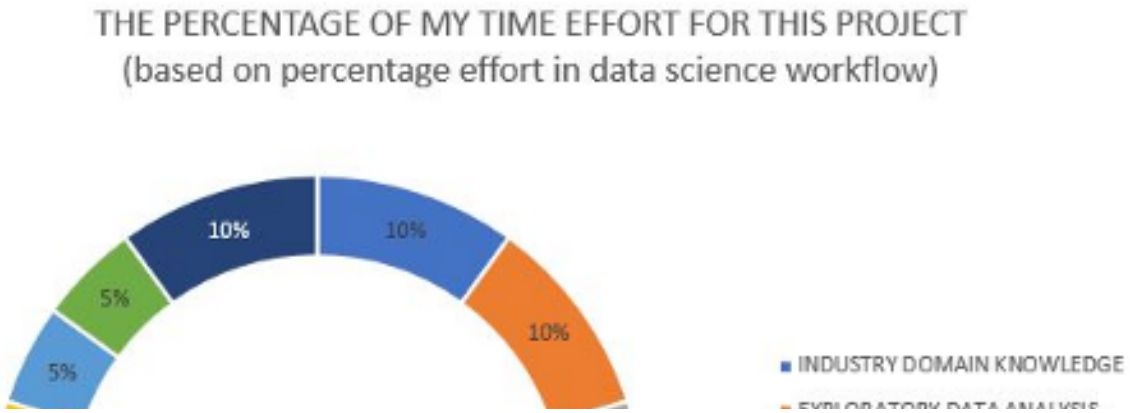
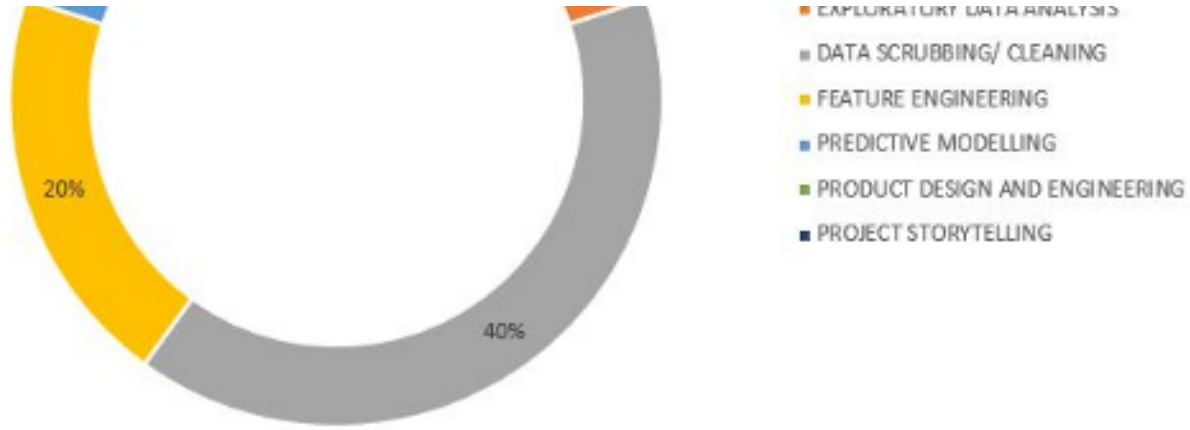
Feature engineering

O ofício de criar novas features (ou colunas) a partir das features existentes

Seja bem vinda, pessoa de Ciência de Dados!
Neste texto vamos abordar um tópicos que mais demandam a atenção, a criatividade e a energia de profissionais de data science. Sim, se você nunca ouviu falar de **feature engineering** sabia que, esta é uma das etapas mais demoradas e que demandam maior atenção e conhecimento do negócio no workflow de data scientists, ficando atrás apenas da limpeza de dados ou data cleaning.

Vamos explorar alguns conceitos e aplicações, mas lembrando que assim como EDA, nunca vai existir um guia definitivo sobre feature engineering devido a natureza dinâmica e suas mais diversas necessidades e aplicações.





Percentual de tempo gasto pelo Cientista de Dados Andrew Wong em um determinado projeto. Fonte: https://medium.com/human-science-ai/how-i-spent-my-time-as-product-data-scientist-90e760044cd7

Seguindo nessa linha sabemos que cada projeto de dado tem suas características e particularidades. Alguns, por exemplo, se quer possuem os dados. Mas considerando que os dados já existam, este gráfico ao lado pode ser uma boa estimativa de tempo gasto em cada etapa de um projeto de data science. **Estamos falando de 20% do tempo em feature engineering.**

Ok, mas o que é Feature Engineering?

Imagine que você tem um dataset de imóveis que possui diversos features ou características dos imóveis como o preço de venda, área do imóvel, endereço do imóvel, endereço da estação de metrô mais próxima, ano de construção e outros. Baseado nessas informações existentes, é possível criar novas colunas com informações adicionais como: preço do m^2, distância do metrô, idade do imóvel, bairro, tipo de logradouro etc. Isso é feature engineering.

Digamos que você decida categorizar os salários em faixas salariais com scores de 1 a 5 em um dataset de segmentação de clientes. Ou que precise extrair apenas as horas de uma coluna de data e hora para criar um nova coluna “turno” para um dataset de escala de funcionários. Isso é feature engineering.

O foco do nosso texto de feature engineering será o pré-processamento de dados para os modelos de machine learning e a criação de novas features que ajudam a explicar o problema.

Imagine que seus dados precisam ser preparados para entrar em um modelo como o de regressão linear. Logo, nenhum dos dados podem ser diferentes de numéricos. Como faremos isso? Rá! Feature Engineering.

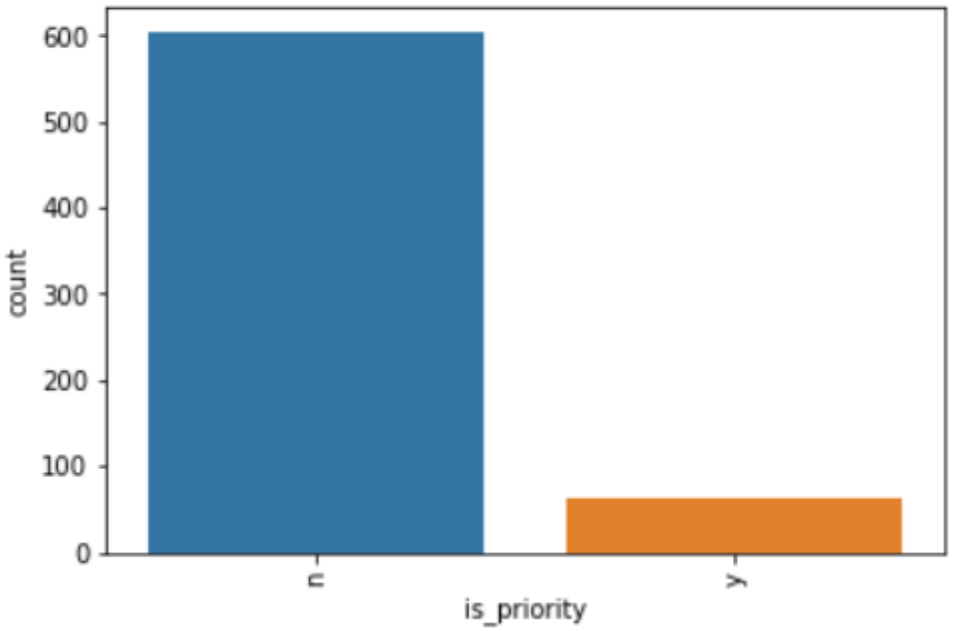
Implementação de feature engineering para dados categóricos

Nestes 2 exemplos vamos utilizar um dataset de oportunidades de trabalho voluntários da cidade de Nova York.

Features binárias com LabelEncoder()

Quando uma feature é binária, podemos assumir um valor de 0 ou 1 para a todas as observações em uma nova coluna. Essa nova coluna será correspondente à original, mas estará transformada pelo LabelEncoder.
No nosso caso temos a coluna `is_priority` .

```
sns.countplot(df.is_priority)
plt.xticks(rotation=90)
plt.show()
```



Após importar o LabelEncoder() e instanciá-lo, podemos aplicar o método .fit_transform() e criar uma nova coluna para receber o dado codificado. Como as prioridades na feature em questão são mais esparsas, usamos o método .sample() ao invés do .head() ou . tail().

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df["priority_enc"] = le.fit_transform(df["is_priority"])
```

```
df[['is_priority', 'priority_enc']].sample(10)
```

	is_priority	priority_enc
301	n	0
239	n	0
47	y	1
64	n	0
402	n	0
238	n	0
372	n	0
460	y	1
554	n	0
289	n	0

Recorte de 10 exemplos aleatórios do dataset com as colunas is_priority, e sua versão codificada , priority_enc

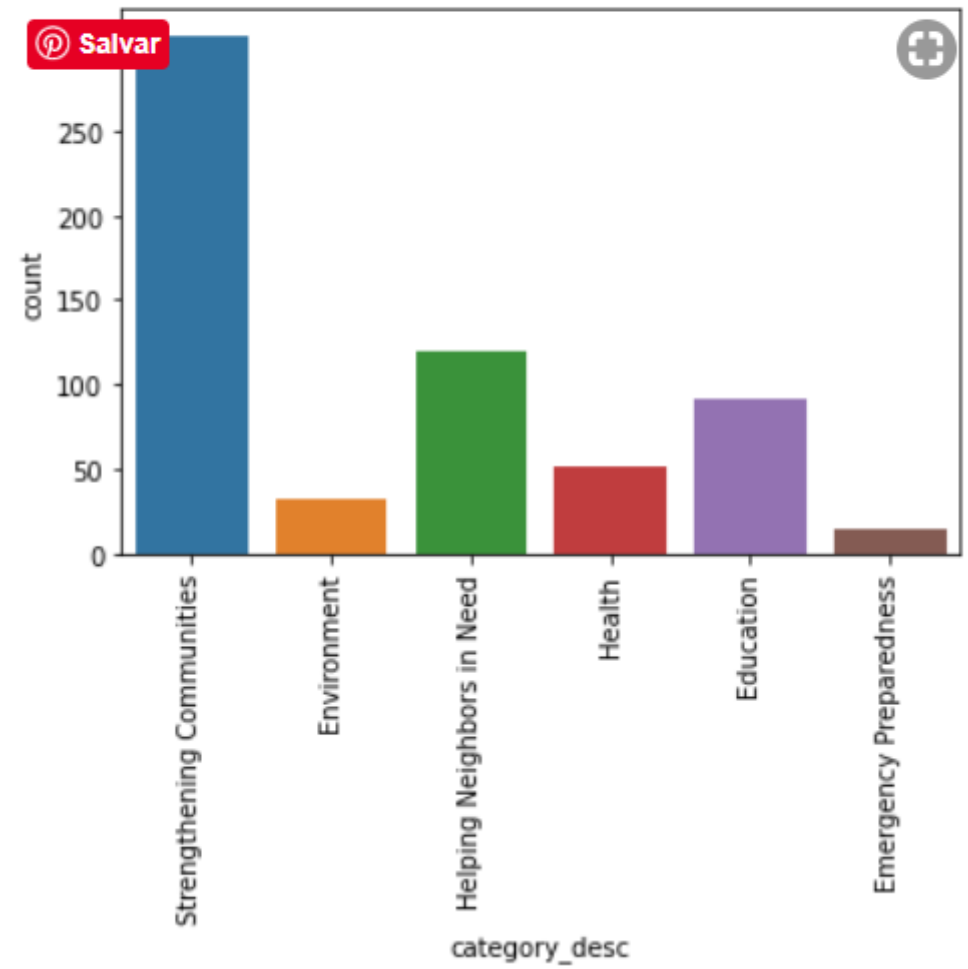
Features multi categóricas com pd.get_dummies()

Quando temos mais de dois diferentes dados categóricos em uma mesma coluna não podemos simplesmente aplicar um código para cada elemento de forma arbitrária. Os modelos de ML, de uma maneira geral, penalizariam os valores mais baixos e dariam mais pesos aos valores mais altos. Até certo ponto isso faz sentido se houver uma relação ordinal, de ranking ou de grandeza e se esses valores atribuídos representarem alguma relação com as categorias. Na maioria das vezes, não é o que desejamos. Nesses casos existem duas abordagens possíveis:

One-hot encoding:
Usando o pd.get_dummies() passamos a coluna que desejamos expandir em novas colunas com 0 ou 1 indicando a qual categoria determinada observação pertence. Ao indicar com zero e um, não penalizamos ou adicionamos peso a nenhuma categoria específica.

No nosso dataset de voluntários, vamos passar a coluna ‘category_desc’ e adicionar um prefixo ‘C_’ para facilitar a identificação das novas features.

```
sns.countplot(df.category_desc)
plt.xticks(rotation=90)
plt.show()
```



Usamos o pd.concat() para combinar as novas features com o dataset original.

```
category_enc = pd.get_dummies(df.category_desc, prefix='C', drop_first=False)
one_enc_vol = pd.concat([df, category_enc], axis=1)
```

```
one_enc_vol[['category_desc', 'C_Education', 'C_Emergency Preparedness', 'C_Environment', 'C_Health',
             'C_Helping Neighbors in Need', 'C_Strengthening Communities' ]].sample(10, random_state=0)
```

	category_desc	C_Education	C_Emergency Preparedness	C_Environment	C_Health	C_Helping Neighbors in Need	C_Strengthening Communities
266	Helping Neighbors in Need	0	0	0	0	1	0
334	Health	0	0	0	1	0	0
14	Helping Neighbors in Need	0	0	0	0	1	0
462	Strengthening Communities	0	0	0	0	0	1
75	Strengthening Communities	0	0	0	0	0	1
432	Helping Neighbors in Need	0	0	0	0	1	0
491	Education	1	0	0	0	0	0
642	Helping Neighbors in Need	0	0	0	0	1	0
644	Strengthening Communities	0	0	0	0	0	1
397	Strengthening Communities	0	0	0	0	0	1

Dummy encoding

Assim como o one-hot encoding, o dummy encoding usa o pd.get_dummies(). Na verdade eles são virtualmente iguais, exceto que o Dummy encoding utiliza um espaco de features (n-1) ou seja, ao invés de criar uma nova coluna para cada dado categórico

da coluna ele exclui o primeiro.
A lógica por trás dessa estratégia é muito simples: se temos 6 features, eu represento 5 delas em colunas. Para as colunas geradas, sempre haverá um 1 correspondendo na linha. Porém se houver apenas zeros nas cinco colunas, é porque estamos falando da coluna omitida. Vai ficar mais fácil de entender com o exemplo desse mesmo dataset.

```
category_enc = pd.get_dummies(df.category_desc, prefix='C', drop_first=True)
dummie_enc_vol = pd.concat([df, category_enc], axis=1)
```

