UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Luan Sales Barroso

1. Introdução

Water quality:

Este é um dataset criado a partir de dados imaginários de qualidade da água em um ambiente urbano. O conteúdo desse dataset são linhas que têm valores numéricos que dizem a quantidade de determinado ingrediente na água e as colunas dizem os nomes dos ingredientes de água sendo a última coluna, is-safe, a conclusão se a água é ou não segura(0 - não seguro, 1 - seguro). Neste, todas as colunas têm relevância, mas o foco principal seria a última coluna a is-safe. O que será analisado neste dataset, é se a água é ou não segura para o consumo.

Car Price Prediction Challenge:

Este é um dataset para praticar regressão prevendo os preços de diferentes carros. O conteúdo deste dataset são colunas indicando as características de alguns carros, como por exemplo: preço, modelo, fabricante, entre outros. E linhas que dizem quais as características dos carros em questão. Os ajustes feitos no dataset foram realizados de forma manual e eles incluem: mudança de tipo de dado(string para float), ajuste de campos e exclusão de colunas inúteis.

2. Observações

O dataset **Water quality** tinha um problema muito estranho, alguns elementos das colunas “ammonia” e “is_safe” estavam com a denominação “#NUM!”, na primeira questão o código não conseguiu identificar essa anomalia como um valor NaN/vazio o que dificultava muito a realização das seguintes questões.

3. Resultados e discussão

Questão 1:

Inicialmente um arquivo utils.py foi criado para facilitar a manipulação dos conjuntos de dados, então as verificações da quantidade de linhas e colunas, da presença de valores NaN/vazios e o tipo de dados foram realizadas, nisso foi constatado que duas colunas tinha o tipo “object” enquanto as demais tinham o tipo “float64”, ao revisar o dataset o problema foi descoberto, nas colunas “ammonia” e “is_safe” tinham, nas mesmas linhas, a seguinte anomalia “#NUM!” que era

completamente ignorado pela função de verificar a presença de valores NaN/vazios a solução para isso foi modificar manualmente os #NUM! para NaN e com isso foi constatado que o dataset tinha 6 valores vazios, a solução para isso foi a criação de um novo arquivo manualmente excluindo as linhas com valores NaN e nomeando ele de "water_quality_ajustado.csv".

Assim ficou os códigos:

utils.py

```python
import pandas as pd

def load_water_quality_old():  2 usages  Luanvraft
    return pd.read_csv("dataset/water_quality.csv")

def load_water_quality():  8 usages  Luanvraft
    return pd.read_csv("dataset/water_quality_ajustado.csv")

def load_car_price_prediction_old():  Luanvraft
    return pd.read_csv("dataset/car_price_prediction.csv")

def load_car_price_prediction():  6 usages  Luanvraft
    return pd.read_csv("dataset/car_price_prediction_ajustado.csv")
```

```python
from AV1.src.utils import load_water_quality_old, load_water_quality

waterQuality = load_water_quality_old()
newWaterQuality = load_water_quality()

print(waterQuality.shape)
print(waterQuality.isna().sum())
print(waterQuality.dtypes)
print(waterQuality)
print(newWaterQuality.isna().sum())
print(newWaterQuality.dtypes)
print(newWaterQuality)
```

Questão 2:

Seguindo as orientações, o dataset foi carregado, os valores de X e y foram definidos, houve a divisão dos dados em treino e teste. Então as funções para os cálculos das distâncias foram definidas: as distâncias em questão são distância

Euclidiana, Manhattan, Chebyshev e Mahalanobis. Então uma implementação manual do KNN foi definida tendo como base K = 7 e por fim, uma implementação para avaliar o KNN foi definida. Infelizmente no meu caso, a distância de Mahalanobis teve o melhor resultado, isso me gerou um pouco de trabalho a mais nas outras questões, mas nada que me impedisse de concluí-las.

