Universidade São Judas Tadeu

UC – Inteligência Artificial (Teórica e Prática)

Curso: Ciência da Computação	**Semestre/Ano:** 02/2022
Nomes dos Alunos:	**RAs:**
Akme Re Monteiro de Almeida	822221862
Bruno Dalla	819165320
Guilherme Peres	822231963
Gustavo Santos	821119996
Romulo Adriel	821136151
Victor Hugo	819166807

Conjunto de dados escolhida pelo grupo: Dataset de Predição de AVC - https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

Definição do problema

De acordo com a Organização Mundial da Saúde (OMS) o AVC é a 2ª causa de morte no mundo, responsável por aproximadamente 11% do total de mortes. É outro problema de saúde que está aumentando em todo o mundo devido à adoção de mudanças no estilo de vida que desconsideram o estilo de vida saudável e os bons hábitos alimentares. Assim, novos dispositivos eletrônicos emergentes que registram os sinais vitais de saúde abriram caminho para a criação de uma solução automatizada com técnicas de IA em seu núcleo. Assim, semelhante às doenças cardíacas, começaram os esforços para criar testes de laboratório que predizem o AVC. O conjunto de dados apresentado aqui tem muitos fatores que destacam o estilo de vida dos pacientes e, portanto, nos dá a oportunidade de criar uma solução baseada em IA para ele.

Objetivo:

- Classificar / prever se o paciente pode sofrer um AVC.
- Este é um problema de classificação binária com múltiplas variáveis numéricas e categóricas.

Obtenção dos dados

Importando as bibliotecas necessárias

In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings

warnings.filterwarnings('ignore')
```

In [ ]:
```
# Lendo o arquivo 'stroke-dataset.csv'
dataset = pd.read_csv('stroke-data.csv')

# Exibindo informações sobre o conjunto de dados
dataset.head()
```

Out[ ]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
0	9046	Male	67.0	0	1	Yes	Private	Urban
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural
2	31112	Male	80.0	0	1	Yes	Private	Rural
3	60182	Female	49.0	0	0	Yes	Private	Urban
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural

O dataset contém as seguintes colunas:

- id: Identificador único do paciente
- gender: Gênero (masculino, feminino ou outro)
- age: Idade do paciente
- hypertension: Hipertensão, 0 se o paciente não tem hipertensão, 1 se o paciente tem hipertensão
- heart_disease: Alguma doença cardíaca, 0 se o paciente não tem nenhuma doença cardíaca, 1 se o paciente tem alguma doença cardíaca
- ever_married: Casado alguma vez, "No" se o paciente nunca foi casado, "Yes" se o paciente foi casado alguma vez
- work_type: Tipo de trabalho do paciente, "children", "Govt_job", "Never_worked", "Private" ou "Self-employed"
- residence: Tipo de residência do paciente, rural ou urbana
- avg_glucose_level: Nível médio de glicose no sangue
- bmi: Índice de massa corporal (IMC)
- smoking_status: Histórico de tabagismo do pacience, "formerly smoked", "never smoked", "smokes" or "Unknown"*
- stroke: AVC (1: teve AVC, 0: Não teve AVC)

In [ ]:
```
## Dimensionalidade do conjunto de dados

dataset.shape
```

Out[ ]:
```
(5110, 12)
```

In [ ]:
```
# Listando as colunas

dataset.columns
```

Out[ ]:
```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_marrie
d',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Análise exploratória dos dados

In [ ]:
```
dataset.head()
```

Out[ ]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
0	9046	Male	67.0	0	1	Yes	Private	Urban
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural
2	31112	Male	80.0	0	1	Yes	Private	Rural
3	60182	Female	49.0	0	0	Yes	Private	Urban
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

In [ ]:
```
# Contagem de registros duplicados

dataset.duplicated().sum()
```

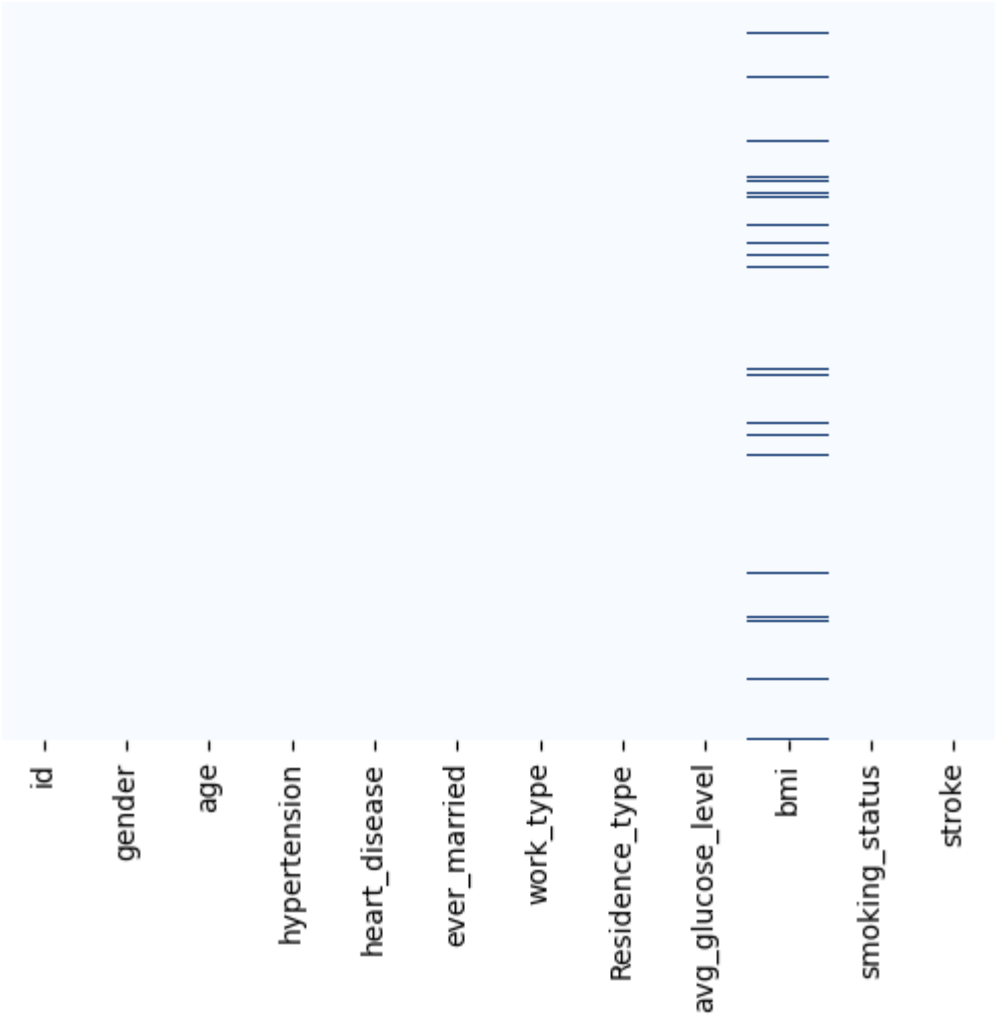
Out[ ]:
```
0
```

In [ ]:
```
# Contagem de registros nulos por coluna

dataset.isnull().sum()
```

Out[ ]:
```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

In [ ]:
```
sns.heatmap(dataset.isnull(), yticklabels = False, cbar = False, cmap="Blues
```

Out[ ]:
```
<AxesSubplot: >
```



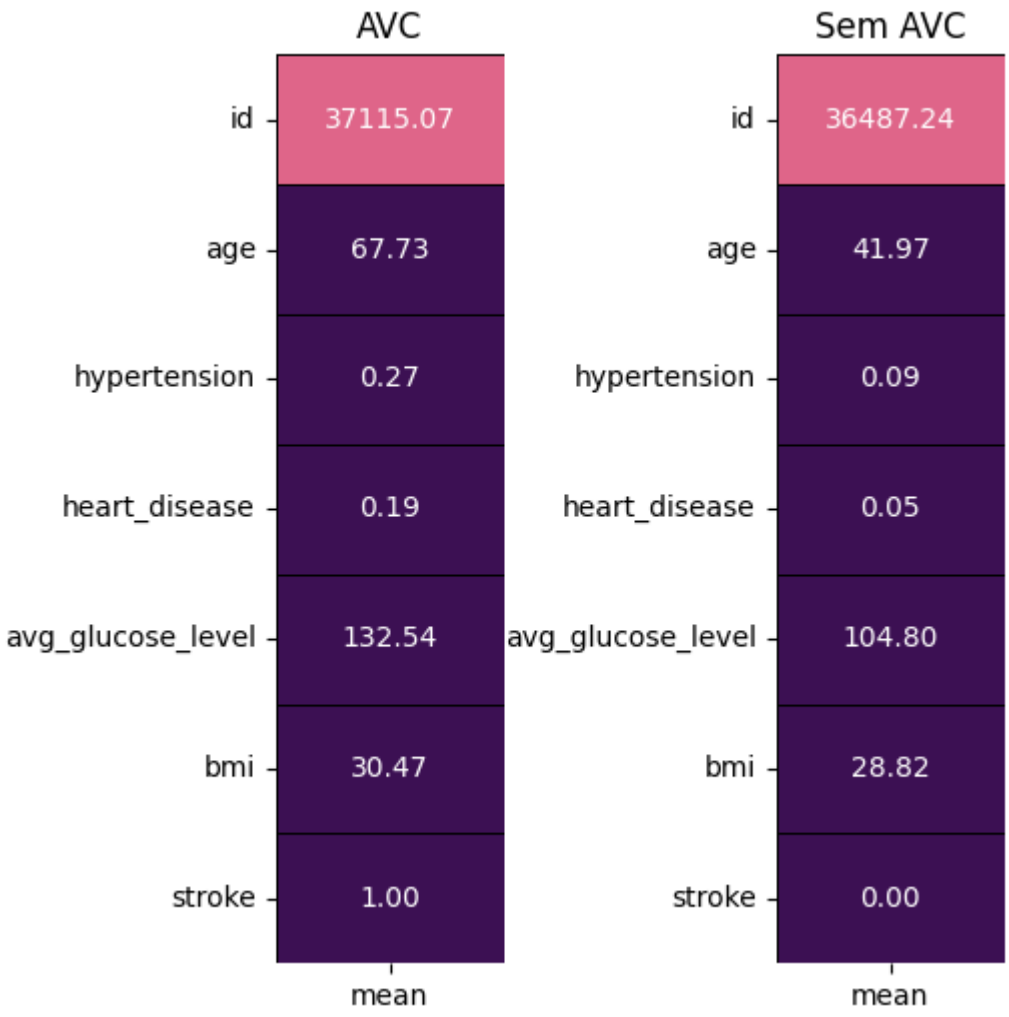
- há campos nulos na coluna IMC em alguns registros.

In [ ]:
```
# Gerando a estatística descritiva com 3 casas decimais

