

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

Victor Hugo Negrisoli

**ANÁLISE SUPERVISIONADA DA ACEITAÇÃO DO USO DE LINGUAGENS DE
PROGRAMAÇÃO NOS ÚLTIMOS ANOS COM FOCO EM LINGUAGENS
INTERPRETADAS**

Belo Horizonte

2022

Victor Hugo Negrisoli

**ANÁLISE SUPERVISIONADA DA ACEITAÇÃO DO USO DE LINGUAGENS DE
PROGRAMAÇÃO NOS ÚLTIMOS ANOS COM FOCO EM LINGUAGENS
INTERPRETADAS**

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.....	5
1.1. Contextualização	5
1.2. O problema proposto.....	6
2. Coleta de Dados.....	8
2.1. Dataset do Github 2004 a 2021	8
2.2. Dataset do Github com dados de 2017	9
2.3. Dataset do Stackoverflow com dados de 2008 a 2021.....	10
3. Processamento/Tratamento de Dados	12
3.1. Tratamento dos dados do Github de 2004 a 2021.....	12
3.2. Tratamento dos dados do Github de 2017	17
3.3. Tratando os dados do StackOverflow	20
3.4. Integração do dataset do GitHub com o dataset do StackOverflow	24
4. Análise e Exploração dos Dados	26
4.1. Análise exploratória dos dados do Github de 2004 a 2021	26
4.2. Análise exploratória dos dados do Github de 2017	39
4.3. Análise exploratória dos dados do StackOverflow de 2008 a 2021	46
5. Criação de Modelos de Machine Learning	54
5.1. Criando os modelos para os dados do GitHub de 2004 a 2021.....	54
5.2. Criando o modelo para os dados do GitHub de 2017.....	59
5.3. Criando o modelo para os dados do StackOverflow de 2008 a 2021	61
5.4. Criando os modelos para o dataset integrado entre GitHub e StackOverflow	64
5.4.1. Regressão Linear.....	64
5.4.2. KNN	66
5.4.3. Naive Bayes	69
5.4.4. SVM.....	71
5.4.5. Prevendo os modelos com e sem normalização	73
6. Apresentação dos Resultados	81
7. Links	91
REFERÊNCIAS.....	92

APÊNDICE.....93

1. Introdução

1.1. Contextualização

Nos últimos 15 anos, as linguagens de programação foram passando por um grande processo de modernização, várias das grandes linguagens utilizadas frequentemente nos tempos atuais foram criadas em meados dos anos 80 e 90, e acabaram por ter uma popularização e aceitação nos últimos anos.

Muitas das linguagens criadas inicialmente eram linguagens compiladas, e, geralmente com forte tipagem, como podemos ver com Java, C++, C, C#, entre outros. Essas linguagens foram desenvolvidas entre os anos 80 e 90, exceto a linguagem C, desenvolvida nos anos 70. O que muitas dessas linguagens possuem em comum é que começaram a ser extremamente utilizadas entre os anos de 1995 a 2010, um período que passou por uma grande evolução na indústria de software.

Com a chegada da internet e com a maior disponibilização do acesso aos usuários comuns a outros meios, o número de profissionais nas áreas de desenvolvimento de software começa a aumentar, abrangendo novas tecnologias criadas a partir destas linguagens, os chamados frameworks, bibliotecas, e até mesmo stacks completas de desenvolvimento.

Em meados de 2010, outros tipos de linguagens de programação começaram a se popularizar, as linguagens interpretadas de script, ou seja, que compilam em tempo de execução. Essas linguagens englobam alguns gigantes da atualidade, como Python, Javascript, Ruby, R, PHP, Lua, Elixir, entre outros. Algumas linguagens interpretadas de script já eram bastante conhecidas e tiveram uma queda no uso, como é o caso de Ruby e PHP, outras como Javascript e Python tornam-se cada vez mais populares, seja no desenvolvimento web nas áreas de front-end, back-end, quanto na área de ciência de dados.

Um grande fator que leva à preferência ao uso de linguagens interpretadas por script é por sua curva de aprendizado mais acelerada e facilidade com configuração de ambiente e início de desenvolvimento. Em partes, isso se dá por geralmente serem linguagens com tipagem dinâmica, ou seja, tipagem fraca, e com IDEs mais acessíveis e rápidas de configuração e uso, sendo possível em instantes iniciar um ambiente de desenvolvimento

seja para um projeto em produção, quanto em cursos ou projetos pequenos que demandam menor tempo. Essas linguagens também consomem menos recursos computacionais.

O fato de linguagens de programação compiladas e com tipagem forte possuírem uma maior curva de aprendizado e consumirem muito mais recursos como processamento e memória fez com que houvesse uma queda no uso e aceitação pela comunidade tecnológica nos últimos 10 anos para alguns cenários específicos, como por exemplo desenvolvimento de interfaces gráficas e experiência de usuário em aplicações web modernas e mobile, assim como processamento de dados para análises estatísticas, não devendo ser confundido com os objetivos de processamento e armazenamento de dados, aos quais linguagens compiladas e com forte tipagem atuam com grande maestria.

1.2. O problema proposto

Serão utilizadas técnicas de análise exploratória de dados e algoritmos de Machine Learning supervisionados para encontrar padrões e validar se nos últimos anos linguagens com forte tipagem tiveram queda de uso com a grande evolução de pequenas aplicações e outras demandas, e, consequentemente observação da queda na preferência por linguagens com forte tipagem e compiladas, elencando pontos explicados no tópico acima sobre questões de curva de aprendizado, complexidade e poder computacional.

A maior importância em entender o aumento de um tipo de linguagens e a queda de outro é para se confirmar se há de fato um aumento na velocidade de entrega e aprendizado por parte de novos desenvolvedores, assim como empresas, o que auxilia na escolha e determinação de uma tecnologia para um problema proposto.

A escolha de uma determinada tecnologia para a solução de um problema é algo crucial para o negócio, impactando diretamente em seu lucro e principalmente em seu custo, pois uma tecnologia incorretamente escolhida pode trazer problemas de performance, o que pode ocasionar a maior consumo desnecessário de recursos de nuvem, servidores, entre outros fatores.

Serão analisados três conjuntos de dados, dois de usuários do Github e um de usuários da plataforma Stackoverflow, sendo essas fontes interessantes para entender o que os usuários estão criando e discutindo.

Dois dos conjuntos analisados serão retirados do Github, um contendo dados sobre uso de várias tecnologias desde 2004 a 2021, e o outro é de uma pesquisa realizada em 2017 sobre a frequência em que as tecnologias em destaque são mencionadas em seus repositórios, podendo analisar se há uma correlação entre essas tecnologias analisadas apenas em 2017 e se o resultado bate com o que foi analisado anualmente de 2004 a 2021. O terceiro conjunto de dados será extraído diretamente da plataforma do Stackoverflow, na ferramenta insights, que permite aos usuários realizar consultas com linguagem SQL para extrair dados das discussões na plataforma, e essa extração será desde 2008 até o final de 2021.

Será analisado se o uso de algumas tecnologias pelos usuários do GitHub está relacionado às perguntas e tópicos no StackOverflow e se esse percentual está relacionado ao aumento das linguagens de script e o declínio das linguagens compiladas e com tipagem forte.

Para as análises, serão utilizadas técnicas de correlação entre as variáveis no conjunto de dados, algoritmos de Regressão Linear, e outros algoritmos supervisionados como K-Nearest Neighbors (KNN), Support Vector Machine (SVM) e Naive-Bayes. Será também realizado um comparativo para analisar qual algoritmo se sai melhor para a análise das linguagens.

2. Coleta de Dados

2.1. Dataset do Github 2004 a 2021

Os conjuntos de dados extraídos da plataforma do Github encontram-se na plataforma Kaggle. O primeiro, chamado “Most Popular Programming Languages from 2004 to 2021 V4.csv”, estará disponível em:

<https://www.kaggle.com/muhammadkhalid/most-popular-programming-languages-since-2004>, contendo 29 colunas e 203 linhas, podendo descrever as colunas com as seguintes informações:

Nome da coluna/campo	Descrição	Tipo
Date	Data sendo analisada no formato MMM YYYY. Ex: “September 2004”	object
Abap	Percentual de uso da linguagem Abap.	float64
Ada	Percentual de uso da linguagem Ada.	float64
C/C++	Percentual de uso da linguagem C/C++.	float64
C#	Percentual de uso da linguagem C#.	float64
Cobol	Percentual de uso da linguagem Cobol.	float64
Dart	Percentual de uso da linguagem Dart.	float64
Delphi	Percentual de uso da linguagem Delphi.	float64
Go	Percentual de uso da linguagem Go.	float64
Groovy	Percentual de uso da linguagem Groovy.	float64
Haskell	Percentual de uso da linguagem Haskell.	float64
Java	Percentual de uso da linguagem Java.	float64
JavaScript	Percentual de uso da linguagem JavaScript.	float64
Julia	Percentual de uso da linguagem Julia.	float64
Kotlin	Percentual de uso da linguagem Kotlin.	float64
Lua	Percentual de uso da linguagem Lua.	float64
Matlab	Percentual de uso da linguagem Matlab.	float64
Objective-C	Percentual de uso da linguagem Objective-	float64

	C.	
Perl	Percentual de uso da linguagem Perl.	float64
PHP	Percentual de uso da linguagem PHP.	float64
Python	Percentual de uso da linguagem Python.	float64
R	Percentual de uso da linguagem R.	float64
Ruby	Percentual de uso da linguagem Ruby.	float64
Rust	Percentual de uso da linguagem Rust.	float64
Scala	Percentual de uso da linguagem Scala.	float64
Swift	Percentual de uso da linguagem Swift.	float64
TypeScript	Percentual de uso da linguagem TypeScript.	float64
VBA	Percentual de uso da linguagem VBA.	float64

2.2. Dataset do Github com dados de 2017

O terceiro conjunto de dados encontra-se disponível em: <https://www.kaggle.com/jaimevalero/developers-and-programming-languages>, e contém mais de 1400 colunas envolvendo várias tecnologias, porém, serão utilizadas apenas 8 colunas, com um total de 17.461 linhas.

A estrutura analisada será:

Nome da coluna/campo	Descrição	Tipo
user_id	Campo que representa o nome do usuário do github utilizando uma determinada tecnologia.	object
java	Frequência de uso da label java em seus repositórios.	float64
javascript	Frequência de uso da label javascript em seus repositórios.	float64
typescript	Frequência de uso da label typescript em seus repositórios.	float64
php	Frequência de uso da label php em seus repositórios.	float64
python	Frequência de uso da label python em seus repositórios.	float64

	repositórios.	
c#	Frequência de uso da label c# em seus repositórios.	float64
go	Frequência de uso da label go em seus repositórios.	float64

2.3. Dataset do Stackoverflow com dados de 2008 a 2021

O último conjunto de dados analisado será extraído diretamente do Stackoverflow através do seguinte endereço <https://data.stackexchange.com/stackoverflow/query/new>, sendo possível escrever uma consulta SQL para buscar os dados do servidor da plataforma e exportá-los em formato .csv.

Os campos que serão utilizados na consulta serão ano, mês, nome da tag e quantidade de perguntas. A consulta utilizada na extração do conjunto de dados foi estruturada conforme o SQL apresentado abaixo:

```

SELECT
    YEAR(Posts.CreationDate) as 'Year',
    MONTH(Posts.CreationDate) as 'Month',
    Tags.tagName,
    COUNT(*) AS Question
FROM Tags
    LEFT JOIN PostTags ON PostTags.TagId = Tags.Id
    LEFT JOIN Posts ON Posts.Id = PostTags.PostId
WHERE
    Tags.tagName IN (
        'java',
        'javascript',
        'python',
        'go',
        'c#',
        'c',
        'c++',
        'php',
        'r',
    )

```

```

'typescript'

)

AND Posts.CreationDate <= '2021-12-31'

GROUP BY

YEAR(Posts.CreationDate), MONTH(Posts.CreationDate), Tags.TagName

ORDER BY

YEAR(Posts.CreationDate), MONTH(Posts.CreationDate) DESC

```

Através da consulta apresentada acima, é possível obter um conjunto de dados contendo 4 colunas e 1545 linhas, estando estruturado no seguinte formato:

Nome da coluna/campo	Descrição	Tipo
Year	Ano da data apresentada.	int64
Month	Mês da data apresentada.	int64
tagName	Nome da tag (tecnologia) que foi pesquisada.	object
Question	Quantidade de questões realizadas sobre a tag.	int64

3. Processamento/Tratamento de Dados

3.1. Tratamento dos dados do Github de 2004 a 2021

Todo o processamento, tratamento e análise dos dados será totalmente dentro da ferramenta Jupyter Notebook na versão 6.0.1, com todos os scripts escritos com a linguagem Python na versão 3.7.4. Os arquivos analisados serão via Comma-Separated Values, ou CSV.

Para iniciar o processo de tratamento de dados, serão feitas importações de todas as bibliotecas que serão utilizadas no projeto, e algumas funções de configuração de plotagens para análises posteriores. As duas principais bibliotecas para análise, manipulação e tratamento dos dados serão o Pandas e o NumPy.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime

from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn import svm
```

A Matplotlib e a Seaborn serão as principais bibliotecas para visualização dos dados, realizando as plotagens das análises dos resultados. A biblioteca Scikit-Learn terá um papel bastante importante em toda a análise, pois será a principal biblioteca de processamento dos modelos de Machine Learning, relatórios e score de classificação, treino e teste de modelo, entre outros.

Antes de iniciar a carga dos datasets, serão criadas duas funções de configuração de plotagens, chamadas, respectivamente, *configurar_plot_com_dimensoes*, que recebe 5 argumentos: título, valor de label do eixo x, valor de label do eixo y, altura e largura da imagem. A segunda função, *configurar_plot*, é bem parecida, porém, sem informar as dimensões, apenas informando título e valores dos eixos x e y e mantendo altura em 16 e largura em 8.

```

def configurar_plot_com_dimensoes(titulo, x, y, h, w):
    plt.title(titulo)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.gcf().set_size_inches(h, w)
    plt.show()

def configurar_plot(titulo, x, y):
    configurar_plot_com_dimensoes(titulo, x, y, 16, 8)

```

Logo em seguida, será carregado o primeiro dataset, o “Most Popular Programming Languages from 2004 to 2021 V4.csv”, e serão mostradas as primeiras 10 linhas do dataset. O dataset é carregado com uma variável chamada *df*.

```

df = pd.read_csv('dados/Most Popular Programming Languages from 2004 to 2021 V4.csv')
df.head(10)

```

	Date	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	...	PHP	Python	R	Ruby	Rust	Scala	Swift	TypeScript	VBA	Visual Basic
0	July 2004	0.34	0.36	10.08	4.71	0.43	0.0	2.82	0.0	0.03	...	18.75	2.53	0.39	0.33	0.08	0.03	0.0	0.0	1.44	8.56
1	August 2004	0.36	0.36	9.81	4.99	0.46	0.0	2.67	0.0	0.07	...	19.26	2.64	0.41	0.40	0.09	0.03	0.0	0.0	1.46	8.57
2	September 2004	0.41	0.41	9.63	5.06	0.51	0.0	2.65	0.0	0.08	...	19.49	2.72	0.40	0.41	0.10	0.03	0.0	0.0	1.55	8.41
3	October 2004	0.40	0.38	9.50	5.31	0.53	0.0	2.77	0.0	0.09	...	19.34	2.92	0.42	0.46	0.11	0.04	0.0	0.0	1.61	8.49
4	November 2004	0.38	0.38	9.52	5.24	0.55	0.0	2.76	0.0	0.07	...	19.43	2.84	0.41	0.45	0.13	0.04	0.0	0.0	1.50	8.24
5	December 2004	0.36	0.37	9.56	5.23	0.53	0.0	2.77	0.0	0.09	...	19.73	2.71	0.40	0.42	0.13	0.04	0.0	0.0	1.46	8.08
6	January 2005	0.39	0.38	9.70	5.23	0.56	0.0	2.65	0.0	0.11	...	19.81	2.91	0.39	0.47	0.15	0.03	0.0	0.0	1.51	7.79
7	February 2005	0.37	0.39	9.88	5.21	0.49	0.0	2.66	0.0	0.07	...	19.63	2.87	0.38	0.45	0.15	0.03	0.0	0.0	1.45	7.67
8	March 2005	0.34	0.37	9.88	5.38	0.45	0.0	2.65	0.0	0.08	...	19.54	2.81	0.42	0.46	0.13	0.03	0.0	0.0	1.44	7.68
9	April 2005	0.34	0.36	9.85	5.42	0.41	0.0	2.56	0.0	0.08	...	19.93	2.78	0.40	0.43	0.11	0.02	0.0	0.0	1.36	7.52

10 rows × 29 columns

Logo abaixo, serão utilizados os métodos *shape* e *info()*. O *shape* irá informar como está a matriz do DataFrame e o *info()* irá nos dar todas as colunas, seus respectivos tipos de dados e se possuem valores nulos e totais presentes nas colunas. É possível visualizar através da imagem abaixo que este dataset não possui valores nulos, e que todas as 203 linhas estão sendo preenchidas.

```
df.shape  
(203, 29)  
  
df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 203 entries, 0 to 202  
Data columns (total 29 columns):  
 Date          203 non-null object  
 Abap          203 non-null float64  
 Ada           203 non-null float64  
 C/C++         203 non-null float64  
 C#            203 non-null float64  
 Cobol         203 non-null float64  
 Dart          203 non-null float64  
 Delphi        203 non-null float64  
 Go             203 non-null float64  
 Groovy        203 non-null float64  
 Haskell        203 non-null float64  
 Java           203 non-null float64  
 JavaScript     203 non-null float64  
 Julia          203 non-null float64  
 Kotlin         203 non-null float64  
 Lua             203 non-null float64  
 Matlab          203 non-null float64  
 Objective-C    203 non-null float64  
 Perl            203 non-null float64  
 PHP             203 non-null float64  
 Python          203 non-null float64  
 R               203 non-null float64  
 Ruby            203 non-null float64  
 Rust            203 non-null float64  
 Scala           203 non-null float64  
 Swift           203 non-null float64  
 TypeScript      203 non-null float64  
 VBA             203 non-null float64  
 Visual Basic    203 non-null float64  
 dtypes: float64(28), object(1)  
 memory usage: 46.1+ KB
```

Logo abaixo, para este mesmo dataset será realizada a chamada ao método *describe()* do Pandas, que irá descrever dados estatísticos para todos os valores numéricos do dataset, como os valores mínimos, máximos, médios, desvio padrão, 25, 50 e 75% dos valores, e uma contagem de todas as linhas. Como não há valores nulos, é possível verificar que todos ficarão travados em 203 na primeira linha.

Na linha abaixo, serão realizadas duas operações para verificar quais meses e anos estarão presentes no campo “Date” do dataset. Como pode ser visto na imagem abaixo, teremos os meses de Janeiro a Dezembro, nos anos de 2004 a 2021.

```
df.describe()

      Abap     Ada   C/C++      C#    Cobol    Dart   Delphi     Go   Groovy  Haskell ...    PHP    Python
count 203.000000 203.000000 203.000000 203.000000 203.000000 203.000000 203.000000 203.000000 203.000000 ... 203.000000 203.000000
mean  0.498030  0.311478  8.333251  7.603202  0.381034  0.121576  1.055172  0.322906  0.334089  0.298621 ... 14.152020 11.917882
std   0.102097  0.110269  1.807976  1.306233  0.071528  0.149658  0.757032  0.451741  0.137936  0.039728 ... 4.983405  8.768295
min   0.320000  0.170000  5.610000  4.710000  0.250000  0.000000  0.230000  0.000000  0.030000  0.200000 ... 5.810000  2.530000
25%   0.400000  0.230000  7.395000  6.575000  0.320000  0.000000  0.370000  0.000000  0.260000  0.280000 ... 9.990000  5.435000
50%   0.510000  0.300000  8.080000  7.330000  0.370000  0.110000  0.810000  0.080000  0.380000  0.300000 ... 14.500000 8.440000
75%   0.580000  0.360000  8.940000  8.935000  0.430000  0.135000  1.680000  0.505000  0.450000  0.320000 ... 19.300000 15.925000
max   0.720000  0.770000 13.000000 10.000000  0.560000  0.580000  2.820000  1.460000  0.520000  0.390000 ... 20.840000 32.110000

8 rows × 28 columns
< >
meses = sorted(list(set([i.split(' ')[0] for i in df['Date'].unique()])))
anos = sorted(list(set([i.split(' ')[1] for i in df['Date'].unique()])))
print('Meses: {}\nAnos: {}'.format(meses, anos))

Meses: ['April', 'August', 'December', 'February', 'January', 'July', 'June', 'March', 'May', 'November', 'October', 'September']
Anos: ['2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021']
```

Será gerado um novo dataset a partir de uma reestruturação do dataset apresentado anteriormente. A transformação será feita através de uma função chamada *createDataFrameFor* informando por parâmetro um DataFrame, um array de colunas, e qual a coluna atual desejada para a criação. Esse valor numérico da coluna atual é para indicar a coluna a partir da data, pois o objetivo desse DataFrame será transpor os dados e manter todas as linguagens de programação em apenas uma coluna, e apenas agrupar seus valores.

```
def createDataFrameFor(df, colunas, colunaAtual):
    return pd.DataFrame(
        [
            'Date': df.Date,
            'Year': pd.DatetimeIndex(df['Date']).year,
            'Timestamp': map(lambda i: datetime.strptime(df["Date"][i], '%B %Y'), range(len(df.Date))),
            'Language': colunas[colunaAtual],
            'Value': df[df.columns[colunaAtual]]
        ]
    )

colunas = df.columns

dados_tratados = createDataFrameFor(df, colunas, 1)

for coluna in range(1, len(colunas)):
    dados_tratados = pd.concat([dados_tratados, createDataFrameFor(df, colunas, coluna)])

dados_tratados.reset_index(drop=True, inplace=True)

dados_tratados['UnixTime'] = list(map(lambda i: \
    (pd.to_datetime([dados_tratados['Timestamp'][i]]).astype(int) / 10**9)[0], \
    range(len(dados_tratados['Date']))))

dados_tratados.head()
```

	Date	Year	Timestamp	Language	Value	UnixTime
0	July 2004	2004	2004-07-01	Abap	0.34	1.088640e+09
1	August 2004	2004	2004-08-01	Abap	0.36	1.091318e+09
2	September 2004	2004	2004-09-01	Abap	0.41	1.093997e+09
3	October 2004	2004	2004-10-01	Abap	0.40	1.096589e+09
4	November 2004	2004	2004-11-01	Abap	0.38	1.099267e+09

Em seguida, um laço do tipo *for* cria o restante das colunas do DataFrame, e concatena ao DataFrame já existente, e, por fim, o índice é resetado, não será necessário para as

análises abaixo, e é definida mais uma coluna de UnixTime, para que seja possível obter o valor numérico do Timestamp da data para análises de séries temporais e análises de regressão. Por fim, serão printadas as 10 primeiras linhas do novo dataset gerado.

```
dados_tratados.to_csv('dados/Dados.csv')
```

Por fim, esse dataset será salvo com o nome de “Dados.csv” na máquina local, para garantir a segurança e atualização quando necessário. Por fim, é possível verificar algumas estatísticas do resultado final do dataset gerado, como ficaram suas novas dimensões, valores não-nulos, entre outros. O resultado final foi um dataset com 6 colunas e 5887 linhas.

```
dados_tratados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5887 entries, 0 to 5886
Data columns (total 6 columns):
Date      5887 non-null object
Year       5887 non-null int64
Timestamp  5887 non-null datetime64[ns]
Language   5887 non-null object
Value      5887 non-null float64
UnixTime   5887 non-null float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(2)
memory usage: 276.1+ KB
```

```
dados_tratados.shape
```

```
(5887, 6)
```

```
dados_tratados.describe()
```

	Year	Value	UnixTime
count	5887.000000	5887.000000	5.887000e+03
mean	2012.458128	3.465436	1.354277e+09
std	4.900599	6.019926	1.541191e+08
min	2004.000000	0.000000	1.088640e+09
25%	2008.000000	0.280000	1.220227e+09
50%	2012.000000	0.670000	1.354320e+09
75%	2017.000000	3.830000	1.488326e+09
max	2021.000000	32.110000	1.619827e+09

A última transformação será a criação de um novo DataFrame a partir deste, porém, utilizando a função *groupby* do Pandas para realização de algumas análises em ranking, para isso, o DataFrame *dados_tratados* terá apenas as colunas Language e Value, e será agrupado pela coluna Language, tendo todos os valores numéricos do agrupamento sendo somados.

Logo em seguida, esses valores serão ordenados pelo valor em ordem decrescente com a função `sort_values` do Pandas.

```
dados_agrupados = dados_tratados[['Language', 'Value']].groupby(by=['Language'], as_index=False).sum()
dados_agrupados = dados_agrupados.sort_values(by=['Value'], ascending=False)
dados_agrupados
```

	Language	Value
10	Java	5265.13
17	PHP	2872.86
19	Python	2419.33
3	C/C++	1691.65
11	JavaScript	1598.16
2	C#	1543.45
27	Visual Basic	754.92
16	Objective-C	546.40
18	Perl	543.92
15	Matlab	531.17
20	R	430.32
21	Ruby	428.81
26	VBA	340.84
24	Swift	225.21
6	Delphi	214.20
0	Abap	202.20
23	Scala	109.39
25	TypeScript	95.57

3.2. Tratamento dos dados do Github de 2017

O terceiro dataset a ser carregado no projeto é o arquivo csv chamado “`user-languages.csv`”, e a forma de realizar a carga dos dados é parecida com a maneira que foi anteriormente mostrada para o dataset temporal, porém, com a diferença que logo na carga do arquivo já será especificado quais colunas serão desejadas para o objeto do tipo Data-Frame do Pandas, que, conforme dito no segundo capítulo, serão as colunas `user_id`, `java`, `javascript`, `typescript`, `php`, `python`, `c#`, `go`. Será realizado um print das primeiras 5 linhas do dataset com o método `head()` e em seguida serão visualizados os tipos de dados com o atributo `dtypes`.

```

dados_2017 = pd.read_csv('dados/user-languages.csv')\
    [['user_id', 'java', 'javascript', 'typescript', 'php', 'python', 'c#', 'go']]

dados_2017.head()

```

	user_id	java	javascript	typescript	php	python	c#	go
0	007lva	0.015326	0.049808	0.000000	0.0	0.010536	0.0000	0.005747
1	06wj	0.000000	0.327881	0.015613	0.0	0.028996	0.0171	0.000000
2	Observer07	0.000000	0.011375	0.000000	0.0	0.185109	0.0000	0.000000
3	Orca	0.018692	0.000000	0.000000	0.0	0.000000	0.0000	0.000000
4	0x00A	0.000000	0.268152	0.000000	0.0	0.000000	0.0000	0.000000

```

dados_2017.dtypes

```

user_id	object
java	float64
javascript	float64
typescript	float64
php	float64
python	float64
c#	float64
go	float64
dtype:	object

Em seguida, será realizada uma verificação dos tipos com os métodos *info()* e *describe()* para analisar a estrutura do dataset. É possível verificar que este também é um dataset sem valores nulos, e que contém 5 colunas e 17.461 linhas, sendo apenas a primeira coluna do tipo object de string, e o restante todos em float64.

```
dados_2017.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17461 entries, 0 to 17460
Data columns (total 8 columns):
user_id      17461 non-null object
java         17461 non-null float64
javascript   17461 non-null float64
typescript   17461 non-null float64
php          17461 non-null float64
python        17461 non-null float64
c#           17461 non-null float64
go           17461 non-null float64
dtypes: float64(7), object(1)
memory usage: 1.1+ MB
```

```
dados_2017.describe()
```

	java	javascript	typescript	php	python	c#	go
count	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000
mean	0.028533	0.108883	0.004245	0.017286	0.042778	0.007451	0.012977
std	0.076522	0.122673	0.021715	0.055088	0.088514	0.040868	0.043427
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.010870	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.067114	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.016990	0.174345	0.000000	0.001043	0.042476	0.000000	0.000000
max	1.000000	1.000000	0.666667	1.000000	1.000000	1.000000	1.000000

Para algumas análises a serem realizadas com este dataset, será necessário recriá-lo e reestruturá-lo de forma agrupada e convertendo as linguagens de programação que são colunas em apenas uma, transpondo em linhas. Essa reestruturação terá apenas duas colunas, Language, representando as linguagens, e Value, representando os percentuais somados. Para isso, o uso das funções *lambda* e *map* será muito útil para a estruturação durante a criação do DataFrame, como visto na imagem abaixo:

```
dft = pd.DataFrame(
    {
        'Language': list(map(lambda i: i, dados_2017.columns[1:])),
        'Value':    list(map(lambda i: dados_2017[i].sum(), dados_2017.columns[1:]))
    }
)

dft.head()
```

	Language	Value
0	java	498.222018
1	javascript	1901.207143
2	typescript	74.127341
3	php	301.829051
4	python	746.939461

Descrevendo então esse DataFrame, teremos a seguinte informação:

```
dft.describe()
```

	Value
count	7.000000
mean	554.144057
std	637.189883
min	74.127341
25%	178.341690
50%	301.829051
75%	622.580740
max	1901.207143

3.3. Tratando os dados do StackOverflow

Os dados serão extraídos do arquivo csv “*QueryResults.csv*”, resultado da extração diretamente da plataforma através do SQL apresentado no capítulo 2. Será feita a extração, um print das primeiras 5 linhas e uma descrição das informações do dataset:

```
df_so = pd.read_csv('dados/stack_overflow/QueryResults.csv')
df_so.head()
```

	Year	Month	tagName	Question
0	2008	12	c	189
1	2008	12	javascript	626
2	2008	12	c#	1595
3	2008	12	c++	632
4	2008	12	python	440

```
df_so.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1545 entries, 0 to 1544
Data columns (total 4 columns):
Year      1545 non-null int64
Month     1545 non-null int64
tagName   1545 non-null object
Question  1545 non-null int64
dtypes: int64(3), object(1)
memory usage: 48.4+ KB
```

E a descrição do dataset:

```
df_so.describe()
```

	Year	Month	Question
count	1545.000000	1545.000000	1545.000000
mean	2014.993528	6.604531	6932.346278
std	3.809071	3.464603	6340.638251
min	2008.000000	1.000000	1.000000
25%	2012.000000	4.000000	1908.000000
50%	2015.000000	7.000000	4586.000000
75%	2018.000000	10.000000	11484.000000
max	2021.000000	12.000000	29530.000000

Este dataset, da maneira que está apresentado já pode ser utilizado para análises, porém, é necessário que seja realizado um tratamento das colunas, pois posteriormente terá que ser integrado ao dataset do Github. Para isso, pode-se observar que no dataset do Github, a coluna “java” está como “Java”, a coluna c++ está como “C/C++”. Para o estudo em destaque, como o Github considera as linguagens C e C++ como única, teremos que realizar uma junção dessas colunas nesse dataset também. Para isso, foi criada a função *tratar_nome*, que recebe o nome da coluna como parâmetro:

```
def tratar_nome(nome):
    if (nome == 'java'):
        return 'Java'

    if (nome == 'javascript'):
        return 'JavaScript'

    if (nome == 'typescript'):
        return 'TypeScript'

    if (nome == 'python'):
        return 'Python'

    if (nome == 'php'):
        return 'PHP'

    if (nome == 'c'):
        return 'C/C++'

    if (nome == 'c++'):
        return 'C/C++'

    if (nome == 'c#'):
        return 'C#'

    if (nome == 'go'):
        return 'Go'

    if (nome == 'r'):
        return 'R'

    return ''
```

Com a função devidamente criada, agora pode ser criada a lógica para recriar o DataFrame informando essa função. Essa lógica será observada na imagem abaixo, acompanhada de um print das primeiras 5 linhas, mostrando o resultado obtido:

```

dados_tratados_so = pd.DataFrame(
    {
        'Year': df_so['Year'],
        'Month': df_so['Month'],
        'Language': list(map(lambda x: tratar_nome(x), df_so['tagName'])),
        'Value': df_so['Question']
    }
)

dados_tratados_so.head()
:
```

	Year	Month	Language	Value
0	2008	12	C/C++	189
1	2008	12	JavaScript	626
2	2008	12	C#	1595
3	2008	12	C/C++	632
4	2008	12	Python	440

É possível observar que o DataFrame se manteve estruturalmente, mudando apenas o nome da coluna tagName para Language e Question para Value, traduzindo para o contexto em que está sendo analisado.

O último tratamento a ser realizado para este dataset é uma transformação do dataset existente para uma estrutura parecida com os datasets do Github, contendo as categorias da coluna Language como colunas, ou seja, transpondo no formato: Year, Month, Java, Javascript, Python...

O objetivo para essa criação é para que seja possível realizar análise de correlação entre as variáveis. Esse DataFrame será chamado *correlation_df*. Para isso, primeiro foi criada uma função auxiliar que irá encontrar uma linha do DataFrame informando o ano, o mês e a linguagem. Caso não existe, será retornado 0. Essa função chama-se *find_by_date_and_language*.

```

def find_by_date_and_language(year, month, language):
    value = dados_tratados_so[
        (dados_tratados_so['Language'] == language)
        & (dados_tratados_so['Year'] == year)
        & (dados_tratados_so['Month'] == month)
    ]['Value']
    if (value.empty):
        return 0
    return value.values[0]

```

Logo em seguida, foi criada uma lógica para adicionar os dados de cada linguagem em dois loops de repetição do tipo *for*, e o resultado adicionado a um dicionário, e, dentro de cada iteração do mês e do ano, esse dicionário era adicionado em uma lista de dicionários. Ao fim da iteração dos loops, o dataset estará criado em formato de lista, e, por fim, será convertido em um DataFrame do Pandas.

```

for year in dados_tratados_so['Year'].unique():
    for month in dados_tratados_so['Month'].unique():
        data = {
            'Year': year,
            'Month': month,
            'JavaScript': find_by_date_and_language(year, month, 'JavaScript'),
            'TypeScript': find_by_date_and_language(year, month, 'TypeScript'),
            'Python': find_by_date_and_language(year, month, 'Python'),
            'Java': find_by_date_and_language(year, month, 'Java'),
            'C#': find_by_date_and_language(year, month, 'C#'),
            'PHP': find_by_date_and_language(year, month, 'PHP'),
            'C/C++': find_by_date_and_language(year, month, 'C/C++'),
            'R': find_by_date_and_language(year, month, 'R'),
            'Go': find_by_date_and_language(year, month, 'Go')
        }
        transposed_data.append(data)

correlation_df = pd.DataFrame(transposed_data, columns = [
    'Year',
    'Month',
    'JavaScript',
    'TypeScript',
    'Python',
    'Java',
    'C#',
    'PHP',
    'C/C++',
    'R',
    'Go'
])

```

Para finalizar, serão apresentadas as últimas 10 linhas do novo DataFrame criado.

	Year	Month	JavaScript	TypeScript	Python	Java	C#	PHP	C/C++	R	Go
158	2021	10	16313	3333	23076	7842	6077	4319	4103	4759	763
159	2021	9	16714	3237	21998	7817	5770	4326	3922	4332	662
160	2021	8	17058	3265	23142	7958	5764	4592	1622	4575	786
161	2021	7	17349	3163	24127	8340	5934	4618	3787	4677	650
162	2021	6	17621	3188	24532	9202	6419	4858	1996	4942	644
163	2021	5	18404	3129	26032	9675	6533	5065	2246	5393	698
164	2021	4	18602	3255	26455	9484	6737	5086	4315	5607	682
165	2021	3	19773	3530	28106	10393	7302	5556	4714	6076	823
166	2021	2	17911	2937	24928	9124	6586	5231	2349	5033	626
167	2021	1	19484	3044	26295	9531	6926	5622	4501	5032	604

3.4. Integração do dataset do GitHub com o dataset do StackOverflow

O último tratamento geral a ser realizado é a junção dos datasets do GitHub de 2004 a 2021 com os dados do StackOverflow de 2008 a 2021, tornando-os na mesma estrutura e contendo os valores numéricos de ambos.

Para esta transformação, primeiramente foi criada uma função chamada *extraír_mes* apenas para tratar o valor numeral do mês que está presente no dataset do GitHub, pois lá o mês é apenas uma descrição em inglês.

```
def extraír_mes_data(data):
    if ('January' in data):
        return 1
    if ('February' in data):
        return 2
    if ('March' in data):
        return 3
    if ('April' in data):
        return 4
    if ('May' in data):
        return 5
    if ('June' in data):
        return 6
    if ('July' in data):
        return 7
    if ('August' in data):
        return 8
    if ('September' in data):
        return 9
    if ('October' in data):
        return 10
    if ('November' in data):
        return 11
    if ('December' in data):
        return 12
```

Em seguida, o dataset *dados_tratados_novo* é criado, contendo os mesmos valores originais, porém, com a adição da coluna Month, e, posteriormente, é filtrado apenas pelas linguagens que também estão presentes no dataset do StackOverflow, que neste caso, são Java, Javascript, C/C++, TypeScript, Python, R, C#, PHP e Go.

E, por fim, será criado o dataset chamad *merge* através do uso da função *merge* da biblioteca Pandas. O *merge* foi utilizado com base no conceito de junção à esquerda (*left join*) nas colunas Year, Month e Language. O merge foi feito entre os datasets *dados_tratados_novo* e *dados_tratados_so*.

A coluna Value_x é referente aos dados do GitHub e Value_y aos do StackOverflow. O objetivo da criação deste dataset é para a criação dos modelos de Machine Learning entre as integrações deste dataset.