```
dummie_enc_vol[['category_desc', 'C_Emergency Preparedness', 'C_Environment',
                'C_Health', 'C_Helping Neighbors in Need', 'C_Strengthening Communities' ]].sample(10, random_state=0)
```

	category_desc	C_Emergency Preparedness	C_Environment	C_Health	C_Helping Neighbors in Need	C_Strengthening Communities
266	Helping Neighbors in Need	0	0	0	1	0
334	Health	0	0	1	0	0
14	Helping Neighbors in Need	0	0	0	1	0
462	Strengthening Communities	0	0	0	0	1
75	Strengthening Communities	0	0	0	0	1
432	Helping Neighbors in Need	0	0	0	1	0
491	Education	0	0	0	0	0
642	Helping Neighbors in Need	0	0	0	1	0
644	Strengthening Communities	0	0	0	0	1
397	Strengthening Communities	0	0	0	0	1

Perceba que a coluna linha Education possui apenas zeros, uma vez que omitimos essa coluna

One-hot x Dummy

One-hot encoding(mais intuitivo):
Maior facilidade em explicar o papel de cada feature

Dummy encoding(melhor performance):
Reduz informações duplicadas e multicolinearidade

Esses dois approachs também podem ser usados através do OneHotEncoder() do sklearn. Confira os detalhes de implementação no artigo abaixo:

Categorical encoding using Label-Encoding and One-Hot-Encoder

Detailing 2 important approaches in Machine Learning to convert categorical data into numerical data

towardsdatascience.com

Criação de Features

Imagine que você tem um dataset com 16 features:

- **price** - The last price the house was sold for
- **num_bed** - The number of bedrooms
- **num_bath** - The number of bathrooms (fractions mean the house has a toilet-only or shower/bathtub-only bathroom)
- **size_house** (includes basement) - The size of the house
- **size_lot** - The size of the lot
- **num_floors** - The number of floors
- **is_waterfront** - Whether or not the house is a waterfront house (0 means it is not a waterfront house whereas 1 means that it is a waterfront house)
- **condition** - How worn out the house is. Ranges from 1 (needs repairs all over the place) to 5 (the house is very well maintained)
- **size_basement** - The size of the basement
- **year_built** - The year the house was built
- **renovation_date** - The year the house was renovated for the last time. 0 means the house has never been renovated
- **zip** - The zip code
- **latitude** - Latitude
- **longitude** - Longitude
- **avg_size_neighbor_houses** - The average house size of the neighbors
- **avg_size_neighbor_lot** - The average lot size of the neighbors

Descrição das features iniciais do dataset

A partir dessas features base, vamos criar uma série de novas features que vão ajudar o nosso modelo a ser mais preciso sem falar que estas novas features podem implicar em variáveis com uma alta correlação com o preço dos imóveis, a nossa variável target.

Idade do imóvel

A idade do imóvel é um dos fatores que mais influenciam o seu preço de venda. De uma maneira geral, imóveis novos valem mais que imóveis mais antigos. Então vamos criar essa variável.

```
#Para criar a building age vamos calcular a diferença entre o ano atual e a idade do imóvel (recent_year)
from datetime import datetime
df["building_age"]= datetime.today().year - df["year_built"]
```

```
df["building_age"].value_counts().head()
```

```
6     468
14    380
16    378
15    372
17    367
Name: building_age, dtype: int64
```

Dummy Reforma

Porém, se um imóvel foi reformado, isso também pode influenciar em seu preço. Vamos criar uma dummy informando se o imóvel foi reformado ou não.

```
# Criando uma DUMMY para representar se a casa foi ou não reformada
df['dummy_reforma'] = np.where(df["renovation_date"]>0, 1, 0)
```

Índice grama do vizinho

Você já deve ter ouvido a expressão de que a grama do vizinho é sempre mais verde. Nesse caso específico se o seu terreno é maior do que a média do bairro, isso pode impactar no preço do imóvel.

```
df['indice_grama_viz'] = df['size_lot'] / df['avg_size_neighbor_lot']
```

Índice inveja mora ao lado

Agora se a sua casa é a maior da rua.. ah… isso não tem preço, ou melhor, tem sim e ele é alto. Pelo menos mais alto do que a média do resto da rua. Para calcular este índice vamos precisar de 3 steps:

1- Calcular o a média dos preços das casas de acordo com o CEP em que elas se encontram:

```
df_zip = df.groupby(['zip']).agg({'size_house':'mean'})
df_zip.shape
```

```
(70, 1)
```

```
df_zip.head()
```

zip	size_house
98001	1883.990415
98002	1616.218935
98003	1897.203320
98004	2945.601504
98005	2653.167832

2- Fazer um pd.merge() passando o CEP como chave para que cada casa possa ter a média do preço das casas vizinhas na mesma linha.

```
df = pd.merge(df, df_zip, how='inner', left_on='zip', right_index=True)
```

```
df.head()
```

	rowid	price	num_bed	num_bath	size_house_x	size_lot	num_floors	is_waterfront	condition	siz
0	0	221900	3	1.00	1180	5650	1.0	0	3	
100	100	205425	2	1.00	880	6780	1.0	0	4	
108	108	445000	3	2.25	2100	8201	1.0	0	3	
230	230	236000	3	1.00	1300	5898	1.0	0	3	
237	237	170000	2	1.00	860	5265	1.0	0	3	

Como agora teríamos duas colunas com o mesmo nome (size_house), automaticamente o Pandas criou dois sufixos _x para o size_house original e _y para o size_house recém criado.

3- Criar o índice e adicioná-lo ano dataset a partir da proporção da sua casa x a casa dos seus vizinhos de rua:

```
df['inveja_mora_ao_lado'] = df['size_house_x'] / df['size_house_y']
```

Mansão de frente para a água

Todo mundo gosta de morar de frente para a água. Pode ser beira-mar, beira rio ou nas margens de um lago. Esse fator é pode impactar bastante no preço de um imóvel. Mas nem todos os imóveis que estão de frente para a água possuem um alto valor. Eventualmente chácaras e sítios possuem terrenos enormes, estão de frente para a água, mas não estão em terrenos muito valorizados. Totalmente diferente é se a casa possui pelo menos um segundo andar e está de frente para uma fonte de água. Vamos criar esse índice.

