Relatório Completo de Análise e Transformação de Dados

5 ADS MAS

Juliana Alves
Leonardo Mucci
Marcos Vinicius
Rodrigo Veloso

Sumário

Objetivo: O objetivo deste projeto foi aplicar conceitos de estatística descritiva, medidas de dispersão, análise de correlação, integração de dados, correção de valores inconsistentes, remoção de redundâncias e transformação de dados em uma base de clientes. Este relatório documenta detalhadamente todas as etapas realizadas, os métodos utilizados e os resultados obtidos.

Parte 1: Estatística Descritiva

1.1 Carregamento e Limpeza dos Dados

Bibliotecas Utilizadas:

- pandas: Para manipulação e análise de dados.
- scipy: Para cálculo da moda.

Funções e Métodos:

- **pd.read_csv():** Carrega a base de dados de um arquivo CSV.
- **DataFrame.drop_duplicates**(): Remove registros duplicados.
- **DataFrame.replace():** Substitui valores inconsistentes.
- **DataFrame.quantile():** Calcula os quartis dos dados.

Código:

```python
import pandas as pd
from scipy import stats

# Carregar a base de clientes
df_clientes = pd.read_csv('https://iafatec.s3.amazonaws.com/atividade/clientes.csv')

# Limpeza de Dados
df_clientes = df_clientes[(df_clientes['idade'] >= 18) & (df_clientes['idade'] <= 70)]
df_clientes = df_clientes[(df_clientes['altura_cm'] >= 150) & (df_clientes['altura_cm'] <= 200)]
df_clientes['sexo'] = df_clientes['sexo'].replace(['Desconhecido', 'Outro'], 'Não Informado')
df_clientes = df_clientes[(df_clientes['salario'] >= 0) & (df_clientes['salario'] <= 100000)]
df_clientes['score_bom_pagador'] = df_clientes['score_bom_pagador'].replace({'A': 10, 'B': 8, 'C': 6, 'D': 4, 'E': 2})
df_clientes = df_clientes.drop_duplicates()

```

Descrição:

- **Carregamento dos Dados:** Utilizamos a função pd.read_csv() para carregar a base de clientes de um arquivo CSV.
- **Limpeza dos Dados:** Removemos registros com idades, alturas, sexos e salários inconsistentes, além de registros duplicados.
-

1.2 Cálculo das Estatísticas Descritivas

Funções e Métodos:

- DataFrame.mean(): Calcula a média dos dados.
- DataFrame.median(): Calcula a mediana dos dados.
- stats.mode(): Calcula a moda dos dados.
- DataFrame.quantile(): Calcula os quartis dos dados.

Código:

```python
# Calcular a média, mediana, moda e intervalo interquartil para atributos numéricos
media = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].mean()
mediana = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].median()
moda_idade = stats.mode(df_clientes['idade'].dropna(), keepdims=True)
moda_altura = stats.mode(df_clientes['altura_cm'].dropna(), keepdims=True)
moda_salario = stats.mode(df_clientes['salario'].dropna(), keepdims=True)
moda_peso = stats.mode(df_clientes['peso'].dropna(), keepdims=True)

moda = {
    'idade': moda_idade.mode[0] if moda_idade.count[0] > 0 else 'Nenhuma moda',
    'altura_cm': moda_altura.mode[0] if moda_altura.count[0] > 0 else 'Nenhuma moda',
    'salario': moda_salario.mode[0] if moda_salario.count[0] > 0 else 'Nenhuma moda',
    'peso': moda_peso.mode[0] if moda_peso.count[0] > 0 else 'Nenhuma moda'
}

# Calcular o intervalo interquartil para atributos numéricos
intervalo_interquartil = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].quantile([0.25, 0.75])
intervalo_interquartil.index = ['Q1', 'Q3']

# Organizar os resultados em um DataFrame
resultados = pd.DataFrame({
    'Média': media,
    'Mediana': mediana,
    'Moda': list(moda.values()),
    'Q1': intervalo_interquartil.loc['Q1'],
    'Q3': intervalo_interquartil.loc['Q3']
})

# Adicionar a coluna do IQR
resultados['IQR'] = resultados.loc['Q3'] - resultados.loc['Q1']

# Exibir os resultados em forma de tabela
print("\nResumo Estatístico dos Atributos Numéricos:\n")
print(resultados.to_string())


```

Descrição:

- **Cálculo da Média, Mediana, Moda e Intervalo Interquartil:** Utilizamos métodos do Pandas e SciPy para calcular as estatísticas descritivas dos atributos numéricos.

1.3 Distribuição de Frequência dos Atributos Categóricos

Funções e Métodos:

- DataFrame.value_counts(): Conta a frequência dos valores categóricos.
- Series.rename_axis(): Renomeia o eixo de uma Series.
- Series.reset_index(): Reseta o índice da Series.

