Machine Learning e Data Science com Python de A à Z – Completo Jones Granatyr



Versão que o curso foi gravado	Nova Versão
	Anaconda 3.7
Spyder 3.1.2	Spyder 3.3.2
	Python 3.7.1
	pandas 0.23.4
	scikit-learn 0.20.1

Transformação de variáveis categóricas I - base censo / Transformação de variáveis categóricas II - base censo pre processamento census.py import pandas as pd import pandas as pd base = pd.read csv('census.csv') previsores = base.iloc[:,0:14].values base = pd.read csv('census.csv') classe = base.iloc[:,14].values previsores = base.iloc[:,0:14].values classe = base.iloc[:.14].values from sklearn.preprocessing import LabelEncoder, OneHotEncoder labelencorder previsores = LabelEncoder() from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.compose import ColumnTransformer #labels = labelencorder previsores.fit transform(previsores[:,1]) previsores[:,1] = labelencorder previsores.fit transform(previsores[:,1]) onehotencorder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(), previsores[:,3] = labelencorder previsores.fit transform(previsores[:,3]) [1,3,5,6,7,8,9,13])],remainder='passthrough') previsores[:,5] = labelencorder previsores.fit transform(previsores[:,5]) previsores = onehotencorder.fit transform(previsores).toarray() previsores[:,6] = labelencorder_previsores.fit_transform(previsores[:,6]) previsores[:,7] = labelencorder previsores.fit transform(previsores[:,7]) labelencorder classe = LabelEncoder() previsores[:,8] = labelencorder_previsores.fit_transform(previsores[:,8]) classe = labelencorder_classe.fit_transform(classe) previsores[:,9] = labelencorder previsores.fit transform(previsores[:,9]) previsores[:,13] = labelencorder previsores.fit transform(previsores[:,13]) onehotencorder = OneHotEncoder(categorical_features= [1,3,5,6,7,8,9,13])]) previsores = onehotencorder.fit transform(previsores).toarray() labelencorder classe = LabelEncoder() classe = labelencorder classe.fit transform(classe)

Divisão das bases de dados em treinamento e teste template credit data.py import pandas as pd import pandas as pd import numpy as np base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read_csv('credit_data.csv') base.loc[base.age < 0, 'age'] = 40.92 previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = imputer.fit(previsores[:, 1:4]) imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit_transform(previsores) from sklearn.cross validation import train test split previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = from sklearn.model_selection import train_test_split train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = # importação da biblioteca train_test_split(previsores, classe, test_size=0.25, random_state=0) # criação do classificador classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) matriz = confusion matrix(classe teste, previsoes)

```
Divisão das bases de dados em treinamento e teste
                                                                                    template census.py
import pandas as pd
                                                                                           import pandas as pd
base = pd.read csv('census.csv')
                                                                                           base = pd.read csv('census.csv')
previsores = base.iloc[:,0:14].values
                                                                                          previsores = base.iloc[:,0:14].values
classe = base.iloc[:.14].values
                                                                                           classe = base.iloc[:.14].values
from sklearn.preprocessing import OneHotEncoder
                                                                                           from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
                                                                                           from sklearn.compose import ColumnTransformer
onehotencorder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(), [1,3,5,6,7,8|column_transformer = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
previsores = onehotencorder.fit transform(previsores).toarray()
                                                                                                                                 [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough')
                                                                                          previsores = column tranformer.fit transform(previsores).toarray()
labelencorder classe = LabelEncoder()
classe = labelencorder classe.fit transform(classe)
                                                                                           labelencorder classe = LabelEncoder()
                                                                                           classe = labelencorder_classe.fit_transform(classe)
from sklearn.preprocessing import StandardScaler
                                                                                           from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit transform(previsores)
                                                                                           scaler = StandardScaler()
                                                                                           previsores = scaler.fit_transform(previsores)
from sklearn.cross validation import train test split
previsores treinamento, previsores teste, classe treinamento,
                                                                                           from sklearn.model selection import train test split
             classe_teste = train_test_split(previsores, classe, test_size=0.15, random_s|previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = train_test_split(
                                                                                                                                      previsores, classe, test size=0.15, random state=0)
# importação da biblioteca
# criação do classificador
classificador.fit(previsores treinamento, classe treinamento)
previsoes = classificador.predict(previsores teste)
from sklearn.metrics import confusion matrix, accuracy score
precisao = accuracy_score(classe_teste, previsoes)
matriz = confusion matrix(classe teste, previsoes)
```