pd.set_option('display.float_format', lambda x: '%.3f' % x)
dataset.describe()
```

Out[ ]:

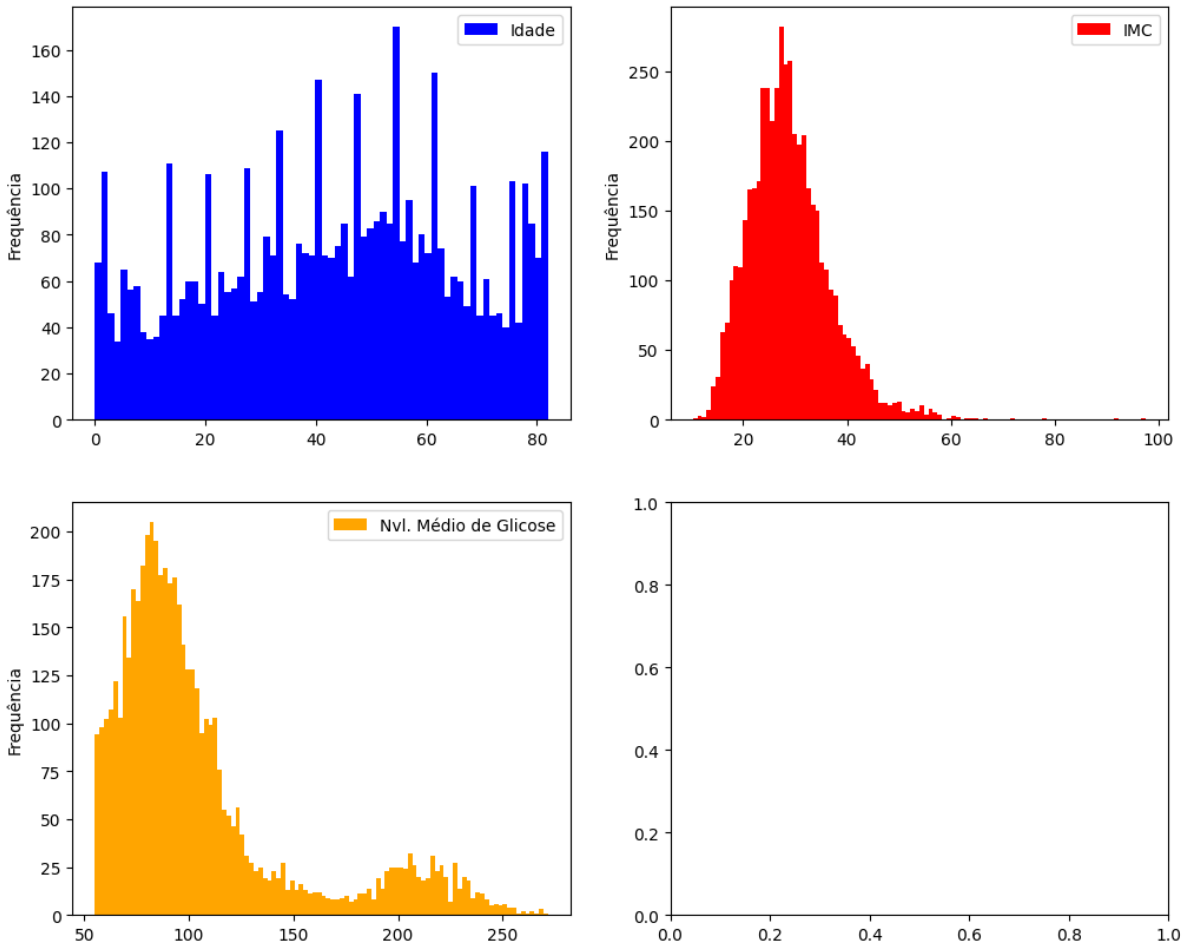
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000	5110.000	5110.000	5110.000	5110.000	4909.000	5110.000
mean	36517.829	43.227	0.097	0.054	106.148	28.893	0.049
std	21161.722	22.613	0.297	0.226	45.284	7.854	0.215
min	67.000	0.080	0.000	0.000	55.120	10.300	0.000
25%	17741.250	25.000	0.000	0.000	77.245	23.500	0.000
50%	36932.000	45.000	0.000	0.000	91.885	28.100	0.000
75%	54682.000	61.000	0.000	0.000	114.090	33.100	0.000
max	72940.000	82.000	1.000	1.000	271.740	97.600	1.000

In [ ]:
```
stroke = dataset[dataset['stroke'] == 1].describe().T
no_stroke = dataset[dataset['stroke'] == 0].describe().T

colors = ['#3C1053','#DF6589']

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (5,5))
plt.subplot(1,2,1)
sns.heatmap(stroke[['mean']],annot = True,cmap = colors,linewidths = 0.4,lin
plt.title('AVC');

plt.subplot(1,2,2)
sns.heatmap(no_stroke[['mean']],annot = True,cmap = colors,linewidths = 0.4,
plt.title('Sem AVC');

fig.tight_layout(pad = 0)
```



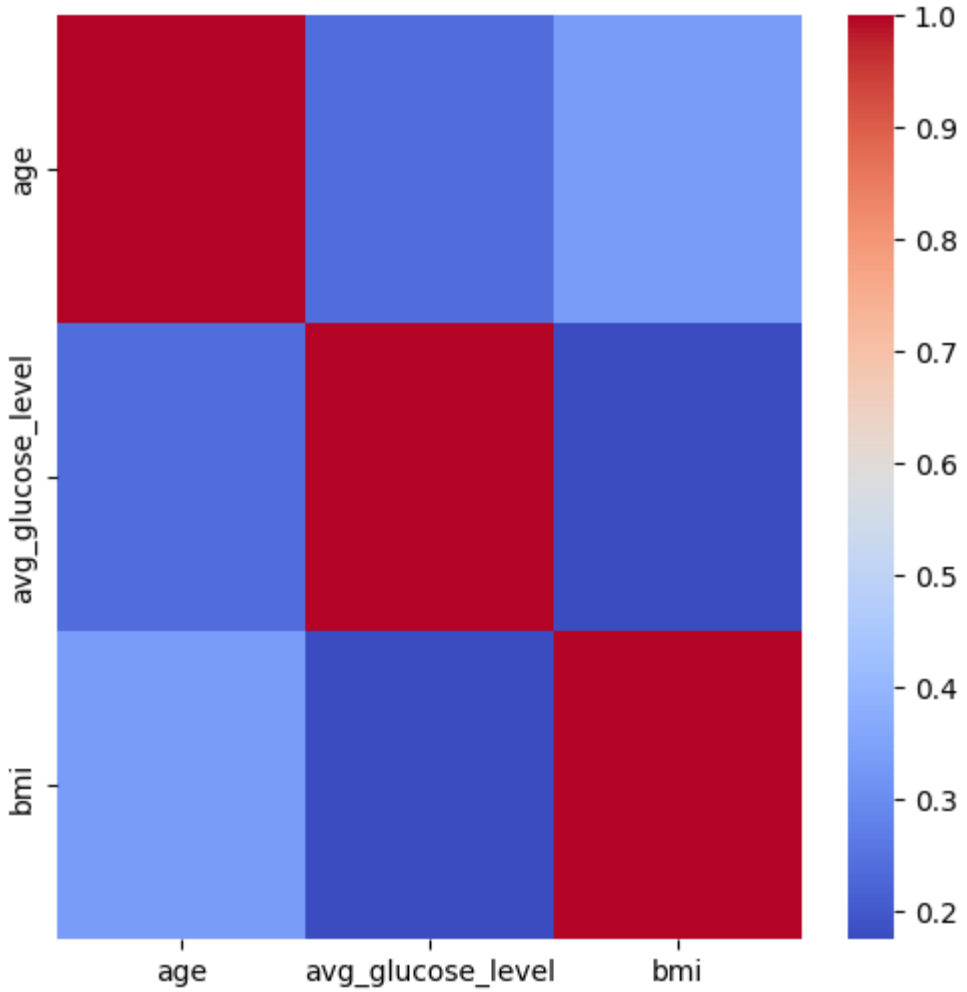
- Valores de média de todas as colunas para pacientes que sofreram e não sofreram AVC
- Idade e nível médio de glicose no sangue são fortes indicadores para identificar um AVC
- A média de idade de pacientes que sofreram AVC(67,73) são muito maiores do que a média dos que não sofreram AVC(41,97).
- O mesmo é válido para nível médio de glicose no sangue. Valores médios de 132,54 indicam uma chance maior de sofrer AVC. Assim como valores médios de 104,80 foram encontrados em paciences que não sofreram AVC.

In [ ]:
```
# histogramas para variáveis numéricas

