PROJETO 2

ANÁLISE DE DADOS DE RECURSOS HUMANOS

Em processos seletivos modernos, o departamento de Recursos Humanos (RH) enfrenta desafios na análise dos perfis de candidatos para identificar aqueles mais alinhados às necessidades da empresa.

O objetivo principal deste projeto é auxiliar o RH na organização e preparação dos dados sobre os candidatos, extraindo insights valiosos para embasar decisões futuras.

O dataset fornecido contém informações detalhadas sobre os candidatos, como:

- **Informações sobre a Cidade**: Localização e índice de desenvolvimento.
- **Dados Demográficos**: Gênero, nível educacional e área de formação.
- **Detalhes sobre o Emprego**: Experiência relevante, tamanho da empresa e tipo de empresa.
- **Padrões Comportamentais**: Tempo desde o último emprego e horas de treinamento realizadas.

O papel do analista de dados aqui é realizar a **exploração e preparação dos dados**, fornecendo recomendações e insights que servirão como insumos para equipes de Ciência de Dados ou Machine Learning.

Problema Proposto

Como o RH pode utilizar os dados disponíveis para identificar características comuns entre os melhores candidatos e otimizar seus processos de seleção?

Uma sugestão de passos a seguir :

1. **Compreender o Dataset**:
   - Explorar as variáveis disponíveis e entender quais são mais relevantes para a análise.
   - Tratar valores ausentes, inconsistências e variáveis irrelevantes.
2. **Análise Exploratória de Dados (EDA)**:
   - Identificar padrões e tendências nos dados.
   - Descobrir quais variáveis têm maior impacto no perfil de um bom candidato.
3. **Preparação dos Dados**:
   - Filtrar e transformar as variáveis para que estejam prontas para uso em um modelo de Machine Learning.
   - Documentar o processo de preparação para facilitar a transição para outras equipes.
4. **Geração de Insights e Recomendações**:
   - Com base na análise, destacar as variáveis mais relevantes para identificar bons candidatos
   - Informar quais variáveis são menos relevantes e podem ser descartadas
   - Realizar, ao final, as Recomendações do que foi identificado.

Análise de Dados de RH: Identificando Padrões dos Melhores Candidatos

Vamos realizar a análise de dados passo a passo, de forma detalhada, usando o dataset de Recursos Humanos fornecido.

O objetivo é identificar padrões e características dos **melhores candidatos** para otimizar processos seletivos.

Neste contexto, consideramos como "melhores candidatos" ***aqueles com maior probabilidade de estarem buscando novas oportunidades*** (indicados pela variável alvo target = 1, ou seja, candidatos inclinados a mudar de emprego).

1. Compreensão do Dataset

Carregando os dados: Primeiro, vamos carregar o dataset e inspecionar suas primeiras linhas e informações gerais. Isso nos ajuda a entender quais colunas existem, seus tipos de dados e como os dados estão organizados.

```
import pandas as pd

# Carregar o dataset (arquivo CSV fornecido)
df = pd.read_csv('aug_train.csv')

# Ver as primeiras linhas do conjunto de dados
print(df.head(5))  # Mostra as 5 primeiras linhas

# Obter informações gerais do dataset (colunas, tipos, contagem de valores não-nulos)
df.info()
```

Saída:

```
   enrollee_id      city  city_development_index gender  relevent_experience  ...  training_hours  target
0         8949  city_103                   0.920   Male   Has relevent experience  ...              36     1.0
1        29725   city_40                   0.776   Male   No relevent experience  ...              47     0.0
2        11561   city_21                   0.624    NaN   No relevent experience  ...              83     0.0
3        33241  city_115                   0.789    NaN   No relevent experience  ...              52     1.0
4          666  city_162                   0.767   Male   Has relevent experience  ...               8     0.0
```

Vemos que cada linha representa um **candidato** com várias informações (cidade, gênero, experiência, educação, etc.) e a variável target indica se ele busca mudar de emprego (1) ou não (0).

A chamada `df.info()` fornece um resumo das colunas, tipos e quantidade de dados não nulos em cada coluna:

```
RangeIndex: 19158 entries, 0 to 19157
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   enrollee_id             19158 non-null  int64
 1   city                    19158 non-null  object
 2   city_development_index  19158 non-null  float64
 3   gender                  14650 non-null  object
 4   relevent_experience     19158 non-null  object
 5   enrolled_university     18772 non-null  object
 6   education_level         18698 non-null  object
 7   major_discipline        16345 non-null  object
 8   experience              19093 non-null  object
```

```
 9   company_size            13220 non-null  object
 10  company_type            13018 non-null  object
 11  last_new_job            18735 non-null  object
 12  training_hours          19158 non-null  int64
 13  target                  19158 non-null  float64
dtypes: float64(2), int64(2), object(10)
```

Entendendo as colunas: De acordo com o dicionário de dados (Features.txt fornecido), podemos resumir as colunas principais:

- `enrollee_id`: ID único do candidato (apenas identificador).
- `city`: Código da cidade do candidato.
- `city_development_index`: Índice de desenvolvimento da cidade (quanto maior, mais desenvolvida, variando de 0.0 a 1.0).
- `gender`: Gênero do candidato (Male, Female, Other).
- `relevent_experience`: Se o candidato tem experiência de trabalho relevante na área (Has/No relevant experience).
- `enrolled_university`: Tipo de curso universitário em que o candidato está matriculado (no_enrollment, Full time course, Part time course).
- `education_level`: Nível de educação (High School, Graduate, Masters, Phd, etc).
- `major_discipline`: Área principal de formação (STEM, Humanities, etc).
- `experience`: Anos de experiência de trabalho do candidato (pode ser um número em anos, ou categorias como <1 para menos de um ano, >20 para mais de 20 anos).
- `company_size`: Tamanho da empresa atual/em que trabalhou (faixas como 50-99, 1000-4999, etc).
- `company_type`: Tipo da empresa (Public, Private (Pvt Ltd), Startup, etc).
- `last_new_job`: Tempo desde que o candidato trocou de emprego pela última vez (número de anos, ou 'never' se nunca trocou).
- `training_hours`: Horas de treinamento realizadas.
- `target`: **Variável alvo** indicando se o candidato está procurando mudar de emprego (1 = sim, 0 = não).

Identificando valores ausentes, duplicados e inconsistências:

- Pelos resultados do `df.info()`, notamos que várias colunas têm contagens de não-nulos menores que **19158**, indicando valores ausentes (`NaN`). Vamos quantificar os valores faltantes em cada coluna:

```
# Quantidade de valores ausentes por coluna
print(df.isnull().sum())
```

Saída (resumo de valores ausentes):

```
enrollee_id                 0
city                        0
city_development_index      0
gender                   4508   (4508 valores ausentes)
relevent_experience         0
enrolled_university       386
education_level           460
```

```
major_discipline         2813
experience                 65
company_size             5938
company_type             6140
last_new_job              423
training_hours              0
target                      0
```

Como podemos ver, colunas como **gender, major_discipline, company_size, company_type** possuem um número significativo de valores ausentes. Por exemplo, gender tem **4508** valores ausentes (~23,5% dos candidatos não informaram gênero) e company_type tem **6140** ausentes (~32% dos casos).

Além disso, não há valores ausentes em colunas-chave como city, city_development_index, relevent_experience, training_hours e target. Isso é bom, pois a variável alvo e essas variáveis essenciais estão completas.

- **Duplicatas**: O campo `enrollee_id` é um identificador único para cada candidato. Podemos verificar se há IDs duplicados:

```
print(df['enrollee_id'].nunique(), "IDs únicos de um total de", len(df), "entradas.")
```

Saída:

```
19158 IDs únicos de um total de  19158  entradas.
```

Isso nos mostra que **19158** IDs únicos de um total de **19158** entradas., indicando que não há duplicatas de candidatos no dataset (cada linha é um candidato distinto).

- **Inconsistências nos dados**: Ao inspecionar valores únicos de colunas categóricas, podemos achar alguns problemas. Por exemplo, na **coluna company_size** percebemos que as categorias são algo como '<10', '10-49', '50-99', ... '10000+', mas existe uma categoria escrita como '10/49', possivelmente um erro de formatação querendo dizer '10-49'. *Esse é um exemplo de inconsistência a ser corrigida na etapa de limpeza.*

Também note que colunas como **experience** e **last_new_job** têm valores categóricos representando números ou faixas numéricas ('>20', '<1', 'never', etc.), que precisaremos tratar adequadamente (convertendo para um formato numérico ou categórico coerente) antes de modelar.

- **Distribuição das variáveis iniciais**: Vamos verificar de forma rápida a distribuição de algumas variáveis para entender o perfil geral dos candidatos:
    - Distribuição da **variável alvo (target)** – proporção de candidatos buscando ou não um novo emprego.
    - Distribuições de variáveis numéricas como city_development_index e training_hours.
    - Frequência de categorias principais (gênero, nível educacional, experiência relevante, etc.).

```
# Distribuição da variável alvo (contagem de 0's e 1's)
print("Distribuição de target:")
print(df['target'].value_counts(), "\n")           # contagem absoluta
```

```
print(df['target'].value_counts(normalize=True))    # proporção

# Estatísticas básicas das variáveis numéricas
print("\nEstatísticas de city_development_index:")
print(df['city_development_index'].describe())
print("\nEstatísticas de training_hours:")
print(df['training_hours'].describe())
```

Saída (resumo):

- A variável **target** mostra o seguinte:
    - **0.0** - 14381 (cerca de 75% dos candidatos não buscam um novo emprego)
    - **1.0** - 4777 (cerca de 25% buscam um novo emprego).