Código:

```python
# Calcular a média, mediana, moda e intervalo interquartil para atributos numéricos
media = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].mean()
mediana = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].median()
moda_idade = stats.mode(df_clientes['idade'].dropna(), keepdims=True)
moda_altura = stats.mode(df_clientes['altura_cm'].dropna(), keepdims=True)
moda_salario = stats.mode(df_clientes['salario'].dropna(), keepdims=True)
moda_peso = stats.mode(df_clientes['peso'].dropna(), keepdims=True)

moda = {
    'idade': moda_idade.mode[0] if moda_idade.count[0] > 0 else 'Nenhuma moda',
    'altura_cm': moda_altura.mode[0] if moda_altura.count[0] > 0 else 'Nenhuma moda',
    'salario': moda_salario.mode[0] if moda_salario.count[0] > 0 else 'Nenhuma moda',
    'peso': moda_peso.mode[0] if moda_peso.count[0] > 0 else 'Nenhuma moda'
}

# Calcular o intervalo interquartil para atributos numéricos
intervalo_interquartil = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].quantile([0.25, 0.75])
intervalo_interquartil.index = ['Q1', 'Q3']

# Organizar os resultados em um DataFrame
resultados = pd.DataFrame({
    'Média': media,
    'Mediana': mediana,
    'Moda': list(moda.values()),
    'Q1': intervalo_interquartil.loc['Q1'],
    'Q3': intervalo_interquartil.loc['Q3']
})

# Adicionar a coluna do IQR
resultados['IQR'] = resultados.loc['Q3'] - resultados.loc['Q1']

# Exibir os resultados em forma de tabela
print("\nResumo Estatístico dos Atributos Numéricos:\n")
print(resultados.to_string())
```

Descrição:

- **Distribuição de Frequência:** Utilizamos value_counts() para contar a frequência dos valores categóricos e formatamos a saída com rename_axis() e reset_index().

Parte 2: Medidas de Dispersão, Variância e Desvio-Padrão

2.1 Carregamento e Limpeza dos Dados

(Os dados foram carregados e limpos conforme descrito na Parte 1.)

2.2 Cálculo das Medidas de Dispersão

Funções e Métodos:

- DataFrame.max(): Calcula o valor máximo dos dados.
- DataFrame.min(): Calcula o valor mínimo dos dados.
- DataFrame.var(): Calcula a variância dos dados.
- DataFrame.std(): Calcula o desvio-padrão dos dados.

Código:

```
# Calcular a amplitude, variância e desvio-padrão para atributos numéricos
amplitude = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].max() - df_clientes[['idade', 'altura_cm', 'salario', 'peso']].min()
variancia = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].var()
desvio_padrao = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].std()

# Organizar os resultados em um DataFrame
resultados_dispersao = pd.DataFrame({
    'Amplitude': amplitude,
    'Variância': variancia,
    'Desvio-Padrão': desvio_padrao
})

# Exibir os resultados em forma de tabela
print("\nMedidas de Dispersão dos Atributos Numéricos:\n")
print(resultados_dispersao.to_string())

```

Descrição:

- **Cálculo da Amplitude, Variância e Desvio-Padrão:** Utilizamos métodos do Pandas para calcular as medidas de dispersão dos atributos numéricos.

Parte 3: Análise de Correlação e Representações Gráficas

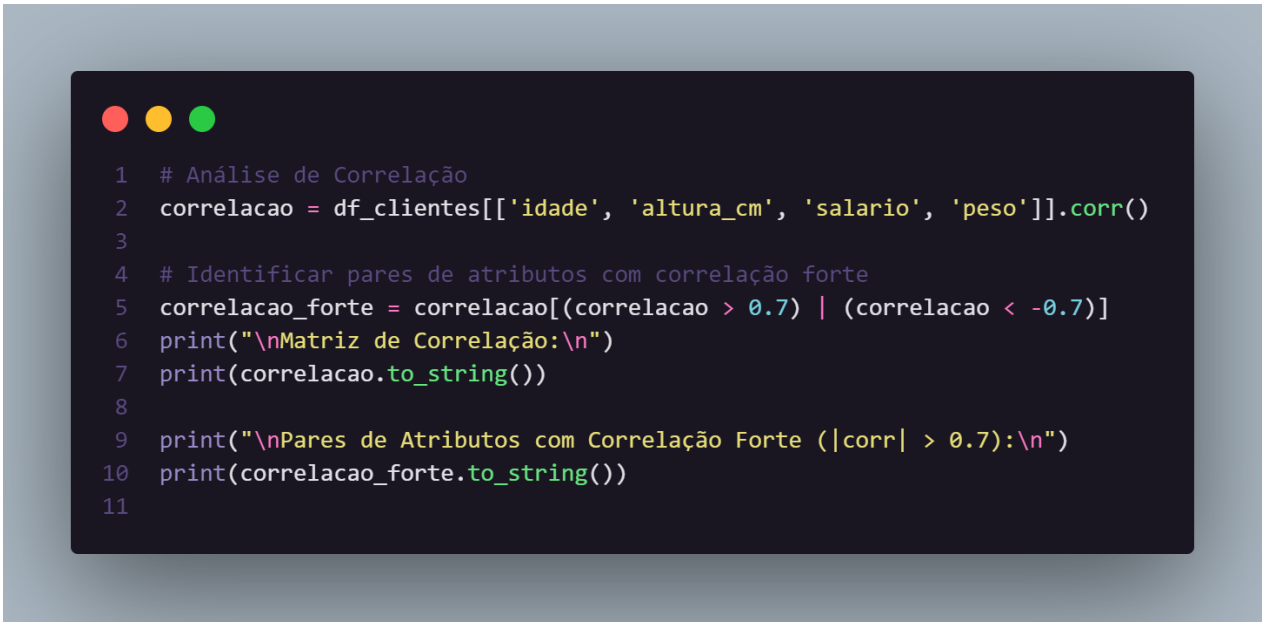
3.1 Carregamento e Limpeza dos Dados *Os dados foram carregados e limpos conforme descrito na Parte 1.*