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

age_fig = dataset.plot(kind="hist", y="age", bins=70, color="b", ax=axes[0][
bmi_fig = dataset.plot(kind="hist", y="bmi", bins=100, color="r", ax=axes[0]
agl_fig = dataset.plot(kind="hist", y="avg_glucose_level", bins=100, color="

age_fig.set_ylabel("Frequência")
bmi_fig.set_ylabel("Frequência")
agl_fig.set_ylabel("Frequência")

plt.show()
```

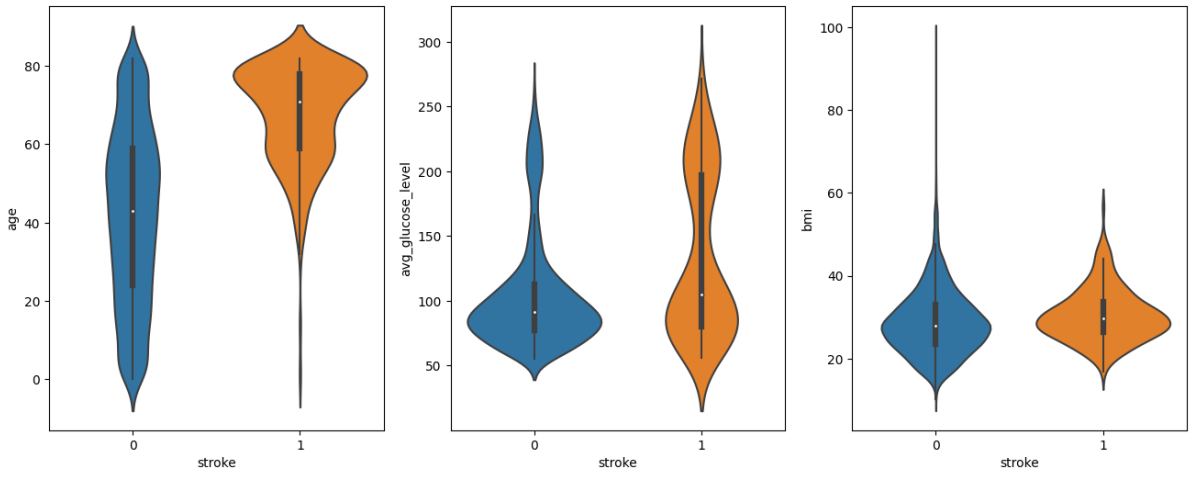


In [ ]:
```
cat_cols = ["gender","hypertension","heart_disease","ever_married","work_typ
cont_cols = ["age","avg_glucose_level","bmi"]
```

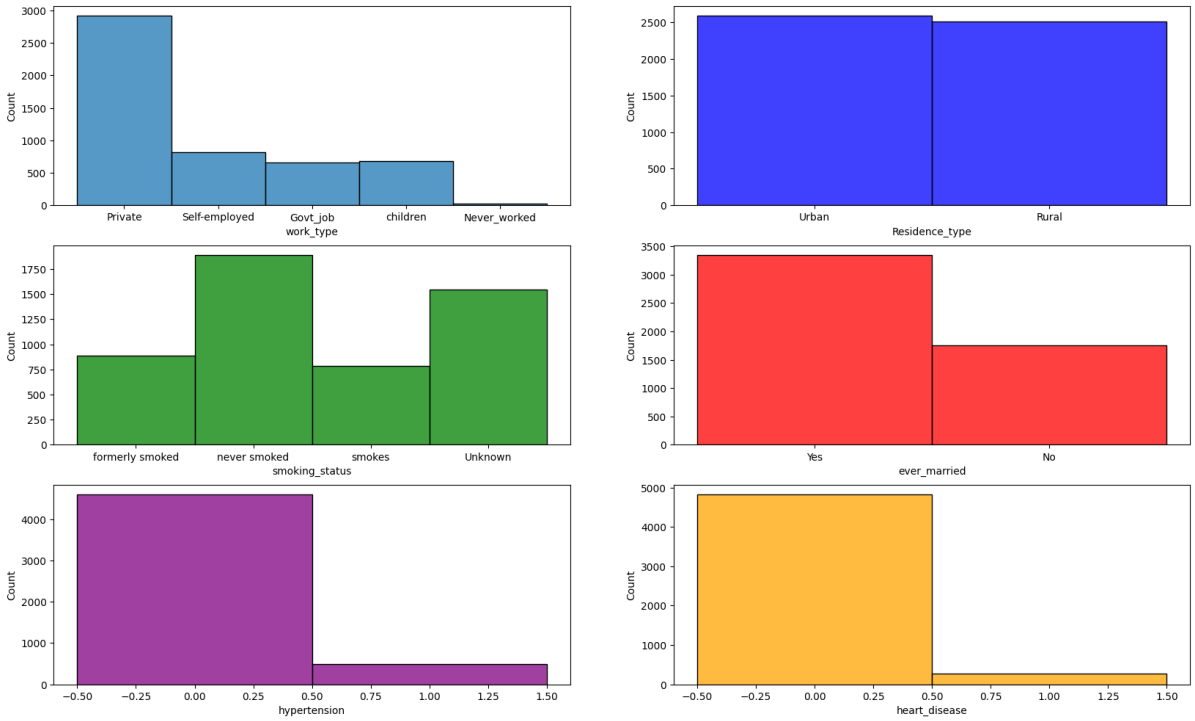
In [ ]:
```
cr = dataset[cont_cols].corr(method='pearson')
plt.figure(figsize = (6,6))
sns.heatmap(cr,cmap="coolwarm")
plt.show()
```



In [ ]:
```
plt.figure(figsize=(16,6))
plt.subplot(1,3,1)
sns.violinplot(x = 'stroke', y = 'age', data = dataset)
plt.subplot(1,3,2)
sns.violinplot(x = 'stroke', y = 'avg_glucose_level', data = dataset)
plt.subplot(1,3,3)
sns.violinplot(x = 'stroke', y = 'bmi', data = dataset)
plt.show()
```

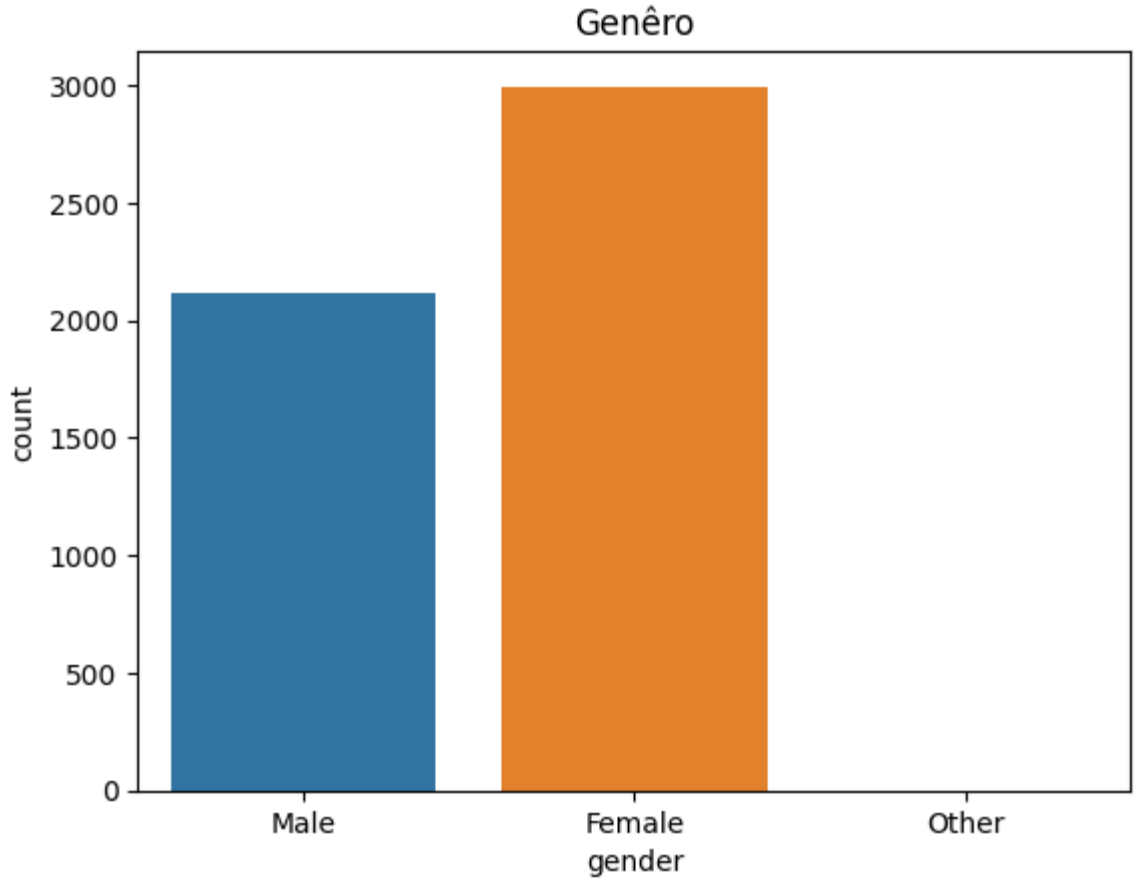


In [ ]:
```
# histogramas para variáveis numéricas

