UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Raimundo Rafael Vieira de Brito e Ryan Guilherme Alves de Mesquita

1. Introdução

Na realização dessa avaliação foram disponibilizados dois dataset da plataforma kaggle, onde um dataset era de regressão e o outro de classificação.

2. Observações

Classificação:

O dataset de classificação foi usado o Dataset Coletado, que foi coletado na plataforma da Kaggle, ele tem dados e informações de recursos de telefones celulares e relações entre elas (por exemplo: RAM, Memória interna e preços de vendas. O objetivo não é prever o preço ideal, mas uma faixa de preço de quão alto é o preço delas.

Características:

- **Poder da bateria**
- **Se possui bluetooth**
- **Velocidade do processador**
- **Se possui Dual Sim**
- **Tamanho de memória interna**

Regressão:

O dataset de regressão foi coletado também do Kaggle, e contém informações sobre estimativas de porcentagem de gordura corporal determinada por água, pesagem e outras medidas de circunferência corporal para homens. Seu objetivo principal é ter uma medida precisa de gordura corporal.

Características:

- **Porcentagem de gordura corporal da equação de siri (1956)**
- **Idade**
- **Peso**
- **Altura**
- **Medidas de circunferências**

Resultados e discussão

Questão 1:

1 - Quando os dados foram coletados do dataset **Dataset_coletado,** todas as colunas foram para uma média de 0 a 1 com o StandardScaler. Porém, uma coluna mostrava dados quase balanceados, no caso a coluna **four_g** apresenta informações quase normalizadas, é a única coluna que apresenta não está completamente balanceada com uma predominância de valor -0.97.

Fluxograma:

1. O código lê o arquivo **Dataset_coletado** e carrega os dados do dataframe**(df)**
2. O **df.info** mostra as informações do dataset
3. O **df.dropna** remove os valores que estão com NaN
4. O **df.cleaned** cria outro dataframe
5. É usado o **df.cleaned.info()** para mostrar as informações do dataframe sem os NaN
6. Em seguida é coletado as 7 primeiras linhas e cria o data frame **df_selected**
7. É usado o **df_selectec.head** para mostras as colunas mais relevantes
8. Em seguida é verificado se as colunas são do tipo objeto. Se forem, ele transforma em valores numéricos.
9. É selecionado as colunas e normalizadas com o **StandardScaler().**
10. E salva o Dataframe no arquivo csv **Dataser_coletado** com o **df_selected.to_csv().**

Implementação:

questao1.py M

AV1 > questao1.py > ...

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

dataset_path = "Dataset_coletado.csv"
df = pd.read_csv(dataset_path)

print("Dados antes de tratar NaN:")
print(df.info())

df_cleaned = df.dropna()
print("Dados após remoção do NaN:")
print(df_cleaned.info())

colunas_relevantes = df_cleaned.columns[:7]
df_selected = df_cleaned[colunas_relevantes]

print("\nDataframe final com colunas relevantes:")
print(df_selected.head())

coluna_classe = df_selected.columns[-1]
print("\nDistribuição das classes:")
print(df_selected[coluna_classe].value_counts())

if df_selected[coluna_classe].dtype == 'object':
    # dtype Converte valores numéricos por códigso
    df_selected[coluna_classe] = df_selected[coluna_classe].astype('category').cat.codes
    print("\nColuna de classes convertida para valores numéricos.")

colunas_numericas = df_selected.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
df_selected[colunas_numericas] = scaler.fit_transform(df_selected[colunas_numericas])
print("\nColunas numéricas normalizadas:")
print(df_selected.head())
df_selected.to_csv("Dataset_coletado.csv", index=False)
print("\nDataset atualizado 'Dataset_coletado.csv'.")
```

Resultados Apresentados:

```
ed\libs\debugpy\adapter/../..\debugpy\launcher' '52078' '--' 'c:\Users\rafae\Documents\GitHub\IA--SI\AV1\questao1.py'
Dados antes de tratar NaN:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             1000 non-null   float64
 1   battery_power  1000 non-null   float64
 5   fc             1000 non-null   float64
 6   four_g         1000 non-null   float64
dtypes: float64(7)
memory usage: 54.8 KB
None
Dados após remoção do NaN:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 2   blue           1000 non-null   float64
 3   clock_speed    1000 non-null   float64
 4   dual_sim       1000 non-null   float64
 5   fc             1000 non-null   float64
memory usage: 54.8 KB
None

Dataframe final com colunas relevantes:
         id  battery_power      blue  clock_speed  dual_sim        fc    four_g
0 -1.730320      -0.475451  0.968496     0.312601  0.966559  2.108676 -0.974329
1 -1.726856      -0.942782  0.968496    -1.255832  0.966559 -0.132927  1.026347
2 -1.723391       1.292077  0.968496     1.519087 -1.034598 -0.805408 -0.974329
3 -1.719927       0.688249 -1.032529    -1.255832  0.966559  3.005317  1.026347
4 -1.716463       0.429135 -1.032529    -0.169994 -1.034598  1.436195  1.026347
```

Questão 2:

2 - Nesse código foi usado o **K-Knearest Neighborn (KNN)** de forma manual onde pode demorar mais tempo para processamento para avaliar seu desempenho. Teve o processamento dos dados onde foi usado a coluna **blue** que foi separada com target, onde foi realizada uma medida com base nas distância dos pontos de conjuntos do treino e do teste.

Medidas avaliadas:

- Mahalanobis
- Chebyshev
- Manhattan
- Euclidiana

Resultados:

- A métrica de **Chebyshev** foi mais eficaz, conseguindo uma acurácia maior em relação às outras medidas.
- A métrica de **Manhattan** foi a menos eficaz do experimento, mostra que não é ideal para esse problema.

Medida das Distâncias	**Acurácia**
Mahalanobis	**54%**
Chebyshev	**57%**
Manhattan	**49%**
Euclidiana	**53%**

Fluxograma:

1. O código lê o o dataset usando o **pd.read_csv()** e armazena em dataset.
2. Usando o **train_test_split** é dividido o dataset em treino e teste
3. Em seguida é definido o knn para calcular a previsão das distâncias e em relação ao valor de k.
4. Converte os valores de treino e teste em números usando o **.to_numpy().**
5. Em seguida a função **compute_accuracy()** calcula a acurácia comparando as predições que tão em **y_pred()** com os valores reais de **y_true().**
6. É definido as quatros funções de distância de Mahalanobis, Chebyshev, Manhattan, e Euclidiana.
7. É feita a chamada do knn em cada função e calcula a acurácia e imprime os resultados.

Implementações:

questao2.py

```python
import pandas as pd
import numpy as np
from scipy.spatial.distance import mahalanobis, cityblock, chebyshev, euclidean
from sklearn.model_selection import train_test_split
from collections import Counter

dataset = pd.read_csv('Dataset_coletado.csv')

X = dataset.drop(columns=['blue'])
y = dataset['blue']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def knn(X_train, y_train, X_test, k, distance_func):
    predictions = []
    for test_point in X_test:
        distances = []
        for i, train_point in enumerate(X_train):
            distance = distance_func(test_point, train_point)
            distances.append((distance, y_train[i]))
        distances.sort(key=lambda x: x[0])
        k_nearest = distances[:k]
        k_nearest_labels = [label for _, label in k_nearest]
        prediction = Counter(k_nearest_labels).most_common(1)[0][0]
        predictions.append(prediction)
    return predictions