```

dados_tratados_novo = pd.DataFrame(
    {
        'Year': dados_tratados['Year'],
        'Month': list(map(lambda x: extrair_mes_data(x), dados_tratados['Date'])),
        'Language': dados_tratados['Language'],
        'Value': dados_tratados['Value']
    }
)

dados_tratados_novo = dados_tratados_novo[
    (dados_tratados_novo['Language'] == 'Java') |
    (dados_tratados_novo['Language'] == 'JavaScript') |
    (dados_tratados_novo['Language'] == 'TypeScript') |
    (dados_tratados_novo['Language'] == 'Python') |
    (dados_tratados_novo['Language'] == 'R') |
    (dados_tratados_novo['Language'] == 'C/C++)' |
    (dados_tratados_novo['Language'] == 'C#') |
    (dados_tratados_novo['Language'] == 'PHP') |
    (dados_tratados_novo['Language'] == 'Go')
]

merge = pd.merge(dados_tratados_novo, dados_tratados_so, how='left', on=['Year', 'Month', 'Language'])
merge = merge.dropna()
merge.head()

```

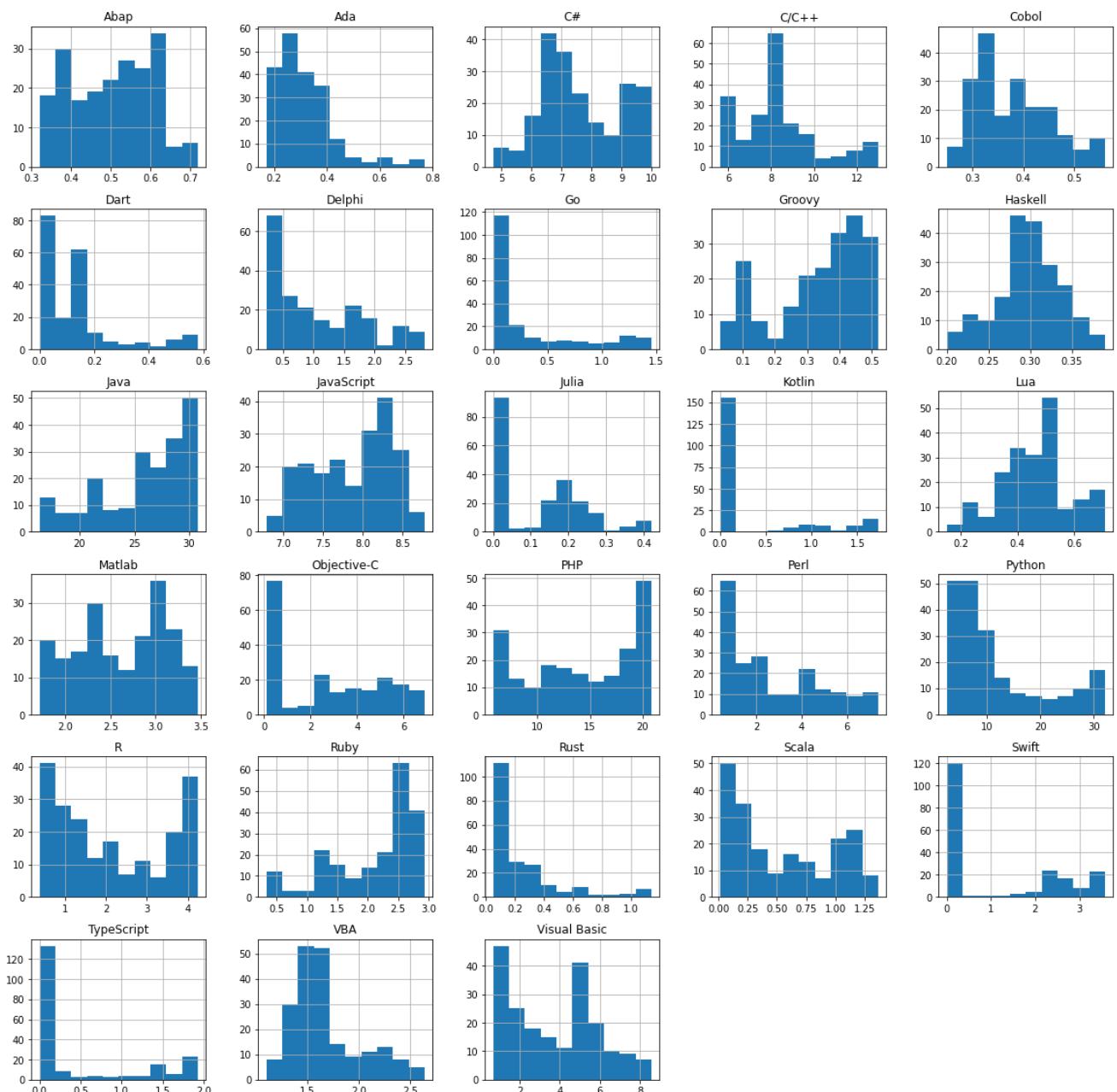
	Year	Month	Language	Value_x	Value_y
49	2008	8	C/C++	7.82	83.0
50	2008	8	C/C++	7.82	163.0
51	2008	9	C/C++	7.94	754.0
52	2008	9	C/C++	7.94	320.0
53	2008	10	C/C++	8.10	303.0

4. Análise e Exploração dos Dados

4.1. Análise exploratória dos dados do Github de 2004 a 2021

A primeira análise a ser feita no primeiro dataset tratado é o histograma para ver em quais momentos estão distribuídas as linguagens de programação analisadas.

```
df.hist()
configurar_plot_com_dimensões('Histograma Linguagens', '', '', 20, 20)
plt.show()
```



É possível visualizar que várias linguagens começam em valores diferentes, como Go, Groovy, Kotlin, Dart, entre outras. Ou seja, linguagens mais novas, pois foram quando começaram a ter valores devido à criação em seus respectivos anos, sendo a maioria a partir de 2005 ou 2010.

Nessa seção você deve mostrar como foi realizada a análise e exploração dos dados do seu trabalho. Mostre as hipóteses levantadas durante essa etapa e os padrões e *insights* identificados.

Através dos dados tratados, agrupados e ordenados, conseguimos ter um ranking e plotar um gráfico para visualizar as principais linguagens mais utilizadas no somatório de todos os anos até o momento.

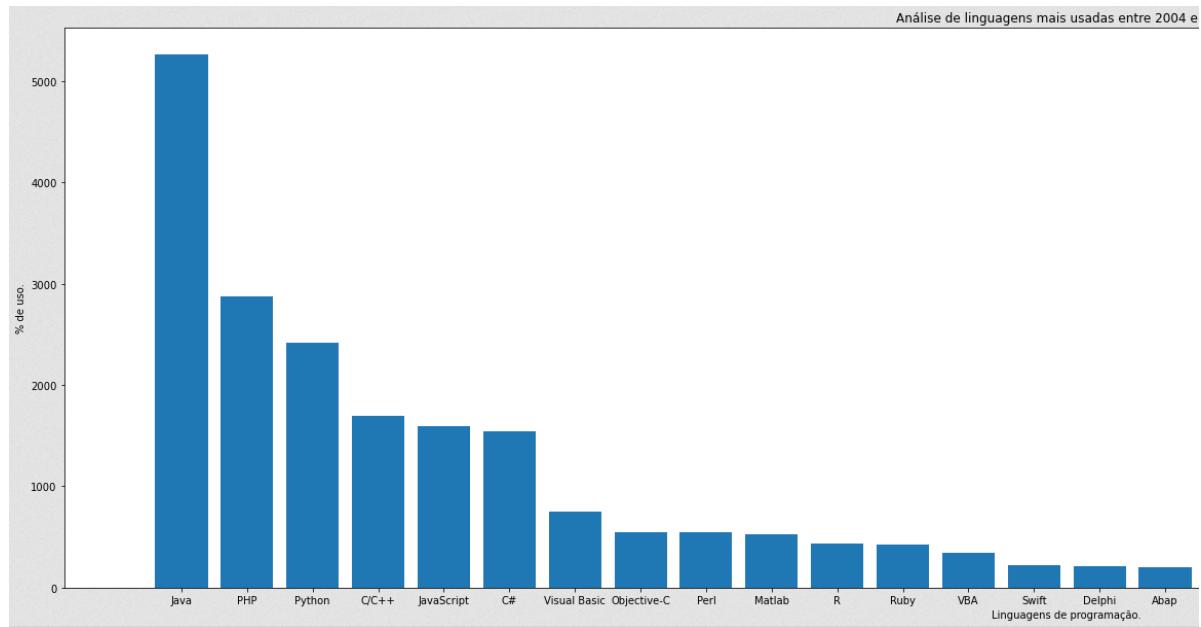
```
fig, ax = plt.subplots()

ax.set_xticklabels(dados_agrupados['Language'])

ax.bar(x = dados_agrupados['Language'], height = dados_agrupados['Value'])
plt.gcf().set_size_inches(35, 10)
plt.savefig('imgs/Analise Linguagens.png')

configurar_plot_com_dimensoes(
    'Análise de linguagens mais usadas entre 2004 e 2021.',
    'Linguagens de programação.',
    '% de uso.',
    35,
    10
)
```

	Language	Value
10	Java	5265.13
17	PHP	2872.86
19	Python	2419.33
3	C/C++	1691.65
11	JavaScript	1598.16
2	C#	1543.45
27	Visual Basic	754.92
16	Objective-C	546.40
18	Perl	543.92
15	Matlab	531.17
20	R	430.32
21	Ruby	428.81
26	VBA	340.84
24	Swift	225.21
6	Delphi	214.20
0	Abap	202.20



Em seguida, é possível verificar se para este dataset existe correlação entre as variáveis, e verificar se os dados estão de acordo com a evolução das linguagens em relação ao ano.

- Igual a 1 -----> Correlação linear positiva perfeita
- Maior que 0 -----> Correlação linear positiva
- Igual a 0 -----> Sem correlação linear
- Menor que 0 -----> Correlação linear negativa
- Igual a -1 -----> Correlação linear negativa perfeita

A relação acima exemplifica o resultado das correlações. Logo abaixo, será mostrada a criação de uma lista de dicionários chamado *targets*, cada dicionário será composto por uma estrutura com *linguagem_1* e *linguagem_2*, contendo as linguagens que serão comparadas para verificar se há correlação. As correlações analisadas serão, nos seguintes pares: Java e Javascript, Java e Python, Javascript e Python, Javascript e Typescript, Java e C/C++, PHP e Javascript, PHP e Java, R e Python, R e Javascript, R e Typescript.

```

targets = [
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'TypeScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'C/C++'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Java'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'TypeScript'
    }
]

```

Em seguida, serão criadas duas funções, a primeira chamada *verificar_correlacao*, que irá verificar a correlação com dados descritos, e outra que apenas irá plotar os resultados das correlações entre as linguagens, chamada *plotar_correlacao*. O cálculo da correlação será utilizando a função *corr()* da biblioteca Pandas.

```

def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = df[linguagem_1].corr(df[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    df.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')

configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), '', '', 20, 10)

```

A execução das funções será através de dois loops *for* de repetição iterando a lista de dicionários criada anteriormente, que estará passando as duas linguagens por parâmetro.

```

for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])

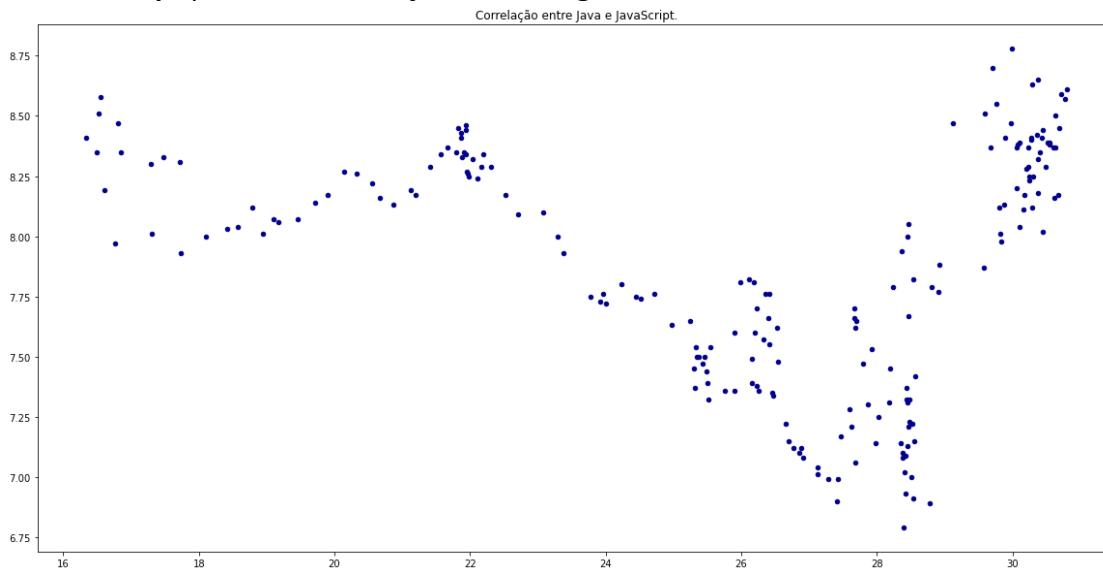
```

Os resultados descritos pela função *verificar_correlacao* foram os seguintes:

Verificando a correlação Pearson entre os % de uso das linguagens: Java e JavaScript.

-0.1759104243423126

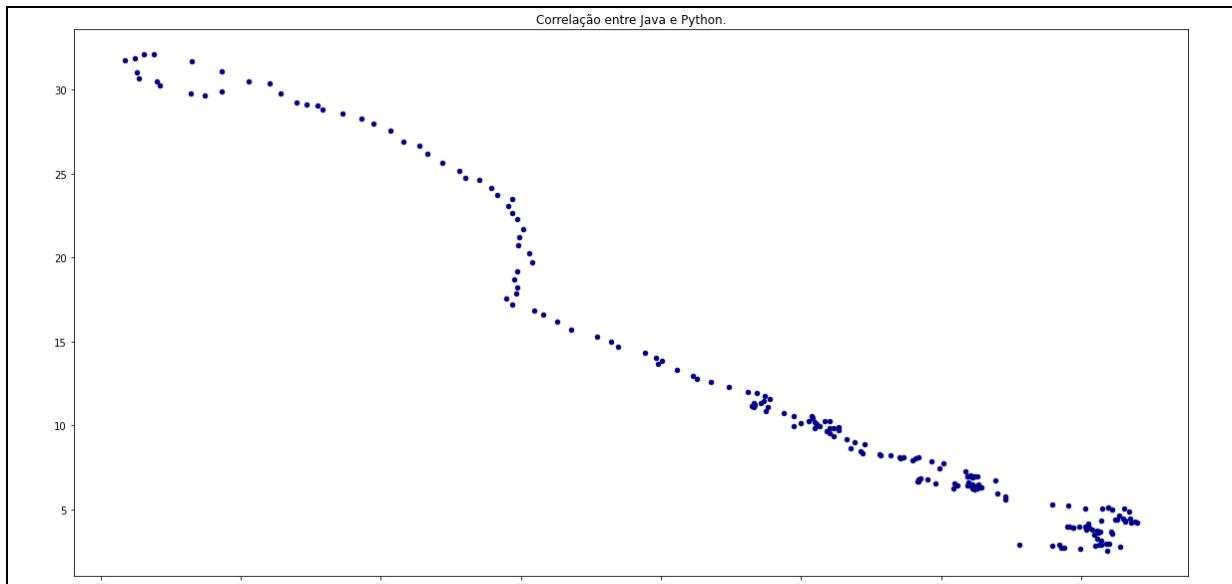
Java e JavaScript possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: Java e Python.

-0.9815602884038716

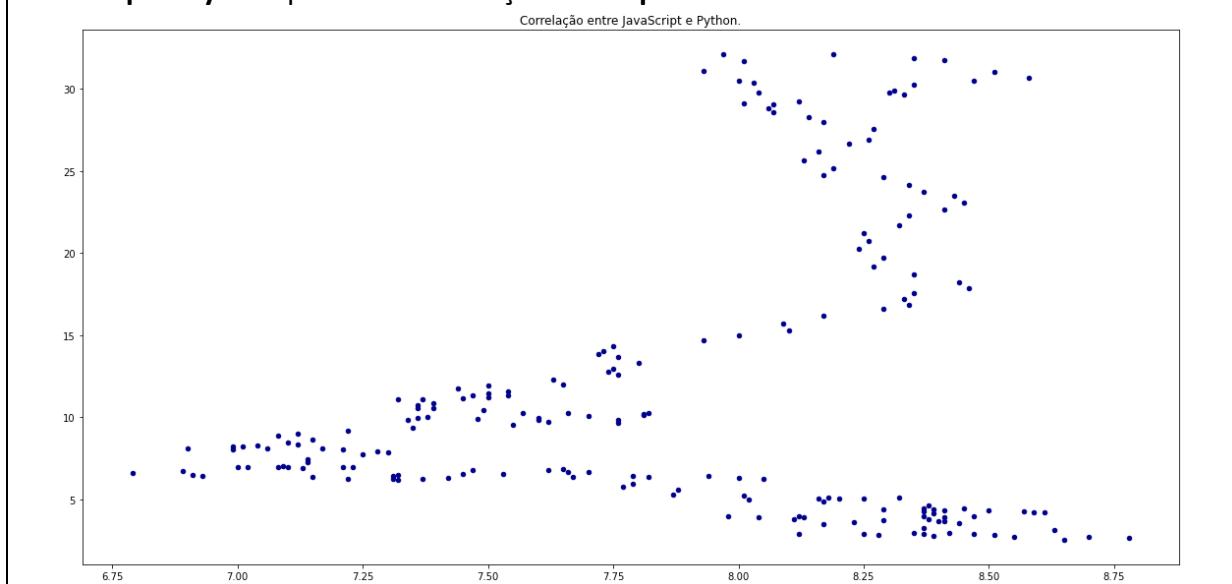
Java e Python possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: JavaScript e Python.

0.24168739345302312

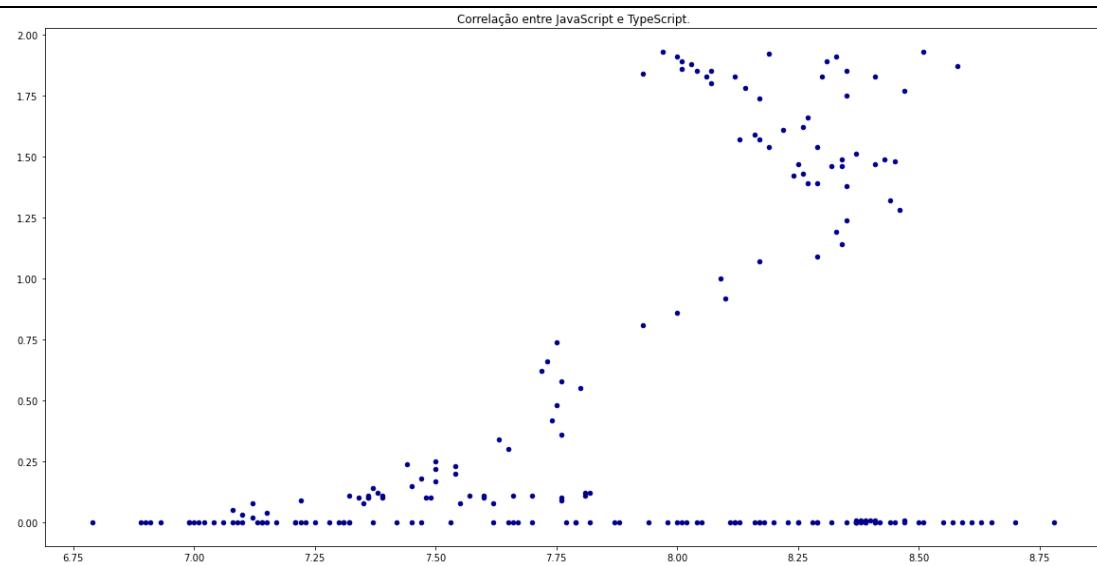
JavaScript e Python possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: JavaScript e TypeScript.

0.4127565560616799

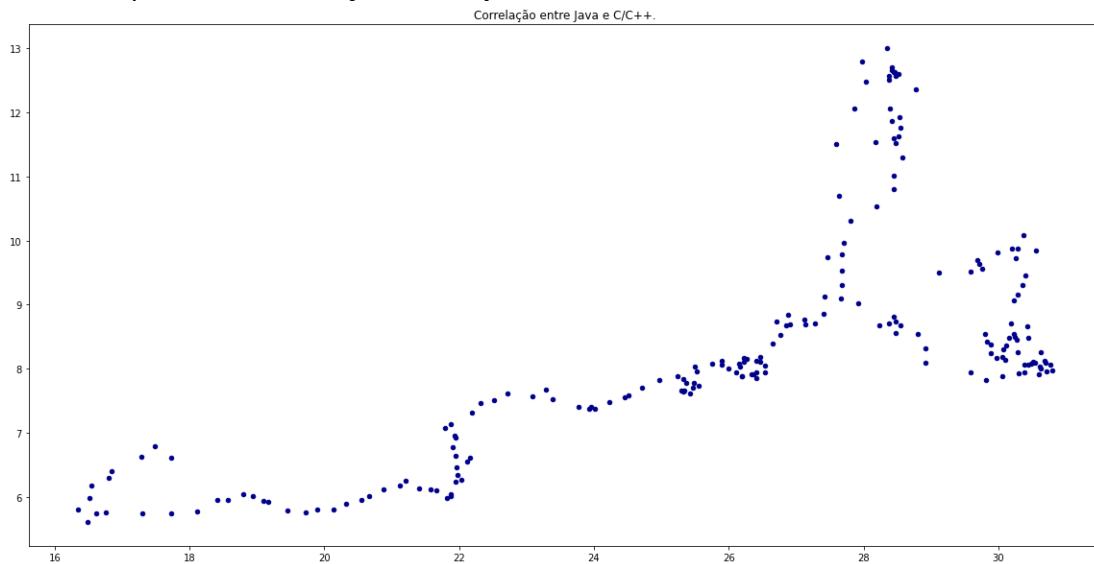
JavaScript e TypeScript possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: Java e C/C++.

0.6779880318013292

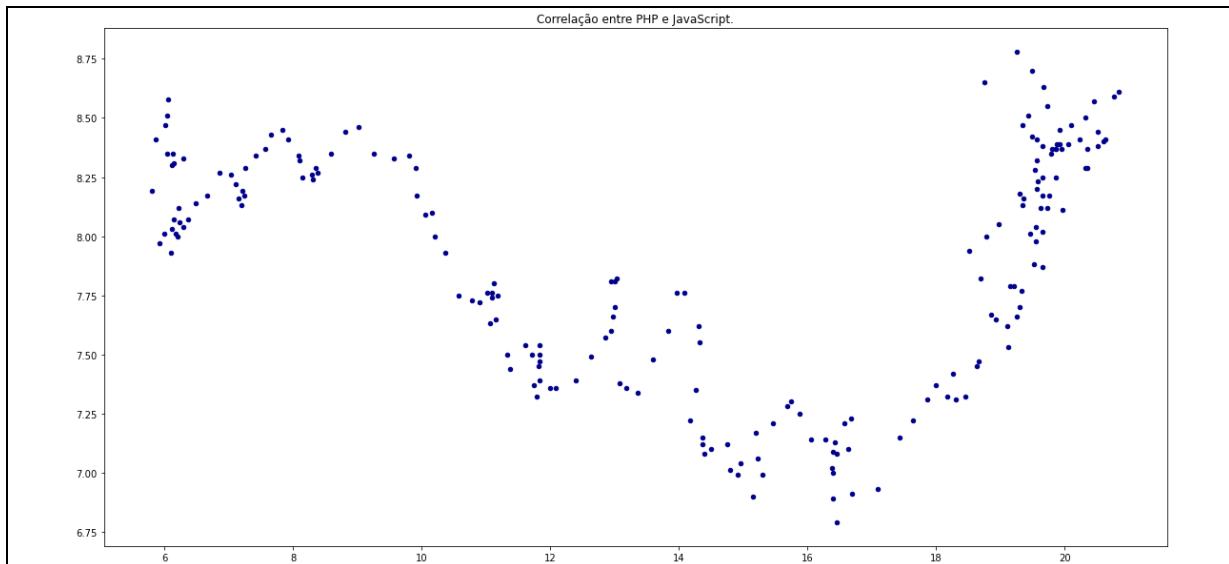
Java e C/C++ possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e JavaScript.

-0.07984499408856426

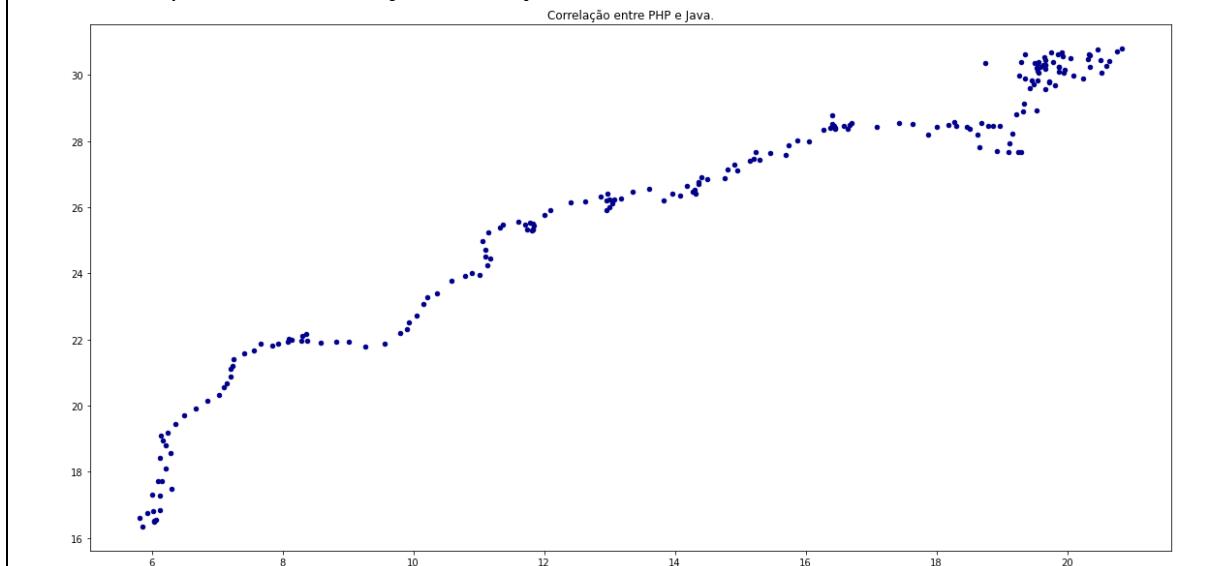
PHP e JavaScript possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e Java.

0.964499161229616

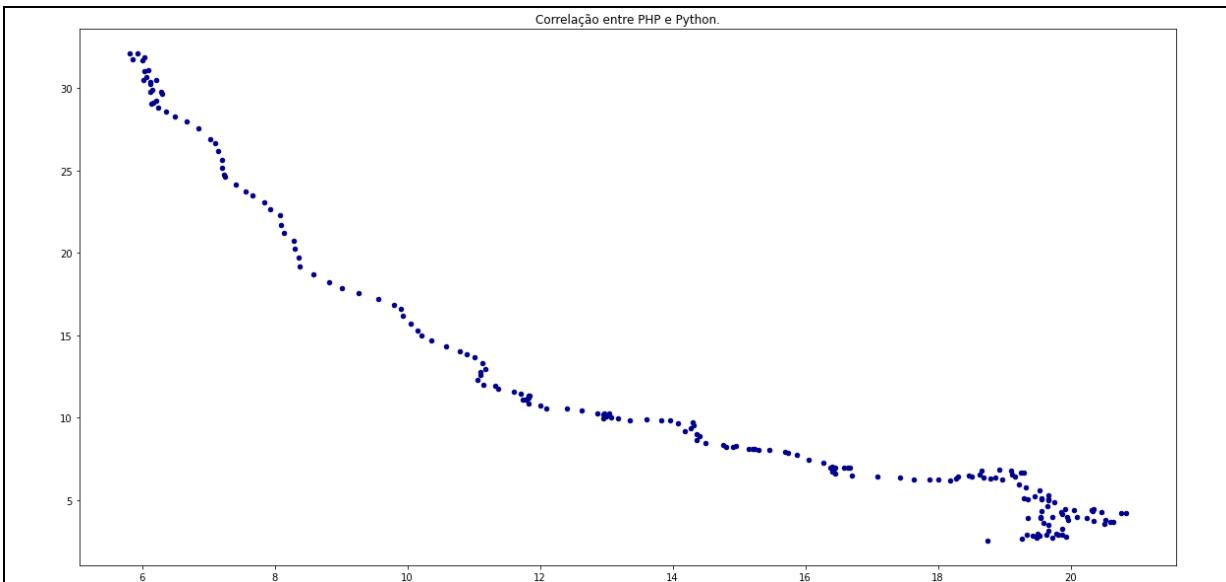
PHP e Java possuem correlação **linear positiva**.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e Python.

-0.9406353162883564

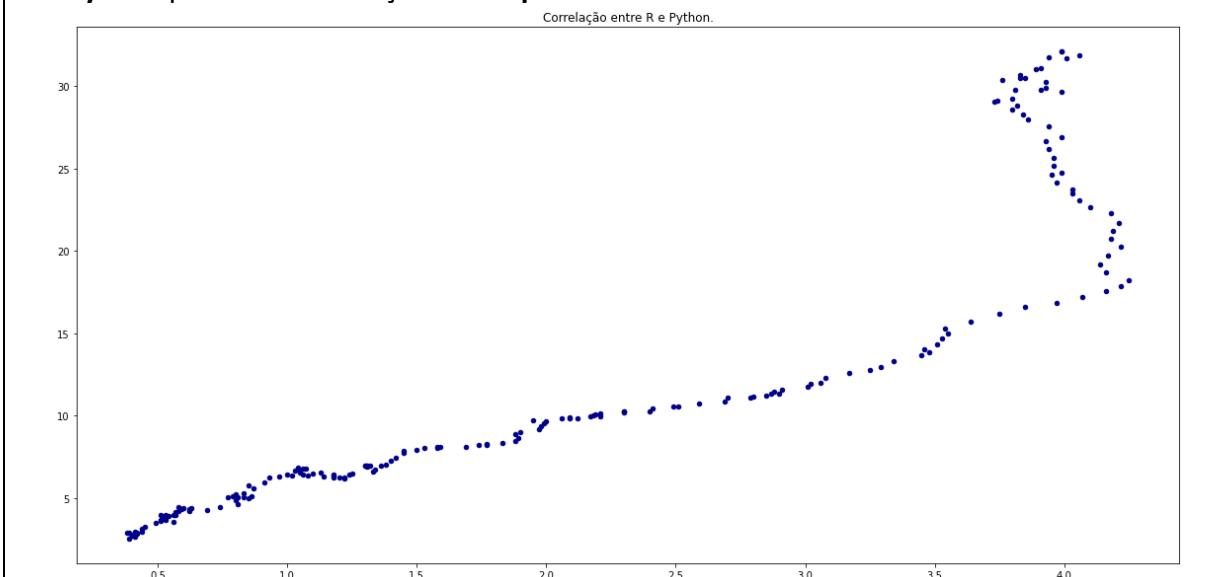
PHP e Python possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: R e Python.

0.9028122654817621

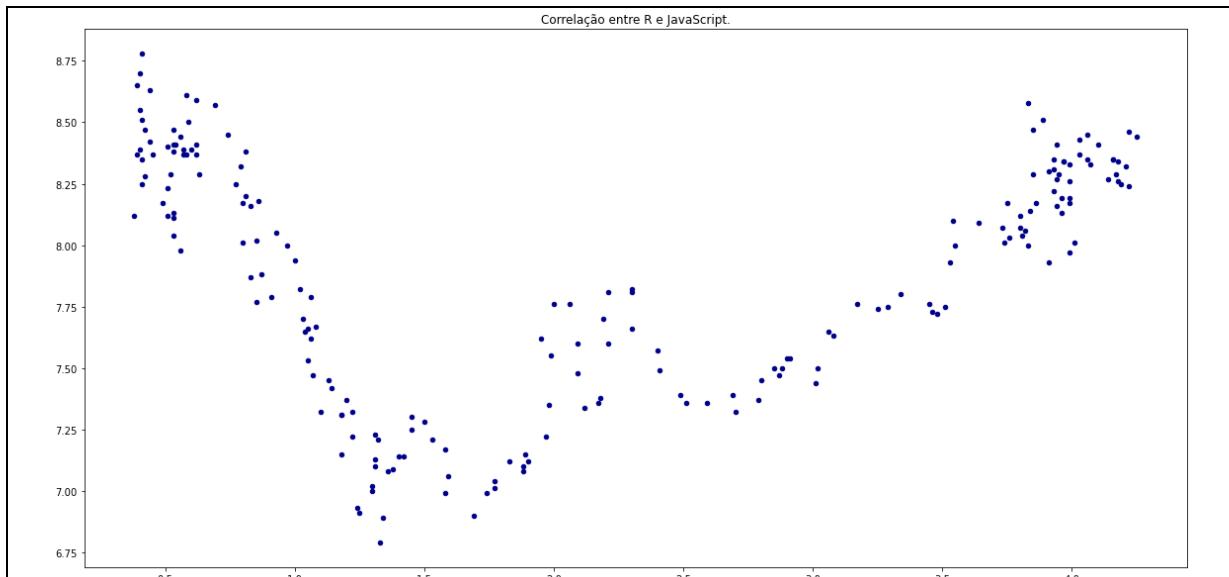
R e Python possuem correlação **linear positiva**.



Verificando a correlação Pearson entre os % de uso das linguagens: R e JavaScript.

0.11105986700049135

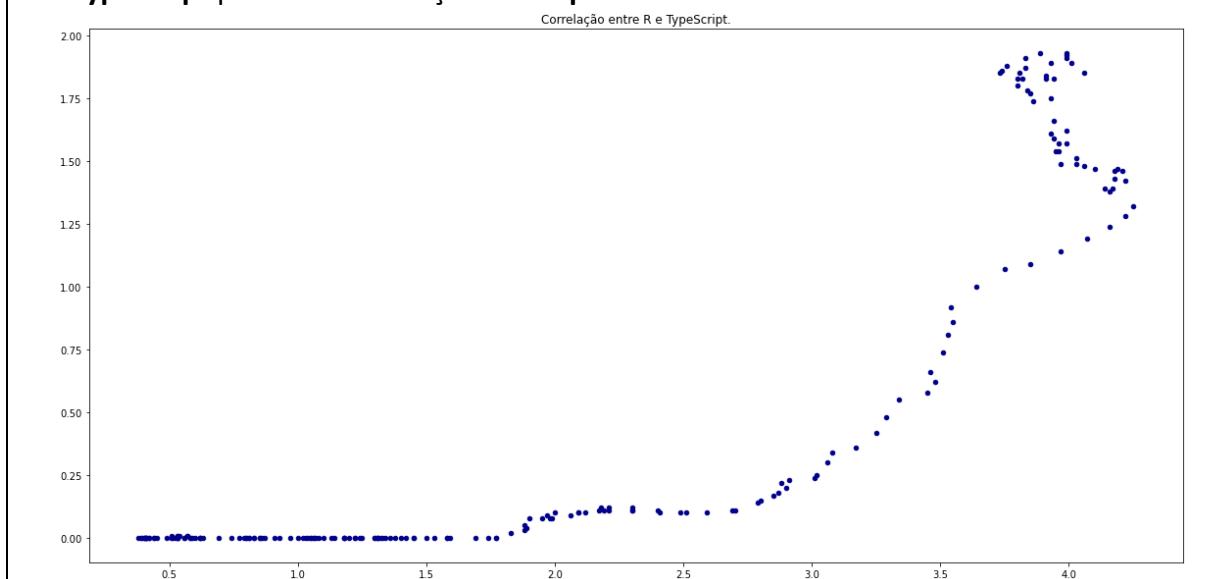
R e JavaScript possuem correlação **linear positiva**.



Verificando a correlação Pearson entre os % de uso das linguagens: R e TypeScript.

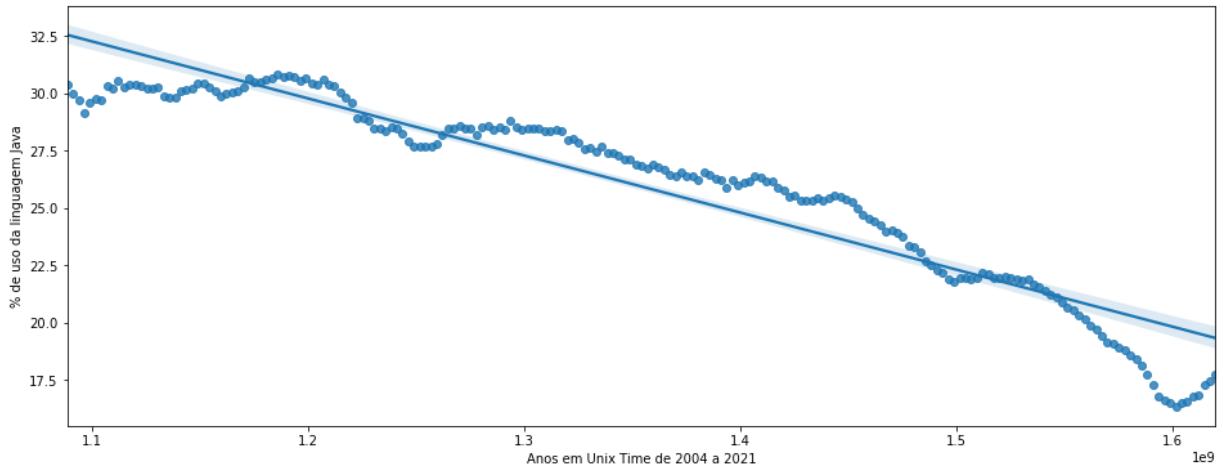
0.868588566414167

R e TypeScript possuem correlação linear positiva.



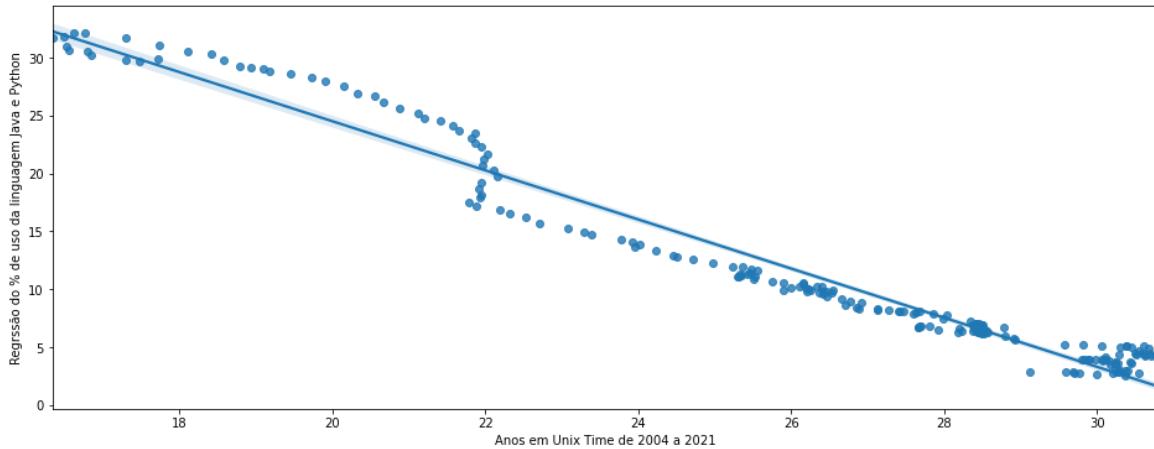
Em seguida será verificado como foi o andamento da linguagem Java através dos anos, para isso, será utilizada a coluna UnixTime e a biblioteca Seaborn para realizar uma plotagem de uma regressão linear simples. Para isso, primeiro é criado um novo DataFrame chamado *df_java* que é apenas um filtro do Pandas para o dataset *dados_tratados*.

```
df_java = dados_tratados[dados_tratados['Language'] == 'Java']
sns.regplot(x="UnixTime", y="Value", data=df_java)
plt.gcf().set_size_inches(16, 6)
plt.ylabel('% de uso da linguagem Java')
plt.xlabel('Anos em Unix Time de 2004 a 2021')
plt.show()
```



Como pode ser visto, houve uma queda no percentual de uso da linguagem Java com o avanço dos anos que se passaram, confirmando, através dos dados do GitHub, que uma das maiores e mais utilizadas linguagens de programação com tipagem forte e compilada teve um declínio de uso, o mesmo pode ser observado se considerar uma regressão linear entre Java e Python:

```
t = df[['Java', 'Python']]
sns.regrplot(x="Java", y="Python", data=t)
plt.gcf().set_size_inches(16, 6)
plt.ylabel('Regressão do % de uso da linguagem Java e Python')
plt.xlabel('Anos em Unix Time de 2004 a 2021')
plt.show()
```



A última análise a ser feita no dataset é uma evolução comparativa entre algumas linguagens com plotagens de gráficos de linhas através da data. Para isso, algumas transformações serão realizadas, como a criação de um novo *dataset* a partir do original importado, o *df*, e será setada a coluna Date como o índice do Pandas para este DataFrame.

Em seguida, será realizada a verificação para as linguagens Java, Javascript, Python, C#, PHP, Delphi, Dart, Cobol, Go, C/C++, Groovy, Typescript, Kotlin e R.

