

## TECHNICAL REPORT

Aluno: José Eric Mesquita Coelho

### 1. Introdução

Este relatório aborda a análise de dois tipos de aprendizado supervisionado: **classificação** e **regressão**, utilizando dois conjuntos de dados distintos (*datasets*).

- **Dataset de Classificação - *star\_classification.csv***

Este conjunto de dados contém informações sobre características de estrelas, galáxias e quasares, fornecendo atributos que ajudam a identificar a que classe cada corpo celeste pertence. O objetivo da análise é construir um modelo de classificação para categorizar os corpos celestes com base nos atributos presentes no dataset.

- **Datasets de Regressão - *tesla.csv* e *ferrari.csv***

Estes datasets apresentam informações sobre preços de venda de ações no mercado das marcas Tesla e Ferrari. O objetivo é construir modelos de regressão para prever o valor em que uma ação fecha o dia.

As análises abordadas neste trabalho buscam explorar técnicas manuais e computacionais de aprendizado de máquina, avaliando a precisão e eficácia dos modelos gerados.

### 2. Observações

Observação 1 - no dataset Ferrari & Tesla: O dataset veio separado em quatro partes distintas, sendo necessário juntá-las para que a análise de regressão possa ser realizada.

Observação 2 - Na questão dois foi necessário aplicar o numpy para que o desempenho e velocidade dos cálculos manuais fossem melhorados.

Observação 3 - Na questão três foi necessário adicionar um argumento para indicar qual dataframe X de treino e teste devem ser utilizados, para o cálculo ideal da melhor normalização.

### 3. Resultados e discussão

*Nesta seção deve-se descrever como foram as resoluções de cada questão. Crie sessões indicando a questão e discuta a implementação e resultados obtidos nesta. Explique o fluxograma do processo de cada questão, indicando quais processamentos são*



*realizados nos dados. Sempre que possível, faça gráficos, mostre imagens, diagramas de blocos para que sua solução seja a mais completa possível. Discuta sempre sobre os números obtidos em busca de motivos de erros e acerto.*

#### a. Questão 1

O processamento dessa questão seguiu os seguintes passos:

**Carregamento do Dataset:** Foi carregado o arquivo [star\\_classification.csv](#), que contém informações sobre 100.000 observações astronômicas.

**Verificação de valores inválidos (Nan's):** Utilizando a função [isnull\(\).sum\(\)](#), para identificar valores ausentes (Nan's). O dataset não apresentou nenhum valor nulo, portanto nenhuma linha foi excluída.

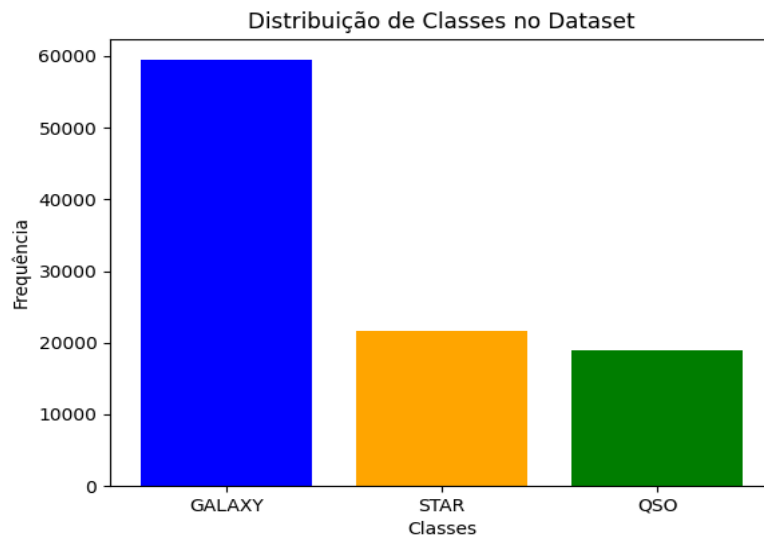
**Seleção de atributos relevantes:** Foram selecionados atributos numéricos relevantes para que assim possa ser feito a classificação do astro como: GALAXY, STAR ou QUASAR. Os atributos u, g, r, i, z representam filtros de cor e luz, redshift apresenta uma forma de identificar a distância do objeto e por fim class identifica a classe do corpo celeste, logo ela é a coluna-alvo da análise.