Naive bayes com scikit learn - base crédito naive bayes credit-data.py import pandas as pd import pandas as pd import numpy as np base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read_csv('credit_data.csv') base.loc[base.age < 0, 'age'] = 40.92 previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = imputer.fit(previsores[:, 1:4]) imputer = SimpleImputer(missing values=np.nan, strategy = 'mean') previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.cross validation import train test split previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = from sklearn.model selection import train test split train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.25, random state=0) from sklearn.naive bayes import GaussianNB classificador = GaussianNB() from sklearn.naive bayes import GaussianNB classificador.fit(previsores treinamento, classe treinamento) classificador = GaussianNB() previsoes = classificador.predict(previsores teste) classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) from sklearn.metrics import confusion_matrix, accuracy_score matriz = confusion matrix(classe teste, previsoes) precisao = accuracy score(classe teste, previsoes) matriz = confusion matrix(classe teste, previsoes)

Árvores de decisão com scikit-learn - base crédito arvores decisao credit data.py import pandas as pd import pandas as pd import numpy as np base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read_csv('credit_data.csv') base.loc[base.age < 0, 'age'] = 40.92previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = imputer.fit(previsores[:, 1:4]) imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn..cross validation import train test split previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = from sklearn.model selection import train test split train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.25, random state=0) from sklearn.tree import DecisionTreeClassifier classificador = DecisionTreeClassifier(criterion='entropy', random_state=0) from sklearn.tree import DecisionTreeClassifier classificador = DecisionTreeClassifier(criterion='entropy', random state=0) classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) from sklearn.metrics import confusion_matrix, accuracy_score matriz = confusion matrix(classe teste, previsoes) precisao = accuracy score(classe teste, previsoes) matriz = confusion matrix(classe teste, previsoes)

Árvores de decisão com scikit-learn - base censo arvores decisao census.py import pandas as pd import pandas as pd base = pd.read csv('census.csv') base = pd.read csv('census.csv') previsores = base.iloc[:, 0:14].values previsores = base.iloc[:, 0:14].values classe = base.iloc[:, 14].values classe = base.iloc[:, 14].values from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.preprocessing import OneHotEncoder labelencoder previsores = LabelEncoder() from sklearn.compose import ColumnTransformer previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) from sklearn.preprocessing import LabelEncoder, OneHotEncoder previsores[:, 5] = labelencoder previsores.fit transform(previsores[:, 5]) labelencoder previsores = LabelEncoder() previsores[:, 6] = labelencoder previsores.fit transform(previsores[:, 6]) previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8]) previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5]) previsores[:, 6] = labelencoder previsores.fit transform(previsores[:, 6]) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 8] = labelencoder previsores.fit transform(previsores[:, 8]) onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13]) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) previsores = onehotencoder.fit transform(previsores).toarray() previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) labelencoder classe = LabelEncoder() column tranformer = ColumnTransformer([('one hot encoder', OneHotEncoder(), classe = labelencoder classe.fit transform(classe) [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough') previsores = column tranformer.fit transform(previsores).toarray() from sklearn.preprocessing import StandardScaler scaler = StandardScaler() labelencoder classe = LabelEncoder() classe = labelencoder_classe.fit_transform(classe) previsores = scaler.fit transform(previsores) from sklearn.preprocessing import StandardScaler from sklearn.cross validation import train test split previsores treinamento, previsores teste, classe treinamento, classe teste = scaler = StandardScaler() train test split(previsores, classe, test size=0.15, random state=0) previsores = scaler.fit transform(previsores) from sklearn.tree import DecisionTreeClassifier from sklearn.model selection import train test split classificador = DecisionTreeClassifier(criterion='entropy', random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = classificador.fit(previsores treinamento, classe treinamento) train test split(previsores, classe, test size=0.15, random state=0) previsoes = classificador.predict(previsores teste) from sklearn.tree import DecisionTreeClassifier classificador = DecisionTreeClassifier(criterion='entropy', random state=0) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) classificador.fit(previsores_treinamento, classe_treinamento) matriz = confusion matrix(classe teste, previsoes) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) matriz = confusion matrix(classe teste, previsoes)"

Random forest com scikit-learn - base crédito random forest credit data.py import pandas as pd import pandas as pd import numpy as np base = pd.read_csv('credit_data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit_transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.cross validation import train test split previsores_treinamento, previsores_teste, classe_treinamento, classe teste = from sklearn.model selection import train test split train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.25, random state=0) from sklearn.ensemble import RandomForestClassifier classificador = RandomForestClassifier(n estimators=40, criterion='entropy', from sklearn.ensemble import RandomForestClassifier classificador = RandomForestClassifier(n estimators=40, criterion='entropy', random state=0) random state=0) classificador.fit(previsores_treinamento, classe_treinamento) classificador.fit(previsores_treinamento, classe_treinamento) previsoes = classificador.predict(previsores teste) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion_matrix, accuracy_score from sklearn.metrics import confusion_matrix, accuracy_score precisao = accuracy score(classe teste, previsoes) precisao = accuracy score(classe teste, previsoes) matriz = confusion_matrix(classe_teste, previsoes) matriz = confusion matrix(classe teste, previsoes)

