Centro Universitário de Excelência
Sistemas de Informação

Título: *Implementação de Métricas Estatísticas em Python para Modelos de Recomendação*

Autores: João Vítor da Conceição de Almeida
André Almeida Gomes Neto
Deivid Souza Dos Santos Oliveira

Feira de Santana, 2025

Agenda

Objetivo da apresentação:
“Apresentar o desenvolvimento da biblioteca estatística em Python, detalhando as métricas implementadas, a metodologia utilizada, os resultados obtidos e os principais aprendizados.”

1. Contexto e Objetivo
Apresentação do programa da FoodDelivery e a importância da estatística no desenvolvimento do projeto.

2. Métricas Implementadas
Revisão das principais medidas estatísticas aplicadas em Python: tendência central, dispersão, covariância, frequências e probabilidade.

3. Metodologia e Resultados
Estrutura de dados utilizada, processo de implementação, testes comparativos e resultados obtidos.

4. → Principais aprendizados, desafios enfrentados e próximos passos para evolução do projeto.

Contexto e Objetivo

- FoodDelivery – Programa de Trainee 2025.
- Desafio: desenvolver biblioteca estatística em Python puro.
- Objetivos:
- Reforçar conceitos estatísticos.
- Implementar funções sem bibliotecas externas.
- Validar resultados com ferramentas consolidadas.
- Aplicar boas práticas de versionamento com Git.

Métricas Implementadas

- **Tendência Central:** média, mediana, moda.
- **Dispersão:** variância, desvio padrão.
- **Associação:** covariância.
- **Frequências:** absoluta, relativa e acumulada.
- **Probabilidade:** probabilidade condicional.
- **Exploração de Dados:** conjunto de itens únicos (itemset).

Desvio Padrão Populacional

```python
def stdev(self, column):
    """
    Calcula o desvio padrão populacional de uma coluna.

    Fórmula:
    $$ \sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}} $$

    Parâmetros
    ----------
    column : str
        O nome da coluna (chave do dicionário do dataset).

    Retorno
    -------
    float
        O desvio padrão dos valores na coluna.
    """

    stats = self.dataset[column]

    if not stats:
        return 0.0

    sum = 0
    for i in stats:
        sum+=i
    average = sum/len(stats)

    squares = 0
    for k in stats:
        squares+=(k-average) ** 2

    variance = squares/len(stats)

    deviation = variance ** 0.5

    return float(deviation)
```

Frequência Acumulada (Absoluta ou Relativa)

```python
def cumulative_frequency(self, column, frequency_method='absolute'):
    """
    Calcula a frequência acumulada (absoluta ou relativa) de uma coluna.

    A frequência é calculada sobre os itens ordenados.

    Parâmetros
    ----------
    column : str
        O nome da coluna (chave do dicionário do dataset).
    frequency_method : str, opcional
        O método a ser usado: 'absolute' para contagem acumulada ou
        'relative' para proporção acumulada (padrão é 'absolute').

    Retorno
    -------
    dict
        Um dicionário ordenado com os itens como chaves e suas
        frequências acumuladas como valores.
    """
    if column not in self.dataset:
        raise KeyError(f"A coluna '{column}' não existe no dataset.")

    if frequency_method == 'absolute':
        freq = self.absolute_frequency(column)
    elif frequency_method == 'relative':
        freq = self.relative_frequency(column)
    else:
        raise ValueError("O 'frequency_method' deve ser 'absolute' ou 'relative'.")

    sorted_values = sorted(freq.keys())
    cumulative_freq = {}
    cumulative = 0

    for value in sorted_values:
        cumulative += freq[value]
        cumulative_freq[value] = cumulative

    return cumulative_freq
```

Covariância

```python
def covariance(self, column_a, column_b):
    """
    Calcula a covariância entre duas colunas.

    Fórmula:
    $$ \text{cov}(X, Y) = \frac{\sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y)}{N} $$

    Parâmetros
    ----------
    column_a : str
        O nome da primeira coluna (X).
    column_b : str
        O nome da segunda coluna (Y).

    Retorno
    -------
    float
        O valor da covariância entre as duas colunas.
    """
    dados_a = self.dataset[column_a]
    dados_b = self.dataset[column_b]

    if len(dados_a) == 0 or len(dados_b) == 0:
        return 0.0

    media_a = sum(dados_a) / len(dados_a)
    media_b = sum(dados_b) / len(dados_b)

    soma_covar = 0
    for x, y in zip(dados_a, dados_b):
        soma_covar += (x - media_a) * (y - media_b)

    covariancia = soma_covar / len(dados_a)

    return covariancia
```

Metodologia e Resultados

- **Metodologia**
- Estrutura de dados em dicionário.
- Implementação passo a passo em Python puro.
- Testes comparativos com *numpy* e *pandas*.
- Versionamento utilizando Git (workflow centralizado).

- **Resultados**
- Funções retornaram valores consistentes e corretos.
- Maior compreensão da lógica dos cálculos estatísticos.
- Desafios: tratamento da moda em múltiplos valores e validação de tipos.

Considerações Finais

- Reforço da união entre **estatística e programação** como base da ciência de dados.
- Uso de boas práticas de versionamento colaborativo.
- Aprendizados principais:
- Lógica por trás de bibliotecas estatísticas.
- Importância do trabalho em equipe.
- Preparação para etapas futuras (modelos de recomendação e mineração de dados).

Referências

- [1] OLIVEIRA, Francisco Estevam Martins de. **Estatística e Probabilidade - Exercícios Resolvidos e Propostos, 3ª edição** . Rio de Janeiro: LTC, 2017. *E-book.* pág.108. ISBN 9788521633846
- [2] **MENEZES, Nilo Ney Coutinho.** *Introdução à programação com Python*. 4. ed. São Paulo: Novatec, 2024. ISBN 978-85-7522-886-9.