O exercício começa com a preparação do ambiente Colab (research.google.com), montando o drive em que o *dataset* está. Além de carregar as bibliotecas e realizar as atualizações. As células a seguir mostram isso.

1. **Preparação** do Ambiente

```
[5] ### Montar o drive com os arquivos de dados

    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] ### Realizar atualizações e carregamento de libs

    !pip install --upgrade scikit-learn
    !pip install --upgrade pivottablejs

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

Em seguida, montado o drive, carrega-se o dataset para a memória do ambiente de execução. Carregado, avalia-se o estado das variáveis. Conforme trecho de código a seguir, referente a uma célula no notebook.

```
DIR_BASE = '/content/drive/MyDrive/Database/COVID19'

with zipfile.ZipFile(DIR_BASE+'/dataset_02.zip','r') as zip_ref:
  zip_ref.extractall(DIR_BASE+'/')

df_origin = pd.read_csv(DIR_BASE+'/dataset_02.csv', sep=';')
df_origin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4643 entries, 0 to 4642
Data columns (total 58 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   IBGE               4643 non-null   int64
 1   PMATPUB_EF         4643 non-null   float64
```

Investigando o relatório da função “info()” (da lib pandas para python), nota-se a presença de variáveis que são desnecessárias, por exemplo: a que especifica o código do IBGE, a que especifica o nome do município, etc. Tais variáveis, então, são removidas do dataset. Avaliando o significado das variáveis, descobrimos que existem algumas que não dizem respeito aos municípios propriamente dito -- são relativas à unidade da federação em que o município está. Resolvemos excluir também esse conjunto variáveis. Além disso, removemos a variável “Capital” e mantivemos a variável que informa se o município está em tal ou qual região metropolitana (“RegMetro”).

Observa-se também que todas as variáveis, ou são ”float”, ou ”int”, ou ”object”. As numéricas foram ajustadas todas para o tipo”float”. Ficamos, por fim, apenas com “float” e “object” (que são variáveis categóricas). De resto, observou-se que todas as variáveis do *dataset* não possuem valores nulos, o que facilita ajuda por manter a dimensão de suas entradas. O trecho de código a seguir identifica a célula em que essa atividade de ajuste é realizada.

```
df = df_origin.drop(columns=['IBGE', 'municipio', 'Capital', 'ESPVIDA', 'MORT1', 'T_ENV', 'RAZDEP', 'ANOSEST',
       'T_ANALF15M', 'T_ATRASO_2_BASICO', 'T_ATRASO_2_FUND', 'T_FUND15A17',
       'R1040', 'RDPC', 'RDPC1', 'RDPC4', 'RMPOB', 'RPOB', 'PMPOB', 'PPOB', 'estado',
       'GINI', 'IDHM_E', 'IDHM_L', 'IDHM_R', 'IDHM'], errors=True) #, 'ANNUAL', 'RegMetro', 'Porte'
df.drop_duplicates(inplace=True)
df = df.astype({
  'POP_TOT': float
})
df.columns
```

Em seguida, concentramo-nos em criar a variável que indica o grau de incidência da Covid19 para cada um dos municípios. Assim é feito por meio da relação entre o total de contaminados do município e o total da população do município. A essa nova variável no dataset demos o nome de “*incidencia*”. Ela nos ajudará a categorizar os municípios conforme o grau de incidência da doença sobre a população do município, por meio da avaliação do histograma da variável.

O critério que empregamos para a categorização da incidência, criando portanto a variável alvo, é o da divisão do espaço de valores da variável ”incidencia” em ”quartiles” (conforme anteriormente discutido). A seguir é apresentada, nos trechos de código recordados das células presentes no notebook, a criação da variável alvo, aqui denominada “Y”.