Random forest com scikit-learn - base censo random forest census.py import pandas as pd import pandas as pd base = pd.read csv('census.csv') base = pd.read csv('census.csv') previsores = base.iloc[:, 0:14].values previsores = base.iloc[:, 0:14].values classe = base.iloc[:, 14].values classe = base.iloc[:, 14].values from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.preprocessing import LabelEncoder, OneHotEncoder labelencoder previsores = LabelEncoder() from sklearn.compose import ColumnTransformer previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) column tranformer = ColumnTransformer([('one hot encoder', OneHotEncoder(), previsores[:, 5] = labelencoder previsores.fit transform(previsores[:, 5]) [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough') previsores[:, 6] = labelencoder previsores.fit transform(previsores[:, 6]) previsores = column tranformer.fit transform(previsores).toarray() previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8]) labelencoder_classe = LabelEncoder() classe = labelencoder classe.fit transform(classe) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) from sklearn.preprocessing import StandardScaler onehotencoder = OneHotEncoder(categorical features = [1,3,5,6,7,8,9,13]) scaler = StandardScaler() previsores = onehotencoder.fit transform(previsores).toarray() previsores = scaler.fit transform(previsores) labelencoder classe = LabelEncoder() from sklearn.model selection import train test split classe = labelencoder classe.fit transform(classe) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.15, random state=0) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.ensemble import RandomForestClassifier previsores = scaler.fit transform(previsores) classificador = RandomForestClassifier(n estimators=40, criterion='entropy', random state=0) classificador.fit(previsores_treinamento, classe_treinamento) from sklearn.cross_validation import train_test_split previsoes = classificador.predict(previsores teste) previsores treinamento, previsores teste, classe treinamento, classe teste = train_test_split(previsores, classe, test_size=0.15, random_state=0) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy score(classe teste, previsoes) from sklearn.ensemble import RandomForestClassifier matriz = confusion matrix(classe teste, previsoes)" classificador = RandomForestClassifier(n estimators=40, criterion='entropy', random state classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) matriz = confusion_matrix(classe_teste, previsoes)

kNN com scikit-learn - base crédito knn credit data.py import pandas as pd import pandas as pd import numpy as np base = pd.read_csv('credit_data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = imputer.fit(previsores[:, 1:4]) imputer = SimpleImputer(missing values=np.nan, strategy='mean') previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.cross validation import train test split previsores treinamento, previsores teste, classe treinamento, from sklearn.model selection import train test split classe teste = train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.25, random state=0) from sklearn.neighbors import KNeighborsClassifier classificador = KNeighborsClassifier(n neighbors=5, metric='minkowski', p = 2) from sklearn.neighbors import KNeighborsClassifier classificador.fit(previsores_treinamento, classe_treinamento) classificador = KNeighborsClassifier(n neighbors=5, metric='minkowski', p = 2) previsoes = classificador.predict(previsores_teste) classificador.fit(previsores_treinamento, classe_treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy score(classe teste, previsoes) from sklearn.metrics import confusion matrix, accuracy score matriz = confusion_matrix(classe_teste, previsoes) precisao = accuracy score(classe teste, previsoes) matriz = confusion_matrix(classe_teste, previsoes) import collections collections.Counter(classe teste) import collections

collections.Counter(classe teste)

