Nome: João Geiger Piza
Repositório no Github: https://github.com/joaogpizza/Lipai
Códigos das Videoaulas:
aula01.py:

```
import pandas as pd

# Parte 1

data =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019.csv')
print(data)

print(data.head())

print(data.head(10))

print(data.info())

print(type(data))

print(data.shape)

print(f'O DataFrame possui {data.shape[0]} linhas/observações/registros
e {data.shape[1]} colunas/atributos/variáveis.')

personagens_df = pd.DataFrame({
    'nome': ['Luke Skywalker', 'Yoda', 'Palpatine'],
    'idade': [16, 1000, 70],
    'peso': [70.5, 15.2, 60.1],
    'eh jedi': [True, True, False]  # o nome das colunas podem ter
espaços
})

print(personagens_df)

print(personagens_df.info())

print(personagens_df.columns)

print(type(personagens_df.columns))

print(list(personagens_df.columns))
```

```
personagens_df_renomeado = personagens_df.rename(columns={
    'nome': 'Nome Completo',  # renomeia a coluna de nome 'nome' para
'Nome Completo'
    'idade': 'Idade'
})

print(personagens_df_renomeado)

personagens_df.rename(columns={'nome': 'Nome Completo','idade':
'Idade'}, inplace=True)
print(personagens_df)

personagens_df.columns = ['NOME', 'IDADE', 'PESO', 'EH_JEDI']
print(personagens_df)

# selecionando uma coluna inteira
print(data['ESTADO'])

# selecionando uma coluna inteira
# esta forma de acesso, só funciona para colunas com nomes sem espaços,
acentos, etc (caracteres inválidos)
print(data.ESTADO)

print(type(data['ESTADO']))

print(data.iloc[1])

print(type(data.iloc[1]))

print(pd.Series([5.5, 6.0, 9.5]))

print(pd.Series([5.5, 6.0, 9.5], index=['prova 1', 'prova 2',
'projeto'], name='Notas dos Luke Skywalker'))

produto_view = data['PRODUTO']  # a series retornada refente à coluna,
NÃO É UMA CÓPIA, mas sim, uma REFERÊNCIA/VIEW à coluna do dataframe
print(produto_view)

produto_copy_bkp = data['PRODUTO'].copy()  # retorna uma cópia da
coluna 'PRODUTO'
print(produto_copy_bkp)
```

```
data['PRODUTO'] = 'Combustível'  # atribuindo o valor constante
'Combustível' para linha do dataframe na coluna 'PRODUTO'
print(data.head())
print(produto_view)
print(produto_copy_bkp)

nrows, ncols = data.shape
print(nrows, ncols)

novos_produtos = [f'Produto {i}' for i in range(nrows)]
print(len(novos_produtos))

# a quantidade de elementos da lista `novos_produtos` é igual ao número
de linhas do dataframe
data['PRODUTO'] = novos_produtos
print(data)

print(produto_view)
print('\n')
print(produto_copy_bkp)

# voltando para os produtos originais
data['PRODUTO'] = produto_copy_bkp  # produto_copy_bkp é uma Series
print(data)

# criando uma coluna a partir de um valor constante/default
# todas as linhas terão o mesmo valor para esta nova coluna
data['coluna sem nocao'] = 'DEFAULT'
print(data)

data['coluna a partir de lista'] = range(data.shape[0])
print(data)

# não funciona pq a quantidade de elementos da lista (a serem
atribuídos a nova coluna) é diferente
# da quantidade de linhas do dataframe
# data['nao funciona'] = [1, 2, 3]

data['PREÇO MÉDIO REVENDA (dólares)'] = data['PREÇO MÉDIO REVENDA'] *