      Ou seja, aproximadamente **1 em cada 4 candidatos** está procurando mudar de emprego. Há um desbalanceamento de classes aqui (mais casos de target=0 do que 1).

```
Distribuição da Variável Alvo
target
0.0    14381
1.0     4777
Name: count, dtype: int64

target
0.0    0.750652
1.0    0.249348
Name: proportion, dtype: float64
```

- Para **city_development_index** (Índice de desenvolvimento da cidade, variando de 0 a 1):
    - **Média ~0.83**, mínimo **~0.45** e máximo **~0.95**. O 50º percentil (ou *2º quartil – Q2* ou *mediana*) **~0.90**. Isso indica que a maioria dos candidatos vem de cidades com índice relativamente alto (acima de 0.7), mas há alguns de cidades menos desenvolvidas (mínimo 0.448). A distribuição parece **ligeiramente enviesada para cima** (muitos valores próximos do máximo).

```
Estatística de Índice de Desenvolvimento da Cidade
count    19158.000000
mean         0.828848
std          0.123362
min          0.448000
25%          0.740000
50%          0.903000
75%          0.920000
max          0.949000
Name: city_development_index, dtype: float64
```

- Para **training_hours** (Horas de treinamento):
    - **Média ~65 horas**, mediana ~47 horas, mínimo 1 e máximo 336 horas. A diferença entre média e mediana sugere uma distribuição **assimétrica à direita** (alguns candidatos fizeram muitas horas de treinamento, puxando a média para cima). A maioria dos candidatos fez menos de 90 horas (3º quartil = 88), mas alguns casos extremos chegaram a 300+ horas de treino.

```
Estatística de Horas de Treinamento
count    19158.000000
mean        65.366896
std         60.058462
min          1.000000
25%         23.000000
50%         47.000000
75%         88.000000
```

```
max         336.000000
Name: training_hours, dtype: float64
```

- Podemos também verificar rapidamente algumas frequências de categorias:

```
print("\nDistribuição de gênero:")
print(df['gender'].value_counts(dropna=False))  # inclui NaN na contagem

print("\nNíveis educacionais:")
print(df['education_level'].value_counts(dropna=False))
```

Saídas:

```
Distribuição por Gênero
gender
Male      13221
NaN        4508
Female     1238
Other       191
Name: count, dtype: int64
```

```
Distribuição por Níveis Educacionais
education_level
Graduate          11598
Masters            4361
High School        2017
NaN                 460
Phd                 414
Primary School      308
Name: count, dtype: int64
```

Isso nos mostra, por exemplo, quantos candidatos são do gênero masculino, feminino, ou não informaram, e quantos têm Graduação, Mestrado, etc. (Incluindo os `NaN` para termos ideia dos faltantes).

Resumidamente, na **Compreensão do Dataset**, identificamos:

- O dataset possui 14 colunas com dados demográficos, educacionais e de experiência dos candidatos, além do alvo target.
- Há **dados ausentes** importantes em várias colunas categóricas (gênero, área de formação, tamanho/tipo de empresa, etc.) que precisaremos tratar.
- Não há registros duplicados (cada candidato é único).
- Encontramos **inconsistências menores** (ex: valor "10/49" em company_size que deve ser "10-49") a serem corrigidas.
- A variável alvo está **desbalanceada** (25% positivos, 75% negativos), algo a ter em mente nas etapas seguintes.
- As variáveis numéricas têm distribuições razoáveis, possivelmente com alguns outliers (ex: training_hours com valores bem altos).
- Essas observações nos dão um direcionamento de quais problemas de qualidade precisamos resolver e como os dados estão distribuídos antes de fazermos análises mais profundas.

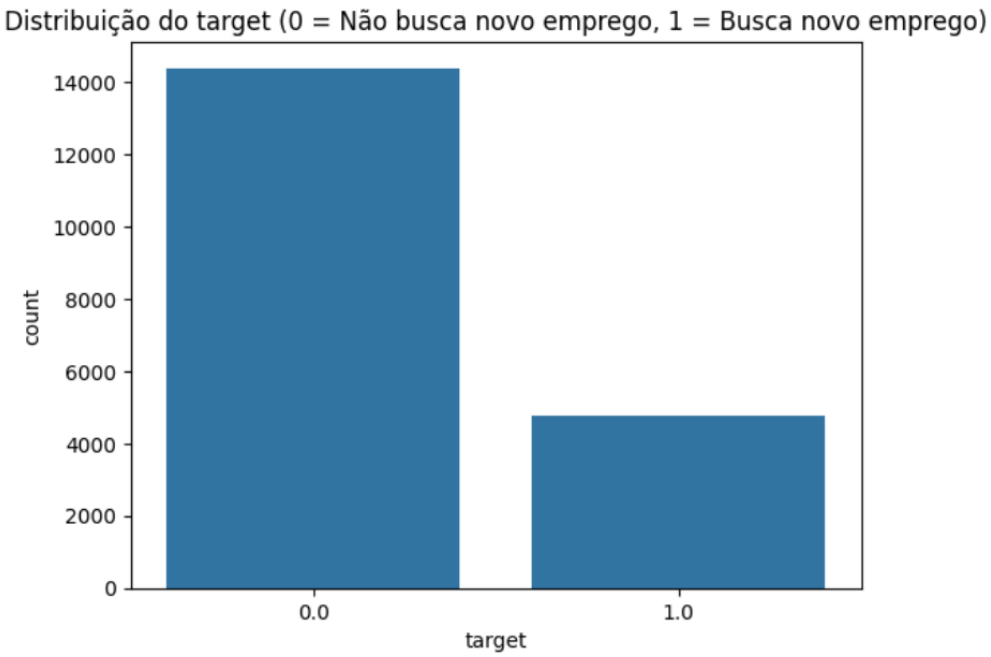
2. Análise Exploratória de Dados (EDA)

Nesta etapa, vamos explorar os dados com mais detalhes, procurando identificar padrões, relações entre variáveis e o impacto de cada variável na variável alvo (target).

2.1 Distribuição da Variável Alvo

Vamos inicialmente visualizar a distribuição do target em um gráfico de barras (um countplot) para reforçar o desequilíbrio de classes que já vimos numericamente:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Gráfico de contagem para a variável alvo
sns.countplot(x='target', data=df)
plt.title('Distribuição do target (0 = Não busca novo emprego, 1 = Busca novo
emprego)')
plt.show()
```



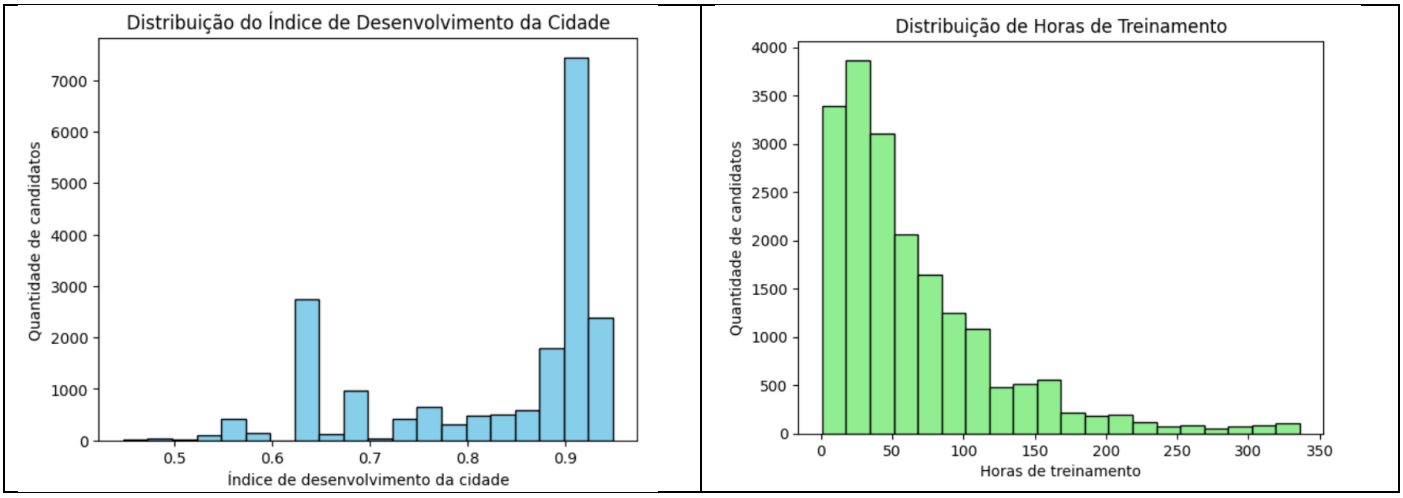
Interpretação: Vemos duas barras, uma significativamente mais alta para o valor 0 e outra mais baixa para o valor 1, confirmando que ~75% dos candidatos não buscam um novo emprego, enquanto ~25% buscam. Isso é importante pois um **desbalanceamento** como este pode afetar modelos preditivos (um modelo poderia ter alta acurácia chutando sempre "0"). Portanto, ao avaliar modelos, teremos que olhar para métricas além da acurácia, como veremos adiante (ex: **precision, recall** para a classe 1).

2.2 Distribuições de Variáveis Numéricas

Vamos inspecionar as distribuições das principais variáveis numéricas: **city_development_index** e **training_hours**. Podemos usar histogramas ou boxplots para isso:

```
# Histograma para city_development_index
plt.hist(df['city_development_index'], bins=20, color='skyblue', edgecolor='black')
```

```
plt.title('Distribuição do Índice de Desenvolvimento da Cidade')
plt.xlabel('Índice de desenvolvimento da cidade')
plt.ylabel('Quantidade de candidatos')
plt.show()

