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

Jones Granatyr

Versão que o curso foi gravado	Nova Versão
Anaconda 3.6	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
base = pd.read_csv('census.csv')
previsores = base.iloc[:,0:14].values
classe = base.iloc[:,14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencorder_previsores = LabelEncoder()

#labels = labelencorder_previsores.fit_transform(previsores[:,1])
previsores[:,1] = labelencorder_previsores.fit_transform(previsores[:,1])
previsores[:,3] = labelencorder_previsores.fit_transform(previsores[:,3])
previsores[:,5] = labelencorder_previsores.fit_transform(previsores[:,5])
previsores[:,6] = labelencorder_previsores.fit_transform(previsores[:,6])
previsores[:,7] = labelencorder_previsores.fit_transform(previsores[:,7])
previsores[:,8] = labelencorder_previsores.fit_transform(previsores[:,8])
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)
```

```
import pandas as pd

base = pd.read_csv('census.csv')
previsores = base.iloc[:,0:14].values
classe = base.iloc[:,14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

onehotencorder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(),
                                            [1,3,5,6,7,8,9,13])],remainder='passthrough')
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

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.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste =
                    train_test_split(previsores, classe, test_size=0.25, 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)
```

```
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)
```

Divisão das bases de dados em treinamento e teste

template_census.py

```
import pandas as pd

base = pd.read_csv('census.csv')
previsores = base.iloc[:,0:14].values
classe = base.iloc[:,14].values

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

onehotencorder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(), [1,3,5,6,7,8,
previsores = onehotencorder.fit_transform(previsores).toarray()

labelencorder_classe = LabelEncoder()
classe = labelencorder_classe.fit_transform(classe)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

from sklearn.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento,
             classe_teste = train_test_split(previsores, classe, test_size=0.15, random_s

# 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)
```

```
import pandas as pd

base = pd.read_csv('census.csv')
previsores = base.iloc[:,0:14].values
classe = base.iloc[:,14].values

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

column_tranformer = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
                                        [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)

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=0.15, random_state=0)
```

Naive bayes com scikit learn - base crédito

naive_bayes_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.cross_validation 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.naive_bayes import GaussianNB
classificador = GaussianNB()
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.naive_bayes import GaussianNB
classificador = GaussianNB()
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 censo

naive_bayes_census.py

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13])
previsores = onehotencoder.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

onehotencorder = ColumnTransformer(transformers=[("OneHot", OneHotEncoder(),
                                          [1,3,5,6,7,8,9,13])],remainder='passthrough')
previsores = onehotencorder.fit_transform(previsores).toarray()

labelencorder_classe = LabelEncoder()
classe = labelencorder_classe.fit_transform(classe)

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=0.15, random_state=0)

from sklearn.naive_bayes import GaussianNB
classificador = GaussianNB()
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)
```

Árvores de decisão com scikit-learn - base crédito

arvores_decisao_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..cross_validation 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.tree import DecisionTreeClassifier
classificador = DecisionTreeClassifier(criterion='entropy', random_state=0)
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.tree import DecisionTreeClassifier
classificador = DecisionTreeClassifier(criterion='entropy', random_state=0)
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)
```

Árvores de decisão com scikit-learn - base censo

arvores_decisao_census.py

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13])
previsores = onehotencoder.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

from sklearn.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste =
                train_test_split(previsores, classe, test_size=0.15, 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)

from sklearn.metrics import confusion_matrix, accuracy_score
precisao = accuracy_score(classe_teste, previsoes)
matriz = confusion_matrix(classe_teste, previsoes)
```

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

column_tranformer = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
                                        [1, 3, 5, 6, 7, 8, 9, 13])],remainder='passthrough')
previsores = column_tranformer.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

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=0.15, 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)

