Projeto Integrador em Ciências de Dados e Inteligência Artificial V - Relatório acerca do software PYTHON - Tutorial - Site Data Flair

Cesar Augusto Pereira[1]

[1]Advogado. Acadêmico do curso de Ciência de Dados e Inteligência Artificial do Centro Universitário Unidombosco. Membro da Associação Brasileira de Jurimetria. Membro da Associação Brasileira de Legaltechs e Lawtechs. Membro efetivo da Comissão de Direito Digital e Proteção de Dados da OAB/PR. Sócio do Escritório Casillo Advogados.

cesaraugustopereirabr@gmail.com.

Abstract. *This mini-article presents a brief summary of the Python data analysis system, highlighting its functionalities and main characteristics. This report aims, through the Directed Study, to be based on the tutorial on data pre-processing in Python available on the Data Flair website. Before going into the merits of the Data Flair Website tutorial, we propose a brief analysis of the Python language applied to data analysis. Entering the material part of the study, the first part of the tutorial shows how to perform normalization, standardization, transformation and binarization of data using Python packages, such as Pandas and SKLearn. The second part of the tutorial shows how to visualize data using histograms, density graphs, and others, using packages such as NumPy and Matplotlib. The purpose of this report is to bring this study to bear on the operations carried out on data and graphs, such as histograms.*

Resumo. *Este mini-artigo apresenta uma breve síntese do sistema de análise de dados Python, destacando suas funcionalidades e principais características. O presente relatório tem por objetivo, atráves do Estudo Dirigido, se basear no tutorial sobre pré-processamento de dados em Python disponível no site Data Flair. Antes de entrar no mérito do tutorial do Site Data Flair, propormos uma breve análise acerca da Linguagem Python aplicada a análise de dados. Entrando na parte material do estudo, a primeira parte do tutorial mostra como realizar a normalização, padronização, transformação e binarização de dados usando pacotes de Python, como o Pandas e SKLearn. Já a Segunda parte do tutorial mostra como realizar a visualização de dados utilizando histogramas, gráficos de densidades, e outros, utilizando pacotes como o NumPy e Matplotlib. O presente relatório tem por objetivo trazer este estudo contemplando as operações realizadas sobre os dados e os gráficos, como os histogramas.*

1. Introdução ao Python

Python é uma linguagem de programação que vem ganhando popularidade nos últimos anos. É uma linguagem versátil e poderosa que pode ser usada para uma ampla variedade de tarefas. Python é uma linguagem de código aberto, o que significa que é gratuita e pode ser usada por qualquer pessoa. Ademais, Python, uma linguagem de

programação de alto nível, de código aberto e geral, tem se destacado como uma das ferramentas mais versáteis e poderosas no mundo da programação. Seu desenvolvimento iniciou no final dos anos 1980, pelas mãos de Guido van Rossum, e desde então, tem evoluído constantemente, adaptando-se às demandas crescentes da comunidade de desenvolvedores.

Histórico

Python foi desenvolvido inicialmente no final da década de 1980 pelo programador holandês Guido van Rossum. A primeira versão do Python foi lançada em 1991. Python foi originalmente projetado como uma linguagem de script para o sistema operacional Unix, mas logo se tornou popular para uma ampla variedade de tarefas.

Versões e Funcionalidades

O Python passou por diversas versões, cada uma introduzindo novos recursos e aprimoramentos. Inicialmente, a versão 1.0 foi lançada em 1994, apresentando estruturas de dados como listas e dicionários. Contudo, foi com o Python 2, lançado em 2000, que a linguagem ganhou popularidade significativa. A versão 2.7, lançada em 2010, tornou-se uma das mais utilizadas por anos, mesmo após o lançamento da versão 3.

A transição para o Python 3, iniciada em 2008, marcou um marco significativo. Embora tenha gerado desafios de compatibilidade, a versão 3.x trouxe melhorias substanciais na linguagem, como suporte a Unicode, melhor gerenciamento de memória e aprimoramentos na sintaxe.

A Versatilidade do Python na Mineração de Dados

A mineração de dados, um campo crucial na era da informação, envolve a descoberta de padrões e informações valiosas a partir de conjuntos massivos de dados. Python se destaca nesse domínio, oferecendo uma ampla gama de bibliotecas e ferramentas dedicadas à análise de dados.

1. NumPy e Pandas:

O NumPy fornece estruturas de dados eficientes para operações numéricas, enquanto o Pandas oferece estruturas de dados de alto nível, como DataFrames, facilitando a manipulação e análise de conjuntos de dados tabulares.

2. Matplotlib e Seaborn:

Matplotlib e Seaborn são bibliotecas poderosas para criar visualizações gráficas. Desde gráficos simples até visualizações complexas, essas ferramentas permitem que os analistas comuniquem insights de maneira eficaz.

3. Scikit-learn:

Scikit-learn simplifica o desenvolvimento de modelos de aprendizado de máquina. Com implementações eficientes de algoritmos, essa biblioteca é uma escolha popular para tarefas de classificação, regressão e agrupamento.

4. TensorFlow e PyTorch:

Para tarefas mais avançadas de aprendizado profundo, TensorFlow e PyTorch se destacam. Essas bibliotecas possibilitam a criação e treinamento de redes neurais, impulsionando a inovação em inteligência artificial.

Estudo de Caso – Explorando dados com Python – Análise de Vinhos Portugueses

Inicialmente o estudo nos informa que Algoritmos de Aprendizado de Máquina (Machine Learning) não funcionam tão bem com o processamento de dados brutos, neste sentido, antes devemos fazer algumas transformações nos dados para que estes possam ficar mais limpos. Neste estudo, traz como exemplo o algoritmo Random Forest, que não aceita valores nulos e portanto, para pré-processar dados, foi utilizado a biblioteca *scikit-learn* ou sklearn.