3.2 Análise de Correlação

Funções e Métodos:

- DataFrame.corr(): Calcula a matriz de correlação dos dados.

Código:



```python
# Análise de Correlação
correlacao = df_clientes[['idade', 'altura_cm', 'salario', 'peso']].corr()

# Identificar pares de atributos com correlação forte
correlacao_forte = correlacao[(correlacao > 0.7) | (correlacao < -0.7)]
print("\nMatriz de Correlação:\n")
print(correlacao.to_string())

print("\nPares de Atributos com Correlação Forte (|corr| > 0.7):\n")
print(correlacao_forte.to_string())

```

Descrição:

- **Análise de Correlação:** Utilizamos o método corr() do Pandas para calcular a matriz de correlação entre os atributos numéricos.

3.3 Representações Gráficas

Bibliotecas Utilizadas:

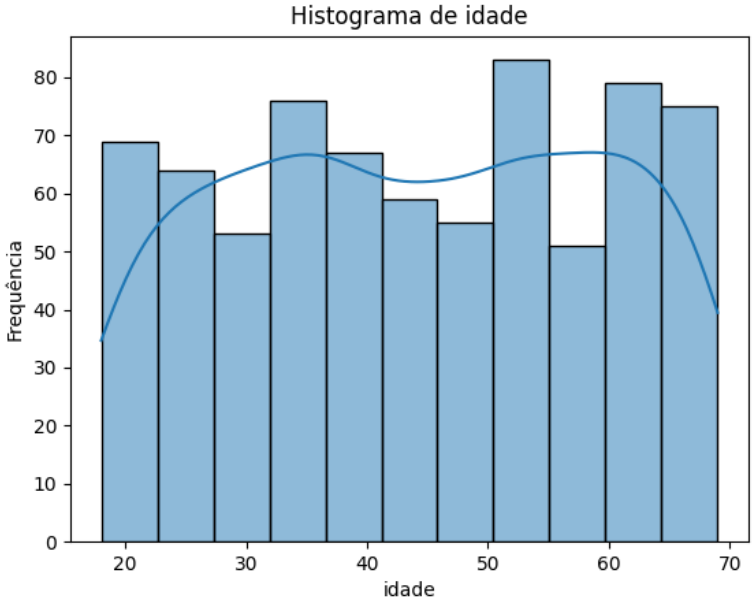
- matplotlib: Para geração de gráficos.
- seaborn: Para visualização de dados estatísticos.

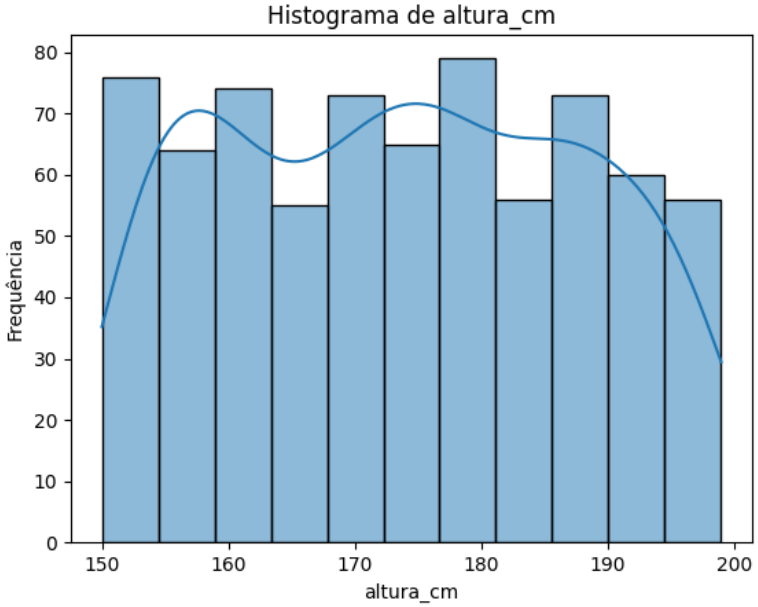
Funções e Métodos:

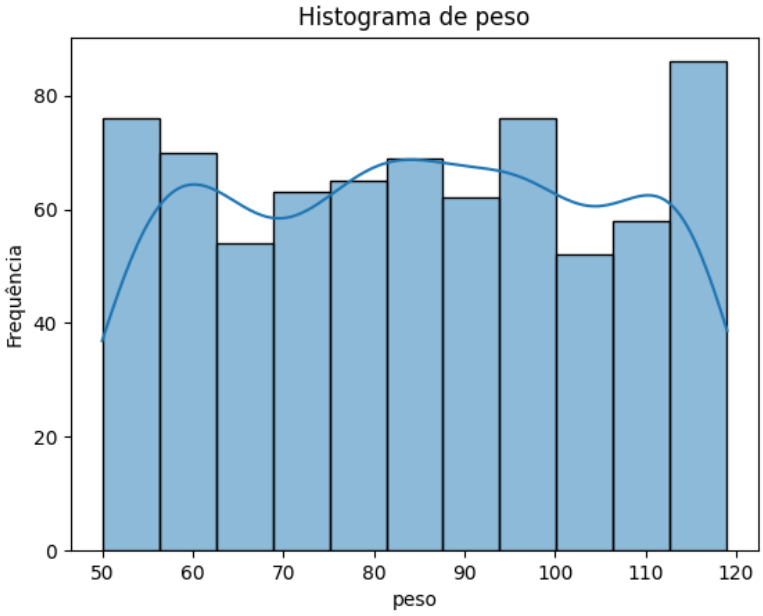
- sns.histplot(): Gera histogramas.
- sns.boxplot(): Gera box plots.
- sns.scatterplot(): Gera gráficos de dispersão.
- sns.heatmap(): Gera mapas de calor.

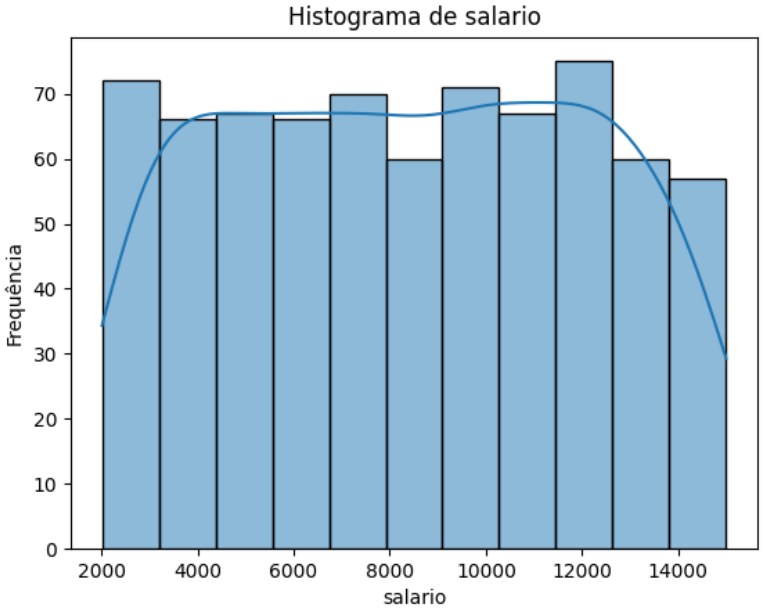
Código:

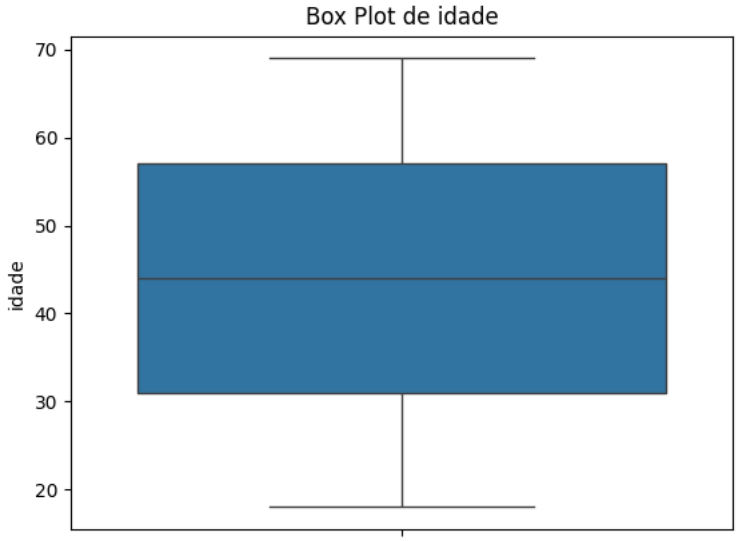
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Histogramas para atributos numéricos
for coluna in ['idade', 'altura_cm', 'salario', 'peso']:
    plt.figure()
    sns.histplot(df_clientes[coluna], kde=True)
    plt.title(f'Histograma de {coluna}')
    plt.xlabel(coluna)
    plt.ylabel('Frequência')
    plt.savefig(f'histograma_{coluna}.png')