```
df['incidencia'].describe()
```

```
count    4643.000000
mean        0.021440
std         0.017617
min         0.000106
25%         0.009433
50%         0.017067
75%         0.028174
max         0.207346
Name: incidencia, dtype: float64
```

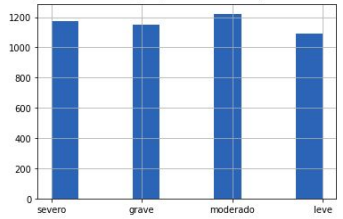
Esse trenho de código imediatamente acima revela os “quartiles” do espaço de valores da variável “incidencia”. Ele que eh utilizado como referênci para o trecho imediatamente a seguir. que mostra o histograma da variável alvo, “Y”.

```
if ('Y' in df.columns):
  df.drop(columns=['Y'], inplace=True)
df = df.assign(Y="")

df.loc[df['incidencia'] <= 0.009, ('Y')] = 'leve'
df.loc[(df['incidencia'] > 0.009) & (df['incidencia'] <= 0.017), ('Y')] = 'moderado'
df.loc[(df['incidencia'] > 0.017) & (df['incidencia'] <= 0.028), ('Y')] = 'grave'
df.loc[df['incidencia'] > 0.028, ('Y')] = 'severo'

df['Y'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa3f93afba8>
```



Aqui é possível observar o estado de balanceamento das entradas no dataset, de modo que nos vimos dispensados de oferecer algum método para balanceamento, com vistas ao treinamento dos classificadores.

A seguir o trecho de código que mostra a remoção das variáveis por meio das quais foram construídas as categorias, além daqui dispensou nesse instante interesse “obitosAcumulado”.

```
df.drop(columns=['casosAcumulado', 'obitosAcumulado', 'POP_TOT'], inplace=True)
```

Em seguida construímos uma classe com o fim de ter a opção de operar dados numéricos normalizados, ou não. Isso está posto na classe “NumericalTransformation”, ao empregarmos os serviços “StandardScaler” e “MinMaxScaler” da biblioteca “sklearn”. A classe pode ser vista no trecho de código a seguir, o uso dos serviços de normalização estão postos no interior da classe, no interior de um de seus métodos, o “transform”.

```
# transform numerical features
class NumericalTransformer( BaseEstimator, TransformerMixin ):
  # Class constructor method that takes a model parameter as its argument
  # model 0: minmax
  # model 1: standard
  # model 2: without scaler
```

E assim, com o feito até aqui, temos todo o conjunto de tarefas necessárias que encerram a preparação do dataset para processamento junto aos classificadores. A seguir, realizamos a divisão da base em conjunto de dados para treino dos tais, e conjunto de dados para testar o treino dos classificadores. Fazemos isso com o comando a seguir:

```
[121] ### split-out train/validation and test dataset

      X_train, X_test, y_train, y_test = train_test_split(df_w.drop(labels="Y",axis=1),
                                                          df_w["Y"],
                                                          test_size=0.20,
                                                          random_state=41,
                                                          shuffle=True,
                                                          stratify=df_w["Y"])
```

Com o fim de facilitar a operação de investigação de qual classificador e seu conjunto de hiperparâmetros melhor adequa-se à questão que investigamos, utilizamos um pipeline. Conforme vê-se no trecho a seguir
:

```
[123] ### Preparação do Pipeline

      # Categrical features to pass down the categorical pipeline
      categorical_features = X_train.select_dtypes("object").columns.to_list()

      # Numerical features to pass down the numerical pipeline
      numerical_features = X_train.select_dtypes("float64").columns.to_list()

      # Defining the steps in the categorical pipeline
      categorical_pipeline = Pipeline(steps = [('cat_selector', FeatureSelector(categorical_features)),
                                               ('cat_transformer', CategoricalTransformer()),
                                               ('cat_encoder','passthrough')
                                               #('cat_encoder',OneHotEncoder(sparse=False,drop="first"))
                                               ]
                                       )

      # Defining the steps in the numerical pipeline
      numerical_pipeline = Pipeline(steps = [('num_selector', FeatureSelector(numerical_features)),
                                             ('num_transformer', NumericalTransformer())
                                             ]
                                    )