Regressão logística com scikit-learn - base censo regressao logistica census.py import pandas as pd import pandas as pd base = pd.read csv('census.csv') base = pd.read csv('census.csv') previsores = base.iloc[:, 0:14].values previsores = base.iloc[:, 0:14].values classe = base.iloc[:, 14].values classe = base.iloc[:, 14].values from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.preprocessing import LabelEncoder, OneHotEncoder labelencoder previsores = LabelEncoder() from sklearn.compose import ColumnTransformer previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) labelencoder previsores = LabelEncoder() previsores[:, 5] = labelencoder previsores.fit transform(previsores[:, 5]) previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 6] = labelencoder previsores.fit transform(previsores[:, 6]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 5] = labelencoder previsores.fit transform(previsores[:, 5]) previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8]) previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6]) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) previsores[:, 8] = labelencoder previsores.fit transform(previsores[:, 8]) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13]) previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) previsores = onehotencoder.fit transform(previsores).toarray() column tranformer = ColumnTransformer([('one hot encoder', OneHotEncoder(), labelencoder classe = LabelEncoder() [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough') classe = labelencoder classe.fit transform(classe) previsores = column tranformer.fit transform(previsores).toarray() from sklearn.preprocessing import StandardScaler from sklearn.preprocessing import StandardScaler scaler = StandardScaler() scaler = StandardScaler() previsores = scaler.fit transform(previsores) previsores = scaler.fit transform(previsores) from sklearn.cross validation import train test split from sklearn.model selection import train test split previsores treinamento, previsores teste, classe treinamento, classe teste = previsores treinamento, previsores teste, classe treinamento, train test split(previsores, classe, test size=0.15, random state=0) classe teste = train test split(previsores, classe, test size=0.15, random state=0) from sklearn.linear model import LogisticRegression from sklearn.linear model import LogisticRegression classificador = LogisticRegression() classificador = LogisticRegression() classificador.fit(previsores treinamento, classe treinamento) classificador.fit(previsores treinamento, classe treinamento) previsoes = classificador.predict(previsores teste) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) precisao = accuracy_score(classe_teste, previsoes) matriz = confusion matrix(classe teste, previsoes) matriz = confusion matrix(classe teste, previsoes) import collections import collections collections.Counter(classe teste) collections.Counter(classe teste)

SVM com scikit-learn - base crédito svc credit data.py 'import pandas as pd import pandas as pd import numpy as np base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92 previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = imputer.fit(previsores[:, 1:4]) imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.cross validation import train test split previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = from sklearn.model selection import train test split train test split(previsores, classe, test size=0.25, random state=0) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.25, random state=0 from sklearn.svm import SVC classificador = SVC(kernel = 'rbf', random state = 1, C = 2.0) from sklearn.svm import SVC classificador.fit(previsores treinamento, classe treinamento) classificador = SVC(kernel = 'rbf', random_state = 1, C = 2.0, gamma='auto') previsoes = classificador.predict(previsores teste) classificador.fit(previsores_treinamento, classe_treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy_score(classe_teste, previsoes) from sklearn.metrics import confusion matrix, accuracy score matriz = confusion matrix(classe teste, previsoes) precisao = accuracy score(classe teste, previsoes) matriz = confusion_matrix(classe_teste, previsoes) import collections collections.Counter(classe teste)" import collections collections.Counter(classe teste)

Redes neurais com scikit-learn - base crédito

redes_neurais_credit_data.py

```
import pandas as pd
base = pd.read csv('credit data.csv')
base.loc[base.age < 0, 'age'] = 40.92
previsores = base.iloc[:, 1:4].values
classe = base.iloc[:, 4].values
from sklearn.preprocessing import Imputer
imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(previsores[:, 1:4])
previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit transform(previsores)
from sklearn.model selection import train test split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste =
                    train test split(previsores, classe, test size=0.25, random state=0)
from sklearn.neural network import MLPClassifier
classificador = MLPClassifier(verbose = True,
                              max iter=1000,
                              tol = 0.0000010.
                              solver = 'adam',
                              hidden laver sizes=(100).
                              activation='relu')
classificador.fit(previsores treinamento, classe treinamento)
previsoes = classificador.predict(previsores teste)
from sklearn.metrics import confusion matrix, accuracy score
precisao = accuracy score(classe teste, previsoes)
matriz = confusion matrix(classe teste, previsoes)"
```

```
import pandas as pd
import numpy as np
base = pd.read_csv('credit_data.csv')
base.loc[base.age < 0, 'age'] = 40.92
previsores = base.iloc[:, 1:4].values
classe = base.iloc[:, 4].values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing values = np.nan, strategy = 'mean')
imputer = imputer.fit(previsores[:, 1:4])
previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit transform(previsores)
from sklearn.model selection import train test split
previsores treinamento, previsores teste, classe treinamento, classe teste =
                 train test split(previsores, classe, test size=0.25, random state=0)
from sklearn.neural network import MLPClassifier
classificador = MLPClassifier()
classificador = MLPClassifier(verbose = True,
                              max iter=1000.
                              tol = 0.0000010,
                              solver = 'adam',
                              hidden layer sizes=(100),
                              activation='relu')
classificador.fit(previsores treinamento, classe treinamento)
previsoes = classificador.predict(previsores teste)
from sklearn.metrics import confusion matrix, accuracy score
precisao = accuracy score(classe teste, previsoes)
matriz = confusion matrix(classe teste, previsoes)"
```