X_train_array = X_train.to_numpy()
X_test_array = X_test.to_numpy()
y_train_array = y_train.to_numpy()

def compute_accuracy(y_true, y_pred):
    return np.sum(y_true == y_pred) / len(y_true)

distance_functions = {
    'Mahalanobis': lambda x, y: mahalanobis(x, y, np.linalg.inv(np.cov(X_train_array, rowvar=False))),
    'Chebyshev': chebyshev,
    'Manhattan': cityblock,
    'Euclidean': euclidean
}
results = {}

for name, func in distance_functions.items():
    print(f"Calculando KNN usando {name}...")
    y_pred = knn(X_train_array, y_train_array, X_test_array, k=7, distance_func=func)
    accuracy = compute_accuracy(y_test.to_numpy(), y_pred)
    results[name] = accuracy
    print(f"Acurácia usando {name}: {accuracy:.2f}")

print("\nResultados finais:")
```

Resultados Apresentados:

```
' '52131' '--' 'c:\Users\rafae\Documents\GitHub\IA--SI\AV1\questao2.py'
Calculando KNN usando Mahalanobis...
Acurácia usando Mahalanobis: 0.54
Calculando KNN usando Chebyshev...
Acurácia usando Chebyshev: 0.57
Calculando KNN usando Manhattan...
Acurácia usando Manhattan: 0.49
Calculando KNN usando Euclidean...
Acurácia usando Euclidean: 0.53

