Gabriel Henrique Alves Spacki **09/12/2023**

Prática: Principais Bibliotecas e Ferramentas Python para Aprendizado de Máquina

Durante essa tarefa prática, fui introduzido às bibliotecas NumPy e Pandas, fundamentais para a análise de dados em Python. Os módulos assistidos mostraram a utilização da ferramenta Jupyter, ambiente que permite escrever, executar e depurar códigos interativamente. Ao assistir os vídeos do curso apresentado, escrevi uma mini documentação dos principais atributos e métodos ensinados. Segue abaixo as anotações:

NumPy:

- **numpy.array():** Usado para criar um array, que é uma estrutura de dados multidimensional. Um array em NumPy é semelhante a uma lista em Python, mas oferece recursos adicionais e é otimizado para operações numéricas eficientes.

- **numpy.arange():** Retorna um array com valores espaçados uniformemente em um intervalo especificado, sendo uma alternativa conveniente para a função range() do Python.

- **numpy.zeros():** Inicializa um array com zeros para preencher com dados durante a execução do programa.

- **numpy.ones():** De maneira semelhante ao numpy.zeros(), Inicializa um array preenchido com o número um.

- **numpy.eye():** Gera uma matriz identidade, que é uma matriz quadrada com todos os elementos iguais a zero, exceto os elementos da diagonal principal, que são iguais a um.

- **numpy.linspace():** Também usada para criar um array de valores espaçados uniformemente dentro de um intervalo especificado. Entretanto, possibilita indicar quantos valores serão gerados dentro do array.

- **numpy.random:** A biblioteca NumPy oferece o sub módulo random para geração de números pseudoaleatórios. Dentro desse módulo, encontramos diversas funções para criar arrays ou valores individuais com base em distribuições de probabilidade.

- **numpy.random.rand():** Gera um array com valores aleatórios em uma distribuição uniforme no intervalo [0, 1)

- **numpy.random.randn():** De maneira parecida, é utilizada para gerar valores aleatórios. Dessa vez, em uma distribuição normal padrão, ou seja, uma distribuição com média 0 e desvio padrão 1.

- **numpy.random.randint():** Gera um array com números inteiros aleatórios.

- **reshape():** Reajusta a forma de um array para que ele tenha as dimensões desejadas para cálculos ou visualização de dados.

- **shape:** Retorna a forma (dimensões) de um array, uma tupla de inteiros representando o número de elementos em cada dimensão

- **max():** Encontra o valor máximo ao longo de um eixo específico de um array.

- **min():** Encontra o valor mínimo ao longo de um eixo específico de um array.

- **argmax():** Retorna o índice em que o maior elemento do array se encontra ao longo de um eixo específico.

- **argmin():** Retorna o índice em que o menor elemento do array se encontra ao longo de um eixo específico.

Pandas:

- **pandas.Series():** Cria um objeto chamado “Series”, uma estrutura unidimensional semelhante a um dicionário python. Cada elemento é associado a um “rótulo” ou uma “chave”.

- **pandas.DataFrame():** Utilizado para criar um objeto chamado "DataFrame". Um DataFrame é uma estrutura de dados bidimensional, semelhante a uma tabela ou planilha.

- **drop():** Remove uma linha ou uma coluna de um DataFrame.

- **loc[]:** Acessa linhas e colunas por meio dos rótulos.

- **iloc[]:** Acessa linhas e colunas por meio de índices numéricos inteiros.

- **reset.index():** Redefine os índices de um DataFrame. Após realizar operações de filtro, seleção ou manipulação de dados, os índices podem ser alterados. Esse comando é útil para reverter essas alterações e restaurar um índice padrão de contagem começando do zero.

- **pandas.MultiIndex:** Uma classe que representa índices hierárquicos. Em vez de ter um único nível de rótulos para índices, um MultiIndex permite a criação de índices com múltiplos níveis, proporcionando uma maneira mais flexível de organizar e representar dados tabulares complexos.

- **xs():** Utilizado para extrair um nível de um MultiIndex DataFrame. Ele permite acessar dados em um nível específico de um índice hierárquico, facilitando a manipulação de dados em DataFrames com índices multiníveis.

- **dropna():** Remove linhas ou colunas que contenham valores ausentes (NaN). Útil para limpar dados, eliminando observações ou variáveis que não possuem informações completas.

- **fillna():** Preenche valores ausentes (NaN) em um DataFrame com valores específicos. Também é útil para tratar dados faltantes, fornecendo uma maneira de preenchê-los com valores padrão ou calculados.

- **groupby():** Agrupa dados com base nos critérios especificados e, em seguida, aplica funções de agregação a esses grupos.

- **pandas.concat():** Concatena (junta) objetos como DataFrames ou Séries. Ele oferece uma maneira flexível de combinar dados ao longo de um eixo específico, seja horizontalmente (ao longo das colunas) ou verticalmente (ao longo das linhas).

- **pandas.merge():** Combina (funde) dois DataFrames com base em colunas ou índices específicos. Essa operação é semelhante à junção de tabelas em bancos de dados SQL.

- **pandas.join():** Combina dois DataFrames com base em índices ou colunas específicas. Assim como o método merge(), o join() é uma forma de realizar operações de junção entre DataFrames.

- **pandas.read_csv():** Lê dados de um arquivo CSV (Comma-Separated Values) e criar um DataFrame. O CSV é um formato comum para armazenar dados tabulares, onde as colunas são separadas por vírgulas e as linhas representam registros.

- **pandas.to_csv():** Cria um arquivo CSV a partir de um DataFrame.

Em conclusão, esta tarefa prática proporcionou uma introdução valiosa às bibliotecas NumPy e Pandas, essenciais para a análise de dados em Python. Através do ambiente interativo Jupyter, explorei funcionalidades-chave, desde a criação de arrays multidimensionais no NumPy até a manipulação eficiente de dados em DataFrames no Pandas. As anotações fornecem uma referência concisa, abrangendo métodos essenciais, como manipulação de forma, estatísticas descritivas, geração de dados aleatórios e operações de limpeza e combinação de dados. Este conhecimento é fundamental para qualquer cientista de dados que busca habilidades sólidas em manipulação e análise de dados em Python.