**Troca de valor das classes:** As classes GALAXY, STAR e QSO tiveram seus valores substituídos por 0, 1 e 2 respectivamente, para facilitar a análise classificatória dos dados no treinamento dos modelos de Machine Learning (ML).

**Salvar o dataset atualizado:** O dataset foi salvo como [star\\_classification\\_ajustado.csv](#), contendo apenas as colunas relevantes e as classes renomeadas.

**Distribuição de classes do dataset:**

Classes	Frequência	%
GALAXY	59445	59,44%
STAR	21594	21,60%
QSO	18961	18,96%



## b. Questão 2

Nessa questão o KNN foi implementado de forma manual, sendo utilizados duas funções para isso, uma para calcular a acurácia, `calcular_acuracia()`, e outra para gerar o modelo KNN, `knn()`. Na função que calcula o KNN, pode se mandar como argumento o tipo de distância que deseja ser calculada, indicando o nome da função do cálculo.

```
def knn(X_train, y_train, nova_amostra, K, dist_func):
    # calculate distances for all training samples
    distances = dist_func(X_train, nova_amostra)

    # get the indices of the K nearest neighbors
    k_indices = np.argsort(distances)[:K]

    # get the labels of the K nearest neighbors
    k_vizinhos = y_train[k_indices]

    # return the most common label
    return np.bincount(k_vizinhos).argmax()

4 usages new *
def calcular_acuracia(X_test, y_test, K, dist_func):
    acertos = 0
    for i in range(len(X_test)):
        classe_prevista = knn(X_train, y_train, X_test[i], K, dist_func)
        if classe_prevista == y_test[i]:
            acertos += 1
    return 100 * acertos / len(X_test)
```

Os resultados para essa questão ao utilizar o KNN com os quatro tipos de funções de distância foram:

```
Distância Euclidiana:  
Porcentagem de acertos: 94.825  
Distância de Manhattan:  
Porcentagem de acertos: 94.995  
Distância de Minkowski com p=3:  
Porcentagem de acertos: 94.645  
Distância de Chebyshev:  
Porcentagem de acertos: 94.32
```

Conforme o que é visto nos resultados (Utilizando K-fold de valor 7) a distância Manhattan apresentou o melhor resultado para o contexto deste dataset, com **94.99%**.

Este resultado pelo fato que Distância Manhattan é menos sensível a grandes discrepâncias em dimensões específicas do espaço vetorial, o que pode ajudar na classificação de objetos com valores extremos em alguns atributos (como filtros de cor).

As distâncias Minkowski e Euclidiana tiveram desempenho próximo a Manhattan. A Chebyshev foi a menos precisa, possivelmente por focar mais em apenas uma dimensão, ignorando outras que são relevantes.

### c. Questão 3

Na questão três foram testadas duas formas de normalização no conjunto de dados do dataset, sendo elas a logarítmica ( $X' = \log(X+1)$ ) e a de Média Zero e Variância Unitária ( $X' = (X - \mu) / \sigma$ ).

A normalização logarítmica faz com que os atributos com variações muito amplas sejam suavizados, o que pode melhorar a performance em datasets com valores discrepantes.