```

dados['Date'] = pd.to_datetime(dados['Date'])
dados.set_index('Date', inplace = True)

dados.head()

```

	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	Haskell	...	PHP	Python	R	Ruby	Rust	Scala	Swift	TypeScript	VBA	Visual Basic
Date																					
2004-07-01	0.34	0.36	10.08	4.71	0.43	0.0	2.82	0.0	0.03	0.22	...	18.75	2.53	0.39	0.33	0.08	0.03	0.0	0.0	1.44	8.56
2004-08-01	0.36	0.36	9.81	4.99	0.46	0.0	2.67	0.0	0.07	0.20	...	19.26	2.64	0.41	0.40	0.09	0.03	0.0	0.0	1.46	8.57
2004-09-01	0.41	0.41	9.63	5.06	0.51	0.0	2.65	0.0	0.08	0.21	...	19.49	2.72	0.40	0.41	0.10	0.03	0.0	0.0	1.55	8.41
2004-10-01	0.40	0.38	9.50	5.31	0.53	0.0	2.77	0.0	0.09	0.20	...	19.34	2.92	0.42	0.46	0.11	0.04	0.0	0.0	1.61	8.49
2004-11-01	0.38	0.38	9.52	5.24	0.55	0.0	2.76	0.0	0.07	0.24	...	19.43	2.84	0.41	0.45	0.13	0.04	0.0	0.0	1.50	8.24

5 rows × 28 columns

Todos os plots das linguagens citadas acima serão então ser realizados com a própria plotagem interna do Pandas que utiliza o back-end da Matplotlib.

```

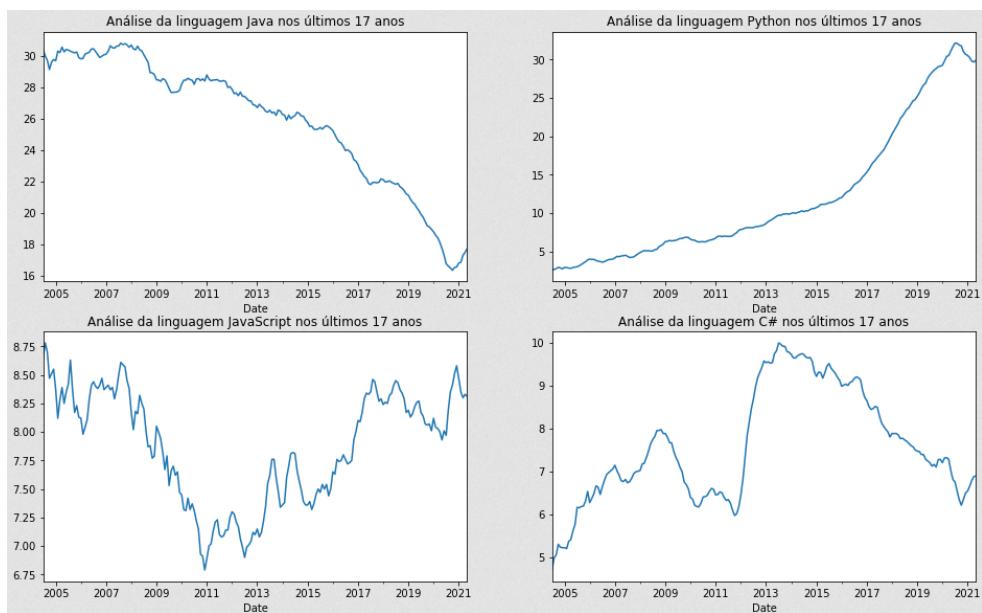
fig, axes = plt.subplots(nrows=7, ncols=2)

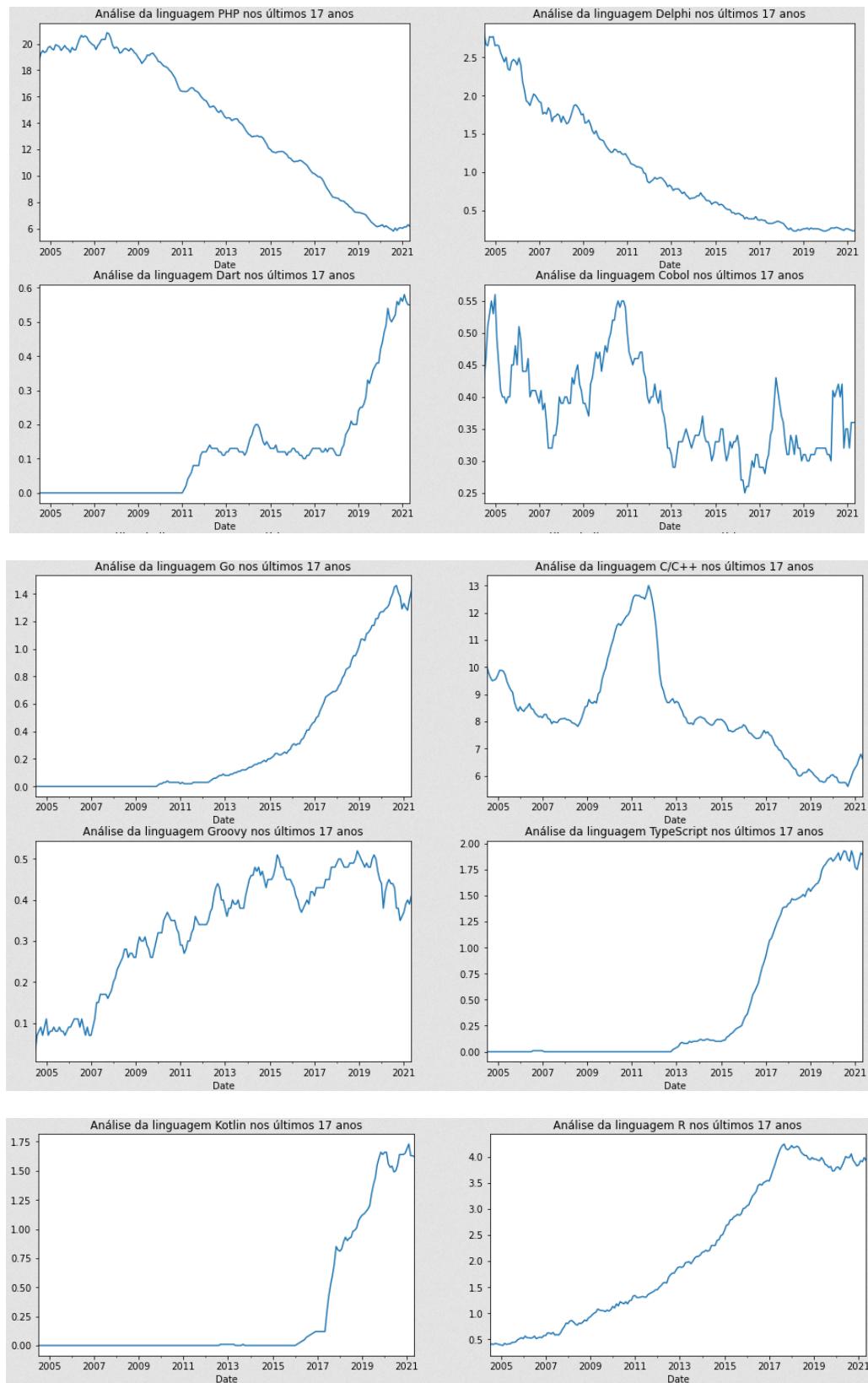
dados['Java'].plot(ax=axes[0,0], title = "Análise da linguagem Java nos últimos 17 anos")
dados['JavaScript'].plot(ax=axes[1,0], title = "Análise da linguagem JavaScript nos últimos 17 anos")
dados['Python'].plot(ax=axes[0,1], title = "Análise da linguagem Python nos últimos 17 anos")
dados['C#'].plot(ax=axes[1,1], title = "Análise da linguagem C# nos últimos 17 anos")
dados['PHP'].plot(ax=axes[2,0], title = "Análise da linguagem PHP nos últimos 17 anos")
dados['Delphi'].plot(ax=axes[2,1], title = "Análise da linguagem Delphi nos últimos 17 anos")
dados['Dart'].plot(ax=axes[3,0], title = "Análise da linguagem Dart nos últimos 17 anos")
dados['Cobol'].plot(ax=axes[3,1], title = "Análise da linguagem Cobol nos últimos 17 anos")
dados['Go'].plot(ax=axes[4,0], title = "Análise da linguagem Go nos últimos 17 anos")
dados['C/C++'].plot(ax=axes[4,1], title = "Análise da linguagem C/C++ nos últimos 17 anos")
dados['Groovy'].plot(ax=axes[5,0], title = "Análise da linguagem Groovy nos últimos 17 anos")
dados['TypeScript'].plot(ax=axes[5,1], title = "Análise da linguagem TypeScript nos últimos 17 anos")
dados['Kotlin'].plot(ax=axes[6,0], title = "Análise da linguagem Kotlin nos últimos 17 anos")
dados['R'].plot(ax=axes[6,1], title = "Análise da linguagem R nos últimos 17 anos")

plt.gcf().set_size_inches(16, 36)

plt.savefig('imgs/Comparação linguagens 17 anos')
plt.show()

```





Com essa análise, pode-se observar que linguagens compiladas e com forte tipagem como Java, C/C++, Delphi, Cobol e C# tiveram um aumento significativo entre os anos de 2004 a 2010, e a partir disso começaram a ter uma queda, enquanto isso, linguagens inter-

pretadas com tipagem fraca como Javascript e Python tiveram um grande aumento a partir de 2010, com significativa curva positiva a partir de 2015.

É possível ver aumentos significativos também para linguagens compiladas e/ou com forte tipagem como Typescript, Groovy e Go a partir desta data também, o que pode ser explicado por serem linguagens mais novas, rápidas e com várias funcionalidades de programação funcionais, embora não sejam linguagens puramente funcionais.

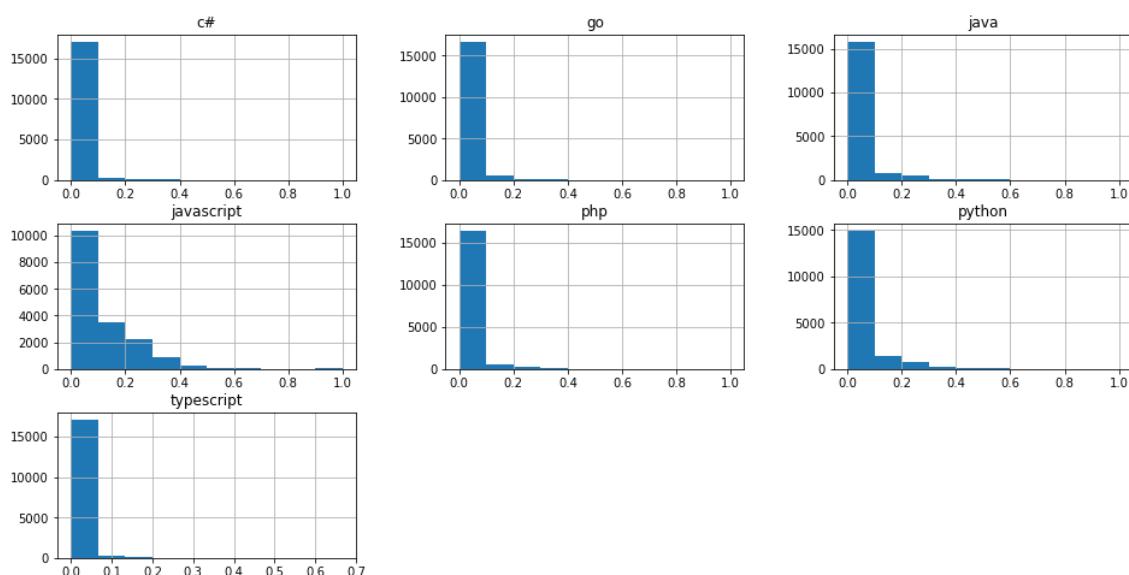
Linguagens como Dart são fortemente tipadas, porém, interpretadas de script, e mesmo com essas características, possui grande auxílio em sua funcionalidade para desenvolvimento de aplicações mobile, o que justifica seu grande aumento a partir de 2017 com a grande ascensão de várias aplicações e serviços mobile, concorrendo diretamente com Kotlin e Javascript.

Kotlin começou a ter popularidade também a partir de 2017 por objetivos parecidos com os citados por Groovy e Go, e a linguagem R, por mais que já existisse há um tempo maior, com a grande procura pela área de ciência de dados teve um crescimento expressivo a partir de 2015.

4.2. Análise exploratória dos dados do Github de 2017

Para começar a análise, será verificado como estão distribuídos os dados através de um gráfico de histograma do dataset de 2017, como pode ser analisado abaixo:

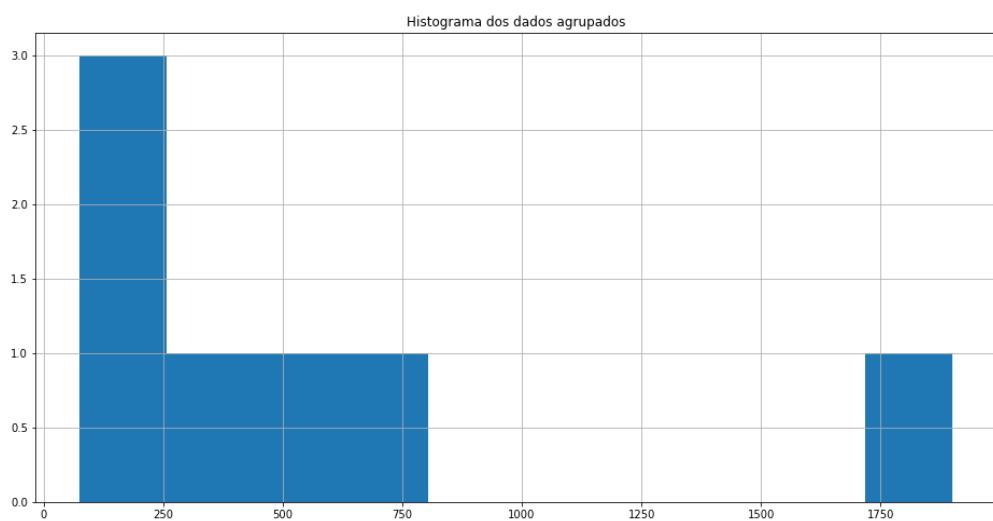
```
dados_2017.hist()
configurar_plot('', '', 'Histograma das linguagens a serem analisadas')
```



É possível verificar que a maioria dos dados ficaram na faixa dos 0 a 0.4, ou seja, de 0 a 40% de uso, com exceção de algumas linguagens como Java, Typescript e Python que chegaram a atingir 60%.

Ao configurar uma plotagem de um histograma do dataset que criamos para ser agrupado em apenas duas colunas de valor e linguagem, temos o seguinte resultado:

```
dft.hist()
configurar_plot('Histograma dos dados agrupados', '', '')
```



É feito um somatório de todas as linguagens agrupando seus percentuais, e com isso pode-se verificar que há uma área que fica em branco nos dados a partir de 750 até 1750. Com esses dados agrupados, agora conseguimos avaliar um ranking das linguagens mais utilizadas no dataset:

```
dft = dft.sort_values(by=['Value'], ascending=False)

fig, ax = plt.subplots()

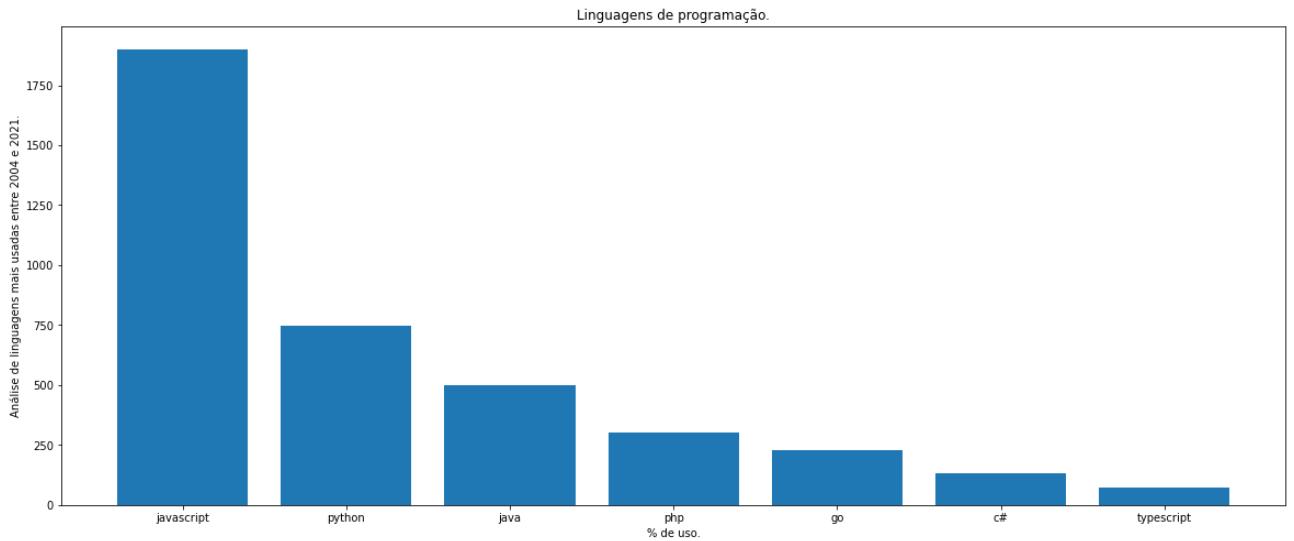
ax.set_xticklabels(dft['Language'])

ax.bar(x = dft['Language'], height = dft['Value'])

plt.gcf().set_size_inches(35, 10)

plt.savefig('imgs/Analise Linguagens Dataset 02.png')

configurar_plot_com_dimensoes(
    'Linguagens de programação.',
    '% de uso.',
    'Análise de linguagens mais usadas entre 2004 e 2021.',
    20,
    8
)
```



Com isso, é possível verificar que, assim como no ranking dos dados de 2004 a 2021, Java, Python, Javascript e PHP estão novamente entre as 5 mais utilizadas. A linguagem C# novamente assumiu a posição de 6º lugar, assim como na comparação com a série temporal visto no subcapítulo anterior.

A próxima análise a ser feita é se este dataset possui correlação entre as linguagens. A análise será uma lista chamada *target* exatamente igual no exemplo realizado acima, e recriando as funções de verificar e plotar correlação para verificar se pode ser vista alguma correlação entre os percentuais apresentados para o dataset.

```
targets = [
    {
        'linguagem_1': 'java',
        'linguagem_2': 'javascript'
    },
    {
        'linguagem_1': 'java',
        'linguagem_2': 'python'
    },
    {
        'linguagem_1': 'javascript',
        'linguagem_2': 'python'
    },
    {
        'linguagem_1': 'javascript',
        'linguagem_2': 'typescript'
    },
    {
        'linguagem_1': 'java',
        'linguagem_2': 'php'
    },
    {
        'linguagem_1': 'java',
        'linguagem_2': 'c#'
    },
    {
        'linguagem_1': 'php',
        'linguagem_2': 'python'
    },
    {
        'linguagem_1': 'php',
        'linguagem_2': 'javascript'
    }
]
```

Em seguida, as criações das funções e a execução de ambas através de dois loops do tipo *for* apontando para as linguagens escolhida em cada dicionário da lista:

```
def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = dados_2017[linguagem_1].corr(dados_2017[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    dados_2017.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')

    configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), '', '', 20, 10)

for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

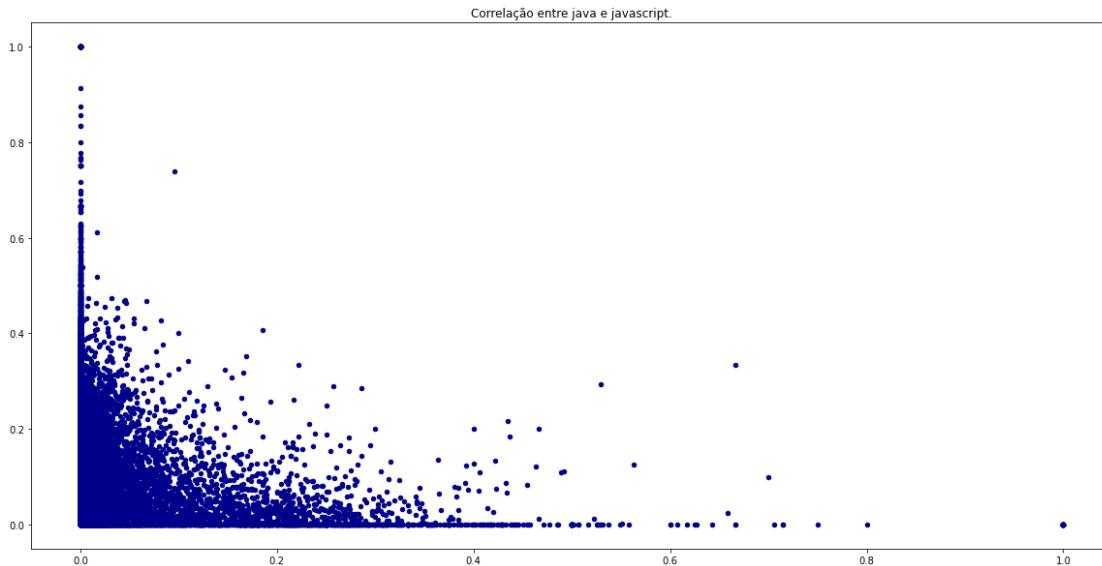
for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])
```

O resultado das análises será exibido na imagem abaixo:

Verificando a correlação Pearson entre os % de uso das linguagens: java e javascript.

-0.19341412138059294

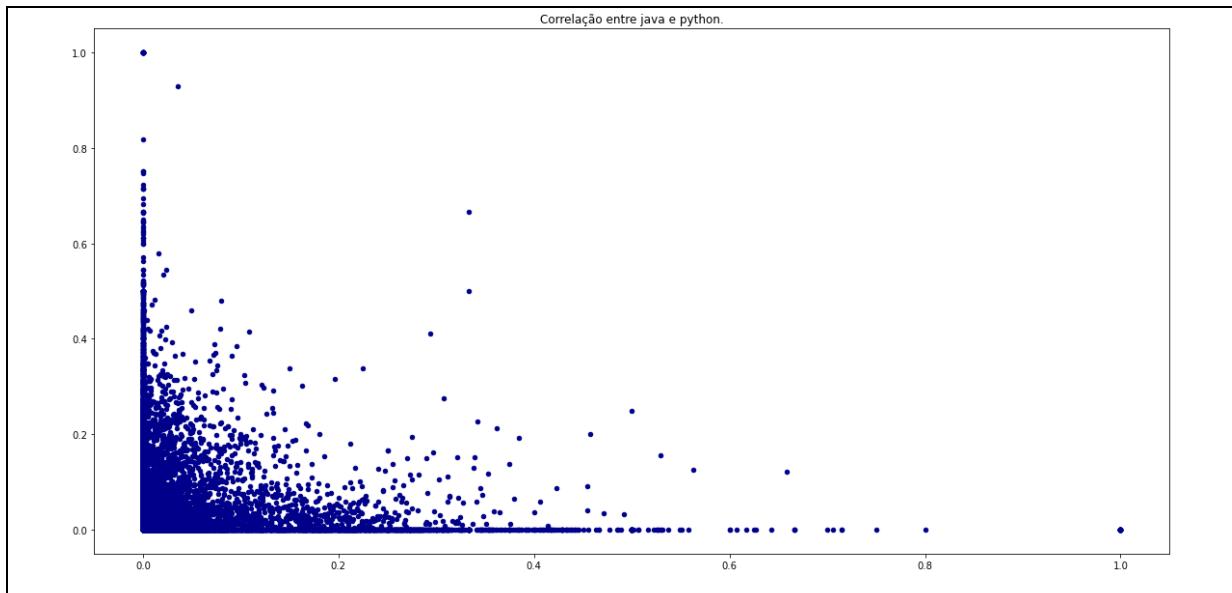
java e javascript possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: java e python.

-0.0679272004455616

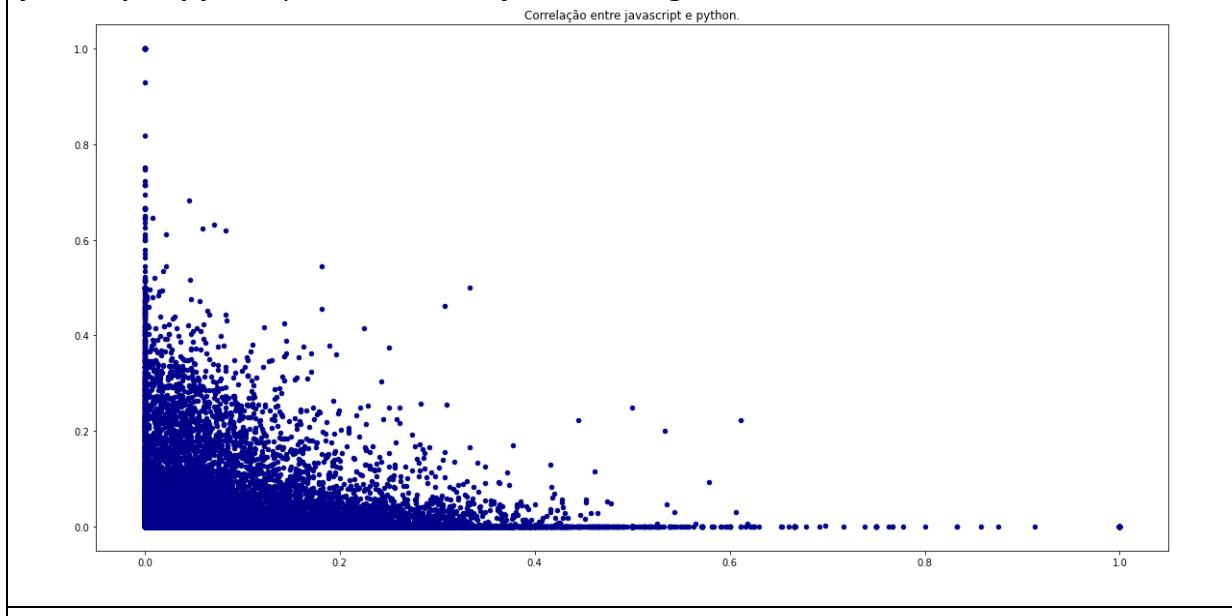
java e python possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: javascript e python.

-0.2122198553761165

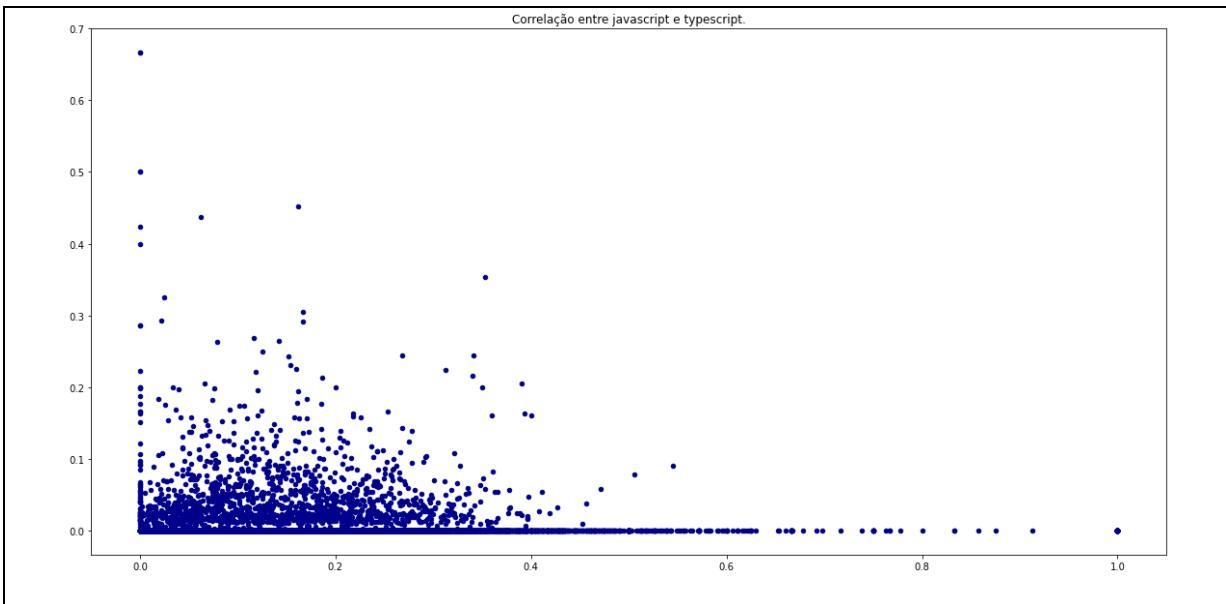
javascript e python possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: javascript e typescript.

0.04626619040646336

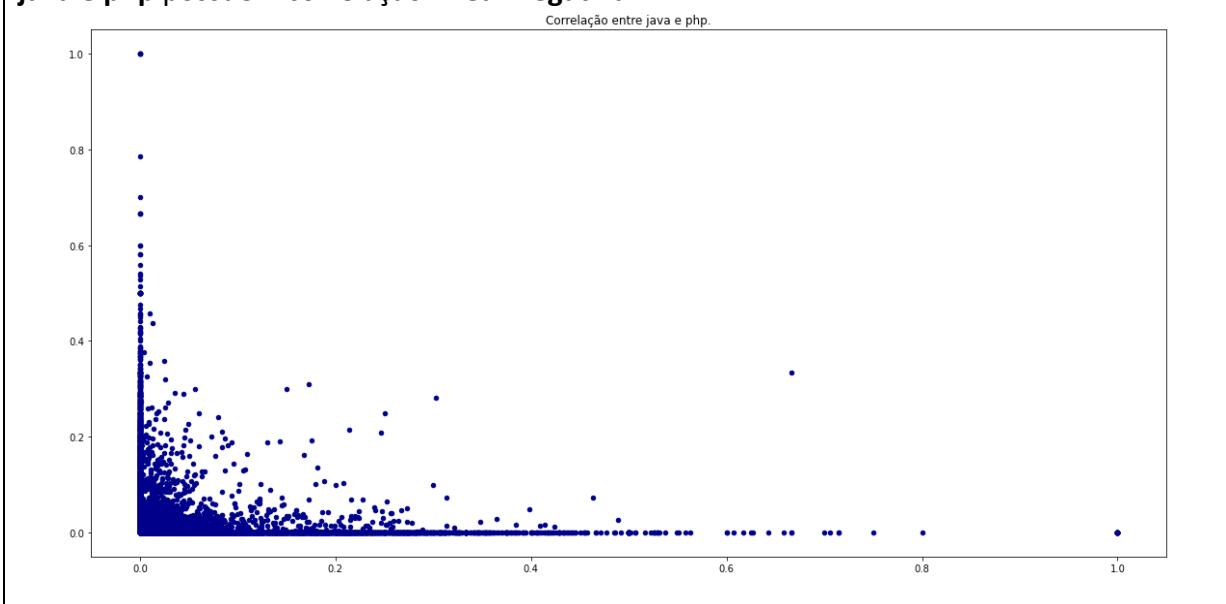
javascript e typescript possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: java e php.

-0.07461796891834088

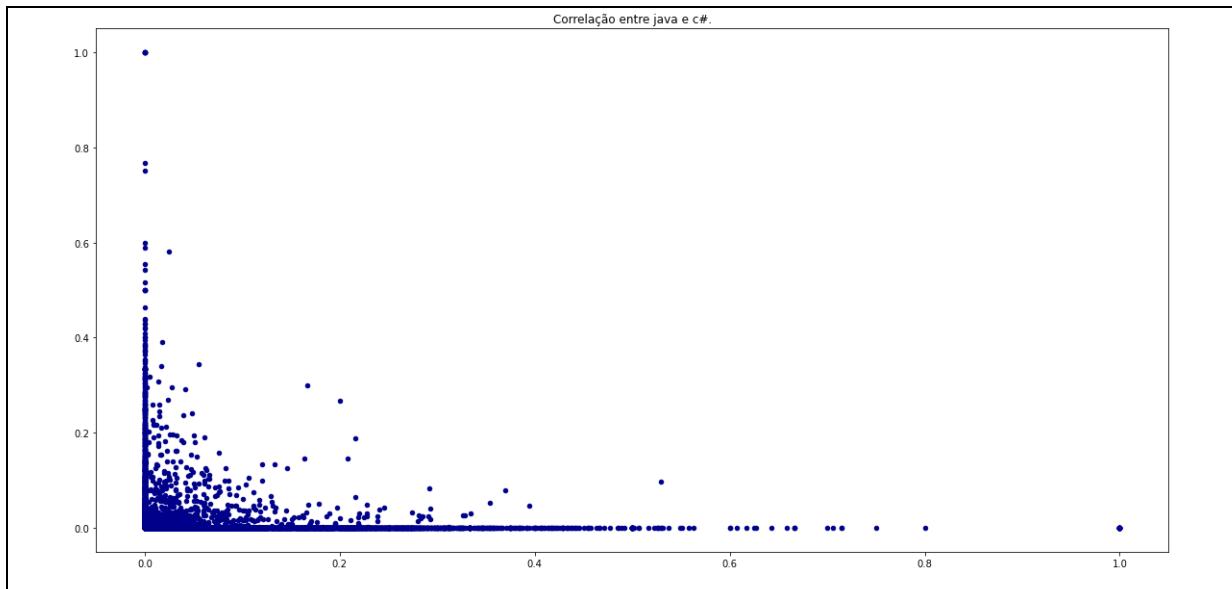
java e php possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: java e c#.

-0.03668505222036968

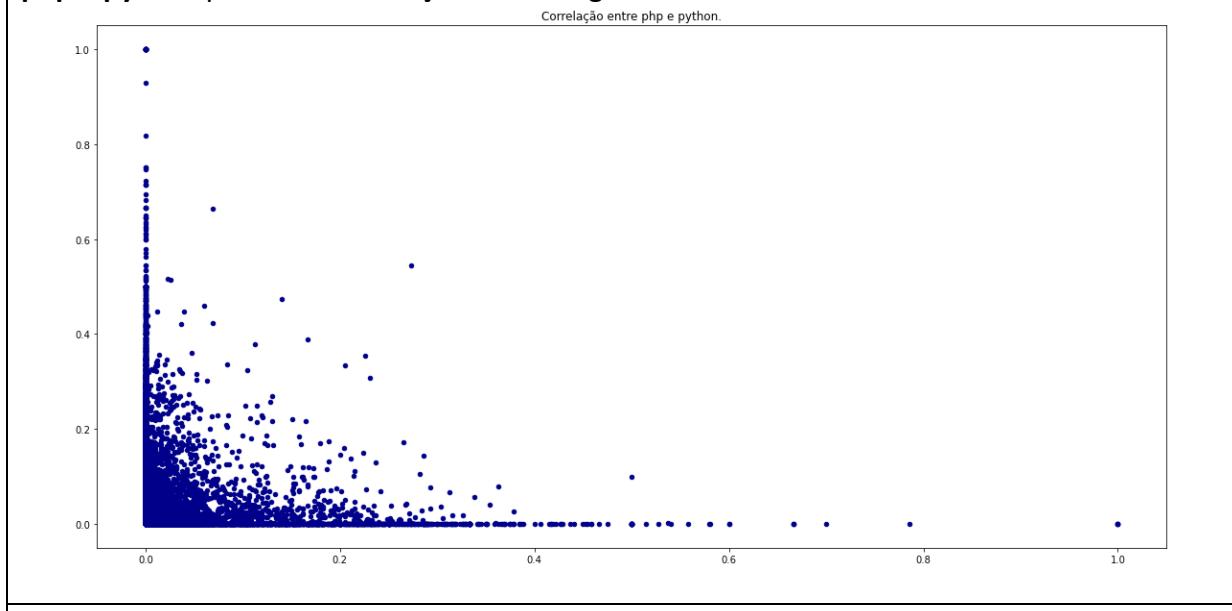
java e c# possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: php e python.

-0.07667578500179618

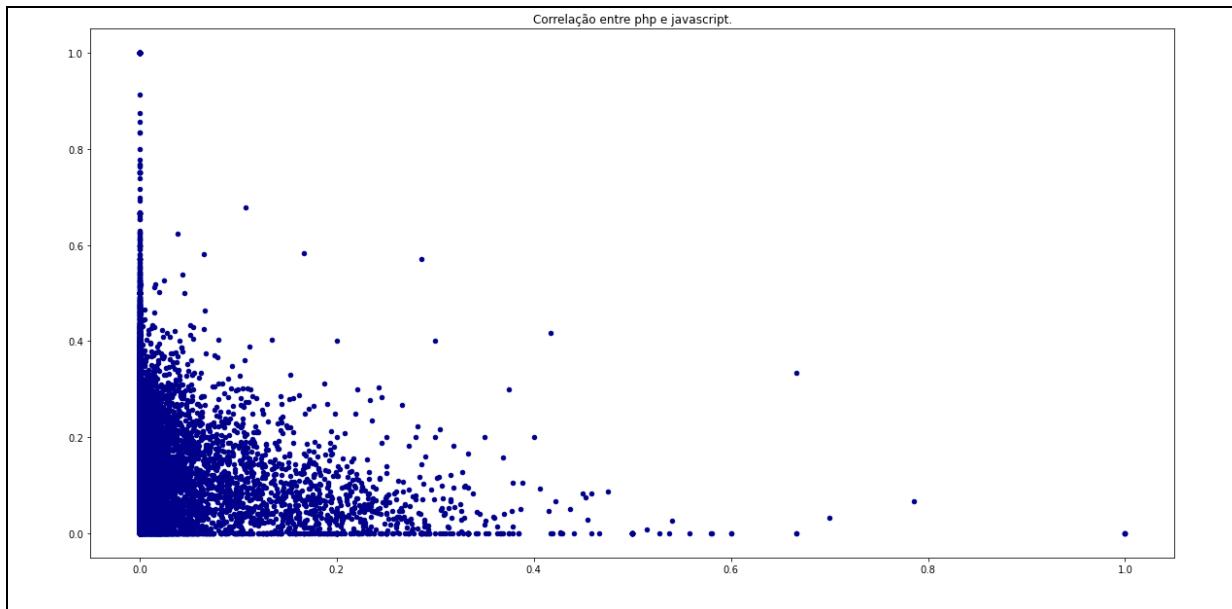
php e python possuem correlação linear negativa.



Verificando a correlação Pearson entre os % de uso das linguagens: php e javascript.

-0.06472421239746132

php e javascript possuem correlação linear negativa.



Como pode ser visto através dos resultados da tabela acima, há correlação positiva e negativa entre as variáveis, no caso, entre as linguagens. Porém, as correlações são quase próximas a 0, indicando que é mínima a correlação existente neste dataset, isso pode ser visto até mesmo entre linguagens que apresentaram excelente correlação, como PHP e Java, que foi de 0.96 no primeiro dataset analisado, e nesse dataset foi de -0.07, ou seja, mesmo tendo correlação negativa, é bem próxima a zero.

4.3. Análise exploratória dos dados do StackOverflow de 2008 a 2021

A primeira análise a ser feita é se o ranking das linguagens de programação com maior quantidade de perguntas feitas na plataforma do StackOverflow, e para essa análise, iremos criar um novo DataFrame chamado *dados_tratados_so_gb* utilizando a função *groupby* da biblioteca Pandas, e em sequência, será plotado o ranking.