Redes neurais com scikit-learn - base censo redes neurais census.py import pandas as pd import pandas as pd base = pd.read csv('census.csv') base = pd.read csv('census.csv') previsores = base.iloc[:, 0:14].values previsores = base.iloc[:, 0:14].values classe = base.iloc[:, 14].values classe = base.iloc[:, 14].values from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.preprocessing import LabelEncoder, OneHotEncoder labelencoder previsores = LabelEncoder() from sklearn.compose import ColumnTransformer previsores[:, 1] = labelencoder previsores.fit transform(previsores[:, 1]) previsores[:, 3] = labelencoder previsores.fit transform(previsores[:, 3]) column tranformer = ColumnTransformer([('one hot encoder', previsores[:, 5] = labelencoder previsores.fit transform(previsores[:, 5]) OneHotEncoder(), [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough') previsores[:, 6] = labelencoder previsores.fit transform(previsores[:, 6]) previsores = column tranformer.fit transform(previsores).toarray() previsores[:, 7] = labelencoder previsores.fit transform(previsores[:, 7]) previsores[:, 8] = labelencoder previsores.fit transform(previsores[:, 8]) previsores[:, 9] = labelencoder previsores.fit transform(previsores[:, 9]) labelencoder classe = LabelEncoder() previsores[:, 13] = labelencoder previsores.fit transform(previsores[:, 13]) classe = labelencoder classe.fit transform(classe) onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13]) from sklearn.preprocessing import StandardScaler previsores = onehotencoder.fit transform(previsores).toarray() scaler = StandardScaler() previsores = scaler.fit transform(previsores) labelencoder classe = LabelEncoder() classe = labelencoder classe.fit transform(classe) from sklearn.model selection import train test split previsores treinamento, previsores teste, classe treinamento, from sklearn.preprocessing import StandardScaler classe teste = train test split(previsores, classe, test size=0.15, random state=0) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.neural network import MLPClassifier classificador = MLPClassifier(verbose = True, max iter=1000, tol=0.000010) from sklearn.model selection import train test split print(classificador.out activation) classificador.fit(previsores treinamento, classe treinamento) previsores treinamento, previsores teste, classe treinamento, classe teste = train test split(previsores, classe, test size=0.15, random state=0) previsoes = classificador.predict(previsores teste) from sklearn.neural network import MLPClassifier from sklearn.metrics import confusion matrix, accuracy score classificador = MLPClassifier(verbose = True, max iter=1000, tol=0.000010) precisao = accuracy score(classe teste, previsoes) print(classificador.out activation) matriz = confusion matrix(classe teste, previsoes)" classificador.fit(previsores_treinamento, classe_treinamento) previsoes = classificador.predict(previsores teste) from sklearn.metrics import confusion matrix, accuracy score precisao = accuracy score(classe teste, previsoes)

matriz = confusion_matrix(classe_teste, previsoes)

Validação cruzada - stratifiedkfold

validação cruzada stratifiedkfold.pv import pandas as pd base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92import pandas as pd import numpy as np previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values a = np.zeros(5)from sklearn.preprocessing import Imputer base = pd.read csv('credit data.csv') imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) base.loc[base.age < 0, 'age'] = 40.92 imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.impute import SimpleImputer previsores = scaler.fit transform(previsores) imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') imputer = imputer.fit(previsores[:, 1:4]) from sklearn.naive bayes import GaussianNB previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) import numpy as np from sklearn.preprocessing import StandardScaler a = np.zeros(5)scaler = StandardScaler() previsores.shape previsores = scaler.fit transform(previsores) previsores.shape[0] b = np.zeros(shape=(previsores.shape[0], 1)) from sklearn.naive bayes import GaussianNB previsores.shape from sklearn.model selection import StratifiedKFold previsores.shape[0] from sklearn.metrics import accuracy_score, confusion_matrix b = np.zeros(shape=(previsores.shape[0],1)) kfold = StratifiedKFold(n splits = 10, shuffle = True, random state = 3) resultados = [] from sklearn.model selection import StratifiedKFold matrizes = [] from sklearn.metrics import accuracy score for indice treinamento, indice teste in kfold.split(previsores, kfold = StratifiedKFold(n splits = 10, shuffle = True, random state = 0) np.zeros(shape=(previsores.shape[0], resultados = [] #print('Índice treinamento: ', indice_treinamento, 'Índice teste: ', indice_teste) for indice treinamento, indice teste in kfold.split(previsores, np.zeros(shape=(classificador = GaussianNB() previsores.shape[0],1))): classificador.fit(previsores[indice_treinamento], classe[indice_treinamento]) #print('Indice treinamento: ', indice treinamento, 'Indice teste: ', indice teste) previsoes = classificador.predict(previsores[indice teste]) classificador = GaussianNB() precisao = accuracy score(classe[indice teste], previsoes) classificador.fit(previsores[indice_treinamento], classe[indice_treinamento])

resultados.mean()

resultados.std()

resultados.append(precisao)

matriz_final = np.mean(matrizes, axis = 0)
resultados = np.asarray(resultados)

matrizes.append(confusion_matrix(classe[indice_teste], previsoes))

previsoes = classificador.predict(previsores[indice teste])

precisao = accuracy score(classe[indice teste], previsoes)

resultados.append(precisao)

resultados = np.array(resultados)

resultados.mean()

resultados.std()