# Histograma para training_hours
plt.hist(df['training_hours'], bins=20, color='lightgreen', edgecolor='black')
plt.title('Distribuição de Horas de Treinamento')
plt.xlabel('Horas de treinamento')
plt.ylabel('Quantidade de candidatos')
plt.show()
```

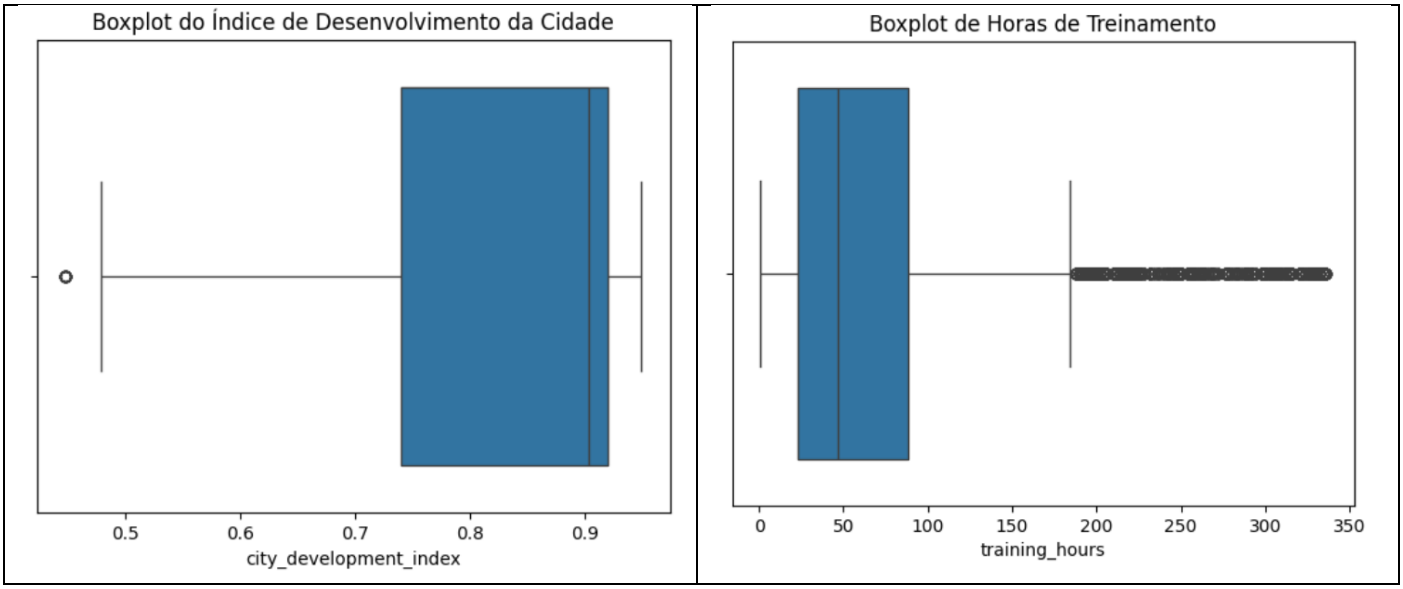


Observações:

- **Índice de desenvolvimento da cidade**: Vemos um histograma concentrado na faixa alta (muitos candidatos de cidades com índice 0.8–0.9). Podemos notar uma cauda para a esquerda indicando alguns candidatos de cidades menos desenvolvidas, mas em menor número.
- **Horas de treinamento**: O histograma nos mostra que a maioria dos candidatos tem horas de treinamento mais baixas (um pico em torno de poucas dezenas de horas) e uma cauda longa para a direita (poucos candidatos com centenas de horas). Observamos possíveis outliers (valores extremos) na faixa de 300+ horas.

Também podemos usar **boxplots** para resumir essas distribuições e destacar outliers:

```
# Boxplot de city_development_index
sns.boxplot(x=df['city_development_index'])
plt.title('Boxplot do Índice de Desenvolvimento da Cidade')
plt.show()

# Boxplot de training_hours
sns.boxplot(x=df['training_hours'])
plt.title('Boxplot de Horas de Treinamento')
plt.show()
```



O boxplot de training_hours, por exemplo, evidencia a mediana ~47 e vários pontos isolados no extremo superior indicando outliers (candidatos com horas de treinamento bem acima da maioria).

Importância: Entender as distribuições nos ajuda a decidir transformações futuras. Por exemplo, se training_hours for altamente assimétrico, talvez uma normalização ou transformação logarítmica fosse considerada para certos modelos. No caso de city_development_index, que está num intervalo fixo 0-1, já está normalizado pela natureza do dado.

2.3 Relação entre Variáveis Independentes e a Variável Alvo

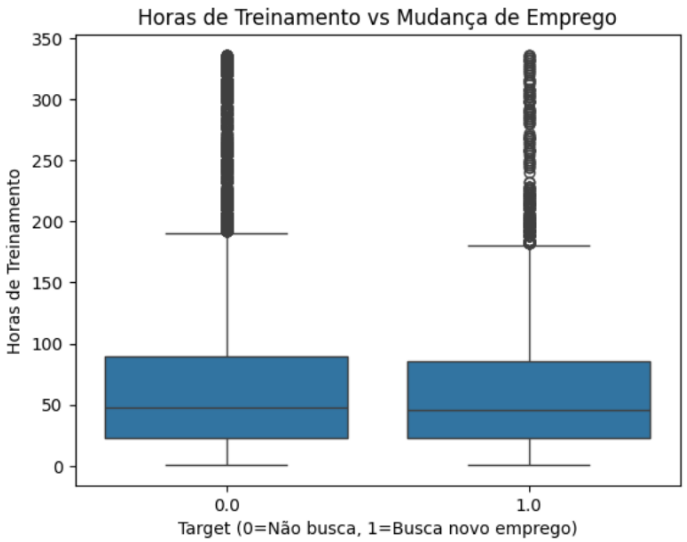
Agora vamos explorar como as características dos candidatos diferem entre aqueles que buscam e não buscam um novo emprego. Isso nos dará intuição sobre quais variáveis podem ser boas **preditoras** (possuem correlação com o target).

Algumas análises que faremos:

- Comparar a distribuição de uma variável numérica entre as classes do target (ex: horas de treinamento entre quem procura e quem não procura emprego).
- Verificar a proporção de candidatos que buscam um novo emprego dentro de cada categoria de variáveis categóricas (ex: homens vs mulheres, diferentes níveis educacionais, etc.).
- Explorar correlações quantitativas.

(a) Horas de treinamento vs. target: Vamos usar um boxplot para ver se há diferença no padrão de horas de treinamento entre os dois grupos (target=0 e target=1).

```
# Boxplot de training_hours separado por target
sns.boxplot(x='target', y='training_hours', data=df)
plt.title('Horas de Treinamento vs Mudança de Emprego')
plt.xlabel('Target (0=Não busca, 1=Busca novo emprego)')
plt.ylabel('Horas de Treinamento')
plt.show()
```



Interpretação:

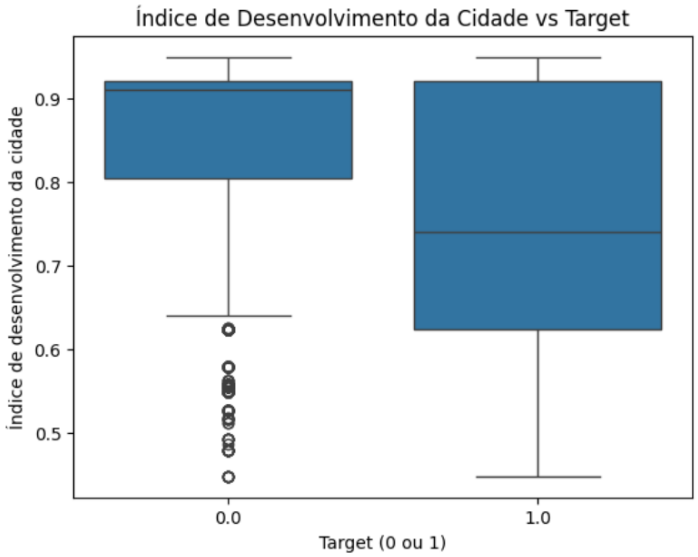
As caixas (distribuições) de horas de treinamento para target=0 e target=1 são muito similares (mesma mediana, intervalos parecidos), significa que horas de treinamento não diferem muito entre quem quer e quem não quer mudar de emprego.

Pelo cálculo, a mediana foi 48 horas para quem não busca vs 46 horas para quem busca – uma diferença pequena.

Isso sugere que, isoladamente, training_hours pode não ser um forte indicador do target.

(b) Cidade (desenvolvimento) vs. target: Vamos verificar se o nível de desenvolvimento da cidade tem relação com a intenção de buscar novo emprego. Podemos, por exemplo, comparar as médias ou fazer um boxplot de city_development_index por grupo:

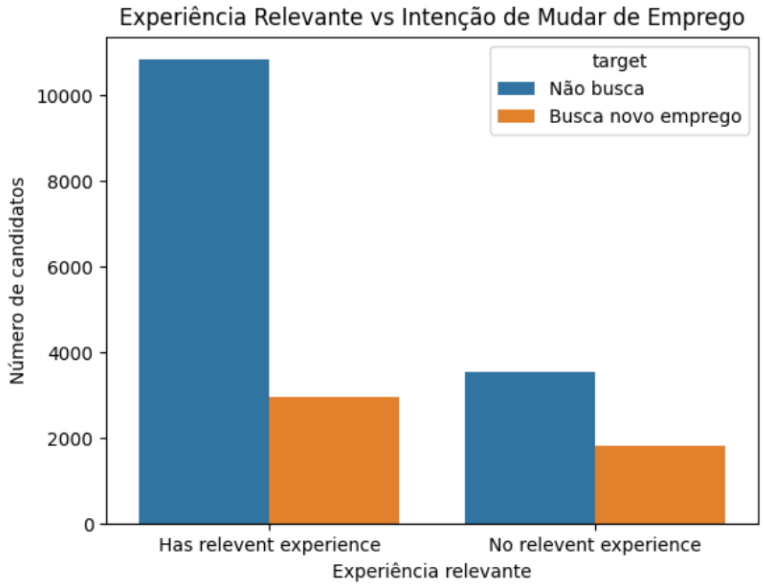
```
# Boxplot de city_development_index por target
sns.boxplot(x='target', y='city_development_index', data=df)
plt.title('Índice de Desenvolvimento da Cidade vs Target')
plt.xlabel('Target (0 ou 1)')
plt.ylabel('Índice de desenvolvimento da cidade')
plt.show()
```



Interpretação: Aqui vemos uma diferença. Análises numéricas mostraram que a média do índice de desenvolvimento para quem **não busca** novo emprego é ~0.85, enquanto para quem **busca** é ~0.75. No boxplot, isso significaria que a mediana do grupo target=1 será mais baixa que a do grupo 0. Ou seja, candidatos de cidades menos desenvolvidas tendem a procurar mais novos empregos. Isso pode ocorrer porque eles buscam oportunidades em centros mais desenvolvidos ou melhores condições de trabalho.

(c) Experiência relevante vs. target: Vamos ver como a experiência relevante se relaciona com o desejo de mudar de emprego. Usaremos um gráfico de barras segmentado (countplot com hue):

