Projeto Visualização de dados

Problema de negócio

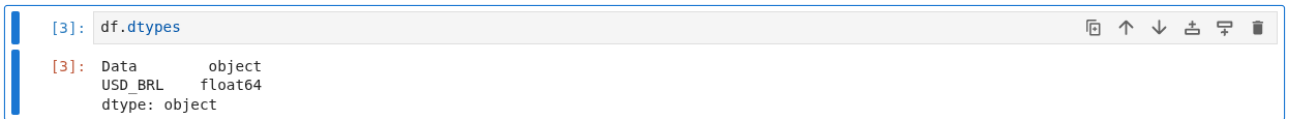
O problema proposto foi, criar três tipo de visualizações diferentes para representer o mesmos dados, vale resaltar que também os gráficos utilizados devem respeitar a natureza dos dados. Dada a situação, foi optado por usar os seguintes gráficos: **Linhas**, **dispersão** e **Historiograma.**

Idle e Módulos utilizados

Todo o projeto foi feito utilizando a IDE **JupyterLab,** já quanto aos módulos utilizados foram: **Pandas** e **Seaborn**.

Normalização e Padronização

O primeiro passo a ser tomado em relação ao dados foi primeiro ver como o dataset se portava, ou seja entender sua estrutura e ver se o tipo atribuído as variáveis realmente eram adquados as variáveis. Com isso meu primeiro passo foi ver os tipos de dados deste dataset.



Bom, como podemos ver a variável **Data** está sendo tratado como tipo **object**, por padrão o Pandas trata todo tipo de string como object. Para converter Data como um tipo adequado podemos usar o método **to_datetime** do Pandas.

Basicamento o que fizemos aqui foi referênciar o campo que queremos converter no método **to_datetime.**

Certo, apesar de ter resolvido o problema, acabou que o problema em si não foi resolvido... Sim eu irei explicar, acontece que optei por separar a Data para **Dia**, **Mês** e **Ano** utilizando o parâmetro .map() de pandas, o que acontece e que ao fazer isso, cada nova coluna irá se portar como tipo inteiro... Sim eu sei, no final acabei pensando o mesmo que você caro leitor.

```
[6]: df['Dia'] = df['Data'].map(lambda x: x.day)
[7]: df['Mes'] = df['Data'].map(lambda x: x.month)
[8]: df['Ano'] = df['Data'].map(lambda x: x.year)
[16]: df.dtypes
[16]: USD_BRL    float64
      Dia          int64
      Mes          int64
      Ano          int64
      dtype: object
```

Porém, como apenas o ano será usado nos três plots, acredito que isso não será um problema. O nosso próximo passo a seguir e definir um limite ao número de casas decimais no campo **USD_BRL**, para isso utilizei o método **.round** para impor o limite de duas casas decimais.

```
[5]: df['USD_BRL'] = df['USD_BRL'].round(2)
```

Por último mas não menos importante, a variável Data foi excluída uma vez que ela foi rafimicada em três novas variáveis.

Visualizando Dados

Bom seguindo para a parte de visualização de dados, agora iremos usar três tipo de charts para representar o mesmo tipo de dados, como já dito anteriormente, os charts escolhidos foram os seguintes: Gráfico de Linhas, Gráfico de Dispersão e Histograma. Para este projeto eu optei pelo uso do pacote **Seaborn** uma vez que estou mais familiarizado com este pacote.

Gráfico de Linhas

Logo abaixo você consegue ver a instrução passada para plotar o gráfico de linhas.

```
sns.lineplot(df, x=df['Ano'], y=df['USD_BRL'])
```

Certo, vamos explicar o passo a passo de instrução usada acima.

- df: indica a referência do dataset de onde queremos plotar nossos dados
- x: indica que coluna queremos referênciar para o eixo das abscissas, no nosso caso df[‘Ano’]
- y: indica que coluna queremos refênciar para o eixo das ordenadas, no nosso caso df[‘USD_BRL’]

O resultado a seguir vemos logo abaixo:

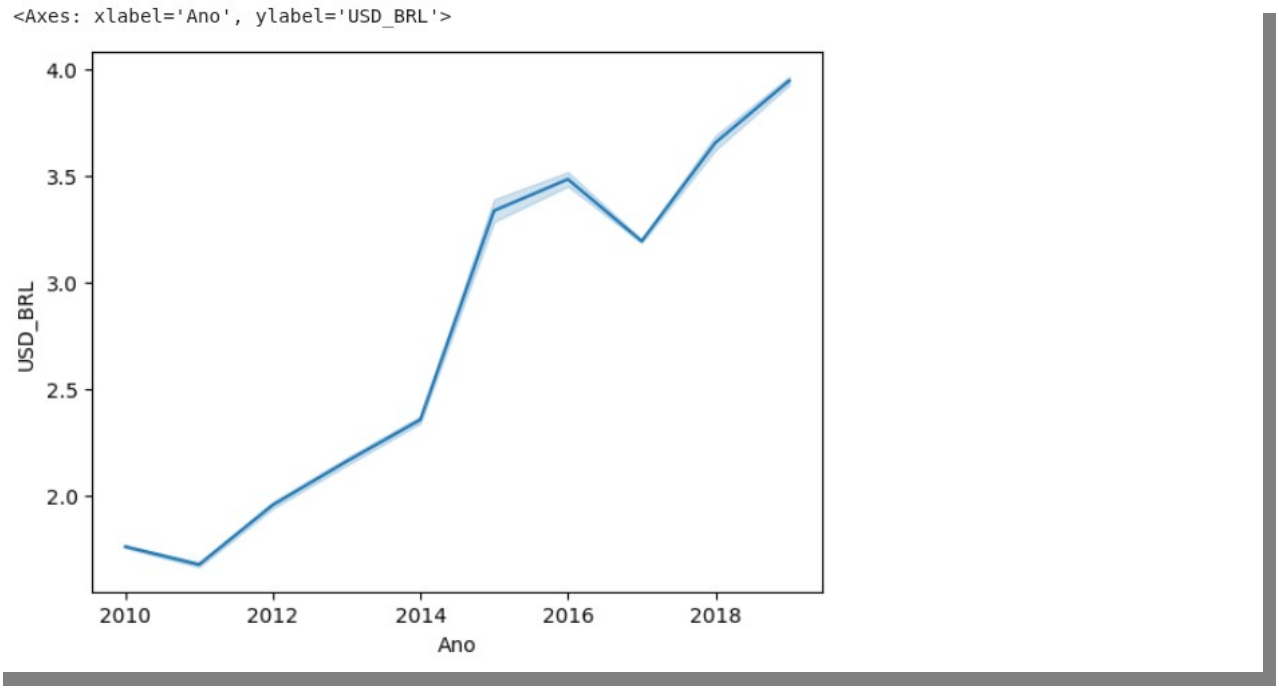
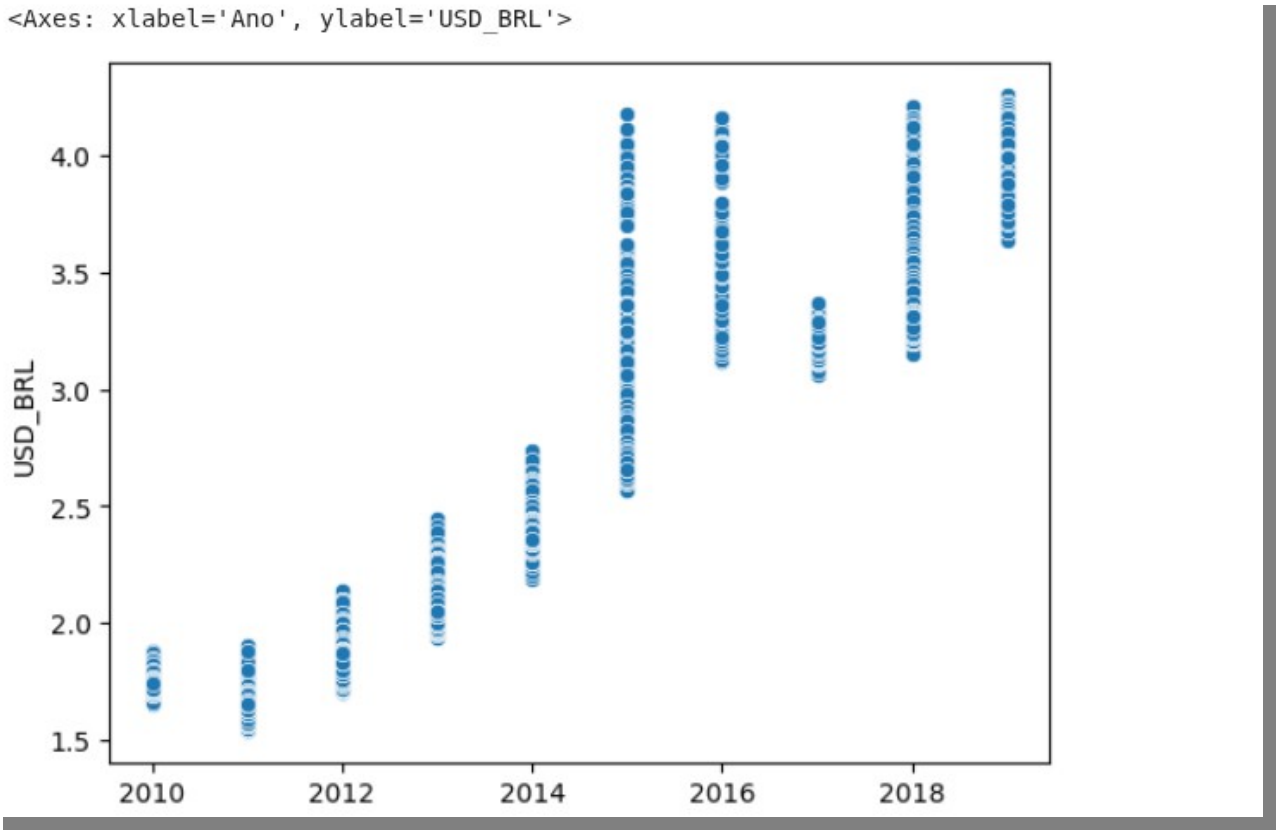


Gráfico de Dispersão

A instrução a seguir foi usada para plotar o gráfico de dispersão.

```
sns.scatterplot(df, x=df['Ano'], y=df['USD_BRL'])
```

O mesmo padrão utilizado para o gráfico de linhas se adota ao gráfico de dispersão. Logo abaixo você podera ver como ficou o resultado do gráfico de dispersão.



Histograma

Aqui os padrões mudam um pouco, porém, nada que possamos resolver com um pouco de esforço.

```
[16]: sns.histplot(df, x=df['USD_BRL'], hue=df['Ano'])
```

Bom vamos explicar cada passo tomado para gerar a instrução acima:

- Referênciamos df como o dataset que plotar os dados do eixo x e hue
- Definimos qual será o eixo x, nosso caso df[‘USD_BRL’]
- Logo após iremos referênciar a faixa de valores para hue no nosso caso df[‘Ano’]

O resultado você poderá ver logo abaixo.