Como informado no resumo deste trabalho, na primeira parte do estudo, se faz um apanhado de técnicas de pré-processamento e dados em Aprendizado de Máquina, e no caso concreto, no presente estudo traz sete (7) técnicas, que são elas:

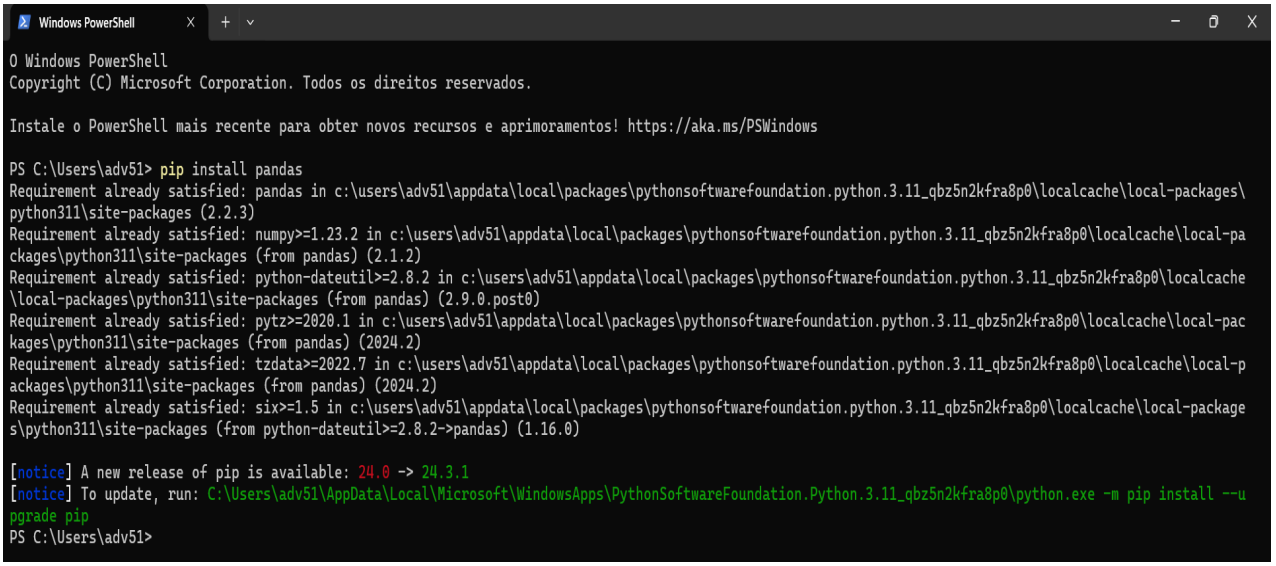
1. Redimensionamento de dados;
2. Padronização de dados;
3. Normalização de dados;
4. Binarização de dados;
5.Remoção Média;
6. Codificação Quente;e
7. Codificação de Rótulos.

Iremos explorar neste relatório, resumidamente, cada uma delas.

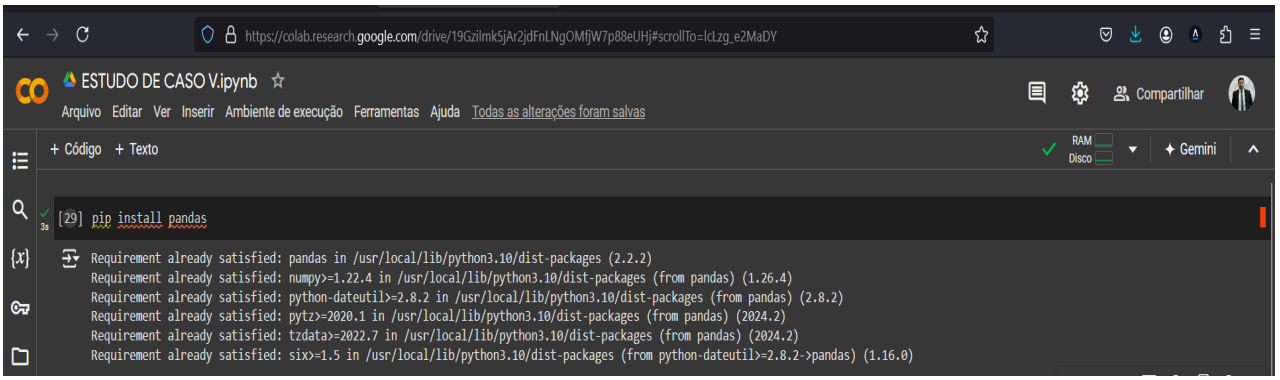
PREPARAÇÃO DO AMBIENTE

Antes de adentrarmos ao estudo de caso, foi necessário preparar a máquina para receber o arquivo CSV do estudo de caso. Inicialmente, instalamos o *Pandas* diretamente no