fig, ax = plt.subplots(3,2,figsize=(20,12))
sns.histplot(data=dataset, discrete=True, x="work_type", ax=ax[0][0])
sns.histplot(data=dataset, discrete=True, x="Residence_type", color='b', ax=
sns.histplot(data=dataset, discrete=True, x="smoking_status", color='g', ax=
sns.histplot(data=dataset, discrete=True, x="ever_married", color='r', ax=ax
sns.histplot(data=dataset, discrete=True, x="hypertension", color='purple',
sns.histplot(data=dataset, discrete=True, x="heart_disease", color='orange',

fig.show()
```



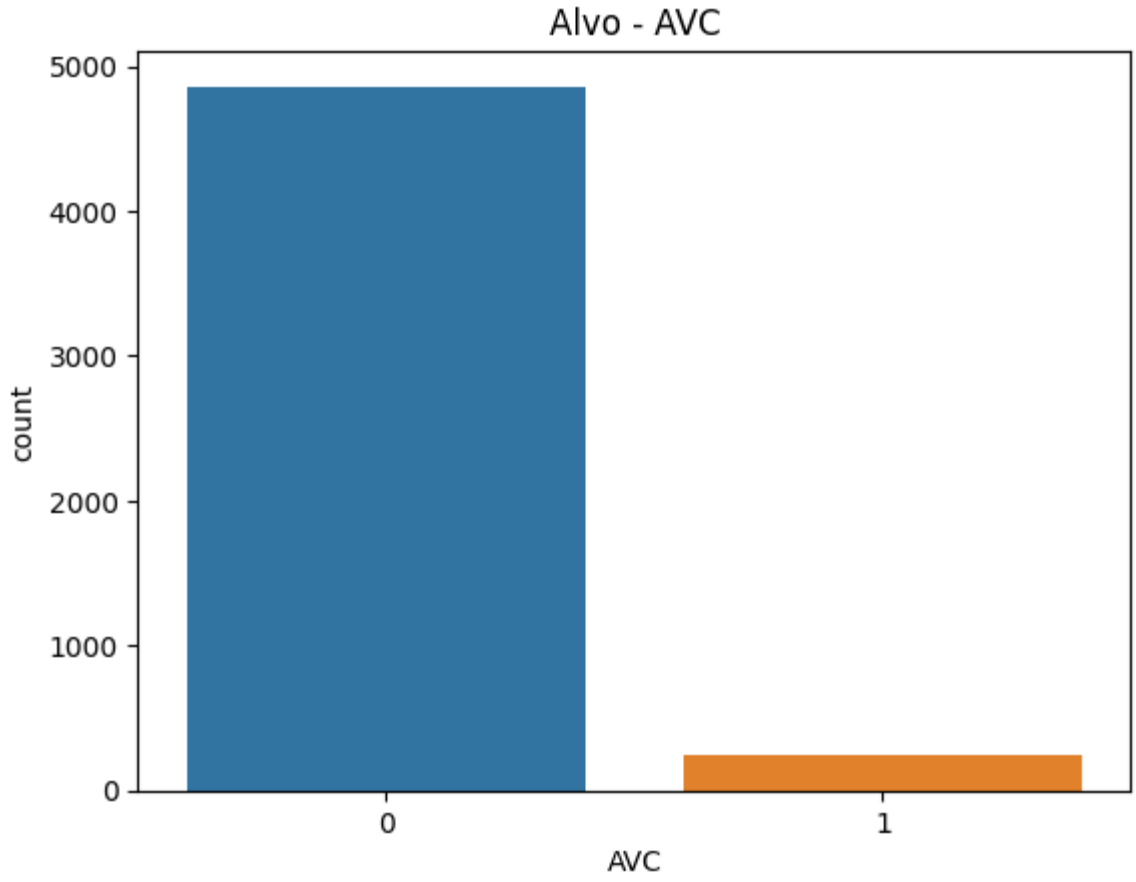
In [ ]:
```
sns.countplot(x=dataset['gender'])
plt.title('Genêro')
```

Out[ ]:
```
Text(0.5, 1.0, 'Genêro')
```



In [ ]:
```
sns.countplot(x=dataset['stroke'])
plt.title('Alvo - AVC')
plt.xlabel('AVC')
```

Out[ ]:
```
Text(0.5, 0, 'AVC')
```



In [ ]:
```
stroke    = dataset[dataset['stroke']==1]
no_stroke = dataset[dataset['stroke']==0]
```

In [ ]:
```
print("Total =", len(dataset))

print("Numero de pessoas que tiveram derrame =", len(stroke))
print("Porcentagem de pessoas que tiveram derrame =", 1.*len(stroke)/len(dat

print("Não tiveram derrame =", len(no_stroke))
print("Porcentagem de pessoas que não tiveram derrame =", 1.*len(no_stroke)/
```

```
Total = 5110
Numero de pessoas que tiveram derrame = 249
Porcentagem de pessoas que tiveram derrame = 4.87279843444227 %
Não tiveram derrame = 4861
Porcentagem de pessoas que não tiveram derrame = 95.12720156555773 %
```

In [ ]:
```
from statistics import median
maleBmi = median(dataset.query('(gender == "Male")')['bmi'])
femaleBmi = median(dataset.query('(gender == "Female")')['bmi'])
print(f'mediana de bmi para genero masculino {maleBmi}')
print(f'mediana de bmi para genero feminino {femaleBmi}')
```

```
mediana de bmi para genero masculino 30.7
mediana de bmi para genero feminino 21.45
```

PREPARAÇÃO DOS DADOS

In [ ]:
```
dataset["gender"].value_counts()
```

Out[ ]:
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [ ]:
```
# Altera o valor "Other" de genêro para "Female", uma vez que estes são maio

dataset.loc[dataset['gender'] == 'Other', 'gender'] = "Female"
dataset["gender"].value_counts()
```

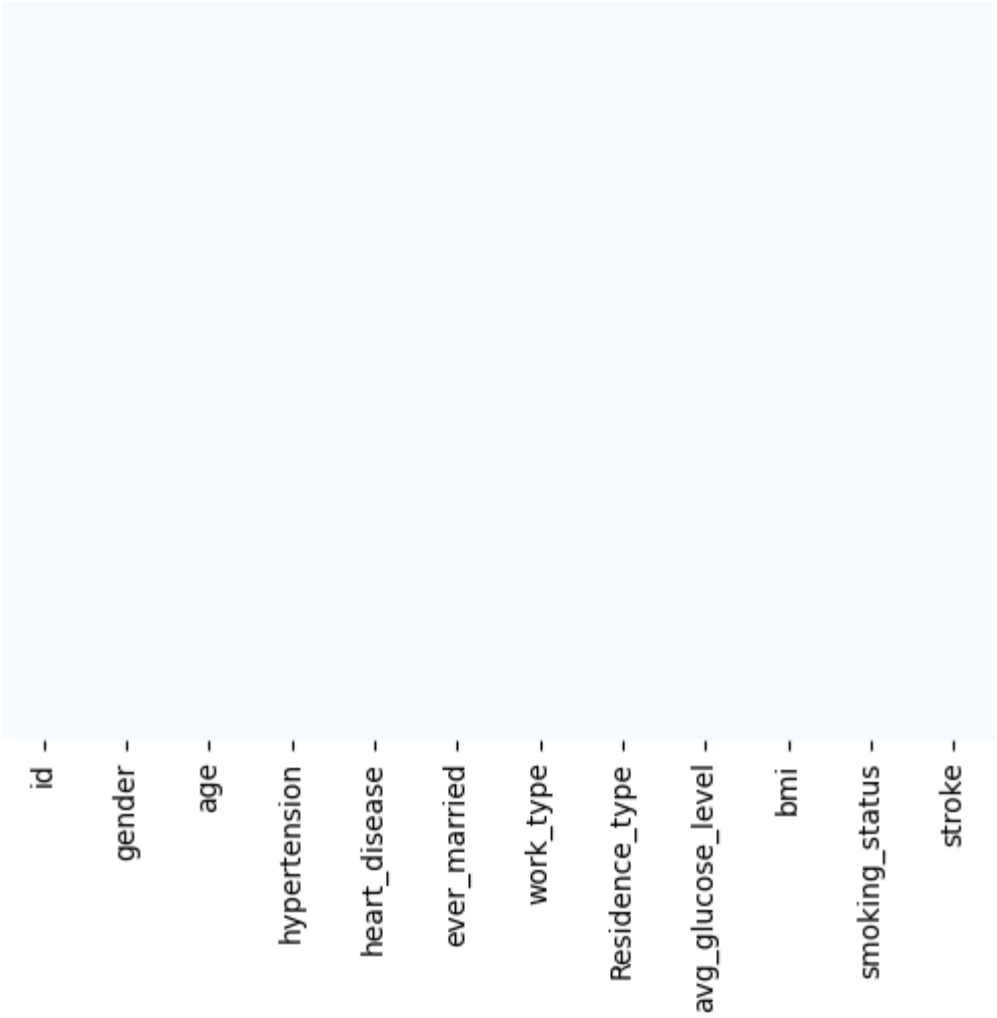
Out[ ]:
```
Female    2995
Male      2115
Name: gender, dtype: int64
```

In [ ]:
```
# Preencher valores nulos de IMC pela média de cada gênero

def fill_bmi(dataset):
    bmi = dataset[0]
    gender = dataset[1]

    if pd.isnull(bmi):
        if gender == 'Male':
            return maleBmi
        else:
            return femaleBmi
    else:
        return bmi
```

In [ ]:
```
dataset['bmi'] = dataset[['bmi','gender']].apply(fill_bmi,axis=1)
```

In [ ]:
```
sns.heatmap(dataset.isnull(), yticklabels = False, cbar = False, cmap="Blues
```

Out[ ]:
```
<AxesSubplot: >
```



In [ ]:
```
dataset['smoking_status'].value_counts()
```

Out[ ]:
```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