Resultados finais:
Mahalanobis: 0.54
Chebyshev: 0.57
Manhattan: 0.49
Euclidean: 0.53
```

3 - Não concluído

4 - Não concluído

5 - Esse é o código do dataset de regressão que tem como objetivo fazer uma análise sobre gordura corporal que é coletado com base no dataset **bodyfat.** O dataset foi dividido em 80% de treinamento e 20% para teste**.** Os dados foram normalizados para garantir que as variáveis estão todas na mesma escala.

Fluxograma:

1. O código lê o dataset **pd.read_csv()** e armazena ele no **df().**
2. É mostrado as cinco primeiras linhas **df.head().**
3. É verificado as linhas e mostradas o número de valores NaN nas colunas **df.isnull().sum().**
4. É removido as linhas com valor NaN **df.dropna().**
5. Em seguida é separado os valores das variáveis **X** e **Y** do dataset **BodyFat**
6. É feito uma matriz de relação com as variáveis **df.corr()**. e um gráfico para exibir **sns.heatmap()**.
7. É normalizado as variáveis com o **StandardScaler.**

Implementações:

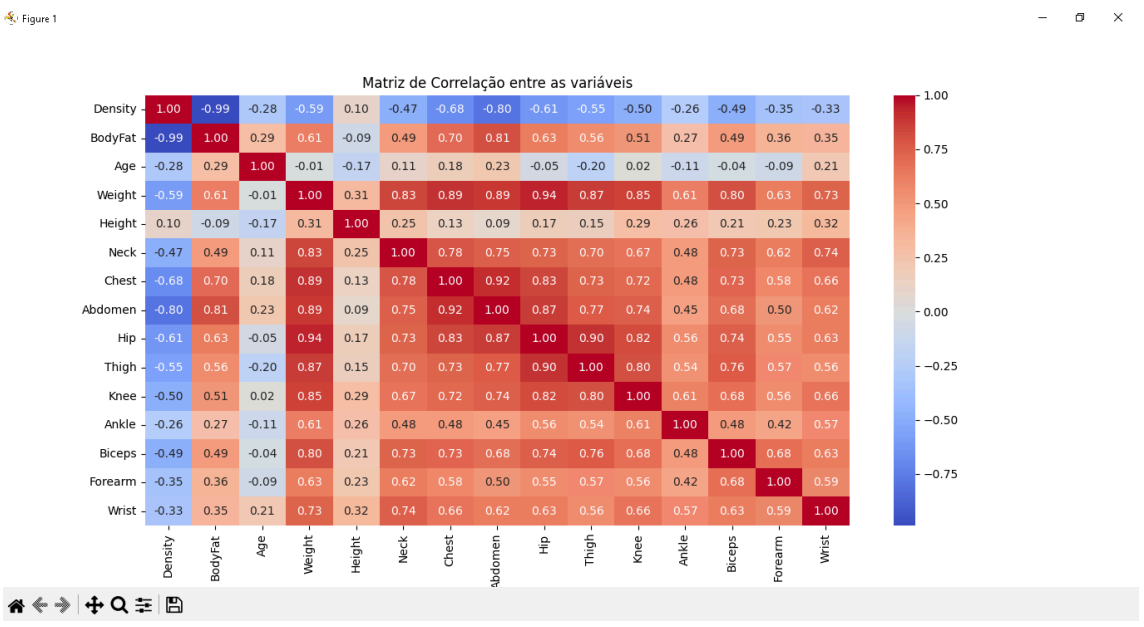
questao5.py

AV1 > questao5.py > ...

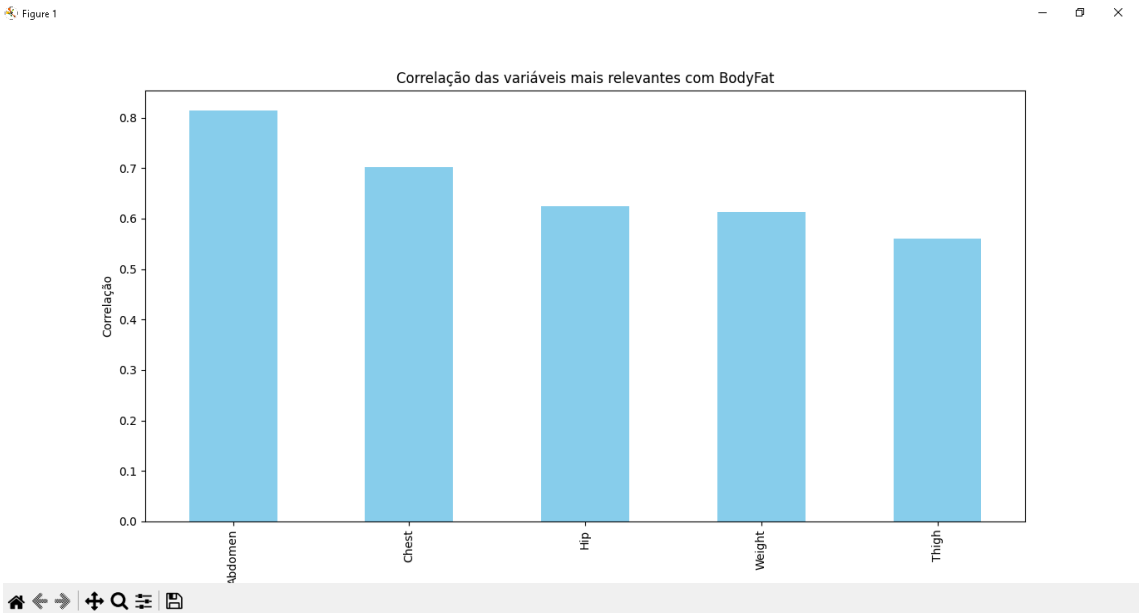
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

dataset_path = 'bodyfat.csv'

df = pd.read_csv(dataset_path)

print(df.head())

X = df.drop("BodyFat", axis=1)
y = df["BodyFat"]

print(df.isnull().sum())
df.dropna(inplace=True)

X = df.drop("BodyFat", axis=1)
y = df["BodyFat"]

correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matriz de Correlação entre as variáveis")
plt.show()

correlations = correlation_matrix["BodyFat"].sort_values(ascending=False)
print(correlations)

top_features = correlations[1:6]
plt.figure(figsize=(10, 6))
top_features.plot(kind='bar', color='skyblue')
plt.title('Correlação das variáveis mais relevantes com BodyFat')
plt.ylabel('Correlação')
plt.xlabel('Variáveis')
plt.show()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Resultados Apresentados:

```
' '52167' '--' 'c:\Users\rafae\Documents\GitHub\IA--SI\AV1\questao5.py'
   Density  BodyFat  Age  Weight  Height  Neck  Chest  Abdomen    Hip  Thigh  Knee  Ankle  Biceps  Forearm  Wrist
0   1.0708      12.3   23  154.25   67.75  36.2   93.1     85.2   94.5   59.0  37.3   21.9    32.0     27.4   17.1
1   1.0853       6.1   22  173.25   72.25  38.5   93.6     83.0   98.7   58.7  37.3   23.4    30.5     28.9   18.2
2   1.0414      25.3   22  154.00   66.25  34.0   95.8     87.9   99.2   59.6  38.9   24.0    28.8     25.2   16.6
3   1.0751      10.4   26  184.75   72.25  37.4  101.8     86.4  101.2   60.1  37.3   22.8    32.4     29.4   18.2
4   1.0340      28.7   24  184.25   71.25  34.4   97.3    100.0  101.9   63.2  42.2   24.0    32.2     27.7   17.7
Density     0
BodyFat     0
Age         0
Weight      0
Height      0
Neck        0
Chest       0
Abdomen     0
Hip         0
Thigh       0
Knee        0
Ankle       0
Biceps      0
Forearm     0
Wrist       0
dtype: int64
```



Esse gráfico apresenta a matriz de relação entre todas as variáveis.



Esse gráfico apresenta as cinco variáveis mais relevantes com o BodyFat. Portanto mostra os cinco locais onde é mais fácil de deduzir que a gordura corporal em uma indivíduo.

Resultados:

Variável	Possui correlação com o bodyfat
Abdômen	**0.80**
Weight	**0.75**
Chest	**0.70**
Hip	**0.65**
Thigh	**0.63**

As variáveis apresentadas são as que mais possuem relação com **bodyfat.** No caso, é mais fácil deduzir que essa pessoa possui gordura corporal nesse ponto.

6 - Nesse código é feito uma análise de regressão linear para prever a porcentagem de gordura corporal com base no dataset bodyfat, a regressão linear é feita usando uma biblioteca chamada **LinearRegression** da **sklearn.**

Medidas avaliadas:

- RSS
- MSE
- RMSE
- R^2

Resultados:

Medidas	**Valores Reais**
RSS	**19.3959**
MSE	**0.3803**
RMSE	**0.6167**
R^2	**0.9918**

O gráfico gerado compara os valores reais que tem no bodyfat. Portanto, com base nas medidas que foram usadas, o **R^2** foi o que conseguiu medidas de erro muito baixas, que mostra que o modelo foi bem preciso, apresentando que tem uma capacidade de prever gordura corporal muito bem.

Fluxograma:

1. É lido o arquivo csv **pd.read_csv** e colocado em **df.**
2. É verificado os valores nulos **df.isnull().sum()** e removidos **df.dropna().**
3. É separado as variáveis do **BodyFat** em **X e Y..**
4. Em seguida é normalizado com o **StandardScale().**
5. É usado **train_test_split** para separar em treino **(X)** e teste **(y)**
6. É feito o treinando do modelo com o **LinearRegression().fit()**
7. É feito as predições no conjunto com testes usando **regression.predict().**
8. Calcula o **RSS, MSE, MRSE, R^2** para comprar as previsões e valores reais e imprimi.

Implementação:

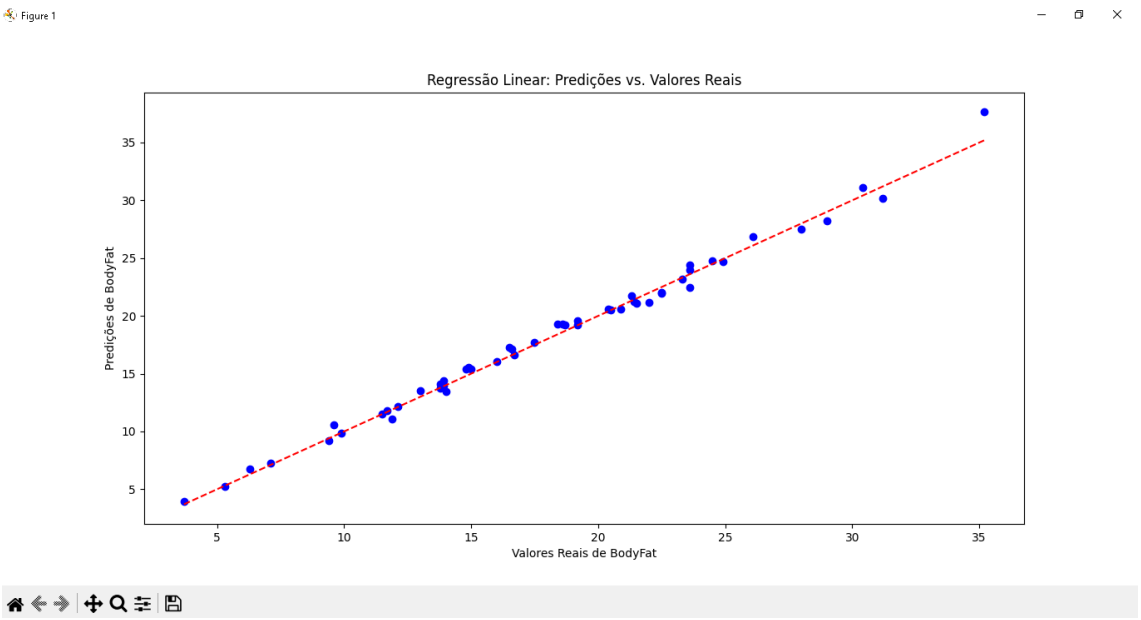
questao6.py M

AV1 > questao6.py > ...

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

dataset_path = 'bodyfat.csv'
df = pd.read_csv(dataset_path)

print(df.isnull().sum())

df = df.dropna()

X = df.drop("BodyFat", axis=1)
y = df["BodyFat"]


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

rss = np.sum((y_test - y_pred) ** 2)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r_squared = r2_score(y_test, y_pred)

print(f"RSS: {rss:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R²: {r_squared:.4f}")

plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
plt.title('Regressão Linear: Predições vs. Valores Reais')
plt.xlabel('Valores Reais de BodyFat')
plt.ylabel('Predições de BodyFat')
plt.show()
```