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

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.cross_validation 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.ensemble import RandomForestClassifier
classificador = RandomForestClassifier(n_estimators=40, criterion='entropy',
                                                                    random_state=0)
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.ensemble import RandomForestClassifier
classificador = RandomForestClassifier(n_estimators=40, criterion='entropy',
                                                                        random_state=0)
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)
```

Random forest com scikit-learn - base censo

random_forest_census.py

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13])
previsores = onehotencoder.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

from sklearn.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste =
                    train_test_split(previsores, classe, test_size=0.15, random_state=0)

from sklearn.ensemble import RandomForestClassifier
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)
```

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

column_tranformer = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
                                  [1, 3, 5, 6, 7, 8, 9, 13])],remainder='passthrough')
previsores = column_tranformer.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

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=0.15,
                                                                  random_state=0)

from sklearn.ensemble import RandomForestClassifier
classificador = RandomForestClassifier(n_estimators=40, criterion='entropy',
                                                                  random_state=0)
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

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.cross_validation 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.neighbors import KNeighborsClassifier
classificador = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p = 2)
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 collections
collections.Counter(classe_teste)
```

```
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.neighbors import KNeighborsClassifier
classificador = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p = 2)
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 collections
collections.Counter(classe_teste)
```

Regressão logística com scikit-learn - base crédito

regress-o-logistica-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.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste = train_test_sp

from sklearn.linear_model import LogisticRegression
classificador = LogisticRegression(random_state = 1)
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.linear_model import LogisticRegression
classificador = LogisticRegression(random_state = 1, solver='lbfgs')
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)
```

Regressão logística com scikit-learn - base censo

regressao_logistica_census.py

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13])
previsores = onehotencoder.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

from sklearn.cross_validation import train_test_split
previsores_treinamento, previsores_teste, classe_treinamento, classe_teste =
                    train_test_split(previsores, classe, test_size=0.15, random_state=0)

from sklearn.linear_model import LogisticRegression
classificador = LogisticRegression()
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 collections
collections.Counter(classe_teste)
```

```
import pandas as pd

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

column_tranformer = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
                                        [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)


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=0.15, random_state=0)

from sklearn.linear_model import LogisticRegression
classificador = LogisticRegression(solver='lbfgs')
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 collections
collections.Counter(classe_teste)
```

SVM com scikit-learn - base crédito	
svc_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.cross_validation 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.svm import SVC classificador = SVC(kernel = 'rbf', random_state = 1, C = 2.0) 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 collections collections.Counter(classe_teste)"	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.svm import SVC classificador = SVC(kernel = 'rbf', random_state = 1, C = 2.0, gamma='auto') 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 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_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)"
```

```
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

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_previsores = LabelEncoder()
previsores[:, 1] = labelencoder_previsores.fit_transform(previsores[:, 1])
previsores[:, 3] = labelencoder_previsores.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder_previsores.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder_previsores.fit_transform(previsores[:, 6])
previsores[:, 7] = labelencoder_previsores.fit_transform(previsores[:, 7])
previsores[:, 8] = labelencoder_previsores.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder_previsores.fit_transform(previsores[:, 9])
previsores[:, 13] = labelencoder_previsores.fit_transform(previsores[:, 13])

onehotencoder = OneHotEncoder(categorical_features = [1,3,5,6,7,8,9,13])
previsores = onehotencoder.fit_transform(previsores).toarray()

labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

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=0.15, random_state=0)

from sklearn.neural_network import MLPClassifier
classificador = MLPClassifier(verbose = True, max_iter=1000, tol=0.000010)
print(classificador.out_activation_)
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

base = pd.read_csv('census.csv')

previsores = base.iloc[:, 0:14].values
classe = base.iloc[:, 14].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

column_tranformer = ColumnTransformer([('one_hot_encoder',
              OneHotEncoder(), [1, 3, 5, 6, 7, 8, 9, 13])],remainder='passthrough')
previsores = column_tranformer.fit_transform(previsores).toarray()


labelencoder_classe = LabelEncoder()
classe = labelencoder_classe.fit_transform(classe)

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=0.15, random_state=0)

from sklearn.neural_network import MLPClassifier
classificador = MLPClassifier(verbose = True, max_iter=1000, tol=0.000010)
print(classificador.out_activation_)
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

validacao_cruzada_stratifiedkfold.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.naive_bayes import GaussianNB

import numpy as np
a = np.zeros(5)
previsores.shape
previsores.shape[0]
b = np.zeros(shape=(previsores.shape[0], 1))

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 3)
resultados = []
matrizes = []
for indice_treinamento, indice_teste in kfold.split(previsores,
                                                    np.zeros(shape=(previsores.shape[0], 1
    #print('Índice treinamento: ', indice_treinamento, 'Índice teste: ', indice_teste)
    classificador = GaussianNB()
    classificador.fit(previsores[indice_treinamento], classe[indice_treinamento])
    previsoes = classificador.predict(previsores[indice_teste])
    precisao = accuracy_score(classe[indice_teste], previsoes)
    matrizes.append(confusion_matrix(classe[indice_teste], previsoes))
    resultados.append(precisao)

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

