Pivô

Modelo de classificação para verificar a se um cliente vai ou não cancelar o serviço no próximo mês

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Lendo os dados temporais

```
df_agosto = pd.read_csv('../data/Agosto/Ana Health_Tabela Modelo
Previsão Churn - Tabela até 08_23.csv', skiprows=1)
df_julho = pd.read_csv('../data/Julho/Ana Health_Tabela Modelo
Previsão Churn - Tabela até 07_23.csv', skiprows=1)
df_junho = pd.read_csv('../data/Junho/Ana Health_Tabela Modelo
Previsão Churn - Tabela até 06_23.csv', skiprows=1)
df_novembro = pd.read_csv('../data/Novembro/Ana Health_Tabela Modelo
Previsão Churn - Tabela Geral.csv', skiprows=1)
df_outubro = pd.read_csv('../data/Outubro/Ana Health_Tabela Modelo
Previsão Churn - Tabela até 10_23.csv', skiprows=1)
df_setembro = pd.read_csv('../data/Setembro/Ana Health_Tabela Modelo
Previsão Churn - Tabela até 09_23.csv', skiprows=1)
```

Fazendo o tratamento de cada um dos datasets via script

```
import script_dataframe
import importlib

importlib.reload(script_dataframe)
tratamento = script_dataframe.tratamento

df_agosto = tratamento(df_agosto)
df_julho = tratamento(df_julho)
df_junho = tratamento(df_junho)
df_novembro = tratamento(df_novembro)
df_outubro = tratamento(df_outubro)
df_setembro = tratamento(df_setembro)
```

Definindo a função para, dado um mês, retornar o dataset do mês seguinte

```

def prox_status(df1, df2):
    df2_novo = df2[df2['id_person'].isin(df1['id_person'].values)]
    return
pd.merge(df1,df2_novo[['id_person','status']],on='id_person',
```

```
how='left',suffixes=['','_prox_mes'])

df_junho = prox_status(df_junho, df_julho)
df_julho = prox_status(df_julho, df_agosto)
df_agosto = prox_status(df_agosto, df_setembro)
df_setembro = prox_status(df_setembro, df_outubro)
df_outubro  = prox_status(df_outubro, df_novembro)
```