      # Combining numerical and categorical piepline into one full big pipeline horizontally
      # using FeatureUnion
      full_pipeline_preprocessing = FeatureUnion(transformer_list = [('cat_pipeline', categorical_pipeline),
                                                                     ('num_pipeline', numerical_pipeline)
                                                                     ]
                                                 )
```

Na preparação do pipeline, separamos os dados conforme seu formato para sofrerem ainda modificações separadas -- conforme sinta-se necessidade -- de acordo com as classes de transformação de variáveis categóricas e numéricas que podem ser visitadas no notebook que conforma todos esses códigos. Destacamos, antes de seguir em frente, que como as variáveis categóricas não precisaram ser ajustadas, a classe de transformação de dados categóricos ficou subutilizada. Por fim, reúnem-se os dados transformados em

“full_pipeline_preprocessing” para que então seja entregue o pipeline. Isso é visto no trecho de código a seguir:

```
### Configuração e execução do Pipeline

seed = 15
num_folds = 10
# scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score)}
scoring = {'Accuracy': make_scorer(accuracy_score)}

# The full pipeline
pipe = Pipeline(steps = [('full_pipeline', full_pipeline_preprocessing),
                         ("fs", SelectKBest()),
                         ("classifier", DecisionTreeClassifier())])

# create a dictionary with the hyperparameters
search_space = [
                {"classifier": [DecisionTreeClassifier()],
                 "classifier__criterion": ["gini", "entropy"],
                 "classifier__splitter": ["best", "random"],
                 "fs__k": [11, 13, 17, 19, 23],
                 "fs__score_func": [mutual_info_classif],  #[f_classif, mutual_info_classif, chi2],
                 "full_pipeline__cat_pipeline__cat_encoder": [OneHotEncoder(sparse=False,drop="first")], #[OneHotEncoder(sparse=False,drop="first")
                 "full_pipeline__cat_pipeline__cat_transformer__new_features": [False],
                 "full_pipeline__num_pipeline__num_transformer__model": [1,0]},
                 {"classifier": [RandomForestClassifier()],
                 "classifier__criterion": ["gini","entropy"],
                 "classifier__n_estimators": [100, 200, 300, 500],
                 "fs__k":[11, 13, 17, 19, 23],
                 "fs__score_func": [chi2],
                 "full_pipeline__cat_pipeline__cat_encoder":[OneHotEncoder(sparse=False,drop="first")]}
                ]
```

```
# create grid search
grid = GridSearchCV(estimator=pipe,
                    param_grid=search_space,
                    cv=StratifiedKFold(n_splits=num_folds,random_state=seed,shuffle=True),
                    scoring=scoring,
                    return_train_score=True,
                    n_jobs=-1,
                    refit="Accuracy",
                    verbose = 1)

### fit grid search
all_models = grid.fit(X_train, y_train)
```

O trecho de código acima apresenta o pipeline, o espaço de busca definido para identificar qual classificador melhor dá resposta aos dados processados, e seus hiperparâmetros. Tal espaço de busca é passado como parâmetro para uma instância do”GridSearchCV”. Destacamos para essa instância de serviço a instância da estratégia de validação cruzada dos dados de treinamento, “StratifiedKFold”. Por ela a base é dividida em 10 partições de treinamento e, uma a uma, é submetida a treinamento por parte do classificador. Além disso a instância de “StratifiedKFold”, preocupa-se em garantir balanceamento entre as entradas conforme suas categorias. Essa foi a razão mais relevante de sua escolha, além, é claro, de automatizar o processo do particionamento para a validação cruzada da base.

Destacamos ainda que os classificadores utilizados, conforme o trecho apresentado acima, são o “DecisionTreeClassifier” e o “RandomForestClassifier”. Quanto aos hiperparâmetros, destacamos que para o DecisionTreeClassifier” -- dos parâmetros “fs__k”, “classifier__criterion” e “classifier__splitter” (que implicam a lida com a seleção da melhores features para o treinamento) -- o único que implicou escolha aleatória nossa foi o “fs__k”. Escolhemos, por uma questão de custo, oferecer para o espaço de busca um máximo de cinco números primos que designam a quantidade das variáveis que podem ser escolhidas para treinamento pelo classificador, com o fim de desenhar a árvore de decisão. Ademais, em testes, a função de score pra ambos os classificadores que melhor apresentou resultados foi “mutual_info_classif” -- testamos também o chi2.

O resultado do treinamento revelou a seguinte acurácia:

```
print("Melhor resultado: %f usando %s" % (all_models.best_score_,all_models.best_params_))
```

```
Melhor resultado: 0.972666 usando {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__splitter': 'best', 'fs__k': 11, 'fs__score_fu
```

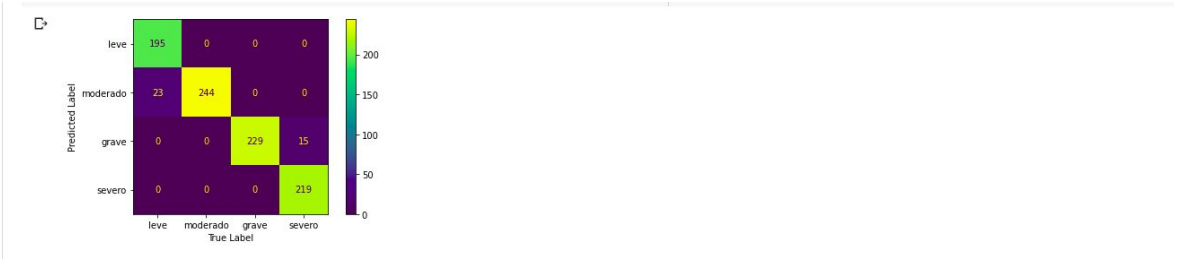
E informou que tal acurácia veio por meio do uso do classificador “DecisionTreeClassifier” (mais detalhes sobre quais hiperparâmetros foram utilizados para chegar-se a esse resultado, ver o notebook)
Quanto à acurácia do modelo agora aplicado ao Y de teste foi:

```
print(accuracy_score(y_test, predict))
```

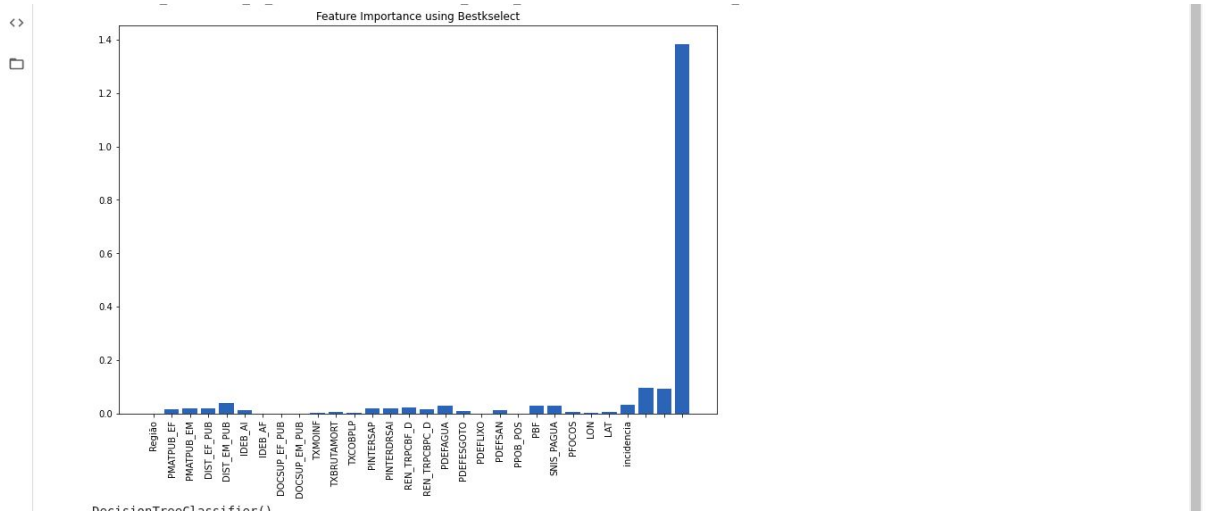
```
0.9589189189189189
```

O que resulta em valor bem próximo ao Y de treino, portanto, para nossa finalidade, considerado adequado o resultado.

Para visualizarmos melhor o resultado do ponto de vista do desafio da classificação dos municípios, plotamos a matriz de confusão para compreender melhor essaa dificuldade. Ela nos mostra que houve alguma dificuldade para rotular os municípios quando da tentativa de categorizá-los como “leve” ou “moderado”, além das categorias “severo” ou “grave”.

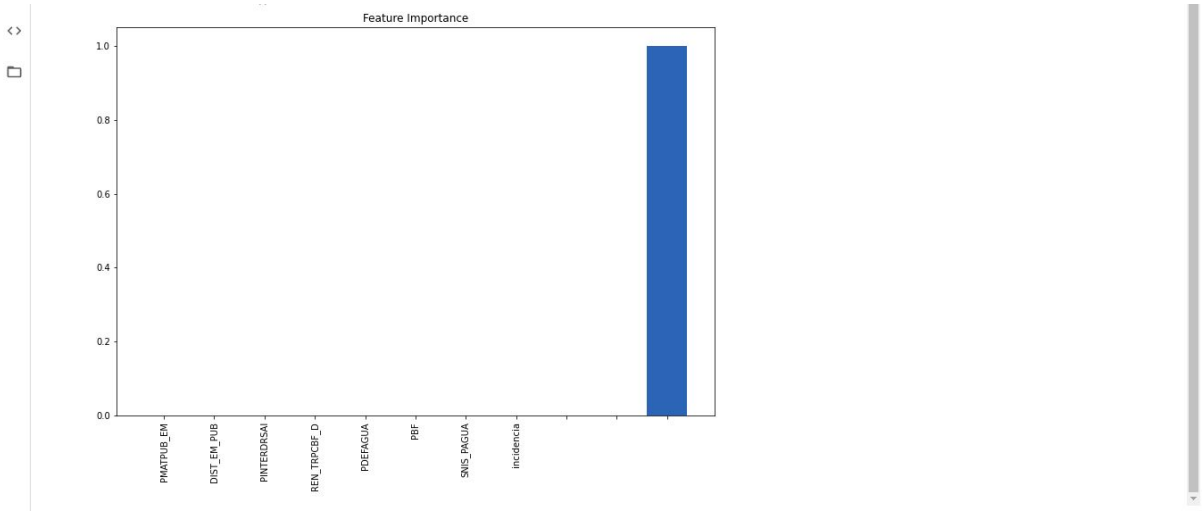


Além dessa imagem, mostramos também como as variáveis do espaço de dados se comportam em relação à categorização dos municípios, para o melhor classificador encontrado. A figura a seguir mostra que as variáveis de natureza socioeconômica pouco influenciam a categorização.



A última imagem destaca as 11 variáveis (este é o número retornado para o hiperparâmetro 'fs__k' que indica quantas são as melhores variáveis selecionadas pelo modelo)

empregadas para a categorização dos municípios conforme a variável alvo, “Y”. Mais uma vez, percebe-se que as variáveis de ordem socioeconômica não são decisivas para determinar a categorização dos municípios conforme os critérios usados no trabalho.



Ressaltamos, contudo, que este é um trabalho cujo principal intuito está no aprendizado de técnicas de Aprendizado de Máquina, e não em responder de forma científica à predizer a categoria dos municípios, conforme a variável alvo por nós aqui, arbitrariamente elencada.