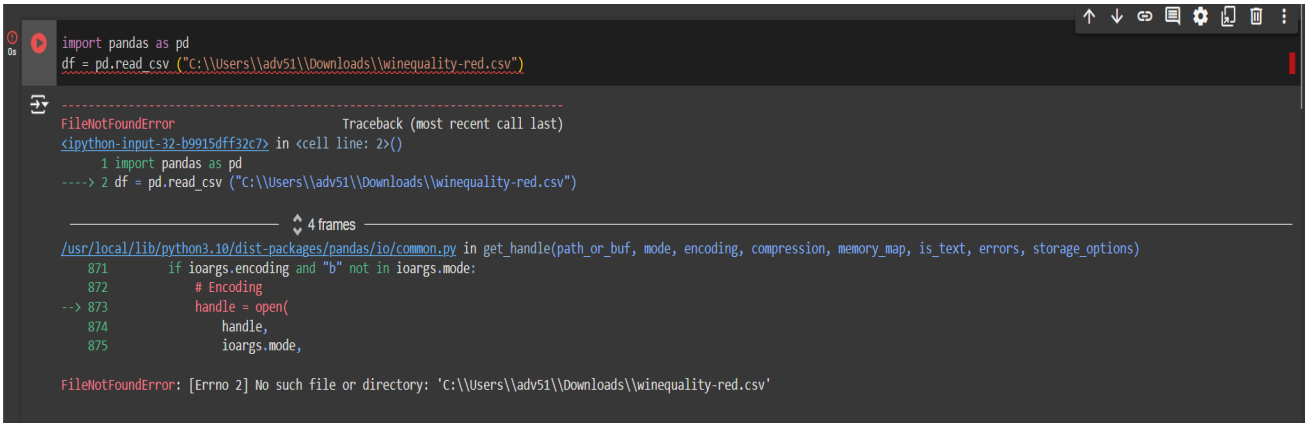
interpretador, versão 3.11 do Python na máquina, utilizando-se o código *pip install,* vejamos:



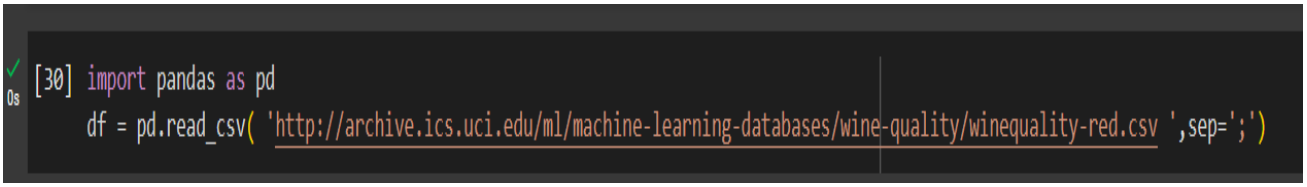
Importante destacar que foi encontrado dificuldade para usar o interpretador do python diretamente no CMD, sendo assim, partiu-se para ser feito no COLAB da Google e lá seguimos instalando o pandas, conforme abaixo:



Após, tendo em vista a dificuldade relatada no CMD, também encontramos via COLAB, porque o arquivo não era encontrado no diretório informado no código, vejamos:



Assim, partiu-se para fazer a impotação do CSV diretamente pelo site do estudo de caso, e desta forma, deu certo:



Importante destacar que estávamos usando uma máquina de terceiro, por ter maior poder computacional do que a nossa máquina pessoal, sendo assim, não sabemos informar se há algum bloqueio institucional na máquina que ocorreu esses erros mencionados acima e documentados neste relatório e nos arquivos em anexo a este relatório.

Após preparar a máquina, seguimos com o estudo de caso.

1. Redimensionamento de dados

Quando os atributos de um conjunto de dados têm escalas diferentes, podemos ajustá-los para uma mesma faixa, geralmente entre 0 e 1. Esse processo é chamado de normalização. No Python, a biblioteca scikit-learn oferece a classe MinMaxScaler para realizar essa normalização de maneira simples. Seguindo o exemplo dado no estudo diretamente no interpretador Python:

import, scipy, numpy
from sklearn.preprocessing import MinMaxScaler
array=df.values

```
#Separating data into input and output components
x=array[:,0:8]
y=array[:,8]
scaler=MinMaxScaler(feature_range=(0,1))
rescaledX=scaler.fit_transform(x)
numpy.set_printoptions(precision=3)
rescaledX[0:5,:]
```

```
array([[0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568],
       [0.283, 0.521, 0.   , 0.116, 0.144, 0.338, 0.216, 0.494],
       [0.283, 0.438, 0.04 , 0.096, 0.134, 0.197, 0.17 , 0.509],
       [0.584, 0.11 , 0.56 , 0.068, 0.105, 0.225, 0.191, 0.582],
       [0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568]])
```

2. Padronização de dados

A padronização transforma atributos que seguem uma distribuição normal, mas com médias e desvios-padrão diferentes, em uma nova distribuição com média 0 e desvio-padrão 1. Para fazer isso no Python, usamos a classe StandardScaler da biblioteca scikit-learn.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(x)
rescaledX=scaler.transform(x)
rescaledX[0:5,:]
```

```
array([[-0.528,  0.962, -1.391, -0.453, -0.244, -0.466, -0.379,  0.558],
       [-0.299,  1.967, -1.391,  0.043,  0.224,  0.873,  0.624,  0.028],
       [-0.299,  1.297, -1.186, -0.169,  0.096, -0.084,  0.229,  0.134],
       [ 1.655, -1.384,  1.484, -0.453, -0.265,  0.108,  0.412,  0.664],
       [-0.528,  0.962, -1.391, -0.453, -0.244, -0.466, -0.379,  0.558]])
```

3. Normalização de dados

Nessa abordagem, ajustamos cada observação para que seu comprimento total seja 1, também conhecida como norma unitária. No Python, usamos a classe Normalizer da biblioteca scikit-learn para realizar esse redimensionamento.