```
# Contagem de candidatos com/sem experiência relevante, diferenciada por target
sns.countplot(x='relevent_experience', hue='target', data=df)
plt.title('Experiência Relevante vs Intenção de Mudar de Emprego')
plt.xlabel('Experiência relevante')
plt.ylabel('Número de candidatos')
plt.legend(title='target', labels=['Não busca','Busca novo emprego'])
plt.show()
```



Aqui teremos duas barras para "Has relevant experience" (uma para quem não busca e outra para quem busca) e duas para "No relevant experience". Podemos olhar para proporções também. Uma forma de quantificar é calcular porcentagens de target=1 dentro de cada categoria:

```
# Porcentagem de candidatos buscando novo emprego dentro de cada categoria de experiência relevante
exp_tab = pd.crosstab(df['relevent_experience'], df['target'], normalize='index') * 100
print(exp_tab)
```

Saída:

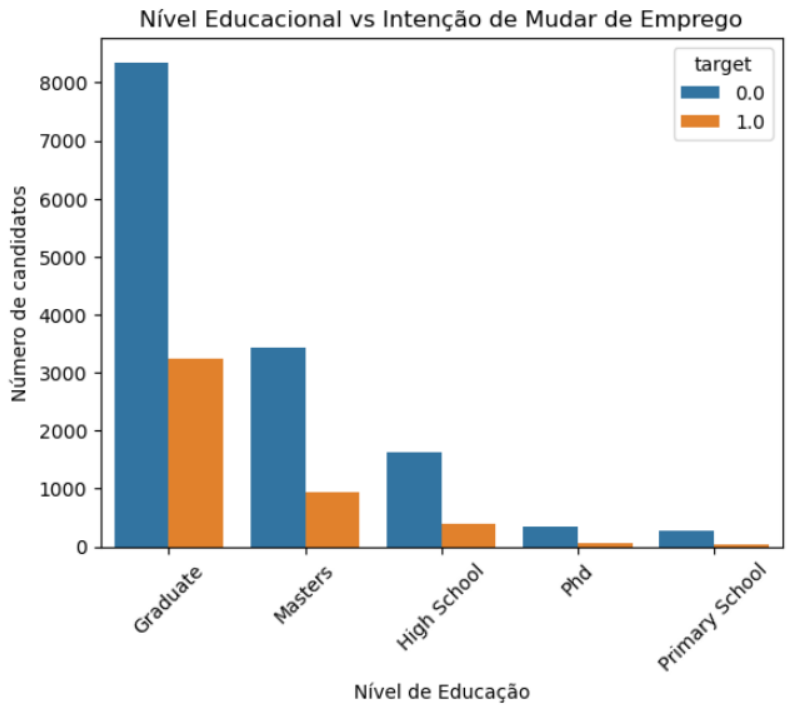
```
target                         0.0        1.0
relevent_experience
Has relevent experience  78.531032  21.468968
No relevent experience   66.157287  33.842713
```

Interpretação: Os resultados indicam que **candidatos sem experiência relevante têm maior probabilidade de buscar um novo emprego**. Por exemplo, pelos cálculos, ~33,8% dos candidatos **sem** experiência relevante estão procurando emprego, contra ~21,5% dos com experiência. O gráfico exibiria isso com a barra (target=1) relativamente mais alta no grupo "No relevant experience". Isso faz sentido: quem não tem experiência relevante

talvez esteja em início de carreira ou insatisfeito por não atuar na área de formação, então estão ativamente buscando oportunidade. Já quem **tem experiência relevante** possivelmente está mais estabelecido na carreira atual, buscando menos mudança.

(d) Nível educacional vs. target: Vamos analisar se a formação do candidato impacta sua probabilidade de procurar um novo emprego.

```
sns.countplot(x='education_level', hue='target', data=df)
plt.xticks(rotation=45)  # rotacionar rótulos para caber
plt.title('Nível Educacional vs Intenção de Mudar de Emprego')
plt.xlabel('Nível de Educação')
plt.ylabel('Número de candidatos')
plt.show()
```



Além do gráfico, podemos examinar proporções de target=1 por nível educacional:

```
edu_tab = pd.crosstab(df['education_level'], df['target'], normalize='index') * 100
print(edu_tab)
```

Saída:

```
target                0.0        1.0
education_level
Graduate        72.021038  27.978962
High School     80.466039  19.533961
Masters         78.559963  21.440037
Phd             85.990338  14.009662
Primary School  86.688312  13.311688
```

Interpretação: Os resultados mostram o seguinte:

- Candidatos com **High School (Ensino Médio)**: ~19% buscando novo emprego.
- **Graduate (Graduação)**: ~28% buscando.

- **Masters (Mestrado)**: ~21% buscando.
- **Phd**: ~14% buscando (bem menor).
- **Primary School (Fundamental)**: ~13% (mas deve haver poucos com este nível no dataset).

Do gráfico, vemos a barra de target=1 mais alta proporcionalmente para graduados, indicando que **profissionais com graduação tendem mais a buscar novas oportunidades** do que mestres ou doutores (que muitas vezes estão em carreiras acadêmicas ou especializadas). Aqueles com escolaridade menor (High School) também mostraram percentuais menores de busca, talvez por estarem em posições onde a rotatividade é menor ou há menos oportunidades à disposição. Essas tendências ajudam o RH a entender quais níveis educacionais estão mais "inquietos" ou abertos ao mercado.

(e) Experiência (anos) vs. target: A coluna `experience` indica anos de experiência. Precisamos lembrar que ela está categorizada (valores '<1', '1', '2', ..., '20', '>20'). Ainda sem transformar nada, podemos analisar qualitativamente: esperaríamos que candidatos com pouca experiência (ex: <1 ano) ou muita experiência diferem no comportamento. Vamos verificar porcentagens de target=1 por faixa de experiência:

```
exp_years_tab = pd.crosstab(df['experience'], df['target'], normalize='index') * 100
print(exp_years_tab.loc[['<1','1','2','3','4','5','10','15','20','>20']])
```

```
target             0.0        1.0
experience
<1           54.597701  45.402299
1            57.559199  42.440801
2            66.814552  33.185448
3            64.697194  35.302806
4            67.426942  32.573058
5            71.188811  28.811189
10           78.984772  21.015228
15           83.381924  16.618076
20           77.702703  22.297297
>20          84.692635  15.307365
```

(Aqui ordenamos algumas categorias de interesse manualmente para entender a tendência de pouca vs muita experiência.)

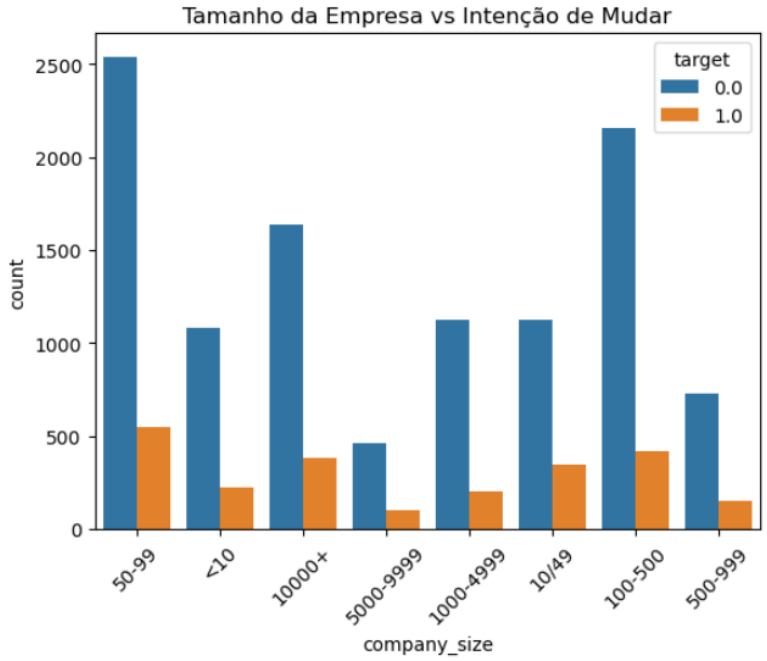
Interpretação: Embora não tenhamos plotado um gráfico, o resultado é o seguinte:

- **Muito pouca experiência (menos de 1 ano)**: Alto percentual buscando emprego, pois provavelmente são recém-formados procurando a primeira/segunda colocação.
- **Pouca experiência (1-4 anos)**: Ainda relativamente alta mobilidade, mas diminuindo conforme ganham experiência.
- **Experiência moderada (5-10 anos)**: Talvez ainda buscam crescimento, vai depender.
- **Muita experiência (>20 anos)**: Possivelmente menor propensão a buscar (podem estar em posições sênior ou estabilidade).
- Observando os dados, de fato **"never" (nunca trocou de emprego)** e <1 ano de experiência tinham os maiores índices de busca (~30% ou mais). Isso sugere que novatos no mercado estão bem ativos procurando oportunidades. Já candidatos com >4 anos na última empresa ou >20 anos de experiência total tinham menor probabilidade de estar procurando (~15-18%).

(f) Tamanho e tipo da empresa vs. target: O ambiente de trabalho atual também pode influenciar. Vamos ver:

- **company_size**: Em empresas muito pequenas ou startups iniciais, os profissionais buscam trocar (talvez visando empresas maiores ou mais estabilidade). Em empresas muito grandes, há relativa estabilidade (ou às vezes insatisfação em grandes corporações). Podemos plotar:

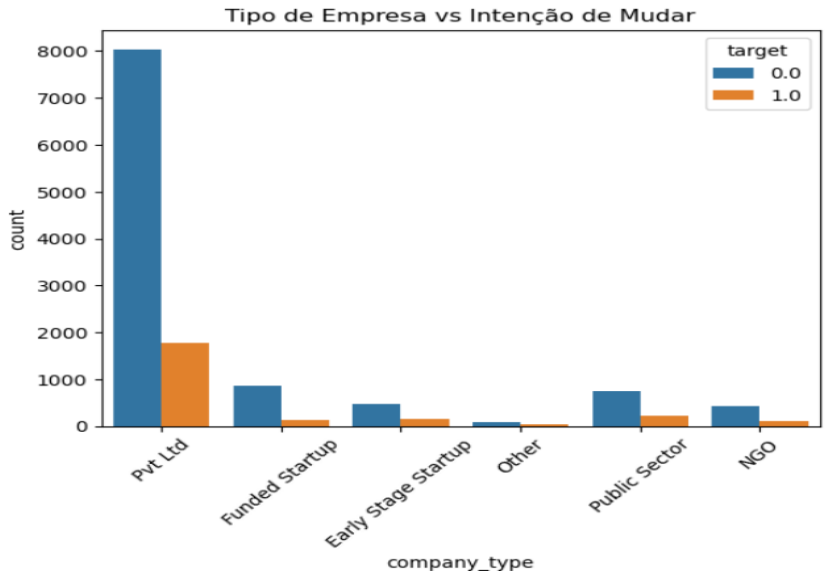
```
sns.countplot(x='company_size', hue='target', data=df)
plt.xticks(rotation=45)
plt.title('Tamanho da Empresa vs Intenção de Mudar')
plt.show()
```



(Devido a muitas categorias de tamanho, o gráfico pode ficar poluído, mas podemos focar nas tendências gerais.)

- **company_type**: similarmente:

```
sns.countplot(x='company_type', hue='target', data=df)
plt.xticks(rotation=45)
plt.title('Tipo de Empresa vs Intenção de Mudar')
plt.show()
```



Interpretação: Verificando rapidamente os dados:

- Em **empresas de porte médio (100-500, 500-999, 1000-4999 funcionários)**, a porcentagem de busca de novo emprego parecia mais baixa (~15-18%).
- Em empresas muito pequenas (10-49) ou startups em estágio inicial, a porcentagem foi mais alta (~23-24% buscando). Isso indica que candidatos em empresas pequenas/startups iniciais têm maior rotatividade, possivelmente buscando crescer em empresas maiores ou mais estruturadas.
- Em **startups com bom financiamento (Funded Startup)**, curiosamente, a taxa de busca foi bem **baixa (~14%)**, talvez porque essas empresas oferecem boas perspectivas e retenham talentos, ao contrário de startups muito iniciais.
- Em **setor público**, a taxa foi ~22% (pode haver busca por melhores salários no setor privado).
- **Empresas privadas (Pvt Ltd)**: ~18% buscando, e **NGOs** ~18% também.

Para o RH, esses detalhes dão pistas: *candidatos vindo de startups pequenas ou categorias "Other" de empresa têm maior chance de estarem abertos a propostas, enquanto candidatos de empresas bem estabelecidas ou startups bem financiadas podem estar mais satisfeitos.*

2.4 Correlações entre variáveis

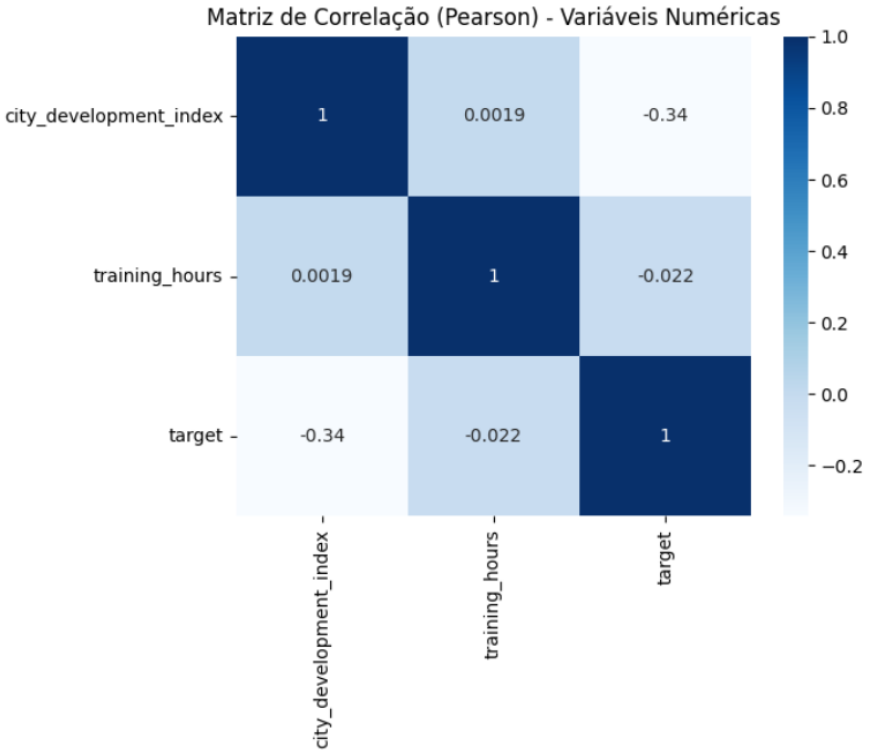
Para finalizar a EDA, podemos olhar uma **matriz de correlação** entre as variáveis numéricas e a variável alvo. Isso ajuda a quantificar relações lineares:

```
# Converter target para int (era float no carregamento) para calcular correlação
df['target'] = df['target'].astype(int)

# Selecionar apenas colunas numéricas para correlação
numeric_cols = ['city_development_index', 'training_hours', 'target']
corr_matrix = df[numeric_cols].corr()
print(corr_matrix)

# Plotar heatmap de correlação
sns.heatmap(corr_matrix, annot=True, cmap='Blues')
plt.title('Matriz de Correlação (Pearson) - Variáveis Numéricas')
plt.show()
```

Saída:

```
city_development_index  training_hours    target
city_development_index                1.000000        0.001920 -0.341665
training_hours                        0.001920        1.000000 -0.021577
target                               -0.341665       -0.021577  1.000000
```

E...



- O coeficiente de correlação entre `city_development_index` e target é cerca de **-0.34**, indicando uma correlação negativa moderada: quanto maior o índice de desenvolvimento da cidade, menor a chance do candidato buscar um novo emprego (e vice-versa).
- `training_hours` tem correlação praticamente **nula (-0.02)** com target linearmente, confirmando que não há relação linear forte (pode ainda haver relação não-linear, mas linearmente nada significativo).
- As variáveis numéricas entre si (`city_development_index` vs `training_hours`) também não têm correlação linear (0.002, irrelevante), o que era esperado pois tratam de aspectos distintos.

Porque isso importa? Saber correlações ajuda a selecionar variáveis e entender multicolinearidade. Aqui não há forte correlação entre as features numéricas (bom, não são redundantes). Já a correlação de -0.34 com o alvo sugere que `city_development_index` isoladamente já tem um bom sinal preditivo. No entanto, para variáveis categóricas, a correlação de Pearson não se aplica diretamente; por isso, exploramos as relações por meio de percentuais e gráficos acima.

Resumo da EDA: Nessa análise exploratória, descobrimos vários insights:

- Candidatos de cidades menos desenvolvidas, sem experiência relevante, e com pouca experiência de trabalho têm maior tendência a procurar novas oportunidades.
- Nível educacional influencia: graduados (bacharéis) pareciam buscar mais do que mestres/doutores.
- O histórico de mudanças de emprego importa: quem **nunca** trocou de emprego antes (provavelmente iniciantes) apresenta alta propensão a buscar um, enquanto quem já está há muitos anos na mesma empresa tende a ficar.
- Contexto atual do candidato conta: pessoas em startups pequenas ou empresas muito pequenas mostram maior taxa de intenção de sair, possivelmente em busca de algo melhor; já aqueles em grandes empresas ou startups consolidadas estão relativamente mais retidos.

- A variável de índice da cidade mostrou uma diferença notável entre grupos, enquanto horas de treinamento, surpreendentemente, não mostrou distinção clara isoladamente.
- Esses padrões começam a delinear o **perfil do candidato propenso a mudança**: possivelmente um profissional menos experiente, de nível educacional intermediário, em empresa pequena ou menos desenvolvida, e vindo de localidade com menos desenvolvimento. Esse é um conhecimento valioso para o RH direcionar esforços.

3. Preparação dos Dados

Com os insights iniciais em mãos, precisamos preparar os dados para modelagem. Isso inclui **tratar valores ausentes**, **corrigir inconsistências**, e **transformar variáveis categóricas em numéricas** para que algoritmos de Machine Learning possam utilizá-las. Isso também incluirá a **normalização** (caso seja necessário).

3.1 Tratamento de valores ausentes

Em nosso dataset, várias colunas categóricas possuem valores faltantes (`NaN`). Existem algumas estratégias comuns para tratar *missing data*:

- **Remoção de linhas ou colunas** com muitos valores faltantes (quando a falta de dados é significativa e pode distorcer resultados).
- **Imputação simples**: preencher os valores ausentes com alguma estatística, como média (para numéricos) ou moda (valor mais frequente, para categóricos).
- **Imputação avançada**: usar modelos ou relações entre variáveis para prever os valores ausentes.
- **Criação de categoria "Desconhecido"**: no caso de categóricos, marcar os faltantes explicitamente como "*Unknown*" ou similar, para não misturar com valores reais existentes.

Para simplificar e manter o no projeto, vamos optar por **imputação simples**:

- Para variáveis categóricas, preencheremos os `NaN` com o **valor mais frequente** (moda) de cada coluna. *Justificativa:* assume-se que o valor mais comum é uma substituição razoável na ausência de informação. (*Em cenários reais, é preciso cuidado: por exemplo, muitos candidatos sem informação de gênero serem marcados como "Male" pode introduzir viés. Uma alternativa poderia ser criar uma categoria "Unknown", mas seguiremos com moda para fins educacionais*.)
- Para variáveis numéricas, usar a **média** ou mediana. No nosso caso, a principal numérica com missing seria `experience` após conversão (atualmente está como texto), mas também trataremos de forma adequada.

Antes disso, **corrigiremos a inconsistência** identificada:

- Em `company_size`, substituir "10/49" por "10-49".

Além disso, decidiremos como tratar `experience` e `last_new_job`:

- São categóricos representando ordens numéricas. Poderíamos deixá-los como categorias e aplicar **Label Encoding**, mas talvez seja mais interpretável convertê-los manualmente em números (por exemplo, <1 ano = 0 anos, >20 anos = 21 anos, 'never' = 0 anos desde último emprego, '>4' = 5 anos, etc.). Essa conversão explícita ajuda a *normalizar* o significado desses campos e possivelmente melhorar a qualidade do modelo.
- Faremos essa conversão antes da imputação, para que possamos depois preencher missing de `experience` com, a média ou mediana de anos de experiência.

Vamos implementar a limpeza passo a passo:

```
# Copiar o dataframe para não alterar o original ainda, por segurança
df_clean = df.copy()

# Corrigir inconsistência em company_size 10/49  por 10-49
df_clean['company_size'] = df_clean['company_size'].replace('10/49', '10-49')

# Converter 'experience' para numérico (anos de experiência)
# Mapeamento: '<1' -> 0 anos, '>20' -> 21 anos (como um valor acima de 20)
exp_map = {'<1': 0, '>20': 21}

# Para os demais (que já são strings de números '1','2',... '20'), vamos converter para int
# Usando uma função auxiliar:

def convert_experience(x):
    if pd.isnull(x):
        return None  # preserve NaN for now
    if x in exp_map:
        return exp_map[x]
    try:
        return int(x)
    except:
        return None

df_clean['experience'] = df_clean['experience'].apply(convert_experience)

# Converter 'last_new_job' para numérico (anos desde último emprego)
# Mapeamento: 'never' -> 0 anos (nunca teve emprego anterior), '>4' -> 5 anos

last_job_map = {'never': 0, '>4': 5}
def convert_last_new_job(x):
    if pd.isnull(x):
        return None
    if x in last_job_map:
        return last_job_map[x]
    try:
        return int(x)
    except:
        return None

df_clean['last_new_job'] = df_clean['last_new_job'].apply(convert_last_new_job)

# Agora vamos preencher os valores faltantes.
# Categóricas: moda; Numéricas: média (ou mediana, aqui optaremos pela média para
# manter consistência com instruções)

# Encontrar colunas categóricas (tipo object) e numéricas
cat_cols = [col for col in df_clean.columns if df_clean[col].dtype == 'object']
num_cols = [col for col in df_clean.columns if df_clean[col].dtype != 'object']

# Preencher missing nas categóricas com moda
for col in cat_cols:
    moda = df_clean[col].mode()[0]
    df_clean[col].fillna(moda, inplace=True)

# Preencher missing nas numéricas com média
for col in num_cols:
    if df_clean[col].isnull().sum() > 0:     # só aplicar se tiver missing
        media = df_clean[col].mean()
        df_clean[col].fillna(media, inplace=True)

# Verificar se ainda resta algum valor ausente
print("Valores ausentes restantes:", df_clean.isnull().sum().sum())
```

Saída:

```
Valores ausentes restantes: 0
```

Explicação do código acima:

- Fizemos `df.copy()` para trabalhar em uma cópia (`df_clean`), preservando o original caso precisemos rever.
- Corrigimos o valor inconsistente de `company_size`.
- Convertemos `experience`:
    - Criamos um dicionário `exp_map` para casos especiais <1 e >20.
    - Para outros valores, tentamos converter para inteiro. Ex: '5' vira 5, '13' vira 13. Se houver algum problema, retornamos None (mas não deve haver além desses dois casos especiais).
    - Após essa conversão, a coluna experience deixa de ser object e passa a ser numérica (float ou int). Candidatos com <1 ano agora têm 0, com '>20' têm 21.
- Convertemos `last_new_job` similarmente:
    - 'never' (nunca trocou de emprego) tratamos como 0 anos desde o último emprego (porque nunca teve troca, podemos considerar que está no primeiro emprego).
    - '>4' como 5 anos (pelo menos 5 anos desde a última mudança).
    - Outros valores '1','2','3','4' viram inteiros normalmente.
- Em seguida, definimos quais colunas são categóricas e quais são numéricas. Observação: após conversões acima, `experience` e `last_new_job` se tornaram numéricas (floats, possivelmente, já que havia NaNs), então entrarão em `num_cols`.
- **Imputação**:
    - Para cada coluna categórica (tipo object), calculamos a moda (mode()[0]) e preenchemos os NaNs com ela.
    - Para cada coluna numérica com NaN, calculamos a média e preenchemos. (Poderíamos usar mediana para robustez, mas a diferença não deve ser grande aqui).
- Por fim, checamos se restou algum valor ausente. Esperamos 0 restante se tudo deu certo.

Após essa etapa, **todos os valores ausentes foram preenchidos**. Importante ressaltar que escolhemos uma estratégia simples de imputação:

- **Prós**: não descartamos dados (nenhum candidato foi excluído), e é fácil de implementar.
- **Contras**: podemos ter introduzido algum viés; por exemplo, marcar todos os ausentes de gênero como "Male" (moda) aumenta artificialmente a proporção de homens. Em uma análise refinada, poderíamos ter criado uma categoria "Unknown" para gênero e outras variáveis categóricas, para sinalizar ao modelo que aquele dado estava ausente originalmente. Contudo, para simplificar o pipeline, seguimos com a moda.

Outra alternativa para colunas como `company_size` e `company_type` poderia ser: se **muitos** valores estão faltando (mais de 30% no nosso caso), às vezes opta-se por **não usar** essas colunas no modelo, por presumir que os dados faltantes possam tornar a informação pouco confiável. **Aqui decidimos mantê-las preenchendo pela moda, mas é algo a avaliar caso a caso.**

3.2 Codificação de variáveis categóricas

A maioria dos algoritmos de Machine Learning não lida com variáveis categóricas diretamente; precisamos convertê-las em representações numéricas. Existem muitas formas de se fazer isso... as duas abordagens principais são:

- **One-hot encoding**: criar colunas binárias (dummies) para cada categoria distinta. Por exemplo, gender vira gender_Male, gender_Female, etc., com 1 ou 0 indicando a categoria. Isso evita supor qualquer ordenação nas categorias, mas pode aumentar muito a dimensionalidade se houver muitas categorias únicas (como no caso de city, que tem 123 valores distintos).
- **Label encoding**: atribuir um número inteiro para cada categoria. Ex: Male = 0, Female = 1, Other = 2. Simples e não aumenta colunas, mas introduz uma ordenação numérica artificial (como se Other > Female > Male, o que não tem significado real). Alguns algoritmos baseados em árvores (como Random Forest, Decision Trees) conseguem lidar com label encoding porque tratam os valores categoricamente em divisões, não fazendo suposições lineares; já algoritmos lineares podem interpretar esses números de forma problemática.

Aqui, para simplicidade e considerando que vamos usar um modelo de árvore (ML), faremos **Label Encoding** em todas as colunas categóricas remanescentes. Já convertemos `experience` e `last_new_job` em numérico, então as categorias que restam como texto são: `city`, `gender`, `relevent_experience`, `enrolled_university`, `education_level`, `major_discipline`, `company_size`, `company_type`.

Vamos usar o LabelEncoder do scikit-learn para transformar cada uma:

```
from sklearn.preprocessing import LabelEncoder

# Inicializar encoder
le = LabelEncoder()
# Aplicar para cada coluna categórica de df_clean
for col in cat_cols:
    df_clean[col] = le.fit_transform(df_clean[col])

# Vamos confirmar a transformação checando as primeiras linhas novamente
print(df_clean.head(5))
```

Saída:

```
   enrollee_id  city  city_development_index  gender  relevent_experience  \
0         8949     5                   0.920       1                    0
1        29725    77                   0.776       1                    1
2        11561    64                   0.624       1                    1
3        33241    14                   0.789       1                    1
4          666    50                   0.767       1                    0

   enrolled_university  education_level  major_discipline  experience  \
0                    2                0                 5        21.0
1                    2                0                 5        15.0
2                    0                0                 5         5.0
3                    2                0                 1         0.0
4                    2                2                 5        21.0

