UNIVERSIDADE FEDERAL DO CEARÁ

# TECHNICAL REPORT

Aluno: Esdras Souza dos Santos

## 1. Introdução

*Classificação: Gender Classification Dataset*

Gender Classification Dataset é um dataset simples com 7 colunas com dados gerados.

As colunas deste dataset são:

**long_hair** - Esta coluna contém 0's e 1's onde 1 é "cabelo longo" e 0 é "cabelo não longo".

**cheek_width_cm** - Esta coluna está em centímetros, e é a largura da testa.

**cheek_height_cm** - Esta é a altura da testa e está em centímetros.

**nose_wide** - Esta coluna contém 0's e 1's onde 1 é "nariz largo" e 0 é "nariz não largo".

**nose_long** - Esta coluna contém 0's e 1's onde 1 é "nariz longo" e 0 é "nariz não longo".

**lips_thin** - Esta coluna contém 0's e 1's onde 1 representa os "lábios finos" enquanto 0 é "lábios não finos".

**distance_nose_to_lip_long** - Esta coluna contém 0's e 1's onde 1 representa a "longa distância entre o nariz e os lábios" enquanto 0 é a "curta distância entre o nariz e os lábios".

**gender** - Pode ser "Masculino" ou "Feminino".

Nesse dataset o objetivo é descobrir se o gênero é masculino ou feminino com base nos dados fictícios que estão no dataset.

*Regressão: House Sales in King County, USA*

House Sales in King County, USA é um dataset antigo em que se tem um conjunto de dados que contém preços de venda de casas para King County. Ele inclui casas vendidas entre maio de 2014 e maio de 2015.

Tem várias colunas sendo algumas delas:

**Date:** data de quando as casas foram vendidas (eu acho).

**Bedrooms:** Quantidade de quartos.

**Bathroom:** Quantidade de banheiros da casa.

**sqft_living:** Tamanho em quadrados da casa.

**Floors:** Andares da casa.

**Price:** Preço pela qual a casa foi vendida.

O dado mais importante é o **Price** pois é o dado mais interessante para se tentar adivinhar com a regressão.

É um ótimo conjunto de dados para avaliar modelos de regressão simples.

## 2. Observações

Adicionei uma cópia das funções de carregamento dos dataset dentro do arquivo src.utils

A questão 2 está demorando muito para terminar de funcionar, provavelmente é por causa que tem que coletar e verificar todos os valores das colunas do dataset.

## 3. Resultados e discussão

### 3.1 Questão 1

A primeira questão foi a mais simples porque era apenas para analisar o dataset e ver se ele precisava de alguma mudanças. Por sorte os meus dataset não vieram precisando de nenhuma mudança grande.

Primeiro comecei carregando os datasets para analisá-los

```python
def load_gender_classification():  2 usages  Esdras S
    return pd.read_csv('dataset/gender_classification_v7.csv')

def load_kc_house_data():  2 usages  Esdras S
    return pd.read_csv('dataset/kc_house_data.csv')
```

```python
dataclassification = load_gender_classification()
dataRegressao = load_kc_house_data()
```

e comecei printando os dois dataset com ".isna().sum()" e ".isnull().sum()" para verificar se tinha células vazia e nula e aproveitei para ver os nomes das colunas.

```python
#Printes verificando se tem NA ou null, Não tem.
print("Verificando se tem NA no dataset de regressão: ",dataRegressao.isna().sum(),'\n')
print("Verificando se tem NA no dataset de classificação: ",dataclassification.isna().sum(),'\n')
print("Verificando se tem Null no dataset de regressão: ",dataRegressao.isnull().sum(),'\n')
print("Verificando se tem Null no dataset de classificação: ",dataclassification.isnull().sum(),'\n')
```

Não foi encontrada nenhuma célula vazia ou nula.

Depois printei o tipo dos dados, para verificar se tinha string ou outras coisa que poderiam atrapalhar no processamentos.

```python
print("Tipos de Dados do dataset de classificação: ",dataclassification.dtypes)
print("Tipos de Dados do dataset de Regressão: ", dataRegressao.dtypes)
```

Do dataset de regressão, decidir remover as colunas id (pois é inútil analisar isso), data(pois era tipo Object e desnecessário), zipcode(ou código-postal), lat e long(desnecessário).

E do dataset de Classificação decidir transformar a coluna gender, que tinha os valores "Male" e "Female" e transformei em "1" e "0" para garantir que não teria nenhum problema depois.

```python
def load_new_dataframe_kc_house():  1 usage  Esdras S
    return load_kc_house_data().drop( labels: ['long', 'lat', 'zipcode', 'date', 'id'], axis=1)

def load_new_dataframe_gender_classification():  1 usage  Esdras S
    data = load_gender_classification()
    data["gender"] = data["gender"].map({'Male':1 ,'Female':0})
    return data
```

e depois verifique se tudo estava certo para terminar essa questão.

```
newDataRegressao = load_new_dataframe_kc_house()
newDataClassification = load_new_dataframe_gender_classification()
```

```
#print para verificar se ta tudo certo
print("Dataset classificação depois das mudanças: ",newDataClassification.dtypes)
print("Dataset Regressão depois das mudanças: ",newDataRegressao.value_counts())
```

### 3.2 Questão 2

Nesta questão comecei importando o dataset e depois comecei a pensar como fazer o knn manualmente.

```
import numpy as np
from collections import Counter
from sklearn.metrics import accuracy_score
from src.utils import load_gender_classification
from sklearn.model_selection import train_test_split

genderClassification_df = load_gender_classification()
```

Como não estava entendendo como fazer de um jeito bom o knn. pedi ajuda ao chat e modifique o código que ele me deu de exemplo.

Mas vamos começar do início do código, o Código iniciar depois das funções, já iniciando as variáveis X e y.

```
X = genderClassification_df[['long_hair', 'forehead_width_cm', 'forehead_height_cm',
                             'nose_wide', 'nose_long', 'lips_thin', 'distance_nose_to_lip_long']].values
y = genderClassification_df['gender'].values
```

iniciei assim pois quando iniciava o X apenas com o .drop(gender) ele não funcionava, porque o dataset, a primeira linha dele é a linha com os nomes das colunas, e como eu não conseguir pensar em um jeito de remover aquilo sem mexer diretamente no dataset. Eu decidi adicionar os valores das variáveis menos a do “gender”, e me pareceu que deu certo.

Logo em seguida dividir o dataset.

```
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, random_state=30)
```

Depois criei um dicionário e um for em que tem os nomes dos cálculos para a hora dos prints, dentro do for tem uma variável que já recebe o score do knn de um cálculo. Na função é enviado “X_train”, “X_test”, “y_train”, “y_test”, “k=x”, “distance_func=func” em que o “k” pode ser escolhido o número de vizinhos e em que “func” é o nome do cálculo que vai ser calculado no knn. e depois de calcular é adicionada no dicionário.

```
results = {}
for name, func in [
    ("Euclidiana", euclidean_distance),
    ("Manhattan", manhattan_distance),
    ("Chebyshev", chebyshev_distance),
]:
    acc = evaluate_knn(X_train, y_train, X_test, y_test, k=7, distance_func=func)
    results[name] = acc
```

Vamos para o evaluate_knn. O evaluate_knn recebe o treino, o teste, o número de vizinho e o nome do cálculo além do cov_inv que é um número usado para calcular a distância Mahalanobis.

```
def evaluate_knn(X_train, y_train, X_test, y_test, k, distance_func, cov_inv=None):
    y_pred = knn_manual(X_train, y_train, X_test, k, distance_func, cov_inv)
    return accuracy_score(y_test, y_pred)
```

A função chama o knn manual e recebe o predictions e logo em seguida é calculado e retornado o accuracy score.

O Knn Manual.

```
def knn_manual(X_train, y_train, X_test, k=3, distance_func=euclidean_distance, cov_inv=None):
    predictions = []
    for test_point in X_test:
        # Calcular a distância entre o ponto de teste e todos os pontos de treinamento
        distances = [
            distance_func(test_point, train_point) if cov_inv is None
            else mahalanobis_distance(test_point, train_point, cov_inv)
            for train_point in X_train
        ]
        # Combinar as distâncias com os rótulos
        neighbors = sorted(zip(distances, y_train))[:k]
        # Extrair os rótulos dos K vizinhos mais próximos
        k_nearest_labels = [label for _, label in neighbors]
        # Determinar a classe predominante
        most_common = Counter(k_nearest_labels).most_common(1)[0][0]
        predictions.append(most_common)
    return predictions
```

O knn manual recebe o treino e o teste do X, o treino do y, o número de vizinho e o nome do cálculo e o cov.

Ele começa criando uma lista onde armazenar os predictions, e logo em seguida cria um for em que test_point será um número do X_test. e começa entrando em um for e calculando a distância escolhida em que o nome que é recebido é substituir o nome da função. se o knn tiver algum número no cov_inv ele chama automaticamente o cálculo do mahalanobis. e depois é só cálculos (com ajuda do GPT). e no final retorna os predictions.

```python
# Função para calcular a distância Euclidiana
def euclidean_distance(point1, point2):  2 usages  Esdras S
    return np.sqrt(np.sum((point1 - point2) ** 2))

def manhattan_distance(point1, point2):  1 usage  Esdras S
    return np.sum(np.abs(point1 - point2))

def chebyshev_distance(point1, point2):  1 usage  Esdras S
    return np.max(np.abs(point1 - point2))

def mahalanobis_distance(point1, point2, cov_inv):  2 usages  Esdras S
    diff = point1 - point2
    return np.sqrt(diff.T @ cov_inv @ diff)
```

```python
for name, func in [
    ("Euclidiana", euclidean_distance),
    ("Manhattan", manhattan_distance),
    ("Chebyshev", chebyshev_distance),
]:
    acc = evaluate_knn(X_train, y_train, X_test, y_test, k=7, distance_func=func)
    results[name] = acc

# Avaliando a distância de Mahalanobis
cov_matrix = np.cov(X_train, rowvar=False)
cov_inv = np.linalg.inv(cov_matrix)
results["Mahalanobis"] = evaluate_knn(X_train, y_train, X_test, y_test, k=7, distance_func=mahalanobis_distance, cov_inv=cov_inv)

# Exibindo os resultados
for name, acc in results.items():
    print(f"Distância {name}: Acurácia = {acc:.4f}")
```

no final depois de calcular todas as distâncias pedidas é printado o nome e a Acurácia das distâncias.

```
Distância Euclidiana: Acurácia = 0.9768
Distância Manhattan: Acurácia = 0.9720
Distância Chebyshev: Acurácia = 0.9616
Distância Mahalanobis: Acurácia = 0.9720
```

A melhor distância foi a Euclidiana.

### 3.3 Questão 3

Agora devemos usar a melhor distância e normalizar e verificar os resultados.

Comecei importando o dataset Igual antes de antes.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from src.utils import load_new_dataframe_gender_classification
import numpy as np

gender = load_new_dataframe_gender_classification()
```

e inicio o StandartScaler para fazer a normalização. e logo em seguida inicio o X e normalizo logaritmicamente e em média zero.

```
scaler = StandardScaler()

X = gender.drop(['gender'], axis=1)

offset = 1e-6
X_norm = scaler.fit_transform(X)
X_log = np.log(X + offset)
```

Início o y.  e divido o dataset com train_test_split em padrão, normalizado e logarítmica, Início o Knn e printei os resultados um abaixo do outro.

```
y = gender['gender'].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, random_state=42)
X_train_norm, X_test_norm, y_train, y_test = train_test_split( *arrays: X_norm, y, stratify=y, random_state=42)
X_train_log, X_test_log, y_train, y_test = train_test_split( *arrays: X_log, y, stratify=y, random_state=42)

knn = KNeighborsClassifier()

knn.fit(X_train,y_train)

print('score padrão', knn.score(X_test, y_test))

knn.fit(X_train_norm,y_train)

print('\nscore normalizado', knn.score(X_test_norm, y_test))

knn.fit(X_train_log,y_train)

print('\nscore log', knn.score(X_test_log, y_test))
```

```
score padrão 0.9624300559552358

score normalizado 0.9680255795363709

score log 0.9680255795363709
```

### 3.4 Questão 4

Nessa atividade eu deveria buscar saber a melhor parametrização do knn implementado na questão anterior.

Importes:

```
from src.utils import load_new_dataframe_gender_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

scaler = StandardScaler()
gender_df = load_new_dataframe_gender_classification()
```

Comecei iniciando o ‘X’ eo ‘y’ e Normalizando o ‘X’ com a melhor normalização e separei o dataset em treino e os testes com o train_test_split.

```
X = gender_df.drop(["gender"], axis=1).values
y = gender_df["gender"].values

X_norm = scaler.fit_transform(X)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split( *arrays: X_norm, y, test_size=0.25, random_state=42, stratify=y)
```

criei uma variável numerada de 1 a 15, para verificar os vizinhos de 1 a 15. E mais duas variáveis com dicionários vazios para armazenar as acurácias de treino e teste.

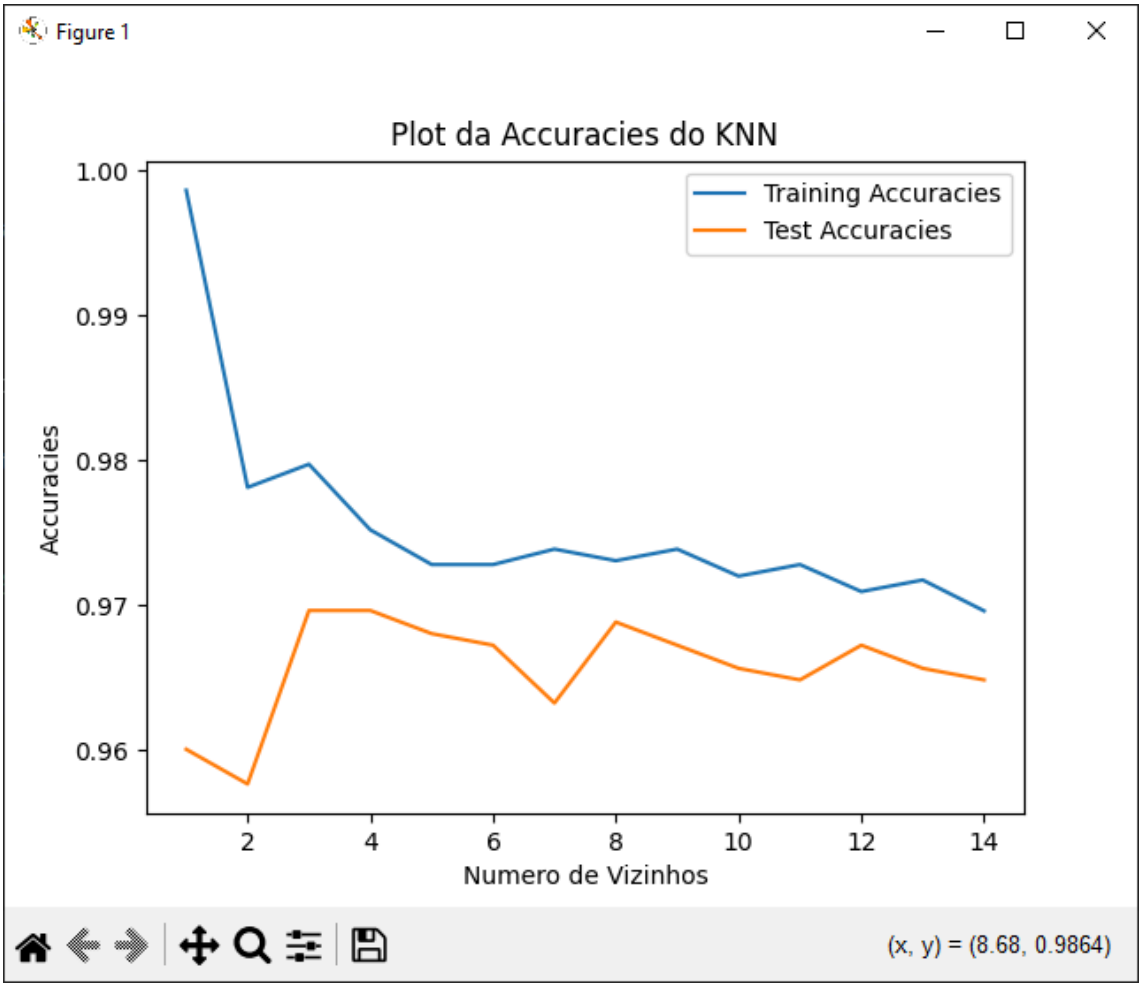
```
neighbors = np.arange(1, 15)
train_accuracies = {}
test_accuracies = {}
```

Depois vem um for para verificar a acurácia de cada ponto (de 1 a 15) usando o knn e depois armazenada nos dicionários. Depois um print dos dicionários.

```
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)

    knn.fit(X_train, y_train)

    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

print("acuracy on train: ",train_accuracies, '\n',"acuracy on test: ", test_accuracies)
```

Depois é só plotar o gráfico para verificar melhor os pontos.

```
print("acuracy on train: ",train_accuracies, '\n',"acuracy on test: ", test_accuracies)

plt.title("Plot da Accuracies do KNN")

plt.plot( *args: neighbors, train_accuracies.values(), label="Training Accuracies")

plt.plot( *args: neighbors, test_accuracies.values(), label="Test Accuracies")

plt.legend()
plt.xlabel("Numero de Vizinhos")
plt.ylabel("Accuracies")

plt.show()
```




```
acuracy on train:  {1: 0.9986666666666667, 2: 0.9781333333333333, 3: 0.97
 acuracy on test:  {1: 0.9600319744204636, 2: 0.9576338928856915, 3: 0.96
```

Analisando o gráfico o melhor ponto seria o 12 pois a variância entre o treino e o teste foi menor.

### 3.5 Questão 5

Agora vamos começar a usar o dataset de Regressão. Primeiro vamos decidir qual o atributo alvo e o atributo mais relevante para realizar a regressão do alvo escolhido.

O atributo alvo escolhido foi “price” e para decidir o atributo alvo usei um gráfico de correlação recomendado pelo Chat.

Começando com as importações.

```
from src.utils import load_new_dataframe_kc_house
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso

houses = load_new_dataframe_kc_house()
```

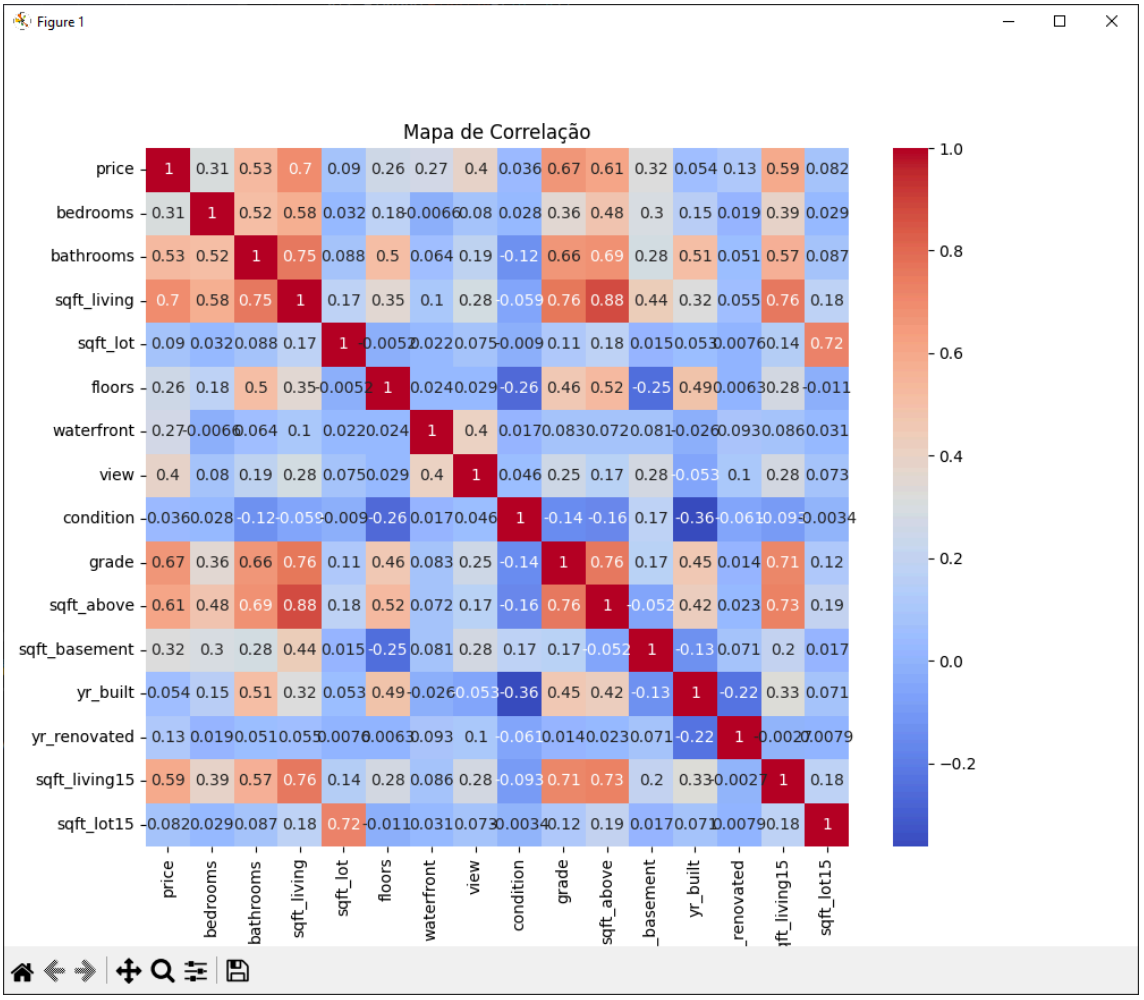
Inicializei as variáveis ‘X’ e ‘y’ com os valores alvo (que seria o ‘price’)

```
X = houses.drop(["price"], axis=1)
y = houses["price"].values
```

Agora inicializo o gráfico de correlação e ploto ele com o estilo de coolwarm.

```
#Grafico de correlação
correlation = houses.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title("Mapa de Correlação")
plt.show()
```

O gráfico fica Assim:




Esse gráfico funciona mostrando a correlação entre os atributos quando maior mais relevante ele é.

Olhando o gráfico foi possível decidir quais atributos são irrelevantes e qual é mais relevante.

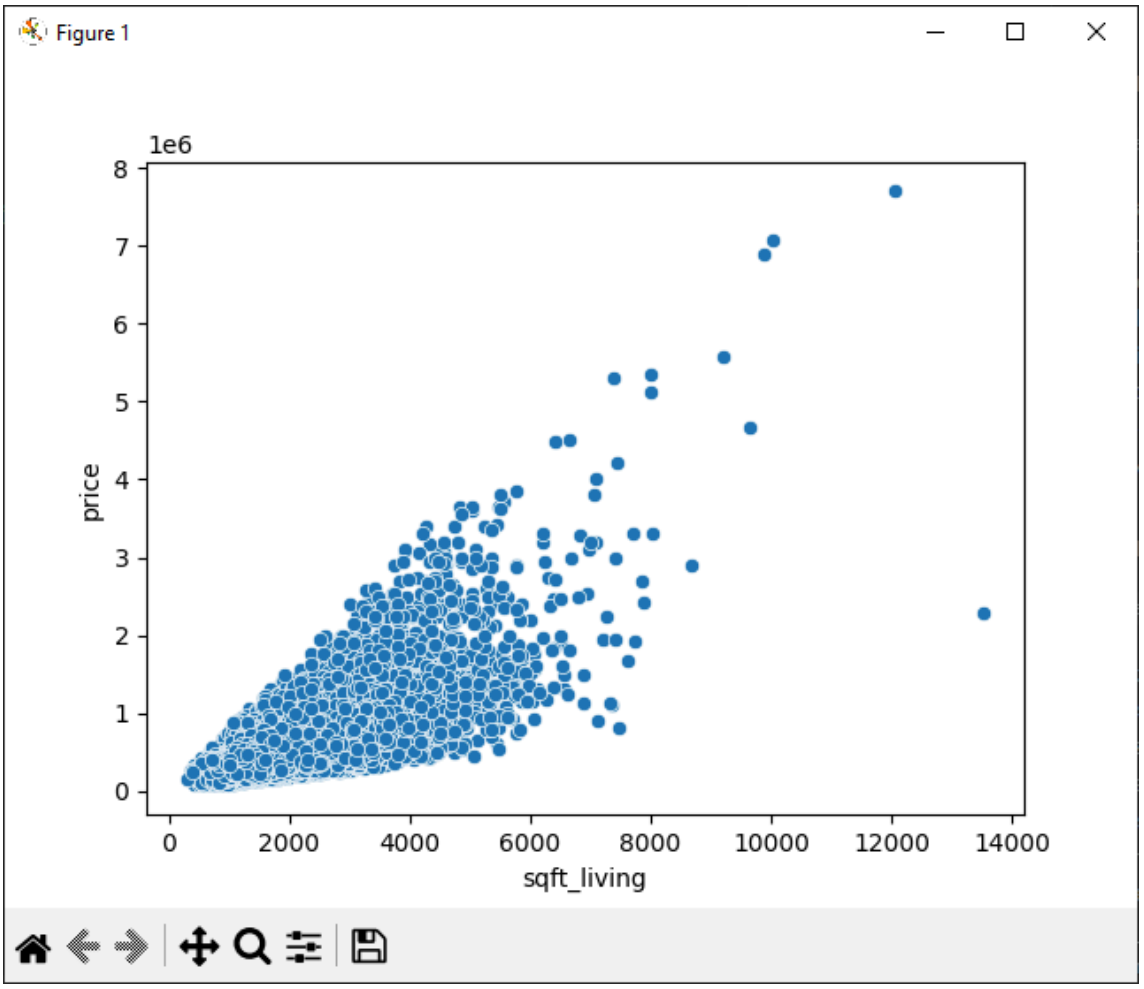
Usando o gráfico removi os atributos 'sqft_lot', 'sqft_lot15', 'yr_built', 'condition' que são os mais irrelevantes e decidir que o atributo mais importante é o ‘sqft_living’ que até faz sentido, porque é a ‘Área interna da casa em pés quadrados’.

```
houses = houses.drop(['sqft_lot', 'sqft_lot15', 'yr_built', 'condition'], axis=1)
```

Depois plotei o normalmente usando o seaborn para facilitar um pouco.

```
sns.scatterplot(x='sqft_living', y='price', data=houses)
plt.show()
```

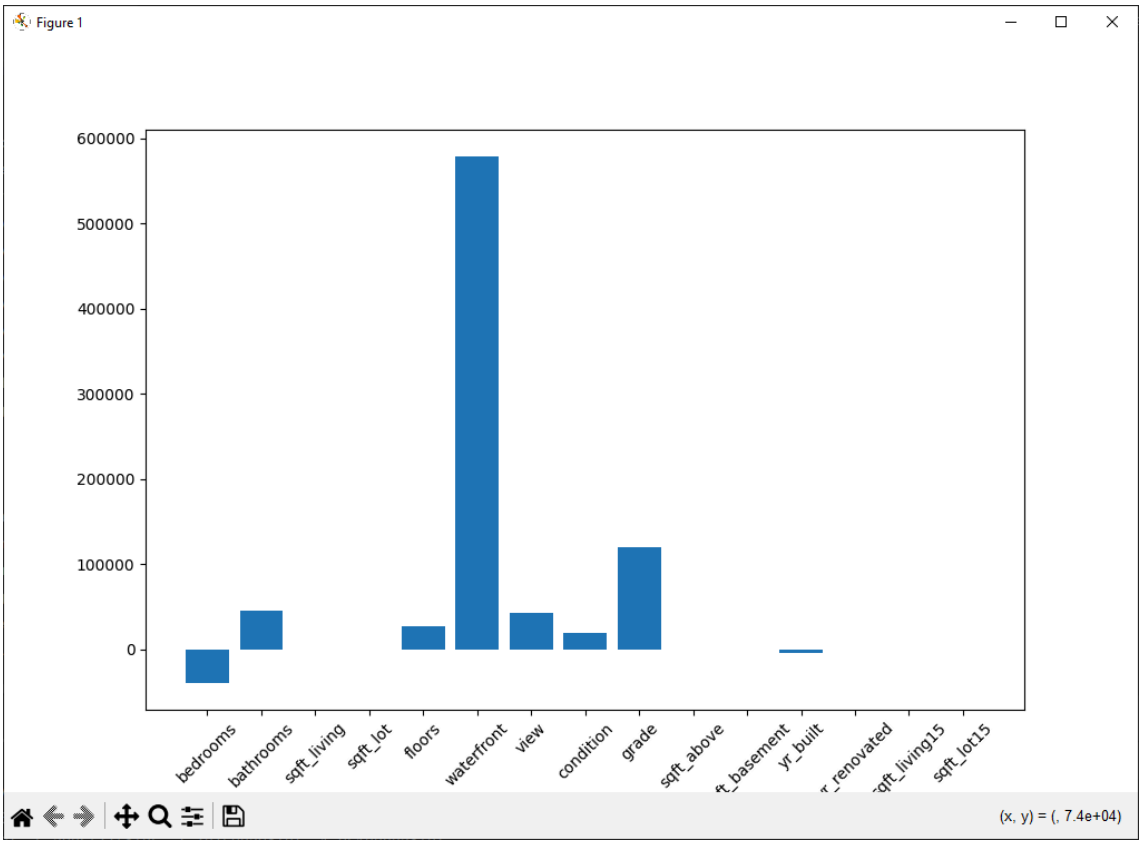
O gráfico ficou assim:




Depois eu estava olhando as atividades e vi que tinham um gráfico chamado LASSO mas olhando o gráfico não entendi direito pra que ele serve.

Ele fica assim:

```
sales_columns = X.columns

# Instantiate a lasso regression model
lasso = Lasso(alpha=0.1)

# Compute and print the coefficients
lasso_coef = lasso.fit(X, y).coef_
print(lasso_coef)
plt.bar(sales_columns, lasso_coef)
plt.xticks(rotation=45)
plt.show()
```




pensei que poderia ser relevância mais waterfront é uma variável de true e false (0 e 1) e a reta da próxima questão não ficaria certo.

### 3.6 Questão 6

Nessa questão deve-se usar o atributo mais importante decidido na questão anterior e fazer uma linha de predição e depois determinar os valores de RSS, MSE, RMSE e R Square.

Começando pelas importações:

```
from sklearn.metrics import mean_squared_error
import numpy as np
from src.utils import load_new_dataframe_kc_house
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
house_df = load_new_dataframe_kc_house()
```

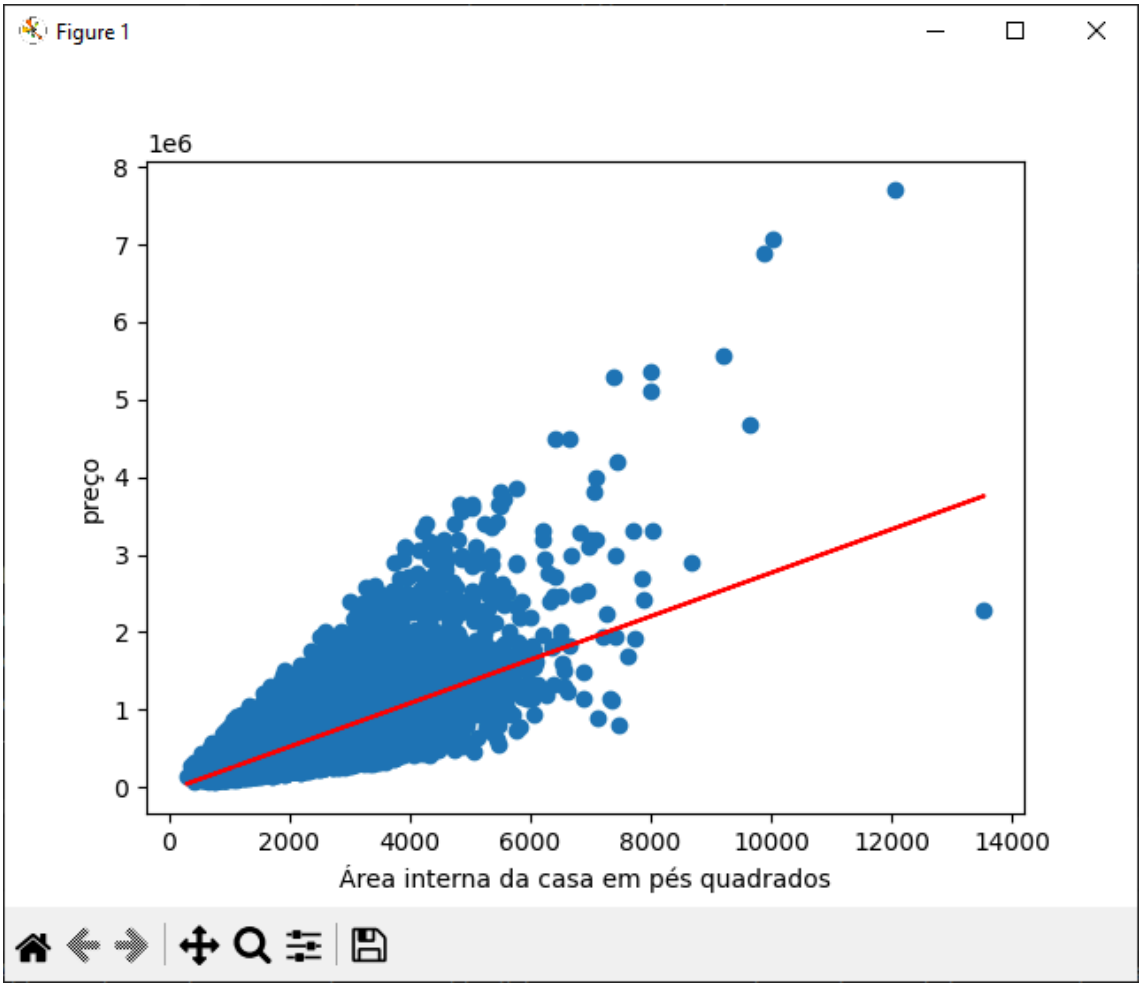
Iniciei a variável ‘X’ e o ‘y’ e já separei em treino e teste.

```
X = house_df["sqft_living"].values.reshape(-1,1)
y = house_df["price"].values

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.4, random_state=47)
```

Logo em seguida, iniciei a Linear Regression, dei o fit e coletei o predict para poder plotar o gráfico com a linha de predição.

```
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("Área interna da casa em pés quadrados")
plt.ylabel("preço")
plt.show()
```

Ficando assim o gráfico:




Logo depois calculando rapidamente os valores de RSS, MSE, RMSE, RSquare

```
#R-squared
r_squared = reg.score(X_test, y_test)

#RSS
rss = np.square(y-y_pred)

#MSE
mse = mean_squared_error(y, y_pred)

#RMSE
rmse = np.sqrt(mse)

# Print the metrics
print("RSS: {}".format(rss))
print("MSE: {}".format(mse))
print("RMSE: {}".format(rmse))
print("R^2: {}".format(r_squared))
```

Ficando assim os resultados:

```
RSS: [4.31058783e+09 1.94942544e+10 5.62589376e+07 ... 2.54229325e+10
 2.93435157e+07 6.78065025e+09]
MSE: 68351286833.039825
RMSE: 261440.79030067177
R^2: 0.48073838168674443
```

### 3.7 Questão 7

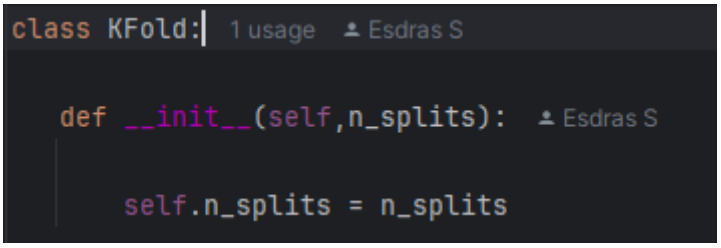
Nessa questão deveria utilizar o K-fold e cross-validation. implementando manualmente e depois calcular o RSS, MSE, RMSE e o RSquare de forma manual também.

As importações:

```
import numpy as np
from src.utils import load_new_dataframe_kc_house
from sklearn.linear_model import LinearRegression
```

Começando pelo KFold implementado manualmente.

Ele foi criado em forma de classe, inicializando com o número de split.




Dentro da classe tem uma função de cross validation score, recebendo a regressão, o X e y.

Nela tem uma variável para armazenar o score. Uma variável para armazenar o tamanho do X. Uma variável em que divide o tamanho do X com o split decidido quando se inicia o kfold.

Depois entra em um for em que 'i' é o range do número de split. Começa iniciando a variável start que recebe 'i' vezes o tamanho do X, e outra variável end que recebe o start mais o tamanho do X se o i for menor que o tamanho de split - 1, se não, o end vai receber o tamanho de X.

Depois é iniciado as variáveis de teste e treino usando os valores adquiridos antes e logo em seguida é calculado o score usando uma variável usando uma função que também está dentro do kfold que usa a regressão linear para calcular o score.

Depois retorna os valores.

```python
def cross_val_score(self, reg, X, y):  1 usage  Esdras S *
    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
```

```python
def _compute_score(self, reg, X, y):  1 usage  new *
    reg.fit(X[0], y[0])
    return reg.score(X[1], y[1])
```

Bom, agora os cálculos.

O cálculo do RSS é basicamente a soma do quadrado de ‘y’ menos ‘predictions’

O cálculo do MSE é basicamente o valor do RSS dividido pelo tamanho do ‘predictions’

O cálculo do RMSE é basicamente o valor do MSE ao quadrado.

O cálculo do RSquare é basicamente “1 - (ss_residual / ss_total)” em que ss_total é a soma dos quadrados totais e p ss_residual é a soma dos quadrados residuais.

```python
def compute_RSS(predictions, y):  3 usages  Esdras S
    # Diferenças quadráticas somadas
    RSS = np.sum(np.square(y - predictions))
    return RSS
def compute_MSE(predictions,y):  2 usages  Esdras S
    RSS = compute_RSS(predictions, y)
    MSE = np.divide(RSS, len(predictions))
    return MSE
def compute_RMSE(predictions,y):  1 usage  Esdras S
    MSE = compute_MSE(predictions, y)
    RMSE = np.sqrt(MSE)
    return RMSE
def compute_R_squared(predictions, y):  1 usage  Esdras S
    ss_total = np.sum(np.square(y - np.mean(y)))  # Soma dos quadrados totais
    ss_residual = compute_RSS(predictions, y)    # Soma dos quadrados residuais
    r_squared = 1 - (ss_residual / ss_total)     # Fórmula do R^2
    return r_squared
```

Agora o corpo principal do código.

Comecei iniciando o ‘X’ eo ‘y’, e logo em seguida iniciando o Kfold, depois inicio o LinearRegression() e faço um fit de X e y, já coletou o predict de X, e logo em seguida executar o kfold.cross_val_score recebendo a regressão o ‘X’ eo ‘y’.

```python
# Create X and y arrays
X = house_df["sqft_living"].values.reshape(-1,1)
y = house_df["price"].values

kf = KFold(n_splits=6)

# Instantiate the model
reg = LinearRegression()

# Fit the model to the data
reg.fit(X,y)

predictions = reg.predict(X)

cv_scores = kf.cross_val_score(reg,X, y)
```

E depois so printei os resultados.

```
print("Cv Score: ",cv_scores)
print("Média: ",np.mean(cv_scores))
print("Desvio Padrão: ",np.std(cv_scores))
print("\n")
print("RSS: {}".format(compute_RSS(predictions,y)))
print("MSE: {}".format(compute_MSE(predictions,y)))
print("RMSE: {}".format(compute_RMSE(predictions,y)))
print("R^2: {}".format(compute_R_squared(predictions,y)))
```

```
Cv Score:  [0.48861281901260256, 0.4927540424997462, 0.48434883551998476, 0.49975740673690683, 0.493582479649735, 0.4917141513484584]
Média:  0.4917949557945723
Desvio Padrão:  0.004709793782661849
```

```
RSS: 1477276362322489.8
MSE: 68351286833.039825
RMSE: 261440.79030067177
R^2: 0.4928532179037931
```

## 4. Conclusões

Eu diria que os resultados foram satisfatórios, provavelmente poderia ser melhor.

Começando pelos resultados da Classificação. Os resultados da Classificação foram o melhor, pois a porcentagem do resultado foi tudo acima de 96% então diria que a taxa de sucesso está ótimo.

Os resultados da Regressão. O resultado Regressão acho que poderia ser melhor, pois se verificar o gráfico com a linha de predição da para perceber que ela deveria ser mais alta. pois os a maiorias dos dados estão concentrado no início. mas dá para perceber que os valores vão subindo mais a cada sqft living, mais do que a linha de predição mostra.

## 5. Próximos passos

Acho que os próximos passo seria encontrar um dataset de regressão mais novo ou modifica-lo mais removendo alguns dados do sqft living para melhorar a precisão da linha de predição. o dataset de classificação eu diria que teve um bom rendimento então não precisa modificá-lo nem troca-lo