6.0
print(data)
```

```
print(data.index)

pesquisa_de_satisfacao = pd.DataFrame({
    'bom': [50, 21, 100],
    'ruim': [131, 2, 30],
    'pessimo': [30, 20, 1]
}, index=['XboxOne', 'Playstation4', 'Switch'])

print(pesquisa_de_satisfacao.head())
print(pesquisa_de_satisfacao.index)

# selecionando as linhas de índice de 0 a 5 (incluso)
print(data.iloc[:6])
print(data.iloc[10:16])

# selecionando as linhas/observações de índice 1, 5, 10, 15
print(data.iloc[[1, 5, 10, 15]])

# selecionando as linhas/observações de índices 5, 1, 15, 10
print(data.iloc[[5, 1, 15, 10]])

# retornar o valor da linha de índice 1, coluna 4 ('ESTADO')
print(data.iloc[1, 4])

# retorna a linha cujo o rótulo do índice é 'XboxOne'
print(pesquisa_de_satisfacao.loc['XboxOne'])

# NÃO FUNCIONA ===> iloc tentando acessar índices com rótulos
# print(pesquisa_de_satisfacao.iloc['XboxOne'])

# NÃO FUNCIONA ===> loc tentando acessar índices rotulados com números
# print(pesquisa_de_satisfacao.loc[0])

# retorna o valor da linha 'Playstation4', coluna 'ruim'
print(pesquisa_de_satisfacao.loc['Playstation4', 'ruim'])

# selecionando as linhas de índices rotulados 'XboxOne', 'Switch'
print(pesquisa_de_satisfacao.loc[['XboxOne', 'Switch']])

# retorna todas as linhas e apenas as colunas com rótulos 'bom' e
```

```
'pessimo'
print(pesquisa_de_satisfacao[['bom', 'pessimo']])

# retorna todas as linhas e apenas as colunas com rótulos 'bom' e
'pessimo'
print(pesquisa_de_satisfacao.loc[:, ['bom', 'pessimo']])

# selecionando múltiplas colunas: 'PRODUTO', 'ESTADO', 'REGIÃO'
print(data[['PRODUTO', 'ESTADO', 'REGIÃO']])

# deleta/remove in-place (ou seja, no próprio dataframe) a coluna de
rótulo 'Unnamed: 0'
# del data['Unnamed: 0']
print(data)

del data['coluna sem nocao']
del data['coluna a partir de lista']
del data['PREÇO MÉDIO REVENDA (dólares)']

print(data.head())

data.to_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019.csv',
index=False)

data =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019.csv')
print(data.head())

# Mostra todos os estados cujos os preços dos combustíveis foram
aferidos
# Mais tecnicamente, mostra os valores únicos presentes para o
atributo/coluna 'ESTADO'.
print(data['ESTADO'].unique())

# faz uma comparação elemento a elemento da series, retornando uma
Series de booleans
print(data['ESTADO'] == 'SAO PAULO')

# salvando essa Series de booleans em uma variável
selecao = data['ESTADO'] == 'SAO PAULO'
print(selecao)
print(type(selecao))
```

```
print(selecao.shape)
print(data.shape)

print(data[selecao])
print(data.loc[selecao])

print(data.query('ESTADO == "SAO PAULO"'))

postos_sp = data.query('ESTADO == "SAO PAULO"')
print(postos_sp)

print(type(postos_sp))
print(postos_sp.shape)
print(postos_sp)

print(postos_sp.reset_index())

print(postos_sp.reset_index(drop=True))

postos_sp.reset_index(drop=True, inplace=True)
print(postos_sp)

postos_sp = data.query('ESTADO == "SAO PAULO"').reset_index(drop=True)
print(postos_sp)

selecao = (data['ESTADO'] == 'RIO DE JANEIRO') & (data['PREÇO MÉDIO
