Relatório Técnico: Implementação de Métricas Estatísticas em Python

Curso de Sistemas de Informação – Centro Universitário de Excelência (Unex)

Equipe: César Filipe Gomes da Silva, Rodrigo Souza Guimarães e Valnei Sousa Conceição Filho

1. Introdução

A análise de dados tornou-se essencial para a tomada de decisões estratégicas em empresas de food delivery, que lidam com grande volume de informações sobre clientes, pedidos e operações logísticas. Para transformar esses dados em insights, é necessário compreender conceitos estatísticos fundamentais, como média, mediana, variância, desvio padrão e probabilidade.

O objetivo deste projeto é implementar uma biblioteca própria de métricas estatísticas em Python, utilizando apenas recursos nativos da linguagem, sem depender de bibliotecas externas como numpy, pandas ou statistics. Essa abordagem permite reforçar o entendimento de estatística e programação de forma integrada, preparando os trainees para manipular e interpretar dados de maneira eficiente.

2. Fundamentação Teórica:

2.1 Média (Inglês: Mean)

- **Definição:** Representa o valor central de um conjunto de dados, calculado pela soma dos valores dividida pelo número de observações.
- **Fórmula:**

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- **Exemplo em Python:**

```
def mean(column):
    return sum(column) / len(column)
```

- **Comportamento esperado:** A média é sensível a outliers; valores extremos podem distorcer a representação do centro dos dados.

2.2 Mediana (Inglês: Median)

- **Definição:** Valor central de um conjunto de dados ordenados.
- **Fórmula:**

$$\text{mediana} = \begin{cases} x_{(n+1)/2}, & \text{se n ímpar} \\ \frac{x_{n/2} + x_{n/2+1}}{2}, & \text{se n par} \end{cases}$$

- **Exemplo em Python:**

```
def median(column):
    sorted_col = sorted(column)
    n = len(sorted_col)
    mid = n // 2
    if n % 2 == 0:
        return (sorted_col[mid - 1] + sorted_col[mid]) / 2
    return sorted_col[mid]
```

- **Comportamento esperado:** Mais robusta que a média, menos afetada por outliers.

2.3 Moda (Mode)

- **Definição:** Valor que ocorre com maior frequência.
- **Exemplo em Python:**

```
def mode(column):
    freq = {}
    for item in column:
        freq[item] = freq.get(item, 0) + 1
    max_freq = max(freq.values())
    return [key for key, value in freq.items() if value == max_freq]
```

- **Comportamento esperado:** Identifica padrões repetitivos nos dados; pode haver mais de uma moda.

2.4 Variância e Desvio Padrão (Inglês: Variance & Standard Deviation)

- **Definição:** Medem a dispersão dos dados em relação à média.
- **Fórmula da Variância:**

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

- **Desvio padrão:**

$$\sigma = \sqrt{\sigma^2}$$

- **Exemplo em Python:**

```
def variance(column):
    m = mean(column)
    return sum((x - m) ** 2 for x in column) / len(column)
def std_deviation(column):
    return variance(column) ** 0.5
```

Comportamento esperado: Quanto maior o valor, maior a dispersão dos dados.

2.5 Frequências e Probabilidade

Frequência absoluta: Número de ocorrências de um valor.

Frequência relativa: Razão entre frequência absoluta e total de observações.

Frequência acumulada: Soma cumulativa das frequências relativas.

Exemplo em Python:

```
def frequencies(column):
    abs_freq = {}
    for item in column:
        abs_freq[item] = abs_freq.get(item, 0) + 1
    total = len(column)
    rel_freq = {k: v/total for k, v in abs_freq.items()}
    cum_freq = {}
```

```
    cum_sum = 0
    for k in sorted(rel_freq.keys()):
        cum_sum += rel_freq[k]
        cum_freq[k] = cum_sum
    return abs_freq, rel_freq, cum_freq
```

- **Comportamento esperado:** Permite análise de padrões e cálculo de probabilidades.

2.6 Pureza e Ganho de Informação

Pureza: Mede a homogeneidade de uma partição de dados em Árvores de Decisão.

Ganho de informação: Redução da incerteza ao dividir dados por um atributo.

Fórmulas:

$$Ganho = Entropia(Pai) - \sum_i \frac{|Filho_i|}{|Pai|} Entropia(Filho_i)$$

Comportamento esperado: A divisão com maior ganho aumenta a homogeneidade das partições.

3. Metodologia

3.1 Validação do Dataset

- Garantir que o dataset é um dicionário.
- Verificar se todas as colunas têm o mesmo número de elementos.
- Garantir que os elementos de cada coluna são do mesmo tipo.

3.2 Estrutura da Classe Statistics

A classe contém métodos para cada métrica. Recebe como entrada uma coluna do dataset (lista de valores). Retorna o valor da métrica calculada.

Exemplo de uso:

```
dataset = {"idade": [23, 25, 30, 22, 25]}

stats = Statistics(dataset)

print(stats.mean("idade"))
```

3.3 Restrições

- Não utilizar bibliotecas externas.
- Operações feitas com listas, dicionários, loops e funções nativas do Python

4. Resultados e Discussões

Aplicando as métricas em no dataset de exemplo:

Idade	Frequência
22	1
23	1
25	2
30	1

Média: 25.0; **Mediana:** 25 **Moda:** 25; **Variância:** 8.8; **Desvio Padrão:** 2.97

Comparando com bibliotecas externas os resultados foram consistentes, consolidando as implementações.

A média do exemplo é sensível a outliers, a mediana é robusta. O cálculo de frequências permitiu identificar padrões na distribuição dos dados. Todas as restrições foram respeitadas.

5. Considerações Finais

Desafios: Validação do dataset, cálculo manual de variância e desvio padrão, lidar com listas vazias ou colunas com valores repetidos.

Anotações: Integração entre estatística e programação, reforço do entendimento de métricas sem depender de bibliotecas externas.

Melhorias futuras: Implementação de mais métricas (quartis, coeficiente de variação), otimização dos métodos e criação de uma interface para manipulação direta do dataset.

6. Referências

REUND, J.; PERLIS, D. **Estatística e Probabilidade**. Rio de Janeiro: LTC, 2018.

MANN, P. **Estatística Básica**. 5. ed. São Paulo: Pearson, 2020.

PYTHON SOFTWARE FOUNDATION. **Python Documentation**. Disponível em: https://docs.python.org/3/. Acesso em: 23 ago. 2025.