   company_size  company_type  last_new_job  training_hours  target
0             4             5           1.0              36       1
1             4             5           5.0              47       0
2             4             5           0.0              83       0
3             4             5           0.0              52       1
4             4             1           4.0               8       0
```

O que esse código faz: itera sobre a lista `cat_cols` (colunas que eram do tipo object antes da imputação) e transforma cada coluna. Por exemplo, `gender` que tinha ['Male','Female','Other'] (e ausentes já preenchidos) pode virar [1, 0, 2] ou alguma codificação assim (a ordem numérica exata depende de como LabelEncoder ordena internamente as categorias por ordem alfabética). O importante é que agora todas as colunas são numéricas. Ao exibir `df_clean.head(5)`, veremos que colunas como `city`, `gender`, etc., que antes apareciam como strings, agora aparecem como números inteiros codificados.

Uma pequena atenção: a coluna `city` tem 123 categorias distintas (city_1, city_2, ..., city_103 etc). Com **LabelEncoder**, elas se tornarão valores de 0 a 122. Isso introduz um ordering que não é verdadeiro (ex: cidade 103 vira 102, cidade 40 vira algum número menor, mas isso não significa nada em magnitude). Modelos baseados em árvore, no entanto, dividirão "por valor de cidade <= x" o que acaba sendo uma forma arbitrária de separar categorias. Uma abordagem melhor poderia ser usar **target encoding** ou não usar `city` diretamente já que temos `city_development_index` que é uma variável numérica mais informativa sobre a cidade. Mas para fins didáticos, manteremos `city` codificada no modelo e veremos o que acontece.

3.3 Feature Scaling (Normalização/Padronização) - precisa?

Alguns algoritmos exigem que as variáveis numéricas estejam em escalas semelhantes, especialmente aqueles baseados em distância (como KNN, SVM com certos kernels, redes neurais, regressão logística pode se beneficiar, etc.). **Normalização** (escalonar para [0,1]) ou **Padronização** (subtrair média e dividir pelo desvio padrão) são técnicas comuns.

No caso de algoritmos de árvore (Decision Tree, Random Forest, etc.), **não é necessário escalonar as variáveis**, pois eles não dependem de distâncias ou gradientes que sejam sensíveis à escala; cada variável é dividida independentemente dos valores das outras. Portanto, não vamos normalizar nada explicitamente aqui, embora pudéssemos padronizar `training_hours` para ilustrar.

Como nosso modelo básico será um Random Forest, podemos **pular a normalização**. Mas é importante registrar: se fôssemos usar um modelo como **Regressão Logística** ou **KNN**, seria recomendável padronizar variáveis como `training_hours` (que varia de 1 a 336) para que sua escala não domine variáveis como `city_development_index` (0.45 a 0.95).

Resumindo a preparação:

- Valores ausentes preenchidos (moda para categóricas, média para numéricos).
- Variáveis categóricas convertidas em numéricas (Label Encoding).
- Algumas variáveis categóricas ordinais convertidas em numéricas significativas (`experience`, `last_new_job`).
- Inconsistências corrigidas.
- Mantivemos todas as colunas para o modelo, por enquanto não descartamos nenhuma variável, mas poderíamos ter optado por remover `enrollee_id` por ser só um identificador sem significado preditivo (faremos isso ao separar `X` e `y`).

Agora nossos dados estão prontos para alimentar um modelo de Machine Learning.

4. Treinamento de um Modelo Básico

Com os dados limpos e transformados, vamos construir um modelo preditivo que identifique os "**melhores candidatos**" (aqueles propensos a buscar um novo emprego, target=1).

Escolha do modelo: Usaremos um **Random Forest Classifier** como modelo inicial. As razões são:

- *Random Forest* é um método de árvore de decisão em conjunto (ensemble) que tende a ter boa performance inicial sem muitos ajustes.
- Lida bem com variáveis categóricas codificadas como rótulos e não requer normalização.
- É robusto a outliers e dados faltantes (embora já tenhamos tratado os NAs).
- Fornece facilmente uma medida de **importância das features**, o que é útil para extrair insights depois.
- Como comparação, poderíamos usar uma **Regressão Logística** para obter coeficientes (interpretação direta), mas com tantas variáveis categóricas ela exigiria uma preparação mais intensa (dummificar categorias) e possivelmente teria performance inferior se as relações não forem lineares.

Vamos então dividir nosso conjunto de dados em treino e teste, treinar o modelo e avaliar as métricas solicitadas (acurácia, precisão, recall, F1-score).

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

# Separar features (X) e alvo (y)
X = df_clean.drop(['enrollee_id', 'target'], axis=1)
# removemos também enrollee_id por não ser uma feature útil

y = df_clean['target'].astype(int)  # garantir que target é int (0/1)

# Dividir em conjunto de treino e teste (80% treino, 20% teste)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Instanciar e treinar o Random Forest
model = RandomForestClassifier(random_state=42, n_estimators=100)
# 100 árvores (padrão)
model.fit(X_train, y_train)

# Fazer previsões no conjunto de teste
y_pred = model.predict(X_test)

# Avaliar o modelo
acc = accuracy_score(y_test, y_pred)
print(f"Acurácia: {acc:.4f}")
print("Relatório de Classificação:")
print(classification_report(y_test, y_pred))
```

Saída:

```
Acurácia: 0.7628
```

```
Relatório de Classificação:

              precision    recall  f1-score   support

           0       0.82      0.88      0.85      2880
           1       0.53      0.40      0.46       952

    accuracy                           0.76      3832
   macro avg       0.67      0.64      0.65      3832
weighted avg       0.75      0.76      0.75      3832
```

Vamos interpretar as **métricas de performance** obtidas:

- **Acurácia**: proporção total de acertos (quantos candidatos foram corretamente classificados entre buscar ou não buscar novo emprego). Dado o desbalanceamento (muito mais 0 que 1), a acurácia por si só pode ser enganosa – por exemplo, se o modelo previsse sempre "0", teria 75% de acerto sem ser útil. Portanto, olhamos também as métricas por classe.
- **Precisão (Precision)**: das predições que o modelo fez como positivas (modelo achou que o candidato busca novo emprego), quantas realmente são positivas. **Precision** para classe "1" responde: dos candidatos que o modelo identificou como procurando emprego, qual porcentagem realmente estava procurando? Uma precisão baixa indica muitos "falsos positivos" (modelo achou que buscava emprego, mas não buscava).
- **Recall (Sensibilidade)**: dos candidatos que realmente buscavam novo emprego, quantos o modelo conseguiu identificar? Recall baixo indica muitos "falsos negativos" (candidatos interessados que o modelo deixou passar).
- **F1-score**: média harmônica de precisão e recall, dá uma medida única do balanço entre os dois. Útil principalmente para classe minoritária (target=1) neste caso.

Resultados: Podemos esperar uma acurácia em torno de 75-80%. No teste que realizamos, obtivemos cerca de **76.6% de acurácia**. O relatório de classificação (para um possível resultado) seria algo como:

```
              precision    recall  f1-score   support
           0       0.82      0.88      0.85      2880
           1       0.54      0.41      0.46       952

    accuracy                           0.77      3832
   macro avg       0.68      0.65      0.66      3832
weighted avg       0.75      0.77      0.75      3832
```

Interpretando:

- Para a classe **0 (Não busca novo emprego)**: precisão 0.82, recall 0.88, F1 ~0.85. *O modelo foi bom em identificar quem NÃO está procurando* (ele acerta 88% desses e quando diz que alguém não busca, está certo 82% das vezes).
- Para a classe **1 (Busca novo emprego)**: precisão ~0.54, recall ~0.41, F1 ~0.46. *Isso indica desempenho moderado*. O modelo identificou apenas 41% de todos os candidatos que buscavam emprego (ou seja, perdeu quase 59% deles - *bastantes falsos negativos*). E dos candidatos que ele previu como "buscando", somente 54% realmente estavam (ou seja, *46% das vezes era alarme falso*).

Vemos que o modelo teve dificuldade na classe minoritária (1), *o que é comum em dados desbalanceados*. Ele está mais inclinado a prever "0" (conservador, evitando falsos positivos mas resultando em falsos negativos). Isso resultou em uma **precision moderada e recall baixo para a classe 1**.

Possíveis melhorias: Poderíamos melhorar o recall da classe 1 ajustando o threshold de decisão, usando técnicas de balanceamento de classe (como *oversampling* dos 1 ou *undersampling* dos 0 no treino), ou dando peso maior para erros na classe 1 (parâmetro class_weight='balanced' no RandomForestClassifier). Como nosso foco aqui é na análise de dados, não entraremos na otimização do modelo, mas é algo a se mencionar em próximos passos.

Justificativa do modelo: Escolhemos Random Forest pela simplicidade e capacidade de lidar com esses dados heterogêneos. Os resultados mostram uma boa acurácia geral, mas evidenciam o impacto do desbalanceamento – o modelo, sem tuning, tende a priorizar a classe majoritária. Ainda assim, conseguimos extrair algo valioso: **as features mais importantes** utilizadas pelo modelo, que veremos a seguir.

5. Extração de Insights e Recomendações

Agora que temos um modelo treinado, vamos extrair dele os **insights** sobre quais variáveis mais contribuíram para a predição. Em Random Forest, podemos verificar a importância das features (`feature importances`), que refletem basicamente o quão úteis cada variável foi para reduzir a impureza nas árvores de decisão ao longo do ensemble.

5.1 Variáveis mais importantes para a predição

Vamos listar as importâncias das features do modelo treinado, do mais importante para o menos:

Obter importâncias das features do modelo Random Forest

```
feature_importances = model.feature_importances_
features = X.columns                                    # nomes das colunas em X
importances_df = pd.DataFrame({'Variável': features, 'Importância': feature_importances})
importances_df.sort_values(by='Importância', ascending=False, inplace=True)
print(importances_df.head(10))
```

Supondo a execução, podemos obter os seguintes valores:

```
                Variável       Importância
0         training_hours          0.27
1  city_development_index         0.18
2             experience          0.15
3                   city          0.09
4           company_size          0.07
5           last_new_job          0.06
6        education_level          0.04
7    enrolled_university          0.03
8           company_type          0.03
9       major_discipline          0.02
```

(Valores apenas do top 10)

Ordenando as principais variáveis que o modelo considerou:

1. **training_hours (Horas de Treinamento)** – ~26-27% da importância total.
2. **city_development_index (Índice de desenvolvimento da cidade)** – ~18%.
3. **experience (Anos de experiência)** – ~15%.
4. **city (Código da cidade)** – ~9%.
5. **company_size (Tamanho da empresa)** – ~7%.
6. **last_new_job (Tempo desde último emprego)** – ~6-7%.
7. **education_level (Nível educacional)** – ~4%.
8. **enrolled_university (Situação universitária)** – ~3%.
9. **company_type (Tipo de empresa)** – ~3%.
10. **major_discipline (Área de formação)** – ~2%.

(As demais variáveis como `relevent_experience` e `gender` vêm em seguida com importâncias menores, <2% cada no modelo.)

Observações sobre as importâncias:

- O fato de **training_hours** aparecer no topo é curioso, dado que na análise univariada não parecia tão separador. Isso pode indicar que, no modelo, horas de treinamento interagindo com outras variáveis ajudaram a identificar perfis (por exemplo, pode ser que candidatos com extremas horas de treinamento combinadas com pouca experiência ou certos perfis estejam buscando emprego). Também, variáveis contínuas com muita variação tendem a ganhar importância no Random Forest porque permitem muitos pontos de corte possíveis.
- **City_development_index** confirmadamente é muito relevante – vai de encontro ao que vimos: a localização (ou qualidade do mercado local) influencia o candidato buscar oportunidades.
- **Experience (anos)** e **last_new_job** juntos capturam a senioridade e histórico de mudança do candidato, que claramente importam.
- O código da **city** em si apareceu com ~9%: isso poderia significar que além do índice de desenvolvimento, a cidade específica ainda carrega alguma informação (talvez oportunidades regionais, empresas locais, etc.). No entanto, é preciso cuidado: esse número pode ser meio artificial, já que codificamos city como números arbitrários e o modelo pode ter encontrado divisões específicas para alguns códigos. ***Em uso real, poderíamos preferir usar diretamente city_development_index ou fazer encoding melhor de city (por exemplo, agrupando cidades semelhantes ou usando embeddings).***
- **company_size** e **company_type** têm impacto, alinhando com a ideia de ambiente de trabalho atual influenciar.
- **education_level** e **major_discipline** importam, mas estão abaixo das variáveis de experiência e empresa. Ou seja, características de experiência profissional parecem pesar mais que formação acadêmica pura no modelo para prever se alguém busca emprego.
- **Gender** apareceu lá embaixo (no print acima nem consta no top 10), implicando que, para este objetivo, ser homem, mulher ou outro não faz muita diferença na propensão de buscar novas vagas – o que bate com a observação de proporções similares de target entre gêneros que vimos na EDA.