In [ ]:
```
# Calcula a probabilidade de fumantes, ex-fumantes e não fumantes. Dado que

prob_FS = 885 / ( 885 + 1892 + 789)
prob_NS = 1892 / (885 + 1892 + 789)
prob_S = 789 / (885 + 1892 + 789)

print(prob_FS)
print(prob_NS)
print(prob_S)
```

```
0.2481772293886708
0.5305664610207516
0.22125630959057768
```

In [ ]:
```
# Distribui os valores da classe "Unknown" na coluna "smoking_status" para a

def spread_smoking_status_by_probability(smoking_status):
    available_classes = ["never smoked", "formerly smoked", "smokes"]
    probabilities = [prob_NS, prob_FS, prob_S]

    if(smoking_status == "Unknown"):
        return np.random.choice(available_classes, p=probabilities)
    else:
        return smoking_status
```

In [ ]:
```
# Aplica a distribuição de valores "Unknown" na coluna "smoking_status"
dataset['smoking_status'] = dataset['smoking_status'].apply(spread_smoking_s
```

In [ ]:
```
dataset['smoking_status'].value_counts()
```

Out[ ]:
```
never smoked       2706
formerly smoked    1266
smokes             1138
Name: smoking_status, dtype: int64
```

In [ ]:
```
# Retirar coluna id

dataset.drop(['id'], axis=1, inplace=True)
```

In [ ]:
```
# Transforma valores categóricos em valores "dummy"
ever_married = pd.get_dummies(dataset['ever_married'], drop_first = True)
residence_type = pd.get_dummies(dataset['Residence_type'], drop_first = True
gender = pd.get_dummies(dataset['gender'], drop_first = True)
work_type = (dataset['work_type'].str.strip('[]')
                    .str.get_dummies(', ')
                    .rename(columns=lambda x: x.strip('"')))
smoking_status = (dataset['smoking_status'].str.strip('[]')
                    .str.get_dummies(', ')
                    .rename(columns=lambda x: x.strip('"')))
```

In [ ]:
```
dataset = pd.concat([dataset, ever_married, residence_type, gender, work_typ
```

In [ ]:
```
dataset.head()
```

Out[ ]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_g
0	Male	67.000	0	1	Yes	Private	Urban	
1	Female	61.000	0	0	Yes	Self-employed	Rural	
2	Male	80.000	0	1	Yes	Private	Rural	
3	Female	49.000	0	0	Yes	Private	Urban	
4	Female	79.000	1	0	Yes	Self-employed	Rural	

5 rows × 22 columns

In [ ]:
```
# Dropa as colunas redundantes.
dataset.drop(['ever_married', 'Residence_type', 'gender', 'work_type', 'smok
```

In [ ]:
```
# Renomeia as colunas "Yes", "Urban" e "Male" para "ever_married", "residenc
dataset = dataset.rename(columns={'Yes':'ever_married', 'Urban': 'residence_
```

In [ ]:
```
dataset.columns
```

Out[ ]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke', 'ever_married', 'residence_type', 'gender', 'Govt_job',
       'Never_worked', 'Private', 'Self-employed', 'children',
       'formerly smoked', 'never smoked', 'smokes'],
      dtype='object')
```

In [ ]:
```
dataset.head()
```

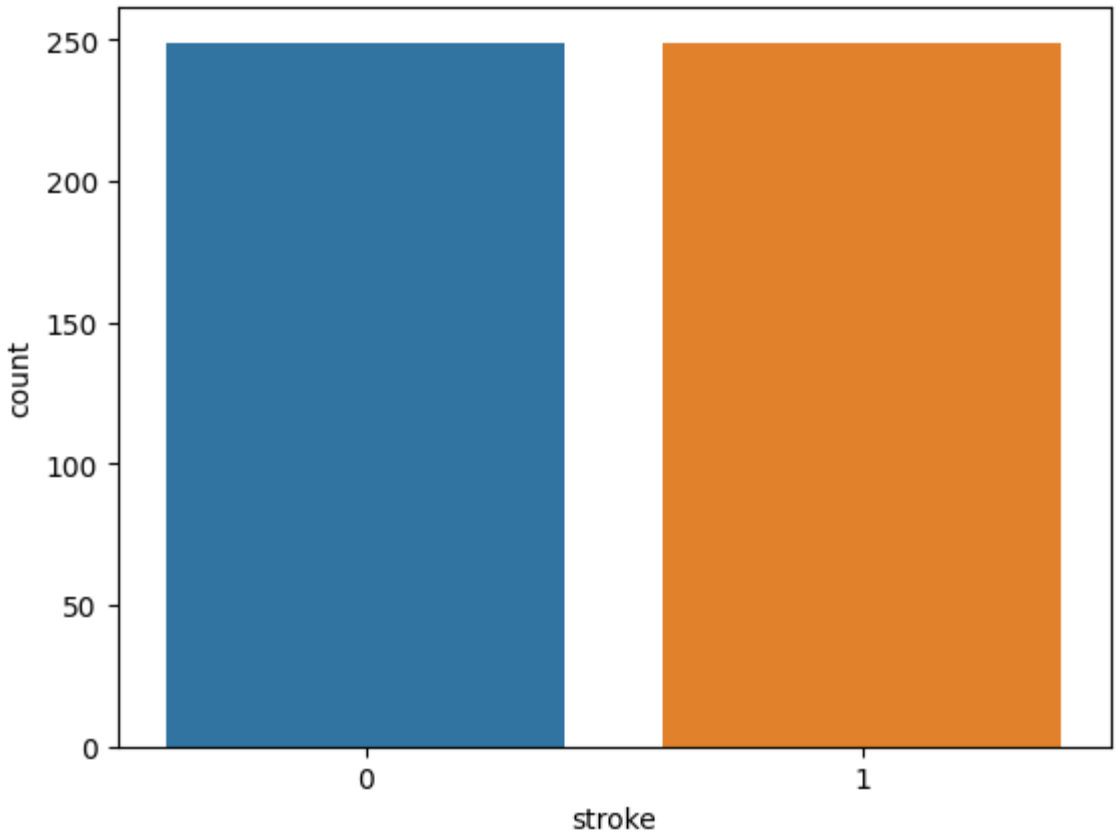
Out[ ]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	ever_married	residenc
0	67.000	0	1	228.690	36.600	1	1	
1	61.000	0	0	202.210	21.450	1	1	
2	80.000	0	1	105.920	32.500	1	1	
3	49.000	0	0	171.230	34.400	1	1	
4	79.000	1	0	174.120	24.000	1	1	

In [ ]:
```
# Aplica técnica de subamostragem com RandomUnderSampler

from imblearn.under_sampling import RandomUnderSampler

oversample = RandomUnderSampler()
dataset, dataset['stroke']=oversample.fit_resample(dataset, dataset['stroke'
```

In [ ]:
```
sns.countplot(x=dataset['stroke'])
```

Out[ ]:
```
<AxesSubplot: xlabel='stroke', ylabel='count'>
```



In [ ]:
```
# Extrai as colunas numéricas

num_columns = [c for c, t in zip(dataset.dtypes.index, dataset.dtypes) if t
num_columns
```

Out[ ]:
```
['age', 'avg_glucose_level', 'bmi']
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

dataset[num_columns] = scaler.fit_transform(dataset[num_columns])
```

In [ ]:
```
dataset.head()
```

Out[ ]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	ever_married	residenc
0	0.213	1	0	1.691	0.952	0	1	
1	1.082	0	0	-0.284	-0.980	0	1	
2	0.076	1	0	-0.507	2.117	0	1	
3	-0.564	0	0	-0.069	-1.231	0	1	
4	-0.747	0	0	0.109	-0.685	0	1	

MODELAGEM

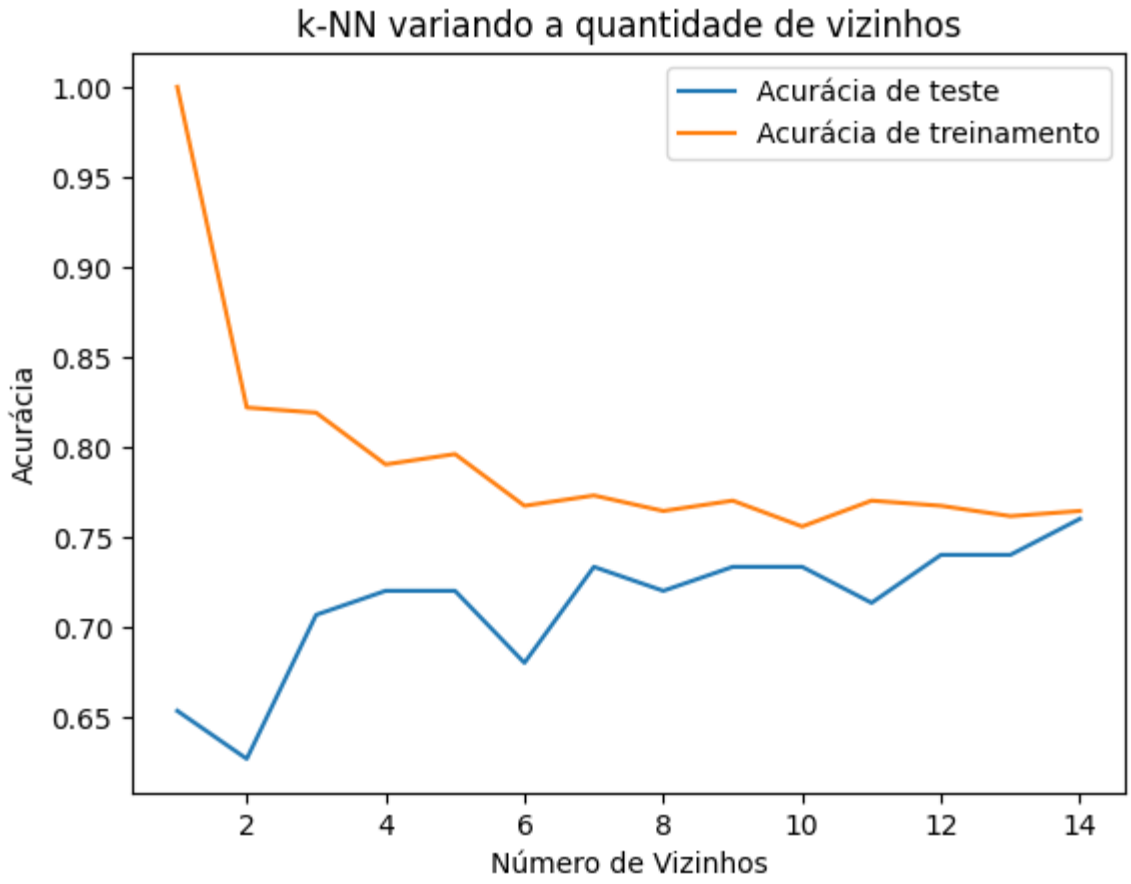
In [ ]:
```
# Cria 2 subconjuntos de dados, 1 somente com a coluna alvo(AVC) e outro com

X_data = dataset.drop('stroke',axis=1).values
y_data = dataset['stroke'].values
```

In [ ]:
```
# Cria subconjuntos para treinamento e teste em uma proporção de 7 para 3, r

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_siz
```

MODELAGEM KNN

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# Configura e armazena as acurácias em matrizes de treinamento e teste.
neighbors = np.arange(1,15)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    # Configura um classificador "knn" com k vizinhos
    knn = KNeighborsClassifier(n_neighbors=k)

    # treina o modelo
    knn.fit(X_train, y_train)

    # Computa a acurácia no conjunto de treinamento
    train_accuracy[i] = knn.score(X_train, y_train)

    # Computa a acurácia no conjunto de teste
    test_accuracy[i] = knn.score(X_test, y_test)
```

In [ ]:
```
# Gera o gráfico
plt.title('k-NN variando a quantidade de vizinhos')
plt.plot(neighbors, test_accuracy, label='Acurácia de teste')
plt.plot(neighbors, train_accuracy, label='Acurácia de treinamento')
plt.legend()
plt.xlabel('Número de Vizinhos')
plt.ylabel('Acurácia')
plt.show()
```



In [ ]:
```
# Configura um classificador "knn" com k vizinhos
knn = KNeighborsClassifier(n_neighbors=7)
```

In [ ]:
```
# Treina o modelo
knn.fit(X_train, y_train)
```

Out[ ]:
```
▾        KNeighborsClassifier

KNeighborsClassifier(n_neighbors=7)
```

In [ ]:
```
# Recupera a acurácia.
# Observação: Em algoritimos de classificação o método "score()" representa

knn.score(X_test,y_test)
```

Out[ ]:
```
0.7333333333333333
```

In [ ]:
```
# Recupera as predições utilizando o classificador que treinamos
y_knn_pred = knn.predict(X_test)
```

Modelagem para Árvore de Decisão

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [ ]:
```
y_dt_predict_test = decision_tree.predict(X_test)
```

In [ ]:
```
decision_tree_score = decision_tree.score(X_test, y_test)
```

Modelagem para regressão logística

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
```

In [ ]:
```
logit.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression

LogisticRegression()
```

In [ ]:
```
y_logit_predict_test = logit.predict(X_test)
```

In [ ]:
```
logit_score = logit.score(X_test, y_test)
logit_score
```

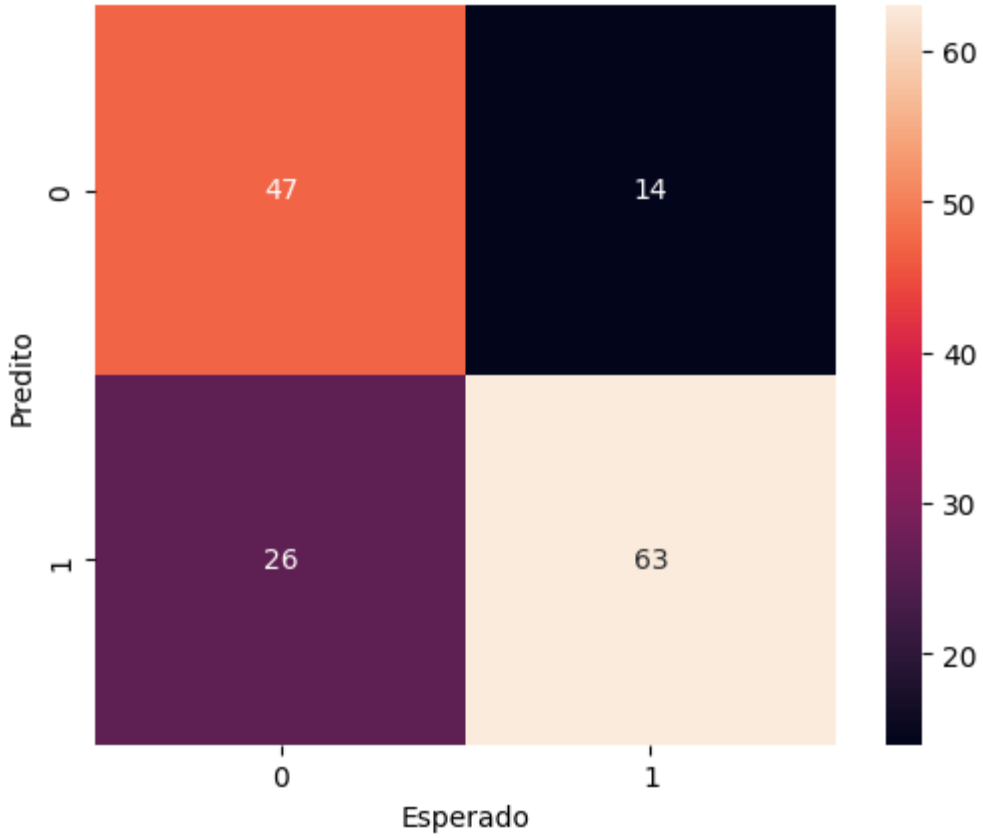
Out[ ]:
```
0.78
```

AVALIAÇÃO

Avaliação KNN

In [ ]:
```
from sklearn.metrics import confusion_matrix
cm_knn = confusion_matrix(y_test, y_knn_pred)
sns.heatmap(cm_knn.T, square=True, annot=True, fmt='d')
plt.xlabel('Esperado')
plt.ylabel('Predito')
```

Out[ ]:
```
Text(77.92222222222227, 0.5, 'Predito')
```



In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.64      0.70        73
           1       0.71      0.82      0.76        77

    accuracy                           0.73       150
   macro avg       0.74      0.73      0.73       150
weighted avg       0.74      0.73      0.73       150
```

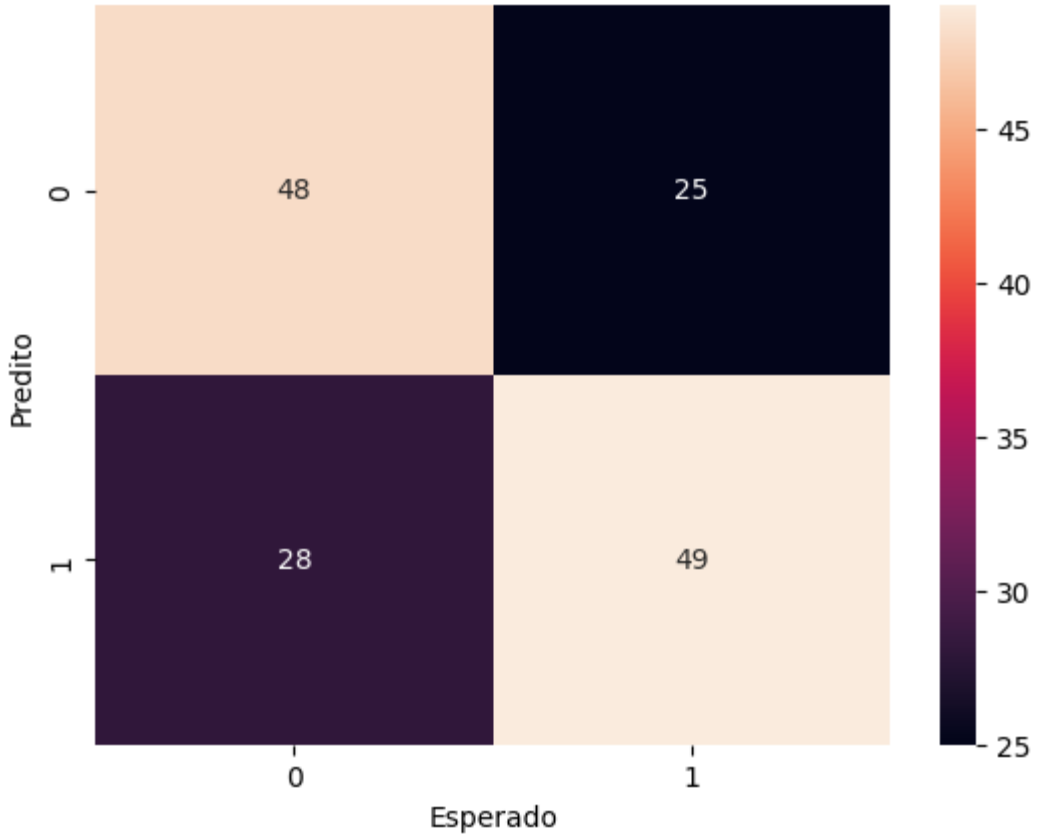
Avaliação Árvore de Decisão

In [ ]:
```
# Acurácia para árvore de decisão
decision_tree_score
```

Out[ ]:
```
0.6466666666666666
```

In [ ]:
```
# Matriz de confusão para árvore de decisão
cm_dt = confusion_matrix(y_test, y_dt_predict_test)
sns.heatmap(cm_dt, annot=True, fmt="d")
plt.xlabel('Esperado')
plt.ylabel('Predito')
```

Out[ ]:
```
Text(50.722222222222214, 0.5, 'Predito')
```



In [ ]:
```
print(classification_report(y_test, y_dt_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.63      0.66      0.64        73
           1       0.66      0.64      0.65        77

    accuracy                           0.65       150
   macro avg       0.65      0.65      0.65       150
weighted avg       0.65      0.65      0.65       150
```

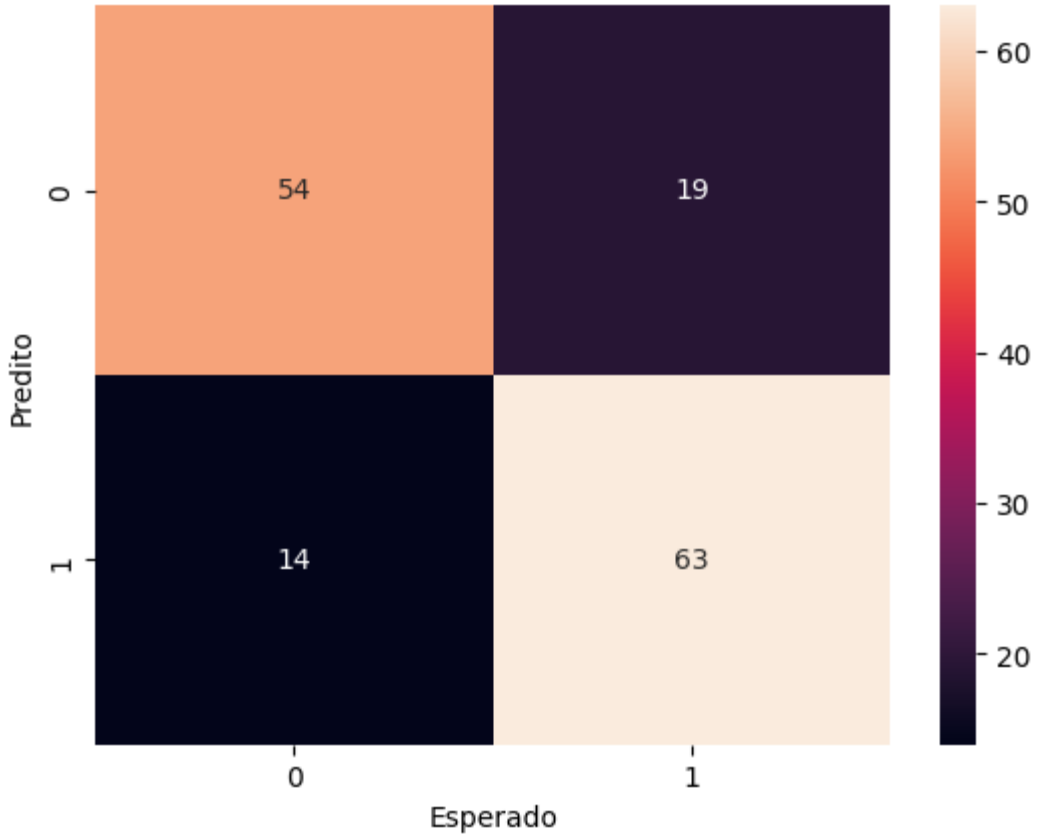
Avaliação para regressão logística

In [ ]:
```
# Acurácia para regressão logística
logit_score
```

Out[ ]:
```
0.78
```

In [ ]:
```
# Matriz de confusão para regressão logística
cm_logit = confusion_matrix(y_test, y_logit_predict_test)
sns.heatmap(cm_logit, annot=True, fmt="d")
plt.xlabel('Esperado')
plt.ylabel('Predito')
```

Out[ ]:
```
Text(50.722222222222214, 0.5, 'Predito')
```



In [ ]:
```
print(classification_report(y_test, y_logit_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.74      0.77        73
           1       0.77      0.82      0.79        77

    accuracy                           0.78       150
   macro avg       0.78      0.78      0.78       150
weighted avg       0.78      0.78      0.78       150
```

Montagem do Comitê

Validação cruzada

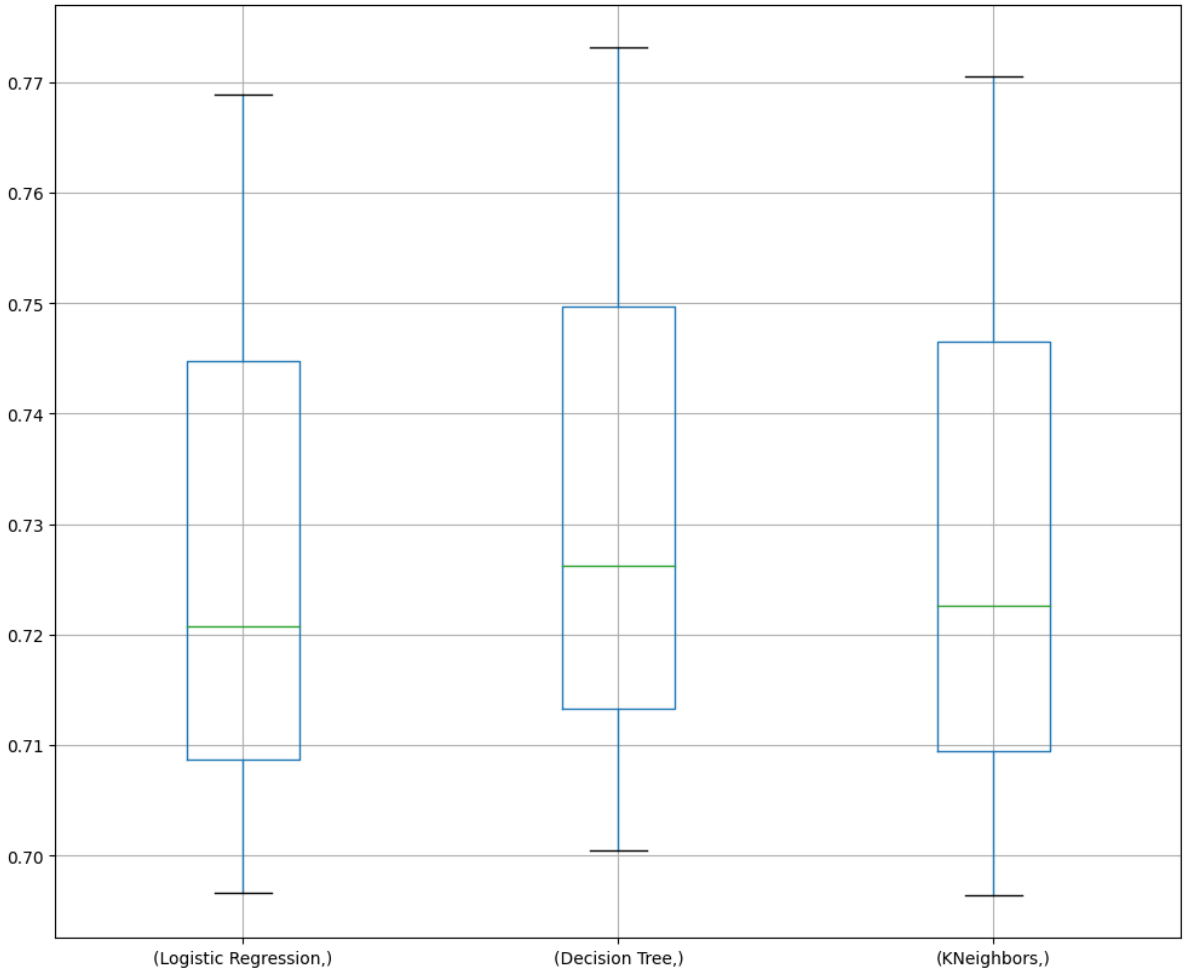
In [ ]:
```
from sklearn.model_selection import KFold # Validação Cruzada com KFold
from sklearn.model_selection import cross_validate # Validação cruzada
from sklearn.model_selection import cross_val_predict # Predição
kfold = KFold(n_splits=10, random_state=2022, shuffle=True) # k=10, divide o
accuracy=[]
recall=[]
precision=[]

classifiers=['Logistic Regression',
             'Decision Tree',
             'KNeighbors']

models=[knn,
        decision_tree,
        logit]

scoring = {'acc': 'accuracy',
           'prec': 'precision_macro',
           'rec': 'recall_macro'}

for model in models:

    cv_score = cross_validate(model, X_data, y_data, cv = kfold,scoring = sc
    accuracy.append(cv_score['test_acc'].mean())
    recall.append(cv_score['test_rec'].mean())
    precision.append(cv_score['test_prec'].mean())

new_models_dataframe2=pd.DataFrame({'accuracy_mean':accuracy,'recall_mean':r
new_models_dataframe2
```