```
from sklearn.preprocessing import Normalizer
scaler=Normalizer().fit(x)
normalizedX=scaler.transform(x)
normalizedX[0:5,:]
```

```
array([[2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
        9.299e-01, 2.729e-02],
       [1.083e-01, 1.222e-02, 0.000e+00, 3.611e-02, 1.361e-03, 3.472e-01,
        9.306e-01, 1.385e-02],
       [1.377e-01, 1.342e-02, 7.061e-04, 4.060e-02, 1.624e-03, 2.648e-01,
        9.533e-01, 1.760e-02],
       [1.767e-01, 4.416e-03, 8.833e-03, 2.997e-02, 1.183e-03, 2.681e-01,
        9.464e-01, 1.574e-02],
       [2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
        9.299e-01, 2.729e-02]])
```

4. Binarização de dados

Com o uso de um limite binário, podemos transformar os dados de modo que valores acima desse limite sejam marcados como 1, e valores iguais ou abaixo sejam marcados como 0. Para realizar essa transformação no Python, utilizamos a classe Binarizer da biblioteca scikit-lear

```
from sklearn.preprocessing import Binarizer
binarizer=Binarizer(threshold=0.0).fit(x)
binaryX=binarizer.transform(x)
binaryX[0:5,:]
```

```
array([[1., 1., 0., 1., 1., 1., 1., 1.],
       [1., 1., 0., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 0., 1., 1., 1., 1., 1.]])
```

5. Remoção Média

É possível centralizar os dados subtraindo a média de cada característica, de forma que seus valores fiquem distribuídos em torno de zero. Isso é útil para muitos algoritmos de aprendizado de máquina.

```
from sklearn.preprocessing import scale
data_standardized=scale(df)
data_standardized.mean(axis=0
```

```
array([ 3.555e-16,  1.733e-16, -8.887e-17, -1.244e-16,  3.733e-16,
       -6.221e-17,  4.444e-17, -3.473e-14,  2.862e-15,  6.754e-16,
        1.066e-16,  8.887e-17])
```

6. Codificação Quente

Quando lidamos com poucas categorias numéricas dispersas, podemos optar por não armazená-las diretamente. Em vez disso, usamos One Hot Encoding, que transforma cada valor distinto em um vetor de dimensão k (onde k é o número de categorias). Nesse vetor, apenas um valor é 1 (representando a categoria presente), enquanto os demais são 0.

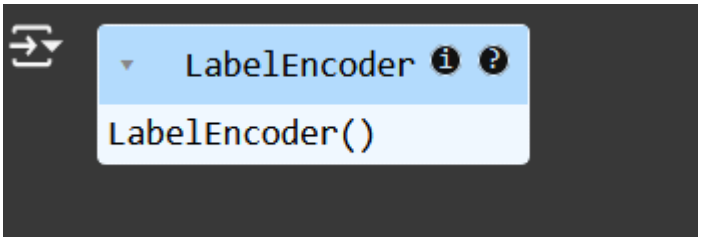
```
from sklearn.preprocessing import OneHotEncoder
encoder=OneHotEncoder()
encoder.fit([[0,1,6,2],
[1,5,3,5],
[2,4,2,7],
[1,0,4,2]
])
```

```
OneHotEncoder
OneHotEncoder()
```

7. **Codificação de Rótulos.**

Alguns rótulos podem ser representados por palavras ou números, e, para tornar os dados mais legíveis, é comum usar palavras. No entanto, para que os algoritmos possam processar esses rótulos, usamos a codificação de rótulos (Label Encoding), que converte as palavras em números, facilitando o trabalho dos modelos de aprendizado de máquina.

```
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
input_classes=['Havells','Philips','Syska','Eveready','Lloyd']
label_encoder.fit(input_classes)
```



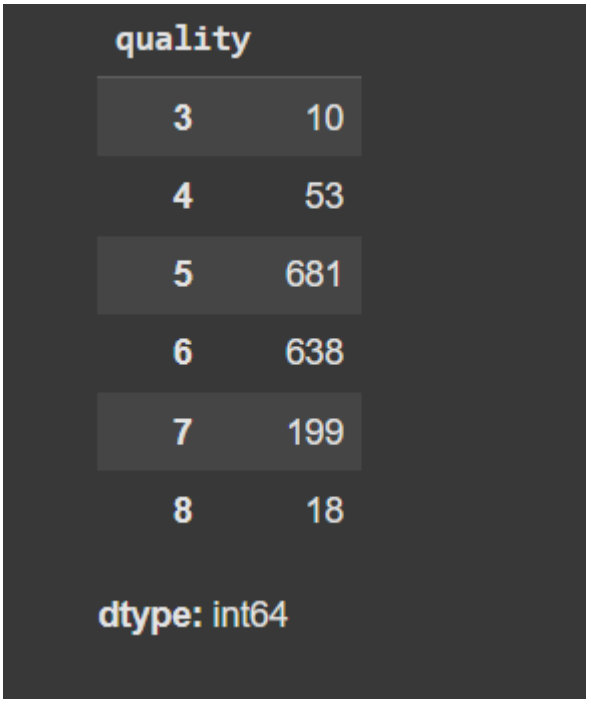
Ainda, no presente estudo, foi possível ver outras técnicas, como a descrição do conjunto de dados através do método *describe()* e da mesma forma, o método head(), ou cabeça, que nos traz a quantidade de linhas que queremos analisar, como exemplo, as 10 primeiras linhas, ficando assim:

```
df.head(10)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

Ainda, podemos executar inúmeras operações a partir de uma variável, para isso, foi visto a função *groupby()*, vejamos:

```
df.groupby('quality').size()
```