REVENDA'] > 2.0)
print(selecao)

print(data[selecao])

# Não funciona
# data.query('ESTADO=="RIO DE JANEIRO" and PREÇO MÉDIO REVENDA > 2')

print(data.query('ESTADO == "RIO DE JANEIRO" or ESTADO == "SAO
PAULO"'))

selecao_1 = data['ESTADO'] == 'RIO DE JANEIRO'
postos_rj = data[selecao_1]
print(postos_rj)
```

```
selecao_2 = postos_rj['PREÇO MÉDIO REVENDA'] > 2
print(selecao_2)

postos_rj_preco_maior_que_2 = postos_rj[selecao_2]
print(postos_rj_preco_maior_que_2)



selecao_1 = (data['ESTADO'] == 'SAO PAULO') | (data['ESTADO'] == 'RIO
DE JANEIRO')
selecao_2 = (data['PRODUTO'] == 'GASOLINA COMUM')
selecao_3 = (data['PREÇO MÉDIO REVENDA'] > 2)

selecao_final = selecao_1 & selecao_2 & selecao_3
print(selecao_final)

data_filtrado = data[selecao_final]
print(data_filtrado)

print(data_filtrado['ESTADO'].unique())
print(data_filtrado['PRODUTO'].unique())

selecao_1 = (data['ESTADO'] == 'SAO PAULO') | (data['ESTADO'] == 'RIO
DE JANEIRO')
postos_sp_rj = data[selecao_1]

# apenas registros de postos dos estados de SP e RJ
print(postos_sp_rj)

selecao_2 = (postos_sp_rj['PRODUTO'] == 'GASOLINA COMUM')
postos_sp_rj_gasolina = postos_sp_rj[selecao_2]

# apenas registros de postos dos estados de SP e RJ cujo produto é
GASOLINA COMUM
print(postos_sp_rj_gasolina)

selecao_3 = (postos_sp_rj_gasolina['PREÇO MÉDIO REVENDA'] > 2)

postos_sp_rj_gasolina_preco_maior_que_2 =
postos_sp_rj_gasolina[selecao_3]
print(postos_sp_rj_gasolina_preco_maior_que_2)
```

```
selecao = (data['ANO'] == 2008) | (data['ANO'] == 2010) | (data['ANO']
== 2012)
print(data[selecao])

lista_de_anos = [2008, 2010, 2012]
selecao = data['ANO'].isin(lista_de_anos)  # retorna uma Series de
booleanos
print(data[selecao])

print(data.query('ANO in @lista_de_anos'))

for index, row in data.head(10).iterrows():
    print(f'indice {index} ==> {row["ESTADO"]}')
```

aula02.py:

```
import pandas as pd

# Parte 2

data =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019.csv')

print(data.info())

data_pre = data.copy()

data_pre['DATA INICIAL'] = pd.to_datetime(data_pre['DATA INICIAL'])
data_pre['DATA FINAL'] = pd.to_datetime(data_pre['DATA FINAL'])

print(data_pre.info())

# convertendo atributos/colunas para 'numeric'
for atributo in ['MARGEM MÉDIA REVENDA', 'PREÇO MÉDIO DISTRIBUIÇÃO',
'DESVIO PADRÃO DISTRIBUIÇÃO', 'PREÇO MÍNIMO DISTRIBUIÇÃO', 'PREÇO
MÁXIMO DISTRIBUIÇÃO', 'COEF DE VARIAÇÃO DISTRIBUIÇÃO']:
    # converte a coluna (de valores string) para um tipo numérico
    # Em caso de erro na conversão (p. ex., uma string que não
representa um número), um valor vazio (null / nan) será
    # atribuído no lugar
    data_pre[atributo] = pd.to_numeric(data_pre[atributo],
errors='coerce')
```

```
print(data_pre.info())

mask = data_pre['PREÇO MÉDIO DISTRIBUIÇÃO'].isnull()
print(data_pre[mask])
# Nos dados originais, quais eram os valores do PREÇO MÉDIO
DISTRIBUIÇÃO dos registros que agora possuem valores NaN
print(data[mask])

# Retorna uma cópia do data fram `data_pre` com todos os valores NaN de
todas as colunas agora preenchidos com 0.