Assim ficou o código(de forma resumida):

```python
from sklearn.model_selection import train_test_split
from AV1.src.utils import load_water_quality
import numpy as np
import math

waterQuality = load_water_quality()

X = waterQuality.drop(['is_safe'], axis=1).values
y = waterQuality['is_safe'].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, random_state=42)

treinamento = np.hstack((X_train, y_train.reshape(-1, 1)))
teste = np.hstack((X_test, y_test.reshape(-1, 1)))

def dist_euclidiana(v1, v2):...

def dist_manhattan(v1, v2):...

def dist_chebyshev(v1, v2):...

def dist_mahalanobis(v1, v2, matriz_cov):...

def knn(treinamento, nova_amostra, K, distancia_func, matriz_cov=None):...
```

```python
def avaliar_knn(teste, treinamento, K, distancia_func, matriz_cov=None):...

matriz_cov = np.cov(X_train.T)

K = 7
print("Porcentagem de acertos distância euclidiana:",
      avaliar_knn(teste, treinamento, K, dist_euclidiana))
print("Porcentagem de acertos distância manhattan:",
      avaliar_knn(teste, treinamento, K, dist_manhattan))
print("Porcentagem de acertos distância chebyshev:",
      avaliar_knn(teste, treinamento, K, dist_chebyshev))
print("Porcentagem de acertos distância mahalanobis:",
      avaliar_knn(teste, treinamento, K, dist_mahalanobis, matriz_cov))
```

Resultados:

```
Porcentagem de acertos distância euclidiana: 88.59429714857428
Porcentagem de acertos distância manhattan: 89.09454727363682
Porcentagem de acertos distância chebyshev: 87.943971985993
Porcentagem de acertos distância mahalanobis: 91.24562281140571
```

Questão 3:

Nessa questão, o objetivo era verificar se com a normalização logarítmica ou a normalização de média zero e variância unitária, teriam uma acurácia melhor do que rodar o KNN sem uma normalização dos dados. Os resultados foram surpreendentes, já que a normalização além da normalização não ter mudado nada, até diminuíram a precisão, como é possível observar na saída do código.

O código ficou assim:

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from AV1.src.utils import load_water_quality
from sklearn.metrics import accuracy_score
import numpy as np

waterQuality = load_water_quality()

X = waterQuality.drop(['is_safe'], axis=1).values
y = waterQuality['is_safe'].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, random_state=42)

X_train_log = np.log1p(X_train)
X_test_log = np.log1p(X_test)

scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)
```

```python
def knn_mahalanobis(X_train, X_test, y_train, y_test, n_neighbors=7):  2 usages
    cov_matrix = np.cov(X_train.T)
    inv_cov_matrix = np.linalg.inv(cov_matrix)

    knn = KNeighborsClassifier(n_neighbors=n_neighbors, metric='mahalanobis',
                               metric_params={'VI': inv_cov_matrix})

    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return accuracy_score(y_test, y_pred)

accuracy_log = knn_mahalanobis(X_train_log, X_test_log, y_train, y_test)
accuracy_norm = knn_mahalanobis(X_train_norm, X_test_norm, y_train, y_test)

print(f"Acurácia com normalização logarítmica: {accuracy_log * 100}%")
print(f"Acurácia com média zero e variância unitária: {accuracy_norm * 100}%")
```

Esse foi o resultado:

```
Acurácia com normalização logarítmica: 90.99549774887443%
Acurácia com média zero e variância unitária: 91.2456228114057%
```

Questão 4:

Para essa questão, temos o objetivo de descobrir qual o melhor k (quantidade de vizinhos) para o KNN executando o código com a melhor normalização com base na questão anterior. No meu caso a melhor normalização do dataset, tinha o mesmo resultado que o dataset sem normalização mas eu optei por rodar com código com a normalização de média zero e variância unitária. Essa questão é semelhante a anterior, a normalização dos dados é realizada, então o KNN com a distância de Mahalanobis, o diferencial é que o KNN é executado algumas vezes para verificar qual o melhor k (especificamente, 20 vezes), então esses resultados são colocados em um gráfico que mostra qual k tem a melhor acurácia.

Assim ficou o código:

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from AV1.src.utils import load_water_quality
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np

waterQuality = load_water_quality()

X = waterQuality.drop(['is_safe'], axis=1).values
y = waterQuality['is_safe'].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

cov_matrix = np.cov(X_train_scaled.T)
inv_cov_matrix = np.linalg.inv(cov_matrix)
```

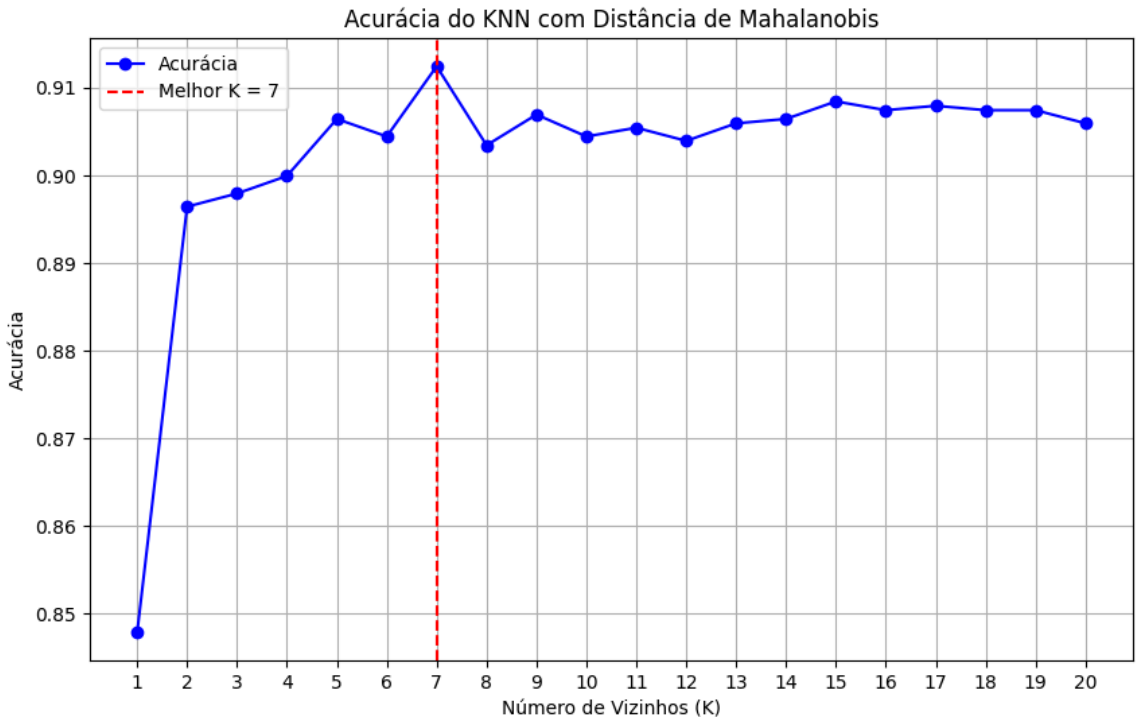
```python
def evaluate_knn_mahalanobis(X_train, X_test, y_train, y_test, n_neighbors):  1 usage  Luanvraft *
    knn = KNeighborsClassifier(n_neighbors=n_neighbors, metric='mahalanobis',
                               metric_params={'VI': inv_cov_matrix})
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return accuracy_score(y_test, y_pred)

k_values = range(1, 21)
accuracies = []

for k in k_values:
    acc = evaluate_knn_mahalanobis(X_train_scaled, X_test_scaled, y_train, y_test, n_neighbors=k)
    accuracies.append(acc)

best_k = k_values[np.argmax(accuracies)]
best_accuracy = max(accuracies)
```

```python
plt.figure(figsize=(10, 6))
plt.plot( *args: k_values, accuracies, marker='o', linestyle='-', color='b', label='Acurácia')
plt.axvline(best_k, linestyle='--', color='r', label=f'Melhor K = {best_k}')
plt.title('Acurácia do KNN com Distância de Mahalanobis')
plt.xlabel('Número de Vizinhos (K)')
plt.ylabel('Acurácia')
plt.xticks(k_values)
plt.legend()
plt.grid()
plt.show()
```

Também é possível ver o resultado sem o gráfico:

```
print(f"Melhor K: {best_k}")
print(f"Acurácia com o melhor K: {best_accuracy * 100}%")
```

Esses foram os resultados:




```
Melhor K: 7
Acurácia com o melhor K: 91.2456228114057%
```

Questão 5:

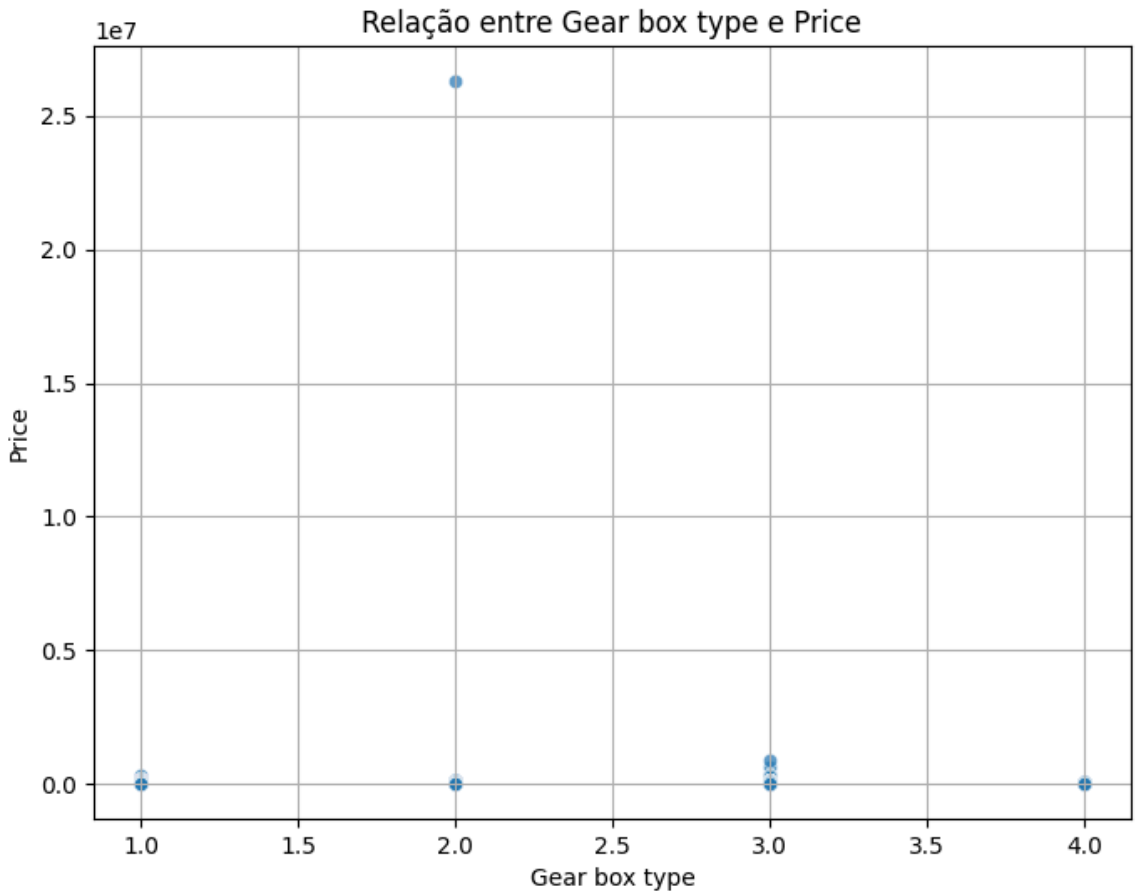
A partir dessa questão, começamos a manipular o outro dataset e para este, o objetivo era realizar uma regressão linear. Inicialmente algumas verificações manuais foram feitas e com isso, valores das colunas foram adaptados para valores numéricos (esses valores antigos e adaptações estão documentados no arquivo “car_aux.txt” que está dentro da pasta “AV1/dataset”) e outras colunas irrelevantes foram deletadas, em seguida foi criado um código para verificar qual atributo é mais relevante para realizar a regressão do alvo escolhido.

Assim ficou o código:

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from AV1.src.utils import (load_car_price_prediction,
                           load_car_price_prediction_old)

carPrice = load_car_price_prediction()
oldCarPrice = load_car_price_prediction_old()

print("Primeiras linhas do dataset antigo:")
print(oldCarPrice.head())
print("Primeiras linhas do dataset ajustado:")
print(carPrice.head())

corr_matrix = carPrice.corr()
print("\nMatriz de Correlação:")
print(corr_matrix)

target_candidates = [col for col in carPrice.columns if
                     carPrice[col].dtype in [np.float64, np.int64]
                     and len(carPrice[col].unique()) > 10]
print("\nCandidatos para alvo (regressão):", target_candidates)

target = "Price"
```

```python
correlations = corr_matrix[target].drop(target).sort_values(key=abs, ascending=False)
print(f"\nCorrelação com o alvo ({target}):")
print(correlations)

most_relevant = correlations.idxmax()
print(f"\nAtributo mais relevante para prever '{target}': {most_relevant}")

plt.figure(figsize=(8, 6))
sns.scatterplot(x=carPrice[most_relevant], y=carPrice[target], alpha=0.7)
plt.title(f'Relação entre {most_relevant} e {target}')
plt.xlabel(most_relevant)
plt.ylabel(target)
plt.grid()
plt.show()
```

Esses foram os resultados:




```
Primeiras linhas do dataset antigo:
         ID  Price  Levy  ...             Wheel   Color  Airbags
0  45654403  13328  1399  ...        Left wheel  Silver       12
1  44731507  16621  1018  ...        Left wheel   Black        8
2  45774419   8467     -  ...  Right-hand drive   Black        2
3  45769185   3607   862  ...        Left wheel   White        0
4  45809263  11726   446  ...        Left wheel  Silver        4

[5 rows x 18 columns]
Primeiras linhas do dataset ajustado:
   Price  Manufacturer  Prod. year  ...  Gear box type  Drive wheels  Airbags
0  13328             1        2010  ...              1             1       12
1  16621             1        2011  ...              3             1        8
2   8467             1        2006  ...              4             2        2
3   3607             1        2011  ...              1             1        0
4  11726             1        2014  ...              1             2        4

[5 rows x 12 columns]
```

```
Matriz de Correlação:
                    Price  Manufacturer  ...  Drive wheels   Airbags
Price            1.000000      0.009573  ...     -0.003419 -0.012824
Manufacturer     0.009573      1.000000  ...      0.002779 -0.002984
Prod. year       0.012982      0.002938  ...     -0.125912  0.236969
Category        -0.019721      0.001381  ...      0.213743  0.131772
Leather interior 0.000754      0.009199  ...     -0.090536  0.165023
Fuel type        0.006364      0.860098  ...      0.003950 -0.007405
Engine volume    0.008753     -0.002668  ...     -0.224052  0.224441
Mileage         -0.001758      0.011239  ...      0.008567 -0.009201
Cylinders        0.007518     -0.002609  ...     -0.208963  0.176868
Gear box type    0.016340      0.005344  ...      0.088333  0.107086
Drive wheels    -0.003419      0.002779  ...      1.000000 -0.023810
Airbags         -0.012824     -0.002984  ...     -0.023810  1.000000

[12 rows x 12 columns]
```

```
Correlação com o alvo (Price):
Category           -0.019721
Gear box type       0.016340
Prod. year          0.012982
Airbags            -0.012824
Manufacturer        0.009573
Engine volume       0.008753
Cylinders           0.007518
Fuel type           0.006364
Drive wheels       -0.003419
Mileage            -0.001758
Leather interior    0.000754
Name: Price, dtype: float64

Atributo mais relevante para prever 'Price': Gear box type
```

Questão 6:

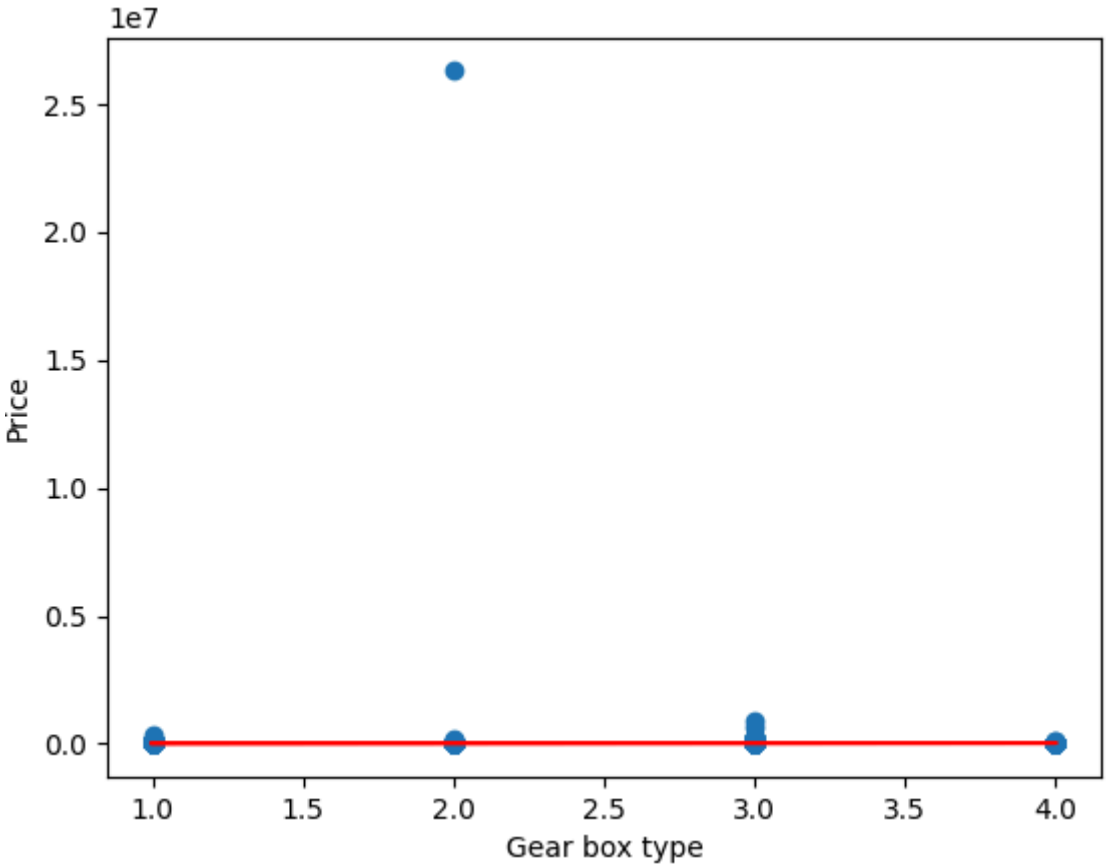
Com o atributo alvo e o atributo mais relevante definidos na questão anterior, o objetivo dessa questão é realizar, e demonstrar com um gráfico, uma regressão linear com esse atributo mais relevante para a predição do atributo alvo e determinar o RSS, MSE, RMSE e R_squared para esta regressão baseada somente no atributo mais relevante. Meu gráfico ficou meio estranho mas teoricamente está certo.

O código ficou assim:

```python
from sklearn.model_selection import train_test_split
from AV1.src.utils import load_car_price_prediction
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

carPrice = load_car_price_prediction()

X = carPrice["Gear box type"].values.reshape(-1, 1)
y = carPrice["Price"].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2,
                                                    random_state=42)

reg = LinearRegression()
reg.fit(X, y)

y_pred = reg.predict(X)

plt.scatter(X, y)
plt.plot( *args: X, y_pred, color="red")
plt.xlabel("Gear box type")
plt.ylabel("Price")
plt.show()
```

```python
r_squared = reg.score(X_test, y_test)
rss = np.square(y-y_pred)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)

print(f"RSS: {rss}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R^2: {r_squared}")
```

E esse foi meu resultado:




```
RSS: [1.13244809e+07 4.92620185e+07 3.47671178e+08 ... 8.86387111e+07
 1.29099285e+08 2.63191784e+08]
MSE: 36309634893.56594
RMSE: 190550.87219313887
R^2: 0.0026068104668254266
```

Questão 7:

Por fim, para essa questão era necessário fazer uma regressão linear utilizando uma implementação manual do KFold e do Cross-validation em seguida calcular as métricas RSS, MSE, RMSE e R_squared que também deveriam utilizar uma implementação manual.

Assim ficou o código:

```python
from AV1.src.utils import load_car_price_prediction
from sklearn.linear_model import LinearRegression
import numpy as np

carPrice = load_car_price_prediction()

def compute_RSS(predictions, y):  3 usages  Luanvraft
    aux = np.square(y - predictions)
    RSS = np.sum(aux)
    return RSS

def compute_MSE(predictions, y):  2 usages  Luanvraft
    RSS = compute_RSS(predictions, y)
    MSE = np.divide( *args: RSS, len(predictions))
    return MSE

def compute_RMSE(predictions, y):  1 usage  Luanvraft
    MSE = compute_MSE(predictions, y)
    RMSE = np.sqrt(MSE)
    return RMSE

def compute_R_squared(predictions, y):  1 usage  Luanvraft
    var_pred = np.sum(np.square(y - np.mean(y)))
    var_data = compute_RSS(predictions, y)
    r_squared = np.divide( *args: var_pred, var_data)
    return r_squared
```

```python
class KFold:  1 usage  Luanvraft

    def __init__(self, n_splits):  Luanvraft
        self.n_splits = n_splits

    def _compute_score(self, obj, X, y):  1 usage  Luanvraft
        obj.fit(X[0], y[0])
        return obj.score(X[1], y[1])

    def cross_val_score(self, reg, X, y):  1 usage  Luanvraft
        scores = []
        n_samples = len(X)
        fold_size = n_samples // self.n_splits

        for i in range(self.n_splits):
            start = i * fold_size
            end = start + fold_size if i < self.n_splits - 1 else n_samples

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

            score = self._compute_score(reg, X: (X_train, X_test), y: (y_train, y_test))
            scores.append(score)

        return scores
```

```
X = carPrice["Gear box type"].values.reshape(-1, 1)
y = carPrice["Price"].values

kf = KFold(n_splits=6)

reg = LinearRegression()
reg.fit(X, y)

pred = reg.predict(X)

cv_score = kf.cross_val_score(reg, X, y)

print("Cross_validation Score: ", cv_score)
print("Média: ", np.mean(cv_score))
print("Desvio padrão: ", np.std(cv_score))
print(f"RSS: {compute_RSS(pred, y)}")
print(f"MSE: {compute_MSE(pred, y)}")
print(f"RMSE: {compute_RMSE(pred, y)}")
print(f"R^2: {compute_R_squared(pred, y)}")
```

Esses foram os resultados:

```
Cross_validation Score:  [0.009578255973352934, -0.018608123203312843,
 0.018063950687622676, 0.0030037283128486747, 0.015963534039815208, -0
 .00016054540907828319]
Média:  0.004640133400208061
Desvio padrão:  0.012240704642716788
RSS: 698488446447528.0
MSE: 36309634893.56594
RMSE: 190550.87219313887
R^2: 1.000267074265957
```

4. Conclusões

Baseado nas orientações do trabalho e nas aulas, acredito que os resultados das questões foram satisfatórios na medida do possível. Alguns resultados foram muito estranhos mas é possível que realmente deva dar um resultado estranho por conta da construção do dataset, algo que era necessário fazer, que ia além do que as questões pediam e do meu conhecimento, para o resultado ser 100% correto, entre outros. No mais, estou satisfeito com os resultados alcançados.

5. Próximos passos

Não tenho nada em mente, nenhuma sugestão para o projeto.