Já a normalização de Média Zero e Variância Unitária garante que todas as variáveis estejam na mesma escala, evitando com que atributos com valores maiores dominem a métrica de distância.

```
# normalização Logarítmica
X_log = X.copy()
X_log[cols_to_normalize] = np.log(X[cols_to_normalize])

# normalização de Média Zero e Variância Unitária
X_zscore = X.copy()
X_zscore[cols_to_normalize] = (X[cols_to_normalize] - X[cols_to_normalize].mean()) / X[cols_to_normalize].std()

X_train_log, X_test_log, y_train, y_test = train_test_split(*arrays: X_log.values, y.values, test_size=0.2, random_state=42)
X_train_zscore, X_test_zscore, _, _ = train_test_split(*arrays: X_zscore.values, y.values, test_size=0.2, random_state=42)
```

A normalização **logarítmica** apresentou a maior acurácia (96.91%), ligeiramente superior à normalização z-score (96.34%). Este resultado sugere que a normalização logarítmica é mais eficaz para o dataset analisado, possivelmente porque suaviza a valores extremos em atributos como filtros de cor.

```
Comparação:
Acurácia Logarítmica: 96.91%
Acurácia Z-Score: 96.34%
```

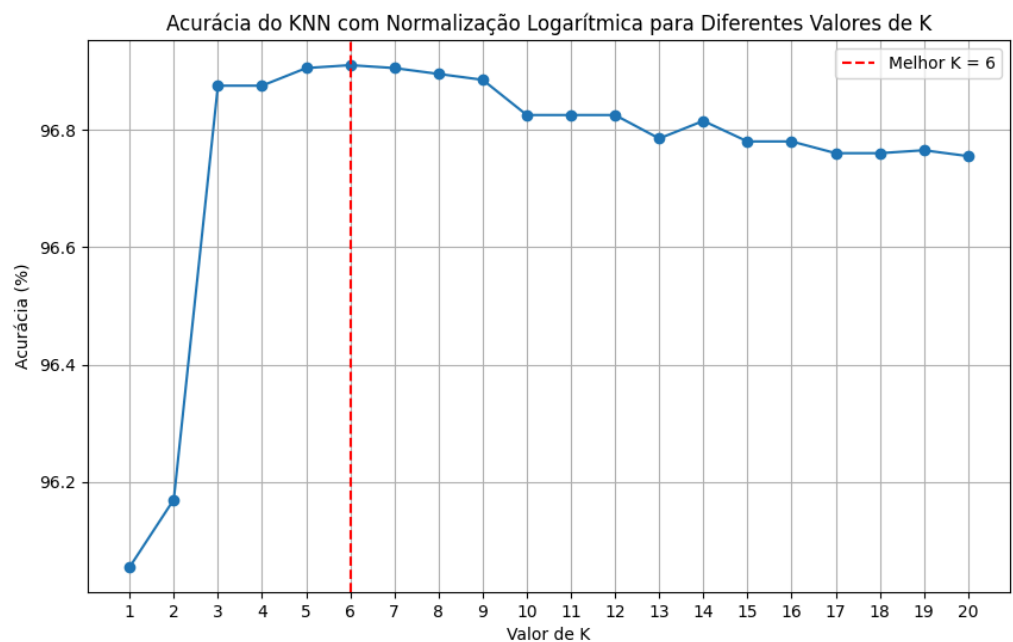
Ambas as normalizações melhoraram a performance geral em relação ao dataset sem normalização (95.15% com Manhattan). A diferença entre os três resultados é pequena, mas isso mostra como o pré-processamento de dados é importante para melhores resultados.

#### d. Questão 4

Segundo os resultados obtidos nesta questão, o melhor valor de **K** seria **6**, com uma acurácia de **96.91%** (Equiparáveis a de K=5 e K=7), apresentando melhor consistência dos resultados. Também é visto pelo gráfico, que a partir do valor 10 a acurácia começou a diminuir, isso ocorre porque incluir mais vizinhos pode introduzir ruído ou considerar amostras menos representativas para a classificação, demonstrando que geralmente um valor alto de K nem sempre é o ideal.