```
import pandas as pd
import numpy as np

a = np.zeros(5)

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.naive_bayes import GaussianNB
previsores.shape
previsores.shape[0]
b = np.zeros(shape=(previsores.shape[0],1))

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 0)
resultados = []
for indice_treinamento, indice_teste in kfold.split(previsores, np.zeros(shape=(
                                                                    previsores.shape[0],1))):
    #print('Indice treinamento: ', indice_treinamento, 'Indice teste: ', indice_teste)
    classificador = GaussianNB()
    classificador.fit(previsores[indice_treinamento], classe[indice_treinamento])
    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.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.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

classificadorSVM = SVC(kernel = 'rbf', C = 2.0)
classificadorSVM.fit(previsores, classe)

classificadorRandomForest = RandomForestClassifier(n_estimators = 40,
                                                   criterion = 'entropy')
classificadorRandomForest.fit(previsores, classe)

classificadorMLP = MLPClassifier(verbose = True, max_iter = 1000,
                                 tol = 0.000010, solver = 'adam',
                                 hidden_layer_sizes=(100), activation = 'relu',
                                 batch_size = 200, learning_rate_init = 0.001)
classificadorMLP.fit(previsores, classe)

import pickle
pickle.dump(classificadorSVM, open('svm_finalizado.sav', 'wb'))
pickle.dump(classificadorRandomForest, open('random_forest_finalizado.sav', 'wb'))
pickle.dump(classificadorMLP, open('mlp_finalizado.sav', 'wb'))
```

```
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.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

classificadorSVM = SVC(kernel = 'rbf', C = 2.0)
classificadorSVM.fit(previsores, classe)

classificadorRandomForest = RandomForestClassifier(n_estimators = 40,
                                                   criterion = 'entropy')
classificadorRandomForest.fit(previsores, classe)

classificadorMLP = MLPClassifier(verbose = True, max_iter = 1000,
                                 tol = 0.000010, solver = 'adam',
                                 hidden_layer_sizes=(100), activation = 'relu',
                                 batch_size = 200, learning_rate_init = 0.001)
classificadorMLP.fit(previsores, classe)

import pickle
pickle.dump(classificadorSVM, open('svm_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 pickle
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler
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
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(previsores[:, 1:4])
previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

svm = pickle.load(open('svm_finalizado.sav', 'rb'))
random_forest = pickle.load(open('random_forest_finalizado.sav', 'rb'))
mlp = pickle.load(open('mlp_finalizado.sav', 'rb'))

resultado_svm = svm.score(previsores, classe)
resultado_random_forest = random_forest.score(previsores, classe)
resultado_mlp = mlp.score(previsores, classe)

novo_registro = [[50000, 40, 5000]]
novo_registro = np.asarray(novo_registro)
novo_registro = novo_registro.reshape(-1, 1)
novo_registro = scaler.fit_transform(novo_registro)
novo_registro = novo_registro.reshape(-1, 3)

resposta_svm = svm.predict(novo_registro)
resposta_random_forest = random_forest.predict(novo_registro)
resposta_mlp = mlp.predict(novo_registro)
```