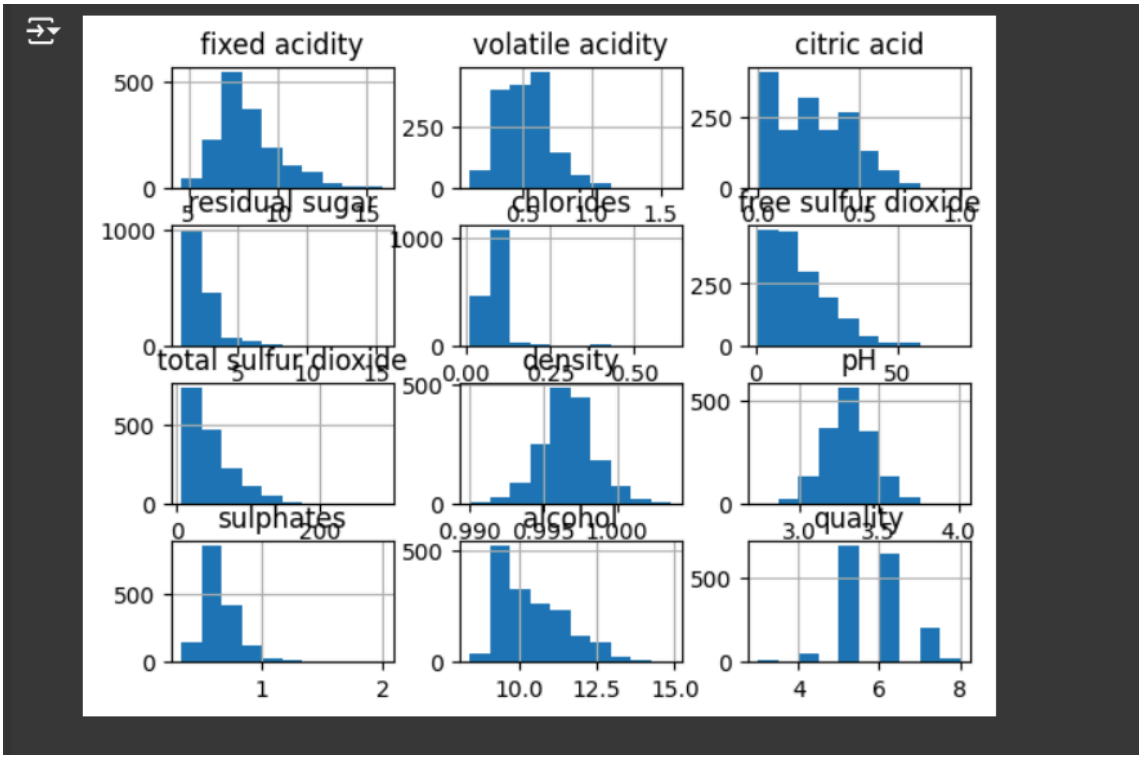
Visualizando dados a partir de Gráficos diversos

Para visualizar dados e extrair informações importantes, podemos usar o *pandas* junto com *Matplotlib* para criar gráficos e tabelas. Existem dois tipos principais de gráficos: univariados e multivariado. Um gráfico univariados foca em apenas uma variável, permitindo analisar sua distribuição ou comportamento de maneira isolada.

HISTOGRAMAS

Os histogramas são úteis em Machine Learning porque agrupam os dados em intervalos (compartimentos) e mostram quantas observações caem em cada um. Isso facilita a visualização da distribuição de um atributo. A forma dos compartimentos revela se a distribuição é gaussiana, enviesada ou exponencial, além de fornecer insights sobre possíveis outliers.

```
import matplotlib.pyplot as plt
df.hist()
plt.show()
```