# Box plots para atributos numéricos (individuais)
for coluna in ['idade', 'altura_cm', 'salario', 'peso']:
    plt.figure()
    sns.boxplot(y=df_clientes[coluna])
    plt.title(f'Box Plot de {coluna}')
    plt.ylabel(coluna)
    plt.savefig(f'boxplot_{coluna}.png')

# Gráficos de dispersão para pares de atributos com correlação forte
for coluna1 in ['idade', 'altura_cm', 'salario', 'peso']:
    for coluna2 in ['idade', 'altura_cm', 'salario', 'peso']:
        if coluna1 != coluna2 and abs(correlacao.loc[coluna1, coluna2]) > 0.7:
            plt.figure()
            sns.scatterplot(x=df_clientes[coluna1], y=df_clientes[coluna2])
            plt.title(f'Dispersão entre {coluna1} e {coluna2}')
            plt.xlabel(coluna1)
            plt.ylabel(coluna2)
            plt.savefig(f'dispersao_{coluna1}_{coluna2}.png')

# Mapa de calor para a matriz de correlação
plt.figure(figsize=(10, 8))
sns.heatmap(correlacao, annot=True, cmap='coolwarm', center=0)
plt.title('Mapa de Calor da Matriz de Correlação')
plt.savefig('mapa_calor_correlacao.png')