```

dados_tratados_so_gb = dados_tratados_so\
    .groupby(by=['Language'], as_index=False)\\
    .sum()\\
    .sort_values(by=['Value'], ascending=False)

fig, ax = plt.subplots()

ax.set_xticklabels(dados_tratados_so_gb['Language'])

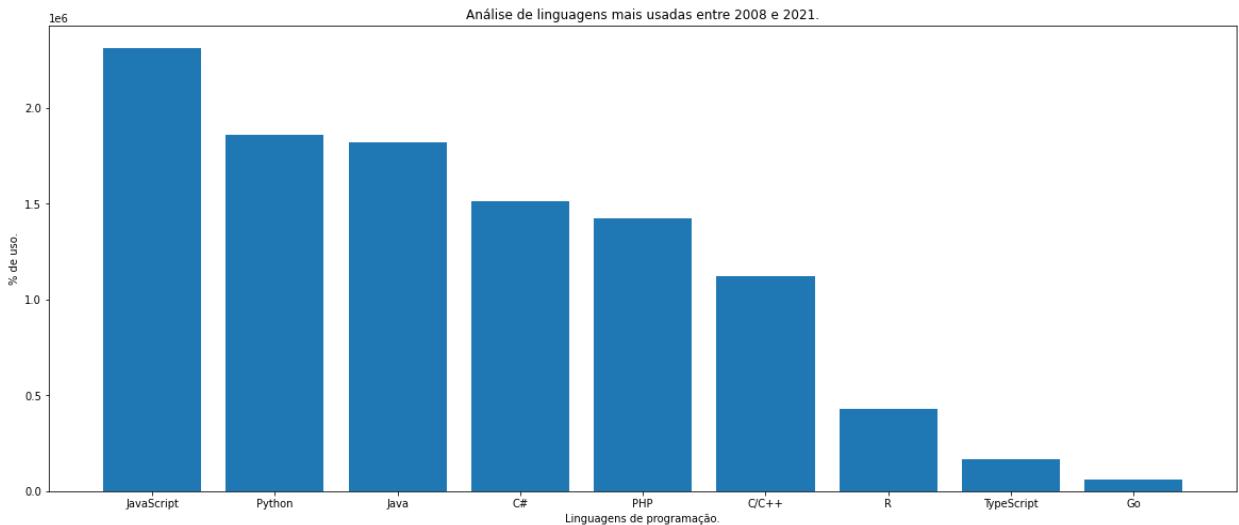
ax.bar(x = dados_tratados_so_gb['Language'], height = dados_tratados_so_gb['Value'])

plt.gcf().set_size_inches(35, 10)

plt.savefig('imgs/Analise Linguagens Dataset 02.png')

configurar_plot_com_dimensoes(
    'Análise de linguagens mais usadas entre 2008 e 2021.',
    'Linguagens de programação.',
    '% de uso.',
    20,
    8
)

```



É possível verificar que, extraíndo a quantidade de perguntas feitas sobre as tecnologias acima, as mesmas 5 linguagens de programação continuam estando em topo no ranking, sendo elas Java, Javascript, Python, C# e PHP. Deste modo, é possível verificar um padrão entre essas tecnologias independente do dataset extraído, se for acima de 2015, é muito provável que essas linguagens aparecerão no top 5, não necessariamente na mesma posição.

Em sequência, será feita análise de correlação entre as variáveis. A lista de dicionários chamada *target* será a mesma apresentada neste capítulo, seção 4.1. Abaixo foi adaptada a função de verificar e plotar a correlação como feito anteriormente, porém, para o atual dataset.

```

def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = correlation_df[linguagem_1].corr(correlation_df[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}\n'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    correlation_df.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')

    configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), 10, 10, 20, 10)

for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])

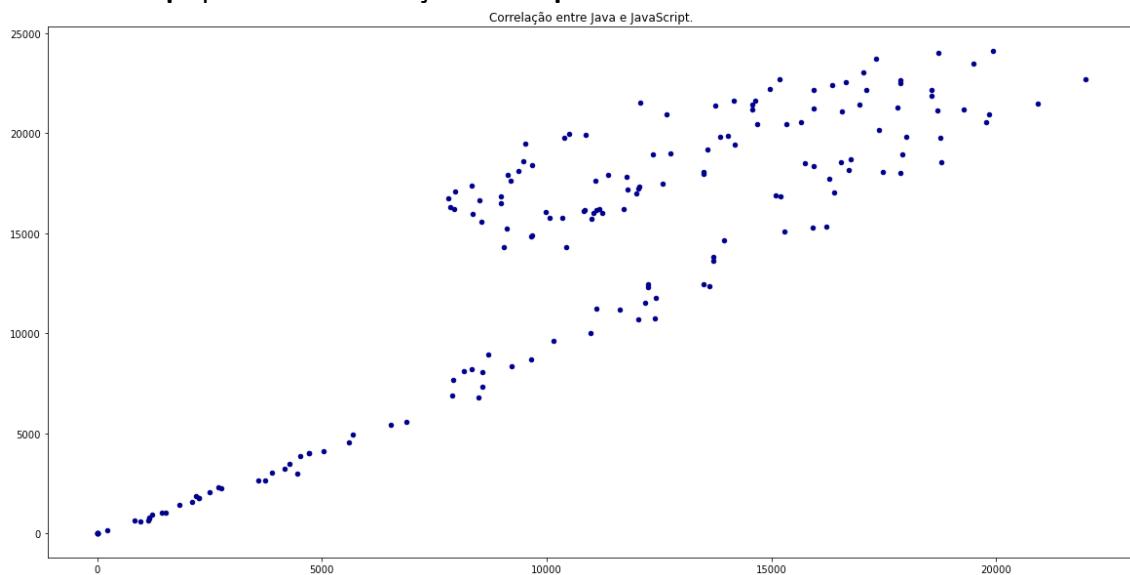
```

E o resultado apresentado foi bem melhor que o apresentado pelo dataset dos dados do GitHub de 2017, podendo observar boas correlações positivas entre as linguagens, como visto na tabela abaixo:

Verificando a correlação Pearson entre os % de uso das linguagens: Java e JavaScript.

0.8963935274210103

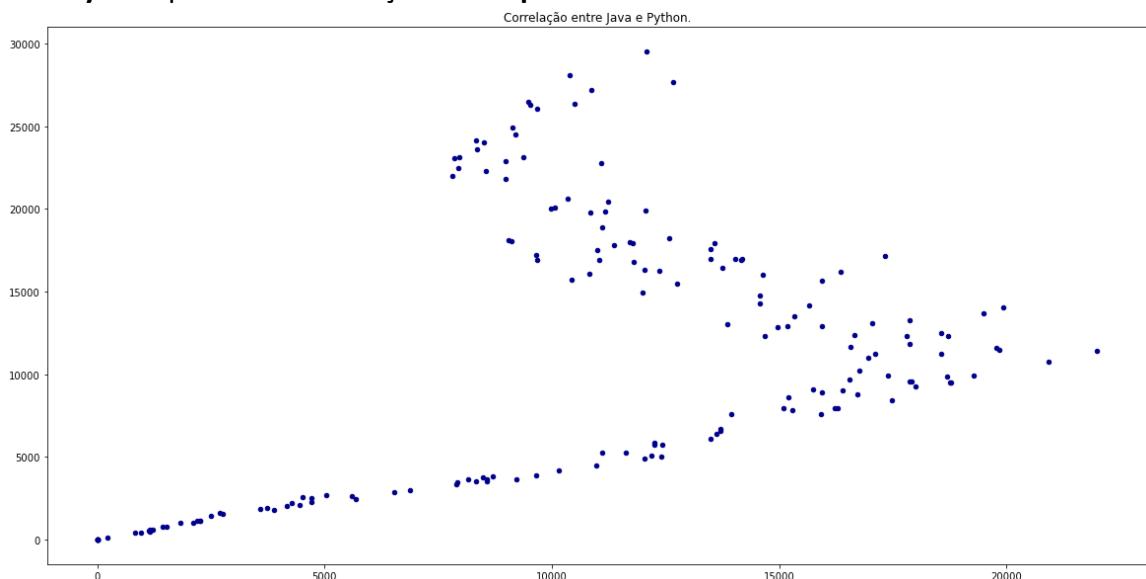
Java e JavaScript possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: Java e Python.

0.41358405693198974

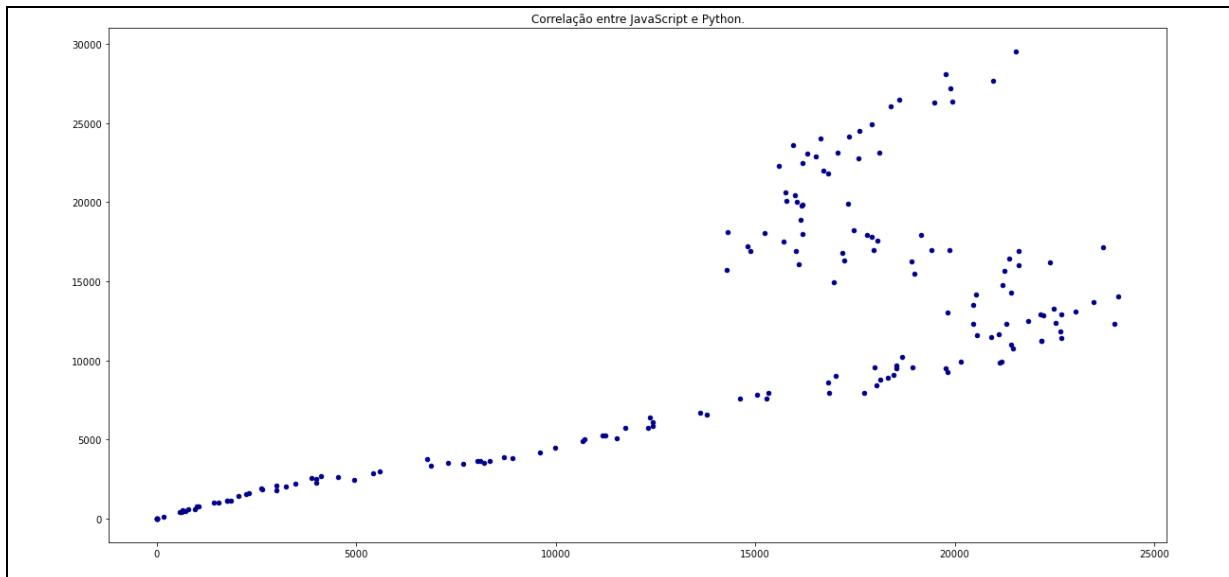
Java e Python possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: JavaScript e Python.

0.7456285733108394

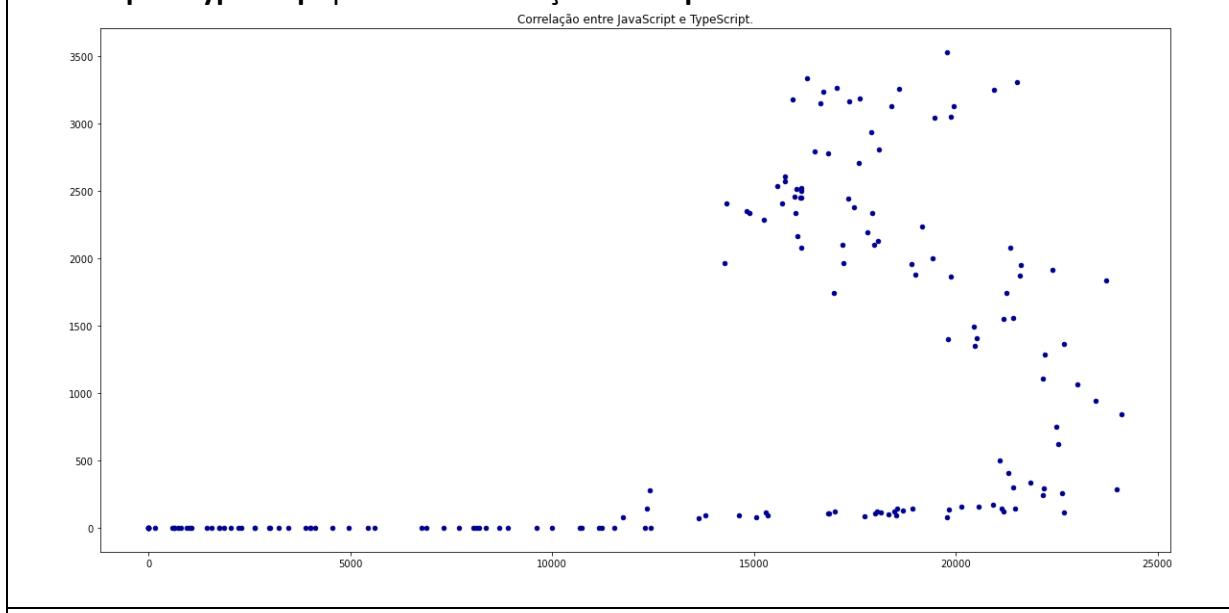
JavaScript e Python possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: JavaScript e TypeScript.

0.5078329711876486

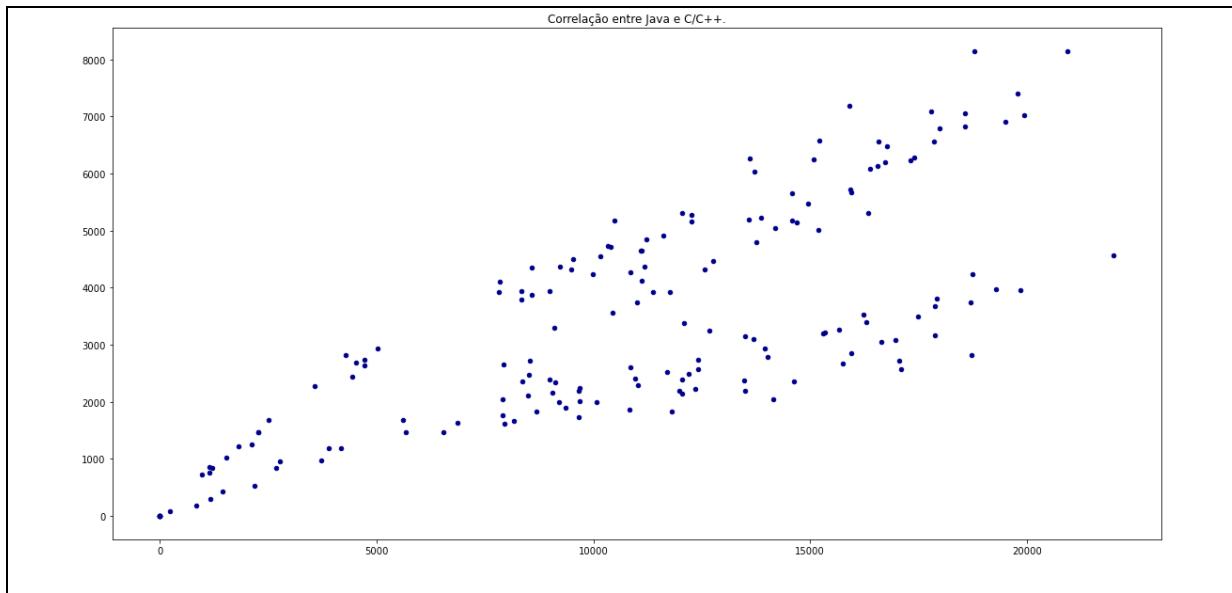
JavaScript e TypeScript possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: Java e C/C++.

0.7681292161842145

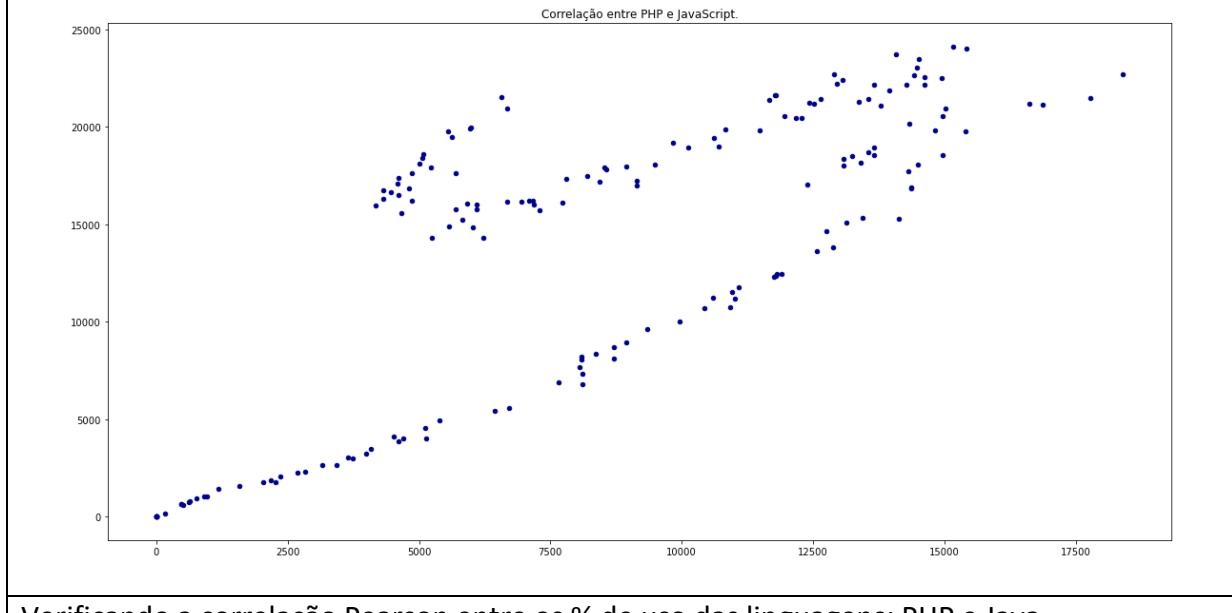
Java e C/C++ possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e JavaScript.

0.765888322106922

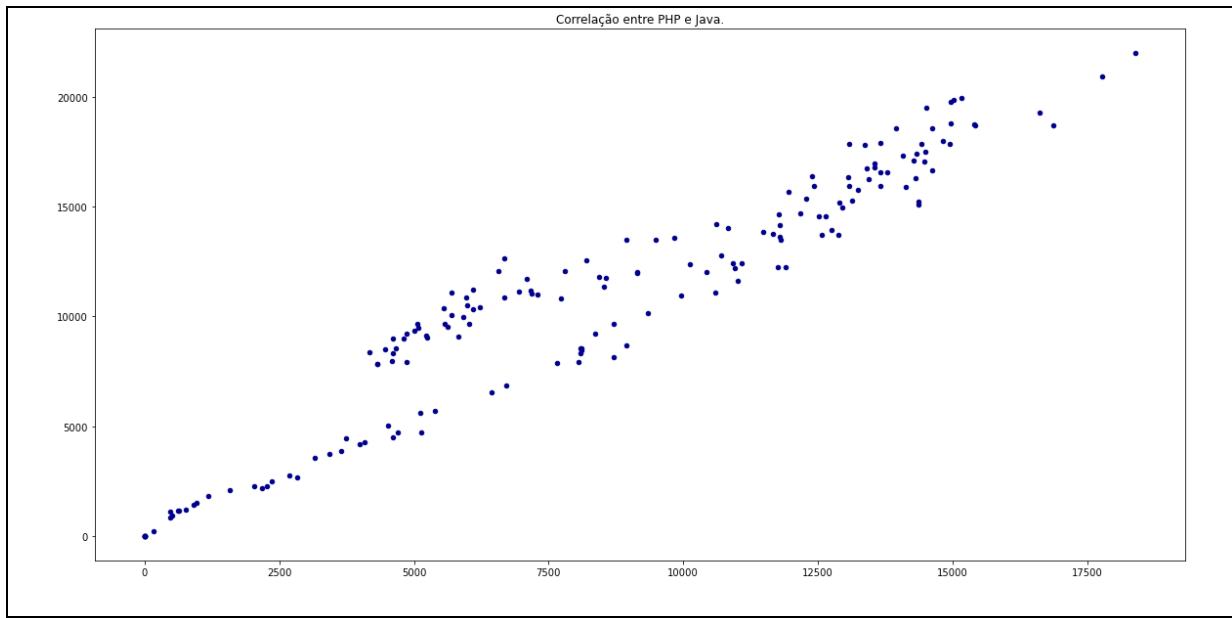
PHP e JavaScript possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e Java.

0.958116000705862

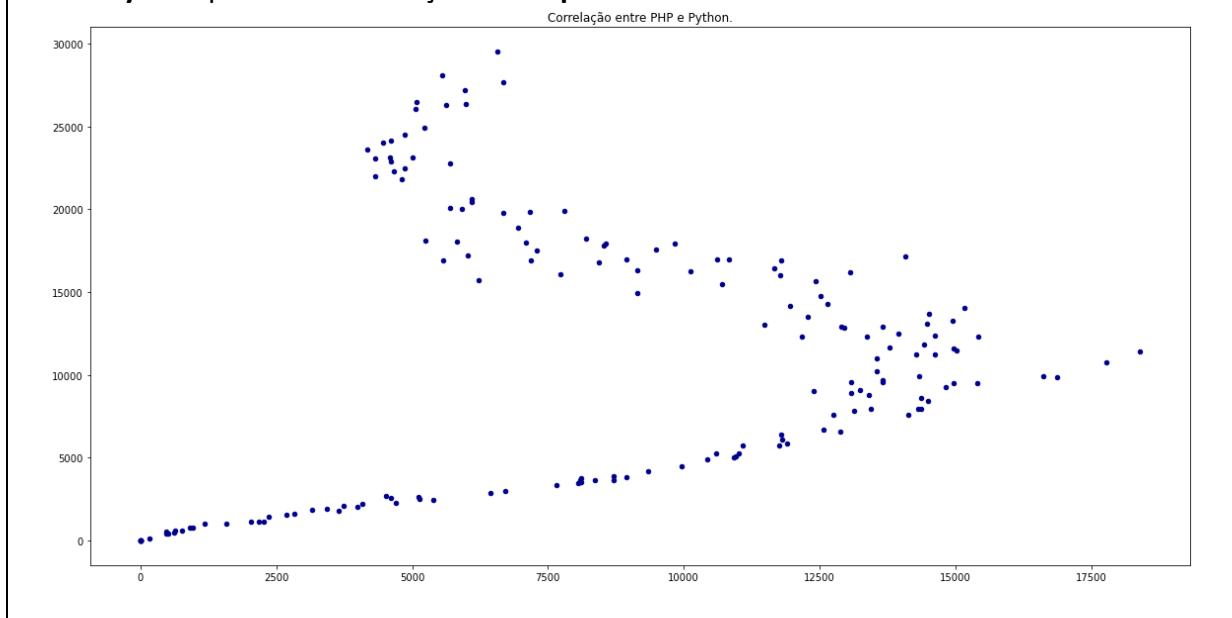
PHP e Java possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: PHP e Python.

0.172769203968479

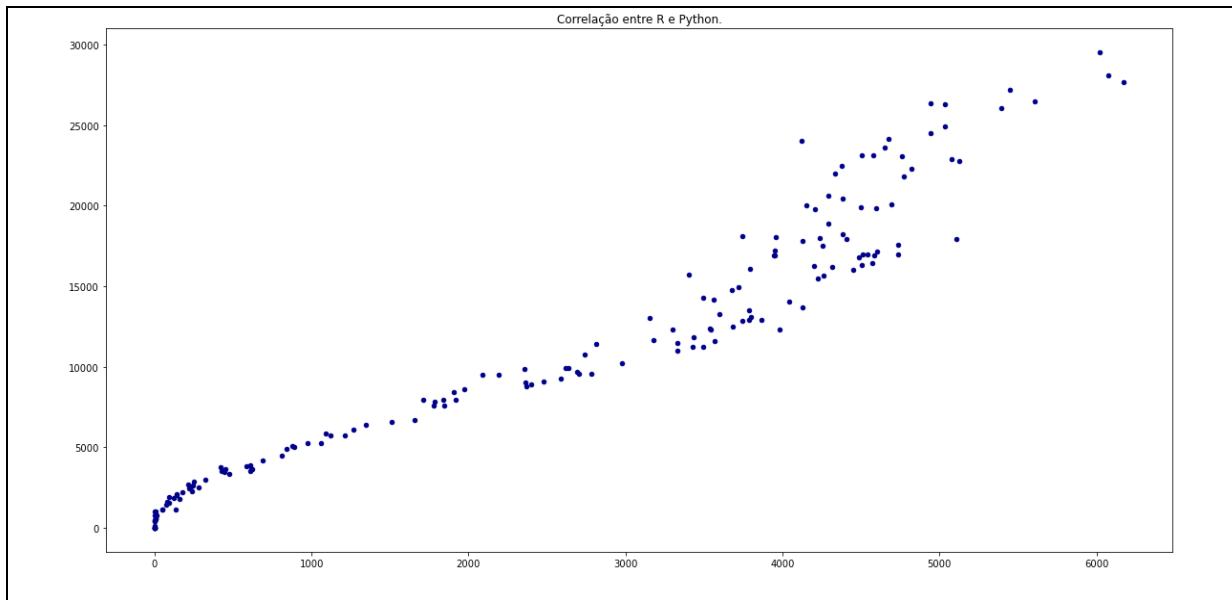
PHP e Python possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: R e Python.

0.9665637944817665

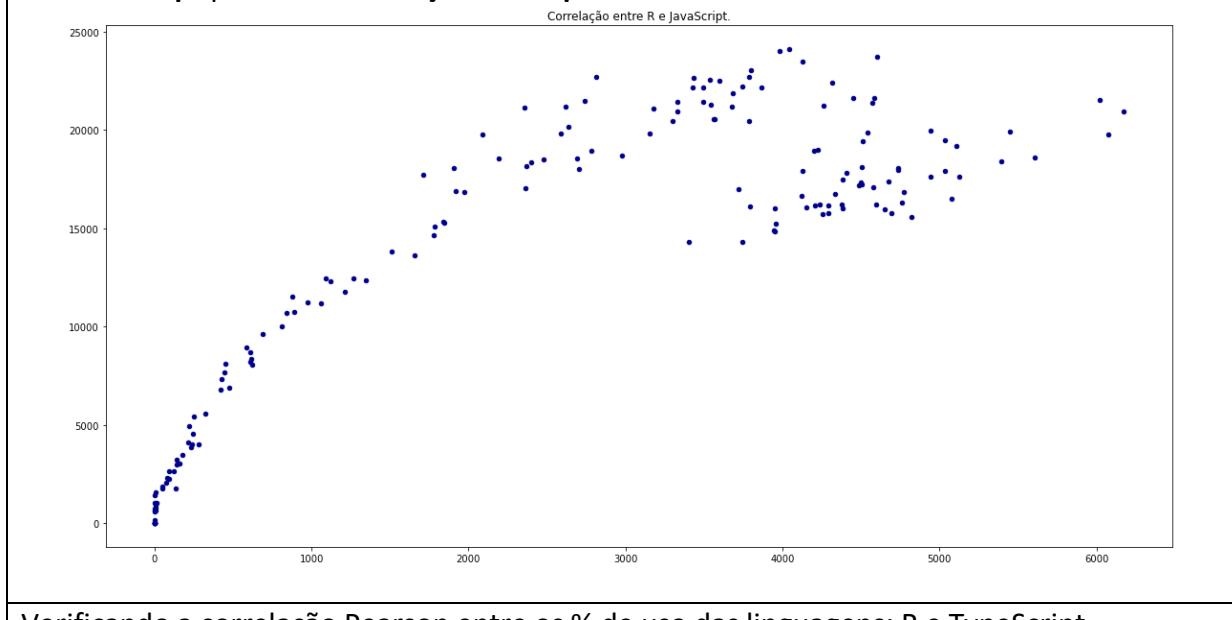
R e Python possuem correlação linear positiva.



Verificando a correlação Pearson entre os % de uso das linguagens: R e JavaScript.

0.846178398146117

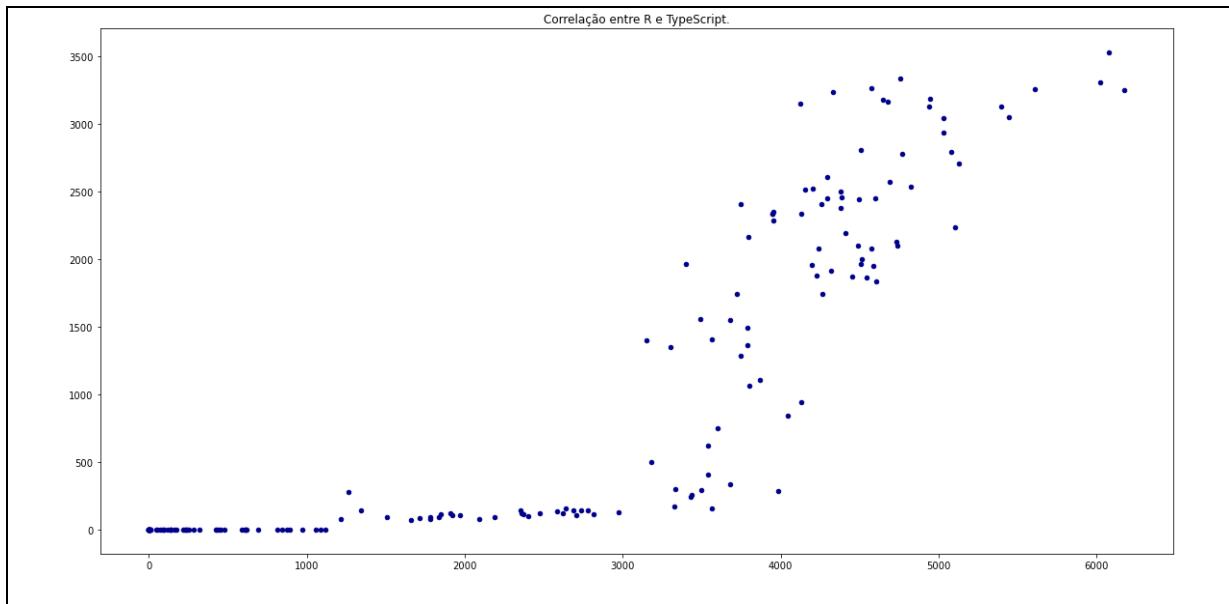
R e JavaScript possuem correlação **linear positiva**.



Verificando a correlação Pearson entre os % de uso das linguagens: R e TypeScript.

0.8671038832445993

R e TypeScript possuem correlação **linear positiva**.



Com base nesses resultados, é possível verificar que linguagens como Java e PHP possuem correlação positiva de 0,96, ou seja, bem próximo de uma correlação positiva perfeita. A correlação entre Java e Javascript também atingiu o esperado, ficando no valor de 0,89, e em contrapartida, Java e Python ficou em 0,41, mostrando exatamente a evolução do Python a partir de 2010 em relação ao Java.

É possível ver uma correlação entre R e Python, que foi de 0,97, quase perfeita também, exemplificando o aumento em seu uso principalmente por causa do crescimento na demanda por cientistas e engenheiros de dados.

Javascript e Typescript tiveram uma correlação baixa, embora positiva, de 0,50, e que explica-se pela demorada adoção da prática de tipagem forte dentro do ecossistema Javascript, uma vez que a maioria do uso do Typescript é para projetos voltados a back-end com plataformas como Node.js e principalmente um de seus frameworks mais recentes, o Nest.js, enquanto o Javascript sem tipagem encontra-se, em sua maioria, em aplicações front-end ou diretamente em códigos de script dentro de HTML até o momento.

5. Criação de Modelos de Machine Learning

5.1. Criando os modelos para os dados do GitHub de 2004 a 2021

A primeira análise a ser feita para os dados do dataset do GitHub é uma análise de Regressão Linear. Serão feitas duas análises, a primeira analisando duas linguagens, assim como na análise de correlação, e uma de uma linguagem específica pela série temporal, no caso, em UnixTime.

```
def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = df[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

    print('Score R2 para a regressão: {}'.format(reg.score(X_test, y_test)))
    plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
    plt.show()
```

O trecho acima estará criando uma função chamada *prever_regressao_linguagem* que receberá duas linguagens informadas por parâmetro, em seguida instanciará um modelo de treinamento e teste para uma regressão linear utilizando a classe *LinearRegression()* importada de *linear_model* na biblioteca Scikit-Learn.

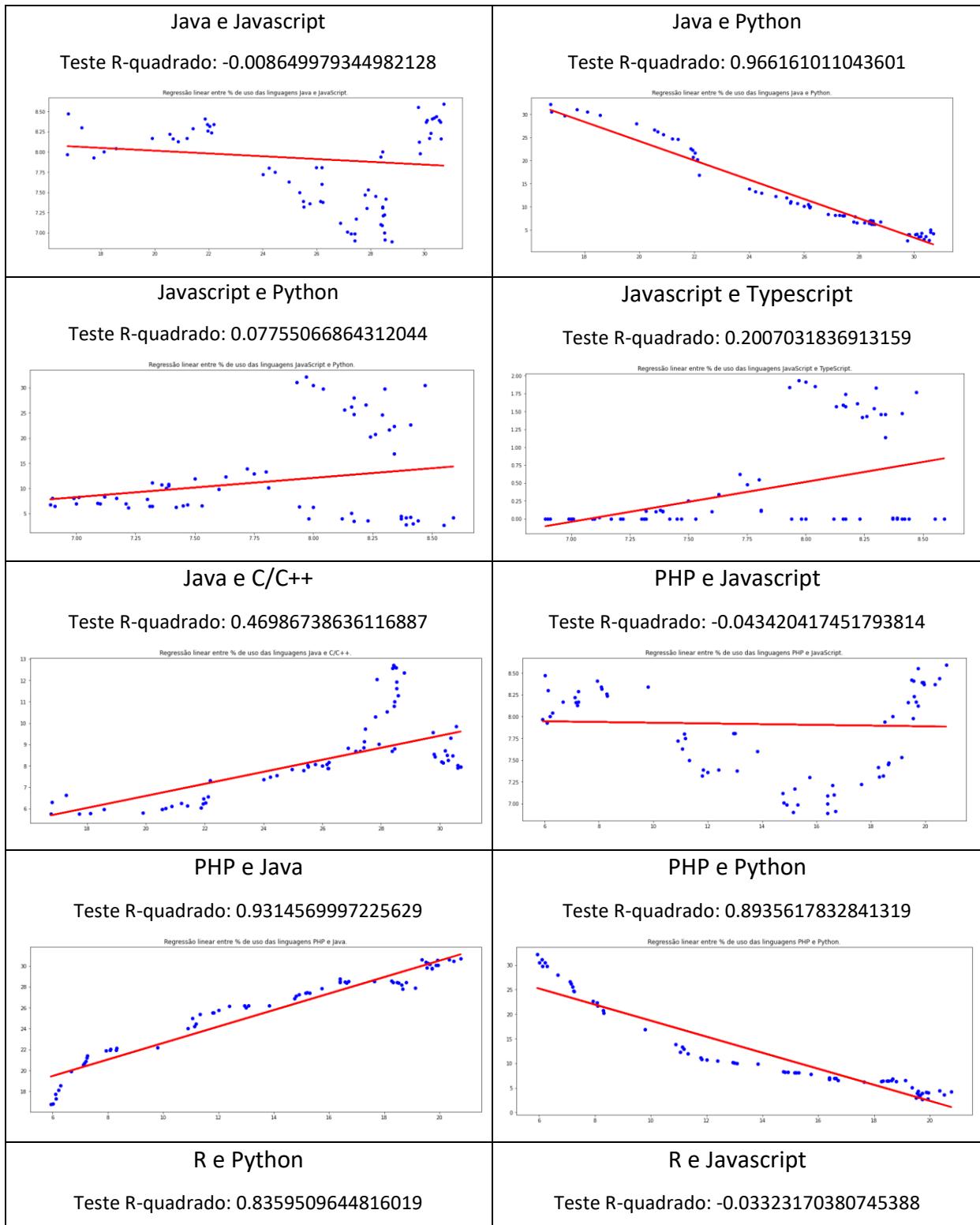
O valor de X será a primeira linguagem, e o de y a segunda informada. Primeiramente, será feito um *reshape* do Pandas Series para remodelar o dataset para a estrutura aceita na classe.

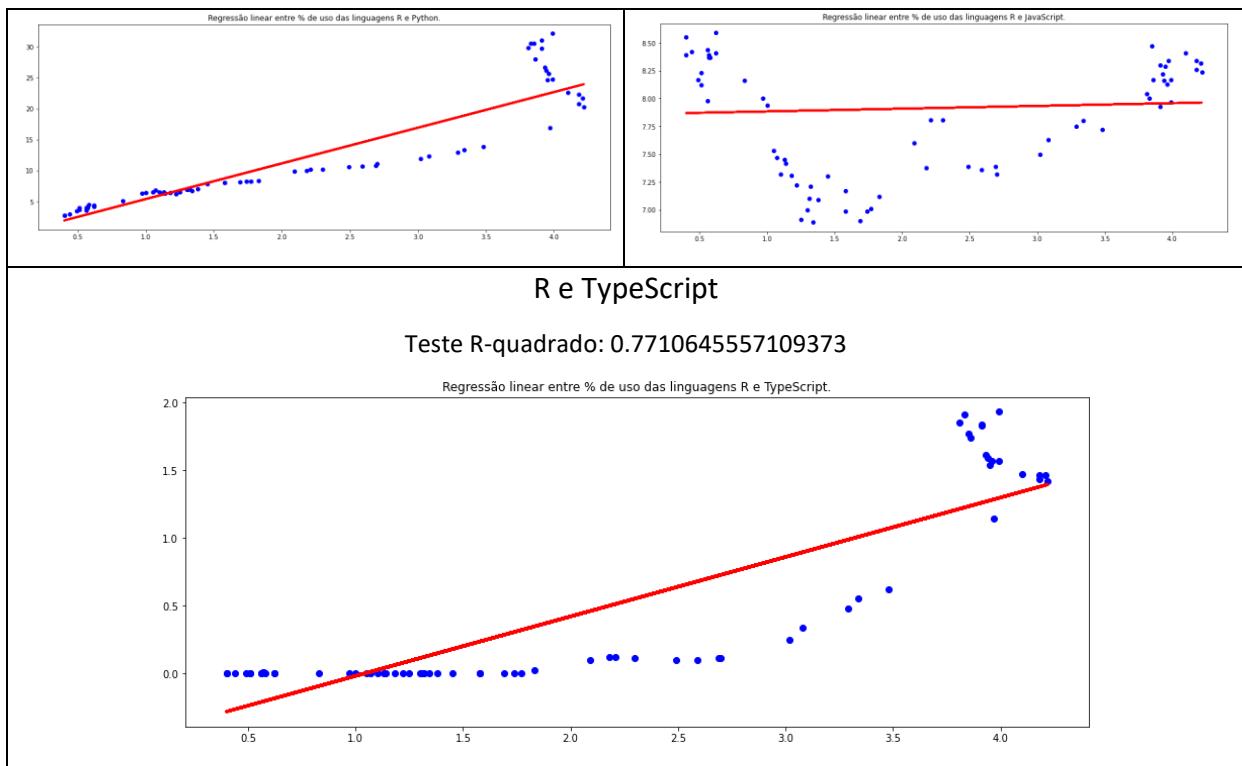
Os datasets serão divididos em 33% dos dados para teste com índice de 42% de estado aleatório de geração. Por fim, será feita a predição com a biblioteca, e será plotado o resultado da regressão linear. Será também informado o valor do teste R-quadrado, responsável por informar quão próximos os dados estão da linha de regressão, servindo como valor de avaliação do algoritmo executado.

```
for target in targets:
    prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])
```

O trecho acima apenas irá iterar a primeira variável *target* que foi criada comparando as linguagens chamando a função *prever_regressao_linguagem* desenvolvida.

O resultado será exibido na tabela abaixo:





Conforme já visto na análise de correlação, foi possível extrair resultados interessantes principalmente do comportamento da regressão e do valor beta para as análises entre Python e R, com o score R-quadrado de 83%, Java e Python com score de 96%, e PHP e Java com 93%, com uma boa linha reta traçada nos pontos acompanhando o que já havia sido analisado anteriormente sobre o aumento e decaimento no uso dessas tecnologias e seus respectivos motivos relacionados à tipagem, complexidade e curva de aprendizado.

A próxima análise de regressão será comparando apenas uma linguagem, sendo esta a variável y , com o target que é o UnixTime. A função será bem parecida, mudando apenas alguns parâmetros, como visto abaixo:

```

def prever_regressao_linguagem(linguagem):
    df_linguagem = dados_tratados[dados_tratados['Language'] == linguagem]

    X = df_linguagem.UnixTime.values.reshape(-1, 1)
    y = df_linguagem.Value.values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

    print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

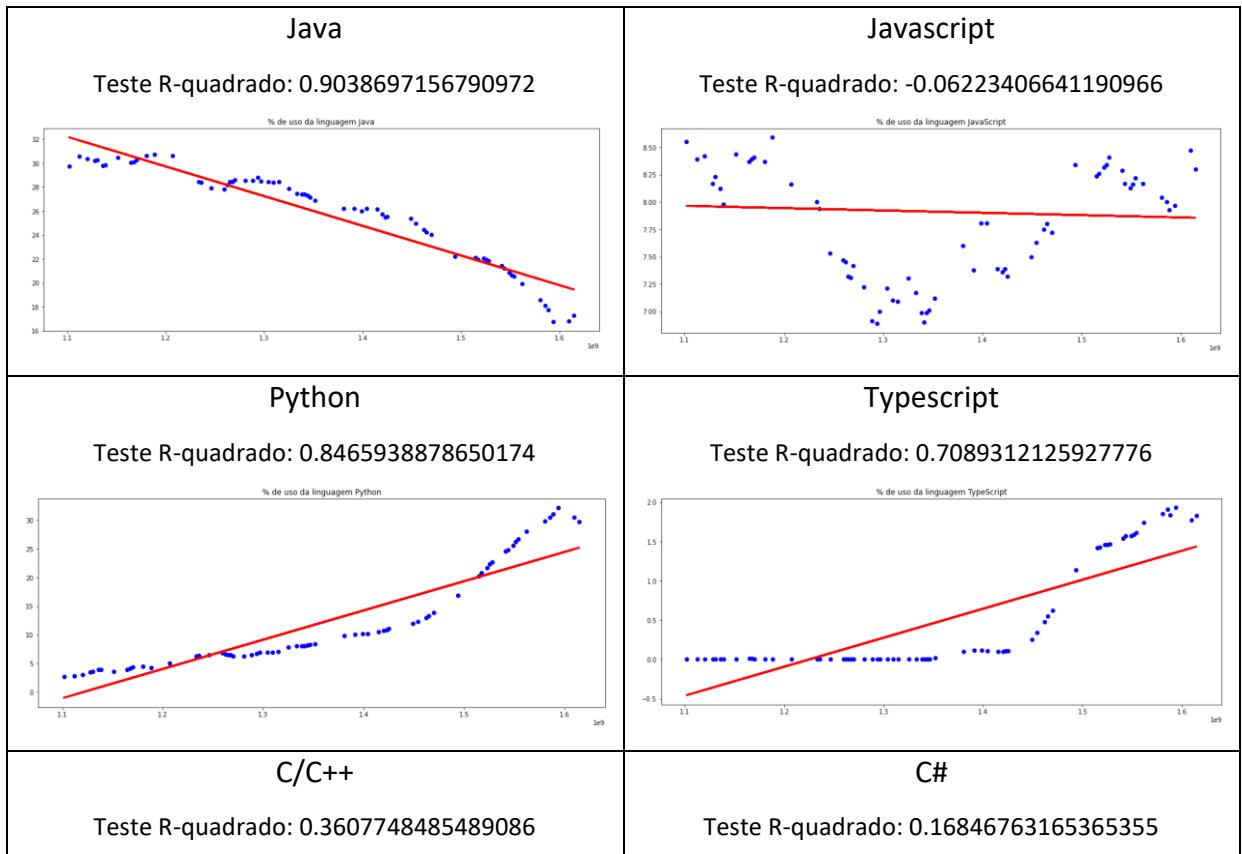
    plt.title('% de uso da linguagem {}'.format(linguagem))

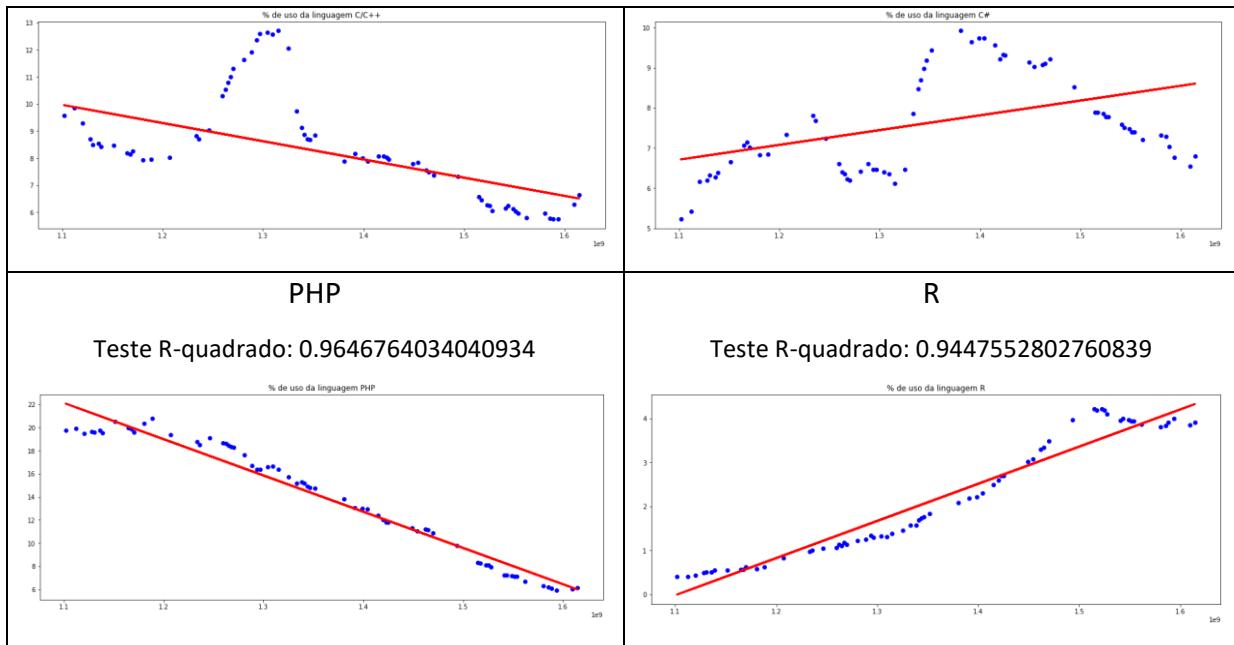
    plt.show()

prever_regressao_linguagem('Java')
prever_regressao_linguagem('JavaScript')
prever_regressao_linguagem('Python')
prever_regressao_linguagem('TypeScript')
prever_regressao_linguagem('C/C++')
prever_regressao_linguagem('C#')
prever_regressao_linguagem('PHP')
prever_regressao_linguagem('R')

```

O resultado será exibido na tabela abaixo:





Os resultados apontam curvas e retas da regressão bem mais assertivas quando comparadas com a série temporal do que quando comparadas às próprias linguagens, podendo ver o algoritmo acompanhar na reta um bom valor de predição para o aumento ou declínio das linguagens.

Linguagens como Javascript, C/C++ e C# possuíram o pior desempenho, isso devido à variação que tiveram conforme o passar dos anos, ora aumentando, ora diminuindo, não mantendo uma consistência entre aumento ou declínio igual a outras linguagens como Java e PHP (declínio) ou Python e R (aumento).

É possível verificar o declínio de linguagens compiladas com tipagem forte como C/C++ e Java. O PHP é uma linguagem interpretada com tipagem fraca e orientada a objetos que possui uma grande queda quando analisada, e o maior fator é sua limitação e falta de estrutura quando comparada às linguagens mais novas com as mesmas características. O PHP é uma linguagem que surgiu em 1995 e foi e ainda é extremamente utilizada para backend de sites e aplicações, tem seu grande uso pela facilidade em instalação, implementação e implantação, principalmente por integração entre HTML, formulários e acesso a bancos de dados.

O PHP teve um grande declínio devido à falta de recursos presentes em outras linguagens, assim como sua sintaxe robusta e competindo diretamente a outras linguagens mais versáteis e que atendem vários outros propósitos. Porém, não é justo dizer que o PHP é uma linguagem que está morrendo quando cerca de mais de 70% dos materiais da internet

ainda são escritos em PHP, embora não com versões em suas últimas atualizações, sendo a maioria código legado, como por exemplo, blogs em Wordpress.

É possível verificar também que o algoritmo falha ao tentar traçar uma reta de predição para o C#, ficando em uma reta positiva, porém, verificando a análise de correlação ao tempo, ela teve um grande aumento entre 2005 e 2010, e começou a ter um declínio em uso a partir de 2015 e não aumentou mais até o final de 2021.

Posteriormente serão realizadas análises com outros algoritmos, quando forem utilizados os dados deste dataset integrados ao do StackOverflow, na seção 5.4.

5.2. Criando o modelo para os dados do GitHub de 2017

Para os dados do GitHub de 2017, a única análise realizada será a de regressão, pois os dados agrupados possuem apenas informações agrupadas em poucas linhas, e no dataset original, as linguagens serão colunas, que serão úteis apenas para análise de regressão.

```
def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = dados_2017[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

    print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

    plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
    plt.ylabel('% de uso')

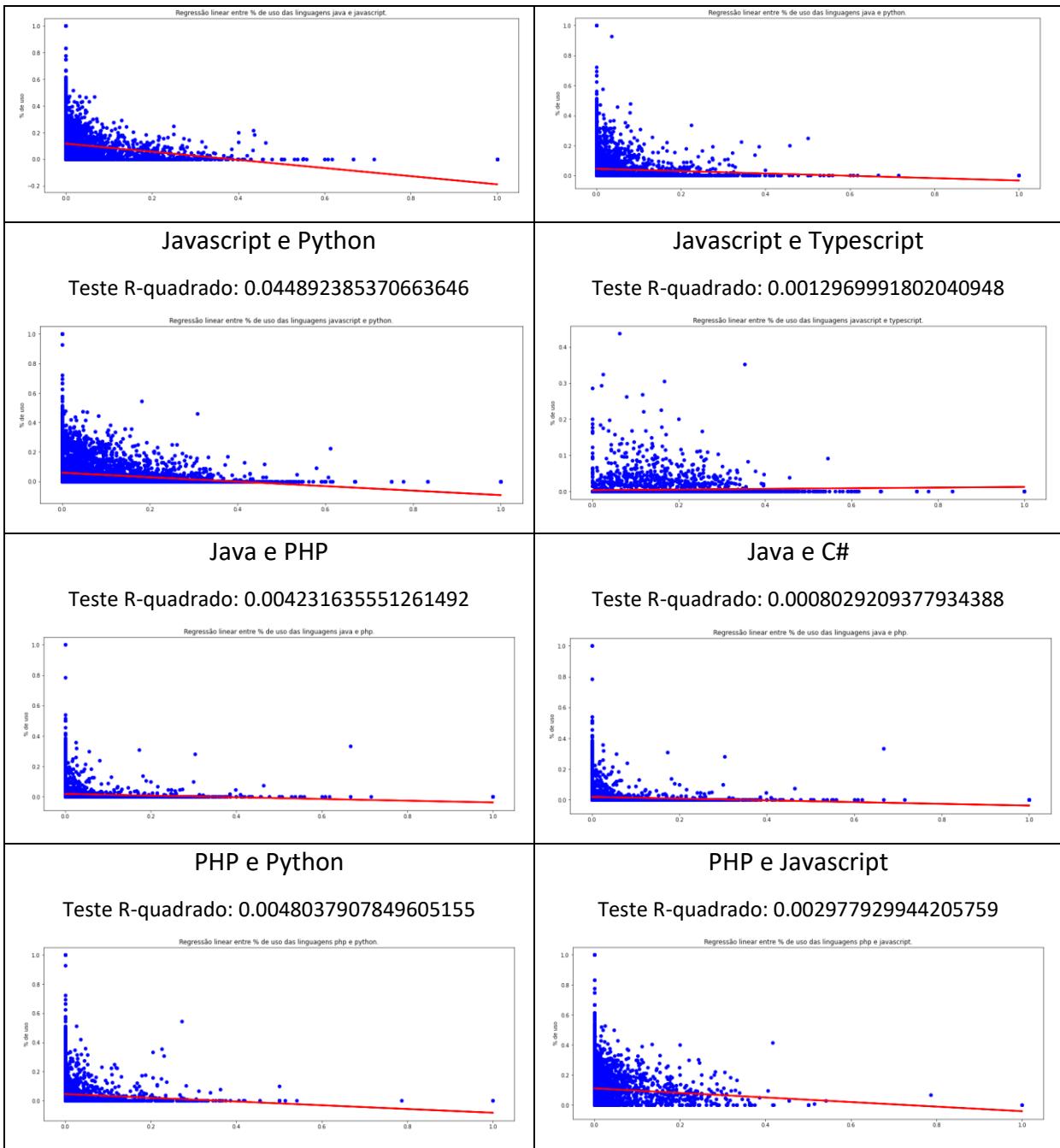
    plt.show()

for target in targets:
    prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])
```

A análise será parecida com as regressões da seção 5.1, porém, comparando duas variáveis através de uma lista chamada *target*, que já foi mostrada para este dataset na seção 4.2.

Os resultados estarão na tabela abaixo:

Java e Javascript	Java e Python
Teste R-quadrado: 0.038984259093661544	Teste R-quadrado: 0.00450242470486617



É possível verificar lá na seção 4.2 que as correlações entre as linguagens, embora existam positiva ou negativamente, eram muito baixas, próximas a 0, então com a aplicação de modelos de regressão linear, é possível verificar que o algoritmo tenta cruzar a linha de previsão para os valores de beta, porém, acaba falhando pela falta de padrões entre as variáveis.

Os melhores valores do teste R-quadrado também não conseguem um nível aceitável para uma análise de regressão para este dataset, mostrando que embora seja possível classificar o ranking das linguagens mais utilizadas, e que esse ranking esteja em

concordância com os rankings estudados anteriormente pelos outros datasets com séries temporais, não é possível tirar mais informações deste dataset para o objetivo de estudo proposto, sendo essa a finalização do uso deste dataset.

5.3. Criando o modelo para os dados do StackOverflow de 2008 a 2021

Para os dados do StackOverflow, será feito uma análise de regressão linear entre as linguagens. Para isso, foi criada uma função exatamente igual das seções anteriores para prever a regressão linear, porém, adaptada para o dataset em estudo, e a lista chamada *target* é a mesma utilizada para a análise de correlação do dataset do StackOverflow visto na seção 4.3:

```
def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = correlation_df[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

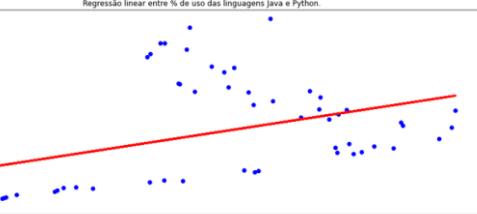
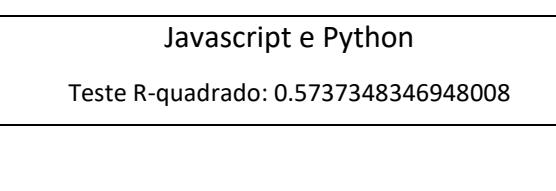
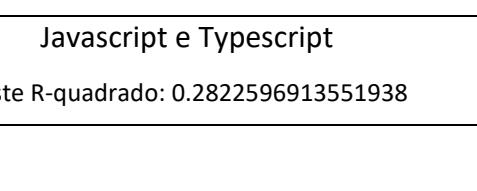
    plt.gcf().set_size_inches(16, 6)

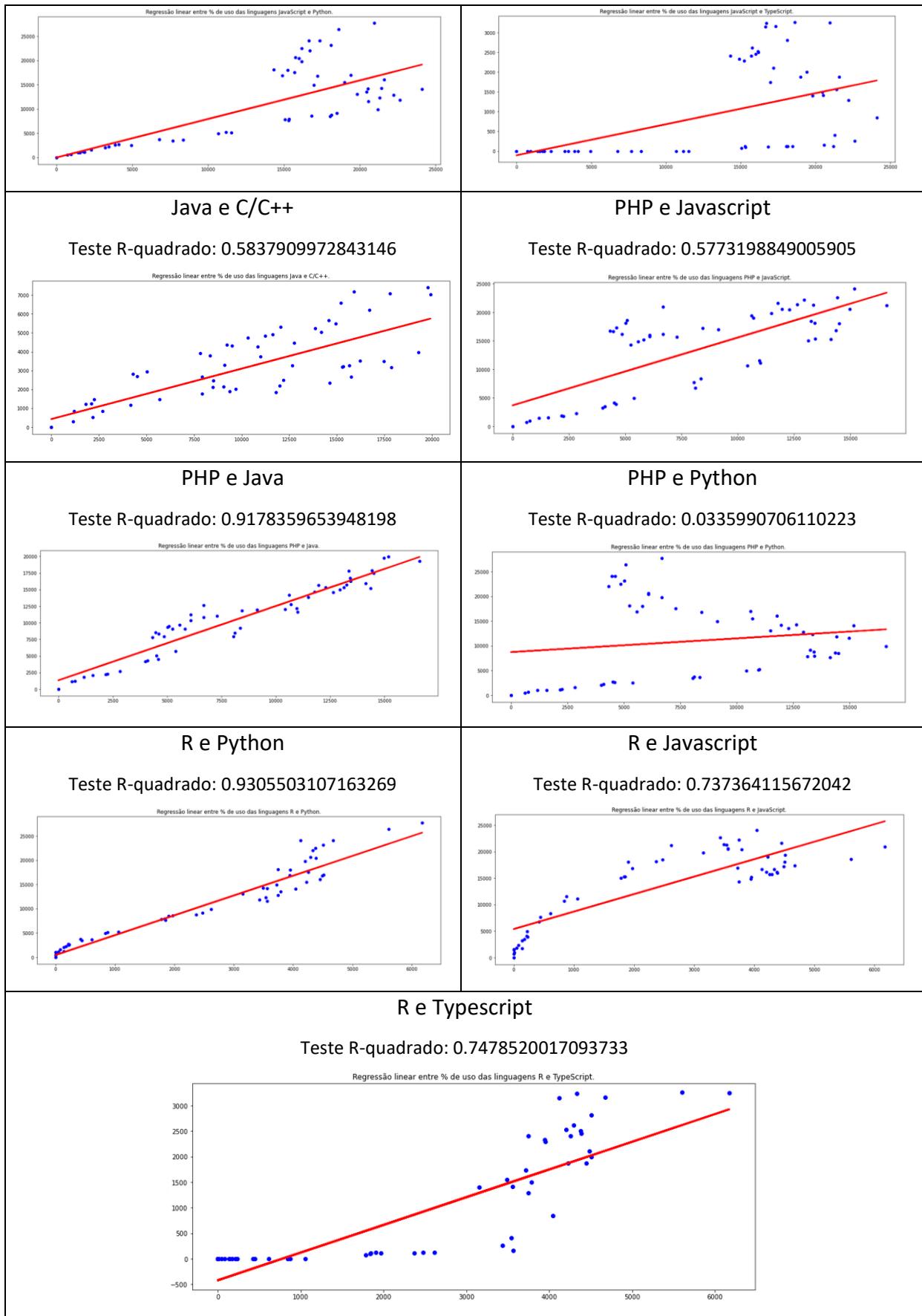
    plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
    plt.ylabel('% de uso')

    plt.show()

for target in targets:
    prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])
```

O resultado dos gráficos de regressão exibidos pode ser visualizado na seguinte tabela:

Java e Javascript Teste R-quadrado: 0.7975881591919097 	Java e Python Teste R-quadrado: 0.18181762794145873 
Javascript e Python Teste R-quadrado: 0.5737348346948008 	Javascript e Typescript Teste R-quadrado: 0.2822596913551938 



Conforme analisado anteriormente, a correlação entre as variáveis de linguagens neste dataset era muito boa, perto de perfeita, e, com isso, gerando uma boa análise de predição do algoritmo de regressão linear, principalmente nas linguagens Java e PHP, R e Python, Java e Javascript, e Java e C/C++.

A partir desta análise, levando em consideração os resultados dos testes R-quadrado, será realizada uma comparação abaixo dos resultados encontrados nos 2 datasets, GitHub e StackOverflow:

Linguagens comparadas	R-quadrado GitHub	R-quadrado StackOverflow
Java e Javascript	-0.008649979344982128	0.7975881591919097
Java e Python	0.966161011043601	0.18181762794145873
Javascript e Python	0.07755066864312044	0.5737348346948008
Javascript e Typescript	0.2007031836913159	0.2822596913551938
Java e C/C++	0.46986738636116887	0.5837909972843146
PHP e Javascript	-0.043420417451793814	0.5773198849005905
PHP e Java	0.9314569997225629	0.9178359653948198
PHP e Python	0.8935617832841319	0.0335990706110223
R e Python	0.8359509644816019	0.9305503107163269
R e Javascript	-0.03323170380745388	0.737364115672042
R e Typescript	0.7710645557109373	0.7478520017093733

Analizando a tabela acima, é possível ver que o modelo preditivo consegue acertar em algumas linguagens com valores de R-quadrado bem parecidos, como Javascript e Typescript, Java e C/C++, Java e PHP, R e Python e, por fim, R e Typescript.

Com base nesses dados, é possível analisar algoritmos de regressão para o dataset combinado, e, posteriormente, realizar a criação de modelos utilizando algoritmos de classificação como K-Vizinhos Mais Próximos (KNN), Máquina de Vetores de Suporte (SVM) e Naive Bayes.

5.4. Criando os modelos para o dataset integrado entre GitHub e StackOverflow

5.4.1. Regressão Linear

A primeira análise a ser feita no dataset integrado é uma regressão entre os valores X e Y que foram definidos através do merge dos datasets, sendo o valor de X os dados do GitHub e o valor de Y os valores do StackOverflow.

Para isso, foi criada uma nova função com o nome *prever_regressao_linguagem* que recebe apenas um parâmetro, o da linguagem desejada. A regressão então é feita sobre as colunas Value_x e Value_y.

```
def prever_regressao_linguagem(linguagem):
    df_linguagem = merge[merge['Language'] == linguagem]

    X = df_linguagem['Value_x'].values.reshape(-1, 1)
    y = df_linguagem['Value_y'].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

    print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

    plt.ylabel('Valor StackOverflow')
    plt.xlabel('Valor GitHub')
    plt.title('Regressão linear entre os valores do StackOverflow e do % uso do Github para linguagem {}'.format(linguagem))

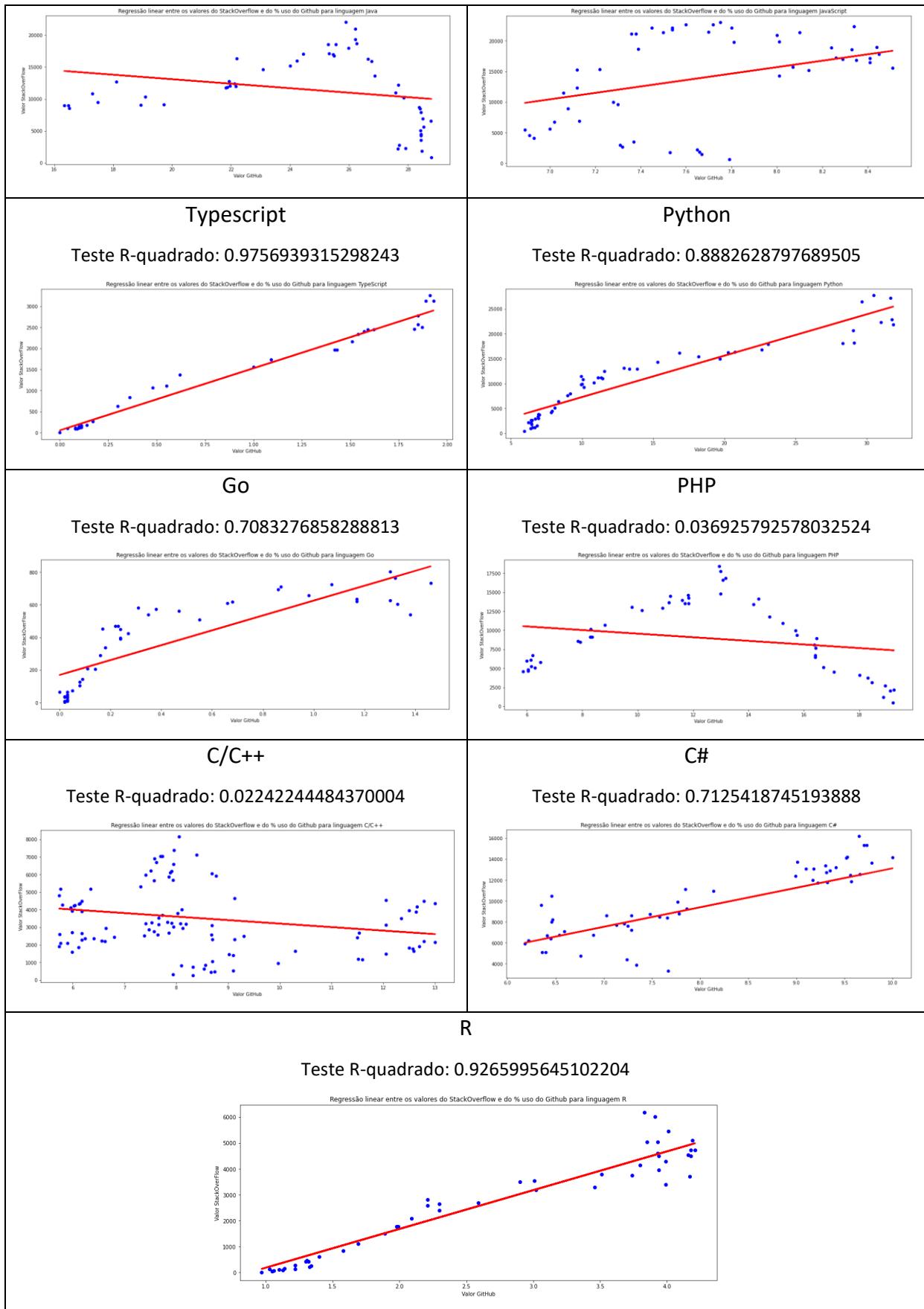
    plt.show()
```

Em seguida, é feita a chamada da função para todas as 9 linguagens que estão disponibilizadas nesse dataset integrado:

```
prever_regressao_linguagem('Java')
prever_regressao_linguagem('JavaScript')
prever_regressao_linguagem('TypeScript')
prever_regressao_linguagem('Python')
prever_regressao_linguagem('Go')
prever_regressao_linguagem('PHP')
prever_regressao_linguagem('C/C++')
prever_regressao_linguagem('C#')
prever_regressao_linguagem('R')
```

Os resultados das regressões podem ser vistos na tabela abaixo:

Java	Javascript
Teste R-quadrado: 0.026578550204984785	Teste R-quadrado: 0.2043674365528898



Como pode ser visto, algumas linguagens tiveram um valor de regressão muito próximo ao integrar ambos os datasets, como pode ser visto com Python, tendo o R-quadrado de 0,88, o Typescript com 0,97, o R com 0,92 e outras que não ficaram na faixa de 0,8 ou 0,9, mas também possuíram valores próximos, como C# com 0,71 e Go com 0,70.

É possível verificar que linguagens como Java e C/C++, PHP, e Javascript foram linguagens que foi possível verificar uma grande variação conforme os anos, e quando colocados juntos na regressão entre os dois datasets, não foi possível de se obter um valor satisfatório, muito provavelmente resultante dessa variação das variáveis, tendo assim pouca correlação entre si.

5.4.2. KNN

Agora, será feita a criação de uma instância do algoritmo do K-Vizinhos Próximos para os valores do GitHub do dataset integrado utilizando a classe *KNeighborsClassifier* da biblioteca Scikit-Learn.

A variável X será o valor do GitHub na coluna Value_x, e a variável y será a coluna Language. A divisão de percentuais de treino e teste será de 33% com 42 de estado aleatório, assim como nos algoritmos de regressão linear. Será realizado o algoritmo com uma quantidade de 3 vizinhos próximos.

```
X = merge.Value_x.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
y_pred
```

Logo em seguida, será mostrado o resultado da variável *y_pred*, uma lista contendo valores de previsão das linguagens a serem definidas pelo algoritmo KNN a partir dos dados de treinamento.

```
array(['C/C++', 'C/C++', 'Go', 'C/C++', 'C#', 'Go', 'C/C++', 'C/C++',
       'JavaScript', 'PHP', 'Java', 'JavaScript', 'Java', 'C#', 'R',
       'C/C++', 'C#', 'JavaScript', 'C/C++', 'C/C++', 'C/C++',
       'JavaScript', 'Java', 'C/C++', 'Python', 'C#', 'C/C++', 'Java',
       'C/C++', 'PHP', 'R', 'C/C++', 'R', 'TypeScript', 'Java', 'Java',
       'C#', 'JavaScript', 'C/C++', 'TypeScript', 'C/C++', 'JavaScript',
       'C/C++', 'TypeScript', 'C#', 'C/C++', 'PHP', 'Go', 'Go', 'C/C++',
       'C/C++', 'C/C++', 'C/C++', 'PHP', 'Java', 'C/C++', 'C/C++', 'C#',
       'Go', 'C/C++', 'R', 'C/C++', 'C#', 'R', 'R', 'C/C++', 'C#',
       'C/C++', 'C/C++', 'C/C++', 'R', 'C#', 'Go', 'PHP', 'C/C++',
       'C/C++', 'C/C++', 'R', 'C/C++', 'R', 'JavaScript', 'Go', 'C/C++',
       'Go', 'JavaScript', 'TypeScript', 'C/C++', 'Go', 'Go', 'C/C++',
```

```
'C/C++', 'C#', 'Go', 'C#', 'JavaScript', 'C/C++', 'C/C++', 'C#',
'Java', 'C/C++', 'C/C++', 'PHP', 'TypeScript', 'JavaScript', 'PHP',
'R', 'C#', 'R', 'C/C++', 'C/C++', 'Java', 'PHP', 'Java', 'C#',
'JavaScript', 'C/C++', 'C/C++', 'C/C++', 'Java', 'C#', 'Go',
>TypeScript', 'Java', 'JavaScript', 'R', 'C/C++', 'JavaScript',
'C/C++', 'Go', 'C/C++', 'C/C++', 'R', 'C/C++', 'Java', 'C/C++',
'PHP', 'C/C++', 'C#', 'C/C++', 'Java', 'PHP', 'Python', 'PHP',
'C/C++', 'R', 'C/C++', 'C/C++', 'Java', 'C/C++', 'C/C++', 'C/C++',
'C#', 'C/C++', 'C/C++', 'C/C++', 'C#', 'Java', 'C/C++', 'C/C++',
'Python', 'Java', 'JavaScript', 'Java', 'C#', 'Python', 'Java',
'Python', 'Java', 'C/C++', 'C#', 'C/C++', 'R', 'C/C++', 'Python',
'JavaScript', 'PHP', 'C#', 'C/C++', 'TypeScript', 'PHP', 'R',
'C/C++', 'C#', 'TypeScript', 'C/C++', 'JavaScript', 'PHP', 'C#',
'Java', 'PHP', 'Java', 'C/C++', 'Java', 'R', 'JavaScript', 'Go',
'JavaScript', 'Go', 'PHP', 'JavaScript', 'Go', 'Python', 'Java',
'Go', 'Go', 'C#', 'C/C++', 'Java', 'Go', 'C#', 'JavaScript', 'C#',
'PHP', 'C#', 'C/C++', 'C/C++', 'C#', 'Go', 'C/C++', 'JavaScript',
'C/C++', 'Go', 'C/C++', 'R', 'Go', 'PHP', 'C/C++', 'C/C++',
'C/C++', 'PHP', 'Python', 'Go', 'Java', 'R', 'R',
>TypeScript', 'PHP', 'PHP', 'Go', 'C/C++', 'PHP', 'R', 'PHP',
>TypeScript', 'Go', 'C/C++', 'C#', 'JavaScript', 'TypeScript',
'JavaScript', 'Go', 'C/C++', 'C/C++', 'Python', 'Python', 'Python',
'Java', 'PHP', 'Java', 'Java', 'JavaScript', 'C/C++', 'Python',
'C#', 'Java', 'C#', 'C/C++', 'C/C++', 'R', 'C/C++', 'Go', 'R',
'C#', 'R', 'PHP', 'Java', 'Go', 'R', 'Java', 'Go', 'C/C++', 'Java',
'C#', 'C/C++', 'R', 'C/C++', 'R', 'C#', 'Go', 'Go', 'PHP',
>TypeScript', 'JavaScript', 'Go', 'PHP', 'C/C++', 'C#', 'Java',
'R', 'Java', 'R', 'PHP', 'TypeScript', 'C/C++', 'Java',
>TypeScript', 'Go', 'C/C++', 'Go', 'Java', 'C#', 'C/C++', 'PHP',
'Go', 'C/C++', 'JavaScript', 'R', 'C/C++', 'Python', 'Java',
'C/C++', 'Go', 'C/C++', 'PHP', 'C#', 'Go', 'JavaScript', 'Java',
'C#', 'Go', 'R', 'Java', 'C/C++', 'C#', 'C#', 'PHP', 'Go', 'Java',
'C/C++', 'Go', 'PHP', 'C/C++', 'PHP', 'C#', 'C#', 'R',
'JavaScript', 'R', 'TypeScript', 'Go', 'JavaScript', 'R',
>TypeScript', 'C/C++', 'Go', 'C/C++', 'PHP', 'Java', 'C/C++',
'JavaScript', 'C/C++', 'Go', 'Go', 'Java', 'C/C++', 'JavaScript',
'Go', 'C#', 'R', 'Python', 'TypeScript', 'Go', 'PHP', 'C/C++',
'Python', 'C/C++', 'Go', 'Go', 'C#', 'Java', 'C/C++', 'JavaScript',
'C/C++', 'JavaScript', 'Java', 'C/C++', 'R', 'C/C++', 'Go',
'JavaScript', 'Java', 'C#', 'C/C++', 'Go', 'Java', 'JavaScript',
'Python', 'C/C++', 'Go', 'Java', 'C/C++', 'Java', 'C/C++',
'C#', 'TypeScript', 'PHP', 'JavaScript', 'C/C++', 'C/C++', 'Go',
'R', 'C#', 'Java', 'Java', 'C/C++', 'Java', 'C/C++', 'Java',
'Go', 'C/C++', 'R', 'TypeScript', 'C/C++', 'C/C++', 'PHP',
'C/C++', 'Go', 'C/C++', 'Go', 'R', 'Java', 'C/C++',
>TypeScript', 'C#', 'Go', 'C/C++', 'Go', 'Java', 'C/C++', 'Python',
'C/C++', 'Java', 'Go', 'C/C++', 'R', 'Java', 'C/C++',
>TypeScript', 'JavaScript', 'R', 'C/C++', 'R', 'C#', 'Go', 'R'],
dtype=object)
```

Logo em seguida, será verificada a probabilidade da lista ao predizer tais linguagens, e, também será feito o uso da classe *accuracy_score*, também da Scikit-Learn, que irá informar qual foi o percentual de acerto da predição realizada.

```
knn.predict_proba(X_test)

array([[0.          , 0.66666667, 0.          , ..., 0.          , 0.          ,
       0.          ],
       [0.          , 0.66666667, 0.          , ..., 0.          , 0.          ,
       0.          ],
       [0.          , 0.          , 0.33333333, ..., 0.          , 0.33333333,
       0.33333333],
       ...,
       [1.          , 0.          , 0.          , ..., 0.          , 0.          ,
       0.          ],
       [0.          , 0.          , 0.33333333, ..., 0.          , 0.33333333,
       0.33333333],
       [0.          , 0.          , 0.33333333, ..., 0.          , 0.66666667,
       0.          ]])

accuracy_score(y_test, y_pred)

0.5379876796714579
```

É possível verificar que para essa primeira análise, o algoritmo deu uma predição de 53,8% de acurácia, não sendo um valor muito alto, e que pode ser explicado também pelos valores nas regressões, que não tiveram, em sua grande parte, um valor de R-quadrado aceitável. Lembrando também que ainda não foi verificada a acurácia do algoritmo ao aplicar técnicas de normalização.

A seguir, utilizando a classe *classification_report*, que assim como *accuracy_score*, também é importada do módulo de métricas da Scikit-Learn, conseguiremos ter uma visualização em larga escala de precisão, recuperação, score-F1 e suporte para cada linguagem.

```
print(classification_report(y_test, y_pred))

           precision    recall  f1-score   support

          C#      0.00     0.00     0.00      57
        C/C++    0.36     0.95     0.52      94
          Go     0.62     0.96     0.75      50
         Java    0.63     0.91     0.74      46
      JavaScript    0.00     0.00     0.00      45
          PHP    0.55     0.40     0.46      55
         Python    1.00     0.03     0.07      59
          R     0.62     0.73     0.67      45
      TypeScript    0.00     0.00     0.00      36

  accuracy                           0.48      487
  macro avg       0.42     0.44     0.36      487
weighted avg     0.43     0.48     0.37      487
```

Com isso, conseguimos verificar que a precisão de Java, R, Go e Java foi a maior, assim como sua recuperação e score-F1. A linguagem Python destaca-se com 100% de precisão e baixa recuperação.

Abaixo será repetido o mesmo processo, porém, para os dados do StackOverflow:

```
X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.35523613963039014
          precision    recall  f1-score   support

        C#       0.32      0.44      0.37       57
      C/C++     0.35      0.69      0.47       94
        Go       0.74      0.70      0.72       50
       Java      0.20      0.28      0.23       46
  JavaScript     0.45      0.42      0.44       45
        PHP      0.31      0.09      0.14       55
      Python     0.18      0.05      0.08       59
        R       0.04      0.02      0.03       45
  TypeScript     0.47      0.19      0.27       36

   accuracy                           0.36      487
  macro avg       0.34      0.32      0.31      487
weighted avg       0.34      0.36      0.32      487
```

É possível verificar que o desempenho foi ainda mais baixo para os dados do StackOverflow, principalmente ao se visualizar a acurácia de previsão de 35,52% e que a maior precisão de acerto foi apenas para a linguagem Go, em comparação aos do GitHub que possuiu maior precisão em Java, Go e R.

5.4.3. Naive Bayes

A seguir, será realizada a instância do algoritmo de Naive Bayes, utilizando a classe *GaussianNB* da biblioteca Scikit-Learn, que possui uma estrutura bem parecida com a que foi vista no KNN. Em seguida, será printada a acurácia encontrada para as previsões deste modelo e também seu relatório de classificação.

```

X = merge.Value_x.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

naive_bayes = GaussianNB()

pred = naive_bayes.fit(X_train, y_train)

y_pred = pred.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

```

0.46406570841889117
      precision    recall  f1-score   support

       C#      0.00     0.00     0.00      57
     C/C++     0.37     0.56     0.44      94
        Go     0.66     0.70     0.68      50
       Java     0.63     0.91     0.74      46
  JavaScript     0.35     0.71     0.47      45
        PHP     0.45     0.36     0.40      55
      Python     0.12     0.02     0.03      59
        R     0.76     0.71     0.74      45
  TypeScript     0.31     0.31     0.31      36

  accuracy                           0.46      487
   macro avg     0.41     0.48     0.42      487
weighted avg     0.39     0.46     0.41      487

```

O que pode ser verificado é que a acurácia do modelo foi ainda menor para este algoritmo do que o KNN, indicando que a correlação durante a regressão tem impacto sobre a predição dos valores, e, no relatório de classificação, o Naive Bayes teve acertos em Go, Java e R também.

A seguir, é feita a mesma análise para os dados do StackOverflow:

```

X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

naive_bayes = GaussianNB()

pred = naive_bayes.fit(X_train, y_train)

y_pred = pred.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.4188911704312115
      precision    recall   f1-score   support
      C#       0.40     0.58     0.47      57
      C/C++    0.41     0.86     0.56      94
      Go       0.64     1.00     0.78      50
      Java      0.22     0.28     0.25      46
      JavaScript  0.44     0.53     0.48      45
      PHP       0.00     0.00     0.00      55
      Python    0.00     0.00     0.00      59
      R         0.00     0.00     0.00      45
      TypeScript 0.18     0.08     0.11      36
      accuracy           0.42      487
      macro avg       0.25     0.37     0.30      487
      weighted avg    0.27     0.42     0.32      487

```

Embora a precisão seja menor que verificado nos dados do GitHub, ela acaba sendo superior à do KNN quando se comparada a diferença de precisão em relação aos dados de ambos os datasets, e no relatório de classificação, apenas a linguagem Go teve precisão mais alta.

5.4.4. SVM

Por fim, será realizada a instância do algoritmo de Máquina de Vetores de Suporte (SVM), utilizando a classe *SVC* da biblioteca Scikit-Learn, e que também possui a mesma estrutura apresentada anteriormente.

```

X = merge.Value_x.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = svm.SVC()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

```

0.48459958932238195
      precision    recall   f1-score   support
      C#       0.00     0.00     0.00      57
      C/C++    0.36     0.95     0.52      94
      Go       0.62     0.96     0.75      50
      Java     0.63     0.91     0.74      46
      JavaScript  0.00     0.00     0.00      45
      PHP      0.55     0.40     0.46      55
      Python    1.00     0.03     0.07      59
      R        0.62     0.73     0.67      45
      TypeScript  0.00     0.00     0.00      36
      accuracy           0.48      487
      macro avg       0.42     0.44     0.36      487
      weighted avg    0.43     0.48     0.37      487

```

A acurácia do modelo SVM ficou em 48,46%, embora tenha tido um desempenho melhor que o Naive Bayes, continua inferior ao KNN e com uma acurácia indesejada. A precisão continua nas 3 linguagens analisadas previamente, Java, Go e R. E a linguagem Python destaca-se novamente com 100% de precisão e baixa recuperação.

Para os dados do StackOverflow, temos as seguintes análises com o algoritmo SVM:

```

X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = svm.SVC()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

```

0.42299794661190965
      precision    recall   f1-score   support
      C#       0.42     0.60     0.50      57
      C/C++    0.42     0.91     0.57      94
      Go       0.53     1.00     0.69      50
      Java     0.21     0.15     0.17      46
      JavaScript  0.45     0.56     0.50      45
      PHP      0.13     0.04     0.06      55
      Python    1.00     0.03     0.07      59
      R        0.00     0.00     0.00      45
      TypeScript  0.00     0.00     0.00      36
      accuracy           0.42      487
      macro avg       0.35     0.37     0.28      487
      weighted avg    0.38     0.42     0.32      487

```

Novamente, o Python destaca-se com 100% de precisão, sendo esta a primeira vez que a linguagem Python tem um valor alto para os dados do StackOverflow.

5.4.5. Prevendo os modelos com e sem normalização

Para facilitar e agilizar o processo das predições dos 3 algoritmos, foi criada uma função chamada *realizar_previsoes* que recebe 3 parâmetros, um informando qual o nome do algoritmo, aceitando os valores *knn*, *naive_bayes* e *svm*, um parâmetro chamado *base*, aceitando os valores *github* e *stackoverflow*, e uma variável booleana chamada *use_scalling*, que informará se deverá usar normalização.

A normalização será realizada através da classe *StandardScaler* da biblioteca Scikit-Learn, esta classe foi escolhida por executar uma tarefa de padronização dos dados, ou seja, as variáveis observadas para predição podem estar em escalas distintas, e ela irá, através do método *fit()* e *transform()*, padronizar para que a distribuição média seja 0 e seu desvio padrão seja 1, normalizando a escala dos dados.

```
def realizar_previsoes(algoritmo, base, use_scalling):
    target = []

    if (base == 'github'):
        target = merge.Value_x
    else:
        target = merge.Value_y

    y = merge.Language.values.reshape(-1, 1)
    X = target.values.reshape(-1, 1)

    if (use_scalling):
        X = preprocessing.StandardScaler().fit(X).transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    y_pred = []

    if (algoritmo == 'knn'):
        knn = KNeighborsClassifier(n_neighbors=3)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)

    if (algoritmo == 'naive_bayes'):
        naive_bayes = GaussianNB()
        naive_bayes.fit(X_train, y_train)
        y_pred = naive_bayes.predict(X_test)

    if (algoritmo == 'svm'):
        clf = svm.SVC()
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

    score = accuracy_score(y_test, y_pred)
    classificacao = classification_report(y_test, y_pred)

    print('Algoritmo: {}\nDados do: {} \nScore: {} \nRelatório de classificação:{}'.format(algoritmo, base, score))
    print(classificacao)
    return score
```

A função instanciará cada algoritmo conforme seja informado, e, por fim, será printado um relatório geral de como o modelo se comportou, ou seja, informando a sua acurácia e seu score de classificação. Ao final da função, será retornado o valor da acurácia da avaliação.

Em sequência, será criado um DataFrame contendo os resultados de execução de todos os algoritmos para poder realizar uma plotagem com os comparativos da acurácia para os 3 algoritmos escolhidos e para os dados do StackOverflow e do GitHub.

```

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')

```

Essa execução já trará os resultados comparativos finais desejados para analisar o melhor algoritmo e o porque ele foi o melhor para a predição das linguagens mais em alta, e confirmando a hipótese de linguagens interpretadas e de fraca tipagem estarem, nos últimos anos, mais em alta.

Primeiro, será realizada a previsão dos dados do GitHub e do StackOverflow com normalização dos dados para os 3 algoritmos.

```

github_knn = realizar_previsoes('knn', 'github', True)
github_naive_bayes = realizar_previsoes('naive_bayes', 'github', True)
github_svm = realizar_previsoes('svm', 'github', True)

stackoverflow_knn = realizar_previsoes('knn', 'stackoverflow', True)
stackoverflow_naive_bayes = realizar_previsoes('naive_bayes', 'stackoverflow', True)
stackoverflow_svm = realizar_previsoes('svm', 'stackoverflow', True)

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')

```

Os resultados estão presentes na tabela abaixo:

Algoritmo: knn				
Dados do: github				
Score: 0.5420944558521561				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.38	0.35	0.36	57
C/C++	0.50	0.73	0.60	94
Go	0.63	0.82	0.71	50
Java	0.63	0.87	0.73	46
JavaScript	0.43	0.42	0.43	45
PHP	0.51	0.38	0.44	55
Python	0.44	0.14	0.21	59
R	0.73	0.73	0.73	45
TypeScript	0.62	0.36	0.46	36
accuracy			0.54	487
macro avg	0.54	0.53	0.52	487
weighted avg	0.53	0.54	0.52	487

Algoritmo: **naive_bayes**

Dados do: **github**

Score: 0.1026694045174538

Relatório de classificação:

	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.00	0.00	0.00	94
Go	0.10	1.00	0.19	50
Java	0.00	0.00	0.00	46
JavaScript	0.00	0.00	0.00	45
PHP	0.00	0.00	0.00	55
Python	0.00	0.00	0.00	59
R	0.00	0.00	0.00	45
TypeScript	0.00	0.00	0.00	36
accuracy			0.10	487
macro avg	0.01	0.11	0.02	487
weighted avg	0.01	0.10	0.02	487

Algoritmo: **svm**

Dados do: **github**

Score: 0.48459958932238195

Relatório de classificação:

	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.36	0.95	0.52	94
Go	0.62	0.96	0.75	50
Java	0.63	0.91	0.74	46
JavaScript	0.00	0.00	0.00	45
PHP	0.55	0.40	0.46	55
Python	1.00	0.03	0.07	59
R	0.62	0.73	0.67	45
TypeScript	0.00	0.00	0.00	36
accuracy			0.48	487
macro avg	0.42	0.44	0.36	487
weighted avg	0.43	0.48	0.37	487

Algoritmo: **knn**

Dados do: **stackoverflow**

Score: 0.35728952772073924

Relatório de classificação:

	precision	recall	f1-score	support
C#	0.32	0.44	0.37	57
C/C++	0.35	0.69	0.47	94
Go	0.75	0.72	0.73	50
Java	0.20	0.28	0.23	46
JavaScript	0.45	0.42	0.44	45
PHP	0.31	0.09	0.14	55
Python	0.18	0.05	0.08	59
R	0.05	0.02	0.03	45
TypeScript	0.47	0.19	0.27	36
accuracy			0.36	487
macro avg	0.34	0.32	0.31	487
weighted avg	0.34	0.36	0.32	487

Algoritmo: **naive_bayes**

Dados do: **stackoverflow**

Score: 0.1026694045174538

Relatório de classificação:

	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.00	0.00	0.00	94
Go	0.10	1.00	0.19	50
Java	0.00	0.00	0.00	46
JavaScript	0.00	0.00	0.00	45
PHP	0.00	0.00	0.00	55
Python	0.00	0.00	0.00	59
R	0.00	0.00	0.00	45
TypeScript	0.00	0.00	0.00	36
accuracy			0.10	487
macro avg	0.01	0.11	0.02	487
weighted avg	0.01	0.10	0.02	487

Algoritmo: **svm**
 Dados do: **stackoverflow**
 Score: 0.42299794661190965
 Relatório de classificação:

	precision	recall	f1-score	support
C#	0.42	0.60	0.50	57
C/C++	0.42	0.91	0.57	94
Go	0.53	1.00	0.69	50
Java	0.21	0.15	0.17	46
JavaScript	0.45	0.56	0.50	45
PHP	0.13	0.04	0.06	55
Python	1.00	0.03	0.07	59
R	0.00	0.00	0.00	45
TypeScript	0.00	0.00	0.00	36
accuracy			0.42	487
macro avg	0.35	0.37	0.28	487
weighted avg	0.38	0.42	0.32	487

	algoritmo	github	stackoverflow
0	KNN	0.542094	0.357290
1	Naive Bayes	0.102669	0.102669
2	SVM	0.484600	0.422998

Análise de score x algoritmo

algoritmo	github	stackoverflow
KNN	0.542094	0.357290
Naive Bayes	0.102669	0.102669
SVM	0.484600	0.422998

É possível verificar que, mesmo com a normalização, o algoritmo Naive Bayes continua sendo o menos performático, e a disputa das melhores acurárias fica entre KNN e SVM para os dados do GitHub ao invés do StackOverflow, porém, para o SVM, os dados do StackOverflow acabam ficando por pouca coisa atrás.

Em sequência, serão executados os algoritmos sem normalização para os dois datasets e para os três algoritmos:

```
github_knn = realizar_previsoes('knn', 'github', False)
github_naive_bayes = realizar_previsoes('naive_bayes', 'github', False)
github_svm = realizar_previsoes('svm', 'github', False)

stackoverflow_knn = realizar_previsoes('knn', 'stackoverflow', False)
stackoverflow_naive_bayes = realizar_previsoes('naive_bayes', 'stackoverflow', False)
stackoverflow_svm = realizar_previsoes('svm', 'stackoverflow', False)

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')
```

E os resultados estarão logo abaixo na tabela:

Algoritmo: knn				
Dados do: github				
Score: 0.5379876796714579				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.38	0.35	0.36	57
C/C++	0.50	0.73	0.59	94
Go	0.62	0.78	0.69	50
Java	0.63	0.87	0.73	46
JavaScript	0.44	0.42	0.43	45
PHP	0.51	0.38	0.44	55
Python	0.44	0.14	0.21	59
R	0.73	0.73	0.73	45
TypeScript	0.57	0.36	0.44	36
accuracy			0.54	487
macro avg	0.54	0.53	0.51	487
weighted avg	0.53	0.54	0.51	487
Algoritmo: naive_bayes				
Dados do: github				
Score: 0.46406570841889117				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.37	0.56	0.44	94
Go	0.66	0.70	0.68	50
Java	0.63	0.91	0.74	46

JavaScript	0.35	0.71	0.47	45
PHP	0.45	0.36	0.40	55
Python	0.12	0.02	0.03	59
R	0.76	0.71	0.74	45
TypeScript	0.31	0.31	0.31	36
accuracy			0.46	487
macro avg	0.41	0.48	0.42	487
weighted avg	0.39	0.46	0.41	487
Algoritmo: svm				
Dados do: github				
Score: 0.48459958932238195				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.36	0.95	0.52	94
Go	0.62	0.96	0.75	50
Java	0.63	0.91	0.74	46
JavaScript	0.00	0.00	0.00	45
PHP	0.55	0.40	0.46	55
Python	1.00	0.03	0.07	59
R	0.62	0.73	0.67	45
TypeScript	0.00	0.00	0.00	36
accuracy			0.48	487
macro avg	0.42	0.44	0.36	487
weighted avg	0.43	0.48	0.37	487
Algoritmo: knn				
Dados do: stackoverflow				
Score: 0.35523613963039014				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.32	0.44	0.37	57
C/C++	0.35	0.69	0.47	94
Go	0.74	0.70	0.72	50
Java	0.20	0.28	0.23	46
JavaScript	0.45	0.42	0.44	45
PHP	0.31	0.09	0.14	55
Python	0.18	0.05	0.08	59
R	0.04	0.02	0.03	45
TypeScript	0.47	0.19	0.27	36
accuracy			0.36	487
macro avg	0.34	0.32	0.31	487
weighted avg	0.34	0.36	0.32	487
Algoritmo: naive_bayes				
Dados do: stackoverflow				
Score: 0.12320328542094455				
Relatório de classificação:				
	precision	recall	f1-score	support
C#	0.00	0.00	0.00	57
C/C++	0.00	0.00	0.00	94
Go	0.00	0.00	0.00	50
Java	0.00	0.00	0.00	46
JavaScript	0.00	0.00	0.00	45
PHP	0.00	0.00	0.00	55
Python	0.12	1.00	0.22	59
R	0.50	0.02	0.04	45

TypeScript	0.00	0.00	0.00	36												
accuracy			0.12	487												
macro avg	0.07	0.11	0.03	487												
weighted avg	0.06	0.12	0.03	487												
Algoritmo: svm																
Dados do: stackoverflow																
Score: 0.42299794661190965																
Relatório de classificação:																
	precision	recall	f1-score	support												
C#	0.42	0.60	0.50	57												
C/C++	0.42	0.91	0.57	94												
Go	0.53	1.00	0.69	50												
Java	0.21	0.15	0.17	46												
JavaScript	0.45	0.56	0.50	45												
PHP	0.13	0.04	0.06	55												
Python	1.00	0.03	0.07	59												
R	0.00	0.00	0.00	45												
TypeScript	0.00	0.00	0.00	36												
accuracy			0.42	487												
macro avg	0.35	0.37	0.28	487												
weighted avg	0.38	0.42	0.32	487												
algoritmo github stackoverflow																
0 KNN	0.537988	0.355236														
1 Naive Bayes	0.464066	0.123203														
2 SVM	0.484600	0.422998														
Análise de score x algoritmo																
<table border="1"> <caption>Data for Bar Chart: Análise de score x algoritmo</caption> <thead> <tr> <th>Algoritmo</th> <th>github</th> <th>stackoverflow</th> </tr> </thead> <tbody> <tr> <td>KNN</td> <td>0.537988</td> <td>0.355236</td> </tr> <tr> <td>Naive Bayes</td> <td>0.464066</td> <td>0.123203</td> </tr> <tr> <td>SVM</td> <td>0.484600</td> <td>0.422998</td> </tr> </tbody> </table>					Algoritmo	github	stackoverflow	KNN	0.537988	0.355236	Naive Bayes	0.464066	0.123203	SVM	0.484600	0.422998
Algoritmo	github	stackoverflow														
KNN	0.537988	0.355236														
Naive Bayes	0.464066	0.123203														
SVM	0.484600	0.422998														

Como pode ser visualizado, sem a normalização da classe *StandardScaler*, o algoritmo Naive Bayes chega a se aproximar do SVM e do KNN com uma acurácia de 46,40%. Porém, para os dados do StackOverflow, os valores de acurárias para os 3 continuam na mesma escala vista anteriormente.

6. Apresentação dos Resultados

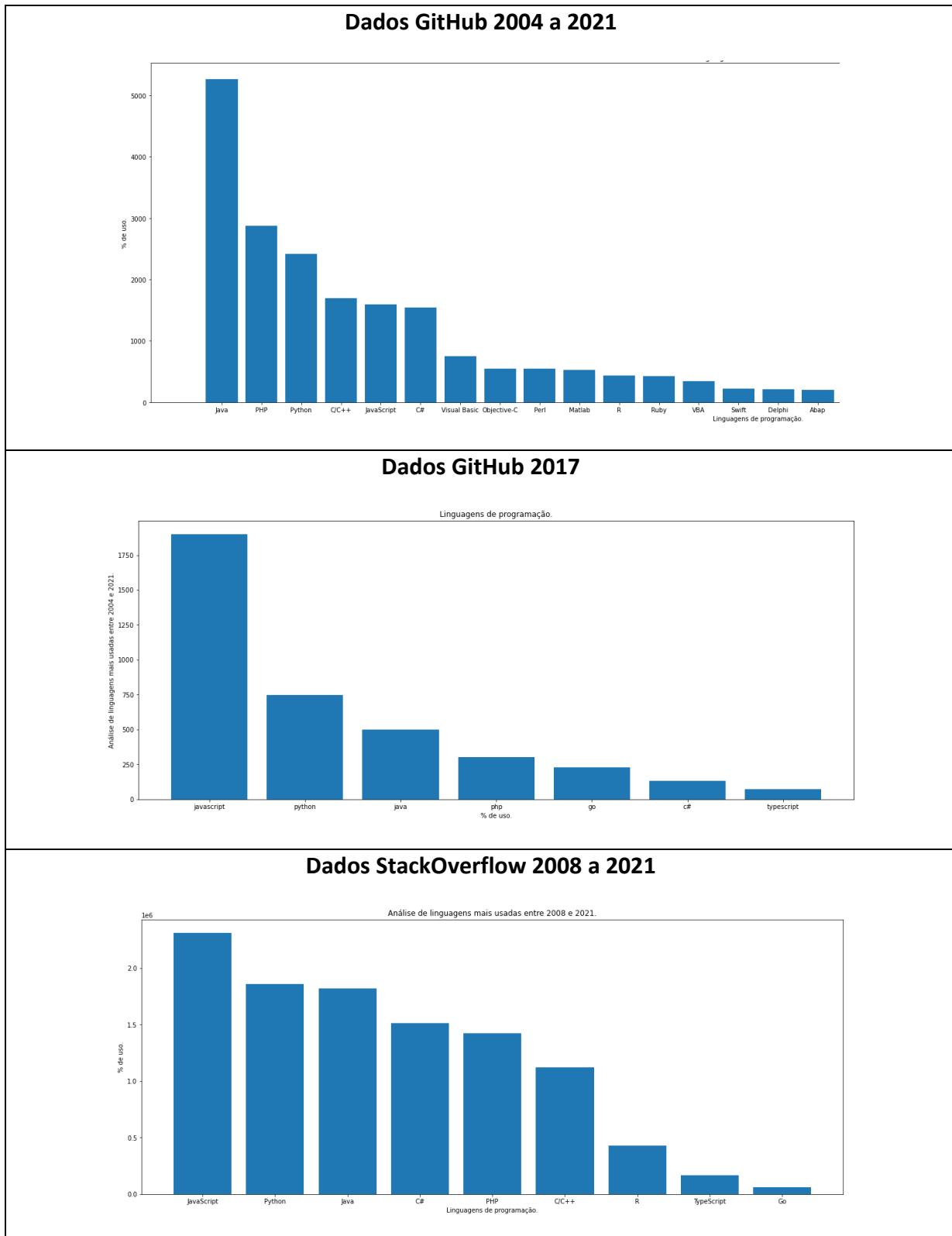
Foi criada uma representação de um modelo Canvas proposto pelo Engenheiro de Machine Learning Louis Dorard em sua plataforma Own Machine Learning, conforme pode ser visto abaixo, respondendo várias perguntas relacionadas às decisões tomadas para criação e desenvolvimento do projeto:

PREDICTION TASK	DECISIONS	VALUE PROPOSITION	DATA COLLECTION	DATA SOURCES
Type of task? Entity on which predictions are made? Possible outcomes? Wait time before observation?	How are predictions turned into proposed value for the end-user? Mention parameters of the process / application that does that.	Who is the end-user? What are their objectives? How will they benefit from the ML system? Mention workflow/interfaces.	Strategy for initial train set & continuous update. Mention collection rate, holdout on production entities, cost/constraints to observe outcomes.	Where can we get (raw) information on entities and observed outcomes? Mention database tables, API methods, websites to scrape, etc.
Serão analisados dois tipos variáveis: Análises regressão entre variáveis temporais e percentuais de uso entre as linguagens de programação.	As decisões serão para corroborar com a análise de que as linguagens de programação compiladas com tipagem forte tiveram uma queda em comparação com as linguagens com tipagem fraca e interpretadas de script devido ao aumento na demanda de velocidade e rápida curva de aprendizado do mercado.	O objeto de estudo realizado é focado para novos desenvolvedores e gestores de projetos para que possam analisar quando e quais serão as melhores tecnologias a serem abordadas para um objetivo de negócio específico levando em consideração a movimentação e aceitação do mercado por uma tecnologia específica.	A estratégia de teste foi definida como 33% do conjunto de dados total analisado, e será comparado para analisar regressão e classificação das linguagens analisadas, assim como a comparação de seus resultados e acurácia com score de avaliação.	Foram utilizadas três bases de dados, duas extraídas da plataforma Kaggle sobre dados do GitHub, e a terceira base é extraída da plataforma StackOverflow sobre quantidade de perguntas relacionadas às tecnologias nos anos de 2008 a 2021.
IMPACT SIMULATION	MAKING PREDICTIONS	BUILDING MODELS		FEATURES
Can models be deployed? Which test data to assess performance? Cost/gain values for (in)correct decisions? <u>Fairness constraint?</u>	When do we make real-time / batch pred? Time available for this + featurization + post-processing? Compute target?	How many prod models are needed? When would we update? Time available for this (including featurization and analysis)?		Input representations available at prediction time, extracted from raw data sources.
O impacto pode ser verificado com a análise histórica da evolução das tecnologias e como começaram a ser abordadas nos últimos 5 anos para novas linguagens criadas e como isso impacta em outras linguagens já existentes e consagradas.	A evolução das previsões pode ser incrementada conforme o conjunto de dados analisado, porém, a análise em estudo foi feita com um conjunto fixo sem atualização, para observar uma hipótese sobre a evolução das tecnologias e comprovar através dos dados dos usuários das duas maiores plataformas de código e dúvidas sobre tecnologia existentes.	Foram criados quatro modelos de previsão, sendo um de análise de regressão e três de algoritmos supervisionados de classificação, tais como KNN, SVM e Naive-Bayes.		Foram analisadas tanto regressão da quantidade de perguntas e percentuais de uso das tecnologias através dos anos quanto os seus percentuais comparando linguagens que cresceram com objetivos diferentes, como Java e Javascript, analisando se há correlação entre esse crescimento. Foram também classificadas linguagens através de análises supervisionadas e comparados os resultados de acurácia entre as diferentes abordagens.
	MONITORING			
	Metrics to quantify value creation and measure the ML system's impact in production (on end-users and business)?	As principais métricas avaliadas são a regressão entre a correlação existente no aumento de linguagens e o declínio de outras em questão de uso e abordagem de mercado quando se diz em velocidade de aprendizado, implementação e implantação para tarefas rápidas e menos complexas.		

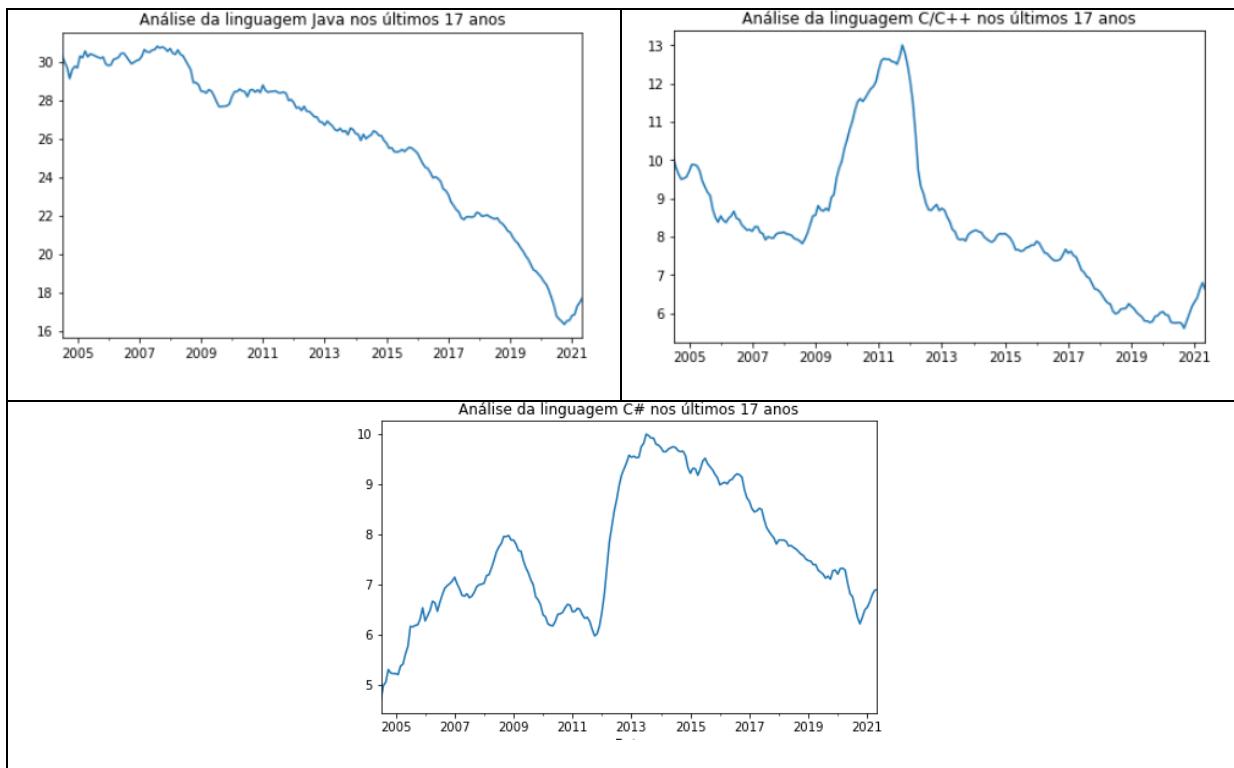
O resultado foi bastante satisfatório para provar a hipótese das linguagens compiladas de fato sofrerem uma queda desde 2010 a 2015 em razão das linguagens interpretadas serem atingidas por um aumento de procura por parte tanto de empresas, com demandas mais rápidas de desenvolvimento e execução, quanto por desenvolvedores que estão entrando no mercado em busca de tecnologias que demandam menor curva de aprendizado em relação à tempo e complexidade, quanto também por desenvolvedores já maduros e com certo tempo de experiência que estão buscando migrar para essas tecnologias e acompanhar a nova onda de uso dessas tecnologias.

A maneira em que conseguimos enxergar esse aumento foi através dos 3 datasets, que, embora não tenham sido usados os 3 para realizar os algoritmos de machine learning,

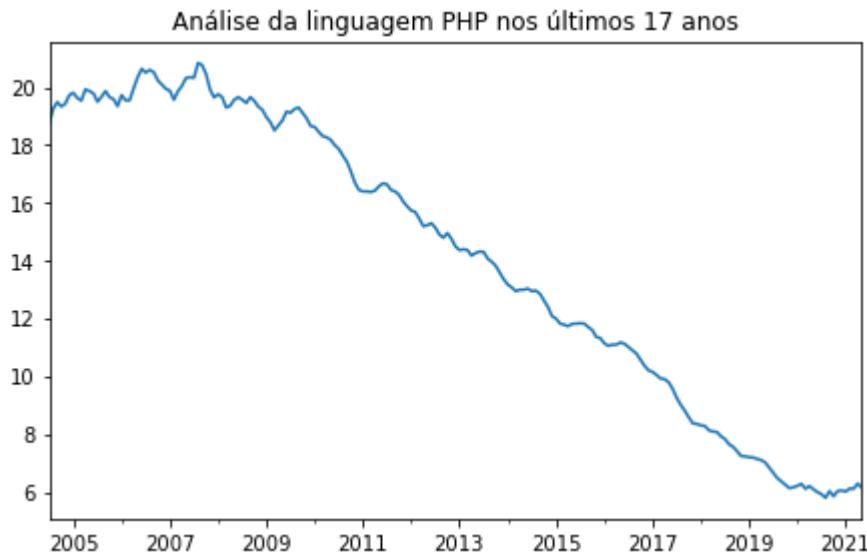
foi possível apenas na análise exploratória dos dados, verificar que o ranking das top 5 linguagens está equiparado se colocados lado a lado, conforme a tabela abaixo:



Ou seja, Java, Javascript, Python, PHP e C# estão, nos 3 datasets retirados de pesquisas de usuários das duas maiores plataformas de tecnologia, e também através da análise de séries temporais realizada, em que vemos o aumento dessas linguagens interpretadas de script e com fraca tipagem subindo bastante, enquanto linguagens compiladas com tipagem forte sofrem uma queda, podemos pegar de exemplo linguagens compiladas com fraca tipagem mais utilizadas: Java, C/C++ e C#, nesta ordem.



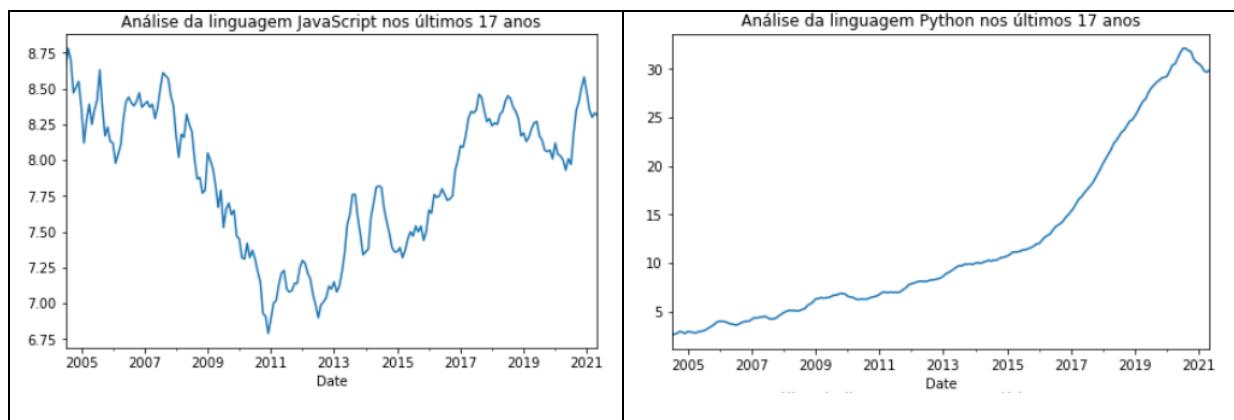
Muitas dessas linguagens tiveram altos e baixos, mas pode-se ver que a partir da década de 2010, a queda foi bastante expressiva. Como foi exemplificado anteriormente, o PHP é uma linguagem interpretada de script e com tipagem fraca, porém, conforme afirmado, o PHP também sofreu uma grande queda nos últimos anos devido à sua estrutura, complexidade, sintaxe verbose e outras funções que levaram a uma decadência, isso pode ser visualizado também nesta imagem:

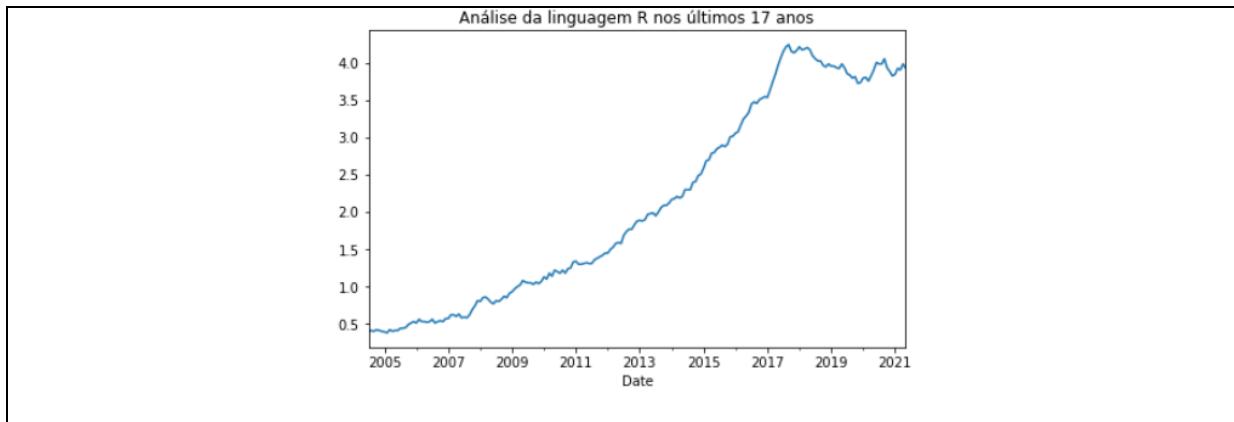


O PHP é uma linguagem que teve uma grande contribuição para a web no geral, principalmente pela facilidade e velocidade em criar ambientes back-end para aplicações, persistência de dados, conexão com bancos de dados, criação de sistemas web e operações de CRUD (Create, Read, Update e Delete).

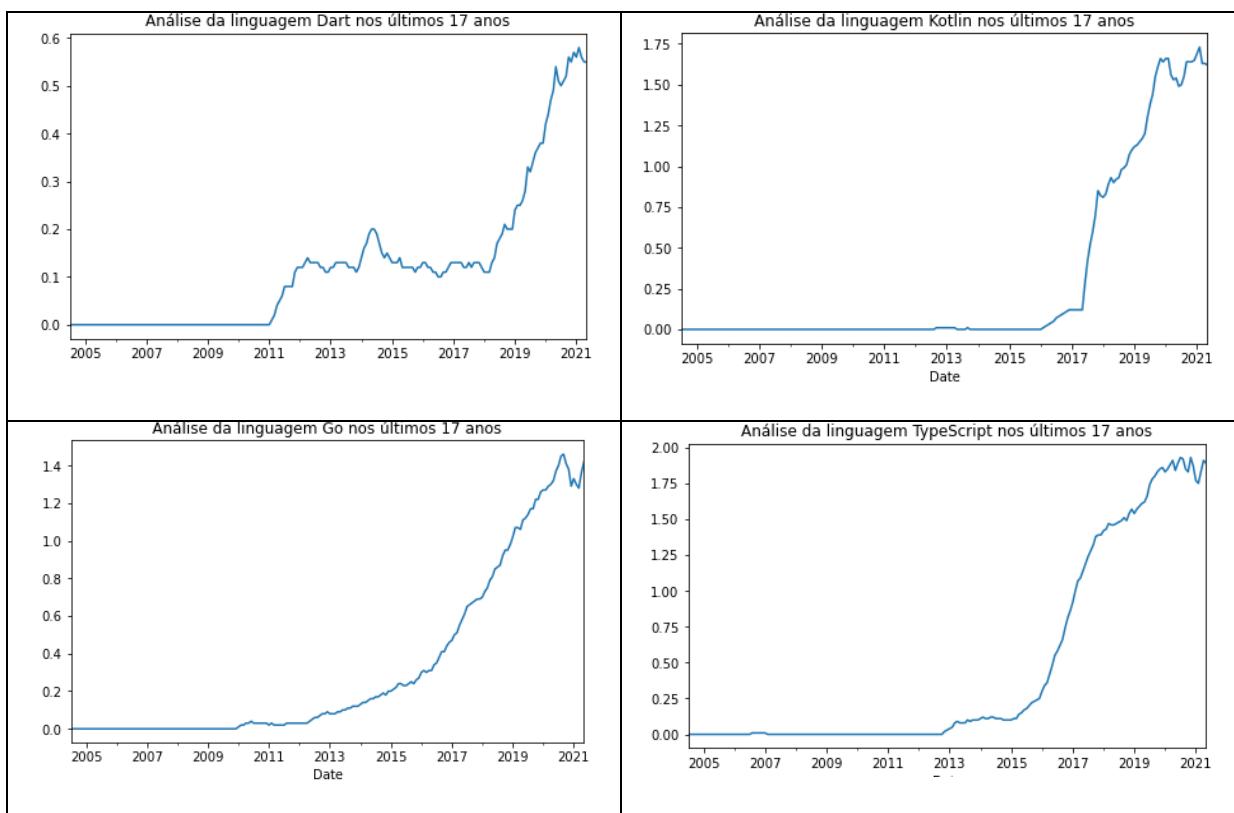
Porém, a linguagem sofre uma queda com o surgimento de grandes frameworks web e principalmente da popularização das REST APIs e do surgimento do modelo arquitetural de microsserviços, que conduziram ao surgimento de grandes frameworks de mercado como ASP.NET, Spring Framework, Express.js, Nest.js, Django, Flask, entre outros. O PHP sustenta-se bastante com frameworks web também, como é o caso do Laravel e Symfony, que permitem a criação de sistemas MVC, REST APIs e microsserviços.

Agora, serão selecionadas as linguagens interpretadas de script com tipagem fraca para ver o mesmo comparativo, como por exemplo, Javascript, Python e R, nesta ordem:





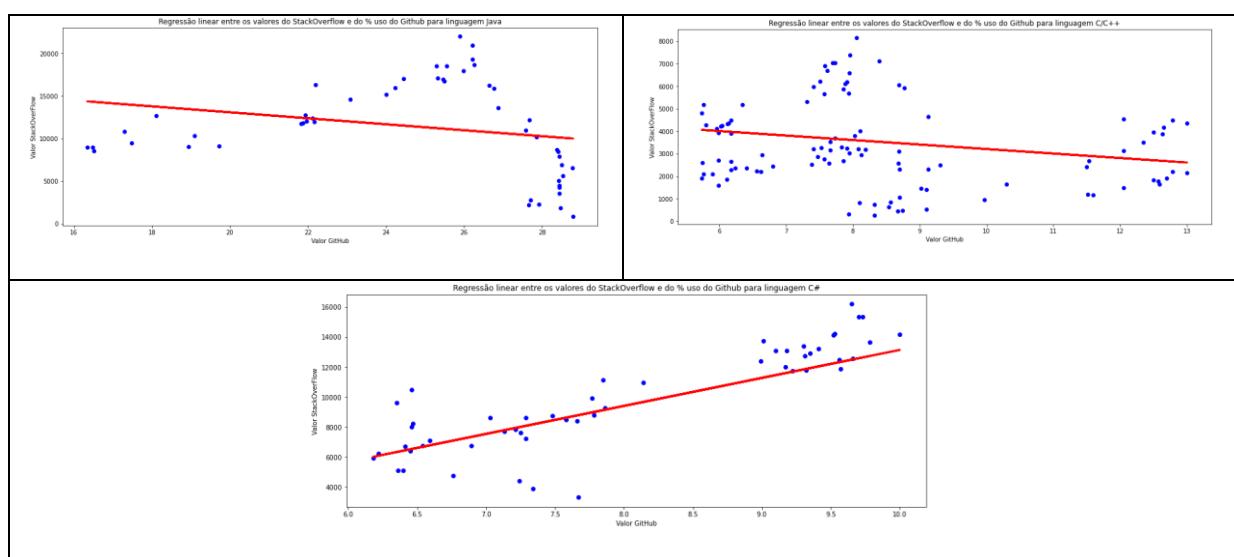
Os resultados são bem satisfatórios acerca de linguagens interpretadas terem sofrido aumento enquanto as compiladas uma queda, mas claro que isso ocorreu mais com as linguagens mais robustas e preestabelecidas no mercado. Afinal, é possível observar um grande aumento de uso também em linguagens compiladas com tipagem fraca ou forte a partir da década de 2010, como é o caso de Dart, Kotlin e Go, e também Typescript, que, embora seja uma linguagem interpretada de script, é uma linguagem com tipagem forte estática inserida. É possível observar a evolução dessas 4 linguagens abaixo que fogem um pouco ao padrão das linguagens em queda observado anteriormente:



Conforme visto na tabela acima, essas 4 linguagens fogem à regra de interpretadas e tipagem fraca, ou compiladas e tipagem forte, e também estão em constante aumento até o ano de 2021.

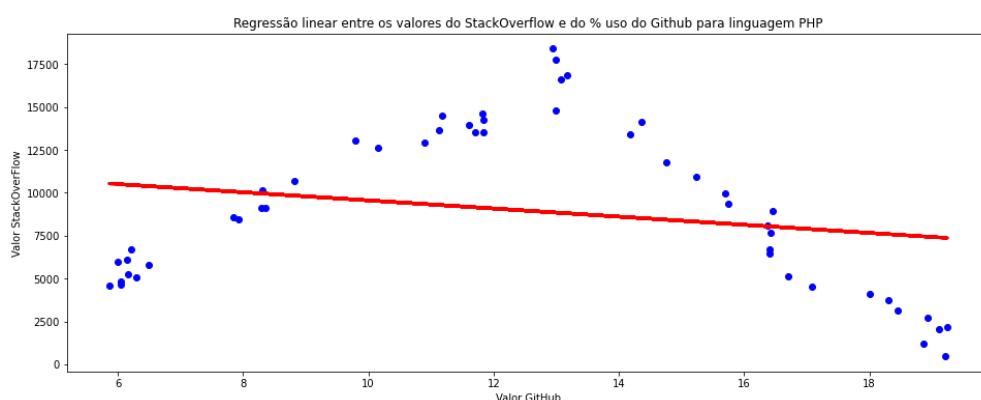
Através do dataset do GitHub de 2017 não é possível verificar correlação entre as variáveis das linguagens, e também não é possível realizar a predição de modelos de regressão. Para verificar se as mesmas linguagens mostradas acima pelo dataset do GitHub também possuem queda ou aumento, é preciso olhar o dataset do StackOverflow, que também trata os dados de maneira temporal.

Olhando então para Java, C/C++ e C#, podemos notar a seguinte curva na regressão entre os dois datasets:



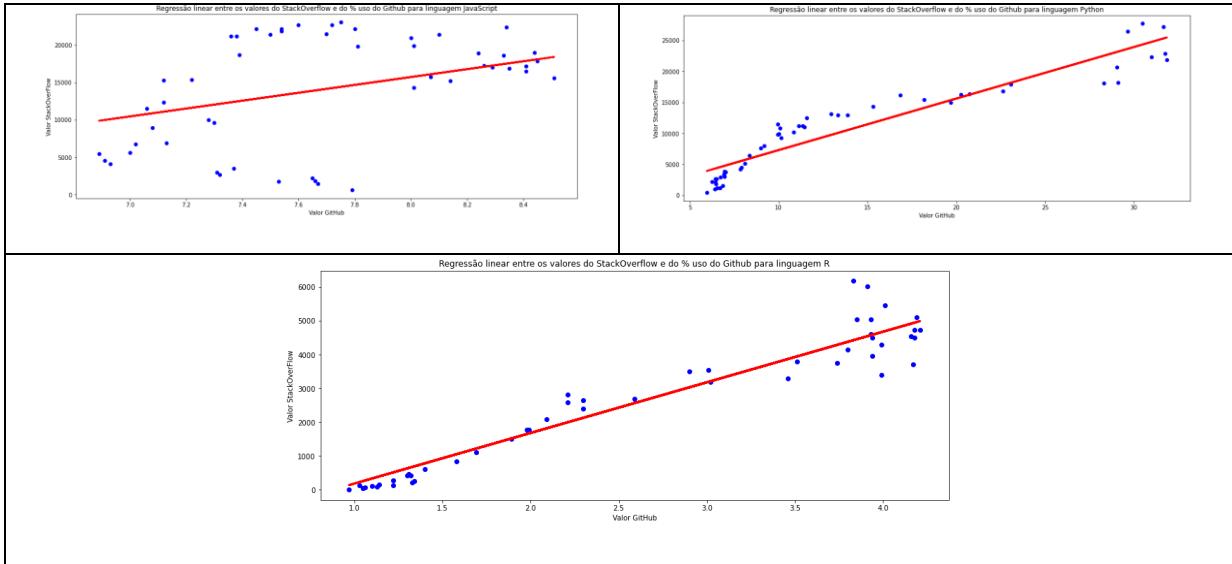
É possível visualizar que os resultados das regressões são indesejáveis, mesmo que o C# possua uma curva de regressão positiva, explicada pela regressão entre ambas as colunas dos datasets de GitHub e StackOverflow.

A imagem abaixo mostra a curva de declínio da linguagem PHP em ambos os datasets:



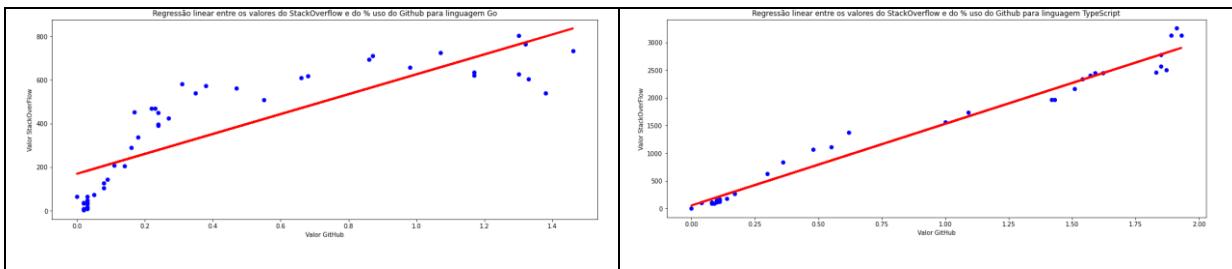
Confirmando, assim como visto anteriormente, que PHP é uma exceção à regra de linguagens interpretadas com fraca tipagem devido às suas características.

Para visualizar os aumentos, pode ser vista a regressão para as linguagens interpretadas com fraca tipagem, assim como visto anteriormente, para Javascript, Python e R, nessa ordem:



Como visto, o Python e o R possuem excelente correlação entre os dois datasets e seus valores de regressão também estão em um nível de excelência, já o Javascript, embora seja claramente visualizável seu crescimento, já foi avaliado anteriormente nas análises que é uma linguagem que sofreu uma grande oscilação até os anos de 2015, pois é uma linguagem que foi criada nos anos 90 e que teve vários usos distintos no mundo da web, e mesmo com uma regressão com menor qualidade se comparada a R e Python, continua na curva de crescimento, conforme visto na análise anterior.

Por fim, pode-se visualizar também as regressões com excelência nas linguagens compiladas e com fraca tipagem, ou interpretadas com forte tipagem, como é o caso analisado em Typescript e Go, como pode-se visualizar na tabela abaixo:



Novamente, mostrando que a integração entre os datasets possuem resultados bastante semelhantes, mesmo extraindo tais informações das duas maiores plataformas de armazenamento e dúvidas sobre tecnologia.

Por fim, pode-se analisar os algoritmos de Machine Learning escolhidos e utilizados para verificar previsões sobre as linguagens.

Os algoritmos escolhidos foram o K-Vizinhos Próximos, também conhecido como K-Nearest Neighbors ou KNN, o algoritmo Naive Bayes e o algoritmo Máquina de Vetores de Suporte, também conhecido como Support Vector Machines ou SVM.

A escolha dos 3 algoritmos foi devido à estrutura do dataset analisado, contendo colunas sobre as linguagens e seus respectivos valores, ou seja, são dados rotulados, que demandam análise de classificação.

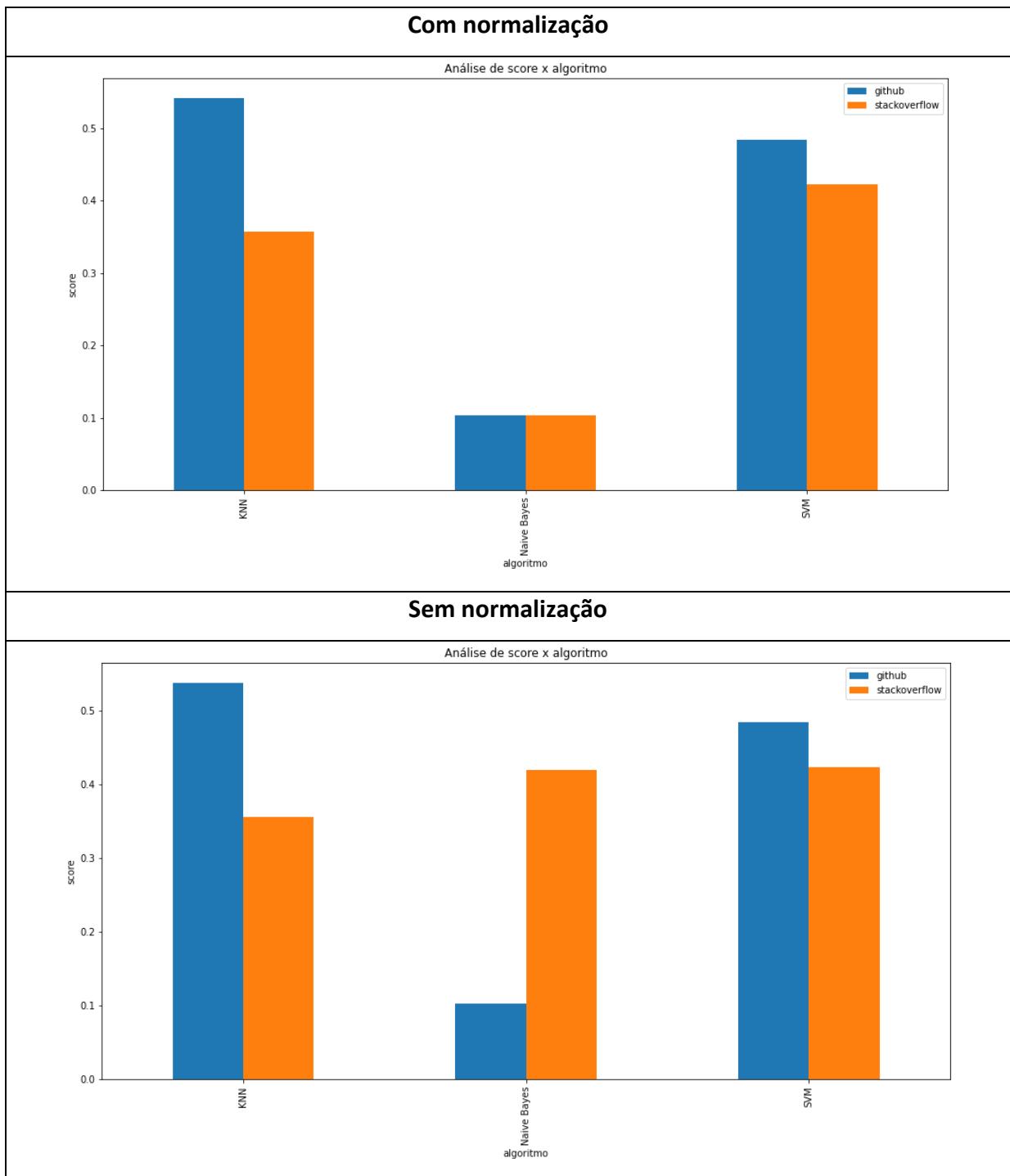
O algoritmo KNN foi escolhido por ser um algoritmo supervisionado de classificação que condiz com sua facilidade de implementação e simplicidade para realizar previsões em séries categóricas, ou seja, é um algoritmo de classificação, conforme o cenário apresentado, de N valores para várias categorias de linguagens.

O algoritmo Naive Bayes foi escolhido por ser um algoritmo supervisionado de classificação que opera de forma a desconsiderar a correlação entre os valores, podendo assim dar um auxílio para verificar se a grande ou falta de correlação entre as variáveis pode ser um impeditivo na previsão do modelo.

O algoritmo SVM foi escolhido por ser um algoritmo supervisionado de classificação e regressão que funciona muito bem quando a quantidade de dimensões de análise é maior que o número de amostras, por ser eficiente quando relacionado à desempenho e uso de memória, buscando uma linha de separação entre duas classes distintas, essa linha é o hiperplano.

Os resultados das acurárias podem ser apresentados nas duas tabelas abaixo:

Normalização	Dataset	KNN	Naive Bayes	SVM
NÃO	GitHub	0.537988	0.102669	0.484600
NÃO	StackOverflow	0.355236	0.418891	0.422998
SIM	GitHub	0.542094	0.102669	0.484600
SIM	StackOverflow	0.357290	0.102669	0.422998



É possível verificar que os algoritmos SVM e KNN foram os que mais performaram com normalização, sendo o KNN o melhor em resultados para os dados do GitHub com 54% de acurácia, em comparação aos dados do StackOverflow, isso devido à melhor correlação e distribuição dos dados quando comparados até mesmo nas séries temporais e nos algoritmos de regressão.

É também possível verificar que o algoritmo Naive Bayes performou de maneira inferior em ambos os datasets, principalmente após a aplicação da normalização dos dados, isso pode ser explicado pelo fato do algoritmo desconsiderar a regressão entre as variáveis, o que pode ser justificado pela maneira que a correlação entre as variáveis é um ponto de bastante impacto para as análises.

A última análise a ser realizada é uma tabela comparando as 3 principais variáveis na análise dos modelos de Machine Learning de classificação apresentados, ou seja, a precisão, o recall e o F1-Score. Os tipos das linguagens podem ser: Compilada com Tipagem Forte (CTFT), Compilada com Tipagem Fraca (CTFC) e Interpretada com Tipagem Fraca (ITFC)

Linguagem	Tipo	Algoritmo	Dataset	Normalizado	Precisão	Recall	F1-Score
Java	CTFT	KNN	GitHub	SIM	0.63	0.87	0.73
R	ITFC	KNN	GitHub	SIM	0.73	0.73	0.73
Go	CTFC	SVM	GitHub	SIM	0.62	0.96	0.75
R	ITFC	Naive Bayes	GitHub	NÃO	0.76	0.71	0.74
Go	CTFC	KNN	StackOverflow	NÃO	0.74	0.70	0.72
Go	CTFC	SVM	StackOverflow	NÃO	0.53	1.00	0.69

Essas foram as melhores performances de linguagens nos algoritmos KNN, Naive Bayes e SVM, com ou sem normalização, podendo verificar que são, em sua maioria, linguagens mais novas, interpretadas de script e com fraca tipagem, ou linguagens compiladas com fraca tipagem, com exceção da linguagem Java, sendo estruturalmente uma linguagem compilada com tipagem forte e muito abrangente no mercado de tecnologia.

7. Links

Link para o vídeo: <https://www.youtube.com/watch?v=DtbqUzOUadE>

Link para o repositório: <https://github.com/vhnegrisolli/tcc-pos-graduacao-puc>

Os datasets encontram-se em */dados* e em */dados/stack_overflow*.

REFERÊNCIAS

- ALINA; CHEKALIN, Dmitry. **Top Programming Languages to Shape Software Development in 2021.** <https://www.codica.com/blog/top-programming-languages-trends/> (acessado em 04/01/2022)
- CHAN, Rosalie. **The 10 fastest-growing programming languages, according to Microsoft-owned GitHub.** <https://www.businessinsider.com/fastest-growing-programming-languages-github-2019-11> (acessado em 05/01/2022)
- Sciedirect. **Interpreted Language.** <https://www.sciencedirect.com/topics/computer-science/interpreted-language> (acessado em 07/01/2022)
- Simplilearn. **The Evolution of Programming Languages in Past 10 Years.** <https://www.simplilearn.com/evolution-of-programming-languages-article> (acessado em 07/01/2022)
- ROBINSON, David. **The Incredible Growth of Python.** <https://stackoverflow.blog/2017/09/06/incredible-growth-python/> (acessado em 08/01/2022)
- IBM. **Compiled versus interpreted languages.** <https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-compiled-versus-interpreted-languages> (acessado em 11/01/2022)
- CABRERA, Italo Baeza. **PHP “dying” this Decade: One Year After.** <https://medium.com/swlh/one-year-from-php-dying-this-decade-32e2b7a79507> (acessado em 11/01/2022)
- Berkeley Extension. **11 Most In-Demand Programming Languages in 2022.** <https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/> (acessado em 11/01/2022)
- GRUS, Joel. **Data Science do Zero:** primeiras regras com o Python. Trad. Wellington Nascimento. Rio de Janeiro: Editora Alta Books, 2016.
- MCKINNEY, Wes. **Python para Análise de Dados:** tratamento de dados com Pandas, Numpy e IPython. Trad. Lúcia A. Kinoshita. São Paulo: Novatec Editora Ltda., 2019.
- GÉRON, Aurélien. **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow:** conceitos, ferramentas e técnicas para construção de sistemas inteligentes. Trad. Rafael Contatori. Rio de Janeiro: Editora Alta Books, 2019.

APÊNDICE

Programação/Scripts

```
In [91]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime

from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn import svm

import warnings
warnings.filterwarnings('ignore')

In [92]: def configurar_plot_com_dimensoes(titulo, x, y, h, w):
    plt.title(titulo)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.gcf().set_size_inches(h, w)
    plt.show()

In [93]: def configurar_plot(titulo, x, y):
    configurar_plot_com_dimensoes(titulo, x, y, 16, 8)

In [94]: df = pd.read_csv('dados/Most Popular Programming Languages from 2004 to 2021 V4.csv')
df.head(10)
```

Out[94]:

	Date	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	...	PHP	Python	R	Ruby	Rust	Scala	Swift	TypeScript	VBA	Visual Basic
0	July 2004	0.34	0.36	10.08	4.71	0.43	0.0	2.82	0.0	0.03	...	18.75	2.53	0.39	0.33	0.08	0.03	0.0	0.0	1.44	8.56
1	August 2004	0.36	0.36	9.81	4.99	0.46	0.0	2.67	0.0	0.07	...	19.26	2.64	0.41	0.40	0.09	0.03	0.0	0.0	1.46	8.57
2	September 2004	0.41	0.41	9.63	5.06	0.51	0.0	2.65	0.0	0.08	...	19.49	2.72	0.40	0.41	0.10	0.03	0.0	0.0	1.55	8.41
3	October 2004	0.40	0.38	9.50	5.31	0.53	0.0	2.77	0.0	0.09	...	19.34	2.92	0.42	0.46	0.11	0.04	0.0	0.0	1.61	8.49
4	November 2004	0.38	0.38	9.52	5.24	0.55	0.0	2.76	0.0	0.07	...	19.43	2.84	0.41	0.45	0.13	0.04	0.0	0.0	1.50	8.24
5	December 2004	0.36	0.37	9.56	5.23	0.53	0.0	2.77	0.0	0.09	...	19.73	2.71	0.40	0.42	0.13	0.04	0.0	0.0	1.46	8.08
6	January 2005	0.39	0.38	9.70	5.23	0.56	0.0	2.65	0.0	0.11	...	19.81	2.91	0.39	0.47	0.15	0.03	0.0	0.0	1.51	7.79
7	February 2005	0.37	0.39	9.88	5.21	0.49	0.0	2.66	0.0	0.07	...	19.63	2.87	0.38	0.45	0.15	0.03	0.0	0.0	1.45	7.67
8	March 2005	0.34	0.37	9.88	5.38	0.45	0.0	2.65	0.0	0.08	...	19.54	2.81	0.42	0.46	0.13	0.03	0.0	0.0	1.44	7.68
9	April 2005	0.34	0.36	9.85	5.42	0.41	0.0	2.56	0.0	0.08	...	19.93	2.78	0.40	0.43	0.11	0.02	0.0	0.0	1.36	7.52

10 rows × 29 columns

```
In [95]: df.shape
Out[95]: (203, 29)
```

```
In [96]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203 entries, 0 to 202
Data columns (total 29 columns):
Date      203 non-null object
Abap      203 non-null float64
Ada       203 non-null float64
C/C++    203 non-null float64
C#        203 non-null float64
Cobol    203 non-null float64
Dart     203 non-null float64
Delphi   203 non-null float64
Go        203 non-null float64
Groovy   203 non-null float64
Haskell  203 non-null float64
Java     203 non-null float64
JavaScript 203 non-null float64
Julia    203 non-null float64
Kotlin   203 non-null float64
Lua      203 non-null float64
Matlab   203 non-null float64
Objective-C 203 non-null float64
Perl     203 non-null float64
PHP      203 non-null float64
Python   203 non-null float64
R        203 non-null float64
Ruby    203 non-null float64
Rust    203 non-null float64
Scala   203 non-null float64
Swift   203 non-null float64
TypeScript 203 non-null float64
VBA     203 non-null float64
Visual Basic 203 non-null float64
dtypes: float64(28), object(1)
memory usage: 46.1+ KB
```

```
In [97]: df.describe()
```

Out[97]:

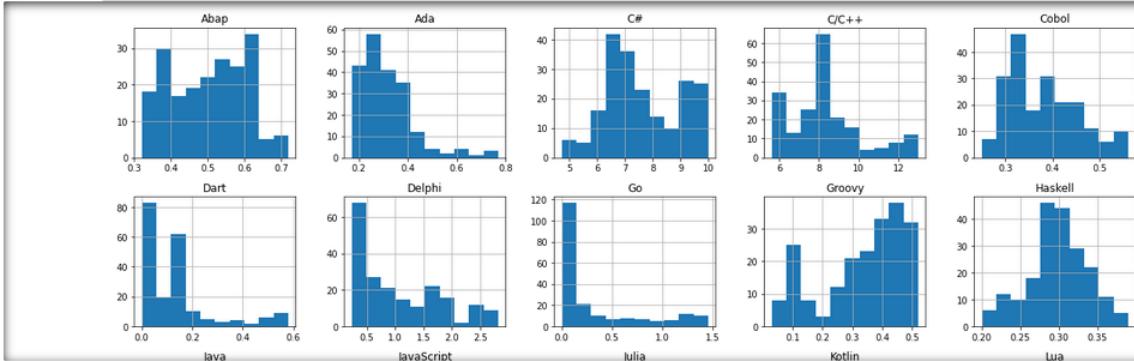
	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	Haskell	...	PHP	Python
count	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	...	203.000000	203.000000
mean	0.498030	0.311478	8.333251	7.603202	0.381034	0.121576	1.055172	0.322906	0.334089	0.298621	...	14.152020	11.917882
std	0.102097	0.110269	1.807976	1.306233	0.071528	0.149658	0.757032	0.451741	0.137936	0.039728	...	4.983405	8.768295
min	0.320000	0.170000	5.610000	4.710000	0.250000	0.000000	0.230000	0.000000	0.030000	0.200000	...	5.810000	2.530000
25%	0.400000	0.230000	7.395000	6.675000	0.320000	0.000000	0.370000	0.000000	0.260000	0.280000	...	9.990000	5.435000
50%	0.510000	0.300000	8.080000	7.330000	0.370000	0.110000	0.810000	0.080000	0.380000	0.300000	...	14.500000	8.440000
75%	0.580000	0.360000	8.940000	8.935000	0.430000	0.135000	1.680000	0.505000	0.450000	0.320000	...	19.300000	15.925000
max	0.720000	0.770000	13.000000	10.000000	0.560000	0.580000	2.820000	1.460000	0.520000	0.390000	...	20.840000	32.110000

8 rows × 28 columns

```
In [98]: meses = sorted(list(set([i.split(' ')[0] for i in df['Date'].unique()])))
anos = sorted(list(set([i.split(' ')[1] for i in df['Date'].unique()])))
print('Meses: {}\\nAnos: {}'.format(meses, anos))

Meses: ['April', 'August', 'December', 'February', 'January', 'July', 'June', 'March', 'May', 'November', 'October', 'September']
Anos: ['2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021']
```

```
In [100]: df.hist()
configurar_plot_com_dimensoes('Histograma Linguagens', '', '', 20, 20)
plt.show()
```



```
In [101]: def createDataFrameFor(df, colunas, colunaAtual):
    return pd.DataFrame(
        {
            'Date': df.Date,
            'Year': pd.DatetimeIndex(df['Date']).year,
            'Timestamp': map(lambda i : datetime.strptime(df["Date"][i], '%B %Y'), range(len(df.Date))),
            'Language': colunas[colunaAtual],
            'Value': df[df.columns[colunaAtual]]
        }
    )
colunas = df.columns
```

```
dados_tratados = createDataFrameFor(df, colunas, 1)

for coluna in range(1, len(colunas)):
    dados_tratados = pd.concat([dados_tratados, createDataFrameFor(df, colunas, coluna)])

dados_tratados.reset_index(drop=True, inplace=True)

dados_tratados['UnixTime'] = list(map(lambda i: \
    (pd.to_datetime([dados_tratados['Timestamp'][i]]).astype(int) / 10**9)[0], \
    range(len(dados_tratados['Date']))))

dados_tratados.head()
```

```
Out[101]:
      Date Year  Timestamp Language  Value  UnixTime
0   July 2004  2004-07-01     Abap  0.34  1.088640e+09
1  August 2004  2004-08-01     Abap  0.36  1.091318e+09
2 September 2004  2004-09-01     Abap  0.41  1.093997e+09
3 October 2004  2004-10-01     Abap  0.40  1.096589e+09
4 November 2004  2004-11-01     Abap  0.38  1.099267e+09
```

```
In [12]: dados_tratados.to_csv('dados/Dados.csv')
```

```
In [13]: dados_tratados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5887 entries, 0 to 5886
Data columns (total 6 columns):
Date      5887 non-null object
Year       5887 non-null int64
Timestamp  5887 non-null datetime64[ns]
Language   5887 non-null object
Value      5887 non-null float64
UnixTime   5887 non-null float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(2)
memory usage: 276.1+ KB
```

```
In [14]: dados_tratados.shape
Out[14]: (5887, 6)

In [15]: dados_tratados.describe()

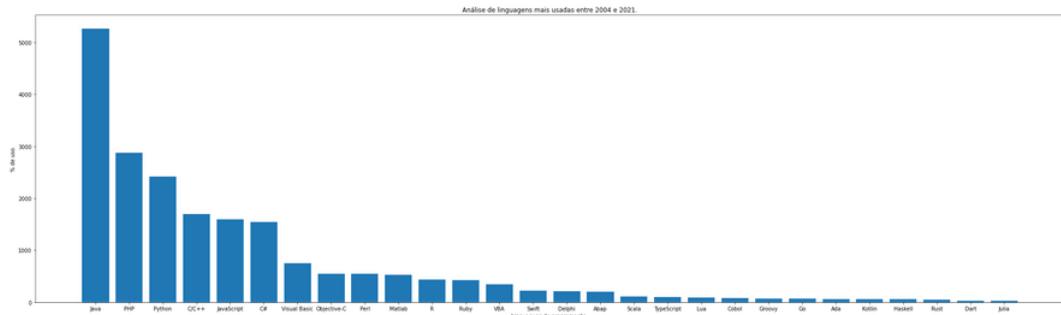
Out[15]:
   Year      Value      UnixTime
count 5887.000000 5887.000000 5.887000e+03
mean 2012.458128 3.465436 1.354277e+09
std 4.900599 6.019926 1.541191e+08
min 2004.000000 0.000000 1.088640e+09
25% 2008.000000 0.280000 1.220227e+09
50% 2012.000000 0.670000 1.354320e+09
75% 2017.000000 3.830000 1.488326e+09
max 2021.000000 32.110000 1.619827e+09
```

```
In [16]: dados_agrupados = dados_tratados[['Language', 'Value']].groupby(by=['Language'], as_index=False).sum()
dados_agrupados = dados_agrupados.sort_values(by=['Value'], ascending=False)
dados_agrupados
```

```
In [17]: fig, ax = plt.subplots()
ax.set_xticklabels(dados_agrupados['Language'])

ax.bar(x = dados_agrupados['Language'], height = dados_agrupados['Value'])
plt.gcf().set_size_inches(35, 10)
plt.savefig('imgs/Analise Linguagens.png')

configurar_plot_com_dimensoes(
    'Análise de linguagens mais usadas entre 2004 e 2021.',
    'Linguagens de programação.',
    '% de uso.',
    35,
    10
)
```



Verificando correlação entre as variáveis

- Igual a 1 -----> Correlação linear positiva perfeita
- Maior que 0 -----> Correlação linear positiva
- Igual a 0 -----> Sem correlação linear
- Menor que 0 -----> Correlação linear negativa
- Igual a -1 -----> Correlação linear negativa perfeita

```
In [18]: targets = [
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'TypeScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'C/C++'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Java'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'TypeScript'
    }
]
```



```
In [19]: def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = df[linguagem_1].corr(df[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}.\n'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    df.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')

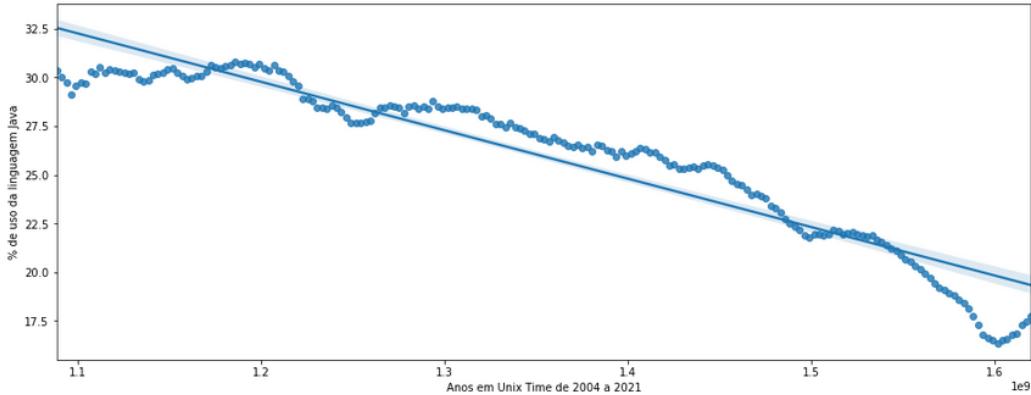
    configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), 11, 11, 20, 10)
```



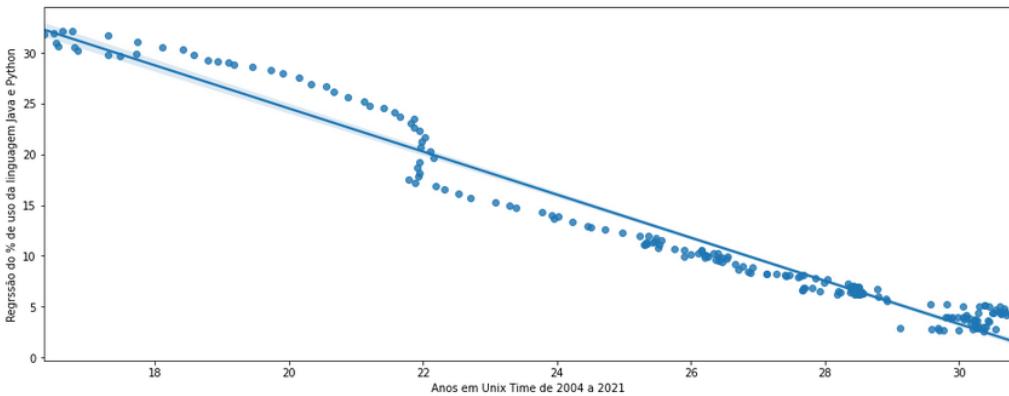
```
In [20]: for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])
```

```
In [21]: df_java = dados_tratados[dados_tratados['Language'] == 'Java']
sns.regplot(x="UnixTime", y="Value", data=df_java)
plt.gcf().set_size_inches(16, 6)
plt.ylabel('% de uso da linguagem Java')
plt.xlabel('Anos em Unix Time de 2004 a 2021')
plt.show()
```



```
In [22]: t = df[['Java', 'Python']]
sns.regplot(x="Java", y="Python", data=t)
plt.gcf().set_size_inches(16, 6)
plt.ylabel('Regressão do % de uso da linguagem Java e Python')
plt.xlabel('Anos em Unix Time de 2004 a 2021')
plt.show()
```



Criando os modelos de Machine Learning para o algoritmo de Regressão Linear para as correlações observadas

```
In [23]: def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = df[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

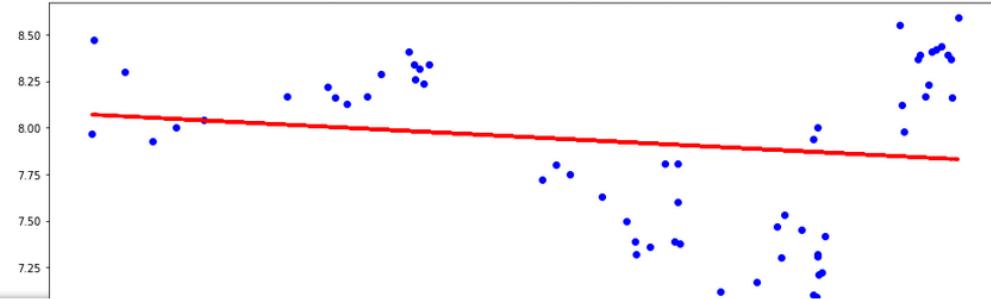
    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)
```

```
plt.gcf().set_size_inches(16, 6)
print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))
plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
plt.show()
```

In [24]: `for target in targets:`
 `prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])`

Teste R-quadrado: -0.008649979344982128

Regressão linear entre % de uso das linguagens Java e JavaScript.



```
In [25]: def prever_regressao_linguagem(linguagem):
    df_linguagem = dados_tratados[dados_tratados['Language'] == linguagem]

    X = df_linguagem.UnixTime.values.reshape(-1, 1)
    y = df_linguagem.Value.values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)
```

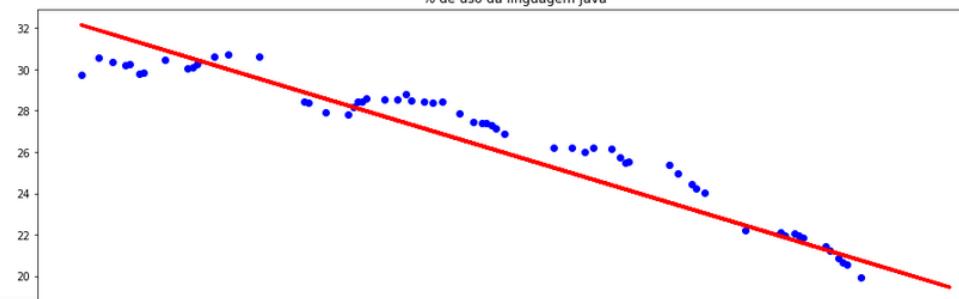
```
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red', linewidth=3)

plt.gcf().set_size_inches(16, 6)
print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))
plt.title('% de uso da linguagem {}'.format(linguagem))
plt.show()
```

```
prever_regressao_linguagem('Java')
prever_regressao_linguagem('JavaScript')
prever_regressao_linguagem('Python')
prever_regressao_linguagem('TypeScript')
prever_regressao_linguagem('C/C++)
prever_regressao_linguagem('C#')
prever_regressao_linguagem('PHP')
prever_regressao_linguagem('R')
```

Teste R-quadrado: 0.9038697156790972

% de uso da linguagem Java



In [26]: `dados = df`
`dados.head()`

```
In [26]: dados = df
dados.head()
```

Out[26]:

	Date	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	...	PHP	Python	R	Ruby	Rust	Scala	Swift	TypeScript	VBA	Visual Basic
0	July 2004	0.34	0.36	10.08	4.71	0.43	0.0	2.82	0.0	0.03	...	18.75	2.53	0.39	0.33	0.08	0.03	0.0	0.0	1.44	8.56
1	August 2004	0.36	0.36	9.81	4.99	0.46	0.0	2.67	0.0	0.07	...	19.26	2.64	0.41	0.40	0.09	0.03	0.0	0.0	1.46	8.57
2	September 2004	0.41	0.41	9.63	5.06	0.51	0.0	2.65	0.0	0.08	...	19.49	2.72	0.40	0.41	0.10	0.03	0.0	0.0	1.55	8.41
3	October 2004	0.40	0.38	9.50	5.31	0.53	0.0	2.77	0.0	0.09	...	19.34	2.92	0.42	0.46	0.11	0.04	0.0	0.0	1.61	8.49
4	November 2004	0.38	0.38	9.52	5.24	0.55	0.0	2.76	0.0	0.07	...	19.43	2.84	0.41	0.45	0.13	0.04	0.0	0.0	1.50	8.24

5 rows × 29 columns

```
In [27]: dados['Date'] = pd.to_datetime(dados['Date'])
dados.set_index('Date', inplace = True)

dados.head()
```

Out[27]:

	Abap	Ada	C/C++	C#	Cobol	Dart	Delphi	Go	Groovy	Haskell	...	PHP	Python	R	Ruby	Rust	Scala	Swift	TypeScript	VBA	Visual Basic
Date																					
2004-07-01	0.34	0.36	10.08	4.71	0.43	0.0	2.82	0.0	0.03	0.22	...	18.75	2.53	0.39	0.33	0.08	0.03	0.0	0.0	1.44	8.56
2004-08-01	0.36	0.36	9.81	4.99	0.46	0.0	2.67	0.0	0.07	0.20	...	19.26	2.64	0.41	0.40	0.09	0.03	0.0	0.0	1.46	8.57
2004-09-01	0.41	0.41	9.63	5.06	0.51	0.0	2.65	0.0	0.08	0.21	...	19.49	2.72	0.40	0.41	0.10	0.03	0.0	0.0	1.55	8.41
2004-10-01	0.40	0.38	9.50	5.31	0.53	0.0	2.77	0.0	0.09	0.20	...	19.34	2.92	0.42	0.46	0.11	0.04	0.0	0.0	1.61	8.49
2004-11-01	0.38	0.38	9.52	5.24	0.55	0.0	2.76	0.0	0.07	0.24	...	19.43	2.84	0.41	0.45	0.13	0.04	0.0	0.0	1.50	8.24

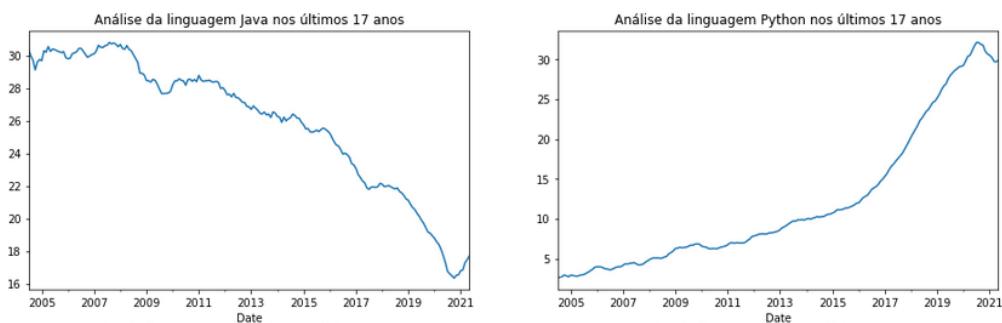
5 rows × 28 columns

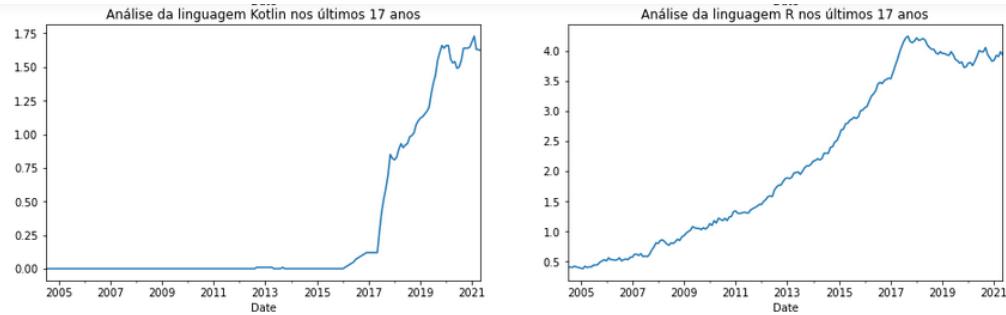
```
In [29]: fig, axes = plt.subplots(nrows=7, ncols=2)

dados['Java'].plot(ax=axes[0,0], title = "Análise da linguagem Java nos últimos 17 anos")
dados['JavaScript'].plot(ax=axes[1,0], title = "Análise da linguagem JavaScript nos últimos 17 anos")
dados['Python'].plot(ax=axes[0,1], title = "Análise da linguagem Python nos últimos 17 anos")
dados['C#'].plot(ax=axes[1,1], title = "Análise da linguagem C# nos últimos 17 anos")
dados['PHP'].plot(ax=axes[2,0], title = "Análise da linguagem PHP nos últimos 17 anos")
dados['Delphi'].plot(ax=axes[2,1], title = "Análise da linguagem Delphi nos últimos 17 anos")
dados['Dart'].plot(ax=axes[3,0], title = "Análise da linguagem Dart nos últimos 17 anos")
dados['Cobol'].plot(ax=axes[3,1], title = "Análise da linguagem Cobol nos últimos 17 anos")
dados['Go'].plot(ax=axes[4,0], title = "Análise da linguagem Go nos últimos 17 anos")
dados['C/C++'].plot(ax=axes[4,1], title = "Análise da linguagem C/C++ nos últimos 17 anos")
dados['Groovy'].plot(ax=axes[5,0], title = "Análise da linguagem Groovy nos últimos 17 anos")
dados['TypeScript'].plot(ax=axes[5,1], title = "Análise da linguagem TypeScript nos últimos 17 anos")
dados['Kotlin'].plot(ax=axes[6,0], title = "Análise da linguagem Kotlin nos últimos 17 anos")
dados['R'].plot(ax=axes[6,1], title = "Análise da linguagem R nos últimos 17 anos")

plt.gcf().set_size_inches(16, 36)

plt.savefig('imgs/Comparação linguagens 17 anos')
plt.show()
```