Salvar um classificador já treinado salvar classificador.pv import pandas as pd import pandas as pd import numpy as np base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92base = pd.read csv('credit data.csv') base.loc[base.age < 0, 'age'] = 40.92previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values previsores = base.iloc[:, 1:4].values classe = base.iloc[:, 4].values from sklearn.preprocessing import Imputer imputer = Imputer(missing values = 'NaN', strategy = 'mean', axis = 0) from sklearn.impute import SimpleImputer imputer = SimpleImputer(missing values = np.nan, strategy = 'mean') imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) imputer = imputer.fit(previsores[:, 1:4]) previsores[:, 1:4] = imputer.transform(previsores[:, 1:4]) from sklearn.preprocessing import StandardScaler scaler = StandardScaler() from sklearn.preprocessing import StandardScaler previsores = scaler.fit transform(previsores) scaler = StandardScaler() previsores = scaler.fit transform(previsores) from sklearn.svm import SVC from sklearn.ensemble import RandomForestClassifier from sklearn.svm import SVC from sklearn.neural network import MLPClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.neural network import MLPClassifier classificadorSVM = SVC(kernel = 'rbf', C = 2.0) classificadorSVM.fit(previsores, classe) classificadorSVM = SVC(kernel = 'rbf', C = 2.0) classificadorSVM.fit(previsores, classe) classificadorRandomForest = RandomForestClassifier(n estimators = 40, criterion = 'entropy') classificadorRandomForest = RandomForestClassifier(n estimators = 40, classificadorRandomForest.fit(previsores, classe) criterion = 'entropy') classificadorRandomForest.fit(previsores, classe) classificadorMLP = MLPClassifier(verbose = True, max iter = 1000, tol = 0.000010, solver = 'adam', classificadorMLP = MLPClassifier(verbose = True, max_iter = 1000, hidden layer sizes=(100), activation = 'relu', tol = 0.000010, solver = 'adam', batch size = 200, learning rate init = 0.001) hidden layer sizes=(100), activation = 'relu', classificadorMLP.fit(previsores, classe) batch size = 200, learning rate init = 0.001) classificadorMLP.fit(previsores, classe) import pickle pickle.dump(classificadorSVM, open('svm finalizado.sav', 'wb')) import pickle pickle.dump(classificadorRandomForest, open('random forest finalizado.sav', 'wb')) pickle.dump(classificadorSVM, open('svm finalizado.sav', 'wb')) pickle.dump(classificadorMLP, open('mlp finalizado.sav', 'wb')) pickle.dump(classificadorRandomForest, open('random forest finalizado.sav', 'wb'))