Out[ ]:

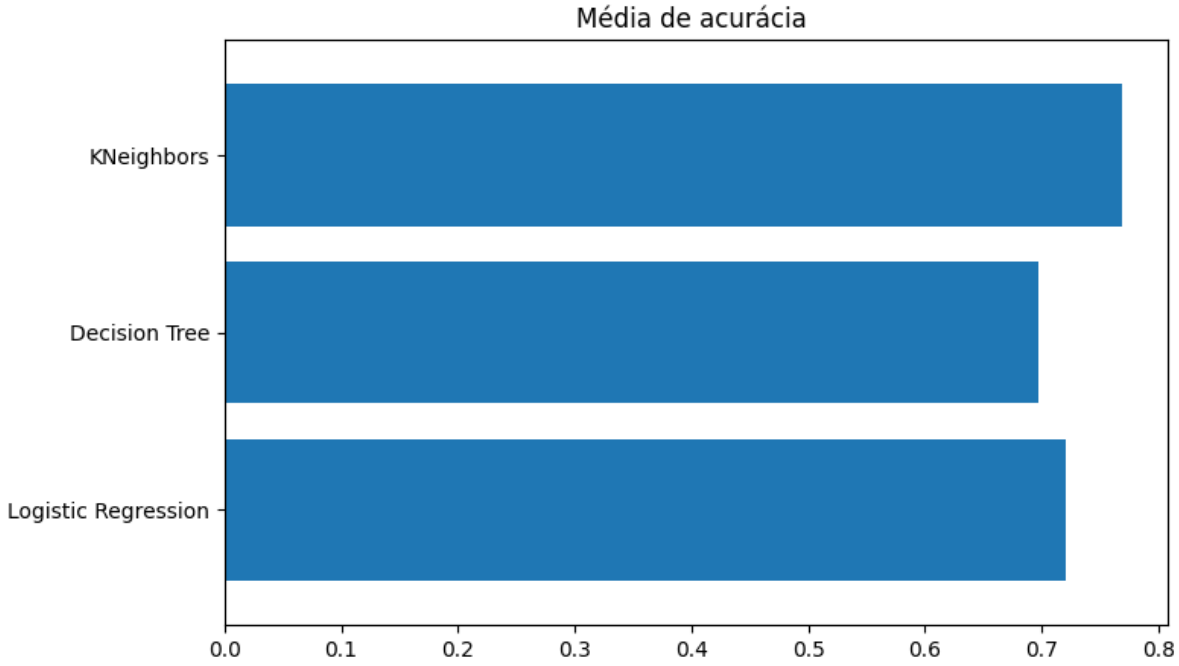
	accuracy_mean	recall_mean	precision_mean
Logistic Regression	0.721	0.726	0.723
Decision Tree	0.697	0.701	0.696
KNeighbors	0.769	0.773	0.771

In [ ]:
```
# Boxplot das médias dos critérios de avaliação
plt.subplots(figsize=(12,10))
box=pd.DataFrame([accuracy, recall, precision],index=[classifiers])
box.T.boxplot()
```

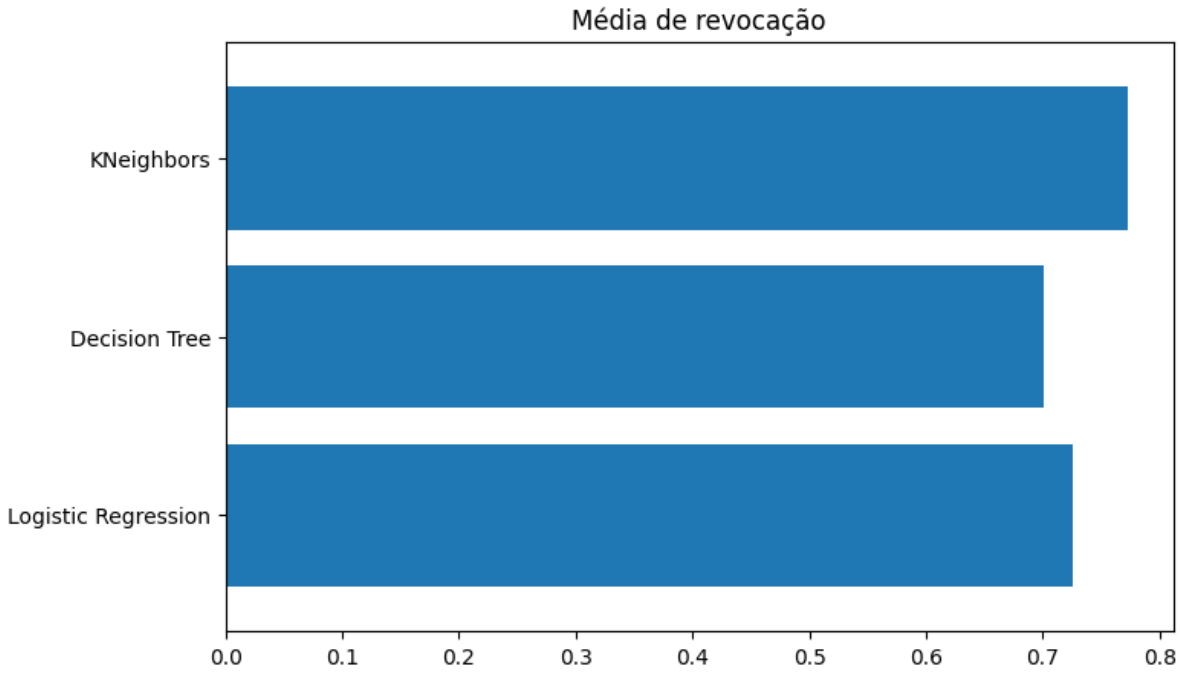
Out[ ]:
```
<AxesSubplot: >
```



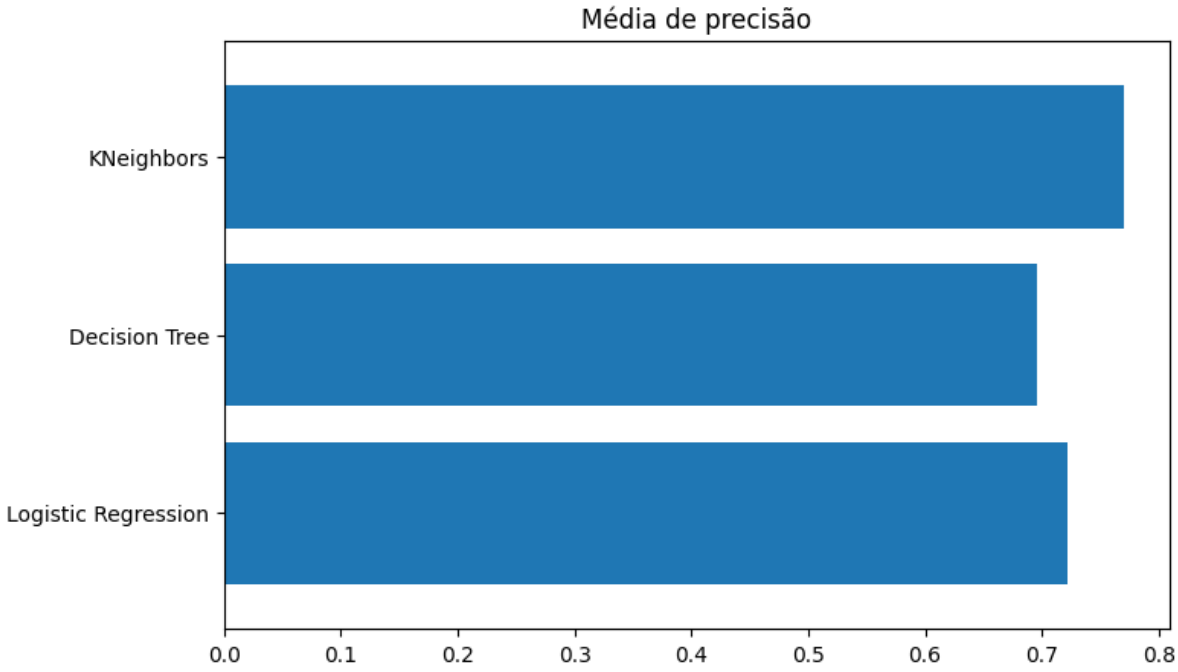
In [ ]:
```
new_models_dataframe2['accuracy_mean'].plot.barh(width=0.8)
plt.title('Média de acurácia ')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```



In [ ]:
```
new_models_dataframe2['recall_mean'].plot.barh(width=0.8)
plt.title('Média de revocação')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```



In [ ]:
```
new_models_dataframe2['precision_mean'].plot.barh(width=0.8)
plt.title('Média de precisão')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```


Média de precisão
KNeighbors
Decision Tree
Logistic Regression
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8