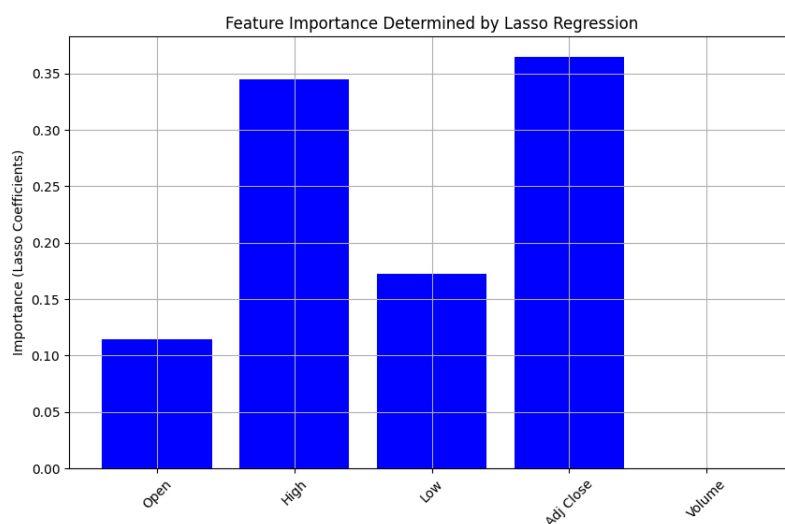
```
Acurácia com K = 1: 96.06%
Acurácia com K = 2: 96.17%
Acurácia com K = 3: 96.88%
Acurácia com K = 4: 96.88%
Acurácia com K = 5: 96.91%
Acurácia com K = 6: 96.91%
Acurácia com K = 7: 96.91%
Acurácia com K = 8: 96.89%
Acurácia com K = 9: 96.89%
Acurácia com K = 10: 96.83%
Acurácia com K = 11: 96.83%
Acurácia com K = 12: 96.83%
Acurácia com K = 13: 96.78%
Acurácia com K = 14: 96.81%
Acurácia com K = 15: 96.78%
Acurácia com K = 16: 96.78%
Acurácia com K = 17: 96.76%
Acurácia com K = 18: 96.76%
Acurácia com K = 19: 96.77%
Acurácia com K = 20: 96.75%
Melhor K: 6
Acurácia com Melhor K: 96.91%
```



**e. Questão 5**

A aplicação do Lasso Regression no dataset combinado identificou os seguintes coeficientes para as variáveis preditoras na tarefa de regressão com o alvo **Close** (variável que representa o preço de fechamento da ação):

Variável	Coeficiente	Importância Relativa
Open	0.114768	Moderada
High	0.345098	Alta
Low	0.172284	Moderada
Adj Close	0.364722	Alta
Volume	0 (Tende a zero)	Irrelevante

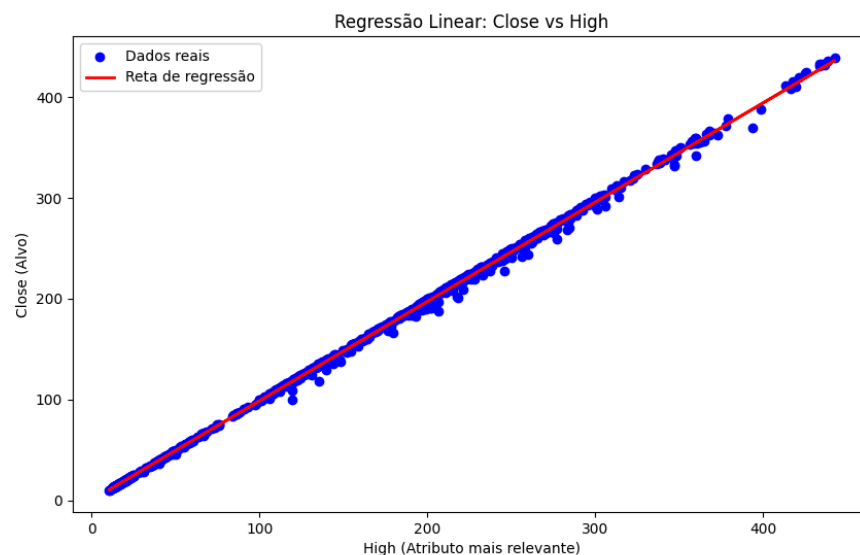


Assim é visto que os atributos mais relevantes são **High** e **Adj Close**. A coluna **Volume** sendo considerada irrelevante, dessa forma ela foi removida do dataset. O Lasso Regression ajudou a identificar as variáveis mais importantes, eliminando ou reduzindo significativamente o peso das menos relevantes. Com base nesses resultados o modelo de regressão pode ser simplificado.