pickle.dump(classificadorMLP, open('mlp_finalizado.sav', 'wb'))"

```
Carregar um classificador já treinado
                                                                                  carregar classificador.py
import pandas as pd
                                                                                           import pandas as pd
import pickle
                                                                                           import pickle
from sklearn.preprocessing import Imputer
                                                                                           from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
                                                                                           from sklearn.preprocessing import StandardScaler
import numpy as np
                                                                                           import numpy as np
base = pd.read csv('credit data.csv')
                                                                                           base = pd.read csv('credit data.csv')
base.loc[base.age < 0, 'age'] = 40.92
                                                                                           base.loc[base.age < 0, 'age'] = 40.92
previsores = base.iloc[:, 1:4].values
                                                                                           previsores = base.iloc[:, 1:4].values
classe = base.iloc[:, 4].values
                                                                                           classe = base.iloc[:, 4].values
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
                                                                                           imputer = SimpleImputer(missing values = np.nan, strategy = 'mean')
imputer = imputer.fit(previsores[:, 1:4])
                                                                                           imputer = imputer.fit(previsores[:, 1:4])
                                                                                           previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
scaler = StandardScaler()
                                                                                           scaler = StandardScaler()
previsores = scaler.fit transform(previsores)
                                                                                           previsores = scaler.fit transform(previsores)
svm = pickle.load(open('svm finalizado.sav', 'rb'))
                                                                                           svm = pickle.load(open('svm finalizado.sav', 'rb'))
random forest = pickle.load(open('random forest finalizado.sav', 'rb'))
                                                                                           random forest = pickle.load(open('random forest finalizado.sav', 'rb'))
mlp = pickle.load(open('mlp finalizado.sav', 'rb'))
                                                                                           mlp = pickle.load(open('mlp finalizado.sav', 'rb'))
resultado svm = svm.score(previsores, classe)
                                                                                           resultado svm = svm.score(previsores, classe)
resultado random forest = random forest.score(previsores, classe)
                                                                                           resultado random forest = random forest.score(previsores, classe)
resultado mlp = mlp.score(previsores, classe)
                                                                                           resultado mlp = mlp.score(previsores, classe)
novo registro = [[50000, 40, 5000]]
                                                                                           novo registro = [[50000, 40, 5000]]
novo registro = np.asarray(novo registro)
                                                                                           novo registro = np.asarray(novo registro)
novo registro = novo registro.reshape(-1, 1)
                                                                                           novo registro = novo registro.reshape(-1, 1)
novo_registro = scaler.fit_transform(novo_registro)
                                                                                           novo_registro = scaler.fit_transform(novo_registro)
novo registro = novo registro.reshape(-1, 3)
                                                                                           novo registro = novo registro.reshape(-1, 3)
resposta svm = svm.predict(novo registro)
                                                                                           resposta svm = svm.predict(novo registro)
resposta random forest = random forest.predict(novo registro)
                                                                                           resposta random forest = random forest.predict(novo registro)
resposta mlp = mlp.predict(novo registro)
                                                                                           resposta mlp = mlp.predict(novo registro)
```

Regressão linear simples - base plano saúde II	
regressao_linear_plano_saude1.py	
<pre>previsao1 = regressor.predict(40)</pre>	<pre>previsao1 = regressor.predict(np.array(40).reshape(1, -1))</pre>

```
Regressão polinomial x linear - base plano saúde

regressao_polinomial_linear_plano_saude.py

import numpy as np

regressor1.predict(40)
--
regressor2.predict(poly.transform(40))

regressor2.predict(poly.transform(np.array(40).reshape(1, -1))
---
regressor2.predict(poly.transform(np.array(40).reshape(1, -1))
```

```
Regressão com árvores de decisão - base plano saúde
                                                                                regressao arvore plano saude.pv
import pandas as pd
                                                                                             import pandas as pd
base = pd.read csv('plano saude2.csv')
                                                                                             base = pd.read csv('plano saude2.csv')
X = base.iloc[:, 0:1].values
                                                                                             X = base.iloc[:, 0:1].values
y = base.iloc[:, 1].values
                                                                                             y = base.iloc[:, 1].values
from sklearn.preprocessing import StandardScaler
                                                                                             from sklearn.tree import DecisionTreeRegressor
scaler x = StandardScaler()
                                                                                             regressor = DecisionTreeRegressor()
                                                                                             regressor.fit(X, y)
X = scaler x.fit transform(X)
scaler y = StandardScaler()
                                                                                             score = regressor.score(X, y)
y = scaler_y.fit_transform(y)
import matplotlib.pyplot as plt
                                                                                             import matplotlib.pyplot as plt
plt.scatter(X, y)
                                                                                             plt.scatter(X, y)
plt.plot(X, regressor.predict(X), color = 'red')
                                                                                             plt.plot(X, regressor.predict(X), color = 'red')
plt.title('Regressão com redes neurais')
                                                                                             plt.title('Regressão com árvores')
plt.xlabel('Idade')
                                                                                             plt.xlabel('Idade')
plt.ylabel('Custo')
                                                                                             plt.vlabel('Custo')
import bumpy as np
                                                                                             import numpy as np
X_{\text{teste}} = \text{np.arange}(\min(X), \max(X), 0.1)
                                                                                             X_{\text{teste}} = \text{np.arange}(\min(X), \max(X), 0.1)
X teste = X teste.reshape(-1,1)
                                                                                             X teste = X teste.reshape(-1,1)
plt.scatter(X, v)
                                                                                             plt.scatter(X, v)
plt.plot(X_teste, regressor.predict(X_teste), color = 'red')
                                                                                             plt.plot(X_teste, regressor.predict(X_teste), color = 'red')
plt.title('Regressão com redes neurais')
                                                                                             plt.title('Regressão com árvores')
plt.xlabel('Idade')
                                                                                             plt.xlabel('Idade')
plt.ylabel('Custo')
                                                                                             plt.ylabel('Custo')
regressor.predict(40)
                                                                                             regressor.predict([[40]])
```

