UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Robert Michael de Ávila Barreira

1. Introdução

Este relatório visa analisar datasets de classificação e regressão aplicando códigos para normalizar, gerar predições e plotar gráficos com base em questões desenvolvidas pelo professor Alisson com o viés de aplicar os conhecimentos aprendidos em sala de aula. Os datasets que foram utilizados para a geração deste relatório foram Wine Quality Dataset e Gold Price Regression ambos do kaggle.

O primeiro dataset está relacionado a variantes tintas do vinho português "Vinho Verde" e descreve a quantidade de diversos compostos químicos presentes no vinho e seu impacto na qualidade. Pode ser usado tanto para tarefas de classificação quanto de regressão, com classes ordenadas, mas desbalanceadas (há mais vinhos normais do que excelentes ou ruins). O objetivo é prever a qualidade do vinho com base nos dados fornecidos.

O segundo dataset é composto por séries temporais de 2010 a 2024, abrangendo índices de mercado, commodities, indicadores econômicos e taxas de câmbio, representados por ETFs e valores financeiros. Seu viés é analisar relações entre ativos financeiros, como o ouro, e a economia global, destacando seu papel como reserva de valor em períodos de incerteza econômica.

2. Observações

Durante o processo, diversos ajustes foram feitos e problemas foram identificados.

O primeiro dataset é simples, mas desafiador, devido ao número reduzido de amostras e ao desbalanceamento das classes. O desafio está em superar essas limitações e construir um modelo preditivo eficaz para classificar a qualidade do vinho.

O segundo dataset apresenta granularidades variadas (diária, mensal e trimestral) e contém valores ausentes, exigindo normalização e tratamento inicial.

Devido a esses imprevistos e limitações, ausências e necessidade de normalização prévia para continuar o processamento aconteceram muitos imprevistos e alguns resultados não foram os esperados.

3. Resultados e discussão

Para a resolução das questões foram seguidas instruções para o pré-processamento e pós-processamento, sempre seguindo a lógica das implementações e resultados obtidos pelas questões anteriores.

Questão 1

Objetivo

Neste primeiro exercício deve realizar manipulação em um dataset com a biblioteca pandas e realizar o pré-processamento deste.

Implementação

Importei o primeiro dataset e verifiquei se existia linhas com NaN com o comando “print(WineQT.isna().sum())”, após essa verificação apliquei um algoritmo para verificar os atributos mais relevantes/importantes para retirar dados desnecessários, depois exibi o dataframe com os atributos selecionados pelo algoritmo e salvei com as alterações aplicadas.

```
# Carregar o dataset
WineQT = pd.read_csv('./dataset/winegt_ajustado.csv')
```

```python
# Calcular correlações com a coluna 'quality'
correlacoes = WineQT.corr()["quality"].sort_values(ascending=False)

# Filtrar colunas relevantes (correlação absoluta > 0.2)
relevantes = correlacoes[correlacoes.abs() > 0.2]

# Gerar automaticamente o DataFrame com as colunas relevantes
colunas_relevantes = ["quality"] + list(relevantes.index[1:])  # Inclui 'quality'
new_wineqt = WineQT[colunas_relevantes]
```

```python
# Exibir o DataFrame final
print("\nDataFrame Final:")
print(new_wineqt.head())
```

Resultado:

1 - Dataset livre de NaN

2 - Um novo dataset com atributos relevantes para o pós-processamento

```
DataFrame Final:
   quality  alcohol  sulphates  citric acid  volatile acidity
0        5      9.4       0.56         0.00              0.70
1        5      9.8       0.68         0.00              0.88
2        5      9.8       0.65         0.04              0.76
3        6      9.8       0.58         0.56              0.28
4        5      9.4       0.56         0.00              0.70
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

Questão 2

Objetivo

Neste segundo exercício deve-se realizar uma classificação utilizando KNN implementado de forma manual.

Implementação

Importei o dataset ajustado na questão anterior e dividi os dados em treino e teste, utilizando fórmula do “train_test_split” com a coluna alvo ‘quality’. Depois de separar os dados em conjunto de treino e teste, preparei os dados e em seguida apliquei a fórmula do KNN. Analisei os resultados gerados pelas diferentes distâncias aplicadas, euclidiana, manhattan, chebyshev e mahalanobis.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Carregar o dataset
WineQT = pd.read_csv('./dataset/wineqt_ajustado.csv')

# Separar os dados em características (X) e rótulos (y)
X = WineQT.drop( labels: "quality", axis=1).values
y = WineQT['quality'].values

# Normalizar os dados (StandardScaler)
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```python
# Funções de distância
def dist_euclidiana(v1, v2):  1 usage  Robertmichaelavila
    return np.sqrt(np.sum((v1 - v2) ** 2))

def dist_manhattan(v1, v2):  1 usage  Robertmichaelavila
    return np.sum(np.abs(v1 - v2))

def dist_chebyshev(v1, v2):  1 usage  Robertmichaelavila
    return np.max(np.abs(v1 - v2))

def dist_mahalanobis(v1, v2, matriz_cov):  2 usages  Robertmichaelavila
    delta = np.array(v1) - np.array(v2)
    inv_cov = np.linalg.inv(matriz_cov)
    return np.sqrt(np.dot(np.dot(delta.T, inv_cov), delta))

# Implementação do KNN
def knn(treinamento, nova_amostra, K, distancia_func, matriz_cov=None):  Robertmichaelavila
    dists = {}
    for i in range(len(treinamento)):
        if matriz_cov is not None and distancia_func == dist_mahalanobis:
            d = distancia_func(treinamento[i][:-1], nova_amostra[:-1], matriz_cov)
        else:
            d = distancia_func(treinamento[i][:-1], nova_amostra[:-1])
        dists[i] = d

    k_vizinhos = sorted(dists, key=dists.get)[:K]
    classes = [treinamento[i][-1] for i in k_vizinhos]
    return max(set(classes), key=classes.count)
```

```python
# Avaliar o KNN
def avaliar_knn(teste, treinamento, K, distancia_func, matriz_cov=None):  4 usage
    predicoes = []
    for amostra in teste:
        classe = knn(treinamento, amostra, K, distancia_func, matriz_cov)
        predicoes.append(classe)
    return accuracy_score([amostra[-1] for amostra in teste], predicoes) * 100

# Matriz de covariância para distância Mahalanobis
matriz_cov = np.cov(X_train.T)

# Avaliar para diferentes distâncias
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))
```

Resultado

Distância Euclidiana: Melhor desempenho, pois é adequada para dados escalonados e distribuições esféricas, capturando bem a similaridade entre amostras.

Distância Manhattan: Desempenho menor, mais robusta a outliers, mas menos eficaz para variáveis correlacionadas.

Distância Chebyshev: Similar à Manhattan, foca na maior diferença entre variáveis, mas não captura bem padrões em dados com várias dimensões relevantes.

Distância Mahalanobis: Alta precisão, considerando a correlação entre variáveis, mas depende da matriz de covariância, que pode ser instável para dados pequenos.

```
Porcentagem de acertos distância euclidiana: 65.73426573426573
Porcentagem de acertos distância manhattan: 61.18881118881119
Porcentagem de acertos distância chebyshev: 61.18881118881119
Porcentagem de acertos distância mahalanobis: 64.33566433566433
```

Questão 3

Objetivo

Considerando a melhor distância observada no exercício anterior, neste deve-se verificar se a normalização interfere nos resultados da classificação.

Implementação

Importei o dataset ajustado e separei o conjunto de dados para treino e teste utilizando a mesma fórmula da questão anterior, após essa etapa normalizei os dados com duas abordagens, a primeira foi normalização logarítmica e a segunda com a média zero e variância unitária. Em seguida, apliquei a fórmula do KNN e exibi a acurácia de ambas normalizações para poder compará-las.

```python
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Carregar o dataset
WineQT = pd.read_csv('./dataset/winegt_ajustado.csv')

# Separando as variáveis independentes (X) e a variável alvo (y)
X = WineQT.drop( labels: "quality", axis=1).values
y = WineQT['quality'].values

# Dividindo os dados em conjuntos de treino e teste com estratificação
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, random_state=42)
```

```python
# Aplicando transformação logarítmica
X_train_log = np.log1p(X_train)
X_test_log = np.log1p(X_test)

# Aplicando normalização com média zero e variância unitária
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)
```

```python
# Função para calcular a acurácia utilizando o KNN com métrica de Mahalanobis
def knn_mahalanobis(X_train, X_test, y_train, y_test, n_neighbors=10):  2 usages  Robertmichaelavila *
    # Calculando a matriz de covariância e sua inversa
    cov_matrix = np.cov(X_train, rowvar=False)
    inv_cov_matrix = np.linalg.inv(cov_matrix)

    # Configurando o KNN com a métrica de Mahalanobis
    knn = KNeighborsClassifier(n_neighbors=n_neighbors, metric='mahalanobis', metric_params={'VI': inv_cov_matrix})

    # Treinando e testando o modelo
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return accuracy_score(y_test, y_pred)

# Avaliando a acurácia com os dados transformados
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)

# Exibindo os resultados
print(f"Acurácia com normalização logarítmica: {accuracy_log * 100:.2f}%")
print(f"Acurácia com média zero e variância unitária: {accuracy_norm * 100:.2f}%")
```