Analisando dados de 2017

```
In [30]: dados_2017 = pd.read_csv('dados/user-languages.csv')\n        [['user_id', 'java', 'javascript', 'typescript', 'php', 'python', 'c#', 'go']]\n\ndados_2017.head()
```

```
Out[30]:
```

	user_id	java	javascript	typescript	php	python	c#	go
0	007iva	0.015326	0.049808	0.000000	0.0	0.010536	0.0000	0.005747
1	06wj	0.000000	0.327881	0.015613	0.0	0.028996	0.0171	0.000000
2	Observer07	0.000000	0.011375	0.000000	0.0	0.185109	0.0000	0.000000
3	Orca	0.018692	0.000000	0.000000	0.0	0.000000	0.0000	0.000000
4	0x00A	0.000000	0.268152	0.000000	0.0	0.000000	0.0000	0.000000

```
In [31]: dados_2017.dtypes
```

```
Out[31]:
```

	user_id	object
java	float64	
javascript	float64	
typescript	float64	
php	float64	
python	float64	
c#	float64	
go	float64	
dtype:	object	

```
In [32]: dados_2017.info()
```

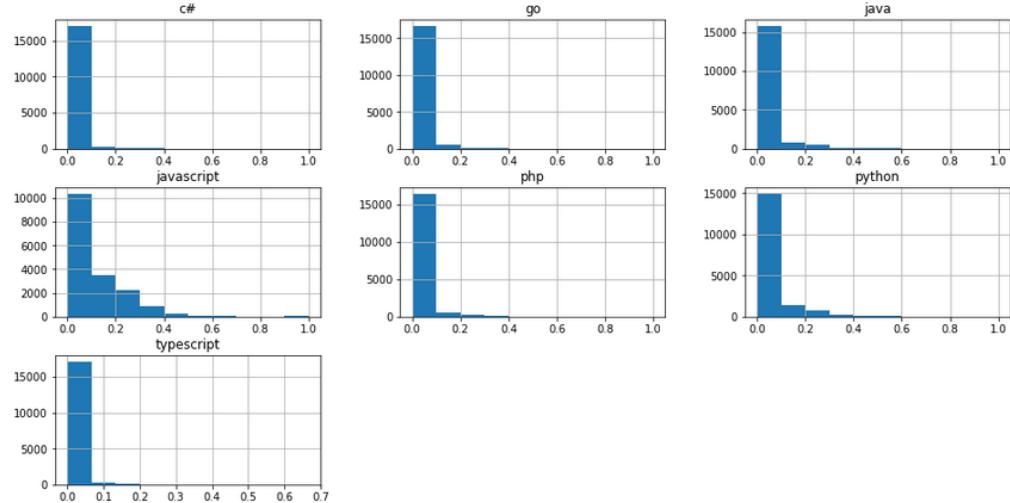
```
<class 'pandas.core.frame.DataFrame'>\nRangeIndex: 17461 entries, 0 to 17460\nData columns (total 8 columns):\nuser_id    17461 non-null object\njava       17461 non-null float64\njavascript 17461 non-null float64\ntypescript  17461 non-null float64\nphp        17461 non-null float64\npython     17461 non-null float64\nc#         17461 non-null float64\ngo        17461 non-null float64\ndtypes: float64(7), object(1)\nmemory usage: 1.1+ MB
```

```
In [33]: dados_2017.describe()
```

```
Out[33]:
```

	java	javascript	typescript	php	python	c#	go
count	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000	17461.000000
mean	0.028533	0.108883	0.004245	0.017286	0.042778	0.007451	0.012977
std	0.076522	0.122673	0.021715	0.055088	0.088514	0.040868	0.043427
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.010870	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.067114	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.016990	0.174345	0.000000	0.001043	0.042476	0.000000	0.000000

```
In [34]: dados_2017.hist()
configurar_plot('', '', 'Histograma das linguagens a serem analisadas')
```



```
In [35]: dft = pd.DataFrame(
    {
        'Language': list(map(lambda i: i, dados_2017.columns[1:])),
        'Value':     list(map(lambda i: dados_2017[i].sum(), dados_2017.columns[1:]))
    }
)

dft.head()
```

Out[35]:

	Language	Value
0	java	498.222018
1	javascript	1901.207143
2	typescript	74.127341
3	php	301.829051
4	python	746.939461

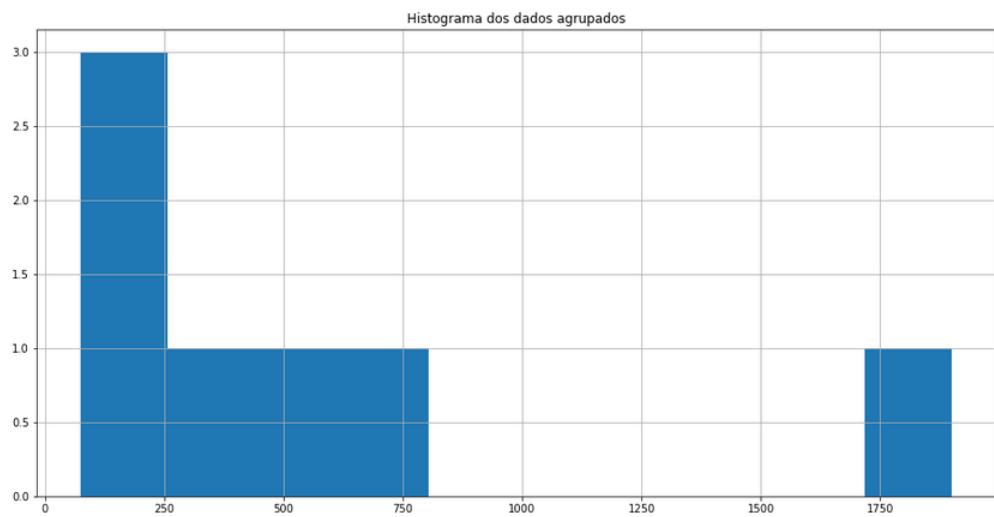
```
In [36]: dft.describe()
```

Out[36]:

	Value
count	7.000000
mean	554.144057
std	637.189883
min	74.127341
25%	178.341690
50%	301.829051
75%	622.580740
max	1901.207143

```
In [37]: dft.hist()
configurar_plot('Histograma dos dados agrupados', '', '')
```

```
In [37]: dft.hist()
configurar_plot('Histograma dos dados agrupados', '', '')
```



```
In [38]: dft = dft.sort_values(by=['Value'], ascending=False)

fig, ax = plt.subplots()

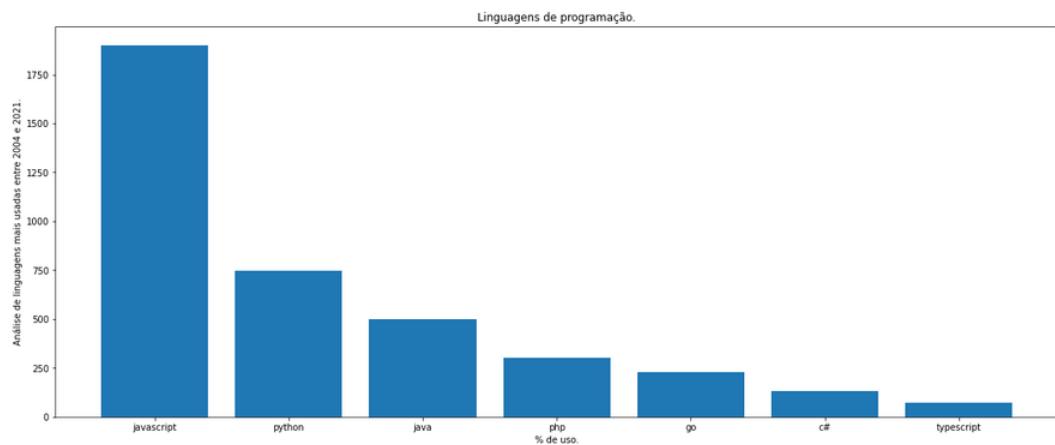
ax.set_xticklabels(dft['Language'])

ax.bar(x = dft['Language'], height = dft['Value'])

plt.gcf().set_size_inches(35, 10)

plt.savefig('imgs/Analise Linguagens Dataset 02.png')
```

```
configurar_plot_com_dimensoes(
    'Linguagens de programação.',
    '% de uso.',
    'Análise de linguagens mais usadas entre 2004 e 2021.',
    20,
    8
)
```



```
In [39]: targets = [
    {
        'linguagem_1': 'java',
        'linguagem_2': 'javascript'
    },
    ...
]
```

```

        {
            'linguagem_1': 'java',
            'linguagem_2': 'python'
        },
        {
            'linguagem_1': 'javascript',
            'linguagem_2': 'python'
        },
        {
            'linguagem_1': 'javascript',
            'linguagem_2': 'typescript'
        },
        {
            'linguagem_1': 'java',
            'linguagem_2': 'php'
        },
        {
            'linguagem_1': 'java',
            'linguagem_2': 'c#'
        },
        {
            'linguagem_1': 'php',
            'linguagem_2': 'python'
        },
        {
            'linguagem_1': 'php',
            'linguagem_2': 'javascript'
        }
    ]
}

In [40]: def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = dados_2017[linguagem_1].corr(dados_2017[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    dados_2017.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')
    configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), 11, 11, 20, 10)

for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])
<
Verificando a correlação Pearson entre os % de uso das linguagens: java e javascript.
-0.19341412138059294
java e javascript possuem correlação linear negativa.

Verificando a correlação Pearson entre os % de uso das linguagens: java e python.
-0.0679272004455616
java e python possuem correlação linear negativa.

Verificando a correlação Pearson entre os % de uso das linguagens: javascript e python.
-0.2122198553761165
javascript e python possuem correlação linear negativa.

Verificando a correlação Pearson entre os % de uso das linguagens: javascript e typescript.
0.04626619040646336
javascript e typescript possuem correlação linear positiva.

Verificando a correlação Pearson entre os % de uso das linguagens: java e php.
-0.07461796891834088
java e php possuem correlação linear negativa.

In [41]: def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = dados_2017[[linguagem_1, linguagem_2]]

```

```
In [41]: def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = dados_2017[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

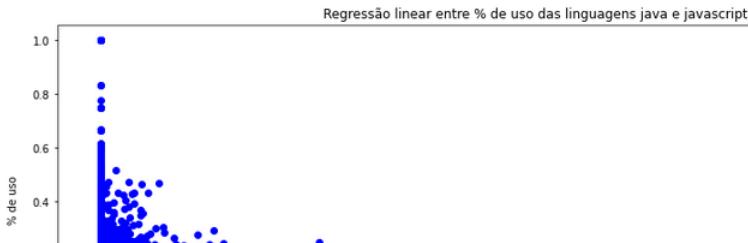
    print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

    plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
    plt.ylabel('% de uso')

    plt.show()

for target in targets:
    prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])

Teste R-quadrado: 0.038984259093661544
```



Analisando o dataset do Stackoverflow

Análise da Query no Stackoverflow:

<https://data.stackexchange.com/stackoverflow/query>

```
SELECT
    YEAR(Posts.CreationDate) AS 'Year',
    MONTH(Posts.CreationDate) AS 'Month',
    Tags.tagName,
    COUNT(*) AS Question
FROM Tags
LEFT JOIN PostTags ON PostTags.TagId = Tags.Id
LEFT JOIN Posts ON Posts.Id = PostTags.PostId
WHERE
    Tags.tagName IN (
        'java',
        'javascript',
        'typescript',
        'python',
        'go',
        'c#',
        'c',
        'c++',
        'php',
        'r'
    )
    AND Posts.CreationDate <= '2021-12-31'
GROUP BY
    YEAR(Posts.CreationDate), MONTH(Posts.CreationDate), Tags.TagName
ORDER BY
    YEAR(Posts.CreationDate), MONTH(Posts.CreationDate) DESC
```

```
In [42]: df_so = pd.read_csv('dados/stack_overflow/QueryResults.csv')
df_so.head()
```

```
Out[42]:
   Year Month tagName Question
0  2008     12        c      189
1  2008     12  javascript    626
2  2008     12       c#    1595
3  2008     12      c++    632
4  2008     12     python    440
```

```
In [43]: df_so.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1545 entries, 0 to 1544
Data columns (total 4 columns):
Year      1545 non-null int64
Month     1545 non-null int64
tagName   1545 non-null object
Question  1545 non-null int64
dtypes: int64(3), object(1)
memory usage: 48.4+ KB
```

```
In [44]: df_so = df_so.dropna()
```

```
In [45]: df_so.tail()
```

```
Out[45]:
   Year Month tagName Question
1540  2021     1     python    26295
1541  2021     1         c     2728
1542  2021     1         r     5032
1543  2021     1  typescript    3044
```

```
In [46]: df_so.describe()
```

```
Out[46]:
   Year      Month      Question
count  1545.000000  1545.000000  1545.000000
mean   2014.993528   6.604531  6932.346278
std    3.809071    3.464603  6340.638251
min    2008.000000  1.000000  1.000000
25%   2012.000000  4.000000  1908.000000
50%   2015.000000  7.000000  4586.000000
75%   2018.000000 10.000000 11484.000000
max    2021.000000 12.000000 29530.000000
```

```
In [47]: df_so.Year
```

```
Out[47]:
0      2008
1      2008
2      2008
3      2008
4      2008
...
1540    2021
1541    2021
1542    2021
1543    2021
1544    2021
Name: Year, Length: 1545, dtype: int64
```

```
In [48]: def tratar_nome(nome):
    if (nome == 'java'):
        return 'Java'

    if (nome == 'javascript'):
        return 'JavaScript'

    if (nome == 'typescript'):
        return 'TypeScript'

    if (nome == 'python'):
        return 'Python'

    if (nome == 'php'):
        return 'PHP'

    if (nome == 'c'):
        return 'C/C++'

    if (nome == 'c++'):
        return 'C/C++'

    if (nome == 'c#'):
        return 'C#'

    if (nome == 'go'):
        return 'Go'

    if (nome == 'r'):
        return 'R'

    return ''

dados_tratados_so = pd.DataFrame(
    {
        'Year': df_so['Year'],
        'Month': df_so['Month'],
        'Language': list(map(lambda x: tratar_nome(x), df_so['tagName'])),
        'Value': df_so['Question']
    }
)
dados_tratados_so.head()
```

Out[48]:

	Year	Month	Language	Value
0	2008	12	C/C++	189
1	2008	12	JavaScript	626
2	2008	12	C#	1595
3	2008	12	C/C++	632
4	2008	12	Python	440

```
In [49]: dados_tratados_so_gb = dados_tratados_so\
    .groupby(by=['Language'], as_index=False)\\
    .sum()\\
    .sort_values(by=['Value'], ascending=False)

fig, ax = plt.subplots()

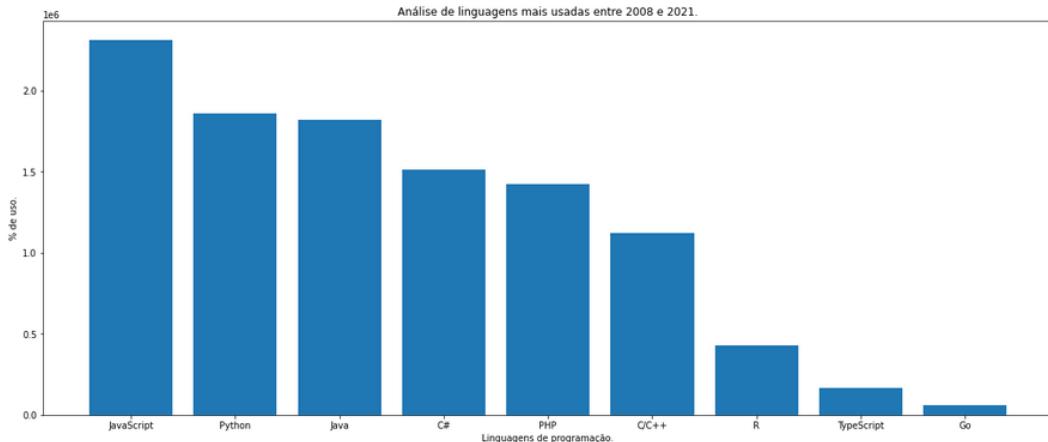
ax.set_xticklabels(dados_tratados_so_gb['Language'])

ax.bar(x = dados_tratados_so_gb['Language'], height = dados_tratados_so_gb['Value'])

plt.gcf().set_size_inches(35, 10)

plt.savefig('imgs/Analise Linguagens Dataset 02.png')

configurar_plot_com_dimensoes(
    'Análise de linguagens mais usadas entre 2008 e 2021.',
    'Linguagens de programação.',
    '% de uso.',
    20,
    8
)
```



```
In [136]: def find_by_date_and_language(year, month, language):
    value = dados_tratados_so[
        (dados_tratados_so['Language'] == language)
        & (dados_tratados_so['Year'] == year)
        & (dados_tratados_so['Month'] == month)
    ]['Value']
    if (value.empty()):
        return 0
    return value.values[0]

transposed_data = []

for year in dados_tratados_so['Year'].unique():
    for month in dados_tratados_so['Month'].unique():
        data = {
            'Year': year,
            'Month': month,
            'JavaScript': find_by_date_and_language(year, month, 'JavaScript'),
            'TypeScript': find_by_date_and_language(year, month, 'TypeScript'),
            'Python': find_by_date_and_language(year, month, 'Python'),
            'Java': find_by_date_and_language(year, month, 'Java'),
            'C#': find_by_date_and_language(year, month, 'C#'),
            'PHP': find_by_date_and_language(year, month, 'PHP'),
            'C/C++': find_by_date_and_language(year, month, 'C/C++'),
            'R': find_by_date_and_language(year, month, 'R'),
            'Go': find_by_date_and_language(year, month, 'Go')
        }
        transposed_data.append(data)

correlation_df = pd.DataFrame(transposed_data, columns = [
    'Year',
    'Month',
    'JavaScript',
    'TypeScript',
    'Python',
    'Java',
    'C#',
    'PHP',
    'C/C++',
    'R',
    'Go'
])
correlation_df.tail(10)
```

```
Out[136]:
```

	Year	Month	JavaScript	TypeScript	Python	Java	C#	PHP	C/C++	R	Go
158	2021	10	16313	3333	23076	7842	6077	4319	4103	4759	763
159	2021	9	16714	3237	21998	7817	5770	4326	3922	4332	662
160	2021	8	17058	3265	23142	7958	5764	4592	1622	4575	786
161	2021	7	17349	3163	24127	8340	5934	4618	3787	4677	650
162	2021	6	17621	3188	24532	9202	6419	4858	1996	4942	644
163	2021	5	18404	3129	26032	9675	6533	5065	2246	5393	698

```
In [118]: targets = [
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'JavaScript',
        'linguagem_2': 'TypeScript'
    },
    {
        'linguagem_1': 'Java',
        'linguagem_2': 'C/C++'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Java'
    },
    {
        'linguagem_1': 'PHP',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'Python'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'JavaScript'
    },
    {
        'linguagem_1': 'R',
        'linguagem_2': 'TypeScript'
    }
]

def verificar_correlacao(linguagem_1, linguagem_2):
    print('Verificando a correlação Pearson entre os % de uso das linguagens: {} e {}'.format(linguagem_1, linguagem_2))
    corr = correlation_df[linguagem_1].corr(correlation_df[linguagem_2])
    result = ''
    print(corr)
    if (corr == 1):
        result = 'correlação linear positiva perfeita'
    if (corr > 0):
        result = 'correlação linear positiva'
    if (corr == 0):
        result = 'correlação linear inexistente'
    if (corr == -1):
        result = 'correlação linear negativa perfeita'
    if (corr < 0):
        result = 'correlação linear negativa'
    print('{} e {} possuem {}'.format(linguagem_1, linguagem_2, result))

def plotar_correlacao(linguagem_1, linguagem_2):
    correlation_df.plot.scatter(x = linguagem_1, y = linguagem_2, c = 'Darkblue')
    configurar_plot_com_dimensoes('Correlação entre {} e {}'.format(linguagem_1, linguagem_2), 10, 10, 20, 10)

for target in targets:
    verificar_correlacao(target['linguagem_1'], target['linguagem_2'])

for target in targets:
    plotar_correlacao(target['linguagem_1'], target['linguagem_2'])

<   Verificando a correlação Pearson entre os % de uso das linguagens: Java e JavaScript.
0.8963935274210103
Java e JavaScript possuem correlação linear positiva.

Verificando a correlação Pearson entre os % de uso das linguagens: Java e Python.
0.41358405693198974
Java e Python possuem correlação linear positiva.
```

```
In [52]: def prever_regressao_linguagem(linguagem_1, linguagem_2):
    df_linguagem = correlation_df[[linguagem_1, linguagem_2]]

    X = df_linguagem[linguagem_1].values.reshape(-1, 1)
    y = df_linguagem[linguagem_2].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)

    plt.scatter(X_test, y_test, color='blue')
    plt.plot(X_test, y_pred, color='red', linewidth=3)

    plt.gcf().set_size_inches(16, 6)

    print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

    plt.title('Regressão linear entre % de uso das linguagens {} e {}'.format(linguagem_1, linguagem_2))
    plt.ylabel('% de uso')

    plt.show()

for target in targets:
    prever_regressao_linguagem(target['linguagem_1'], target['linguagem_2'])
```

Teste R-quadrado: 0.7975881591919097



Realizando o Merge com os dados do Github

```
In [53]: dados_tratados.head()

Out[53]:
      Date   Year Timestamp Language Value   UnixTime
0 July 2004 2004 2004-07-01 Abap  0.34 1.088640e+09
1 August 2004 2004 2004-08-01 Abap  0.36 1.091318e+09
2 September 2004 2004 2004-09-01 Abap  0.41 1.093997e+09
3 October 2004 2004 2004-10-01 Abap  0.40 1.096589e+09
4 November 2004 2004 2004-11-01 Abap  0.38 1.099267e+09
```

```
In [54]: def extraer_mes_data(data):
    if ('January' in data):
        return 1
    if ('February' in data):
        return 2
    if ('March' in data):
        return 3
    if ('April' in data):
        return 4
    if ('May' in data):
        return 5
    if ('June' in data):
        return 6
    if ('July' in data):
        return 7
    if ('August' in data):
        return 8
    if ('September' in data):
        return 9
    if ('October' in data):
        return 10
    if ('November' in data):
        return 11
    if ('December' in data):
        return 12
```

```

dados_tratados_novo = pd.DataFrame(
    {
        'Year': dados_tratados['Year'],
        'Month': list(map(lambda x: extrair_mes_data(x), dados_tratados['Date'])),
        'Language': dados_tratados['Language'],
        'Value': dados_tratados['Value']
    }
)

dados_tratados_novo = dados_tratados_novo[
    (dados_tratados_novo['Language'] == 'Java') |
    (dados_tratados_novo['Language'] == 'JavaScript') |
    (dados_tratados_novo['Language'] == 'TypeScript') |
    (dados_tratados_novo['Language'] == 'Python') |
    (dados_tratados_novo['Language'] == 'R') |
    (dados_tratados_novo['Language'] == 'C/C++)' |
    (dados_tratados_novo['Language'] == 'C#') |
    (dados_tratados_novo['Language'] == 'PHP') |
    (dados_tratados_novo['Language'] == 'Go')
]

```

In [55]:
`merge = pd.merge(dados_tratados_novo, dados_tratados_so, how='left', on=['Year', 'Month', 'Language'])
merge = merge.dropna()
merge.head()`

Out[55]:

	Year	Month	Language	Value_x	Value_y
49	2008	8	C/C++	7.82	83.0
50	2008	8	C/C++	7.82	163.0
51	2008	9	C/C++	7.94	754.0
52	2008	9	C/C++	7.94	320.0
53	2008	10	C/C++	8.10	303.0

In [56]:
`def prever_regressao_linguagem(linguagem):
 df_linguagem = merge[merge['Language'] == linguagem]

 X = df_linguagem['Value_x'].values.reshape(-1, 1)
 y = df_linguagem['Value_y'].values.reshape(-1, 1)

 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

 reg = linear_model.LinearRegression()
 reg.fit(X_train, y_train)
 y_pred = reg.predict(X_test)

 plt.scatter(X_test, y_test, color='blue')
 plt.plot(X_test, y_pred, color='red', linewidth=3)

 plt.gcf().set_size_inches(16, 6)

 print('Teste R-quadrado: {}'.format(reg.score(X_test, y_test)))

 plt.ylabel('Valor StackOverflow')
 plt.xlabel('Valor GitHub')
 plt.title('Regressão linear entre os valores do StackOverflow e do % uso do Github para linguagem {}'.format(linguagem))

 plt.show()

prever_regressao_linguagem('Java')
prever_regressao_linguagem('JavaScript')
prever_regressao_linguagem('TypeScript')
prever_regressao_linguagem('Python')
prever_regressao_linguagem('Go')
prever_regressao_linguagem('PHP')
prever_regressao_linguagem('C/C++)
prever_regressao_linguagem('C#')
prever_regressao_linguagem('R')`

< Teste R-quadrado: 0.026578550204984785 >

Instanciando o algoritmo K-Nearest Neighbors para os valores do Github e StackOverflow

```
In [57]: X = merge.Value_x.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
y_pred
```

```
Out[57]: array(['C/C++', 'C/C++', 'Go', 'C/C++', 'C#', 'Go', 'C/C++', 'C/C++',
       'JavaScript', 'PHP', 'Java', 'JavaScript', 'Java', 'C#', 'R',
       'C/C++', 'C#', 'JavaScript', 'C/C++', 'C/C++',
       'JavaScript', 'Java', 'C/C++', 'Python', 'C#', 'C/C++', 'Java',
       'C/C++', 'PHP', 'R', 'C/C++', 'R', 'TypeScript', 'Java', 'Java',
       'C#', 'JavaScript', 'C/C++', 'TypeScript', 'C/C++', 'JavaScript',
       'C/C++', 'TypeScript', 'C#', 'C/C++', 'PHP', 'Go', 'Go', 'C/C++',
       'C/C++', 'C/C++', 'C/C++', 'PHP', 'Java', 'C/C++', 'C/C++', 'C#',
       'Go', 'C/C++', 'R', 'C/C++', 'C#', 'R', 'R', 'C/C++', 'C#',
       'C/C++', 'C/C++', 'R', 'C#', 'Go', 'PHP', 'C/C++',
       'C/C++', 'C/C++', 'R', 'C/C++', 'R', 'JavaScript', 'Go', 'C/C++',
       'Go', 'JavaScript', 'TypeScript', 'C/C++', 'Go', 'C/C++',
       'C/C++', 'C#', 'Go', 'JavaScript', 'C/C++', 'C/C++', 'C#',
       'Java', 'C/C++', 'C/C++', 'PHP', 'TypeScript', 'JavaScript', 'PHP',
       'R', 'C#', 'R', 'C/C++', 'C/C++', 'Java', 'PHP', 'Java', 'C#',
       'JavaScript', 'C/C++', 'C/C++', 'Java', 'C#', 'Go',
       'TypeScript', 'Java', 'JavaScript', 'R', 'C/C++', 'JavaScript',
       'C/C++', 'C/C++', 'R', 'C/C++', 'Java', 'C/C++',
       'PHP', 'C/C++', 'C#', 'C/C++', 'Java', 'PHP', 'Python', 'PHP',
       'C/C++', 'R', 'C/C++', 'C/C++', 'Java', 'C/C++', 'C/C++', 'C/C++',
       'C#', 'C/C++', 'C/C++', 'C/C++', 'Java', 'C/C++', 'C/C++', 'C/C++',
       'Python', 'Java', 'JavaScript', 'Java', 'C#', 'Python', 'Java',
       'Python', 'Java', 'C/C++', 'C#', 'C/C++', 'R', 'C/C++', 'Python',
       'JavaScript', 'PHP', 'C#', 'C/C++', 'TypeScript', 'PHP', 'R',
       'C/C++', 'C#', 'TypeScript', 'C/C++', 'JavaScript', 'PHP', 'C#',
       'Java', 'PHP', 'Java', 'C/C++', 'Java', 'R', 'JavaScript', 'Go',
       'JavaScript', 'Go', 'PHP', 'JavaScript', 'Go', 'Python', 'Java',
       'Go', 'Go', 'C/C++', 'Java', 'Go', 'C#', 'JavaScript', 'C#',
       'PHP', 'C#', 'C/C++', 'C/C++', 'Go', 'C/C++', 'JavaScript',
       'C/C++', 'Go', 'C/C++', 'R', 'Go', 'PHP', 'C/C++', 'C/C++',
       '...', ...]
```

```
In [58]: knn.predict_proba(X_test)
```

```
Out[58]: array([[0.          , 0.66666667, 0.          , ..., 0.          , 0.          ,
       0.          , 0.66666667, 0.          , ..., 0.          , 0.          ,
       0.          , 0.          , 0.33333333, ..., 0.          , 0.33333333,
       0.33333333],
      ...,
      [1.          , 0.          , 0.          , ..., 0.          , 0.          ,
       0.          , 0.          , 0.33333333, ..., 0.          , 0.33333333,
       0.33333333],
      [0.          , 0.          , 0.33333333, ..., 0.          , 0.66666667,
       0.          ]])
```

```
In [59]: accuracy_score(y_test, y_pred)
```

```
Out[59]: 0.5379876796714579
```

```
In [60]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
C#	0.38	0.35	0.36	57
C/C++	0.50	0.73	0.59	94
Go	0.62	0.78	0.69	50
Java	0.63	0.87	0.73	46
JavaScript	0.44	0.42	0.43	45
PHP	0.51	0.38	0.44	55
Python	0.44	0.14	0.21	59
R	0.73	0.73	0.73	45
TypeScript	0.57	0.36	0.44	36
accuracy			0.54	487
macro avg	0.54	0.53	0.51	487
weighted avg	0.53	0.54	0.51	487

```
In [61]: X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.35523613963039014
      precision    recall  f1-score   support

       C#      0.32     0.44     0.37      57
      C/C++    0.35     0.69     0.47      94
        Go      0.74     0.70     0.72      50
       Java      0.20     0.28     0.23      46
    JavaScript    0.45     0.42     0.44      45
        PHP      0.31     0.09     0.14      55
      Python      0.18     0.05     0.08      59
        R       0.04     0.02     0.03      45
    TypeScript    0.47     0.19     0.27      36

   accuracy                           0.36      487
  macro avg       0.34     0.32     0.31      487
weighted avg       0.34     0.36     0.32      487
```

Instanciando o algoritmo Naive Bayes para os valores do Github e StackOverflow

```
In [62]: X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

naive_bayes = GaussianNB()

pred = naive_bayes.fit(X_train, y_train)

y_pred = pred.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.4188911704312115
      precision    recall  f1-score   support

       C#      0.40     0.58     0.47      57
      C/C++    0.41     0.86     0.56      94
        Go      0.64     1.00     0.78      50
       Java      0.22     0.28     0.25      46
    JavaScript    0.44     0.53     0.48      45
        PHP      0.00     0.00     0.00      55
      Python      0.00     0.00     0.00      59
        R       0.00     0.00     0.00      45
    TypeScript    0.18     0.08     0.11      36

   accuracy                           0.42      487
  macro avg       0.25     0.37     0.30      487
weighted avg       0.27     0.42     0.32      487
```

```
In [63]: X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

naive_bayes = GaussianNB()
```

```

pred = naive_bayes.fit(X_train, y_train)

y_pred = pred.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.4188911704312115
      precision    recall  f1-score   support

       C#      0.40     0.58     0.47      57
      C/C++    0.41     0.86     0.56      94
        Go     0.64     1.00     0.78      50
       Java     0.22     0.28     0.25      46
  JavaScript    0.44     0.53     0.48      45
        PHP     0.00     0.00     0.00      55
      Python     0.00     0.00     0.00      59
        R       0.00     0.00     0.00      45
  TypeScript    0.18     0.08     0.11      36

   accuracy                           0.42      487
  macro avg       0.25     0.37     0.30      487
weighted avg     0.27     0.42     0.32      487

```

Instanciando o algoritmo Support Vector Machines (SVM) para os valores do Github e StackOverflow

```

In [64]: X = merge.Value_x.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = svm.SVC()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.48459958932238195
      precision    recall  f1-score   support

       C#      0.00     0.00     0.00      57
      C/C++    0.36     0.95     0.52      94
        Go     0.62     0.96     0.75      50
       Java     0.63     0.91     0.74      46
  JavaScript    0.00     0.00     0.00      45
        PHP     0.55     0.40     0.46      55
      Python     1.00     0.03     0.07      59
        R       0.62     0.73     0.67      45
  TypeScript    0.00     0.00     0.00      36

   accuracy                           0.48      487
  macro avg       0.42     0.44     0.36      487
weighted avg     0.43     0.48     0.37      487

```

```

In [65]: X = merge.Value_y.values.reshape(-1, 1)
y = merge.Language.values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = svm.SVC()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

0.42299794661190965
      precision    recall  f1-score   support

       C#      0.42     0.60     0.50      57
      C/C++    0.42     0.91     0.57      94
        Go     0.53     1.00     0.69      50
       Java     0.21     0.15     0.17      46
  JavaScript    0.45     0.56     0.50      45
        PHP     0.13     0.04     0.06      55
      Python     1.00     0.03     0.07      59
        R       0.00     0.00     0.00      45
  TypeScript    0.00     0.00     0.00      36

   accuracy                           0.42      487
  macro avg       0.35     0.37     0.28      487
weighted avg     0.38     0.42     0.32      487

```

```
In [66]: def realizar_previsoes(algoritmo, base, use_scalling):
    target = []
    if (base == 'github'):
        target = merge.Value_x
    else:
        target = merge.Value_y
    y = merge.Language.values.reshape(-1, 1)
    X = target.values.reshape(-1, 1)
    if (use_scalling):
        X = preprocessing.StandardScaler().fit(X).transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
    y_pred = []
    if (algoritmo == 'knn'):
        knn = KNeighborsClassifier(n_neighbors=3)
        knn.fit(X_train, y_train)
        Y_pred = knn.predict(X_test)
    if (algoritmo == 'naive_bayes'):
        naive_bayes = GaussianNB()
        naive_bayes.fit(X_train, y_train)
        Y_pred = naive_bayes.predict(X_test)
    if (algoritmo == 'svm'):
        clf = svm.SVC()
        clf.fit(X_train, y_train)
        Y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    classificacao = classification_report(y_test, y_pred)
    print('Algoritmo: {} \nDados do: {} \nScore: {} \nRelatório de classificação:{}'.format(algoritmo, base, score))
    print(classificacao)
    return score
```

Realizando previsões com normalização

```
In [67]: github_knn = realizar_previsoes('knn', 'github', True)
github_naive_bayes = realizar_previsoes('naive_bayes', 'github', True)
github_svm = realizar_previsoes('svm', 'github', True)

stackoverflow_knn = realizar_previsoes('knn', 'stackoverflow', True)
stackoverflow_naive_bayes = realizar_previsoes('naive_bayes', 'stackoverflow', True)
stackoverflow_svm = realizar_previsoes('svm', 'stackoverflow', True)

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')
```

Algoritmo: knn
 Dados do: github
 Score: 0.5420944558521561
 Relatório de classificação:

	precision	recall	f1-score	support
C#	0.38	0.35	0.36	57
C/C++	0.50	0.73	0.60	94
Go	0.63	0.82	0.71	50
Java	0.63	0.87	0.73	46
JavaScript	0.43	0.42	0.43	45
PHP	0.51	0.38	0.44	55
Python	0.44	0.14	0.21	59
R	0.73	0.73	0.73	45
TypeScript	0.62	0.36	0.46	36
accuracy			0.54	487
macro avg	0.54	0.53	0.52	487
weighted avg	0.53	0.54	0.52	487

Realizando previsões sem normalização

```
In [68]: github_knn = realizar_previsoes('knn', 'github', False)
github_naive_bayes = realizar_previsoes('naive_bayes', 'github', False)
github_svm = realizar_previsoes('svm', 'github', False)

stackoverflow_knn = realizar_previsoes('knn', 'stackoverflow', False)
stackoverflow_naive_bayes = realizar_previsoes('naive_bayes', 'stackoverflow', False)
stackoverflow_svm = realizar_previsoes('svm', 'stackoverflow', False)

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')

Algoritmo: knn
Dados do: github
Score: 0.5379876796714579
Relatório de classificação:
      precision    recall   f1-score   support
C#       0.38     0.35     0.36      57
C/C++    0.50     0.73     0.59      94
Go        0.62     0.78     0.69      50
Java      0.63     0.87     0.73      46
JavaScript  0.44     0.42     0.43      45
PHP       0.51     0.38     0.44      55
Python    0.44     0.14     0.21      59
R         0.73     0.73     0.73      45
TypeScript  0.57     0.36     0.44      36
accuracy          0.54     0.54      487
macro avg       0.54     0.53     0.51      487
weighted avg    0.53     0.54     0.51      487
```

Realizando previsões com normalização em KNN e SVM, e sem normalização em GNB

```
In [69]: github_knn = realizar_previsoes('knn', 'github', True)
github_naive_bayes = realizar_previsoes('naive_bayes', 'github', False)
github_svm = realizar_previsoes('svm', 'github', True)

stackoverflow_knn = realizar_previsoes('knn', 'stackoverflow', True)
stackoverflow_naive_bayes = realizar_previsoes('naive_bayes', 'stackoverflow', False)
stackoverflow_svm = realizar_previsões('svm', 'stackoverflow', True)

resultados = pd.DataFrame({
    'algoritmo': ['KNN', 'Naive Bayes', 'SVM'],
    'github': [github_knn, github_naive_bayes, github_svm],
    'stackoverflow': [stackoverflow_knn, stackoverflow_naive_bayes, stackoverflow_svm]
})

print(resultados.head())

resultados.plot(kind = 'bar', x='algoritmo')
configurar_plot('Análise de score x algoritmo', 'algoritmo', 'score')

Algoritmo: knn
Dados do: github
Score: 0.5420944558521561
Relatório de classificação:
      precision    recall   f1-score   support
C#       0.38     0.35     0.36      57
C/C++    0.50     0.73     0.60      94
Go        0.63     0.82     0.71      50
Java      0.63     0.87     0.73      46
JavaScript  0.43     0.42     0.43      45
PHP       0.51     0.38     0.44      55
Python    0.44     0.14     0.21      59
R         0.73     0.73     0.73      45
TypeScript  0.62     0.36     0.46      36
accuracy          0.54     0.54      487
macro avg       0.54     0.53     0.52      487
weighted avg    0.53     0.54     0.52      487
```