Regressão com vetores de suporte - base plano saúde regressao svr plano saude.py import pandas as pd import numpy as np import pandas as pd base = pd.read_csv('plano_saude2.csv') base = pd.read csv('plano saude2.csv') X = base.iloc[:, 0:1].values y = base.iloc[:, 1:2].values X = base.iloc[:, 0:1].values y = base.iloc[:, 1:2].values # kernel linear from sklearn.svm import SVR # kernel linear regressor linear = SVR(kernel = 'linear') from sklearn.svm import SVR regressor linear.fit(X, y) regressor linear = SVR(kernel = 'linear') regressor linear.fit(X, y) import matplotlib.pyplot as plt plt.scatter(X, y) import matplotlib.pvplot as plt plt.plot(X, regressor linear.predict(X), color = 'red') plt.scatter(X, y) regressor_linear.score(X, y) plt.plot(X, regressor linear.predict(X), color = 'red') # kernel poly regressor_linear.score(X, y) regressor_poly = SVR(kernel = 'poly', degree = 3, gamma = 'auto') # kernel poly regressor poly.fit(X, y) regressor_poly = SVR(kernel = 'poly', degree = 3) regressor_poly.fit(X, y) plt.scatter(X, y) plt.plot(X, regressor poly.predict(X), color = 'red') plt.scatter(X, y) regressor poly.score(X, y) plt.plot(X, regressor_poly.predict(X), color = 'red') # kernel rbf regressor poly.score(X, y) from sklearn.preprocessing import StandardScaler # kernel rhf scaler x = StandardScaler() from sklearn.preprocessing import StandardScaler X = scaler x.fit transform(X) scaler x = StandardScaler() scaler y = StandardScaler() X = scaler x.fit transform(X) y = scaler y.fit transform(y) scaler y = StandardScaler() y = scaler y.fit transform(y) regressor rbf = SVR(kernel = 'rbf', gamma = 'auto') regressor_rbf.fit(X, y) regressor rbf = SVR(kernel = 'rbf') regressor rbf.fit(X, v) plt.scatter(X, v) plt.plot(X, regressor rbf.predict(X), color = 'red') plt.scatter(X, y) regressor_rbf.score(X, y) plt.plot(X, regressor rbf.predict(X), color = 'red') regressor_rbf.score(X, y) previsao1 = scaler_y.inverse_transform(regressor_linear.predict(previsao1 = scaler y.inverse transform(regressor linear.predict(scaler x.transform(40))) scaler x.transform(np.array(40).reshape(1, -1)))) previsao2 = scaler y.inverse transform(regressor poly.predict(scaler x.transform(40))) previsao2 = scaler_y.inverse_transform(regressor_poly.predict(previsao3 = scaler y.inverse transform(regressor rbf.predict(scaler x.transform(40))) scaler_x.transform(np.array(40).reshape(1, -1)))) |previsao3 = scaler_y.inverse_transform(regressor_rbf.predict()) scaler_x.transform(np.array(40).reshape(1, -1))))

Regressão com redes neurais - base plano saúde regressao rna plano-saude.py import pandas as pd import pandas as pd import numpy as np base = pd.read csv('plano saude2.csv') base = pd.read_csv('plano_saude2.csv') X = base.iloc[:, 0:1].values v = base.iloc[:, 1:2].values X = base.iloc[:, 0:1].values y = base.iloc[:, 1:2].values from sklearn.preprocessing import StandardScaler scaler x = StandardScaler() X = scaler x.fit transform(X) from sklearn.preprocessing import StandardScaler scaler y = StandardScaler() scaler x = StandardScaler() v = scaler v.fit transform(v) X = scaler x.fit transform(X) scaler v = StandardScaler() from sklearn.neural network import MLPRegressor y = scaler y.fit transform(y) regressor = MLPRegressor() from sklearn.neural_network import MLPRegressor regressor.fit(X, y) regressor = MLPRegressor() regressor.score(X, y) regressor.fit(X, y) import matplotlib.pyplot as plt regressor.score(X, y) plt.scatter(X, y) plt.plot(X, regressor.predict(X), color = 'red') import matplotlib.pyplot as plt plt.title('Regressão com redes neurais') plt.scatter(X, v) plt.xlabel('Idade') plt.plot(X, regressor.predict(X), color = 'red') plt.ylabel('Custo') plt.title('Regressão com redes neurais') plt.xlabel('Idade') previsao = scaler y.inverse transform(regressor.predict(scaler x.transform(40))) plt.ylabel('Custo') previsao = scaler_y.inverse_transform(regressor.predict(scaler_x.transform(np.array(40).reshape(1, -1))))"