**f. Questão 6**

Ao realizar a regressão linear foi utilizado o atributo **High**, já que **Adj Close** está estritamente associado ao **Close**. O **High** reflete o valor mais alto de uma ação durante o dia, não sendo dependente do atributo alvo.

```
RSS: 1656.72  
MSE: 1.93  
RMSE: 1.39  
R2: 0.9989
```



A partir dos resultados mostrados, pode-se concluir que o modelo de regressão linear conseguiu capturar muito bem a relação entre essas variáveis.

Um valor baixo de **RSS** indica que os resíduos (erros) são pequenos, sugerindo que o modelo ajustou bem os dados.

Para dados financeiros, onde os valores das ações podem variar em escala limitada, um **MSE** baixo é um bom sinal, pois ele é usado para entender a magnitude média dos erros do modelo.

O **RMSE** significa que o erro médio do modelo é aproximadamente 1.39 unidades do preço de fechamento. Esse é um erro muito baixo,



considerando que preços de ações geralmente operam em escalas bem maiores.

O  $R^2$  com valor de 0.9989 indica que a **variação no preço de fechamento** pode ser explicada pelos preços máximos do dia. Isso demonstra uma fortíssima relação linear entre essas variáveis.

No gráfico, a **reta de regressão** ajusta bem a nuvem de pontos, com pouca dispersão ao redor da linha.

Tudo isso mostra que **High** é uma variável relacionada fortemente com a Close.

A normalização com foi testada, porém, piorou os resultados.

```
RSS: 6389.22  
MSE: 7.45  
RMSE: 2.73  
R2: 0.9956
```

Já a normalização com logaritmo melhorou bastante o resultado, mas é necessário analisar se o modelo não está com **overfitting**.

```
RSS: 0.25  
MSE: 0.00  
RMSE: 0.02  
R2: 0.9978
```

#### g. Questão 7

Nessa questão o K Fold e cross-validation são implementados de modo manual.

```
class KFold:
    """
    new *
    def __init__(self, n_splits):
        self.n_splits = n_splits

    1 usage new *
    def cross_val_score(self, model, X, y):
        n_samples = X.shape[0]
        indices = np.arange(n_samples)
        np.random.shuffle(indices)
        fold_size = n_samples // self.n_splits
        scores = []

        for i in range(self.n_splits):
            # Criar os índices de teste e treino
            test_indices = indices[i * fold_size:(i + 1) * fold_size]
            train_indices = np.setdiff1d(indices, test_indices)

            # Dividir em conjuntos de treino e teste
            X_train, X_test = X[train_indices], X[test_indices]
            y_train, y_test = y[train_indices], y[test_indices]

            model.fit(X_train, y_train)

            # Fazer previsões no conjunto de teste
            y_pred = model.predict(X_test)

            # Calcular as métricas para este fold
            RSS = compute_RSS(y_pred, y_test)
            MSE = compute_MSE(y_pred, y_test)
            RMSE = compute_RMSE(y_pred, y_test)
            R_squared = compute_R_squared(y_pred, y_test)

            # Armazenar as métricas
            scores.append((RSS, MSE, RMSE, R_squared))

        return scores
```