# Para alterar o próprio data frame, use o argumento `inplace=True`.
data_pre_fill = data_pre.fillna(0)
print(data_pre_fill)
print(data_pre_fill[mask])

# retorna uma cópia do data frame `data_pre` com todos os valores NaN
das colunas:
# 'PREÇO MÉDIO DISTRIBUIÇÃO', 'DESVIO PADRÃO DISTRIBUIÇÃO', 'PREÇO
MÍNIMO DISTRIBUIÇÃO' e 'PREÇO MÁXIMO DISTRIBUIÇÃO', respectivamente,
com os valores: 10, 20, 30, 'vazio'.
data_pre_fill = data_pre.fillna(value={
    'PREÇO MÉDIO DISTRIBUIÇÃO': 10,
    'DESVIO PADRÃO DISTRIBUIÇÃO': 20,
    'PREÇO MÍNIMO DISTRIBUIÇÃO': 30,
    'PREÇO MÁXIMO DISTRIBUIÇÃO': 'vazio'
})

print(data_pre_fill[mask])

# remove, no próprio dataframe, todas as linhas/registros com valores
NaN (vazios) em quaisquer colunas/atributos.
data_pre.dropna(inplace=True)
print(data_pre.info())

data_pre.to_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019_pre
processado_final.csv', index=False)
```

aula03.py:

```
import pandas as pd

# Parte 3
```

```
data_final =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019_preproc
essado_final.csv')
print(data_final)

print(data_final.describe())

print(data_final['PREÇO MÉDIO REVENDA'].describe())

print(data_final.describe()['PREÇO MÉDIO REVENDA'])

stats = data_final.describe()
print(stats)

print(stats[['PREÇO MÉDIO REVENDA', 'PREÇO MÁXIMO REVENDA', 'PREÇO
MÉDIO DISTRIBUIÇÃO']])

# ALTERNATIVA MAIS EFICIENTE
# apenas computa as estatísticas descritivas para 3 atributos
print(data_final[['PREÇO MÉDIO REVENDA', 'PREÇO MÁXIMO REVENDA', 'PREÇO
MÉDIO DISTRIBUIÇÃO']].describe())

print(stats.loc[['min', 'max', 'mean']])

print(stats.loc[['min', 'max', 'mean'], 'PREÇO MÉDIO REVENDA'])

print(stats.loc[['min', 'max', 'mean'], ['PREÇO MÉDIO REVENDA', 'PREÇO
MÉDIO DISTRIBUIÇÃO']])

print(data_final['PREÇO MÍNIMO REVENDA'].min())

mean = data_final['PREÇO MÍNIMO REVENDA'].mean()
std = data_final['PREÇO MÍNIMO REVENDA'].std()

print(f'A média dos preços mínimos de revenda é {mean:.2f} +-
{std:.2f}')

print(sorted(data_final['ESTADO'].unique()))

print(data_final['ESTADO'].value_counts())  # retorna em ordem
decrescente, a quantidade de registros/linhas para cada estado
```

```
print(data_final['ESTADO'].value_counts().to_frame())  # converte uma
Series para um DataFrame

data_final.to_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019_p
reprocessado_final.csv', index= False)
```

aula04.py:

```
import pandas as pd

# Parte 4

df = pd.DataFrame({ 'A': [1, 2, 3, 4],
                    'B': [10, 20, 30, 40],
                    'C': [100, 200, 300, 400]},
                     index=['Linha 1', 'Linha 2', 'Linha 3', 'Linha
4'])

print(df)

def nossa_soma(series):
    return series.sum()  # retorna a soma de todos os valores de uma
series

# aplica a função soma para cada linha do dataframe
df['SOMA(A, B, C)'] = df.apply(nossa_soma, axis=1)
print(df)

# aplica a função soma para cada coluna do dataframe
df.loc['Linha 5'] = df.apply(nossa_soma, axis=0)
print(df)

df['MEDIA(A, B, C)'] = df[['A', 'B', 'C']].apply(lambda series:
series.mean(), axis=1)
print(df)

# Aplica a lambda function abaixo para cada elemento da coluna
df['C * 2'] = df['C'].apply(lambda x: x * 2)
print(df)

df['A * 2'] = df['A'] * 2
print(df)
```

