Drawing

A Acme Inc. utiliza um serviço terceirizado de computação distribuída para a execução de diversas funções (código que executa alguma tarefa específica). O modelo de precificação deste fornecedor ocorre de acordo com o tempo que cada função permanece rodando dentro da plataforma.

Origem dos dados

Esta base de dados faz parte do projeto de Business Case da Hype para a vaga de Estágio em Dados. Todos os dados expostos são fictícios.

Observações

- **O modelo de precificação se baseia em custo por milissegundo de execução**
- **O custo é de R$0,000015/milissegundo**
- **O número de milissegundos sempre é arredondado para cima a cada centena, desta forma, uma invocação que leva 33 milissegundos seria cobrada pelo tempo de 100 milissegundos.**
- **Este modelo sofreu uma alteração em 13/06/2020 de forma que não existe mais esse arredondamento e o tempo de execução é sempre igual ao tempo que será cobrado da Acme.**

Perguntas que serão respondidas com base na análise:

- **A) Calcular o custo total que a Acme teve levando em conta toda a base de dados de execuções**
- **B) Calcular quanto a Acme deixou de gastar com a implementação do novo modelo de precificação**
- **C) Calcular qual seria o custo total se todas as transações estivessem no modelo antigo**
- **D) Calcular qual seria o custo total se todas as transações estivessem no modelo novo**
- **Mostrar um gráfico comparativo com os cenários descritos em B, C e D ao longo do tempo.**

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import datetime
sns.set()
sns.set_theme(style="whitegrid")

df = pd.read_csv('executions.csv',index_col= None)
df_func = pd.read_csv('functions.csv', index_col=0)
tax = 0.000015
```

Agora que os dados estão carregados, mudanças serão feitas:

- **Verificar como os dados estão catalogados.**
- **Criar três novas colunas com base nos resultados esperados**
- **Preencher essas colunas de acordo com as métricas de precificação**

In [2]:

```
df.head()
```

Out[2]:

	id	date	function_id	execution_time
0	6491	2020-05-27	2	373
1	12540	2020-02-08	4	5806
2	17756	2020-04-19	5	154
3	15598	2020-08-26	5	149
4	5916	2020-01-20	1	257

Aqui verifica-se como o Python está tratando esses dados, ou seja, se há algum formato incoerente.

In [3]:

```
df.dtypes
```

Out[3]:

```
id                 int64
date              object
function_id        int64
execution_time     int64
dtype: object
```

Limpeza de dados:

A coluna `id` não está em ordem e a coluna `date` está como um objeto, por isso retira-se a coluna `id`, organizando o dataset pela data, transformando-a em tipo `datetime` e transformando-a no indice do dataset

In [4]:

```
df.sort_values(by = ['date'], inplace= True)
df.index = pd.to_datetime(df['date'])
df.drop(['id', 'date'], axis=1, inplace = True)
df['old_cost'] = 0
df['new_cost'] = 0
df['real_cost'] = 0
df.head()
```

Out[4]:

date	function_id	execution_time	old_cost	new_cost	real_cost
2020-01-01	1	256	0	0	0
2020-01-01	0	59	0	0	0
2020-01-01	1	255	0	0	0
2020-01-01	4	5646	0	0	0
2020-01-01	0	68	0	0	0

Verificando como está o dataset que representa as funções

In [5]:

```
df_func
```

Out[5]:

id	function_name	external_component_avg_latency	has_external_component
2	doThatThing	0	False
1	doThisThing	30	True

	function_name	external_component_avg_latency	has_external_component
3	accessAnotherThing	1000	True
id			
5	fetchThisMetadata	20	True
4	sendAFile	2500	True
0	checkSomething	0	False

Tudo ok!

Como será necessário consultar o tempo de latência de funções que utilizam componentes externos convém criar uma função para retornar esse valor.

Também será criada uma função para renomear a data para o mês daquele registro. Uma vez que iremos analisar os dados por mês e assim podemos agrupá-los somando os seus valores.

In [29]:

```
def get_function_latency(id):
    return df_func.external_component_avg_latency[id]

def date_to_monthName(df):
    for index in df.index:
        if '2020-01-01' <=df.loc[index,'date'] <= '2020-01-31':
            df.loc[index,'date'] = '01-Jan'
        elif '2020-02-01' <=df.loc[index,'date'] <= '2020-02-31':
            df.loc[index,'date'] = '02-Fev'
        elif '2020-03-01' <=df.loc[index,'date'] <= '2020-03-31':
            df.loc[index,'date'] = '03-Mar'
        elif '2020-04-01' <=df.loc[index,'date'] <= '2020-04-31':
            df.loc[index,'date'] = '04-Abr'
        elif '2020-05-01' <=df.loc[index,'date'] <= '2020-05-31':
            df.loc[index,'date'] = '05-Mai'
        elif '2020-06-01' <=df.loc[index,'date'] <= '2020-06-31':
            df.loc[index,'date'] = '06-Jun'
        elif '2020-07-01' <=df.loc[index,'date'] <= '2020-07-31':
            df.loc[index,'date'] = '07-Jul'
        elif '2020-08-01' <=df.loc[index,'date'] <= '2020-08-31':
            df.loc[index,'date'] = '08-Ago'
        elif '2020-09-01' <=df.loc[index,'date'] <= '2020-09-31':
            df.loc[index,'date'] = '09-Set'
        elif '2020-10-01' <=df.loc[index,'date'] <= '2020-10-31':
            df.loc[index,'date'] = '10-Out'
        elif '2020-11-01' <=df.loc[index,'date'] <= '2020-11-31':
            df.loc[index,'date'] = '11-Nov'
        else:
            df.loc[index,'date'] = '12-Dez'
```

Aqui vamos fazer duas coisas:

- **Utilizar a função `get_function_latency` para popular cada linha com a latência referenciada pela coluna `function_id` e subtrair o valor com a coluna `execution_time` fazendo o cálculo da taxa por milissegundo.**

In [30]:

```
df.new_cost = (df.execution_time) * tax
df.old_cost = ((df.execution_time + 100)//100) * 100 * tax
```

In [8]:

```
df.head()
```

Out[8]:

	function_id	execution_time	old_cost	new_cost	real_cost
date					

	function_id	execution_time	old_cost	new_cost	real_cost
2020-01-01	1	256	0.0045	0.003840	0
2020-01-01	0	59	0.0015	0.000885	0
2020-01-01	1	255	0.0045	0.003825	0
2020-01-01	4	5646	0.0855	0.084690	0
2020-01-01	0	68	0.0015	0.001020	0

Como o gasto real leva em consideração os valores do novo modelo(sem arredondamento) a partir de 13 de junho de 2020, foi dividida a tabela em dois grupos, um do inicio do ano até dia 12 de junho e outra a partir do dia 13 de junnho.

In [9]:

```
split_date = np.datetime64('2020-06-13')
df_before = df.copy()
df_before = df_before.loc[df.index < split_date]
df_before.real_cost = df_before.old_cost
df_after = df.copy()
df_after = df_after.loc[df.index >= split_date]
df_after.real_cost = df_after.new_cost
```

In [10]:

```
df_before.tail()
```

Out[10]:

date	function_id	execution_time	old_cost	new_cost	real_cost
2020-06-12	3	1027	0.0165	0.015405	0.0165
2020-06-12	3	1021	0.0165	0.015315	0.0165
2020-06-12	2	273	0.0045	0.004095	0.0045
2020-06-12	5	152	0.0030	0.002280	0.0030
2020-06-12	4	5854	0.0885	0.087810	0.0885

In [11]:

```
df_after.head()
```

Out[11]:

date	function_id	execution_time	old_cost	new_cost	real_cost
2020-06-13	5	147	0.0030	0.002205	0.002205
2020-06-13	0	55	0.0015	0.000825	0.000825
2020-06-13	3	1032	0.0165	0.015480	0.015480
2020-06-13	3	1029	0.0165	0.015435	0.015435
2020-06-13	4	6018	0.0915	0.090270	0.090270

Agora é feita a união desses dois datasets

In [12]:

```
df_total = df_before.append(df_after)
df_total.head()
```

Out[12]:

date	function_id	execution_time	old_cost	new_cost	real_cost
2020-01-01	1	256	0.0045	0.003840	0.0045
2020-01-01	0	59	0.0015	0.000885	0.0015
2020-01-01	1	255	0.0045	0.003825	0.0045
2020-01-01	4	5646	0.0855	0.084690	0.0855
2020-01-01	0	68	0.0015	0.001020	0.0015

Foram agrupados todos os dias iguais e somado seus valores

In [13]:

```
df_days = df_total.groupby('date').sum().reset_index()
df_days.tail()
```

Out[13]:

	date	function_id	execution_time	old_cost	new_cost	real_cost
361	2020-12-27	115	47635	0.7485	0.714525	0.714525
362	2020-12-28	139	77623	1.2090	1.164345	1.164345
363	2020-12-29	144	110948	1.6995	1.664220	1.664220
364	2020-12-30	188	140387	2.1570	2.105805	2.105805
365	2020-12-31	65	33198	0.5235	0.497970	0.497970

Como dito anteriormente os dados foram agrupados por mês. A coluna `date` foi alterada para `string` para ser possível rodar a função `date_to_monthName`

In [14]:

```
df_months = df_days.copy()
df_months = df_months.astype({"date": str})
date_to_monthName(df_months)
```

In [15]:

```
df_months.head()
```

Out[15]:

	date	function_id	execution_time	old_cost	new_cost	real_cost
0	01-Jan	91	34070	0.5445	0.511050	0.5445
1	01-Jan	131	54700	0.8565	0.820500	0.8565
2	01-Jan	141	39131	0.6360	0.586965	0.6360
3	01-Jan	89	38272	0.6060	0.574080	0.6060
4	01-Jan	96	36633	0.5850	0.549495	0.5850

Aqui foi feita a mesma operação de antes, o dataset teve seus valores somados e agrupado pelos meses. Como os dados já estão agrupados as colunas `function_id` e `execution_time` foram retiradas.

In [16]:

```
df_months = df_months.groupby('date').sum()
df_months.sort_values(by = ['date'], inplace= True)
df_months.drop(columns=['function_id','execution_time'], inplace= True)
```

In [17]:

```
df_months.head(12)
```

Out[17]:

date	old_cost	new_cost	real_cost
01-Jan	27.2130	26.046870	27.213000
02-Fev	29.0955	27.967920	29.095500
03-Mar	32.3430	31.158945	32.343000
04-Abr	28.9800	27.855735	28.980000
05-Mai	31.0995	29.910180	31.099500
06-Jun	28.8480	27.700095	28.144575
07-Jul	29.4030	28.238595	28.238595
08-Ago	30.4125	29.222610	29.222610
09-Set	27.7710	26.690565	26.690565
10-Out	29.3670	28.245480	28.245480
11-Nov	31.9005	30.722265	30.722265
12-Dez	31.0890	29.915670	29.915670

Como em uma das perguntas será necessário responder *quanto a Acme deixou de gastar com a implementação do novo modelo de precificação* foi separado um dataset apenas de 13/06/2020 até o fim do ano

In [18]:

```
df_dif = df.copy()

df_dif = df_dif.groupby('date').sum().reset_index()
df_dif = df_dif.loc[df_dif.date >= split_date]
df_dif = df_dif.astype({"date": str})
df_dif.head()
```

Out[18]:

	date	function_id	execution_time	old_cost	new_cost	real_cost
164	2020-06-13	113	63920	0.9945	0.958800	0
165	2020-06-14	169	73565	1.1505	1.103475	0
166	2020-06-15	127	58211	0.9090	0.873165	0
167	2020-06-16	159	88040	1.3635	1.320600	0
168	2020-06-17	77	48989	0.7575	0.734835	0

Foi feita a mesma alteração de data para o nome do mês, agrupados os resultados baseados na data e deixando apenas a coluna `economy` que é a diferença do modelo de precificação novo para o antigo.

In [19]:

```
date_to_monthName(df_dif)

df_dif['economy'] = df_dif.old_cost - df_dif.new_cost
df_dif = df_dif.groupby(df_dif.date).sum().reset_index()
df_dif.index = df_dif.date
df_dif.drop(columns= ['function_id','execution_time','real_cost','old_cost','new_cost','
date'],inplace = True)
```

In [20]:

```
df_dif.head(7)
```

Out[20]:

date	economy
06-Jun	0.703425
07-Jul	1.164405
08-Ago	1.189890
09-Set	1.080435
10-Out	1.121520
11-Nov	1.178235
12-Dez	1.173330

Data Visualization

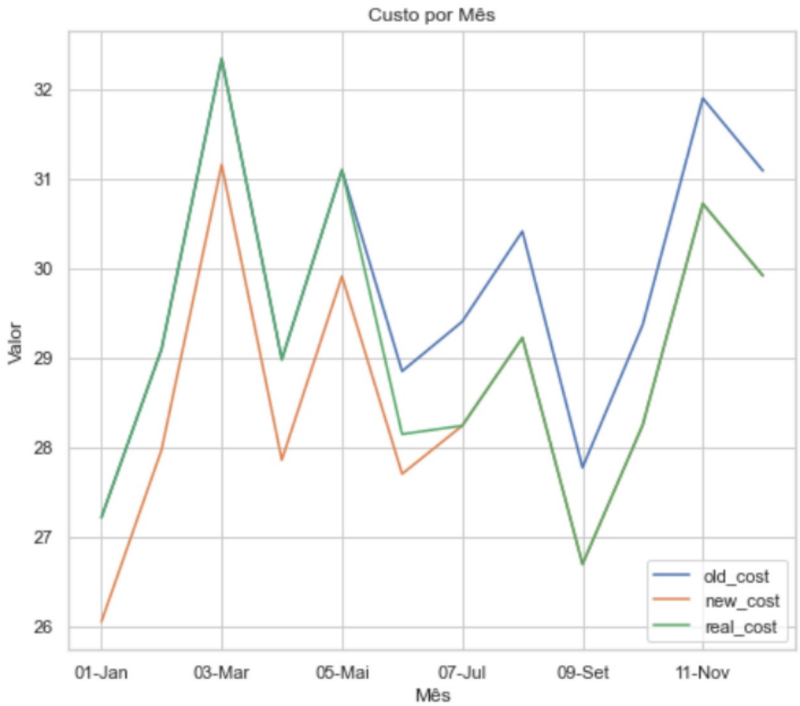
Primeiro é possível ver o valor dos três cenários que foram apresentados ao longo do tempo.

In [23]:

```
df_months.plot(
    y = ['old_cost','new_cost','real_cost'],
    figsize=(8,7),
    title = 'Custo por Mês',
    xlabel = 'Mês',
    ylabel = 'Valor')
```

Out[23]:

```
<AxesSubplot:title={'center':'Custo por Mês'}, xlabel='Mês', ylabel='Valor'>
```

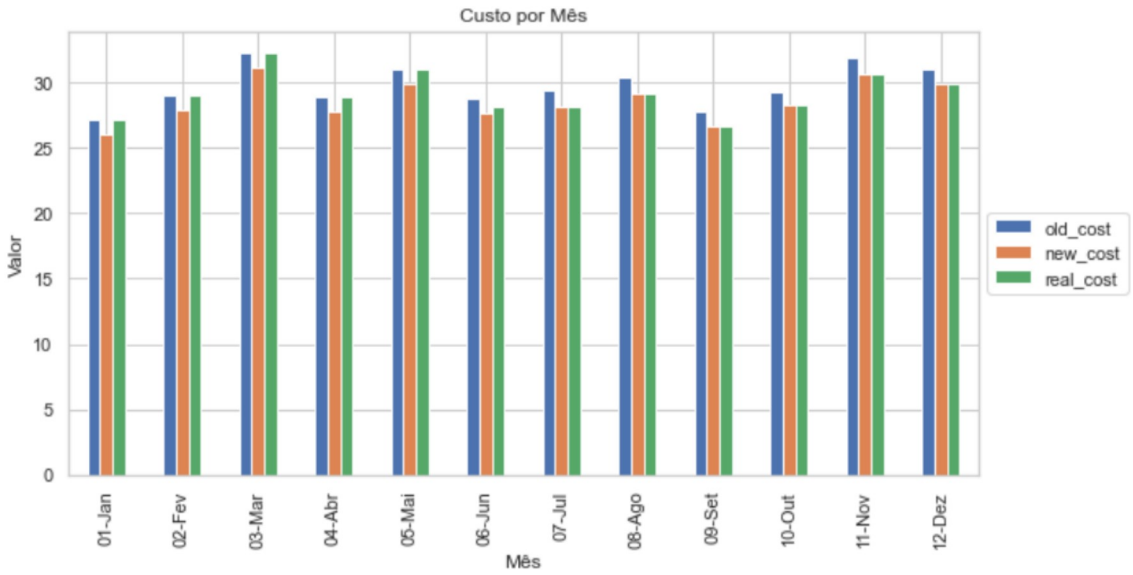


É possível observar no gráfico acima os momentos em que a linha verde(custo real) esta sobreposta a linha azul(custo no modelo antigo) e depois a separação e consequentemente a sobreposição sob a linha laranja(custo no modelo novo)

In [24]:

```
df_months.plot.bar(
```

```
    figsize=(10,5),
    title = 'Custo por Mês',
    xlabel = 'Mês',
    ylabel = 'Valor').legend(loc='center left',bbox_to_anchor=(1.0, 0.5));
```



Versão em barras do mesmo exposto anteriormente, é possível perceber mais sutilmente a diferença entre os modelos de precificação ao longo do tempo.

In [25]:

```
df_months.sum()
```

Out[25]:

```
old_cost     357.52200
new_cost     343.67493
real_cost    349.91076
dtype: float64
```

Em valores reais, ao longo do ano o custo real foi de 349,91 reais. No modelo de precificação antigo, com arredondamento foi de 357,52 reais e no novo modelo de precificação sem arredondamento foi de 343,67 reais

Quanto a Acme deixou de gastar com a implementação do novo modelo de precificação?

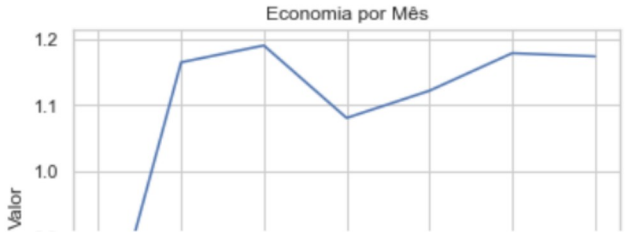
Analisaremos a partir da data da mudança 16/06/2020

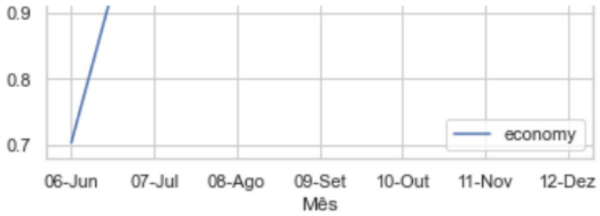
In [27]:

```
df_dif.plot(
    title = 'Economia por Mês',
    xlabel = 'Mês',
    ylabel = 'Valor')
```

Out[27]:

```
<AxesSubplot:title={'center':'Economia por Mês'}, xlabel='Mês', ylabel='Valor'>
```





É perceptível que a economia só se mostrou latente com um mês inteiro de mudanças, tendo aproximadamente um real de economia por mês.

In [28]:

```
df_dif.sum()
```

Out[28]:

```
economy    7.61124
dtype: float64
```

A Acme Inc. deixou de gastar 7,61 reais com a implementação do novo modelo.

Resultados e conclusões:

- **A Acme teve um custo total de 349,91 reais levando em conta toda a base de dados de execuções**
- **A Acme deixou de gastar 7,61 reais com a implementação do novo modelo de precificação**
- **O custo total se todas as transações estivessem no modelo antigo seria de 357,52 reais**
- **O custo total se todas as transações estivessem no modelo novo seria de 343,67 reais**
- **Os números de economia podem ser perceptivelmente pequenos, entretanto, se a Acme tivesse 1000 vezes mais solicitações no ano(um cenário realista ao longo do tempo) a economia seria de quase 7 salários mínimos no ano.**

Melhorias possíveis:

- **Com uma base de dados histórica seria possível aplicar modelos de regressão para prever os custos dos meses seguintes.**