

Modelo Previsão de Turnover

Turnover é o processo de um colaborador sair de uma organização.

Isso pode acarretar em aumentos de custos operacionais para as empresas, mudanças nas contratações e decisões de retenção.

```
In [48]: # Importar bibliotecas
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [49]: # Carregar CSV
df = pd.read_csv('turnover.csv')
```

```
In [50]: # Verificar o dataframe
df.head()
```

```
Out[50]: satisfaction evaluation number_of_projects average_montly_hours time_spend_company
0 0.38 0.53 2 157
1 0.80 0.86 5 262
2 0.11 0.88 7 272
3 0.72 0.87 5 223
4 0.37 0.52 2 159
```

```
In [51]: # Verificar informações das variáveis
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   satisfaction    14999 non-null   float64
 1   evaluation      14999 non-null   float64
 2   number_of_projects 14999 non-null   int64  
 3   average_montly_hours 14999 non-null   int64  
 4   time_spend_company 14999 non-null   int64  
 5   work_accident    14999 non-null   int64  
 6   churn            14999 non-null   int64  
 7   promotion         14999 non-null   int64  
 8   department        14999 non-null   object 
 9   salary            14999 non-null   object 
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

Observação de categorias

Percebe-se que as variáveis `salary` e `department` são variáveis categóricas.

Sendo que `salary` é categórica do tipo ordinal e `department` é nominal.

```
In [52]: # Verificar os valores das variáveis salary e department
print(df['department'].unique())
print('*'*80)
print(df['salary'].unique())
```

```
[sales' 'accounting' 'hr' 'technical' 'support' 'management' 'IT'
 'product_mng' 'marketing' 'RandD']
```

```
['low' 'medium' 'high']
```

Transformação de valores categóricos

```
In [53]: # Mudar o tipo de dado em salary para categórico
df.salary = df.salary.astype('category')

# Providenciar a ordem dessas categorias de salary
df.salary = df.salary.cat.reorder_categories(['low', 'medium', 'high'])

# Fazer encoding para as categorias receberem valores numéricos
df.salary = df.salary.cat.codes
```

```
In [54]: # Obter dummies e salvar em um novo dataframe
depart = pd.get_dummies(df.department)

# Visualizar esse novo dataframe
depart.head()
```

```
Out[54]:   IT  RandD  accounting    hr  management  marketing  product_mng  sales  su
0  False  False      False  False      False  False  False  False  True
1  False  False      False  False      False  False  False  False  True
2  False  False      False  False      False  False  False  False  True
3  False  False      False  False      False  False  False  False  True
4  False  False      False  False      False  False  False  False  True
```

```
In [55]: # Evitar "DUMMIE TRAP" -> para isso, deve-se retirar uma coluna
depart = depart.drop('accounting', axis=1)

# Retirar a coluna department do dataframe original
df = df.drop('department', axis=1)

# Juntar os dataframes
df = df.join(depart)
```

Estatística Descritiva

```
In [56]: # Número de funcionários
n_fun = len(df)

# Imprimir numero de colaboradores que saíram da empresa
print(df.churn.value_counts())

# Imprimir a porcentagem de colaboradores que saíram da empresa
print((df.churn.value_counts() / n_fun) * 100)
```

```
churn
0    11428
1     3571
Name: count, dtype: int64
churn
0    76.191746
1    23.808254
Name: count, dtype: float64
```

Dividir os dados para previsão

Nesse caso, vamos definir nosso alvo e features.

O alvo é `churn` ou turnover, sendo as features, todas as outras variáveis.

- Target / Variável Dependente = `churn`
- Features / Variáveis Independentes = `satisfaction evaluation number_of_projects average_montly_hours time_spend_company work_accident promotion department salary`
 - Sendo que todos os departamentos em `department` foram incluídos ao dataframe: ['sales' 'accounting' 'hr' 'technical' 'support' 'management' 'IT' 'product_mng' 'marketing' 'RandD']

Somado a isso, faremos a divisão dos dados para treinamento e teste do algoritmo.

```
In [58]: # Definir o alvo de previsão, ou variável dependente
target = df.churn

# Definir as features e retirar o nosso alvo das variáveis independentes
features = df.drop('churn', axis=1)

In [44]: # Importar biblioteca de seleção e separação dos dados para treinamento e teste
from sklearn.model_selection import train_test_split

# Função para separar os alvos e features
# Definir amostra de teste para 25%
target_train, target_test, features_train, features_test = train_test_split(
```

Classificador Árvore de Decisão

```
In [77]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Supondo que você já tenha lido os dados em um DataFrame chamado 'data'
# E a variável alvo está em uma Series chamada 'target'

# Exemplo de DataFrame e Series com dados fictícios
data = df.drop('churn', axis=1)
target = target = df.churn

# Separar as características (features) e o alvo (target)
features = data
target = target

# Dividir os dados em conjuntos de treino e teste
features_train, features_test, target_train, target_test = train_test_split()