Resultados Apresentados:

```
' '52189' '--' 'c:\Users\rafae\Documents\GitHub\IA--SI\AV1\questao6.py'
Density    0
BodyFat    0
Age        0
Weight     0
Height     0
Neck       0
Chest      0
Abdomen    0
Hip        0
Thigh      0
Knee       0
Ankle      0
Biceps     0
Forearm    0
Wrist      0
dtype: int64
RSS: 19.3959
MSE: 0.3883
RMSE: 0.6167
R²: 0.9918
```



Esse gráfico mostra a regressão linear em comparação com os valores reais em relação às previsões apresentadas pelos modelos.

7 - Esse código tem a função de fazer uma validação cruzada (k-fold) para três modelos de regressão linear para prever a porcentagem de gordura corporal.

Modelos de Regressão Linear:

- Linear
- Lasso
- Ridge

Resultados:

Modelo Linear:

Modelo Linear	**Medidas de Desempenho**
RSS	**100.16**
MSE	**1.997**
RMSE	**1.205**
R^2	**0.9711**

Modelo Ridge:

Modelo Ridge	**Medidas de Desempenho**
RSS	**99.76**
MSE	**1.988**
RMSE	**1.219**
R^2	**0.9712**

Modelo Lasso:

Modelo Lasso	**Medidas de Desempenho**
RSS	**93.03**
MSE	**1.857**
RMSE	**1.141**

R^2	**0.9733**

Das três tabelas apresentadas o **Modelo Lasso** possui menor margem de erros do que os outros modelos e maior R^2, mostrando que possui maior desempenho em prever as porcentagem de gordura corporal.