```
df['festa_no_yacht'] = df['is_waterfront'] * df['num_floors']
```

```
df['festa_no_yacht'].value_counts()
```

```
0.0    18307
2.0       66
1.0       45
1.5       21
3.0        7
2.5        2
```

Feature Engineering Automatizado

O processo de feature engineering pode ser algo longo, lento, manual e embora extremamente necessário pode testar a sua memória e o seu grau de conhecimento sobre o negócio em que o problema de negócio está inserido. Estamos dizendo isso, considerando apenas um dataframe. Agora imagina, como é muito comum na área de

dados, que estejamos falando de múltiplas tabelas relacionais (que podem ser relacionadas entre si por uma chave única). Considere o seguinte exemplo: um banco que possui 3 tabelas, sendo uma com os dados dos clientes, uma com os empréstimos tomados anteriormente e uma última com o registro de pagamentos e atrasos.

Primeiro vamos importar as tabelas (em csv)e transformá-las em dataframes:

```
# Read in the data
clients = pd.read_csv('data/clients.csv', parse_dates = ['joined'])
loans = pd.read_csv('data/loans.csv', parse_dates = ['loan_start', 'loan_end'])
payments = pd.read_csv('data/payments.csv', parse_dates = ['payment_date'])
```

Agora vamos dar uma olhada nelas:

```
clients.head(10)
```

	client_id	joined	income	credit_score
0	46109	2002-04-16	172677	527
1	49545	2007-11-14	104564	770
2	41480	2013-03-11	122607	585
3	46180	2001-11-06	43851	562
4	25707	2006-10-06	211422	621
5	39505	2011-10-14	153873	610
6	32726	2006-05-01	235705	730
7	35089	2010-03-01	131176	771
8	35214	2003-08-08	95849	696
9	48177	2008-06-09	190632	769

```
loans.sample(10)
```

	client_id	loan_type	loan_amount	repaid	loan_id	loan_start	loan_end	rate
275	44601	home	4930	0	10732	2003-02-15	2005-01-09	3.28
268	44601	credit	10120	0	10177	2001-11-03	2004-02-13	2.42
154	35089	cash	2568	1	11418	2006-01-04	2008-06-15	1.76
184	48177	home	5818	1	10261	2014-08-26	2017-01-26	1.22
82	25707	other	2477	1	10009	2009-05-25	2011-02-17	1.82
306	49068	credit	6620	0	11953	2012-02-12	2014-06-19	0.05
168	35214	home	3852	0	11147	2005-09-20	2007-09-22	2.10
269	44601	cash	2003	1	11173	2008-09-13	2011-03-05	1.25
7	46109	home	12656	0	11658	2006-05-26	2007-10-15	4.14
131	32726	credit	14522	0	10202	2005-06-23	2007-07-01	3.74

```
payments.sample(10)
```

	loan_id	payment_amount	payment_date	missed
2061	10798	137	2009-08-25	0
284	10302	355	2006-04-30	0
3044	11353	112	2013-11-22	0
2544	10827	548	2006-07-16	0
330	11328	2120	2005-09-13	0
1900	11716	1610	2010-11-28	1
1848	11827	2573	2005-05-28	1
1375	10956	630	2001-09-05	1
176	11247	2273	2011-12-10	0
2573	10872	1915	2006-07-19	1

Ok! Vamos aquecer. Vou começar criando duas novas features manualmente: 1. Extraindo o mês da data de entrada do cliente.
2. Aplicando log na renda do cliente.

```
# Create a month column
clients['join_month'] = clients['joined'].dt.month

# Create a log of income column
clients['log_income'] = np.log(clients['income'])

clients.head()
```

	client_id	joined	income	credit_score	join_month	log_income
0	46109	2002-04-16	172677	527	4	12.059178
1	49545	2007-11-14	104564	770	11	11.557555
2	41480	2013-03-11	122607	585	3	11.716739

3	46180	2001-11-06	43851	562	11	10.688553
4	25707	2006-10-06	211422	621	10	12.261611

Até aqui tudo bem, nada de novo. Agora vamos incorporar o valor médio, máximo e minimo dos empréstimos (na tabela loans) à tabela clientes. Isso vai ser uma operação em dois steps. Primeiro vamos agrupar os valores médio, máximo e mínimo em torno do id de cada cliente:

```
# Groupby client id and calculate mean, max, min previous loan size
stats = loans.groupby('client_id')['loan_amount'].agg(['mean', 'max', 'min'])
stats.columns = ['mean_loan_amount', 'max_loan_amount', 'min_loan_amount']
stats.head()
```

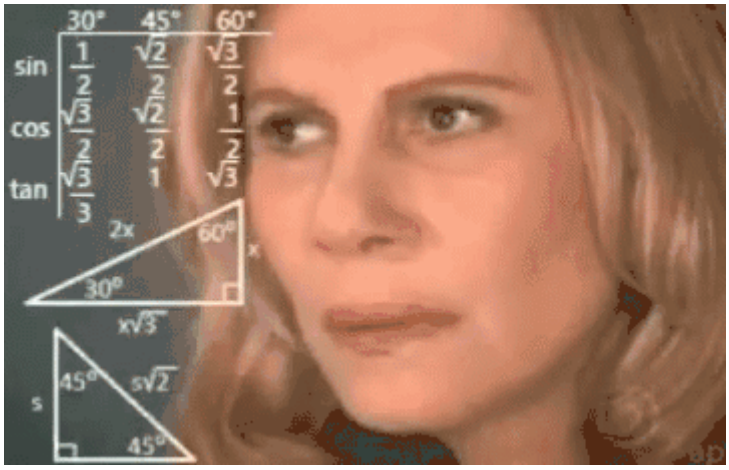
client_id	mean_loan_amount	max_loan_amount	min_loan_amount
25707	7963.950000	13913	1212
26326	7270.062500	13464	1164
26695	7824.722222	14865	2389
26945	7125.933333	14593	653
29841	9813.000000	14837	2778

Agora vamos fazer um pd.merge() passando o id do cliente. Dessa forma incorporamos todas as colunas criadas (valor médio dos empréstimos, valor máximo do empréstimo e valor do empréstimo mínimo).

```
# Merge with the clients dataframe
clients.merge(stats, left_on = 'client_id', right_index=True, how = 'left').head(10)
```

	client_id	joined	income	credit_score	join_month	log_income	mean_loan_amount	max_loan_amount	min_loan_amount
0	46109	2002-04-16	172677	527	4	12.059178	8951.600000	14049	559
1	49545	2007-11-14	104564	770	11	11.557555	10289.300000	14971	3851
2	41480	2013-03-11	122607	585	3	11.716739	7894.850000	14399	811
3	46180	2001-11-06	43851	562	11	10.688553	7700.850000	14081	1607
4	25707	2006-10-06	211422	621	10	12.261611	7963.950000	13913	1212
5	39505	2011-10-14	153873	610	10	11.943883	7424.050000	14575	904
6	32726	2006-05-01	235705	730	5	12.370336	6633.263158	14802	851
7	35089	2010-03-01	131176	771	3	11.784295	6939.200000	13194	773
8	35214	2003-08-08	95849	696	8	11.470529	7173.555556	14767	667
9	48177	2008-06-09	190632	769	6	12.158100	7424.368421	14740	659

Ufa! Mas e se quiséssemos investigar as médias dos últimos pagamentos de cada cliente?



Feature Tools

Calma, para isso existe o feature tools. Essa ferramenta permite criar uma incrível combinação de features realizando agregações e transformações em uma ou mais camadas de profundidade, **tudo isso de forma automatizada!**

> *Eu sei que essa última frase ficou meio abstrata. Mas vamos aos exemplos e tudo vai ficar mais claro.*

Instalando
O feature tools não é uma pacote que vem instalado no anaconda por default.
Para instalar direto do notebook:

```
!pip install featuretools
```

ou direto do prompt para Windows ou terminal para Mac e Linux:

```
conda install -c conda-forge featuretools
```

Antes de melhorar, vai complicar um pouco!
O Feature Tools não é parte da biblioteca do Scikit-Learn, ou pandas. Ele usa princípios de relações entre cada tabela ou Entity do SQL, mas ao invés de uma consulta única, ele permite a expansão de features a partir de aggregações e transformações. Vamos importar o pacote e iniciar uma EntitySet, ou seja, uma espécie de objeto que vai

guardar qual é a key id única de todo o conjunto de tabelas, assim como todas as relações entre essas tabelas ou entidades. Quando chamado, ele vai ter essas características:

```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    loans.client_id -> clients.client_id
    payments.loan_id -> loans.loan_id
```

Vamos construir esse objeto passo a passo.

1. Importando o pacote e criando a EntitySet:

```
# featuretools for automated feature engineering
import featuretools as ft

es = ft.EntitySet(id = 'clients')
```

2. Construindo as entidades:

```
# Create an entity from the client dataframe
# This dataframe already has an index and a time index
es = es.entity_from_dataframe(entity_id = 'clients', dataframe = clients,
                              index = 'client_id', time_index = 'joined')
```

```
# Create an entity from the loans dataframe
# This dataframe already has an index and a time index
es = es.entity_from_dataframe(entity_id = 'loans', dataframe = loans,
                              variable_types = {'repaid': ft.variable_types.Categorical},
                              index = 'loan_id',
                              time_index = 'loan_start')
```

```
# Create an entity from the payments dataframe
# This does not yet have a unique index
es = es.entity_from_dataframe(entity_id = 'payments',
                              dataframe = payments,
                              variable_types = {'missed': ft.variable_types.Categorical},
                              make_index = True,
                              index = 'payment_id',
                              time_index = 'payment_date')
```

Vamos criar cada entidade a partir dos dataframes que importamos lá em cima (clientes, loans e payments). Note que é preciso sinalizar com um dicionário as variáveis categóricas do dataframe, informar

o index e o time index, e ainda, criar uma nova coluna com o argumento make_index= True para nos certificarmos de que todas as tabelas tenham um índice.

Vamos inspecionar como elas estão:

```
es
```

```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    No relationships
```

```
es['clients']
```

```
Entity: clients
  Variables:
    client_id (dtype: index)
    joined (dtype: datetime_time_index)
    income (dtype: numeric)
    credit_score (dtype: numeric)
    join_month (dtype: numeric)
    log_income (dtype: numeric)
  Shape:
    (Rows: 25, Columns: 6)
```

```
es['loans']
```

```
Entity: loans
  Variables:
    loan_id (dtype: index)
    client_id (dtype: numeric)
    loan_type (dtype: categorical)
    loan_amount (dtype: numeric)
    loan_start (dtype: datetime_time_index)
    loan_end (dtype: datetime)
    rate (dtype: numeric)
    repaid (dtype: categorical)
  Shape:
    (Rows: 443, Columns: 8)
```

```
es['payments']
```

```
Entity: payments
  Variables:
    payment_id (dtype: index)
    loan_id (dtype: numeric)
    payment_amount (dtype: numeric)
    payment_date (dtype: datetime_time_index)
    missed (dtype: categorical)
  Shape:
    (Rows: 3456, Columns: 5)
```

Conseguimos ver o shape dos dados e os tipos de dados em cada uma das colunas, mas ainda não temos a relação entre as entidades, ou seja, como as tabelas se relacionam entre si. Este é o próximo passo.

3. Criando as relações:

```
# Relação entre clients e loans
r_client_previous = ft.Relationship(es['clients']['client_id'],
                                    es['loans']['client_id'])

# Add the relationship to the entity set
es = es.add_relationship(r_client_previous)
```

```
# Relação entre loans e payments
r_payments = ft.Relationship(es['loans']['loan_id'],
                             es['payments']['loan_id'])

# Add the relationship to the entity set
es = es.add_relationship(r_payments)

es
```

Resultado:

```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    loans.client_id -> clients.client_id
    payments.loan_id -> loans.loan_id
```

Criando features automatizados:
O Feature Tools, para criar novas features, usa o conceito de Feature Primitives, que são operações simples, muitas delas bem conhecidas, que podem ser empilhadas em vários níveis para criar features muito mais complexas.

Feature Primitives

As feature primitives podem ser agrupadas em duas categorias.

Agregações: funções que agrupam dados 'filhos' para cada dado 'pai' passando um calculo estatístico como mínimo, média ou desvio padrão. Um exemplo é calcular o empréstimo máximo dado a cada cliente. Uma agregação se dá através de múltiplas tabelas, usando a relação entre as tabelas.

	name	type	dask_compatible	description
20	num_unique	aggregation	True	Determines the number of distinct values, ignoring `NaN` values.
8	skew	aggregation	False	Computes the extent to which a distribution differs from a normal distribution.
14	std	aggregation	True	Computes the dispersion relative to the mean value, ignoring `NaN`.
11	max	aggregation	True	Calculates the highest value, ignoring `NaN` values.
19	all	aggregation	True	Calculates if all values are 'True' in a list.
5	avg_time_between	aggregation	False	Computes the average number of seconds between consecutive events.
0	median	aggregation	False	Determines the middlemost number in a list of values.
13	time_since_last	aggregation	False	Calculates the time elapsed since the last datetime (default in seconds).

16	n_most_common	aggregation	False	Determines the `n` most common elements.
21	num_true	aggregation	True	Counts the number of `True` values.

Exemplo de agregações. Temos 22 agregações diferentes possíveis no Feature Tools.

Transformações: uma operação aplicada a uma ou mais colunas de uma única tabela. Um exemplo seria extrair o dia de uma coluna de datas ou calcular a diferença entre duas colunas de uma tabela.

	name	type	dask_compatible	description
53	less_than	transform	True	Determines if values in one list are less than another list.
35	not_equal	transform	False	Determines if values in one list are not equal to another list.
57	multiply_numeric	transform	True	Element-wise multiplication of two lists.
78	diff	transform	False	Compute the difference between the value in a list and the
51	cum_sum	transform	False	Calculates the cumulative sum.
72	percentile	transform	False	Determines the percentile rank for each value in a list.
27	cum_count	transform	False	Calculates the cumulative count.
62	longitude	transform	False	Returns the second tuple value in a list of LatLong tuples.
46	year	transform	True	Determines the year value of a datetime.
52	cum_max	transform	False	Calculates the cumulative maximum.

Exemplos de transformações. Temos 57 transformações possíveis no Feature Tools.

Feature Tools em ação

Vamos passar 5 agregações (mean, max, mode, percent_true e last) e 2 transformações (month, year) especificando a quantidade máxima de camadas em 2.

```
# Create new features using specified primitives
features, feature_names = ft.dfs(entityset = es, target_entity = 'clients',
                                 agg_primitives = ['mean', 'max', 'mode', 'percent_true', 'last'],
                                 trans_primitives = ['month', 'year'],
                                 max_depth = 2)
                                 #agg_primitives = ['mean', 'max', 'percent_true', 'last'],
                                 #trans_primitives = ['year', 'month', 'subtract_numeric', 'divide_numeric'])
features.head()
```

client_id	income	credit_score	join_month	log_income	MEAN(loans.loan_amount)	MEAN(loans.rate)	MAX(loans.loan_amount)	MAX(loans.rate)
42320	229481	563	4	12.343576	7062.066667	2.457333	13887	6.74
39384	191204	617	6	12.161096	7865.473684	3.538421	14654	9.23
26945	214516	806	11	12.276140	7125.933333	2.855333	14593	5.65
41472	152214	638	11	11.933043	7510.812500	3.981250	13657	9.82
46180	43851	562	11	10.688553	7700.850000	3.502500	14081	9.26

5 rows × 60 columns

Woww!!! Geramos 60 features de uma só vez. Vamos explorar quais são elas...

Listando todas as colunas geradas:

```
features.columns.to_list()
```

```
['income',
 'credit_score',
 'join_month',
 'log_income',
 'MEAN(loans.loan_amount)',
 'MEAN(loans.rate)',
 'MAX(loans.loan_amount)',
 'MAX(loans.rate)',
 'MODE(loans.loan_type)',
 'MODE(loans.repaid)',
 'LAST(loans.loan_type)',
 'LAST(loans.repaid)',
 'LAST(loans.loan_amount)',
 'LAST(loans.rate)',
 'LAST(loans.loan_id)',
 'MEAN(payments.payment_amount)',
 'MAX(payments.payment_amount)',
 'MODE(payments.missed)',
 'LAST(payments.payment_amount)',
 'LAST(payments.payment_id)',
 'LAST(payments.missed)',
 'MONTH(joined)',
 'YEAR(joined)',
 'MEAN(loans.LAST(payments.payment_amount))',
 'MEAN(loans.MAX(payments.payment_amount))',
 'MEAN(loans.MEAN(payments.payment_amount))',
 'MAX(loans.LAST(payments.payment_amount))',
 'MAX(loans.MEAN(payments.payment_amount))',
 'MODE(loans.MONTH(loan_end))',
 'MODE(loans.MODE(payments.missed))',
 'MODE(loans.LAST(payments.missed))',
 'MODE(loans.YEAR(loan_end))',
 'MODE(loans.LAST(payments.payment_id))',
 'MODE(loans.MONTH(loan_start))',
 'MODE(loans.YEAR(loan_start))',
 'LAST(loans.MONTH(loan_end))',
 'LAST(loans.MODE(payments.missed))',
 'LAST(loans.YEAR(loan_end))',
 'LAST(loans.MAX(payments.payment_amount))',
 'LAST(loans.MONTH(loan_start))',
 'LAST(loans.MEAN(payments.payment_amount))',
 'LAST(loans.YEAR(loan_start))',
 'MEAN(payments.loans.loan_amount)',
 'MEAN(payments.loans.rate)',
 'MAX(payments.loans.loan_amount)',
 'MAX(payments.loans.rate)',
 'MODE(payments.loans.client_id)',
 'MODE(payments.loans.repaid)',
 'MODE(payments.loans.loan_type)',
```

```
'LAST(payments.loans.client_id)',
'LAST(payments.loans.repaid)',
'LAST(payments.loans.rate)',
'LAST(payments.loans.loan_amount)',
'LAST(payments.loans.loan_type)',
'MONTH(LAST(loans.loan_start))',
'MONTH(LAST(payments.payment_date))',
'MONTH(LAST(loans.loan_end))',
'YEAR(LAST(loans.loan_start))',
'YEAR(LAST(payments.payment_date))',
'YEAR(LAST(loans.loan_end))']
```

Ok! Mas o que quer dizer o `max_depth = 2` !?

As features podem ter a profundidade de uma camada, como por exemplo, a média dos pagamentos feitas por cada cliente:

```
# Show a feature with a depth of 1
pd.DataFrame(features['MEAN(payments.payment_amount)'].head(10))
```

client_id	MEAN(payments.payment_amount)
42320	1021.483333
39384	1193.630137
26945	1109.473214
41472	1129.076190
46180	1186.550336
46109	1375.560284
32885	1396.495652
29841	1439.433333
38537	1340.682927
35214	1076.987952

Ou ainda o último pagamento feito por cada cliente:

```
# Show a feature with a depth of 1
pd.DataFrame(features['LAST(payments.payment_amount)'].head(10))
```

LAST(payments.payment_amount)

client_id	
42320	1082
39384	2045
26945	1597
41472	1453
46180	697
46109	1827
32885	1909
29841	800
38537	1615
35214	1615

Mas adicionando uma segunda camada nós teríamos a média dos últimos pagamentos realizados por cada cliente:

```
# Show a feature with a depth of 2
pd.DataFrame(features['LAST(loans.MEAN(payments.payment_amount))']).head(10)
```

	LAST(loans.MEAN(payments.payment_amount))
client_id	
42320	1192.333333
39384	2311.285714
26945	1598.666667
41472	1427.000000
46180	557.125000
46109	1708.875000
32885	1729.000000
29841	1125.500000
38537	1348.833333
35214	1410.250000

Qual é o limite para a geração de features? Com a Feature Tools é basicamente a capacidade de processamento do seu computador e o quanto faz sentido combinar diferentes features em diferentes camadas de complexidade.

Conclusão

Que bom que você chegou até aqui!!! Agora você possui os conceitos e as técnicas de feature engineering para criar features novas a partir das já existentes no seu dataset.

Durante este texto abordamos como transformar features categóricas em dummies e label encoding. Vimos como criar novas features de forma manual seja transformando informações por meio de boleanos, fazendo cálculos entre as colunas ou criando features relacionais. Por fim abordamos o processo de criação de features em uma escala industrial com a ferramenta de geração de features automatizadas, o Feature Tools. Depois de tanto trabalhar no dataset talvez seja a hora de falar de modelos. Na verdade não é exatamente a hora. Isso é assunto para o nosso próximo texto onde vamos falar sobre ensembles.

Por enquanto eu vou ficando por aqui. Baixe os notebooks e datasets para testar o código e gerar novos experimentos. Te desafio a postar uma nova feature no grupo do slack explicando o princípio por trás dela. E nunca é demais lembrar que se tiver qualquer dúvida ou dificuldade, sempre terá alguém do time TERA pronto para te ajudar. Até mais!

Considerações finais: o que não consideramos Feature Engineering nesse texto (mas não é que seja um consenso entre data scientists)

- Transformações que envolvam a padronização de escalas ou a normalização dos dados.
- Imputação e estratégias para lidar com dados faltantes e nulos (data clean)
- Correção e imputação de labels ou valores na coluna target
- Seleção de features de acordo com sua importância ou extração de features (vamos abordar este assunto em redução de dimensionalidade)

Para saber mais:

Automated Feature Engineering in Python

How to automatically create machine learning features

towardsdatascience.com

Case de feature engineering no desafio do Titanic do Kaggle

Feature Engineering with Kaggle Tutorial

In the two previous Kaggle tutorials, you learned all about how to get your data in a form to build your first machine...

www.datacamp.com

A Practical Guide to Feature Engineering in Python

Learn the underlying techniques and tools for effective feature engineering in Python

heartbeat.fritz.ai

FEATURE ENGINEERING

Feature Engineering is a blanket term used for a bunch of activities performed on the features of the dataset making...

www.datavedas.com

Categorical encoding using Label-Encoding and One-Hot-Encoder

Detailing 2 important approaches in Machine Learning to convert categorical data into numerical data

towardsdatascience.com

Automated Feature Engineering in Python

How to automatically create machine learning features

towardsdatascience.com

Automated Feature Engineering Basics

Explore and run machine learning code with Kaggle Notebooks | Using data from multiple data sources

www.kaggle.com