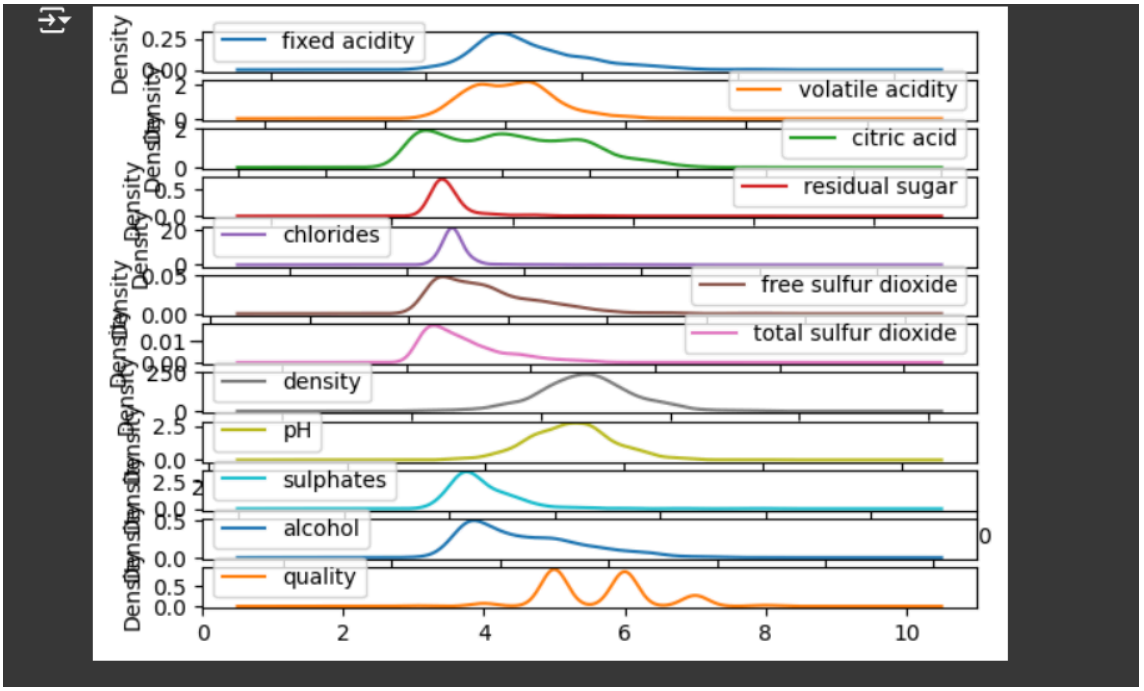


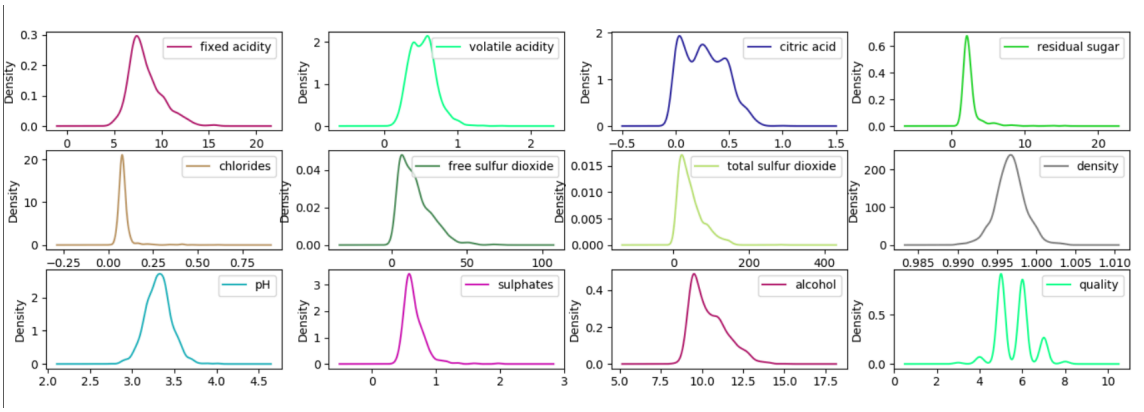
Como vimos no estudo, é possível que os atributos 'dióxido de enxofre total', 'dióxido de enxofre livre' e 'açúcar residual' apresentam uma distribuição exponencial. Por outro lado, os atributos 'densidade', 'pH', 'acidez fixa' e 'acidez volátil' podem ter distribuições gaussianas ou se aproximar de uma distribuição normal.

GRÁFICOS DE DENSIDADE

Um gráfico de densidade é semelhante a um histograma, mas apresenta uma representação mais suave dos dados. Em vez de barras, ele utiliza uma curva desenhada sobre os pontos médios de cada bin, permitindo uma visualização mais clara da distribuição dos dados. É uma forma relaxante e eficaz de entender a densidade das observações em diferentes intervalos.

```
df.plot(kind='density',subplots=True,sharex=False)
plt.show()
```





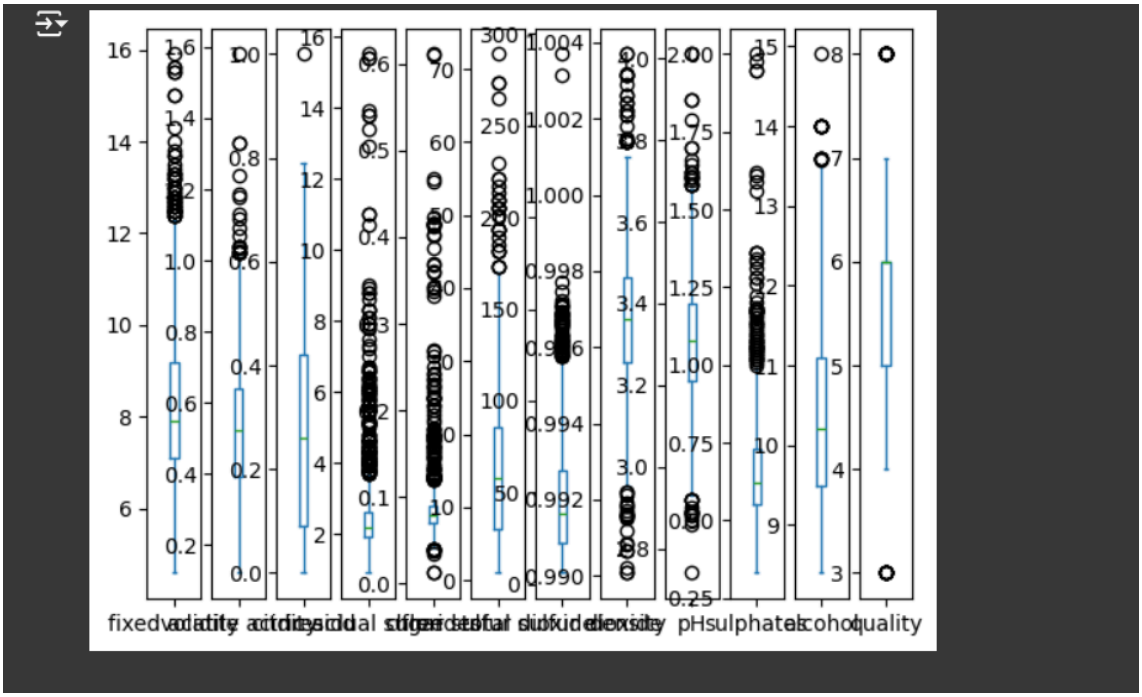
DIAGRAMAS DE CAIXA E BIGODE

Um box plot fornece um resumo visual da distribuição de cada atributo. Ele mostra uma linha representando a mediana e uma caixa que envolve os percentis 25 e 75. Os "bigodes" do gráfico indicam a dispersão dos dados, enquanto os pontos que estão além dos bigodes identificam potenciais valores atípicos (outliers). Essa representação facilita a análise da variabilidade e da presença de outliers em um conjunto de dados. Vejamos:

df.plot(kind='box',subplots=True,sharex=False,sharey=False)

```
Acidoity fixo AxesSubplot(0.125,0.11;0.0545775-0.77)
Acidentes de acidez voláteis Acrecicultura(0.190493,0.11;0.0545775 ? 0,777)
Ácido cítrico AxesSubplot (0,2559986,0.11;0.0545775 ? 0,77)
Açúcar residual AxesSubparecido(0.321479,0.11;0.0545775 ? 0,77)
cloretos EixosSubplot(0.386972,0.11;0.0545775-0.77)
Eixos de dióxido de enxofre livresSubparcelo(0.452465,0.11;0.0545775 ? 0,77)
Eixos de dióxido de enxofre totalSubparcelo (0,5517958,0,11;0.0545775 ? 0,777)
densidade Eixo Subparcelo (0,584511.1,11;0.0545775 ? 0,777)
Eixos de pHSubparecido(0.648944,0.11;0.0545775-0.77)
sulfatos EixosSubparecidos (0,714437,0,1;0.0545775 ? 0,77)
Álcool AxesSubplot(0.77993,0.11;0.0545775'0.77)
qualidade AxesSubplot(0.845423,0.11;0.0545775-0.77)
dtype: objeto
```

```
plt.show()
```



Como vimos no estudo, os atributos como 'dióxido de enxofre total', 'sulfatos' e 'açúcar residual' apresentam uma distorção em direção a valores menores, o que sugere uma distribuição assimétrica. Essa característica pode influenciar a análise e a modelagem dos dados, pois pode indicar a necessidade de transformação para uma melhor normalização ou interpretação.

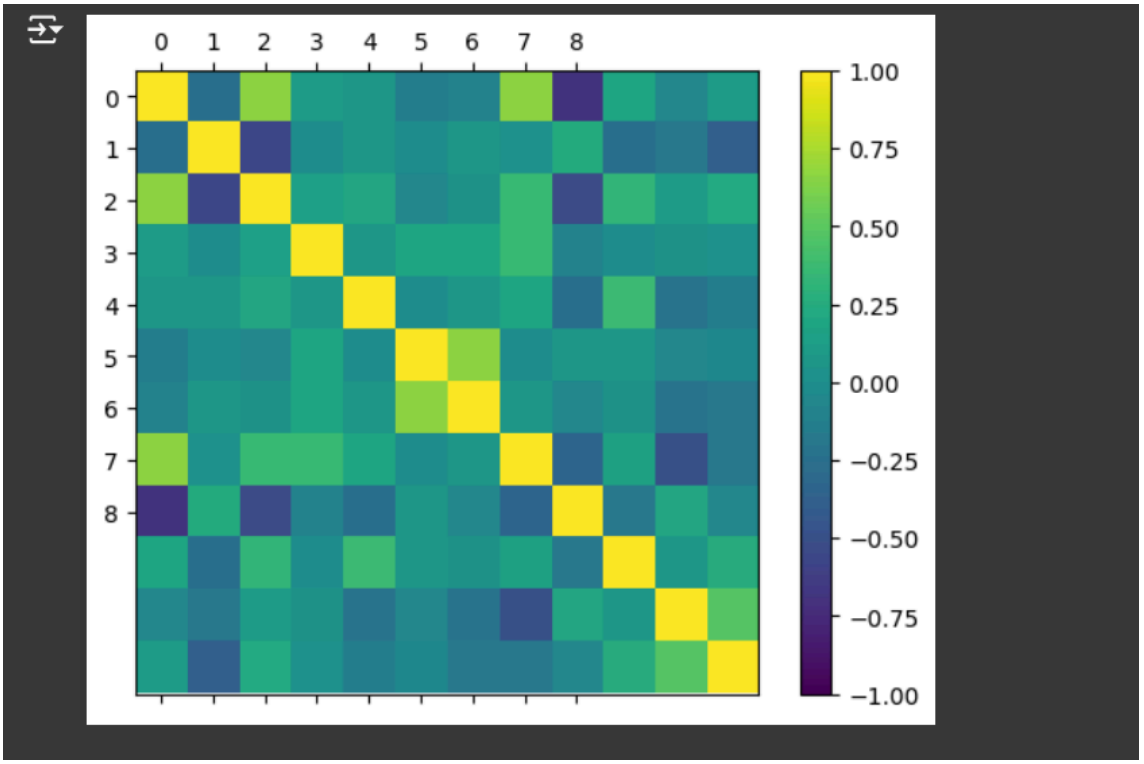
Ainda, o estudo traz uma análise multivariada, ou seja, aquela que traz mais de duas variáveis. Vejamos:

Gráfico de matriz de correlação

Esse tipo de gráfico ilustra a relação entre duas variáveis, onde uma correlação positiva indica que ambas mudam na mesma direção, enquanto uma correlação negativa significa que elas se movem em direções opostas. Para analisar essas relações de forma mais abrangente, podemos plotar uma matriz de correlação, que apresenta as correlações entre todos os pares de variáveis em um conjunto de dados. Isso nos ajuda a identificar padrões e a força das relações entre as variáveis.

```
correlations=df.corr()
fig=plt.figure()
ax=fig.add_subplot(111)
cax=ax.matshow(correlations,vmin=-1,vmax=1)
fig.colorbar(cax)

ticks=numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
plt.show()
```