```
# retorna um novo dataframe com todos os elementos ao quadrado.
# poderíamos usar uma função ao invés de uma lambda function
print(df.map(lambda x: x ** 2))
print(df)

nomes = pd.Series(['João', 'Maria', 'Alice', 'Pedro'])
print(nomes)

# retorna uma nova Series com todos os nomes com letras maiuscúlas.
# poderíamos usar uma função ao invés de uma lambda function
print(nomes.map(lambda x: x.upper()))

# O Pandas já fornece uma série de métodos para manipulação de strings.
# Assim, poderíamos usar o código abaixo para obter o mesmo resultado.
print(nomes.str.upper())
```

aula05.py:

```
import pandas as pd

# Parte 5

data_final =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019_preproc
essado_final.csv')

# agrupa as linhas da tablea de acordo com suas respectivas regiões
grupos = data_final.groupby('REGIÃO')
print(grupos)

# retorna os grupos obtidos pelo groupby
print(grupos.groups)

# retorna os índices das linhas/observações de cada grupo
print(grupos.indices)

# retorna um dataframe apenas com as observações do grupo 'CENTRO
OESTE'
print(grupos.get_group('CENTRO OESTE'))

# descreva para nós algumas estatística descritivas para as observações
de cada grupo
```

```
print(grupos.describe())

# print(grupos.mean()) (não funciona +, função mean antigamente
ignorava colunas não numéricas)

print(grupos.min())

print(data_final.groupby('REGIÃO').min())

# Agrupa os registros do dataframe primeiramente por suas regiões.
# Então, agrupa os registros de cada região (grupo) de acordo com seus
produtos.
grupos = data_final.groupby(['REGIÃO', 'PRODUTO'])
print(grupos)

print(grupos.groups)

# print(grupos.mean())

print(grupos['PREÇO MÉDIO REVENDA'].mean())

print(grupos['PREÇO MÉDIO REVENDA'].describe())

df = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9],
                   [None, None, None]],
                  columns=['A', 'B', 'C'])
print(df)

print(df.agg([sum, min]))  # ignora o nan

grupos = data_final.groupby('REGIÃO')
print(grupos)

# Computa o menor e maior valor do 'PREÇO MÉDIO REVENDA' para cada
região (grupo)
print(grupos['PREÇO MÉDIO REVENDA'].agg([min, max]))
```

aula06.py:

```
import pandas as pd
```

```
# Parte 6

notas = pd.DataFrame({
    'nome': ['João', 'Maria', 'José', 'Alice'],
    'idade': [20, 21, 19, 20],
    'nota_final': [5.0, 10.0, 6.0, 10.0]
})
print(notas)

print(notas.sort_values(by='nota_final'))

print(notas.sort_values(by='nota_final', ascending=False))

print(notas.sort_values(by=['nota_final', 'nome'], ascending=[False,
True]))

print(notas)
print(notas.sort_values(by=['nota_final', 'nome'], ascending=[False,
True], inplace=True))
print(notas)
```

aula07.py:

```
import pandas as pd

# Parte 7

data_final =
pd.read_csv('D:/LIPAI/src/07-pandas/GasPricesinBrazil_2004-2019_preproc
essado_final.csv')

print(data_final.query('ANO != 2019'))

df = data_final.query('ANO != 2019')
print(df)

grupos = df.groupby('PRODUTO')
print(grupos['REGIÃO'].value_counts().to_frame())

gasolina_sp_2018 = df.query('PRODUTO == "GASOLINA COMUM" and ESTADO ==
"SAO PAULO" and ANO == 2018')
print(gasolina_sp_2018.head())
```

```
print(gasolina_sp_2018.shape)

