022.

Parameter Selection for HDBSCAN. **Read the docs**, 2022. Disponível em: <https://hdbscan.readthedocs.io/en/latest/parameter_selection.html>. Acesso em 16 de abril de 2022.

Taxi Gov: Mobilidade de Servidores no Governo Federal. **Repositório ENAP**, 2022. Disponível em: <https://repositorio.enap.gov.br/bitstream/1/4154/1/Taxi%20Gov_Mobilidade%20de%20Servidores%20no%20Governo%20Federal.pdf>. Acesso em: 14 de abril de 2022.

TáxiGov - Ministério da Economia. **Gov.br**, 2022. Disponível em: <https://www.gov.br/economia/pt-br/assuntos/gestao/central-de-compras/taxigov>. Acesso em: 14 de abril de 2022.

APÊNDICE

Programação/Scripts

Código feito no Notebook Jupyter:

```
#!/usr/bin/env python
# coding: utf-8

# In[269]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import datetime


# In[270]:


df = pd.read_csv('taxigov-corridas-7-dias-vF.csv')


# In[271]:


df.head()


# In[272]:


df.shape


# In[273]:
```

```
df.dtypes

# In[274]:

df.describe()

# In[275]:

# Como a coluna conteste_info está vazia, ela será deletada

# In[276]:

del df["conteste_info"]

# In[277]:

df.shape

# In[278]:

# Substituição do valor vazio da coluna qru_corrida

# In[279]:

df['qru_corrida'].fillna(784630.0, inplace=True)
df.describe() # Para conferir a substituição
```

In[280]:

Preenchendo os valores vazios das colunas km_total e valor_corrida

In[281]:

```
df['km_total'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['km_total'].fillna(0.0, inplace=True)

df['valor_corrida'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['valor_corrida'].fillna(0.0, inplace=True)
```

In[282]:

```
df['km_total'].describe()
```

In[283]:

```
df['valor_corrida'].describe()
```

In[284]:

Preenchendo os valores vazios das colunas que faltam

In[285]:

```
df['destino_solicitado_latitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_solicitado_latitude'].fillna(0.0, inplace=True)
```

```
df['destino_solicitado_longitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_solicitado_longitude'].fillna(0.0, inplace=True)

df['destino_efetivo_latitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_efetivo_latitude'].fillna(0.0, inplace=True)

df['destino_efetivo_longitude'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['destino_efetivo_longitude'].fillna(0.0, inplace=True)


# In[286]:


df.describe()


# In[287]:


# Padronizando os dados da coluna motivo_corrida


# In[288]:


df.motivo_corrida.unique()


# In[289]:


df.loc[df['motivo_corrida'] == '01 - Reunião Externa (Ida/Volta)', 'motivo_corrida'] = '1 - REUNIAO EXTERNA'
df.loc[df['motivo_corrida'] == '01 Reunião Externa (Ida/Volta)', 'motivo_corrida'] = '1 - REUNIAO EXTERNA'
df.loc[df['motivo_corrida'] == '02 - Entrega de Documentos', 'motivo_corrida'] = '2 - ENTREGA DE DOCUMENTOS'
df.loc[df['motivo_corrida'] == '02 Entrega de Documentos', 'motivo_corrida'] = '2 - ENTREGA DE DOCUMENTOS'
```

```
df.loc[df['motivo_corrida'] == '04 Outros', 'motivo_corrida'] = '4 - OUTROS'
df.loc[df['motivo_corrida'] == '05 - Perícia Médica', 'motivo_corrida'] = '5 - PERICIA MEDICA'
df.loc[df['motivo_corrida'] == '06 - Inspeção/Fiscalização', 'motivo_corrida'] = '6 - INSPECAO/FISCALIZACAO'
df.loc[df['motivo_corrida'] == '06 Fiscalização', 'motivo_corrida'] = '6 - INSPECAO/FISCALIZACAO'
df.loc[df['motivo_corrida'] == '07 - Evento', 'motivo_corrida'] = '7 - EVENTO'
df.loc[df['motivo_corrida'] == '08 - Atendimento Técnico', 'motivo_corrida'] = '8 - ATENDIMENTO TECNICO'
df.loc[df['motivo_corrida'] == '03 - Capacitação/Treinamento', 'motivo_corrida'] = '3 - CAPACITACAO/TREINAMENTO'
df.loc[df['motivo_corrida'] == '10 - Outros', 'motivo_corrida'] = '10 - OUTROS'


# In[290]:


df.motivo_corrida.unique()


# In[291]:


# Analisando as bases de origem


# In[292]:


df.base_origem.unique()


# In[293]:


# Analisandos os motivos das corridas para cada base de origem


# In[294]:
```

```
motivos_df_series = df[df.base_origem == 'TAXIGOV_DF'].motivo_corrida.value_counts()
motivos_df_series


# In[295]:


plt.bar(motivos_df_series.keys(), motivos_df_series.array)
plt.xticks(rotation=45)


# In[296]:


motivos_rj_series = df[df.base_origem == 'TAXIGOV_RJ_10'].motivo_corrida.value_counts()
motivos_rj_series


# In[297]:


plt.bar(motivos_rj_series.keys(), motivos_rj_series.array)
plt.xticks(rotation=90)


# In[298]:


motivos_sp_series = df[df.base_origem == 'TAXIGOV_SP_10'].motivo_corrida.value_counts()
motivos_sp_series


# In[299]:


plt.bar(motivos_sp_series.keys(), motivos_sp_series.array)
plt.xticks(rotation=45)


# In[300]:
```

```
# Importando a planilha "preco-km-rodado.xlsx"

# In[301]:

df_preco_km = pd.read_excel('preco-km-rodado.xlsx')

# In[302]:

df_preco_km.head()

# In[303]:

# Padronizando a coluna origem_cidade de df para não ter nome diferente da mesma cidade

# In[304]:

df.origem_cidade.unique()

# In[305]:

df.loc[df['origem_cidade'] == 'BRASÍLIA', 'origem_cidade'] = 'BRASILIA'
df.loc[df['origem_cidade'] == 'RJ', 'origem_cidade'] = 'RIO DE JANEIRO'
df.loc[df['origem_cidade'] == 'Rio de Janeiro', 'origem_cidade'] = 'RIO DE JANEIRO'
df.loc[df['origem_cidade'] == 'Duque de Caxias', 'origem_cidade'] = 'DUQUE DE CAXIAS'
df.loc[df['origem_cidade'] == 'Niterói', 'origem_cidade'] = 'NITEROI'
df.loc[df['origem_cidade'] == 'São Caetano do Sul', 'origem_cidade'] = 'SAO CAETANO DO SUL'
df.loc[df['origem_cidade'] == 'São Paulo', 'origem_cidade'] = 'SAO PAULO'
```

```
# In[306]:


df.origem_cidade.unique()


# In[307]:


# Unindo os dois dataframes


# In[308]:


df_merged = pd.merge(df, df_preco_km, how='left', left_on='origem_cidade', right_on='Cidade')


# In[309]:


df_merged.head()


# In[310]:


df_merged.describe()


# In[311]:


# Preenhcer os campos vazios das colunas R$/km (bandeira 1), R$/km (bandeira 2), Bandeirada e Tarifa horária com a média


# In[312]:
```

```
df_merged['R$/km (bandeira 1)'].fillna(2.867977, inplace=True)
df_merged['R$/km (bandeira 2)'].fillna(3.641700, inplace=True)
df_merged['Bandeirada'].fillna(5.352066, inplace=True)
df_merged['Tarifa horária'].fillna(32.600792, inplace=True)


# In[313]:


# Cálculo do R$/km descontando a bandeirada


# In[314]:


# se km_total = 0 -> valor_por_km_desc_band = 110 (valor acima dos demais que mostrará que é um valor discrepante)
# não posso preencher com a média, por exemplo, pois perderá a informação que procuro


# In[315]:


for i in range(2837):
    if (df_merged.loc[i, 'km_total'] == 0):
        df_merged.loc[i, 'valor_por_km_desc_band'] = 110
    else:
        df_merged.loc[i, 'valor_por_km_desc_band'] = (df_merged.loc[i, 'valor_corrida'] - df_merged.loc[i, 'Bandeirada']) / df_merged.loc[i, 'km_total']


# In[316]:


df_merged.head()


# In[317]:
```

Cálculo da diferença do R$/km pago no TáxiGov e o R$/km cobrado pelos táxis em cada cidade na bandeira 1

In[318]:

df_merged['dif_valorKM_band1'] = df_merged['valor_por_km_desc_band'] - df_merged['R$/km (bandeira 1)']

In[319]:

Gráfico do R$/km calculado descontada a bandeirada e o R$/km cobrado pelos táxis

In[320]:

```
plt.scatter(df_merged['valor_por_km_desc_band'], df_merged['R$/km (bandeira 1)'])
plt.ylabel('R$/km (bandeira 1)')
plt.xlabel('valor_por_km_desc_band')
plt.show()
```

In[321]:

Eliminando o ponto mais discrepante para enxergar melhor os demais

In[322]:

```
filter_vKM = df_merged[(df_merged.valor_por_km_desc_band < 20)]
plt.scatter(filter_vKM['valor_por_km_desc_band'], filter_vKM['R$/km (bandeira 1)'])
plt.ylabel('R$/km (bandeira 1)')
plt.xlabel('valor_por_km_desc_band')
```

```
plt.show()


# In[323]:


# Cálculo da distância usando as coordenadas de origem e de destino


# In[324]:


# vectorized haversine function (obtido em:
#           https://stackoverflow.com/questions/40452759/pandas-latitude-longitude-to-distance-between-successive-rows)
def haversine(lat1, lon1, lat2, lon2, to_radians=True, earth_radius=6371):
    """
    slightly modified version: of http://stackoverflow.com/a/29546836/2901002

    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees or in radians)

    All (lat, lon) coordinates must have numeric dtypes and be of equal length.

    """
    if to_radians:
        lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    a = np.sin((lat2-lat1)/2.0)**2 +         np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return earth_radius * 2 * np.arcsin(np.sqrt(a))


# In[325]:


df_merged['dist_coords'] = haversine(df_merged.origem_latitude, df_merged.origem_longitude,
                                     df_merged.destino_efetivo_latitude, df_merged.destino_efetivo_longitude)
```

```
# In[326]:


df_merged.head()


# In[327]:


df_merged.describe()


# In[328]:


# Cálculo da diferença entre a quilometragem total registrada e a distância obtida pelas coordenadas


# In[329]:


df_merged['dif_distancias'] = df_merged['km_total'] - df_merged['dist_coords']


# In[330]:


# Gráfico dist_coords e km_total


# In[331]:


plt.scatter(df_merged['dist_coords'], df_merged['km_total'])
plt.ylabel('km_total')
plt.xlabel('dist_coords')
plt.show()


# In[332]:
```

```
# Eliminando os pontos mais discrepantes para enxergar melhor os demais

# In[333]:

filter_dist = df_merged[(df_merged.dist_coords < 1000)]
plt.scatter(filter_dist['dist_coords'], filter_dist['km_total'])
plt.ylabel('km_total')
plt.xlabel('dist_coords')
plt.show()

# In[334]:

# Gráfico das diferenças de distância e de R$/km

# In[335]:

plt.scatter(df_merged['dif_distancias'], df_merged['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')
plt.show()

# In[336]:

# Eliminando os pontos mais discrepantes para enxergar melhor os demais

# In[337]:

filter_difs = df_merged[(df_merged.dif_distancias > -100) & (df_merged.dif_distancias < 200)
```

```
                    & (df_merged.dif_valorKM_band1 < 20)]
plt.scatter(filter_difs['dif_distancias'], filter_difs['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')
plt.show()


# In[338]:


# Criar coluna da % da dif_distancias em relação ao km_total
# se km_total = 0 -> perc_dif_dist = 1000 (valor acima dos demais que mostrará que é um valor discrepante)


# In[339]:


for i in range(2837):
    if (df_merged.loc[i, 'km_total'] == 0):
        df_merged.loc[i, 'perc_dif_dist'] = 1000
    else:
        df_merged.loc[i, 'perc_dif_dist'] = (df_merged.loc[i, 'dif_distancias'] / df_merged.loc[i, 'km_total'])*100


# In[340]:


df_merged.head()


# In[341]:


df_merged.describe()


# In[342]:
```

```
# Novo gráfico com dif_valorKM_band1 e perc_dif_dist


# In[421]:


plt.scatter(df_merged['perc_dif_dist'], df_merged['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('perc_dif_dist')
plt.xticks(rotation=90)
plt.show()


# In[344]:


filter_difs_perc = df_merged[(df_merged.perc_dif_dist > -100) & (df_merged.dif_valorKM_band1 < 20)]
plt.scatter(filter_difs_perc['perc_dif_dist'], filter_difs_perc['dif_valorKM_band1'])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('perc_dif_dist')
plt.show()


# In[345]:


# Cálculo da duração da corrida


# In[346]:


df_merged['tempo_corrida'] = pd.to_datetime(df_merged['data_final']) -
pd.to_datetime(df_merged['data_inicio'])
t1 = pd.to_datetime(df_merged['data_final'])
t2 = pd.to_datetime(df_merged['data_inicio'])
df_merged['minutos_corrida'] = (df_merged['tempo_corrida']).astype('timedelta64[m]')
```

```
# In[347]:


df_merged.describe()


# In[348]:


# Os valores vazios de tempo_corrida e minutos_corrida são por falta de alguma informação, então eles serão preenchidos
# por zero para mostrar que a informação está faltando


# In[349]:


df_merged['tempo_corrida'].fillna(0.0, inplace=True)
df_merged['minutos_corrida'].fillna(0.0, inplace=True)


# In[350]:


df_merged.describe()


# In[351]:


# Avaliar se as corridas de maior km, maior duração e de maior valor têm R$/km menor


# In[352]:


plt.scatter(df_merged['km_total'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('km total')
plt.show()
```

```
# In[353]:


plt.scatter(df_merged['valor_corrida'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('valor corrida')
plt.show()


# In[354]:


plt.scatter(df_merged['minutos_corrida'], df_merged['valor_por_km_desc_band'])
plt.ylabel('valor por km')
plt.xlabel('minutos corrida')
plt.show()


# In[355]:


# Gráfico para avaliar a relação entre os valores de km_total, valor_corrida e minutos_corrida


# In[356]:


fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = plt.axes(projection='3d')
xline = df_merged['km_total']
yline = df_merged['valor_corrida']
zline = df_merged['minutos_corrida']
ax.scatter3D(xline, yline, zline)
ax.set_xlabel('km_total')
ax.set_ylabel('valor_corrida')
ax.set_zlabel('minutos_corrida')
```

In[357]:

Gráfico de minutos_corrida por valor_corrida

In[358]:

```
plt.scatter(df_merged['minutos_corrida'], df_merged['valor_corrida'])
plt.ylabel('valor corrida')
plt.xlabel('minutos corrida')
plt.show()
```

In[359]:

Gráfico de minutos_corrida por km_total

In[360]:

```
plt.scatter(df_merged['minutos_corrida'], df_merged['km_total'])
plt.ylabel('km total')
plt.xlabel('minutos corrida')
plt.show()
```

In[361]:

Gráfico de km_total por valor_corrida

In[362]:

```
plt.scatter(df_merged['km_total'], df_merged['valor_corrida'])
plt.ylabel('valor corrida')
plt.xlabel('km total')
plt.show()


# In[365]:


# Aplicação do algoritmo K-means nos dados obtidos pelas colunas dif_distancias e dif_valorKM_band1


# In[366]:


# Primeiramente, os dados serão colocados em escalas mais próximas com RobustScaler


# In[367]:


from sklearn.preprocessing import RobustScaler


# In[368]:


filter_difs_rs = df_merged[(df_merged.dif_distancias > -100) & (df_merged.dif_distancias < 200)
                           & (df_merged.dif_valorKM_band1 < 20)]


# In[369]:


array_difs_rs = filter_difs_rs[['dif_distancias', 'dif_valorKM_band1']].to_numpy()


# In[370]:
```

```
transformer = RobustScaler().fit(array_difs_rs)

# In[371]:

transform = transformer.transform(array_difs_rs)

# In[372]:

plt.scatter(transform[:,0], transform[:,1])
plt.ylabel('dif_valorKM_band1')
plt.xlabel('dif_distancias')

# In[373]:

# Aplicação do algoritmo K-means

# In[374]:

from sklearn.cluster import KMeans

# In[375]:

# Definição do nº de clusters pelo Método do Cotovelo

# In[376]:

Sum_of_squared_distances = []
```

```
K = range(1, 10)
for num_clusters in K:
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(transform)
    Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()


# In[377]:


# Aplicação do K-means com 4 clusters


# In[378]:


model_k_4 = KMeans(4)
model_k_4.fit(transform)
y_model_k_4 = model_k_difs_4.predict(transform)

plt.scatter(transform[:, 0], transform[:, 1], c=y_model_k_4, s=50, cmap='viridis')

centers = model_k_4.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')


# In[379]:


# Aplicação do K-means com 5 clusters


# In[380]:
```

```
model_k_5 = KMeans(5)
model_k_5.fit(transform)
y_model_k_5 = model_k_5.predict(transform)

plt.scatter(transform[:, 0], transform[:, 1], c=y_model_k_5, s=50, cmap='viridis')

centers = model_k_5.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')


# In[382]:


# Aplicação do algoritmo DBSCAN


# In[383]:


from sklearn.cluster import DBSCAN


# In[384]:


from sklearn.neighbors import NearestNeighbors


# In[385]:


neigh = NearestNeighbors(n_neighbors = 2)
nbrs = neigh.fit(transform)
distances, indices = nbrs.kneighbors(transform)
```

```
# In[386]:


distances = np.sort(distances, axis = 0)
distances = distances[:,1]
plt.plot(distances)
plt.ylim(top=0.6)


# In[387]:


model_db = DBSCAN(eps=0.2)
model_db.fit(transform)
plt.scatter(transform[:,0], transform[:,1], c=model_db.labels_)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')


# In[388]:


# Aplicação do algoritmo HDBSCAN


# In[389]:


import hdbscan


# In[390]:


clusters_sizes = []
num_outliers = []

for k in range (2,50):
 clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
 clusterer.fit(transform)
```

```
 clusters_sizes.append(len(set(clusterer.labels_)))
 num_outliers.append(sum(clusterer.labels_ == -1))


# In[391]:


plt.style.use("fivethirtyeight")
plt.figure(figsize=(12,6))
plt.plot(range(2, 50), clusters_sizes, marker='o')
plt.xticks(range(2, 50))
plt.xlabel("min_cluster_size")
plt.ylabel("Número de Clusters")
plt.xlim(0,20)
plt.show()


# In[392]:


plt.style.use("fivethirtyeight")
plt.figure(figsize=(12,6))
plt.plot(range(2, 50), num_outliers, marker='o')
plt.xticks(range(2, 50))
plt.xlabel("min_cluster_size")
plt.ylabel("Quantidade de Outliers")
plt.xlim(0,20)
plt.show()


# In[393]:


model_hdb = hdbscan.HDBSCAN(min_cluster_size=16)
model_hdb.fit(transform)
plt.scatter(transform[:,0], transform[:,1], c=model_hdb.labels_)
plt.xlabel('dif_distancias')
plt.ylabel('dif_valorKM_band1')
```

```
# In[394]:


# Análise dos resultados


# In[395]:


# Criar df com outliers (considerando o resultado com HDBSCAN)


# In[396]:


# Criar coluna HDBSCAN_labels


# In[397]:


df_merged['HDBSCAN_labels'] = 0.0


# In[398]:


# Os valores discrepantes que foram retirados do array na análise já são claramente outliers
# Então esses registros serão preenchidos por -1


# In[399]:


filter_outliers = df_merged[(df_merged.dif_distancias < -100) | (df_merged.dif_distancias > 200)
                            | (df_merged.dif_valorKM_band1 > 20)]


# In[400]:
```

```
filter_outliers


# In[401]:


df_merged['HDBSCAN_labels'] = np.where((df_merged.dif_distancias < -100) |
(df_merged.dif_distancias > 200)
                | (df_merged.dif_valorKM_band1 > 20), -(1.0), df_merged['HDBSCAN_labels'])


# In[402]:


df_merged['HDBSCAN_labels'].unique()


# In[403]:


count = df_merged[df_merged.HDBSCAN_labels == -1.0].HDBSCAN_labels.value_counts()
count


# In[404]:


filter_not_outliers = df_merged[(df_merged.dif_distancias > -100) & (df_merged.dif_distancias < 200)
                & (df_merged.dif_valorKM_band1 < 20)]


# In[405]:


filter_not_outliers


# In[406]:
```

```
label_idx = 0
for i in range(2837):
    if (df_merged.loc[i, 'dif_distancias'] > -100) & (df_merged.loc[i, 'dif_distancias'] < 200) &
    (df_merged.loc[i, 'dif_valorKM_band1'] < 20):
        df_merged.loc[i, 'HDBSCAN_labels'] = model_hdb.labels_[label_idx]
        label_idx = label_idx + 1


# In[407]:


df_merged.HDBSCAN_labels.value_counts()


# In[408]:


df_outliers = df_merged[(df_merged.HDBSCAN_labels == -(1.0))]
df_outliers


# In[409]:


df_outliers.status_corrida.value_counts()


# In[410]:


df_outliers.base_origem.value_counts()


# In[411]:


orgaos_out = df_outliers.nome_orgao.value_counts()
orgaos_out
```