Concatenando os datasets

```
df_total = pd.concat([df_junho, df_julho, df_agosto, df_setembro,
df_outubro])

df_total['Target'] = df_total['status_prox_mes'] == 'won'

df_total['Target'].value_counts()
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4680 entries, 0 to 1022
Data columns (total 54 columns):
 #   Column                                    Non-Null Count  Dtype

---  ------                                    --------------  -----

 0   id_person                                 4680 non-null   int64

 1   birthdate                                 4675 non-null
float64
 2   id_gender                                 4662 non-null
float64
 3   id_marrital_status                        4670 non-null
float64
 4   id_health_plan                            2204 non-null
float64
 5   contract_start_date                       4680 non-null
datetime64[ns]
 6   contract_end_date                         2292 non-null
datetime64[ns]
 7   id_continuity_pf                          1687 non-null
float64
 8   Canal de Preferência                      1473 non-null
float64
 9   notes_count                               4680 non-null   int64

 10  done_activities_count                     4680 non-null   int64

 11  status                                    4680 non-null
object
 12  start_of_service                          4680 non-null
```

```
float64
 13  lost_time                                2169 non-null
float64
 14  lost_reason                              2292 non-null
object
 15  add_time                                 4502 non-null
float64
 16  id_label                                 389 non-null
float64
 17  won_time                                 2890 non-null
float64
 18  lost_time.1                              1293 non-null
float64
 19  lost_reason.1                            1293 non-null
object
 20  Qde Todos Atendimentos                   4680 non-null   int64

 21  Faltas Todos Atendimento                 4680 non-null   int64

 22  Qde Atendimento Médico                   801 non-null
float64
 23  Faltas Atendimento Médico                801 non-null
float64
 24  Qde Atendimentos Acolhimento             2744 non-null
float64
 25  Faltas Acolhimento                       2744 non-null
float64
 26  Qde Psicoterapia                         1797 non-null
float64
 27  Faltas Psicoterapia                      587 non-null
object
 28  Físico                                   3562 non-null
float64
 29  Psicológico                              3562 non-null
float64
 30  Social                                   3562 non-null
float64
 31  Ambiental                                3562 non-null
float64
 32  Problemas Abertos                        2402 non-null
object
 33  Mensagens Inbound                        4367 non-null
float64
 34  Mensagens Outbound                       4606 non-null
float64
 35  Ligações Inbound                         430 non-null
float64
 36  Data Última Ligações Inbound             430 non-null
object
```

```
 37  Ligações Outbound                              2271 non-null
float64
 38  Data Última Ligações Outbound                  2271 non-null
object
 39  Qde Total de Faturas                           880 non-null
float64
 40  Qde Total de Tentativas de Cobrança            878 non-null
float64
 41  Método de Pagamento                            880 non-null
object
 42  Valor Médio da Mensalidade                     880 non-null
float64
 43  Qde Total de Faturas Pagas após Vencimento     880 non-null
float64
 44  Qde Total de Faturas Inadimpletes              880 non-null
float64
 45  Valor Total Inadimplência                      880 non-null
float64
 46  Qde Perfis de Pagamento Inativos               0 non-null
float64
 47  Tempo até Sair                                 2292 non-null
float64
 48  Tem Problema em Aberto                         4680 non-null   int64

 49  Tempo Última Mensagem Inbound                  4367 non-null
float64
 50  Tempo Última Mensagem Outbound                 4606 non-null
float64
 51  Quem Enviou Última Mensagem                    4680 non-null
object
 52  status_prox_mes                                4680 non-null
object
 53  Target                                         4680 non-null   bool

dtypes: bool(1), datetime64[ns](2), float64(35), int64(6), object(10)
memory usage: 1.9+ MB
```

Criando DataFrames -

Temos dois dataframes que pretendemos utilizar. O primeiro, e principal, é referente a análise de quem sai no próximo mës. O segundo, é uma tentativa de prever quanto tempo até o usuário sair.

DataSet 1 - Saída de Clientes no Próximo Mës

- Nele, temos uma coluna de ***target*** que é binária (valores verdadeiros indicam que a pessoa saiu no próxima mës). As colunas que contemplamos tem relação com o perfil do cliente, como idade, sexo, estado civil, etc. A maioria dos dados já passaram pelo script_dataframe (e a função `tratamento`) para adição de colunas novas, tratamento de datas e criação de dummies.