print(gasolina_sp_2018.describe())

print(gasolina_sp_2018['PREÇO MÉDIO REVENDA'].describe().to_frame())

print(df.query('(PRODUTO == "GASOLINA COMUM" or PRODUTO == "ETANOL
HIDRATADO") and ESTADO == "SAO PAULO" and ANO == 2018'))

lista_de_estados = ["GASOLINA COMUM", "ETANOL HIDRATADO"]
print(df.query('PRODUTO in @lista_de_estados and ESTADO == "SAO PAULO"
and ANO == 2018'))

gasolina_etanol_sp_2018 = df.query('PRODUTO in ["GASOLINA COMUM",
"ETANOL HIDRATADO"] and ESTADO == "SAO PAULO" and ANO == 2018')
print(gasolina_etanol_sp_2018)

# considerando os preços do etanol e da gasolina juntos, teremos essas
estatística descritivas
print(gasolina_etanol_sp_2018['PREÇO MÉDIO
REVENDA'].describe().to_frame())

print(gasolina_etanol_sp_2018.groupby('PRODUTO')['PREÇO MÉDIO
REVENDA'].describe())
```

Exercícios:

ex01.py:

```
""" Exercicio 1 S6A2 """

import pandas as pd
import os

# Caminho absoluto baseado no local do ex01.py
caminho = os.path.join(os.path.dirname(__file__),
"classification_results_trial_0001.csv")

df = pd.read_csv(caminho)

# Item 1:
benign_malign = df.groupby('real_class').count()['image_path']
benign = int(benign_malign.get('benign', 0))
malign = int(benign_malign.get('malign', 0))
```

```
print(f'1) Existe {benign} imagens "benign" e {malign} imagens
"malign"')

# Item 2:
errou = df.query('real_class != predicted_class')
print('\n2) Imagens em que errou:')
for imagem in errou['image_path']:
    print(imagem)

# Item 3: (Nota: confiança foi tratada como superior a 0.75)
def confianca(linha):
    if linha['predicted_class'] == 'benign':
        return linha['prob_benign']
    else:
        return linha['prob_malign']
errou['confianca'] = errou.apply(confianca, axis=1)
errou_confiante = errou[errou['confianca'] > 0.75]
print('\n3) Errou de forma confiante:')
for imagem in errou_confiante['image_path']:
    print(imagem)

# Item 4:
positivo = 'malign'
negativo = 'benign'

TP = ((df['real_class'] ==positivo) & (df['predicted_class']
==positivo)).sum()
TN = ((df['real_class'] == negativo) & (df['predicted_class'] ==
negativo)).sum()
FP = ((df['real_class'] == negativo) & (df['predicted_class']
==positivo)).sum()
FN = ((df['real_class'] ==positivo) & (df['predicted_class'] ==
negativo)).sum()

print(f"\n4) TP: {TP}")
print(f"TN: {TN}")
print(f"FP: {FP}")
print(f"FN: {FN}")

# Item 5:
print(f'\n5) Acurácia: {(TP + TN)/(TP + TN + FP + FN)}')
print(f'Precisão: {TP/(TP+FP)}')
```

```
print(f'Recall: {TP/(TP+FN)}')
print(f'Especificidade: {TN/(TN+FP)}')

# Item 6:
menor_prob_benign = df.sort_values(by='prob_benign')
top_menor_prob_benign = (menor_prob_benign.query('real_class ==
"benign"')).head()
print('\n6) 5 benigns com menor prob_benign:')
for linha in top_menor_prob_benign.itertuples(index=False):
    print(f'{linha.image_path}: {linha.prob_benign:.4f}')

# Item 7:
maior_prob_benign = df.sort_values(by='prob_benign', ascending= False)
top_maior_prob_benign = (maior_prob_benign.query('real_class ==
"malign"')).head(5)
print('\n7) 5 maligns com maior prob_benign:')
for linha in top_maior_prob_benign.itertuples(index=False):
    print(f'{linha.image_path}: {linha.prob_benign:.4f}')
```