Fluxograma:

1. É lido o arquivo csv com o **df.read_csv()** e armazenado no **df**
2. É removido os valores NaN **df.dropna.**
3. Em seguida é separados as variáveis **dependentes(y)** e **independentes(X)**
4. É feito a normalização com o **StandardScaler().**
5. Implementada a função **k_fold_cross_validation()** que divide os dados de **k**, treina os modelos com os folds de treino.
6. Predição com os **folds(k)** de testes e cálculo das métricas **RSS, MSE, RMSE, R^2.**
7. É feito a validação cruzada com **k=5** para a **Regressão Linear, Regressão Ridge e Regressão Lasso.**
8. É realizado a comparação dos modelos e exibidos a medição de cada um e identifica o com menor RMSE, mostrando o mais eficaz.

Implementação:

questao7.py

AV1 > questao7.py > ...

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

dataset_path = 'bodyfat.csv'
df = pd.read_csv(dataset_path)

df = df.dropna()

X = df.drop("BodyFat", axis=1)
y = df["BodyFat"]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

def k_fold_cross_validation(X, y, k, model_class, **model_kwargs):
    fold_size = len(X) // k
    results = {"RSS": [], "MSE": [], "RMSE": [], "R2": []}

    for i in range(k):
        # Divisão dos dados em treino e teste
        test_start = i * fold_size
        test_end = (i + 1) * fold_size if i < k - 1 else len(X)

        X_train = np.concatenate([X[:test_start], X[test_end:]])
        y_train = np.concatenate([y[:test_start], y[test_end:]])
        X_test, y_test = X[test_start:test_end], y[test_start:test_end]

        model = model_class(**model_kwargs)
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        rss = np.sum((y_test - y_pred) ** 2)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)

        results["RSS"].append(rss)
        results["MSE"].append(mse)
        results["RMSE"].append(rmse)
        results["R2"].append(r2)

    metrics = {key: np.mean(value) for key, value in results.items()}
    return metrics
```

```python
    return metrics

k = 5

linear_metrics = k_fold_cross_validation(X_scaled, y, k, LinearRegression)

ridge_metrics = k_fold_cross_validation(X_scaled, y, k, Ridge, alpha=1.0)

lasso_metrics = k_fold_cross_validation(X_scaled, y, k, Lasso, alpha=0.1)

print(f"Modelo Linear (Clássico) - Métricas: {linear_metrics}")
print(f"Modelo Ridge - Métricas: {ridge_metrics}")
print(f"Modelo Lasso - Métricas: {lasso_metrics}")
models = ["Linear", "Ridge", "Lasso"]
metrics_dict = {"Linear": linear_metrics, "Ridge": ridge_metrics, "Lasso": lasso_metrics}

best_model = min(models, key=lambda model: metrics_dict[model]["RMSE"])
print(f"\nO melhor modelo é: {best_model}")
```

Resultados Esperados:

```
4.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '52231' '--' 'c:\Users\rafae\Documents\GitHub\IA--SI\AV1\questao7.py'
Modelo Linear (Clássico) - Métricas: {'RSS': 100.15681831785741, 'MSE': 1.9971736929376251, 'RMSE': 1.2047280754774439, 'R2': 0.9711267589947374}
Modelo Ridge - Métricas: {'RSS': 99.75643560894264, 'MSE': 1.988290204711289, 'RMSE': 1.2185169330012697, 'R2': 0.9711758108891393}
Modelo Lasso - Métricas: {'RSS': 93.0928383140289, 'MSE': 1.85739128604461, 'RMSE': 1.140595379374625, 'R2': 0.9732519940104198}

O melhor modelo é: Lasso
PS C:\Users\rafae\Documents\GitHub\IA--SI>
```