DataSet 2 - Tempo até o Cliente Sair

- Processo de criação foi parecida com o anterior, mas aqui temos uma coluna de ***target*** que é contínua (valores indicam quantos dias até o cliente sair). As colunas que contemplamos tem relação com o perfil do cliente, como idade, sexo, estado civil, etc. A maioria dos dados já passaram pelo script_dataframe (e a função `tratamento`) para adição de colunas novas, tratamento de datas e criação de dummies. A diferença é que contemplamos a colunas de faltas de atendimentos e consultas.

```
def transform_to_category(x,qtd_itens,lista_itens):
   for i in range(qtd_itens):
      if x == lista_itens[i]:
         return str(x)
   return 'Outros'

colunas_dropadas =
['id_person','contract_start_date','contract_end_date','id_continuity_
pf','Canal de
Preferência','status','lost_time','add_time','id_label','won_time','lo
st_time.1','lost_reason','lost_reason.1',\
                  'Qde Atendimento Médico','Faltas Atendimento
Médico',   'Qde Atendimentos Acolhimento',  'Faltas Acolhimento', 'Qde
Psicoterapia',   'Faltas Psicoterapia','Data Última Ligações
Outbound',\
                   'Data Última Ligações Inbound','Qde Total de
Faturas Pagas após Vencimento','Qde Perfis de Pagamento
Inativos','Tempo até Sair', 'Valor Médio da Mensalidade',
'status_prox_mes', 'Qde Total de Faturas','Problemas Abertos']


colunas_dropadas_regr =
['id_person','contract_start_date','contract_end_date','id_continuity_
pf','Canal de
Preferência','status','lost_time','add_time','id_label','won_time','lo
st_time.1','lost_reason','lost_reason.1',\
                   'Qde Atendimento Médico','Faltas Atendimento
Médico',   'Qde Atendimentos Acolhimento',  'Faltas Acolhimento', 'Qde
Psicoterapia',   'Faltas Psicoterapia','Data Última Ligações
Outbound',\
                   'Data Última Ligações Inbound','Qde Total de
Faturas Pagas após Vencimento','Qde Perfis de Pagamento Inativos',
'Valor Médio da Mensalidade', 'status_prox_mes', 'Qde Total de
Faturas', 'Target','Problemas Abertos']


colunas_get_dummies =
['id_gender','id_marrital_status','id_health_plan','notes_count','Méto
do de Pagamento','Qde Total de Faturas Inadimpletes', 'Quem Enviou
Última Mensagem']

#coluna de genero
```

```
df_total['id_gender'] = df_total['id_gender'] =
df_total['id_gender'].apply(lambda x: transform_to_category(x,2,
[63,64]))
#coluna de estado civil
#drop marrital status diferentes de 80,82,83
df_total = df_total[df_total['id_marrital_status'].isin([80,82,83])]
df_total['id_marrital_status'] =
df_total['id_marrital_status'].apply(lambda x:
transform_to_category(x,3,[80,82,83]))
#coluna de health plan
df_total['id_health_plan'] = df_total['id_health_plan'].apply(lambda
x: transform_to_category(x,4,[412,415,418,435]))
#coluna de notes count
df_total = df_total[df_total['notes_count'] < 7]


#coluna birthdate
df_total = df_total[df_total['birthdate'].notna()]
#coluna de fisico
df_total['Físico'] =
df_total['Físico'].fillna(df_total['Físico'].mean())
#coluna de psicologico
df_total['Psicológico'] =
df_total['Psicológico'].fillna(df_total['Psicológico'].mean())
#coluna de social
df_total['Social'] =
df_total['Social'].fillna(df_total['Social'].mean())
#coluna de ambiental
df_total['Ambiental'] =
df_total['Ambiental'].fillna(df_total['Ambiental'].mean())
#coluna de mensagens inbound
df_total['Mensagens Inbound'] = df_total['Mensagens
Inbound'].fillna(0)
#coluna de mensagens outbound
df_total['Mensagens Outbound'] = df_total['Mensagens
Outbound'].fillna(0)
#coluna de ligacoes inbound
df_total['Ligações Inbound'] = df_total['Ligações Inbound'].fillna(0)
#coluna de ligacoes outbound
df_total['Ligações Outbound'] = df_total['Ligações
Outbound'].fillna(0)
#coluna qtd tentativas de cobrança
df_total['Qde Total de Tentativas de Cobrança'] = df_total['Qde Total
de Tentativas de Cobrança'].fillna(0)
#coluna método de pagamento
df_total['Método de Pagamento'] = df_total['Método de
Pagamento'].apply(lambda x: transform_to_category(x,2,["Cartão de
crédito","Dinheiro"]))
#coluna total de faturas inadimplentes
```

```
df_total['Qde Total de Faturas Inadimpletes'] = df_total['Qde Total de
Faturas Inadimpletes'].fillna(0)
df_total['Qde Total de Faturas Inadimpletes'] = df_total['Qde Total de
Faturas Inadimpletes'].apply(lambda x: True if x > 0 else False)
#coluna valor total inadimplente
df_total['Valor Total Inadimplência'] = df_total['Valor Total
Inadimplência'].fillna(0)
#coluna Tempo Última Mensagem Inbound
df_total['Tempo Última Mensagem Inbound'] = df_total['Tempo Última
Mensagem Inbound'].fillna(0)
#coluna Tempo Última Mensagem Outbound
df_total['Tempo Última Mensagem Outbound'] = df_total['Tempo Última
Mensagem Outbound'].fillna(0)
#rename coluna birthdate para idade
df_total = df_total.rename(columns={'birthdate':'idade'})
# dropa linhas duplicadas

df_total = df_total.drop_duplicates()
df_regr = df_total.copy()



df_total = df_total[df_total['status'] != 'won']
df_total = df_total.drop(colunas_dropadas, axis=1)



df_regr = pd.get_dummies(df_regr, columns=colunas_get_dummies)

df_total = pd.get_dummies(df_total, columns=colunas_get_dummies)

df_regr_won =  df_regr[df_regr['status'] == 'won'] # Dataframe para
teste de regressão (Coerência)

df_regr = df_regr[df_regr['status'] != 'won'] # Dataframe para treino
de regressão


df_regr_won['Tempo até Sair'] =
df_regr_won['contract_start_date'].apply(lambda x:
(pd.to_datetime('today') - pd.to_datetime(x)).days)

df_regr = df_regr.drop(colunas_dropadas_regr, axis=1)

df_regr_won = df_regr_won.drop(colunas_dropadas_regr, axis=1)
```