O mesmo vale para as métricas RSS, MSE, RMSE e  $R^2$ :

```
def compute_RSS(predictions, y):
    """
    return np.sum(np.square(y - predictions))

2 usages new *
def compute_MSE(predictions, y):
    """
    return np.mean(np.square(y - predictions))

1 usage new *
def compute_RMSE(predictions, y):
    """
    return np.sqrt(compute_MSE(predictions, y))

1 usage new *
def compute_R_squared(predictions, y):
    """
    total_variance = np.sum(np.square(y - np.mean(y)))
    explained_variance = np.sum(np.square(predictions - np.mean(y)))
    return explained_variance / total_variance
```

Com base nas funções acima, os resultados a seguir foram obtidos:

```
Fold 1 - RSS: 3843.74; MSE: 8.98; RMSE: 3.00; R²: 0.9988
-----
Fold 2 - RSS: 2013.14; MSE: 4.70; RMSE: 2.17; R²: 0.9926
-----
Fold 3 - RSS: 3464.46; MSE: 8.09; RMSE: 2.85; R²: 0.9966
-----
Fold 4 - RSS: 4204.76; MSE: 9.82; RMSE: 3.13; R²: 1.0015
-----
Fold 5 - RSS: 2748.12; MSE: 6.42; RMSE: 2.53; R²: 0.9932
-----
Fold 6 - RSS: 3503.77; MSE: 8.19; RMSE: 2.86; R²: 0.9979
-----
Fold 7 - RSS: 5742.83; MSE: 13.42; RMSE: 3.66; R²: 1.0042
-----
Fold 8 - RSS: 3479.65; MSE: 8.13; RMSE: 2.85; R²: 1.0025
-----
Fold 9 - RSS: 3347.59; MSE: 7.82; RMSE: 2.80; R²: 0.9997
-----
Fold 10 - RSS: 4182.61; MSE: 9.77; RMSE: 3.13; R²: 1.0041
-----
Média das Métricas (em todos os folds):
Média RSS: 3653.07
Média MSE: 8.54
Média RMSE: 2.90
Média R²: 0.9991
```

Pode-se notar que em cada Fold obteve bons resultados, porém em alguns  $R^2$  o valor estava próximo ou acima de 1, o que pode indicar overfitting, algo que deve ser testado posteriormente. A consistência nos resultados também indica que o modelo está generalizando bem e que consegue fazer boas previsões.

#### 4. Conclusões

Os resultados apresentados neste relatório demonstram que as técnicas de aprendizado supervisionado, tanto na classificação quanto na regressão, foram eficazes para os datasets (Stellar Classification e Ferrari&Tesla) analisados. No caso do dataset de classificação, o KNN com normalização logarítmica destacou-se, alcançando alta acurácia e evidenciando a importância de um bom pré-processamento. Na análise de



regressão, o uso do Lasso Regression ajudou a identificar variáveis preditoras relevantes, permitindo simplificar os modelos. A regressão linear com a variável High mostrou excelente ajuste, com métricas como  $R^2$  e RMSE indicando forte relação linear e baixos erros. No entanto, alguns indícios de overfitting foram observados, especialmente em análises com normalizações específicas e em métricas elevadas como o  $R^2$  acima de 1 em alguns folds, o que merece atenção em análises futuras. No mais, a maioria dos resultados foram o que se era esperado, pois os datasets já estavam com ótima usabilidade, não contendo Nan's em seus dados e com colunas que já continham informações sobre seu objetivo no Kaggle. Assim a conclusão é que a análise ocorreu de forma tranquila e com bons resultados.

## 5. Próximos passos

O que poderia ser melhorado nos modelos seria uma análise para mitigar possíveis casos de overfitting, especialmente ajustando os hiperparâmetros dos modelos e validando-os com conjuntos de dados separados. No contexto do dataset de classificação, vale experimentar outros métodos de normalização para refinar o modelo. Por fim, no caso dos datasets financeiros, é essencial incluir séries temporais e métricas preditivas baseadas em padrões históricos para validar a robustez do modelo frente a dados futuros, porém isso exigirá implementações técnicas mais complexas.