```
import pandas as pd
import pickle
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
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
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer = imputer.fit(previsores[:, 1:4])
previsores[:, 1:4] = imputer.transform(previsores[:, 1:4])
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

svm = pickle.load(open('svm_finalizado.sav', 'rb'))
random_forest = pickle.load(open('random_forest_finalizado.sav', 'rb'))
mlp = pickle.load(open('mlp_finalizado.sav', 'rb'))

resultado_svm = svm.score(previsores, classe)
resultado_random_forest = random_forest.score(previsores, classe)
resultado_mlp = mlp.score(previsores, classe)

novo_registro = [[50000, 40, 5000]]
novo_registro = np.asarray(novo_registro)
novo_registro = novo_registro.reshape(-1, 1)
novo_registro = scaler.fit_transform(novo_registro)
novo_registro = novo_registro.reshape(-1, 3)

resposta_svm = svm.predict(novo_registro)
resposta_random_forest = random_forest.predict(novo_registro)
resposta_mlp = mlp.predict(novo_registro)
```

Regressão linear simples - base plano saúde II

regressao_linear_plano_saude1.py

```
previsao1 = regressor.predict(40)
```

```
previsao1 = regressor.predict(np.array(40).reshape(1, -1))
```

Regressão polinomial x linear - base plano saúde

regressao_polinomial_linear_plano_saude.py

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

```
import numpy as np

regressor1.predict(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.py

```
import pandas as pd

base = pd.read_csv('plano_saude2.csv')

X = base.iloc[:, 0:1].values
y = base.iloc[:, 1].values

from sklearn.preprocessing import StandardScaler
scaler_x = StandardScaler()
X = scaler_x.fit_transform(X)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)


import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor.predict(X), color = 'red')
plt.title('Regressão com redes neurais')
plt.xlabel('Idade')
plt.ylabel('Custo')

import bumpy as np
X_teste = np.arange(min(X), max(X), 0.1)
X_teste = X_teste.reshape(-1,1)
plt.scatter(X, y)
plt.plot(X_teste, regressor.predict(X_teste), color = 'red')
plt.title('Regressão com redes neurais')
plt.xlabel('Idade')
plt.ylabel('Custo')

regressor.predict(40)
```

```
import pandas as pd

base = pd.read_csv('plano_saude2.csv')

X = base.iloc[:, 0:1].values
y = base.iloc[:, 1].values

from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X, y)
score = regressor.score(X, y)



import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor.predict(X), color = 'red')
plt.title('Regressão com árvores')
plt.xlabel('Idade')
plt.ylabel('Custo')

import numpy as np
X_teste = np.arange(min(X), max(X), 0.1)
X_teste = X_teste.reshape(-1,1)
plt.scatter(X, y)
plt.plot(X_teste, regressor.predict(X_teste), color = 'red')
plt.title('Regressão com árvores')
plt.xlabel('Idade')
plt.ylabel('Custo')

regressor.predict([[40]])
```