Usando Decision Tree para classificar quem sai no próximo mês

```
#decision tree
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(df_total.drop('Target', axis=1), df_total['Target'],
test_size=0.2, random_state=42)

# Create the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_mat)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 1.0
Confusion Matrix:
[[137]]
Classification Report:
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       137

    accuracy                           1.00       137
   macro avg       1.00      1.00      1.00       137
weighted avg       1.00      1.00      1.00       137
```

Usando Random Forest para classificar quem sai no próximo mês

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
```

```
X_train, X_test, y_train, y_test =
train_test_split(df_total.drop('Target', axis=1), df_total['Target'],
test_size=0.2, random_state=42)

# Create the decision tree classifier
clf = RandomForestClassifier(random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_mat)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 1.0
Confusion Matrix:
[[137]]
Classification Report:
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       137

    accuracy                           1.00       137
   macro avg       1.00      1.00      1.00       137
weighted avg       1.00      1.00      1.00       137
```

Pivo Modelo Regressão

O tratamento dos dados foi feito nos notebooks anteriores, a diferença é a utilização do tempo do cliente até o cancelamento como variável target.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
from sklearn.ensemble import RandomForestRegressor

# Split the dataset into training and testing sets

y = df_regr['Tempo até Sair']
X = df_regr.drop(['Tempo até Sair'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42)  # Adjust test_size

# Create the regression model
model = RandomForestRegressor(random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the performance of the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Erro em dias de atraso:", np.sqrt(mse))
```

```
Mean Squared Error: 2438.936278260869
Erro em dias de atraso: 49.385587758584684
```

Teste de Coerência

Aqui realizamos um teste para verificar se o modelo está coerente com a realidade.

Para isso, utilizamos o dataset de pessoas que ainda estão ativas para tentar entender o comportamento do modelo ao prever o tempo de cancelamento e comparar com o tempo real que a pessoa está ativa.

```
# Usando os dados de clientes que ainda estão para comparar com os
dados de clientes que sairam apenas para testar o modelo
y = df_regr_won['Tempo até Sair']
X = df_regr_won.drop(['Tempo até Sair'], axis=1)
# Make predictions on the testing set
y_pred = model.predict(X)
# Evaluate the performance of the model
mse = mean_squared_error(y, y_pred)
print("Mean Squared Error:", mse)
print("Erro em dias de atraso:", np.sqrt(mse))

# Verificar se está errando para mais ou para menos
print('Erro médio:', np.mean(y_pred - y))
```

```
Mean Squared Error: 11862.394181586094
Erro em dias de atraso: 108.91461876895174
Erro médio: -82.33510592069527
```