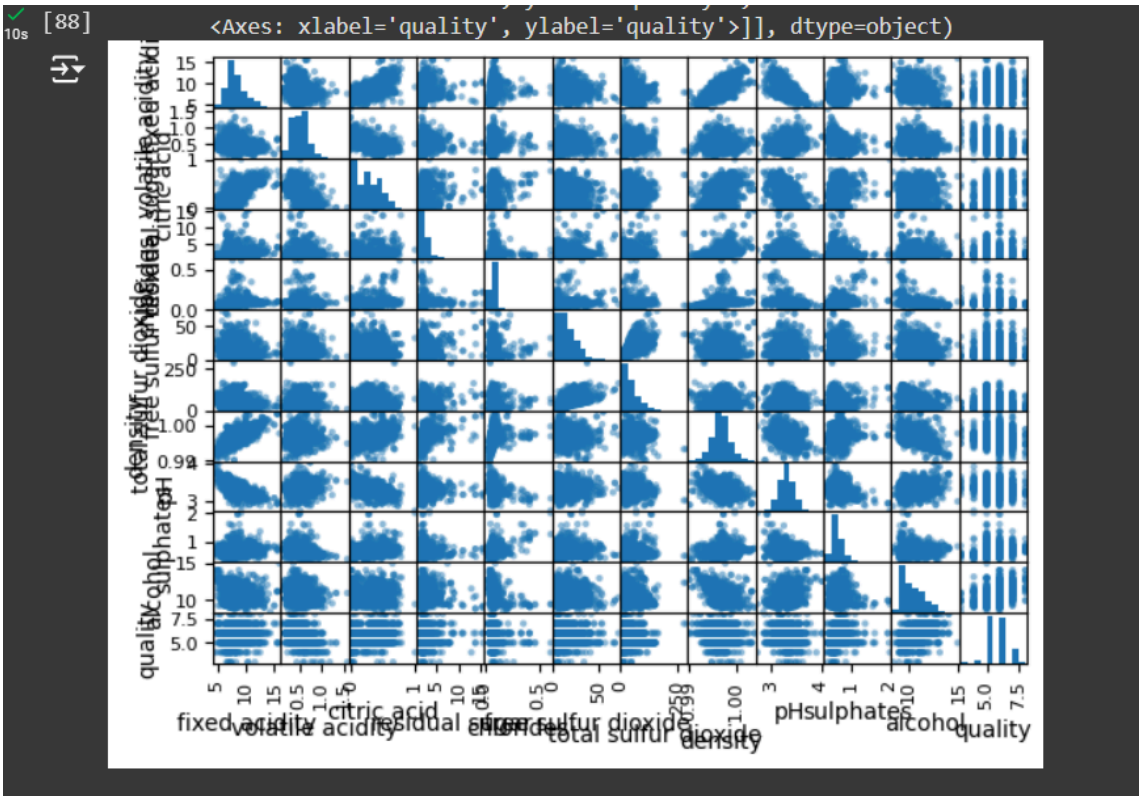


A matriz de correlação é simétrica em torno da diagonal principal, que vai do canto superior esquerdo ao canto inferior direito. Isso significa que a correlação entre a variável (X) e a variável (Y) é a mesma que a correlação entre (Y) e (X). Os valores ao longo dessa diagonal são sempre 1, pois representam a correlação de cada variável consigo mesma. Essa simetria facilita a interpretação das relações entre as variáveis no conjunto de dados.

Matriz de dispersão

As matrizes de diagrama de dispersão mostram como duas variáveis se relacionam, representando-as como pontos em um gráfico bidimensional. Ao plotar todos os pares de variáveis de um conjunto de dados em um único gráfico, obtemos uma matriz de diagramas de dispersão. Essa matriz permite visualizar rapidamente as relações entre diferentes variáveis, ajudando a identificar padrões ou relacionamentos estruturados que podem ser importantes para a análise e modelagem dos dados.

pandas.plotting.scatter_matrix(df)



A matriz de diagramas de dispersão também é simétrica, assim como a matriz de correlação. Na diagonal principal, são exibidos histogramas dos atributos, pois não faz sentido plotar um diagrama de dispersão de um atributo consigo mesmo. Os histogramas oferecem uma visualização da distribuição de cada variável, enquanto os diagramas de dispersão nas outras posições da matriz mostram as relações entre diferentes pares de variáveis. Essa combinação fornece uma visão abrangente da estrutura dos dados.

Conclusão

Python é uma linguagem de programação versátil e poderosa que pode ser usada para uma ampla variedade de tarefas. É uma linguagem popular para desenvolvimento web, ciência de dados, aprendizado de máquina, inteligência artificial, automação e muito mais. A evolução contínua do Python e sua crescente adoção em projetos de mineração de dados destacam a importância e versatilidade dessa linguagem de programação. À medida que novas versões são lançadas e a comunidade de desenvolvedores continua a expandir, podemos antever um futuro promissor para Python na análise e interpretação inteligente de dados, alimentando inovações em diversos setores. A jornada do Python na mineração de dados é apenas o começo de uma história que continua a se desdobrar, moldando o panorama tecnológico com cada linha de código. No tutorial foi visto como explorar o pré-processamento de dados, destacando técnicas como reescalonamento, normalização, binarização e padronização.

Referências Bibliográficas

BARBOSA, J. et al. Introdução ao processamento de linguagem natural usando python. **III Escola Regional de Informatica do Piauí**, v. 1, p. 336-360, 2017.

BORGES, Luiz Eduardo. **Python para desenvolvedores: aborda Python 3.3**. Novatec Editora, 2014.

DA SILVA, Rogério Oliveira; SILVA, Igor Rodrigues Sousa. Linguagem de Programação Python. **Tecnologias em Projeção**, v. 10, n. 1, p. 55-71, 2019.

MANZANO, José Augusto NG**. Introdução à linguagem Python**. Novatec Editora, 2018.

UCI Machine Learning Repository. Disponível em: https://data-flair.training/blogs/python-ml-data-preprocessing/. Acesso em 05/10/2024.