In[412]:

```
motivos_out = df_outliers.motivo_corrida.value_counts()
motivos_out
```

In[413]:

```
# Analisar alguns gráficos feitos inicialmente, mas agora com o resultado do modelo
```

In[414]:

```
plt.scatter(filter_difs_rs['dist_coords'], filter_difs_rs['km_total'], c=model_hdb.labels_)
plt.xlabel('dist_coords')
plt.ylabel('km_total')
# pontos fora da reta como outliers
```

In[415]:

```
# Análise dos gráficos abaixo:
# Pontos muito discrepantes foram realmente detectados como outliers. Mas também mostra que pontos dentro da reta,
# que à primeira vista não seriam vistos como outliers, podem sim apresentar alguma inconformidade e devem ser verificados.
```

In[416]:

```
plt.scatter(df_merged['minutos_corrida'], df_merged['valor_corrida'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('minutos_corrida')
plt.ylabel('valor_corrida')
```

```
# In[417]:


plt.scatter(df_merged['minutos_corrida'], df_merged['km_total'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('minutos_corrida')
plt.ylabel('km_total')


# In[418]:


plt.scatter(df_merged['km_total'], df_merged['valor_corrida'], c=df_merged['HDBSCAN_labels'])
plt.xlabel('km_total')
plt.ylabel('valor_corrida')


# In[266]:


# Exportar df_merged para excel


# In[267]:


import openpyxl


# In[268]:


df_merged.to_excel('df_merged_final.xlsx', index=False)
```