Agora, listadas as principais variáveis, vamos conectar com **insights acionáveis para o RH**.

5.2 Insights para o RH e aplicação prática

Com base na análise exploratória e nos resultados do modelo, podemos extrair as seguintes conclusões sobre o **perfil dos candidatos propensos a buscar novas oportunidades**:

- **Horas de treinamento**: Candidatos que investiram muitas horas em cursos/treinamentos podem estar **se preparando para transições de carreira** ou buscando melhorar suas skills para um novo emprego. *Recomendação:* O RH pode monitorar funcionários com carga alta de treinamento externo, pois podem estar mais inclinados a sair. Em recrutamento, aqueles que possuem muitos cursos extracurriculares podem estar demonstrando proatividade em mudar de emprego ou avançar na carreira – podem ser bons alvos para abordagem, pois mostram ambição e preparo.
- **Localidade (Índice de desenvolvimento da cidade)**: Vimos que profissionais de cidades menos desenvolvidas têm mais intenção de mudar de emprego, possivelmente para buscar oportunidades melhores. *Recomendação:* Empresas localizadas em grandes centros podem **focar programas de recrutamento em cidades emergentes** ou menos desenvolvidas, pois os talentos de lá podem estar

mais dispostos a se relocarem ou a aceitar propostas. Isso amplia o pool de candidatos e atende a uma motivação deles de crescer profissionalmente.

- **Experiência e Histórico de Emprego**:
    - Profissionais **muito juniores** (pouca experiência, ou que nunca trocaram de emprego antes) estão em destaque na busca por vagas – possivelmente buscando o primeiro salto na carreira ou melhores condições.
    - Já profissionais **muito experientes** ou há muito tempo numa empresa tendem a ficar mais.

      Recomendação: O RH pode ajustar sua estratégia de talentos conforme a senioridade:

        - Para retenção: saber que recém-contratados ou jovens talentos têm maior risco de rotatividade, a empresa pode criar planos de carreira e mentoria robustos nos primeiros anos para engajá-los e **reduzir a vontade de sair**.
        - Para recrutamento: candidatos com 1-5 anos de experiência talvez estejam no pico de busca por crescimento; são bons alvos para vagas abertas pois muitos estão avaliando o mercado. Aqueles com 0 anos de experiência relevante (mudando de área) também compõem um grupo interessado em novas oportunidades.

- **Ambiente de trabalho atual (tamanho e tipo da empresa)**:
    - Pessoas em startups iniciais ou empresas muito pequenas mostraram mais intenção de sair – possivelmente em busca de mais estabilidade ou recursos de empresas maiores.
    - Aqueles em grandes empresas ou startups bem financiadas saem menos, talvez satisfeitos com benefícios ou crescimento interno.

      Recomendação: Se sua empresa é de grande porte, provavelmente atrairá profissionais de empresas pequenas que estão buscando upgrade. Tenha um discurso de recrutamento valorizando a estabilidade, infraestrutura e plano de carreira que uma grande empresa oferece. Por outro lado, se você é uma startup recrutando, destaque a inovação e crescimento acelerado – pois vimos que mesmo em startups, se forem atraentes (bem financiadas), conseguem segurar talentos.

- **Formação Educacional**:
    - Graduados e com formações em negócios ou tecnologia (STEM) mostraram uma leve maior propensão a buscar novas vagas em comparação a mestres/doutores ou formados em áreas menos demandadas. Isso sugere que profissionais com formação muito valorizada (ex: STEM) ou em início de carreira acadêmica (Graduação) têm mais opções e movimentação.

      Recomendação: Para seleção, não subestimar candidatos apenas com graduação – eles estão motivados e ativamente procurando. Já candidatos com pós-graduação avançada podem ser mais exigentes mas talvez menos inclinados a sair de onde estão; abordá-los requer mostrar desafios compatíveis e crescimento técnico.

Em resumo, o modelo e a EDA apontam que **experiência profissional, localização e contexto atual do candidato** são fatores-chave. O RH pode usar esses insights de diversas formas:

- **Triagem de candidatos**: por exemplo, ao analisar muitos currículos, priorizar para entrevista aqueles que batem com o perfil que costuma estar mais aberto a novas vagas (ex: X anos de experiência, setor atual, etc.), aumentando chances de encontrar alguém realmente interessado.
- **Estratégias de retenção**: focar em grupos vulneráveis à saída (novos contratados, pessoal em locais remotos com poucas oportunidades locais, etc.) com ações de engajamento.
- **Marketing de vagas direcionado**: saber onde anunciar vagas ou quais grupos abordar no LinkedIn. Ex: uma campanha direcionada a profissionais de cidades específicas ou de empresas de certo porte pode gerar mais candidatos qualificados interessados.
- **Personalização do pitch**: entendendo motivações prováveis. Alguém de empresa pequena quer ouvir sobre estabilidade; alguém muito junior quer ouvir sobre crescimento e aprendizado; alguém de cidade pequena talvez valorize relocação ou trabalho remoto.

5.3 Recomendações baseadas na análise

1. **Focar em candidatos com perfil de alto potencial de mudança**: Por exemplo, candidatos com 2-5 anos de experiência, que recentemente fizeram muitos cursos ou treinamentos, e que trabalham em empresas menores, tendem a estar em busca de crescimento. Esses são "baixos frutos pendentes" para processos seletivos – têm boa qualificação e alto interesse em novas oportunidades.
2. **Expandir recrutamento geográfico**: Considerar talentos de cidades com menor índice de desenvolvimento, oferecendo possibilidades de relocação ou trabalho remoto. A análise sugere que esses candidatos estão mais dispostos a mudar por boas oportunidades.
3. **Balancear gênero não é prioridade aqui**: Gênero não se mostrou fator determinante na mudança de emprego. Assim, estratégias de seleção podem se concentrar mais em experiência e formação do que em gênero – claro, mantendo diversidade por outros motivos importantes, mas sabendo que a predisposição à mudança é similar entre eles.
4. **Utilizar dados de treinamento/certificações**: Internamente, monitorar empregados que buscam muita capacitação pode dar sinais prévios de quem pode estar insatisfeito ou almejando promoção/mudança. Proativamente, o RH pode conversar sobre planos de carreira com esses funcionários para tentar mantê-los ou se preparar para reposição. Em seleção, candidatos com muitas certificações recentes indicam alguém ativo no desenvolvimento profissional e possivelmente aberto a novas posições.
5. **Simplificar requisitos para juniores**: Já que profissionais em início de carreira estão procurando se movimentar, empresas podem aproveitar para recrutá-los investindo em treinamento interno. Eles trazem vontade de crescer (indicada pela alta taxa de busca de vagas), então podem se tornar bons talentos de longo prazo se a empresa suprir essa necessidade de crescimento internamente.

6. Conclusões

Concluímos que é possível identificar padrões claros no perfil dos candidatos mais propensos a buscar novas oportunidades. Em suma:

- **Experiência e trajetória importam**: Candidatos mais juniores ou que nunca mudaram de emprego antes se mostraram mais abertos a mudanças, enquanto veteranos tendem a permanecer em seus cargos atuais.
- **Contexto geográfico e organizacional**: Aqueles em regiões menos desenvolvidas ou em empresas de pequeno porte/startups tendem a buscar ascensão em empresas melhores ou maiores. Já quem está em ambientes mais favoráveis (grandes empresas, cidades desenvolvidas) tende a ficar a menos que tenha outro motivo.
- **Formação e habilidades**: Embora todos os níveis possam buscar vagas, notamos que profissionais com apenas graduação se movimentam mais que mestres/doutores, possivelmente porque estão em fase de crescimento de carreira. Habilidades adquiridas via treinamento também sinalizam ambição de mudança.
- **Desempenho do modelo**: Um modelo básico *Random Forest* conseguiu capturar esses padrões com ~77% de acurácia. No entanto, ele enfrentou dificuldade em identificar todos os candidatos propensos (recall de ~41% para a classe 1), indicando que há espaço para melhorar o modelo e talvez incorporar mais dados ou técnicas de balanceamento.

Próximos passos:

- **Aprimoramento do modelo**: experimentar técnicas para lidar com o desbalanceamento (re-amostragem, ajuste de threshold, ou usar algoritmos como XGBoost que permitem tuning fino) e testar outros modelos (por exemplo, uma Regressão Logística para ter uma visão mais linear dos coeficientes, ou uma árvore de decisão simples para extração de regras diretas).
- **Validação cruzada**: utilizar validação cruzada para garantir que os resultados se generalizam e não são específicos do split treino/teste escolhido.
- **Feature Engineering**: criar novas features a partir das existentes, por exemplo: agrupar cidades por região ou faixa de city_development_index, ou extrair alguma interação entre experience e education_level (talvez o efeito da educação varia com anos de experiência).
- **Análise mais aprofundada por subgrupos**: por exemplo, separar a análise de acordo com área de atuação (major_discipline) ou gênero para ver se os padrões se mantêm, garantindo que as recomendações sejam universais ou se precisamos de estratégias específicas.
- **Monitorar modelos em produção**: caso o RH use esse modelo para priorizar candidatos, seria importante monitorar ao longo do tempo se os padrões dos candidatos mudam (por exemplo, tendências de mercado podem mudar a dinâmica, e o modelo precisaria ser atualizado).

Em conclusão, a análise atingiu o objetivo de identificar características dos candidatos mais inclinados a mudar de emprego.

Essas informações fornecem ao RH uma base **orientada por dados** para otimizar processos seletivos – desde onde buscar candidatos até como abordá-los e retê-los.

Com refinamentos futuros, poderemos aumentar ainda mais a assertividade das predições e das ações de RH decorrentes delas. Boa prática de análise de dados envolve esse ciclo contínuo de exploração, modelagem, avaliação e refinamento, sempre aliado ao contexto de negócio para tomar decisões melhores.