Resultado

A acurácia com normalização logarítmica foi ligeiramente superior à normalização de média zero e variância unitária. A normalização logarítmica ajudou a reduzir a influência de valores extremos e distribuições assimétricas nos dados, resultando em um desempenho ligeiramente melhor para o modelo KNN. A diferença entre os resultados é pequena, sugerindo que ambos os métodos são eficazes, mas o modelo KNN ainda tem margem para melhorias.

```
Acurácia com normalização logarítmica: 67.48%
Acurácia com média zero e variância unitária: 66.78%
```

Questão 4

Objetivo

Com base nas parametrizações vistas anteriormente, neste exercício deve-se buscar saber a melhor parametrização do knn implementado na questão anterior.

Implementação

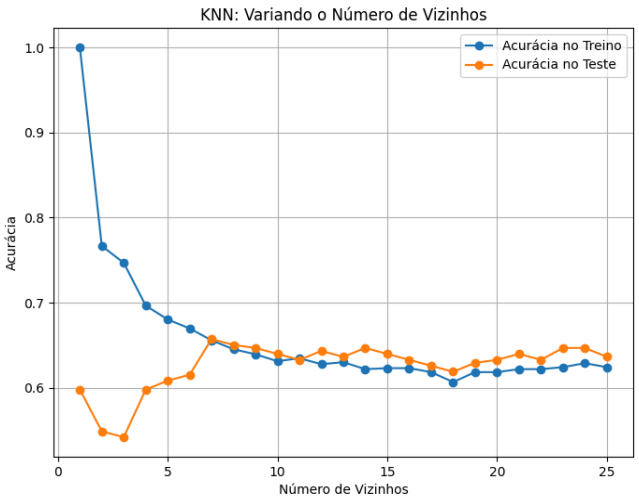
Importei o dataset ajustado e separei o conjunto de dados para treino e teste utilizando a mesma fórmula da questão anterior, após essa etapa normalizei os dados com a média zero e variância unitária, separei o conjunto de testes e treino com o “train_test_split” e apliquei o KNN, em seguida utilizei uma função “for” para encontrar o melhor valor de K e ao final plotei o gráfico com os

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

# Carregar o dataset
WineQT = pd.read_csv('./dataset/wineqt_ajustado.csv')

# Dividir os dados em features (X) e labels (y)
X = WineQT.drop( labels: "quality", axis=1).values
y = WineQT['quality'].values

# Normalizar os dados (média zero e variância unitária)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split( *arrays: X_scaled, y, stratify=y, random_state=42)
```

```python
# Intervalo de vizinhos a ser testado
neighbors_range = np.arange(1, 26)
train_accuracies = []
test_accuracies = []

# Avaliar o desempenho para cada número de vizinhos
for n_neighbors in neighbors_range:
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)

    train_accuracies.append(knn.score(X_train, y_train))
    test_accuracies.append(knn.score(X_test, y_test))

# Exibir os resultados
print("Acurácias no treino:", train_accuracies)
print("Acurácias no teste:", test_accuracies)

# Plotar os resultados
plt.figure(figsize=(8, 6))
plt.plot( *args: neighbors_range, train_accuracies, label="Acurácia no Treino", marker='o')
plt.plot( *args: neighbors_range, test_accuracies, label="Acurácia no Teste", marker='o')
plt.title("KNN: Variando o Número de Vizinhos")
plt.xlabel("Número de Vizinhos")
plt.ylabel("Acurácia")
plt.legend()
plt.grid()
plt.show()
```

Resultado

O gráfico mostra como a acurácia do KNN varia com o número de vizinhos:



1. **Linha Azul (Treino)**: Representa a acurácia no conjunto de treino.
    - Para poucos vizinhos (k pequeno), a acurácia é muito alta, indicando **overfitting** (o modelo se adapta demais aos dados de treino).
    - Conforme o número de vizinhos aumenta, a acurácia no treino diminui, já que o modelo se torna mais generalista.
2. **Linha Laranja (Teste)**: Representa a acurácia no conjunto de teste.
    - Com poucos vizinhos, a acurácia é baixa devido ao **overfitting**.
    - Para valores intermediários de k, a acurácia melhora, atingindo um equilíbrio entre ajuste e generalização.
    - Com valores muito altos de k, a acurácia cai, pois o modelo se torna muito genérico, resultando em **underfitting**.

Questão 5

Objetivo

Observar o dataset de regressão e realizar o pré-processamento. Verifique qual atributo será o alvo para regressão no seu dataset e faça uma análise de qual atributo é mais relevante para realizar a regressão do alvo escolhido. Lembre de comprovar via gráfico. Caso necessário remova colunas que são insignificantes, valores NaN também devem ser removidos.

Implementação

Importei o segundo dataset, destinado para regressão, e comecei a fazer o pré-processamento, retirei os NaN que tinham, e criei outro dataframe sem a coluna date, pois ela estava impossibilitando a obtenção dos resultados pois ela é do tipo string e estava tentando ser convertido para float. Após isso, usei um algoritmo para calcular as correlações entre a matrizes e selecionei possiveis candidatos a variáveis alvo para regressão, depois dessa análise defini a coluna “sp500 open” como target, calculei a correlação com o alvo, ordenando por magnitude, e encontrei o atributo mais relevante, ao final plotei o gráfico com a relação entre a variável mais relevante e o alvo e salvei o dataset com os ajustes aplicados.

```
# Carregar o dataset
Regression = pd.read_csv('./dataset/financial_regression.csv')

# Exibir as primeiras linhas do dataset
print("Primeiras linhas do dataset:")
print(Regression.head())

# Excluindo linhas com NaN
regression = Regression.dropna(axis=0)

regression_drop_date = regression.drop( labels: "date", axis=1)
```

```python
# Calcular a matriz de correlação
corr_matrix = regression_drop_date.corr()
print("\nMatriz de Correlação:")
print(corr_matrix)

# Selecionar possíveis candidatos a variáveis alvo para regressão
target_candidates = [
    col for col in regression_drop_date.columns
    if regression_drop_date[col].dtype in [np.float64, np.int64] and len(regression_drop_date[col].unique()) > 10
]
print("\nCandidatos para alvo (regressão):", target_candidates)
```

```python
# Definir a variável alvo
target = "sp500 open"

# Calcular a correlação com o alvo, ordenando por magnitude
correlations = corr_matrix[target].drop(target).sort_values(key=abs, ascending=False)
print(f"\nCorrelação com o alvo ({target}):")
print(correlations)

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

```python
# Visualizar a relação entre a variável mais relevante e o alvo
plt.figure(figsize=(8, 6))
sns.scatterplot(x=regression_drop_date[most_relevant], y=regression_drop_date[target], alpha=0.7)
plt.title(f'Relação entre {most_relevant} e {target}')
plt.xlabel(most_relevant)
plt.ylabel(target)
plt.grid()
plt.show()

regression_drop_date.to_csv( path_or_buf: './dataset/Regression_ajustado.csv', index=False)
print("As alterações foram salvas no arquivo 'Regression_ajustado.csv'.")
```

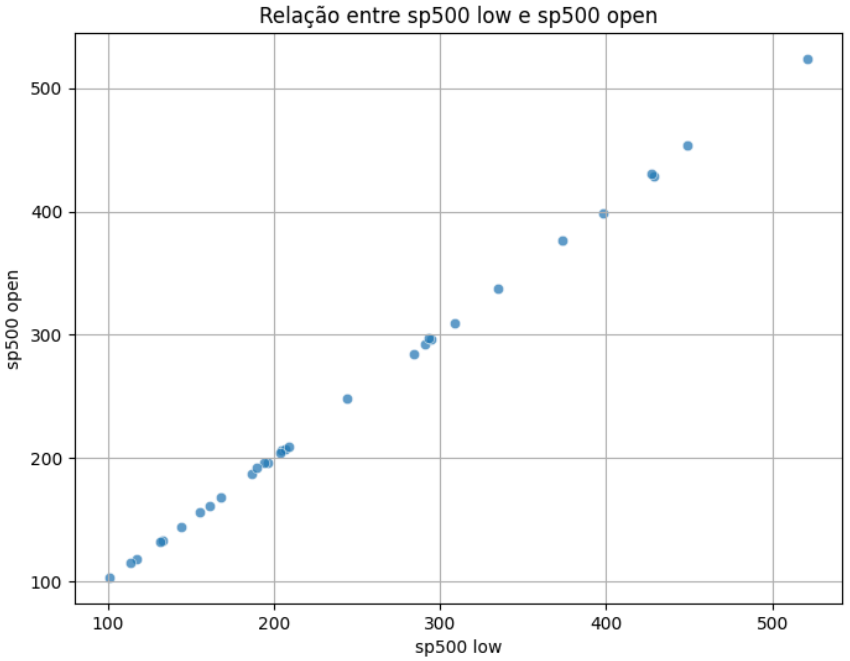
Resultados

Como resultado obtive o gráfico personalizado indicando a variável alvo que eu poderia utilizar durante as próximas questões.

```
Matriz de Correlação:
                  sp500 open  sp500 high  ...  gold close  gold volume
sp500 open          1.000000    0.999856  ...    0.699705    -0.262232
sp500 high          0.999856    1.000000  ...    0.701180    -0.262002
sp500 low           0.999938    0.999756  ...    0.699721    -0.263504
sp500 close         0.999886    0.999821  ...    0.699622    -0.262678
sp500 volume       -0.583248   -0.580379  ...   -0.347804     0.648320
sp500 high-low      0.401877    0.415346  ...    0.344335    -0.044281
nasdaq open         0.990472    0.990991  ...    0.752061    -0.209302
nasdaq high         0.990240    0.990848  ...    0.752980    -0.208599
nasdaq low          0.990498    0.990969  ...    0.752277    -0.209590
nasdaq close        0.990285    0.990824  ...    0.752352    -0.209296
nasdaq volume      -0.235395   -0.234377  ...   -0.047486     0.663175
nasdaq high-low     0.712847    0.721886  ...    0.598054    -0.091873
us_rates_%          0.507333    0.503499  ...    0.301440     0.047103
CPI                 0.965223    0.966570  ...    0.728289    -0.214149
usd_chf            -0.207110   -0.207559  ...   -0.419089     0.309040
eur_usd            -0.646247   -0.648113  ...   -0.243484     0.209009
```

```
Correlação com o alvo (sp500 open):
sp500 low              0.999938
sp500 close            0.999886
sp500 high             0.999856
nasdaq low             0.990498
nasdaq open            0.990472
nasdaq close           0.990285
nasdaq high            0.990240
GDP                    0.982726
CPI                    0.965223
palladium close        0.765915
palladium low          0.763576
palladium open         0.763397
palladium high         0.763388
nasdaq high-low        0.712847
silver volume          0.708252
platinum high         -0.708121
```

```
Atributo mais relevante para prever 'sp500 open': sp500 low
As alterações foram salvas no arquivo 'Regression_ajustado.csv'.
```



Questão 6

Objetivo

Utilizando o atributo mais relevante calculado na questão 5, implemente uma regressão linear utilizando somente este atributo mais relevante, para predição do atributo alvo determinado na questão 5 também. Mostre o gráfico da reta de regressão em conjunto com a nuvem de atributo. Determine também os valores: RSS, MSE, RMSE e R_squared para esta regressão baseada somente no atributo mais relevante.

Implementação

Importei o dataset utilizado na questão anterior e ajustei as dimensões para utilizar na fórmula “train_test_spli” para dividir os conjuntos de dados em treino e teste. Apliquei uma função para treinar o modelo de regressão linear e em seguida uma função para fazer as predições. Ao final, plotei o gráfico com os dados da linha de regresso e calculei as métricas de RSS, MSE, RMSE e R_squared.

```python
# Carregar o dataset
Regression = pd.read_csv('./dataset/Regression_ajustado.csv')

# Ajustar as dimensões de X e y
X = Regression["sp500 open"].values.reshape(-1, 1)
y = Regression["sp500 low"].values

# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
```

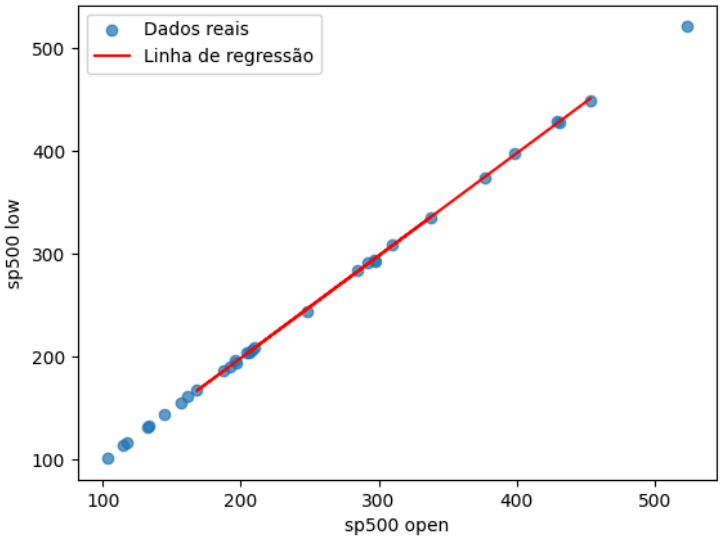
```python
# Treinar o modelo de regressão linear
reg = LinearRegression()
reg.fit(X_train, y_train)

# Fazer predições
y_pred = reg.predict(X_test)

# Visualizar os dados e a linha de regressão
plt.scatter(X, y, alpha=0.7, label="Dados reais")
plt.plot( *args: X_test, y_pred, color="red", label="Linha de regressão")
plt.xlabel("sp500 open")
plt.ylabel("sp500 low")
plt.legend()
plt.show()
```

```python
# Calcular métricas
r_squared = reg.score(X_test, y_test)
rss = np.sum(np.square(y_test - reg.predict(X_test)))
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Exibir os resultados
print(f"RSS: {rss:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R_Squared2: {r_squared:.4f}")
```

Resultados

Como resultado, obtive o gráfico que indica uma linha de regressão muito boa além do valor de cada métrica calculada.

```
RSS: 6.9307
MSE: 1.1551
RMSE: 1.0748
R_Squared2: 0.9999
```



Questão 7

Objetivo

Utilizando kfold e cross-validation faça uma regressão linear. Utilize uma implementação manual do kfold e cross-validation. calcule as métricas RSS, MSE, RMSE e R_squared que também devem ser implementadas manualmente.

Implementação

Primeiro implementei as funções das métricas RSS, MSE, RMSE e R_squared de forma manual assim como a do Kfold, em seguida importei o dataset de regressão

ajustado e ajustei as dimensões para em seguida aplicar a regressão linear. Depois fiz a predição e apliquei o cross-validation. Ao final exibi os resultados da validação, média e desvio padrão e os valores de das métricas RSS, MSE, RMSE e R_squared.

Resultado

Os valores altos de Cv Score indicam que o modelo Linear Regression ajusta bem os dados em todas as partições do K-Fold. Além disso, o desvio padrão é pequeno, o que sugere que o desempenho do modelo é consistente em diferentes folds. Isso é um ótimo sinal.

```
Cv Score:  [0.9965722251822569, 0.9966969625405737, 0.9713835414469871, 0.9993160193475991, 0.9982212618543306, 0.9990184364459828]
Média:  0.9935347411362884
Desvio Padrão:  0.009961251340915219


RSS: 46.524155877438304
MSE: 1.55080519591461
RMSE: 1.2453132922741208
R_Squared2: 8024.17522802828
```

4. Conclusões

O primeiro dataset de classificação não gerou resultados tão promissores como o segundo, não é estranho já que a aplicação para ambas foram diferentes, mas o segundo dataset teve um melhor desempenho dentro do que foi proposto durante os processos.

5. Próximos passos

Seria interessante alimentar os datasets com informações mais precisas e significativas, determinando os atributos realmente necessários para a manipulação.