Regressão com vetores de suporte - base plano saúde

regressao_svr_plano_saude.py

```
import pandas as pd

base = pd.read_csv('plano_saude2.csv')

X = base.iloc[:, 0:1].values
y = base.iloc[:, 1:2].values

# kernel linear
from sklearn.svm import SVR
regressor_linear = SVR(kernel = 'linear')
regressor_linear.fit(X, y)

import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor_linear.predict(X), color = 'red')
regressor_linear.score(X, y)

# kernel poly
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')
regressor_poly.score(X, y)

# kernel rbf
from sklearn.preprocessing import StandardScaler
scaler_x = StandardScaler()
X = scaler_x.fit_transform(X)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)

regressor_rbf = SVR(kernel = 'rbf')
regressor_rbf.fit(X, y)

plt.scatter(X, y)
plt.plot(X, regressor_rbf.predict(X), color = 'red')
regressor_rbf.score(X, y)

previsao1 = scaler_y.inverse_transform(regressor_linear.predict(scaler_x.transform(40)))
previsao2 = scaler_y.inverse_transform(regressor_poly.predict(scaler_x.transform(40)))
previsao3 = scaler_y.inverse_transform(regressor_rbf.predict(scaler_x.transform(40)))
```

```
import pandas as pd
import numpy as np

base = pd.read_csv('plano-saude2.csv')

X = base.iloc[:, 0:1].values
y = base.iloc[:, 1:2].values

# kernel linear
from sklearn.svm import SVR
regressor_linear = SVR(kernel = 'linear')
regressor_linear.fit(X, y.ravel())

import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor_linear.predict(X), color = 'red')
regressor_linear.score(X, y)

# kernel poly
regressor_poly = SVR(kernel = 'poly', degree = 3, gamma = 'auto')
regressor_poly.fit(X, y.ravel())

plt.scatter(X, y)
plt.plot(X, regressor_poly.predict(X), color = 'red')
regressor_poly.score(X, y)

# kernel rbf
from sklearn.preprocessing import StandardScaler
scaler_x = StandardScaler()
X = scaler_x.fit_transform(X)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)

regressor_rbf = SVR(kernel = 'rbf', gamma = 'auto')
regressor_rbf.fit(X, y.ravel())

plt.scatter(X, y)
plt.plot(X, regressor_rbf.predict(X), color = 'red')
regressor_rbf.score(X, y)

previsao1 = scaler_y.inverse_transform(regressor_linear.predict(
                                        scaler_x.transform(np.array(40).reshape(1, -1))))
previsao2 = scaler_y.inverse_transform(regressor_poly.predict(
                                        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

base = pd.read_csv('plano_saude2.csv')

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)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)

from sklearn.neural_network import MLPRegressor
regressor = MLPRegressor()
regressor.fit(X, y)

regressor.score(X, y)

import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor.predict(X), color = 'red')
plt.title('Regressão com redes neurais')
plt.xlabel('Idade')
plt.ylabel('Custo')

previsao = scaler_y.inverse_transform(regressor.predict(scaler_x.transform(40)))
```

```
import pandas as pd
import numpy as np

base = pd.read_csv('plano_saude2.csv')

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)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)

from sklearn.neural_network import MLPRegressor
regressor = MLPRegressor()
regressor.fit(X, y)

regressor.score(X, y)

import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.plot(X, regressor.predict(X), color = 'red')
plt.title('Regressão com redes neurais')
plt.xlabel('Idade')
plt.ylabel('Custo')

previsao = scaler_y.inverse_transform(regressor.predict(
                                        scaler_x.transform(np.array(40).reshape(1, -1))))
```

Instalação do R e do pacote TStools

Comandos-R.txt

```
MAIS INFORMAÇÕES

http://kourentzes.com/forecasting/2014/04/19/tstools-for-r/

INSTALAÇÃO

if (!require("devtools"))
    install.packages("devtools")
devtools::install_github("trnnick/TStools")

require(TStools)

dados <- read.csv("<path do arquivo>")
matriz <- as.matrix(dados)
TStools::nemenyi(matriz,conf.int=0.95, plottype="vline")
```

```
MAIS INFORMAÇÕES

http://kourentzes.com/forecasting/2014/04/19/tstools-for-r/

INSTALAÇÃO

if (!require("devtools"))
    install.packages("devtools")
devtools::install_github("trnnick/TStools", force = TRUE)

require(TStools)

dados <- read.csv("<path do arquivo>")
matriz <- as.matrix(dados)
tsutils::nemenyi(matriz,conf.int=0.95, plottype="vline")
```