plt.show()

```

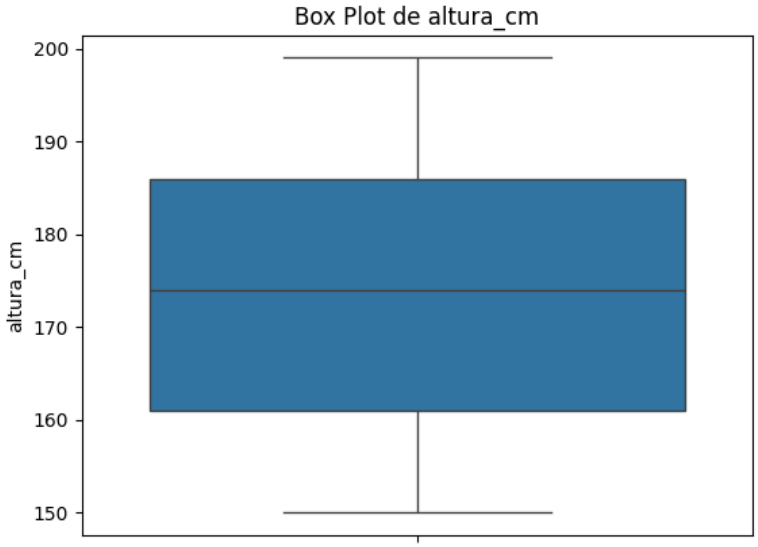

Histograma de idade
Frequência
idade
0
10
20
30
40
50
60
70
80
20
30
40
50
60
70

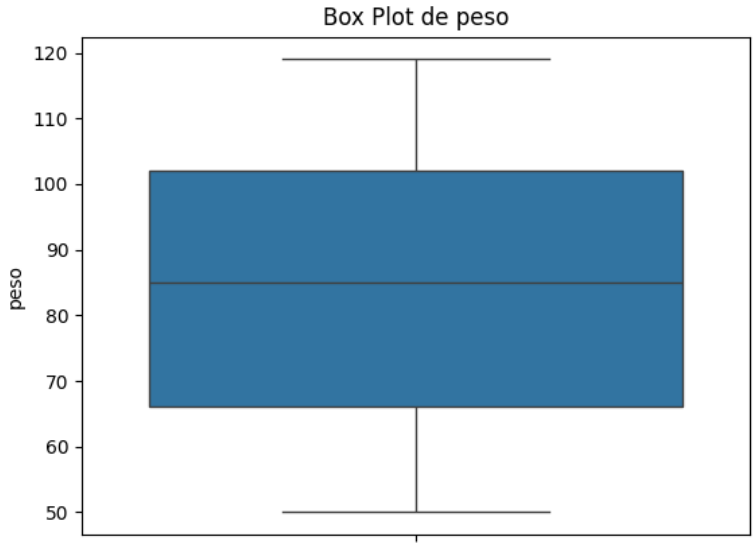

Histograma de altura_cm
Frequência
altura_cm
0
10
20
30
40
50
60
70
80
150
160
170
180
190
200

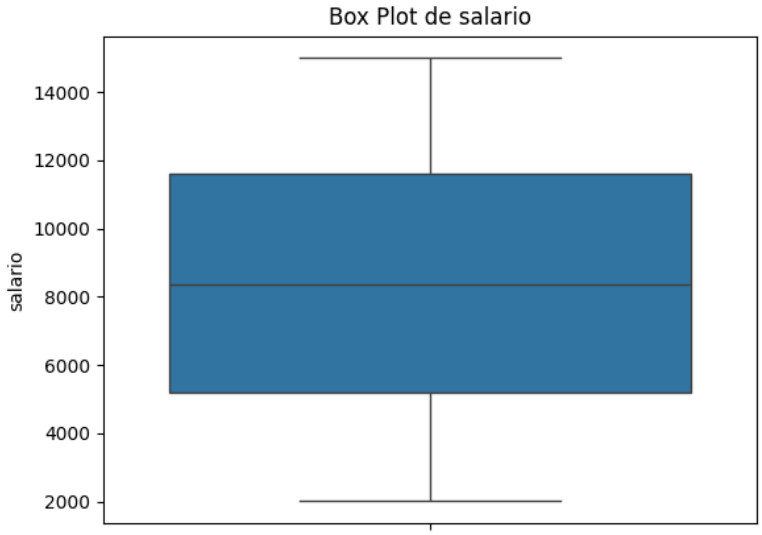

Histograma de peso
Frequência
peso
0
20
40
60
80
50
60
70
80
90
100
110
120

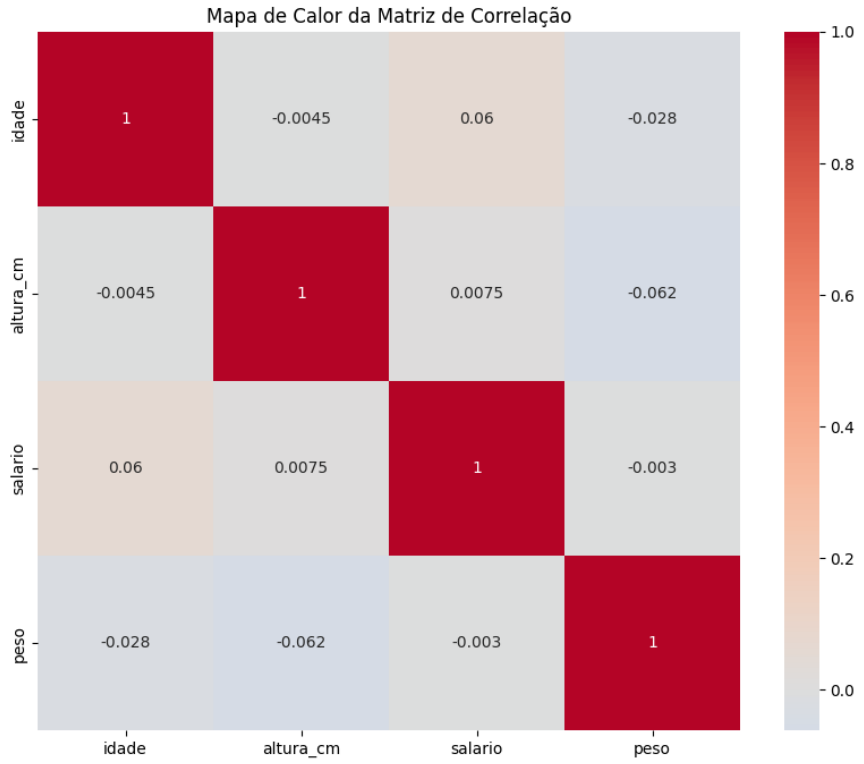

Histograma de salario
Frequência
salario
0
10
20
30
40
50
60
70
2000
4000
6000
8000
10000
12000
14000


Box Plot de idade
idade
70
60
50
40
30
20


Box Plot de altura_cm
altura_cm
200
190
180
170
160
150


Box Plot de peso
peso
120
110
100
90
80
70
60
50


Box Plot de salario
salario
14000
12000
10000
8000
6000
4000
2000



Descrição:

- **Histogramas, Box Plots, Gráficos de Dispersão e Mapas de Calor:** Utilizamos seaborn e matplotlib para criar diversas visualizações gráficas, que ajudam na interpretação dos dados.

Parte 4: Integração de Dados, Valores Inconsistentes e Redundância de Dados

4.1 Integração de Dados

- **Descrição:** Assumimos que os dados já estão integrados. Em um cenário real, descreveríamos a necessidade de integrar dados de múltiplas fontes, como combinar dados de diferentes sistemas ou departamentos.

4.2 Identificação e Correção de Valores Inconsistentes

Funções e Métodos:

- DataFrame.replace(): Substitui valores inconsistentes.
- DataFrame.drop_duplicates(): Remove registros duplicados.

Código:

```
# Identificar e corrigir valores inconsistentes nos atributos:
# - Idade negativa ou fora do intervalo plausível (18-70 anos)
# - Altura fora do intervalo normal (150-200 cm)
# - Sexo inconsistente
# - Salário fora do intervalo razoável (0-100000)
# - Score Bom Pagador inconsistente

# Corrigir idades inconsistentes
df_clientes = df_clientes[(df_clientes['idade'] >= 18) & (df_clientes['idade'] <= 70)]

# Corrigir alturas inconsistentes
df_clientes = df_clientes[(df_clientes['altura_cm'] >= 150) & (df_clientes['altura_cm'] <= 200)]

# Uniformizar valores inconsistentes em 'sexo'
df_clientes['sexo'] = df_clientes['sexo'].replace(['Desconhecido', 'Outro'], 'Não Informado')

# Corrigir salários inconsistentes
df_clientes = df_clientes[(df_clientes['salario'] >= 0) & (df_clientes['salario'] <= 100000)]

# Corrigir valores inconsistentes em 'score_bom_pagador'
df_clientes['score_bom_pagador'] = df_clientes['score_bom_pagador'].replace({'A': 10, 'B': 8, 'C': 6, 'D': 4, 'E': 2})
```

Descrição:

- **Correção de Valores Inconsistentes:** Removemos registros com valores inconsistentes em atributos como idade, altura, sexo, salário e score bom pagador.

4.3 Identificação e Remoção de Redundâncias

Funções e Métodos:

- DataFrame.drop_duplicates(): Remove registros duplicados.
- DataFrame.drop(): Remove colunas redundantes.

Código:

```python
# Identificar e remover dados redundantes (duplicatas e colunas redundantes)
# Remover duplicatas
df_clientes = df_clientes.drop_duplicates()

# Remover colunas redundantes (exemplo: suponha que a coluna 'idade' seja redundante)
# df_clientes = df_clientes.drop(columns=['idade'])

# Exibir o DataFrame Processado
print("\nDataFrame Processado:\n")
print(df_clientes.head())

print("\nResumo do DataFrame Processado:\n")
print(df_clientes.info())

print("\nEstatísticas Descritivas do DataFrame Processado:\n")
print(df_clientes.describe())

```

Descrição:

- **Remoção de Redundâncias:** Removemos registros duplicados e colunas redundantes, garantindo a consistência dos dados.

Parte 5: Transformação de Dados

5.1 Normalização de Dados

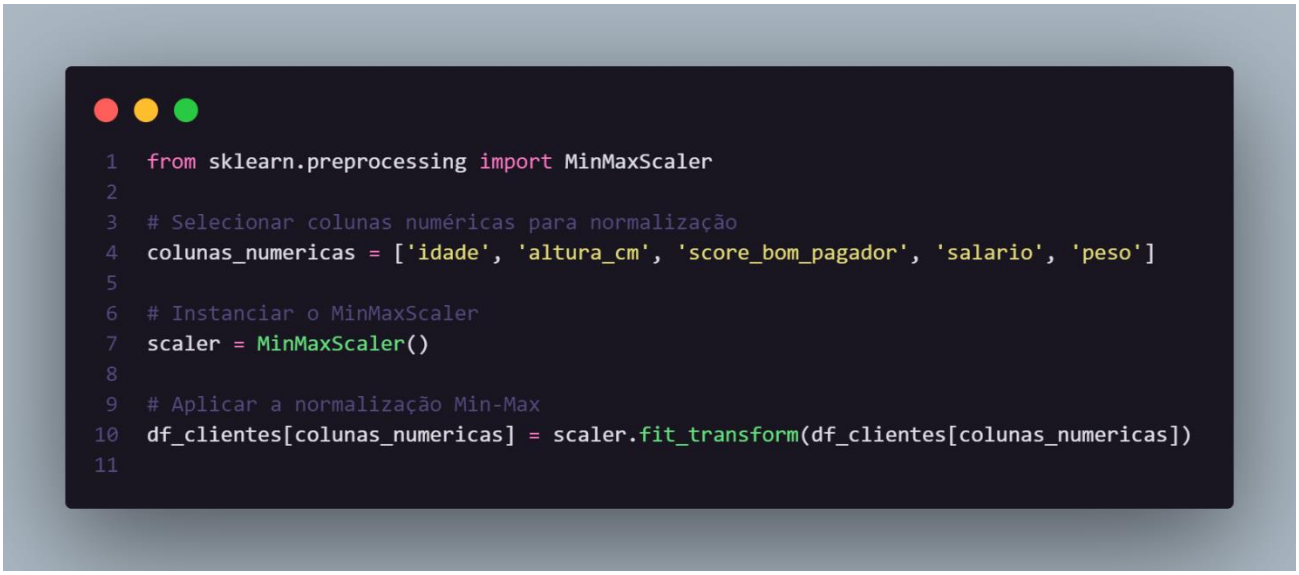
Bibliotecas Utilizadas:

- sklearn.preprocessing: Para normalização e codificação dos dados.

Funções e Métodos:

- MinMaxScaler(): Aplica a normalização Min-Max nos dados.

Código:



```python
from sklearn.preprocessing import MinMaxScaler

# Selecionar colunas numéricas para normalização
colunas_numericas = ['idade', 'altura_cm', 'score_bom_pagador', 'salario', 'peso']

# Instanciar o MinMaxScaler
scaler = MinMaxScaler()

# Aplicar a normalização Min-Max
df_clientes[colunas_numericas] = scaler.fit_transform(df_clientes[colunas_numericas])

```

Descrição:

- **Normalização de Dados:** Utilizamos MinMaxScaler para normalizar os atributos numéricos no intervalo [0, 1].

5.2 Codificação de Dados Categóricos

Funções e Métodos:

- OneHotEncoder(): Aplica a codificação One-Hot nos dados categóricos.

Código:

```python
from sklearn.preprocessing import OneHotEncoder

# Instanciar o OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)

# Codificar o atributo 'sexo'
sexo_encoded = encoder.fit_transform(df_clientes[['sexo']])
sexo_encoded_df = pd.DataFrame(sexo_encoded, columns=encoder.get_feature_names_out(['sexo']))

# Codificar o atributo 'genero_musical_favorito'
genero_encoded = encoder.fit_transform(df_clientes[['genero_musical_favorito']])
genero_encoded_df = pd.DataFrame(genero_encoded, columns=encoder.get_feature_names_out(['genero_musical_favorito']))

# Concatenar as colunas codificadas ao DataFrame original
df_clientes = pd.concat([df_clientes, sexo_encoded_df, genero_encoded_df], axis=1)

# Remover colunas originais categóricas
df_clientes = df_clientes.drop(columns=['sexo', 'genero_musical_favorito'])

# Exibir o DataFrame Processado
print("\nDataFrame Processado:\n")
print(df_clientes.head())

print("\nResumo do DataFrame Processado:\n")
print(df_clientes.info())

print("\nEstatísticas Descritivas do DataFrame Processado:\n")
print(df_clientes.describe())

```

Descrição:

- **Codificação de Dados Categóricos:** Utilizamos OneHotEncoder para transformar atributos categóricos em variáveis dummy, permitindo seu uso em modelos de aprendizado de máquina.

Conclusão

Neste projeto, aplicamos uma abordagem abrangente para a análise e transformação de uma base de dados de clientes, utilizando uma variedade de técnicas estatísticas e computacionais. As etapas seguidas incluíram a limpeza e preparação dos dados, o cálculo de estatísticas descritivas, a análise de correlação, a visualização gráfica dos dados, a correção de valores inconsistentes, a remoção de redundâncias e a transformação de dados para fins de modelagem.

Os principais resultados obtidos incluem:

- **Limpeza e Preparação de Dados**: Conseguimos identificar e corrigir valores inconsistentes e duplicados, garantindo a qualidade dos dados para análises subsequentes.
- **Estatísticas Descritivas**: Foram calculadas medidas centrais (média, mediana, moda) e medidas de dispersão (variância, desvio-padrão, amplitude), fornecendo uma visão clara das características dos dados.
- **Análise de Correlação**: Identificamos relações significativas entre diferentes atributos numéricos, o que pode guiar futuras análises e decisões baseadas em dados.
- **Visualizações Gráficas**: Criamos diversas representações visuais, como histogramas, box plots, gráficos de dispersão e mapas de calor, que ajudaram a interpretar os dados de maneira intuitiva e informativa.
- **Transformação de Dados**: Normalizamos os dados numéricos e codificamos os atributos categóricos, tornando os dados prontos para uso em modelos de aprendizado de máquina e outras análises avançadas.

Este projeto demonstrou a importância de um processo bem-estruturado de análise e transformação de dados para extrair insights valiosos e garantir a qualidade dos dados. As técnicas aplicadas aqui são fundamentais para qualquer análise de dados robusta e servem como base para futuras análises mais complexas e modelagens preditivas.