# Inicializar o modelo especificando a semente aleatória
modelo = DecisionTreeClassifier(random_state=42)

# Aplicar o modelo para fazer o fit das variáveis independentes (features) à
modelo.fit(features_train, target_train)

# Fazer previsões no conjunto de teste
predictions = modelo.predict(features_test)

# Avaliar a acurácia do modelo
accuracy = accuracy_score(target_test, predictions)

# Mostrar a acurácia
print(f'Acurácia Teste: {accuracy:.2f}')
#-----

# Fazer previsões no conjunto de teste
predictions_train = modelo.predict(features_train)
```

```
# Avaliar a acurácia do modelo
accuracy_train = accuracy_score(target_train, predictions_train)

# Mostrar a acurácia
print(f'Acurácia Treino: {accuracy_train:.2f}')
```

Acurácia Teste: 0.98
Acurácia Treino: 1.00

Overfitting

O treinamento do modelo aparentemente está overfitted:

- Acurácia Teste: 0.98
- Acurácia Treino: 1.00

Esses valores significam que o modelo está muito otimizado para prever apenas baseado nesses dados. Ou seja, modelo está enviesado pelos dados. Caso utilizássemos novos dados, não conseguíramos uma previsão com boa acurácia.

Um modelo mais realístico, teria uma acurácia menor.

Plotagem

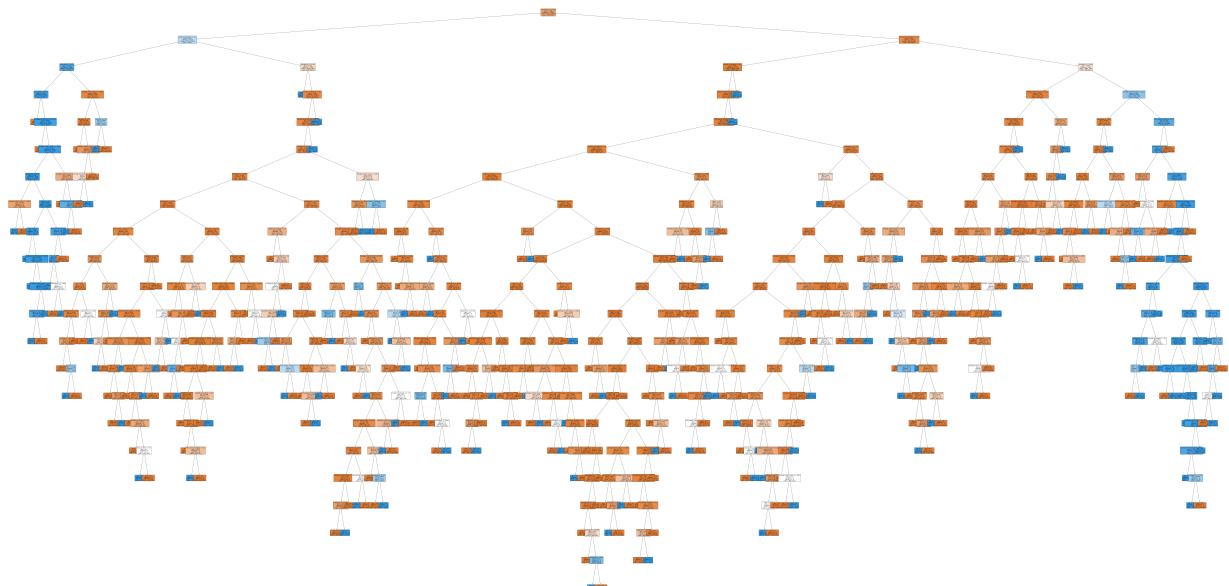
```
In [75]: # Importar visualizacao gráfica e função de exportacao
from sklearn.tree import export_graphviz

# Fazer fit do modelo
modelo.fit(features_train, target_train)

# Fazer exportacao do grafico da arvore para .dot (copiar o codigo do arquivo)
export_graphviz(modelo, "arvore_test.dot")
```

```
In [74]: # Ou podemos criar a visualizacao do gráfico aqui mesmo
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(200,100)) # Tamanho da figura
plot_tree(modelo, feature_names=features.columns, class_names=['Not Churn',
plt.show()
```



Corrigindo o overfitting

```
In [78]: # Initialize the DecisionTreeClassifier while limiting the depth of the tree
model_depth_5 = DecisionTreeClassifier(max_depth=5, random_state=42)

# Fit the model
model_depth_5.fit(features_train,target_train)

# Print the accuracy of the prediction for the training set
print(model_depth_5.score(features_train,target_train)*100)

# Print the accuracy of the prediction for the test set
print(model_depth_5.score(features_test,target_test)*100)
```

97.65813817818152
97.1333333333334

In []: