**FACULDADE FIA DE ADMINISTRAÇÃO E NEGÓCIOS**

**Fundação Instituto de Administração**

Rodrigo Quinelato Cosin

# GERAÇÃO DE CÓDIGO FONTE PARA AUTOMATIZAR OS PROCESSOS ENVOLVIDOS NOS ALGORITMOS DE APRENDIZAGEM SUPERVISIONADA

São Paulo

2019

Rodrigo Quinelato Cosin

# GERAÇÃO DE CÓDIGO FONTE PARA AUTOMATIZAR OS PROCESSOS ENVOLVIDOS NOS ALGORITMOS DE APRENDIZAGEM SUPERVISIONADA

Monografia apresentada à Faculdade FIA de Administração e Negócios mantida pela Fundação Instituto de Administração como requisito para obtenção do certificado de conclusão do curso de Pós-Graduação "Lato Sensu" Especialização em Análise de Big Data.

Orientadores: Profª. Drª. Alessandra de Ávila Montini, Prof. Dr. Adolpho Walter Pimazoni Canton

São Paulo

2019

FOLHA DE APROVAÇÃO

Rodrigo Quinelato Cosin

**GERAÇÃO DE CÓDIGO FONTE PARA AUTOMATIZAR OS PROCESSOS ENVOLVIDOS NOS ALGORITMOS DE APRENDIZAGEM SUPERVISIONADA**

___/___/___

Banca examinadora:

_____

Profº. Dr. Orientador

_____

Profº. Dr.

_____

Profº. Dr.

Julgamento: _____ Assinatura: _____

## RESUMO

Estudos mostram o contínuo crescimento da demanda por profissionais das áreas de *Big Data*, dentre estes profissionais existe o papel do Cientista de Dados, que tem como tarefa trazer valor aos dados através da análise dos dados e construção de modelos que possibilitem responder questões importantes ou predizer situações de interesse (VALLEY, 2019). O perfil do cientista de dados é um dos mais valorizados no mercado, atingindo uma média salarial de U$ 108.000,00 anuais, pois se trata de um profissional com conhecimentos de programação, estatística e negócios (VALLEY, 2019). Por este motivo, a otimização do tempo deste profissional é algo desejado, este estudo propõe estudar a automatização de parte do trabalho realizado pelo cientista de dados, alguns passos realizados por este profissional são, de certa forma, repetitivos e procedurais. Desta forma é possível automatizar os passos repetitivos e empíricos do cientista de dados, como por exemplo alguns passos para auxiliar na seleção e tratamento de variáveis, simulação e ajuste de modelos e geração de código python base, possibilitando desta forma que o programa não precise ser criado desde o início, atingindo assim o objetivo de otimizar o tempo deste profissional.



**Palavras-chave:** *Machine Learning*. Automatização. Ciência de dados. Cientista de dados. Python. Estatística. *Big Data*.

## ABSTRACT

Studies show the continuous growth of demand by professionals in the Big Data area, among these professionals there is the role of Data Scientist, whose task is to bring value to data through data analysis and construction of models that allow answering important questions or predicting situations of interest (VALLEY, 2019). The data scientist's profile is one of the most valued in the market, reaching a salary average of $ 108,000 per year, as he is a professional with programming, statistical and business knowledge (VALLEY, 2019). For this reason, the optimization of this professional's time is something desired, this study proposes to study the automation of part of the work done by the data scientist, some steps performed by this professional are, in a way, repetitive and procedural. In this way it is possible to automate the repetitive and empirical steps of the data scientist, as for example some steps to assist in the selection and treatment of variables, simulation and adjustment of models and generation of base python code, thus allowing the program not to be created from the beginning, thus achieving the goal of optimizing the time of this professional.



**Keywords:** Machine learning. Automation. Data science. Data scientist. Python. Statistic. Big data.

**SUMÁRIO**

**1 INTRODUÇÃO ..... 8**
1.1 Objetivos ..... 9
1.2 Organicação do trabalho ..... 10
**2 REFERENCIAL TEÓRICO ..... 11**
**3 IMPLEMENTAÇÃO DA SOLUÇÃO ..... 23**
3.1 Pré requisitos e parametrizações ..... 23
3.2 Importação do arquivo de configurações ..... 25
3.3 Funções de geração de código ..... 25
3.4 Função para ajuste de modelos de machine learning ..... 26
3.5 Importação de módulos e pacotes ..... 27
3.6 Leitura do dataset ..... 27
3.7 Seleção de variáveis e identificação de tratamento ..... 27
3.8 Tratamento de dados ..... 30
3.9 Separação das bases de treino e teste ..... 32
3.10 Aplicando e ajustando os modelos de machine learning ..... 32
3.11 Geração de código python final ..... 33
**4 ANÁLISE DOS RESULTADOS ..... 34**
**5 CONCLUSÃO ..... 44**
**REFERÊNCIAS ..... 46**
**APÊNDICE A - Arquivo de configurações para processamento da base titanic ..... 49**
**APÊNDICE B - Solução de automatização de ml ..... 50**
**APÊNDICE C - Código final gerado pela solução após processamento da base titanic .. 57**
**APÊNDICE D - Arquivo de configurações para processamento da base investimento .. 75**
**APÊNDICE E - Código final gerado pela solução após processamento da base investimento ..... 76**
**APÊNDICE F - Arquivo de configurações para processamento da base compras online 93**
**APÊNDICE G - Código final gerado pela solução após processamento da base compras online ..... 94**

## LISTA DE ILUSTRAÇÕES

Figura 1: Processos para tratamento de dados. ..... 11
Figura 2: Área sob a curva ROC ..... 22
Figura 3: Fluxo realizado pela solução de automatização de machine learning ..... 23
Figura 4: Código final Python - Importação de módulos e dados ..... 36
Figura 5: Código final Python - Análise de variáveis ..... 37
Figura 6: Código final Python - Correlação entre variáveis ..... 37
Figura 7: Código final Python - Preparação de dados ..... 38
Figura 8: Código final Python - Separação de bases teste e treino ..... 39
Figura 9: Código final Python - Modelo: DecisionTreeClassifier ..... 40
Figura 10: Código final Python - Comparativo de modelos ..... 41

## LISTA DE TABELAS

Tabela 1: Matriz de confusão ..... 19
Tabela 2: Derivação da matriz de confusão ..... 19
Tabela 3: Métricas utilizadas para seleção e tratamento de variáveis ..... 28
Tabela 4: Regras para enquadramento de variáveis explicativas ..... 29
Tabela 5: Regras para tratamento de variáveis ..... 31
Tabela 6: Resultados de bases processadas ..... 42

# 1 INTRODUÇÃO

Atualmente as empresas estão observando nos dados algo fundamental para o seu crescimento, o termo “Empresas *Data Driven*” se refere a esta crescente importância que as empresas percebem em seus dados, que afeta diretamente nas decisões de negócios e direcionamento de suas atividades (ROLLINGS, 2019).

Por este crescente interesse no consumo de dados pelas empresas, surgiu a necessidade de adoção de soluções *Big Data*, descritas segundo a SAS como “termo que descreve o imenso volume de dados – estruturados e não estruturados – que impactam os negócios no dia a dia. Mas o importante não é a quantidade de dados. E sim o que as empresas fazem com os dados que realmente importam. Big Data pode ser analisado para a obtenção de insights que levam a melhores decisões e direções estratégicas de negócio.” (SAS, 2019).

Porém o mercado profissional não tem capacidade de suprir a demanda por mão de obra especializada para trabalhar com dados (Arquiteto de dados, Engenheiro de dados e cientista de dados), desafio este enfrentado até por grandes empresas como *Facebook* e *Google*, o reflexo de tal escassez de mão de obra se reflete nos elevados salários de alguns profissionais desta área (FILHO, 2019).

Uma solução completa de Big Data é composta por diversas camadas, é necessário ter o papel de arquiteto responsável por definir uma arquitetura para suportar a solução, assim como o papel de infraestrutura para implementar um *framework* robusto e seguro para processamento e armazenamento de dados distribuídos, não menos importante, se faz necessário o papel do engenheiro de dados, responsável pela carga e preparação dos dados (FUJIMAKI, 2019).

Existe ainda o papel do cientista de dados, que se trata de um perfil diferenciado no mercado, pois necessita ter conhecimentos avançados de estatística, habilidades de programação e conhecimento de negócio. Este profissional utiliza estes conhecimentos para aplicar modelos estatísticos e *machine learning* para realização de análises descritivas (identificar algum comportamento que tenha ou esteja ocorrendo), análises diagnósticas (identificar o motivo de algum evento ter ou esteja ocorrendo), análise preditiva (identificar possíveis ocorrências que possam vir a ocorrer) e análise prescritiva (identificar e automatizar uma ação para prevenir uma ocorrência indesejada ou situação que possa vir a ocorrer) (MAYDON, 2017).

Os conhecimentos necessários exigidos de um profissional com perfil de cientista de dados são mistos, sendo que este profissional necessita ter aptidões em estatística,

programação e conhecimento de negócio (VALLEY, 2019). Este profissional necessita ter tais aptidões pois irá desempenhar vários tipos de atividades altamente complexas, tais como, coleta de dados, organização dos dados, engenharia de variáveis (*feature engineering*), aprendizado de máquina (*machine learning*) e construção de painéis e gráficos para apresentar as conclusões observadas após o trabalho realizado (FUJIMAKI, 2019).

Dado o cenário apresentado acima, onde existe uma grande demanda por profissionais com o perfil de cientista de dados, que por sua vez, são profissionais altamente especializados, com múltiplos conhecimentos e em escassez no mercado de trabalho, algumas empresas tem como objetivo automatizar parte do trabalho realizado por estes profissional, algumas destas empresas são: *DataRobot*, que provem soluções para automatização e aceleração destes trabalhos (*webpage*: https://www.datarobot.com/) e H2O.ai, uma *startup* que visa disponibilizar uma plataforma para agilizar e automatizar parte dos trabalhos realizados pelo perfil de cientista de dados (*webpage*: https://www.h2o.ai/).

## 1.1 Objetivos

Este trabalho tem como objetivo propor uma solução técnica para automatizar parte do trabalho realizado pelo cientista de dados, analisar os resultados gerados de forma automatizada pela solução. Conforme apresentado no capítulo introdutório, existem algumas empresas focadas em trazer soluções de automatização de processos de ciência de dados, este trabalho tem um objetivo similar no sentido de automatização, porém visa não ser uma solução fechada, onde não se tem acesso ao algoritmo responsável pela automatização.

Como resultado da solução de automatização, será gerado um código fonte com todos os passos utilizados para processamento da base e aplicação dos modelos de *machine learning*. As etapas que serão automatizadas neste trabalho serão os procedimentos repetitivos e que não dependam de conhecimento de negócio.

A aplicação de *machine learning* apresenta uma abrangência muito grande, sendo possível aplicar diferentes tipos de algoritmos para responder diversos tipos de problemas, o presente trabalho visa tratar apenas de problemas de classificação, por se tratarem de problemas bem definidos, facilitando a criação de um algoritmo base para posteriormente ser adaptado para os demais tipos de análises.

## 1.2 Organicação do trabalho

Nos próximos capítulos será apresentado um estudo sobre a aplicabilidade de uma solução para automatização de parte do trabalho realizado pelo cientista de dados, viabilizando ganho de agilidade para o trabalho deste profissional tão requisitado e valorizado.

Como resultado da solução de automatização proposta por este estudo, será gerado um código fonte com todos os passos realizados pelo mecanismo automatizado, sendo o ponto inicial para as demais análises a serem realizadas pelo cientista de dados.

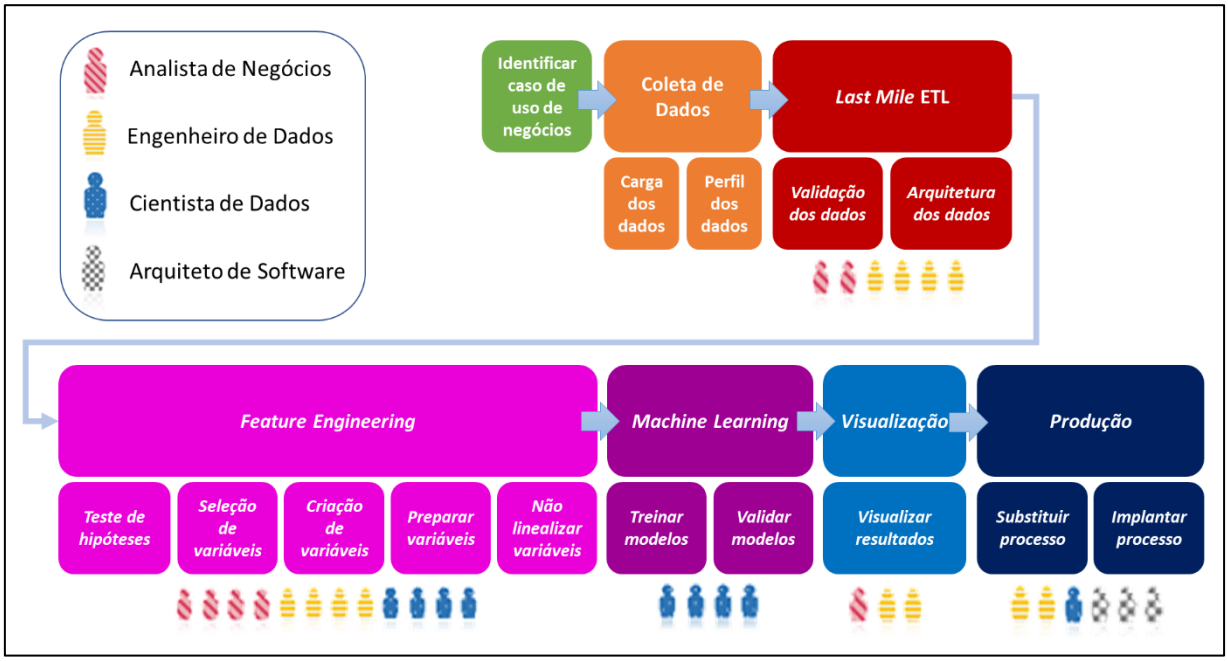
Por fim será realizada a análise dos resultados deste trabalho, identificando a aplicabilidade, pontos positivos e limitações da solução proposta.

# 2 REFERENCIAL TEÓRICO

Existe uma sequência de etapas responsável pela coleta, carga e processamento dos dados, transformando-os em valor para empresa. Por este motivo, pode ser necessário muitos meses para a implementação de um projeto tradicional de ciência de dados.

Esta sequência de etapas é composta por diversos profissionais, responsável por preparar o dado para a próxima camada, conforme demonstrado pelo figura 1 (FUJIMAKI, 2019).

Figura 1: Processos para tratamento de dados.




Fonte: Adaptado de FUJIMAKI, 2019

Como pode ser observado na figura 1, o trabalho realizado pelo time de ciência de dados não se limita apenas a aplicação de modelos estatísticos, de forma simplificada o cientista de dados é o profissional responsável por extrair conhecimento e gerar valor de dados desorganizados, sendo necessário desempenhar papeis conforme apresentados no diagrama acima (GRUS, 2016).

Um exemplo de desafio encontrado pelo cientista de dados é a preparação do dado no formato adequado para ser trabalhado, as empresas normalmente armazenam os dados em bancos de dados relacionais e com estruturas normalizadas, ou seja, os dados estão distribuídos em diversas entidades contidas em uma ou mais bases de dados, porém o cientista de dados necessita dos dados de forma desnormalizada, ou seja, todos os dados em uma

estrutura única que comporte todos os dados necessário em apenas uma entidade, por exemplo, converter os dados contidos em 10 entidades que se relacionam no banco de dados relacional em apenas um arquivo no formato CSV (FUJIMAKI, 2019).

Etapas relativas a aquisição de dados, entendimento dos dados e origens das informações, não serão abordadas neste trabalho pois se trata de etapas invariavelmente tecnológicas, dependendo de cada ambiente e com grande dependência de regra de negócio para o mapeamento correto dos dados.

Conforme apresentado na figura 1, o cientista de dados participa de algumas fases do processo total de ciência de dados, o primeiro deles é chamado de *Feature Engineering*, etapa ontem são realizados testes de hipóteses, onde um conhecimento prévio sobre o negócio gera uma hipótese sobre um determinado comportamento e cabe ao cientista de dados comprovar ou refutar tal hipótese; outra etapa se refere a seleção de variáveis, onde nem todas as variáveis da base de dados irá ajudar nos modelos, desta forma, devem ser desprezadas; ainda existe a etapa de criação de variáveis, onde são criadas novas variáveis, seja aplicando regras ou combinando variáveis já existentes; existe ainda a necessidade de preparação das variáveis, etapa responsável por tratar dados faltantes e padronização dos dados; por último os dados podem precisar ser transformados de forma que os modelos que serão aplicados consigam executar, como transformar uma variável categórica, por exemplo variável sexo, em outras variáveis *dummies*, sexo_m e sexo_f, que receberão 1 ou 0, para indicar condições de verdadeiro ou falso respectivamente.

De todas as etapas contidas na fase de *Feature Engineering*, serão tratadas neste trabalho apenas as etapas de seleção de variáveis, preparar variáveis e transformação de variáveis, descritas no parágrafo anterior. Para as etapas de teste de hipóteses e criação de variáveis existe uma dependência de conhecimento das regras de negócio e por este motivo não serão tratadas neste trabalho.

A próxima fase em que existe a participação do cientista de dados é a fase de *machine learning*, esta fase é composta primeiramente pela etapa de treinar modelos, responsável por ajustar os dados aos modelos estatísticos que se tem interesse em estudar, assim como fazer a separação da base de dados entre bases de teste e treino, gerando insumos para a fase de validação de modelos, onde pode ser ajustamos os melhores x para cada modelo e calculado o score atingido pelos modelos, possibilitando assim a escolha pelo modelo que mais explicou os dados. Ambas as etapas da fase de *machine learning* serão tratadas pelo processo de automatização proposto por este trabalho.

A última fase apresentada na figura 1 se refere a implantar e submeter o processo de *machine learning* em produção, estas etapas são pertinentes e diferentes para cada empresa, por este trabalho ter um objetivo de estudo acadêmico e não haver necessidade de se criar um ambiente produtivo, estas etapas não serão tratadas pelo processo de automatização proposto por este trabalho.

Existe uma grande abrangência no que se refere a aplicação de soluções de *machine learning*, será abordado por este estudo métodos de aprendizagem supervisionados, visando resolver problemas de classificação.

Como parte das soluções de Big Data, temos as *engines* de processamento distribuídos, uma das soluções utilizadas para este fim é o Apache Spark, possibilitando o desenvolvimento de programas em linguagens como Scala, Java e Python.

Visando o processamento de grandes volumes de dados e de forma distribuída, a *engine* Apache Spark traz muitos benefícios, possibilitando programar códigos em linguagem Python (pySpark), utilizando *data frames* e aplicação de diversos modelos de *machine learning* (EDUCBA, 2019).

Para o presente trabalho, não será utilizada uma *engine* de processamento distribuído, pois para atender o objetivo de estudar a viabilidade da automatização de alguns passos do cientista de dados, não se faz necessário o processamento de grandes volumes de dados e performance de execução. Para tanto, será utilizada a linguagem python e processamento local.

A etapa de testes de diferentes algoritmos de *machine learning* e seus respectivos ajustes de hiper-parâmetros é muito importante, porém tem uma característica empírica, onde o cientista de dados realiza testes com diversas combinações procurando atingir o menor erro. Devido à natureza procedural deste tipo de análise, é possível validar a automatização de alguns modelos de *machine learning* e seus respectivos hiper-parâmetros (KUHN; JOHNSON, 2013).

Foram selecionados alguns modelos de *machine learning* para aplicação neste trabalho, a seguir será apresentado cada um deles, abordando a sua definição e seus respectivos hiper-parâmetros, que podem ser ajustados com objetivo de melhorar o modelo preditivo.

*DecisionTree*: Os modelos de árvore de decisão são algoritmos que dividem a base de dados em grupos menores, de forma que cada um destes grupos após a divisão seja mais homogêneo em relação a resposta, por exemplo, se inicialmente a base de dados (S0) contém 60% dos

registros com a variável resposta igual a ‘SIM’ e 40% dos registros com a variável resposta igual a ‘NÃO’, após a primeira divisão se espera que as duas bases sejam mais homogêneas, por exemplo, base derivada 1 (S1) contém 80% % dos registros com a variável resposta igual a ‘SIM’ e base derivada 2 (S2) contém 90% % dos registros com a variável resposta igual a ‘NÃO’. Seguindo a mesma estratégia, as bases S1 e S2 também podem sofrer divisões até se atingir um grau de erro desejado (MITCHELL, 1997).

As árvores de decisão calculam quais variáveis e valores serão utilizados em cada divisão e os hiper-parâmetros ajustam a profundidade da árvore entre outros aspectos, abaixo podem ser observados alguns hiper-parâmetros:

Hiper-parâmetros de modelos *DecisionTree* (BEN FRAJ, 2017):

- ***max_depth:*** Profundidade da árvore. Quanto mais profunda a árvore for, mais ela se divide e captura mais informações sobre os dados.
- ***min_samples_split:*** Número mínimo de amostras necessárias para dividir um nó interno. Isso pode variar entre uma amostra em cada nó e todas as amostras em cada nó.
- ***min_samples_leaf:*** Número mínimo de amostras necessárias para estar em um nó folha.
- ***max_features:*** Número de atributos/colunas a serem considerados ao procurar a melhor divisão.

*RandomForest*: O método de *random forest*, que traduzindo literalmente significa floresta aleatória, recebe este nome por utilizar o método de árvore de decisões para assim criar diversas árvores, porém árvores diferentes entre si, variando as variáveis que são utilizadas por cada uma delas ou substituindo registros originais por registros duplicados. Cada uma das árvores geradas pelo modelo *Random Forest* terá uma lógica específica para realizar a classificação, desta forma ao processar uma base de dados, internamente serão geradas várias árvores de decisão, supondo que para um registro específico 70% delas indique que a classificação deve ser “SIM”, é este o resultado final do modelo *Random Forest*. Esta estratégia minimiza o erro e a variância observados pelo modelo de árvore de decisão (KUHN; JOHNSON, 2013). A seguir podem ser observados alguns hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos *RandomForest* (KOEHRSEN, 2018):

- ***n_estimators:*** Número de árvores utilizadas (na “floresta”).
- ***max_depth:*** Número máximo de níveis para cada árvore de decisão (*decision tree*).
- ***min_samples_split:*** Número mínimo de amostras necessárias para dividir um nó interno. Isso pode variar entre uma amostra em cada nó e todas as amostras em cada nó.
- ***min_samples_leaf:*** Número mínimo de amostras necessárias para estar em um nó folha.
- ***bootstrap:*** Método para amostragem de pontos de dados (com ou sem substituição).
- ***max_features:*** Número de atributos/colunas a serem considerados ao procurar a melhor divisão.

SVC (*Support Vectors Classifier*): O modelo de classificação SVC utiliza algoritmos do tipo SVM*(Support Vector Machines),* que tenta encontrar uma linha que separe os dados de acordo com a variável resposta, esta linha recebe o nome de hiperplano, o objetivo é identificar a posição da linha que maximize a distância entre os registros com variável resposta diferentes, ao encontrar a posição do hiperplano que melhor separa os grupos dado o valor de suas variáveis respostas é calculado a margem, que é a distância entre o hiperplano e os indivíduos mais próximos de cada um dos grupos dado os valores de suas variáveis resposta. O objetivo é conseguir classificar os registros utilizando a separação que reduza mais o erro de classificação (KUHN; JOHNSON, 2013). A seguir podem ser observados alguns hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos SVC (BEN FRAJ, 2018):

- ***gamma:*** Parâmetro para hiperplanos não lineares. Quanto mais alto o valor gama, ele tenta ajustar exatamente o conjunto de dados de treinamento.
- ***C:*** Parâmetro de penalidade do termo de erro. Ele controla a troca entre o limite de decisão suave e a classificação correta dos pontos de treinamento.
- ***degree:*** Parâmetro usado quando o kernel está definido como "poli". É basicamente o grau do polinômio usado para encontrar o hiperplano para dividir os dados.

*GradientBoosting Classifier*: Algoritmos de *boosting* são técnicas desenvolvidas mais recentemente e está sendo muito utilizada em competições de *machine learning*, por

apresentarem resultados superiores a outros modelos de *machine learning* tradicionais (DATALAB SERASA EXPERIAN, 2019). O mecanismo utilizado por algoritmos de *Gradient Boosting* é a utilização de diversos classificadores fracos, que ao serem combinados geram um classificador forte, através da estratégia de comitê. Outra característica importante deste algoritmo é a utilização de pesos, onde após uma classificação, todas os registros que foram classificados de forma equivocada recebem um peso maior do que haviam anteriormente, desta forma, na próxima iteração de classificação o algoritimo deverá dar mais importância aos registros que não foram classificados corretamente nas tentativas anteriores, aplicando esta estratégia a cada ciclo de classificação (KUHN; JOHNSON, 2013). A seguir podem ser observados alguns hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos *GradientBoosting Classifier* (BEN FRAJ, 2017):

- ***learning_rate:*** Geralmente entre 0,1 e 0,01. Se você está focado no desempenho, diminua gradualmente a taxa de aprendizado enquanto aumenta o número de árvores.
- ***n_estimators:*** Número de árvores utilizadas (na "floresta").
- ***max_depth:*** Número máximo de níveis para cada árvore.
- ***min_samples_split:*** Número mínimo de amostras necessárias para dividir um nó interno. Isso pode variar entre uma amostra em cada nó e todas as amostras em cada nó.
- ***min_samples_leaf:*** Número mínimo de amostras necessárias para estar em um nó folha.
- ***max_features:*** Número de atributos/colunas a serem considerados ao procurar a melhor divisão.

*XGBoost*: O modelo *XGBoost (eXtreme Gradient Boosting)* é uma implementação avançada do *Gradient Boosting*, trazendo algumas melhorias, como por exemplo: tempo de processamento consideravelmente reduzido, devido a implementação de processamento paralelo; implementação de regularização, que tende a evitar modelos excessivamente complexos e consequentemente diminuindo a chance de ocorrer *overfiting* (modelos muito ajustados aos dados, gerando bons resultados apenas nas bases de teste); consegue lidar com valores faltantes nativamente (ANALYTICS VIDHYA, 2016). Logo abaixo podem ser observados alguns hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos *XGBoost* (JAIN, 2016) e (REVERT, 2018):

- ***nthread***: Usado para processamento paralelo, é o número de núcleos no sistema a ser utilizado.
- ***objective:*** define a função de perda a ser minimizada. Os valores mais usados são binary:logistic, multi:softmax e multi:softprob.
- ***learning_rate:*** Geralmente entre 0,1 e 0,01. Se você está focado no desempenho, diminua gradualmente a taxa de aprendizado enquanto aumenta o número de árvores.
- ***max_depth:*** Número máximo de níveis para cada árvore.
- ***min_child_weight:*** Define a soma mínima de pesos de todas as observações necessárias em um nível abaixo. Utilizado para controlar *overfitting*.
- ***silent:*** Controle a exibição de mensagens durante execução.
- ***subsample:*** Fração de observações escolhidas aleatoriamente para cada árvore.
- ***colsample_bytree:*** Fração de atributos/colunas escolhidas aleatoriamente para cada árvore.
- ***n_estimators:*** Número de árvores utilizadas (na "floresta").

*Kneighbors Classifier*: São modelos preditivos que utilizam a lógica de fazer a predição de uma nova amostra utilizando as K amostras mais próximas, K normalmente recebe um valor ímpar para classificações binárias. A resposta gerada pelo modelo para uma nova amostra é a média entre das respostas do K vizinhos mais próximos, pode ser utilizado outra métrica no lugar da média, exemplo mediana. Como descrito anteriormente, a distância entre as amostras é de suma importância para o modelo e existem algumas formas de calcular distância entre elas, por exemplo, distância euclidiana, distância de hamming, distância manhattan e distância de markowski, a distância euclidiana é a mais comumente utilizada (KUHN; JOHNSON, 2013). A seguir, podem ser observados alguns hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos *Kneighbors Classifier* (BEN FRAJ, 2017):

- ***n_neighbors:*** Número de vizinhos a serem usados para consultas de vizinhos.
- ***p:*** Parâmetro de potência para a métrica de Minkowski. Quando p = 1, isso equivale a usar manhattan_distance (l1) e euliddean_distance (l2) para p = 2.

*Logistic Regression*: A regressão logística tem como objetivo produzir um modelo matemático que explique os dados, realizando assim a predição de uma variável de interesse, sendo ela categórica e binária. Este método estatístico se assemelha a regressão linear por produzir um modelo matemático com diferentes betas, que atribuem pesos para as variáveis explicativas, com o intuito de fazer uma predição da variável alvo, porém diferentemente da regressão linear que tem um comportamento linear, a regressão logística apresenta uma função sigmoide, ou seja, a curva se assemelha a letra S. Outra principal diferença quando comparada a regressão linear é que a variável resposta  é categórica e binária, o que a torna uma solução para classificação (KUHN; JOHNSON, 2013). Abaixo pode ser observado um dos hiper-parâmetros referentes a este modelo:

Hiper-parâmetros de modelos *Logistic Regression* (QIAO, 2019):

- ***C:*** Inverso da força de regularização, deve ser uma flutuação positiva. Como nas SVM, valores menores especificam uma regularização mais forte.

Além da possibilidade de escolher quais modelos de *machine learning* serão utilizados e seus respectivos hiper-parâmetros, a solução proposta por este trabalho possibilita a escolha de quais métricas para apuração de *score* serão utilizadas para medir o desempenho dos modelos. Foram selecionados um total de cinco métricas de *score*: *accuracy*, *f1*, *precision*, *recall e roc_auc*.

Antes de apresentar o funcionamento de cada métrica utiliza por este trabalho, é necessário ter o entendimento do funcionamento da matriz de confusão e alguns termos derivados dela, que serão utilizados pelos cálculos de *score*. A matriz de confusão apresenta a distribuição das predições realizadas pelo modelo, possibilitando identificar as quantidades de erros e acertos para cada classe. Na tabela 1 pode ser observado um exemplo de matriz de confusão, supondo uma amostra de 330 predições realizadas com o objetivo de predizer duas possíveis classes: “SIM” e “NÃO” (KUHN; JOHNSON, 2013).

Tabela 1: Matriz de confusão

| **n = 330**<br>**(número de amostras)** | **Predição:**<br>**NÃO** | **Predição:**<br>**SIM** |
|---|---|---|
| **Real:**<br>**NÃO** | 100 | 20 |
| **Real:**<br>**SIM** | 10 | 200 |

Fonte: Adaptado de MISHRA, 2018

O entendimento dos dados apresentados na tabela 1, pode ser feito em conjunto com a tabela 2, onde quatro definições são derivadas das possibilidades combinatórias da matriz de confusão, estas quatro definições serão utilizadas para os cálculos das métricas de *score* utilizadas neste estudo e podem ser observadas na tabela 2 (MISHRA, 2018).

Tabela 2: Derivação da matriz de confusão

| **Termo** | **Descrição** | **Quantidade de ocorrências encontradas na tabela 1** |
|---|---|---|
| *True Positives* | Acerto: Modelo previu “SIM” e dado real é “SIM” | 200 |
| *True Negatives* | Acerto: Modelo previu “NÃO” e dado real é “NÃO” | 100 |
| *False Positives* | Erro: Modelo previu “SIM” e dado real é “NÃO” | 20 |
| *False Negatives* | Erro: Modelo previu “NÃO” e dado real é “SIM” | 10 |

Fonte: Adaptado de MISHRA, 2018

A tabela 2 apresenta o total de ocorrências de predições, identificando as duas possíveis combinações de acerto: *true positives e true negatives*, e as duas possíveis combinações de erro: *false positives e false negatives*. Nos próximos parágrafos, cada uma das métricas de *score* disponíveis como opções pela solução proposta.

*Accuracy*: Traduzindo para português, acurácia, é calculada através da divisão entre o número correto de predições e o total de predições realizadas, por exemplo, em um cenário onde foram realizadas 100 predições, sendo 93 delas feitas de forma correta, o valor de acurácia será de 93%. Essa métrica funciona bem quando existe um equilíbrio entre a quantidade de registros de cada classe, por exemplo, supondo se tratar de um modelo preditivo para uma doença rara, onde 99% das amostras sejam da classe "negativo" e 1% sejam da classe "positivo", um modelo de *machine learning* que classifique toda base como "negativo", ainda assim atingiria uma acurácia de 99% (MISHRA, 2018).

A acurácia pode ser calculada utilizando os termos derivados da matriz de confusão, conforme exemplo apresentado pela tabela 2. O cálculo a ser realizado é o mesmo descrito no parágrafo anterior, porém utilizando os termos apresentados pela tabela 2.

$$Accuracy = \frac{TruePositives + TrueNegatives}{NúmeroAmostras} = \frac{200 + 100}{330} = 0{,}909$$

*Precision*: É o número de predições positivas realizadas corretamente (*True Positives*) dividido pelo número de total de predições positivas feitas pelo modelo (corretas e incorretas), aplicando o mecanismos aos dados do exemplo:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} = \frac{200}{200 + 20} = 0{,}909$$

*Recall*: É o número de predições positivas realizadas corretamente (*True Positives*) dividido pelo número de total amostras que deveriam ter sido classificadas como positivas, aplicando o mecanismos aos dados do exemplo:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} = \frac{200}{200 + 10} = 0{,}952$$

*F1*: A métrica de *score F1* é uma média harmônica entre as duas métricas anteriores, combinando as métricas *precision* e *recall*. Utilizando esta combinação é possível calcular o quão preciso e robusto é o modelo, ou seja, quantas predições faz corretamente, porém

considerando também se comete muitos erros relevantes. Utilizando esta métrica, pode se evitar o falso desempenho apresentado pela métrica de acurácia, para os casos de quantidade de amostras desbalanceadas entre as classes, aplicando a fórmula aos dados do exemplo (MISHRA, 2018):

$$F1 = 2 \cdot \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \cdot \frac{1}{\frac{1}{0,909} + \frac{1}{0,952}} = 0,93$$

*roc_auc*: a métrica *roc_auc*, é a combinação entre AUC (*area under the curve*) e ROC (*Receiver Operating Characteristic*), ou seja, roc_auc é a área sob a curva ROC, por este motivo, inicialmente é necessário ter o entendimento sobre a curva ROC (RODRIGUES, 2018).

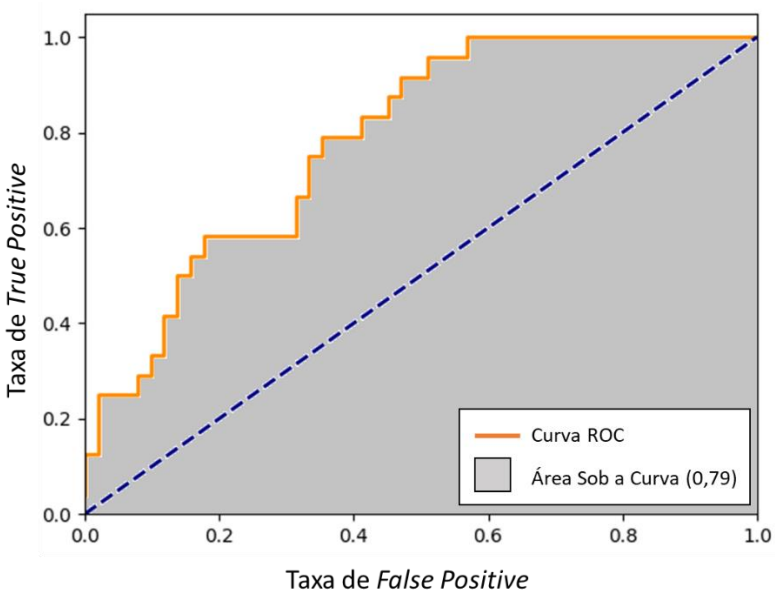
A curva ROC é representada pela plotagem de duas métricas em diferentes limiares do gráfico, as duas métricas utilizadas são: taxa de verdadeiro positivo (*True Positive Rate*), corresponde a proporção de predições corretamente classificadas como positivas em relação a todas as predições positivas e taxa de falso positivo (*False Positive Rate*), corresponde a proporção de predições negativas erroneamente classificadas como positivas em relação a todas as predições negativas, matematicamente representadas por:

$$True\ Positive\ Rate = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$False\ Positive\ Rate = \frac{FalsePositives}{FalsePositives + TrueNegatives}$$

Ambas as taxas, *True Positive Rate e False Positive Rate*, variam no intervalo entre 0 e 1, o gráfico de curva ROC calcula ambas as taxas para valores limiares entre 0 e 1, gerando assim um gráfico representado pela figura 2 (RODRIGUES, 2018).

Figura 2: Área sob a curva ROC




Fonte: Adaptado de MISHRA, 2018

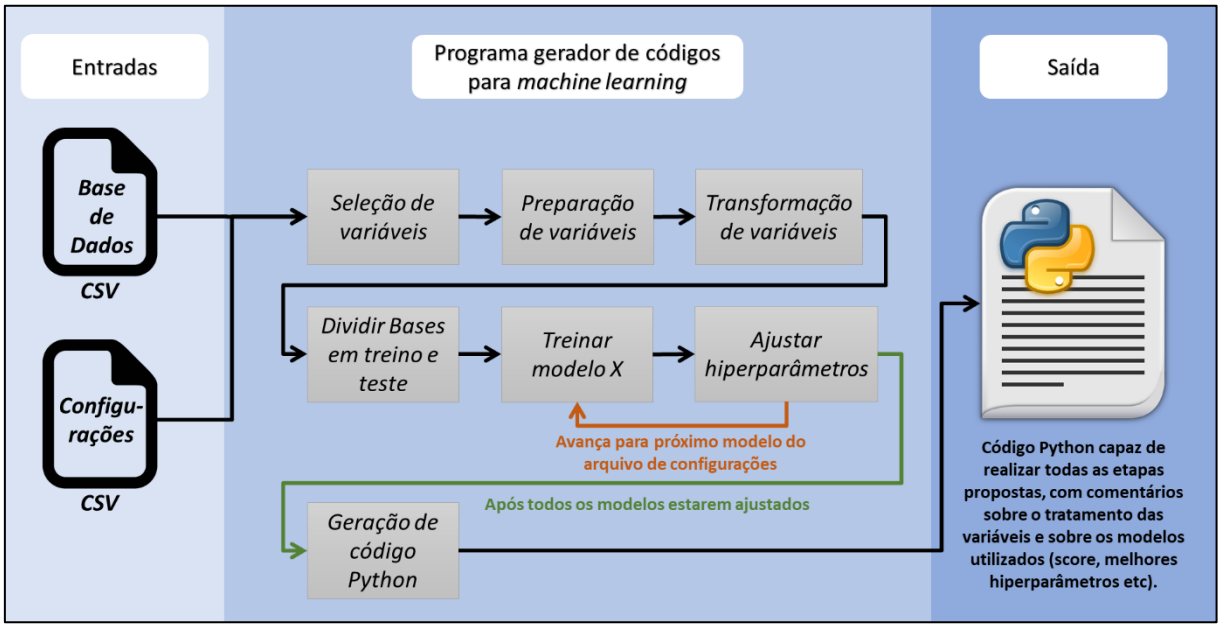
Após a plotagem da curva ROC, conforme exemplificada na figura 2, a métrica roc_auc é gerada através do cálculo de área que está abaixo da curva ROC, destacado pela cor cinza na figura 2, pelo fato de ambos os eixos utilizados no gráfico variarem entre 0 e 1, a área sob a curva terá valor máximo de 1, ou seja, o máximo valor possível para a métrica roc_auc é 1, o que equivaleria a um modelo perfeito (RODRIGUES, 2018).

Todas as combinações entre modelos de *machine learning* e métricas de *score* poderão ser utilizados pela solução proposta neste estudo, no próximo capítulo será apresentada a implementação da solução, onde será possível escolher quais modelos e métricas se deseja utilizar.

# 3 IMPLEMENTAÇÃO DA SOLUÇÃO

Neste capítulo será apresentada a implementação da solução proposta para o estudo de automatização de geração de código para *machine learning*. Na figura 3 pode ser observado o fluxo das etapas abordadas pela solução de automatização de geração de código para *machine learning*.

Figura 3: Fluxo realizado pela solução de automatização de machine learning




Fonte: Desenvolvido pelo autor

A implementação foi realizada em linguagem Python e tem como objetivo gerar um segundo programa, também em linguagem Python, que seja capaz de realizar os processos descritos como escopo deste estudo nos capítulos 2 e 3.

Serão abordados neste capítulo os pré-requisitos da solução, assim como suas limitações e etapas de importação dos dados, funções utilizadas pela solução, seleção de variáveis, tratamento dos dados, separação de bases de teste e treino, aplicação dos modelos de *machine learning* e por fim, geração de código python objetivo fim do estudo.

## 3.1 Pré requisitos e parametrizações

O programa desenvolvido para atender a solução proposta, espera como entrada dois arquivos, o primeiro deles é a base de dados, que deve estar em formato CSV, conter um cabeçalho e suas características devem ser configuradas no segundo arquivo de entrada,

denominado arquivo de configurações, características estas: caractere utilizado como separador de campos, caractere utilizado como identificador de casa decimal, nome da variável *target* e local onde o arquivo de dados estará disponível. Além das configurações pertinentes ao arquivo que contém a base de dados, também devem ser configurados quais os modelos de *machine learning* e seus respectivos hiper-parâmetros a serem testados.

Registros com valores nulos (*missing*), devem estar sem valores na base, caso a base contenha algum valor representando a falta de dados, por exemplo, ‘N/A’ ou ‘?’, este valor deve ser substituído por nulo para que o algoritmo possa reconhecer os casos em que ocorram valores nulos, realizando assim o tratamento de variáveis corretamente.

O arquivo de configurações deve estar em formato texto (TXT), a escolha por este formato foi feita, pois a intenção é que este arquivo seja o mais amigável possível, sendo que qualquer alteração de configuração da base de dados ou alteração de valores para testes de hiper-parâmetros, por exemplo, possam ser realizados por pessoas que não conheçam formatos mais estruturados, como por exemplo, o formato JSON. A utilização de formatos estruturados ou semiestruturados possibilitaria uma diminuição na complexidade da codificação da solução, em contrapartida não estaria em um formato tão amigável para pessoas que não conheçam a forma como estes arquivos devem ser estruturados.

Todas as parametrizações utilizadas pela solução proposta, conforme descrito anteriormente, devem ser realizadas através do arquivo de configuração. Um exemplo do arquivo de configurações, que foi utilizado para o estudo da primeira base de dados analisada por este estudo, pode ser observado no apêndice A deste trabalho.

Os parâmetros que devem ser definidos no arquivo de configurações são: local do *dataset* (dados), delimitador utilizado no CSV (delimitador), campo que contém a variável resposta a ser utilizada pelos modelos de *machine learning* (target), separador de casas decimais (separador_decimal) e local que deve ser gerado o código final, resultante do processamento da solução proposta (codigo_out).

Após as configurações pertinentes ao arquivo que contém a base de dados, devem ser adicionados os modelos que serão utilizados pela solução de automatização e seus respectivos hiper-parâmetros a serem testados.

Foi adotada a solução de parametrização dos hiper-parâmetros para possibilitar o ajuste preciso por parte do cientista de dados, caso os hiper-parâmetros estivessem pré-estabelecidos, os ajustes dos modelos poderiam demorar demasiadamente, dependendo do volume de dados e quantidade de hiper-parâmetros a serem testados, por outro lado, caso

estivessem pré-estabelecidos poucos hiper-parâmetros, a solução não seria capaz de ajustar com muita eficácia cada modelo a ser utilizado.

Nos próximos tópicos serão apresentados todos os módulos desenvolvidos para o funcionamento da solução de automatização de *machine learning*, detalhando cada passo e estratégia adotada para possibilitar o estudo de viabilidade proposto.

## 3.2 Importação do arquivo de configurações

O primeiro módulo utilizado pela solução, é responsável pela leitura do arquivo de configurações e criação de variáveis que serão utilizadas posteriormente. A implementação realizada pode ser observada no apêndice B, após a linha de comentário “Lendo arquivo de Configurações”.

Inicialmente é realizado um laço para leitura das 5 primeiras linhas e utilizada a função “exec()”, função esta que executa o comando presente na linha previamente lida, desta forma as variáveis *dados*, *delimitador*, *target*, *separador_decimal*, *codigo_out* são criadas conforme configurado no arquivo de configurações.

No segundo laço implementado, a mesma lógica descrita no parágrafo anterior foi implementada, porém neste caso, a variável *score* será criada, conforme arquivo de configuração, contendo todas as métricas de *score* que se deseja calcular para cada modelo.

No terceiro laço implementado, as informações dos modelos a serem utilizados são lidos, informações presentes a partir da oitava linha do arquivo de configurações, a variável *model_param* será utilizada posteriormente para execução dos modelos.

## 3.3 Funções de geração de código

Foram desenvolvidas duas funções para auxiliar na reutilização de código, simplificando as chamadas recorrentes utilizadas posteriormente pela solução. Visando atender a necessidade de gerar o código final para apoio ao cientista de dados, com todos os passos necessários para o tratamento e seleção de variáveis, assim como ajuste dos modelos e comentários pertinentes para auxiliar na análise, foi implementado o código disponível no apêndice B a partir da linha de comentário “Funcoes de apoio”.

Ambas as funções incrementam a variável *codigo*, variável que posteriormente será utilizada para geração do código python, objetivo fim do estudo. Esta variável foi implementada como global, pois seu valor será incrementado durante toda a execução da solução, sendo que as funções que a utilizarão são chamadas diversas vezes ao longo do

código, caso esta variável não fosse global, ao ser chamada uma segunda vez pela função *gera_codigo*, por exemplo, teria seu valor anterior perdido.

A função *gera_codigo* é responsável por adicionar uma nova linha que será gerada no código final, recebendo como parâmetros o texto que se deseja gerar no código final e os parâmetros que definam a quantidade de quebras de linha que devem ser utilizadas antes e depois do texto.

A função *gera_codigo_df* tem uma finalidade similar a função *gera_codigo*, porém adiciona um *data frame* como comentário ao código final, esta abordagem foi implementada para facilitar a visualização de dados tabulares, que podem ser úteis para auxiliar nas análises a serem realizadas pelo cientista de dados. Para os dados contidos em um *data frame* serem apresentados em formato texto, foi utilizada a função *tabulate* da biblioteca de mesmo nome.

Ambas as funções serão muito utilizadas durante toda codificação da solução de automatização de *machine learning*, pois para cada comando executado pela solução, a mesma linha de código também deverá ser gerada no código final, objetivo fim deste trabalho.

### 3.4 Função para ajuste de modelos de machine learning

Para realização dos ajustes e testes de hiper-parâmetros conforme parametrizados no arquivo de configurações, foi implementada a função *executa_modelo*. Esta função recebe como parâmetros a métrica de *score*, o modelo que deve ser ajustado e quais hiper-parâmetros devem ser testados.

São utilizadas algumas variáveis para armazenar os dados após ajuste e testes dos modelos executados, são elas: *lmod_nome* (nome do modelo), *lmod_score* (métrica de score), *lmod_tempo* (tempo para testar todas as combinações de hiper-parâmetros), *lmod_acc_treino* (score na base de treino), *lmod_acc_teste* (score na base de teste) e *lmod_acc_hiperparam* (melhor hiper-parâmetro). Essas variáveis serão utilizadas para geração de bloco comentado no código final, com todas as informações coletadas para cada modelo de *machine learning*.

A função *executa_modelo* também é responsável pela geração das linhas para ajuste de todos os modelos estatísticos no código final, auxiliando assim a posterior utilização do modelo pelo cientista de dados, não sendo necessário outras fontes de pesquisa para apoiar na escrita do código para tais ajustes. A implementação da função *executa_modelo* pode ser observada no código disponível no apêndice B, algumas linhas abaixo da linha de comentário “Funcoes de apoio”.

Esta função será executada de forma cíclica, para que todas as combinações de métricas de *score* e modelos parametrizados no arquivo de configurações, sejam processados. Esta chamada de forma cíclica da função será descrita no capítulo 3.10.

### 3.5 Importação de módulos e pacotes

Todos os módulos e pacotes utilizados pela solução de automatização de *machine learning* são importados no trecho de código disponível no apêndice B logo após a linha de comentário “Importação de Modulos e Pacotes”.

Todos comandos para importação dos módulos e pacotes serão gerados no código final, sendo que as mesmas importações serão necessárias para execução do código final, quando realizada de forma independente.

### 3.6 Leitura do dataset

Utilizando os parâmetros do arquivo de configurações, explicado no item 3.2, o código implementado irá gravar na variável “*codigo*" (responsável por receber todas as linhas do código final que será gerado ao final da solução) o nome da variável resposta (*target*), assim como a criação de um *data frame* com os dados da base a ser estudada, a partir do arquivo CSV de entrada, respeitando as configurações anteriormente citadas. A implementação realizada pode ser observada no apêndice B logo após a linha de comentário “Criando Data Frame inicial”.

### 3.7 Seleção de variáveis e identificação de tratamento

Uma das tarefas mais importantes realizadas pela solução de automatização de *machine learning*, é a de seleção de variáveis, para realização desta tarefa sem a intervenção humana, foi adotada uma abordagem de análise descritiva das variáveis, com exceção da variável resposta, e análise dos resultados para enquadrar cada variável em uma das condições prevista para tratamento.

Na tabela 3 podem ser observados todos os cálculos realizados para cada variável, a tabela em questão não apresenta os resultados de nenhum cálculo e sim a relação dos cálculos que serão realizados quando a solução de automatização de *machine learning* for executada.

Dependendo do valor, ou combinação de valores, dos atributos calculados para uma variável, a solução irá adotar uma estratégia para seleção e tratamento desta variável, por

exemplo, um dos atributos calculados é o percentual de valores distintos na base, caso este valor seja muito elevado, a solução irá desconsiderar esta variável, pois uma variável com muitos valores distintos tem uma característica de valores sequenciais, como IDs (código de identificação) por exemplo, não tendo relevância para estudos de *machine learning*.

Todas as estratégias de seleção e tratamento de variáveis serão descritas nas próximas páginas e foram assumidas pelo autor, considerando sua vivência na área de dados, não sendo apresentado uma referência formal para tal embasamento, o intuito do estudo proposto é apresentar uma solução viável e funcional para automatização de *machine learning*, porém é esperado que nem todas as regras sejam satisfatórias para todos os cenários, assim como a necessidade de criação de novas regras é algo também esperado.

Tabela 3: Métricas utilizadas para seleção e tratamento de variáveis

| Atributo | Descrição | Fórmula |
|---|---|---|
| Coluna | Nome da Coluna. | Não se aplica |
| Tipo | Tipo da Coluna (*int / float / object*). | Não se aplica |
| Media | Valor em torno do qual há um equilíbrio na distribuição dos dados, aplicável as variáveis numéricas. | $\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$ |
| Mediana | Valor central do conjunto de dados ordenado, aplicável as variáveis numéricas. | Se número de observação (n) for ímpar, é o valor no meio, posição: $\frac{(n+1)}{2}$ Caso contrário, é a média aritmética entre os elementos de posições: $\frac{n}{2}$ e $\left(\frac{n}{2}\right)+1$ |
| Desvio Padrao | Medida de dispersão, quantifica a variabilidade dos dados em torno da média estando na mesma escala em que os dados foram medidos, aplicável as variáveis numéricas. | $dp(x) = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}}$ |
| Coef de Variacao | Percentual de variabilidade em relação à média observada, aplicável as variáveis numéricas. | $cv = 100\frac{dp(x)}{\bar{x}}$ |
| Coef de Centralidade | Métrica criada para o estudo, utilizada para indicar a disparidade entre a média | $coef_{centralidade} = \frac{mediana}{média}$ |

| | | |
|---|---|---|
| | e mediana, aplicável as variáveis numéricas. | |
| Perc_Distintos | Percentual de valores distintos observados. | Não se aplica |
| Valores_Distintos | Quantidade de valores distintos observados. | Não se aplica |
| Perc_Null | Percentual de valores nulos observados. | Não se aplica |

Fonte: Desenvolvido pelo autor baseado em BUSSAB; MORETTIN, 2010

Todas as métricas descritas na tabela 3 serão calculadas para as variáveis quantitativas, porém apenas as métricas *Perc_Distintos*, *Valores_Distintos* e *Perc_Null* serão calculadas para as variáveis categóricas.

A partir da obtenção dos resultados das métricas para cada variável, é realizado o enquadramento das variáveis em uma estratégia a ser adotada. Conforme abordado anteriormente, as regras utilizadas para enquadramento de cada variável foram pensadas na vivência de análise de dados do autor deste trabalho, por este motivo, é esperado que sejam necessários ajustes futuros nas regras de enquadramento.

O mecanismo de enquadramento verifica algumas condições segundo a ordem definida no código, ao ser enquadrado em uma das regras, não será enquadrada em nenhuma regra subsequente. Na tabela 4 podemos visualizar todas as regras de enquadramento adotadas pela solução de automatização de *machine learning*.

Tabela 4: Regras para enquadramento de variáveis explicativas

| **Regra para enquadramento** | **Ação Adotada** | **Descrição da Ação Adotada** |
|---|---|---|
| Se Tipo = 'Numérico' E Perc_Distintos > 70% E Coef de Variacao > 50 % E Coef de Centralidade > 80 % | Excluir variável | alta dispersao |
| Se Perc_Null > 75 % | Excluir variável | muito nulo |
| Se Valores_Distintos < 30% E Perc_Null > 0% | Utilizar variável | dummy_com_null |
| Se Valores_Distintos < 30% E Perc_Null = 0% | Utilizar variável | dummy_sem_null |
| Se Tipo = 'Numérico' E | Utilizar variável | continua_com_null |

| | | |
|---|---|---|
| Perc_Null > 0% | | |
| Se Tipo = 'Numérico' E Perc_Null = 0% | Utilizar variável | continua_sem_null |
| Se Tipo = 'object' E Perc_Distintos > 80% | Excluir variável | categorica com muitas categorias |
| Se Tipo = 'object' | Excluir variável (filtrar) | possivel Feature Engineering |
| Não enquadramento em nenhuma regra anterior | nao especificado | - |

Fonte: Desenvolvido pelo autor

Visando auxiliar o cientista após a execução da solução de automatização de *machine learning*, será adicionado ao código final a relação das variáveis, as métricas calculadas e a ação que foi tomada pela solução de forma automatizada, dando insumos para caso necessário, ser alterada alguma das regras de enquadramento acima ou a alteração da ação efetiva realizada em uma determinada variável.

Todo código responsável pelo cálculo das métricas para cada variável, descrito na tabela 3, e definição das regras de enquadramento para cada variável e definição das ações realizadas sobre cada variável, descritas na tabela 4, podem ser encontradas no apêndice B, a partir da linha em que existe o comentário “Seleção de Variáveis e Identificação de tratamento”.

### 3.8 Tratamento de dados

Após o enquadramento de cada variável ter sido realizada, a solução de automatização de *machine learning* irá realizar o tratamento dos dados conforme enquadramento identificado.

Todos os comandos para preparar o novo *data frame*, apenas com as variáveis identificadas para serem utilizadas e com seus respectivos tratamentos, serão adicionados ao código final. As Regras para tratamento das variáveis, conforme enquadramento realizado, podem ser observadas na tabela 5.

Tabela 5: Regras para tratamento de variáveis

| **Ação identificada no Enquadramento** | **Tratamento Realizado** |
|---|---|
| Variável resposta | Sem tratamento.<br>Adicionar ao Novo *Data Frame*. |
| continua_sem_null | Sem tratamento.<br>Adicionar ao Novo *Data Frame*. |
| continua_com_null | Substitui valores nulos pela média.<br>Adicionar ao Novo *Data Frame*. |
| dummy_sem_null | Criação de colunas *Dummies*.<br>Adicionar ao Novo *Data Frame*. |
| dummy_com_null | Substituir valores nulos pelo texto "NULL".<br>Criação de colunas *Dummies*.<br>Adicionar ao Novo *Data Frame*. |
| Demais ações, como:<br>excluir<br>filtrar<br>nao especificado | Não serão adicionadas ao Novo *Data Frame*. |

Fonte: Desenvolvido pelo autor

Inicialmente foi adotado uma abordagem única para tratamento de nulos na base, para variáveis categóricas a estratégia foi criar uma categoria com valor "NULL", para as variáveis contínuas a estratégia foi utilizar a média. Foram realizados testes de combinação para identificar a melhor estratégia para substituição de valores nulos para as variáveis contínuas, utilização da média e utilização da mediana, porém a alteração no resultado não tem um comportamento linear para todos os modelos, por este motivo foi definido realizar apenas a média, caso o cientista de dados prefira alterar este tratamento pode realizar a alteração na linha do código final em que ocorre tal tratamento.

Novamente, cabe ressaltar, que as regras para tratamento de variáveis foram concebidas com intuito de ser um estudo inicial para automatização de processos de *machine learning*, sendo esperados ajustes futuros para melhoria da solução. A implementação do tratamento de dados de acordo com as regras enquadradas para cada variável pode ser observada no apêndice B, a partir da linha com o comentário "Tratamento de Dados".

### 3.9 Separação das bases de treino e teste

Com o objetivo de validar os modelos de dados trabalhados pela solução de automatização de *machine learning* e validar a possibilidade de um modelo estar super ajustado aos dados, comumente conhecido por *overfitting* ou modelo "viciado", foi utilizado o mecanismo de separação da base de dados, gerando duas bases: base de treino e base de testes (KUHN; JOHNSON, 2013).

Os modelos serão construídos utilizando apenas a base de treino, definida para ser 75% dos dados da base original e utilizaremos a base de testes, definida para ser 25% dos dados da base original, para calcular o *score* do modelo e assim identificar o real potencial do modelo, sendo possível identificar *overfitting*, no caso do *score* do modelo sobre a base de treino seja muito superior ao *score* do modelo sobre a base de teste.

Foram realizados alguns testes para identificar qual a melhor proporção para separação da base original em bases de treino e teste, não foi encontrada uma combinação em que se sobressaiu em todos os modelos e bases aplicadas neste estudo, porém a divisão apresentada no parágrafo anterior, apresentou uma média razoável entre os cenários testados. A implementação deste mecanismo pode ser observada no apêndice B, após a linha de comentário "Separando bases de teste e treino".

### 3.10 Aplicando e ajustando os modelos de machine learning

Após todos os dados terem sido selecionados, tratados e separados em bases de treino e testes, os modelos podem ser executados e medidos, a solução de automatização de *machine learning* utiliza a função descrita no capítulo 3.4 de forma repetitiva, ajustando todos os modelos com seus respectivos hiper-parâmetros conforme parametrizados no arquivo de configurações, descritos no capítulo 3.1.

Além da parametrização dos modelos e seus respectivos hiper-parâmetros, as métricas para o cálculo do *score* também serão podem ser parametrizadas, adicionando ou removendo as métricas desejadas. Desta forma, caso sejam adicionados ou removidos modelos ao arquivo de configurações, adicionado ou removido seus respectivos hiper-parâmetros e adicionando ou removendo métricas de *score*, a solução é capaz de iterar sobre estas combinações. A implementação deste mecanismo pode ser observada no apêndice B a partir da linha de comentário "Aplicando e Ajustando Modelos".

Após a aplicação de todos os modelos e seus respectivos hiper-parâmetros, algumas variáveis receberão todos os dados referentes a tempos de execução, melhores hiper-parâmetros, métricas de *score* e valores de *scores* observados nas bases de treino e de teste. Estas variáveis são utilizadas para criação de um *data frame* com todas as informações que serão adicionadas ao código final como comentários, para assim auxiliar ao cientista de dados na escolha do modelo de *machine learning* mais adequado.

A implementação realizada para preparação do *data frame* mencionado anteriormente, a partir das variáveis populadas em tempo de execução dos modelos e a gravação do conteúdo deste *data frame* em formato de comentário, na variável que recebe as linhas do código final, pode ser observada no apêndice B, a partir da linha de comentário “Comparativo de Modelos”.

### 3.11 Geração de código python final

Durante toda a execução dos códigos contidos na solução de automatização de *machine learning*, foram executadas as funções descritas no capítulo 3.3, *gera_codigo* e *gera_codigo_df*, ambas as funções são responsáveis pelo incremento da variável *codigo*, desta forma, esta variável contém todo o código Python que se deseja gerar para o cientista de dados.

Para geração do código final, que é o objetivo fim deste estudo, foram implementadas as linhas de código presentes no apêndice B, logo após a linha de comentário “Gerando codigo.py”. O arquivo com o código final receberá o nome parametrizado no arquivo de configurações, conforme explicado no capítulo 3.2.

# 4 ANÁLISE DOS RESULTADOS

Neste capítulo analisaremos a saída disponibilizada pela solução de automatização de *machine learning*. A saída esperada da solução de automatização de *machine learning* é um código em linguagem Python, que contém todos os comandos necessários para que o cientista de dados possa executar todas as etapas abordadas nos capítulos anteriores e utilize a solução proposta como ferramenta inicial para fazer todos os ajustes que julgar necessário.

Para realização dos testes apresentados neste capítulo, foram utilizadas três bases de dados, a primeira delas é *"Titanic: Machine Learning from Disaster"*, utilizada como case introdutório pelo site de competições de *machine learning www.kaggle.com*. O objetivo proposto pelo estudo desta base é de característica classificatória, onde devem ser analisados os dados de cada passageiro e predizer quais passageiros sobreviveram ou morreram no naufrágio do Titanic (KAGGLE, 2012). Esta base tem um total de 892 registros, para simplificar a referenciação desta base ao longo do trabalho, esta será referenciada como "base TITANIC".

A segunda base utilizada neste estudo, contém dados provenientes de telemarketing bancário, sendo dados de clientes em que foram alvos de campanhas de *marketing* para fomentar o investimento a curto e médio prazo. O objetivo do estudo sobre esta base é identificar quais clientes irão ou não irão realizar investimentos, dadas as características de seus atributos. Da mesma forma que a primeira base, esta segunda também está disponível no site www.kaggle.com (SHARAN, 2018). Esta base tem um total de 45.212 registros, para simplificar a referenciação desta base ao longo do trabalho, esta será referenciada como "base INVESTIMENTO".

A terceira base utilizada, apresenta dados que possibilitam a predição de intensão de compras *online*, cada registro da base consiste em um usuário e seu perfil de navegação. Da mesma forma que as bases anteriores, esta base também pode ser acessada pela plataforma do site www.kaggle.com (SHARMA, 2019). Esta base tem um total de 12.331 registros, para simplificar a referenciação desta base ao longo do trabalho, esta será referenciada como "base COMPRAS ONLINE".

Os arquivos de configuração e códigos finais gerados pela solução, para cada base utilizada por este estudo, podem ser observados na íntegra na sessão de apêndices, organizados da seguinte forma:

- **Apêndice A:** Arquivo de configurações para processamento da base TITANIC.
- **Apêndice C:** Código final gerado pela solução após processamento da base TITANIC.
- **Apêndice D:** Arquivo de configurações para processamento da base INVESTIMENTO.
- **Apêndice E:** Código final gerado pela solução após processamento da base INVESTIMENTO.
- **Apêndice F:** Arquivo de configurações para processamento da base COMPRAS ONLINE.
- **Apêndice G:** Código final gerado pela solução após processamento da base COMPRAS ONLINE.

Todos os códigos finais gerados pela solução, apresentarão os mesmos blocos de códigos responsáveis por executar as funções: Importação de módulos e dados, análise de variáveis, correlação entre variáveis, preparação de dados, separação de base de dados em treino e teste, aplicação dos modelos de *machine learning* e comparativo de modelos.

A solução proposta por este estudo irá gerar códigos finais diferentes para cada base de dados utilizada neste estudo, devido as variáveis pertencentes a cada base e seus respectivos valores, porém os blocos de códigos descritos no parágrafo anterior se repetem para todas as bases, por este motivo, neste capítulo, serão apresentados todos os blocos de códigos gerados apenas para a base TITANIC, pois o objetivo de se ter o entendimento do código final gerado será atingido com o estudo de apenas uma das bases, não sendo necessário repetir a análise para cada uma das bases.

O comparativo e análise dos resultados das três bases utilizadas por este estudo, serão apresentadas neste mesmo capítulo, após a análise do código final gerado para a base TITANIC, conforme descrito no parágrafo anterior.

Inicialmente o código final apresenta todos os módulo e pacotes que serão utilizados, definição de variável alvo e leitura de arquivo com os dados, conforme pode ser observado na figura 4.

Figura 4: Código final Python - Importação de módulos e dados

```python
# Importando Modulos e Pacotes
import numpy as np
import pandas as pd
from tabulate import tabulate
import sklearn.model_selection as model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import datetime

# Atribuindo parametros de entrada
target = 'Survived'

# Importando os dados de entrada
df = pd.read_csv('C:/Users/rcosin/Desktop/FIA/TCC/Titanic/train.csv', sep = ',', decimal = '.')
```

Fonte: Desenvolvido pelo autor

Todas as importações de pacotes e módulos são realizadas no código apresentado na figura 4, assim como a criação do *dataframe* inicial que será utilizado nos próximos passos, utilizando as configurações de caractere separador de colunas e caractere separador de casas decimais, ambas oriundas do arquivo de configurações utilizado pela solução. Da mesma forma, através do arquivo de configurações, o código define qual o nome da variável *target*.

Conforme descrito no capítulo 3.7, referente a seleção de variáveis e identificação de tratamento, a solução realiza determinadas análises e define ações que serão aplicadas, a depender do resultado das análises. Na figura 5 é possível visualizar o resultado desta etapa da solução.

Figura 5: Código final Python - Análise de variáveis

```
# Analise de variaveis e identificacao de tratamento:
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Coluna      | Tipo    | Desvio Padrao| Media ... Perc_Null | Acao              | Desc Acao                         |
#+=============+=========+==============+====== ... ==========+===================+===================================+
#| PassengerId | int64   | 257.35       | 446.0 ...       0   | excluir           | alta dispersao                    |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Pclass      | int64   | 0.84         | 2.31  ...       0   | dummy_sem_null    | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Name        | object  | -            | -     ...       0   | excluir           | categorica com muitas categorias  |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Sex         | object  | -            | -     ...       0   | dummy_sem_null    | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Age         | float64 | 14.53        | 29.7  ...   19.87   | continua_com_null | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| SibSp       | int64   | 1.1          | 0.52  ...       0   | dummy_sem_null    | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Parch       | int64   | 0.81         | 0.38  ...       0   | dummy_sem_null    | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Ticket      | object  | -            | -     ...       0   | filtrar           | possivel Feature Engineering      |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Fare        | float64 | 49.69        | 32.2  ...       0   | continua_sem_null | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Cabin       | object  | -            | -     ...    77.1   | excluir           | muito nulo                        |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
#| Embarked    | object  | -            | -     ...    0.22   | dummy_com_null    | -                                 |
#+-------------+---------+--------------+------ ... ----------+-------------------+-----------------------------------+
```

Fonte: Desenvolvido pelo autor

Conforme pode ser observado na figura 5, a análise das variáveis e a descrição da ação que será realizada sobre elas, é apresentada em forma de comentário, facilitando o entendimento do cientista de dados sobre as estratégias e cálculos que foram realizados. Os dados apresentados na figura 5 foram suprimidos, para que seja possível a visualização no quadro, no código original são apresentados os valores: *Coluna, Tipo, Desvio Padrao, Media, Coef de Variacao, Mediana, Coef de Central, Perc_Distintos, Valores_Distintos, Perc_Null, Acao e Desc Acao*. O código na íntegra pode ser encontrado no apêndice C.

Com o objetivo de dar insumos ao cientista de dados, novamente em forma de comentários, pode ser visto na figura 6 a tabela de correlação entre as variáveis numéricas.

Figura 6: Código final Python - Correlação entre variáveis

```
# Correlacao entre as variaveis:
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#|             |   PassengerId |   Survived |   Pclass |    Age |   SibSp |   Parch |   Fare |
#+=============+===============+============+==========+========+=========+=========+========+
#| PassengerId |         1     |     -0.005 |   -0.035 |  0.037 |  -0.058 |  -0.002 |  0.013 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Survived    |        -0.005 |      1     |   -0.338 | -0.077 |  -0.035 |   0.082 |  0.257 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Pclass      |        -0.035 |     -0.338 |    1     | -0.369 |   0.083 |   0.018 | -0.549 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Age         |         0.037 |     -0.077 |   -0.369 |  1     |  -0.308 |  -0.189 |  0.096 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| SibSp       |        -0.058 |     -0.035 |    0.083 | -0.308 |   1     |   0.415 |  0.16  |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Parch       |        -0.002 |      0.082 |    0.018 | -0.189 |   0.415 |   1     |  0.216 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Fare        |         0.013 |      0.257 |   -0.549 |  0.096 |   0.16  |   0.216 |  1     |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
```

Fonte: Desenvolvido pelo autor

A correlação entre variáveis só ocorre para as variáveis quantitativas, por este motivo as variáveis categóricas não fazem parte deste quadro, o objetivo deste trecho de código, em forma de comentário, é novamente gerar insumos para auxiliar o cientista de dados em sua análise.

Dando continuidade à análise do código gerado, o próximo passo realiza o tratamento de dados descritos no capítulo 3.8, aplicando as regras definidas para cada ação conforme enquadramento ocorrido para cada variável. Esta etapa pode ser observada na figura 7.

Figura 7: Código final Python - Preparação de dados

```
# Tratamento de Dados
df2 = pd.concat([df['Survived']])

# Criando Dataframe auxiliar
df_temp = pd.DataFrame()

# Adicionando colunas continuas sem nulos
df2 = pd.concat([df['Fare'], df2], axis=1)

# Adicionando colunas continuas com nulos
df_temp = df.Age.fillna( df.Age.mean() )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies sem tratamento de NULL
df_temp = pd.get_dummies( df.Pclass, prefix='Pclass' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Sex, prefix='Sex' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.SibSp, prefix='SibSp' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Parch, prefix='Parch' )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies com tratamento de NULL
df_temp = df.Embarked.fillna( 'NULL' )
df_temp = pd.get_dummies( df_temp, prefix='Embarked' )
df2 = pd.concat([df_temp, df2], axis=1)
```

Fonte: Desenvolvido pelo autor

Na etapa apresentada pela figura 7, ocorre a criação de um novo *dataframe*, com todos os campos que serão utilizados pelos modelos de *machine learning*, adicionando a variável alvo e posteriormente cada variável com suas respectivas tratativas, de acordo com as ações respectivamente enquadradas.

Após a preparação do novo *dataframe*, com todos os atributos necessários para realizar o processamento pelos modelos de *machine learning*, é necessário fazer a separação dos dados em bases de treino e teste, como visto anteriormente no capítulo 3.9, código apresentado pela figura 8.

Figura 8: Código final Python - Separação de bases teste e treino

```
105  # Dividindo a base entre variaveis explicativas e variavel resposta
106  X = df2.loc[:, df2.columns !='Survived']
107  Y = df2.Survived
108
109  # Separando a Base entre Teste e Treino
110 X_train, X_test, Y_train, Y_test = model_selection.train_test_split \
111              (X, Y, train_size=0.75, test_size=0.25, random_state=7)
```

Fonte: Desenvolvido pelo autor

Para a etapa de separação da base de dados, primeiramente é realizada a separação entre as variáveis explicativas e a variável resposta gerando assim dois *dataframes* (X e Y), posteriormente ambos os *dataframes* são divididos em dois, neste estudo foi utilizado a estratégia de separação descrita no capítulo 3.9, gerando por fim um total de quatro *dataframes*: X_train, contendo as variáveis explicativas de 75% da base original; X_test, contendo as variáveis explicativas de 25% da base original; Y_train, contendo a variável resposta de 75% da base original; Y_test, contendo a variável resposta de 25% da base original.

Com as bases previamente separadas entre teste e treino, os modelos de *machine learning* podem ser aplicados, o código final contém a aplicação de todos os modelos parametrizados combinados com as métricas de *score* também parametrizadas.

No arquivo de configurações utilizado para o processamento da base TITANIC, foram parametrizados sete modelos de *machine learning* (*DecisionTree Classifier, XGB Classifier, RandomForest Classifier, GradientBoosting Classifier, KNeighbors Classifier, Logistic Regression e SVC*), e cinco métricas de *score* (*accuracy, f1, precision, recall e roc_auc*), desta forma o código final apresentará a combinação de todos os modelos e métricas de *score* totalizando 35 modelos de *machine learning* e respectivas métricas de *score*.

Com o objetivo de atingir o entendimento do código gerado, não se faz necessária a análise te todos os 35 modelos supracitados, desta forma, a figura 9 apresenta o primeiro modelo ajustado no código final.

Figura 9: Código final Python - Modelo: DecisionTreeClassifier

```
114  # Modelo: DecisionTreeClassifier
115  modelo = DecisionTreeClassifier()
116
117  # Tunning:
118  parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_sample
119  mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10, refit=
120  mod.fit(X_train, Y_train)
121
122  # Melhores Parametros: {'max_depth': 12, 'max_features': 18, 'min_samples_leaf': 5, 'min_samp
123
124  # Aplicando o Modelo
125  model = DecisionTreeClassifier(**mod.best_params_)
126  model.fit(X_train, Y_train)
127  predict_train = model.predict(X_train)
128  predict_test = model.predict(X_test)
129
130  # Calculando o Score
131  score_treino = round(accuracy_score(Y_train, predict_train),6)
132  score_teste = round(accuracy_score(Y_test, predict_test),6)
133
134  # ------------------------------------------------
```

Fonte: Desenvolvido pelo autor

Ao executar o código apresentado na figura 9, é executado o modelo de *machine learning DecisionTreeClassifier*, assim como a iteração da combinação de hiper-parâmetros definidos no arquivo de configurações. Também são apresentados os comandos necessários para o cálculo do *score* e adicionalmente ao código, é apresentado a melhor combinação de hiper-parâmetro em forma de comentário.

Algumas linhas de código foram truncadas para possibilitar a apresentação no quadro. De forma similar, todas as demais 34 combinações de modelos de *machine learning e* métricas de *score*, estão contidas no código final, disponíveis na íntegra, no apêndice C

Por fim, em forma de comentário, é adicionado ao código final o comparativo entre todos os modelos processados, apresentando o nome do modelo, métrica de *score*, tempo de execução (em segundos), *score* apurado na base de treino, *score* apurado na base de teste e os hiper-parâmetros que apresentaram melhor performance. Na figura 10 pode ser observado, de forma parcial, o conteúdo desta etapa.

Figura 10: Código final Python - Comparativo de modelos

```
# Comparativo de Modelos:

#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| Modelo                    | Metrica_Score   |   Tempo_Exec(s) |   Score_Treino |   Score_Teste | Melhor_hiper_param
#+===========================+=================+================+================+===============+====================
#| DecisionTreeClassifier    | accuracy        |            9.49 |       0.845808 |      0.753363 | {'max_depth': 12, '
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| XGBClassifier             | accuracy        |            7.38 |       0.80988  |      0.753363 | {'colsample_bytree'
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| RandomForestClassifier    | accuracy        |           84.66 |       0.844311 |      0.762332 | {'bootstrap': False
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| GradientBoostingClassifier | accuracy        |           78.82 |       0.898204 |      0.789238 | {'learning_rate': 0
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| KNeighborsClassifier      | accuracy        |            1.09 |       0.782934 |      0.721973 | {'n_neighbors': 8,
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| LogisticRegression        | accuracy        |            0.3  |       0.823353 |      0.748879 | {'C': 0.1}
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| SVC                       | accuracy        |           13.23 |       0.932635 |      0.690583 | {'C': 10, 'degree':
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| DecisionTreeClassifier    | f1              |            2.17 |       0.785863 |      0.654088 | {'max_depth': 8, 'm
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| XGBClassifier             | f1              |            9.26 |       0.728051 |      0.645161 | {'colsample_bytree'
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
              ...                                      ...                              ...
              ...                                      ...                              ...
              ...                                      ...                              ...
              ...                                      ...                              ...
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| DecisionTreeClassifier    | roc_auc         |            2.64 |       0.826914 |      0.72904  | {'max_depth': 12, '
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| XGBClassifier             | roc_auc         |            8.18 |       0.787459 |      0.706566 | {'colsample_bytree'
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| RandomForestClassifier    | roc_auc         |           78.42 |       0.797834 |      0.736195 | {'bootstrap': False
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| GradientBoostingClassifier | roc_auc         |           83.74 |       0.830984 |      0.73447  | {'learning_rate': 0
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| KNeighborsClassifier      | roc_auc         |            0.68 |       0.739672 |      0.671465 | {'n_neighbors': 8,
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| LogisticRegression        | roc_auc         |            0.18 |       0.818907 |      0.740404 | {'C': 1}
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
#| SVC                       | roc_auc         |           13.28 |       0.92435  |      0.671254 | {'C': 10, 'degree':
#+---------------------------+-----------------+----------------+----------------+---------------+--------------------
```

Fonte: Desenvolvido pelo autor

A etapa responsável pela geração do conteúdo apresentado pela figura 10, tem como objetivo apresentar um comparativo que agregue valor ao estudo, pois de forma preliminar, o cientista de dados terá uma ideia inicial sobre a performance de cada modelo em relação a cada métrica de *score*, não tendo a necessidade de executar cada modelo manualmente.

Todo o código apresentado neste capítulo foi gerado de forma automática, sem nenhuma intervenção manual, sendo necessário apenas definir os parâmetros desejados no arquivo de configurações. O código final foi executado na íntegra, não ocorrendo erros de execução em nenhuma das etapas previstas no capítulo 3.

Os códigos finais gerados de forma automática para as demais bases estudadas neste estudo, apresentam uma estrutura similar aos códigos apresentados nos parágrafos anteriores. Um comparativo entre os códigos gerados e eficácia dos modelos para as diferentes bases, pode ser observada na tabela 6.

Tabela 6: Resultados de bases processadas

| **Base** | **Tempo geração do código** | **Métrica de *Score*** | **Melhor Modelo por Métrica de *Score*** | **Score Calculado** |
|---|---|---|---|---|
| TITANIC | 15 minutos | accuracy | RandomForestClassifier | 0.802691 |
| | | f1 | GradientBoostingClassifier | 0.696203 |
| | | precision | GradientBoostingClassifier | 0.956522 |
| | | recall | GradientBoostingClassifier | 0.647727 |
| | | roc_auc | LogisticRegression | 0.740404 |
| INVESTIMENTO | 5 horas e 25 minutos | accuracy | RandomForestClassifier | 0.910909 |
| | | f1 | RandomForestClassifier | 0.543342 |
| | | precision | GradientBoostingClassifier | 0.764706 |
| | | Recall | GradientBoostingClassifier | 0.502373 |
| | | roc_auc | GradientBoostingClassifier | 0.726684 |
| COMPRAS ONLINE | 13 horas e 7 minutos | accuracy | RandomForestClassifier | 0.907233 |
| | | f1 | DecisionTreeClassifier | 0.688153 |
| | | precision | GradientBoostingClassifier | 0.942857 |
| | | recall | DecisionTreeClassifier | 0.812757 |
| | | roc_auc | RandomForestClassifier | 0.782181 |

Fonte: Desenvolvido pelo autor

Os valores apresentados na tabela 6 se limitam aos melhores modelos para cada métrica de *score* para cada base, o *score* calculado se refere ao *score* aferido sobre a base de teste. Como pode ser observado, os valores de *score* variam bastante, dependendo da métrica para apuração do melhor modelo a ser escolhido pelo cientista de dados. O tempo necessário para geração do código final para cada base, foi obtido utilizando o poder computacional de um computador com as configurações: processador Intel i5-4310U CPU 2 GHz, 2601 Mhz, 2 Cores e 4 processadores lógicos; memória: 8GB DDR3.

O tempo de processamento necessário para processar todas as combinações parametrizadas para cada base, pode ser consideravelmente alto, a depender do volume de dados de cada base. Para ajustar as 35 combinações de modelos e métricas de *score* para a base TITANIC, foram necessários 15 minutos, porém para realizar o mesmo procedimento para a base INVESTIMENTO, foram necessárias 5 horas e 25 minutos. O volume de dados da

base TITANIC é de 892 registros, enquanto a base INVESTIMENTO é composta por 45.212 registros, por este motivo existe uma diferença considerável no tempo de processamento para cada uma delas.

O objetivo do cálculo combinado entre modelos e métricas de *score* é possibilitar uma visão abrangente para que o cientista de dados possa decidir qual será o ponto inicial de seus trabalhos, por exemplo, supondo que o estudo do cientista de dados fosse sobre a base INVESTIMENTO, a acurácia medida pelo modelo de *RandomForestClassifier* foi de 91%, o que a primeira vista parece ser um valor muito positivo, porém ao observar as demais métricas de *score* é possível entender que a acurácia não é uma métrica confiável para este caso, pois a distribuição de pessoas que optaram por fazer um investimento após a campanha de *marketing* apresentado por esta base é de apenas 11%, ou seja, case o modelo classifique todos os registros como classe “Não fará investimento”, ainda assim teria uma acurácia de 89%. Com estes dados previamente disponíveis para o cientista de dados, este profissional poderia iniciar os seus trabalhos com este entendimento e sendo assim desconsiderar a utilização de um determinado modelo ou métrica de *score*.

Desta forma, ao se utilizar esta solução de automatização de etapas de *machine learning* com geração de código fonte, o cientista de dados teria como ponto inicial para seus trabalhos, um código fonte base, com todos os comandos necessários para execução das etapas abordadas, os resultados das análises dos dados da base e o comparativo dos modelos em forma de comentários, possibilitando ganho de produtividade para este profissional.

# 5 CONCLUSÃO

Com o intuito de mensurar a eficácia da solução de automatização de *machine learning*, o resultado do melhor modelo sobre a base de teste foi comparado com os resultados em todas as submissões realizadas na plataforma *Kaggle*, para o desafio de *machine learning* sobre a mesma base de naufrágio do Titanic (*Titanic: Machine Learning from Disaster*).

A métrica de *score* utiliza pelo site *Kaggle* para o desafio no *Titanic: Machine Learning from Disaster* foi a acurácia, o melhor modelo de *machine learning* apresentado pela solução, de forma automática, foi o *RandomForestClassifier*, atingindo uma acurácia de 80,269 % sobre a base de testes.

No exato dia de 30 de outubro de 2019, momento em que tal mensuração foi realizada, existia um total de 27.079 submissões no desafio supracitado, o modelo de *machine learning* ajustado pela solução proposta por este estudo superou um total de 22.256 submissões da plataforma *Kaggle*, apresentando uma eficácia maior que 92,43% das soluções apresentadas para o desafio.

Analisando os resultados obtidos, a solução de automatização de processos de *machine learning* apresentou resultados positivos, pois de forma automática gerou um código python capaz de superar um percentual relativamente elevado de submissões realizadas por seres humanos.

Por se tratar de um estudo incipiente sobre o tema, cabe ressaltar as limitações da solução proposta pelo estudo: apenas problemas de classificação foram abordados; as regras utilizadas para seleção de variáveis foram pensadas na vivência de análise de dados do autor deste trabalho; solução comporta modelos de processamento não distribuído (sklearn); Não foi desenvolvido um módulo para automatização de *deploy*.

Como sugestão de continuidade de estudo, podem ser aprofundados e ampliados os pontos levantados como limitações do estudo, como automatização de soluções para *machine learning* de problemas com característica não classificatória, estudar diversos tipos de bases, para ajustar o mecanismo de enquadramento de variáveis e seus respectivos tratamentos. O estudo de automatização de etapas de *machine learning* poderia ser expandido para as bibliotecas de *machine learning* do Spark e processamento de grandes volumes de dados, pois como visto na tabela 6, ao se utilizar bases de dados em que os volumes não sejam pequenos, o tempo de processamento pode ser alto.

Posteriormente a etapa de adoção do modelo que apresentou o melhor resultado, ocorre a etapa de *deploy* do modelo de *machine learning* em ambiente produtivo, este

procedimento tem uma característica unicamente tecnológica e poderia ser automatizável, porém a depender do ambiente a ser aplicado o modelo. Esta etapa cabe em estudos futuros sobre o tema de automatização de *machine learning*.

De maneira geral, este estudo contribuiu para apresentar um ganho positivo no uso de soluções de automatização de *machine learning*, que mesmo de forma incipiente, se mostrou eficaz ao comparar seus resultados a modelos ajustados de forma tradicional. Atingindo ainda o objetivo de geração de código fonte para ser posteriormente trabalhado pelo cientista de dados, não tendo como objetivo apenas uma ferramenta de predição e sim uma solução para auxílio ao cientista de dados, possibilitando aumentar o desempenho para esse profissional.

# REFERÊNCIAS

A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python). Analytics Vidhya, 2016. Disponível em: <https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/>. Acesso em: 02 de nov. de 2019.

BEN FRAJ, Mohtadi. In Depth: Parameter tuning for Gradient Boosting. Medium, 2017. Disponível em: <https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae>. Acesso em: 02 de set. de 2019.

BEN FRAJ, Mohtadi. In Depth: Parameter tuning for KNN. Medium, 2017. Disponível em: <https://medium.com/@mohtedibf/in-depth-parameter-tuning-for-knn-4c0de485baf6>. Acesso em: 02 de set. de 2019.

BEN FRAJ, Mohtadi. In Depth: Parameter tuning for SVC. Medium, 2018. Disponível em: <https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769>. Acesso em: 02 de set. de 2019.

BEN FRAJ, Mohtadi. InDepth: Parameter tuning for Decision Tree. Medium, 2017. Disponível em: <https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3>. Acesso em: 02 de set. de 2019.

Big Data, O que é e qual sua importância? SAS, 2019. Disponível em: <https://www.sas.com/pt_br/insights/big-data/what-is-big-data.html>. Acesso em: 02 de set. de 2019.

Boosting (1): Árvores de Decisão e Gradient Boosting. Datalab Serasa Experian, 2019. Disponível em: <https://www.datalabserasaexperian.com.br/datalab/boosting-1-arvores-de-decisao-e-gradient-boosting/>. Acesso em: 02 de nov. de 2019.

BUSSAB, Wilton de O; MORETTIN, Pedro A. **Estatística Básica.** São Paulo: Saraiva, 2010. 557p.

Differences Between MapReduce And Apache Spark. EDUCBA, 2019. Disponível em: <https://www.educba.com/mapreduce-vs-apache-spark/>. Acesso em: 02 de set. de 2019.

FILHO, Othamar Gama. DATA-DRIVEN TALENT ACQUISITION IN THE GOOGLE AND FACEBOOK ERA. Climate17, 2019. Disponível em: <https://www.climate17.com/data-driven-talent-acquisition-in-the-google-and-facebook-era/>. Acesso em: 28 de out. de 2019.

FUJIMAKI, Ryohei. How will automation tools change data science?. Kdnuggets, 2019. Disponível em: <https://www.kdnuggets.com/2018/12/automation-data-science.html>. Acesso em: 02 de set. de 2019.

GRUS, Joel. **Data Science do Zero**. Rio de Janeiro: Alta Books, 2016. 336 p.

JAIN, Aarshay. Complete Guide to Parameter Tuning in XGBoost with codes in Python. Analytics Vidhya, 2016. Disponível em: <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>. Acesso em: 02 de set. de 2019.

KOEHRSEN, Will. Hyperparameter Tuning the Random Forest in Python. Towards Data Science, 2018. Disponível em: <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74>. Acesso em: 02 de set. de 2019.

KUHN, Max.; JOHNSON, Kjell. **Applied Predictive Modeling**. New York: Springer, 2013. 615 p.

MAYDON, Thomas. The 4 Types of Data Analytics. Kdnuggets, 2017. Disponível em: <https://www.kdnuggets.com/2017/07/4-types-data-analytics.html>. Acesso em: 02 de set. de 2019.

MISHRA, Aditya. Metrics to Evaluate your Machine Learning Algorithm. Towards Data Science, 2018. Disponível em: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>. Acesso em: 07 de nov. de 2019.

MITCHELL, Tom M. **Machine Learning**. New York: McGraw-Hill Science, 1997. 432 p.

QIAO, Finn. Logistic Regression Model Tuning with scikit-learn. Towards Data Science, 2019. Disponível em: <https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5>. Acesso em: 02 de set. de 2019.

REVERT, Félix. Fine-tuning XGBoost in Python like a boss. Towards Data Science, 2018. Disponível em: <https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e>. Acesso em: 02 de set. de 2019.

RODRIGUES, Vinícius. Entenda o que é AUC e ROC nos modelos de Machine Learning. Medium, 2018. Disponível em: <https://medium.com/bio-data-blog/entenda-o-que-%C3%A9-auc-e-roc-nos-modelos-de-machine-learning-8191fb4df772>. Acesso em: 07 de nov. de 2019.

ROLLINGS, Mike. Build a Data-Driven Organization. Gartner, 2019. Disponível em: <https://www.gartner.com/smarterwithgartner/build-a-data-driven-organization/>. Acesso em: 02 de set. de 2019.

SHARAN, MK. Bank Customers Survey - Marketing for Term Deposit. Kaggle, 2018. Disponível em: <https://www.kaggle.com/sharanmk/bank-marketing-term-deposit/metadata>. Acesso em: 02 de set. de 2019.

SHARMA, Roshan. Online Shopper's Intention Kaggle, 2019. Disponível em: <https://www.kaggle.com/roshansharma/online-shoppers-intention>. Acesso em: 02 de set. de 2019.

Titanic: Machine Learning from Disaster. Kaggle, 2012. Disponível em: <https://www.kaggle.com/c/titanic>. Acesso em: 02 de set. de 2019.

VALLEY, Mill. GLASSDOOR REVEALS THE 50 BEST JOBS IN AMERICA FOR 2019. Glassdoor, 2019. Disponível em: <https://www.glassdoor.com/about-us/bestjobs2019/>. Acesso em: 02 de set. de 2019.

# APÊNDICE A - Arquivo de configurações para processamento da base titanic

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
dados = 'C:/Users/rcosin/Desktop/FIA/TCC/Titanic/train.csv'
delimitador = ','
target = 'Survived'
separador_decimal = '.'
codigo_out = 'C:/Users/rcosin/Desktop/FIA/TCC/Titanic/ML_titanic.py'

score = ['accuracy', 'f1', 'precision', 'recall', 'roc_auc']

Modelo; parametros
DecisionTreeClassifier;{'max_depth':[2,4,8,12,16], 'min_samples_split':[0.1, 1.0, 10],
'min_samples_leaf':[0.1, 0.5, 5], 'max_features':[int(X_train.shape[1]*0.25),
int(X_train.shape[1]*0.50), int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
XGBClassifier;{'nthread':[4],'objective':['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8,9,10], 'min_child_weight': [12,20,30], 'silent': [1],'subsample':
[0.5,0.6,0.7], 'colsample_bytree': [0.6,0.7,0.8], 'n_estimators': [5,10,20] }
RandomForestClassifier;{'n_estimators':[10,50,100], 'max_depth':[2,8,16],
'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5, 5], 'bootstrap':[True,
False], 'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
GradientBoostingClassifier;{'learning rate':[1, 0.25, 0.05, 0.01], 'n estimators':[4, 16,
100], 'max_depth':[2,8,16], 'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5,
5], 'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])] }
KNeighborsClassifier;{'n_neighbors':[2,8,16,32], 'p':[1, 3, 5] }
LogisticRegression;{'C':[0.001,0.01,0.1,1,10,100] }
SVC;{'gamma':[0.05, 0.1, 1, 5, 10], 'C':[0.1, 1, 10, 100, 1000], 'degree':[0, 1, 2, 3, 5, 7] }
```

# APÊNDICE B - Solução de automatização de ml

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
# ## Lendo arquivo de Configurações


f = open("C:/Users/rcosin/Desktop/FIA/TCC/titanic.conf", "r")
ct = 0
model_param = ''

arq = f.readlines()
for l in arq:
    exec(l)
    ct = ct+1
    if ct == 6:
        ct = 0
        break

for l in arq:
    ct = ct+1
    if ct == 7:
        exec(l)
        ct = 0
        break

for l in arq:
    ct = ct+1
    if ct > 9:
        model_param = model_param + str(l)


# ## Funcoes de apoio


global codigo
codigo = ''

def gera_codigo(txt, ant, dep):
    global codigo
    codigo = codigo + '\n'*ant + txt + '\n'*dep

def gera_codigo_df(df, showidx, ant, dep):
    global codigo

    txt = tabulate(df, headers='keys', tablefmt='grid',showindex = showidx)
    txt = '#' + txt.replace('\n', '\n#')

    codigo = codigo + '\n'*ant + txt + '\n'*dep


global lmod_nome
global lmod_score
global lmod_tempo
global lmod_acc_treino
global lmod_acc_teste
global lmod_acc_hiperparam

lmod_nome = []
lmod_score = []
lmod_tempo = []
lmod_acc_treino = []
lmod_acc_teste = []
lmod_acc_hiperparam = []

def executa_modelo(Score, Modelo, parametros):

    global lmod_nome
    global lmod_score
    global lmod_tempo
    global lmod_acc_treino
```

```
    global lmod_acc_teste
    global lmod_acc_hiperparam

    print("Inicio: " + Modelo)

    DataA = datetime.datetime.now()

    gera_codigo("# Modelo: " + Modelo, 2, 1)

    cmd = Modelo +'()'
    modelo = eval(cmd)
    gera_codigo("modelo = " + cmd, 0, 1)

    gera_codigo("# Tunning: ", 1, 1)

    gera_codigo("parametros = " + str(parametros), 0, 1)

    mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4,  scoring=Score, verbose=0,
refit=True)
    gera_codigo("mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='"+ Score
+"', verbose=10, refit=True)", 0, 1)

    mod.fit(X_train, Y_train)
    gera_codigo("mod.fit(X_train, Y_train)", 0, 1)

    gera_codigo("# Melhores Parametros: " + str(mod.best_params_), 1, 1)

    gera_codigo("# Aplicando o Modelo", 1, 1)

    cmd = Modelo + '(**mod.best_params_)'
    model = eval(Modelo + '(**mod.best_params_)')
    gera_codigo("model = " + cmd, 0, 1)

    model.fit(X_train, Y_train)
    gera_codigo("model.fit(X_train, Y_train)", 0, 1)

    predict_train = model.predict(X_train)
    gera_codigo("predict_train = model.predict(X_train)", 0, 1)

    predict_test = model.predict(X_test)
    gera_codigo("predict_test = model.predict(X_test)", 0, 1)

    DataX = datetime.datetime.now() -DataA

    lmod_nome.append(Modelo)
    lmod_score.append(Score)
    lmod_acc_hiperparam.append(mod.best_params_)
    lmod_tempo.append(round(DataX.total_seconds(),2))

    gera_codigo("# Calculando o Score", 1, 1)

    if Score == 'accuracy':
        lmod_acc_treino.append(round(accuracy_score(Y_train, predict_train),6))
        lmod_acc_teste.append(round(accuracy_score(Y_test, predict_test),6))
        gera_codigo("score_treino = round(accuracy_score(Y_train, predict_train),6)", 0, 1)
        gera_codigo("score_teste = round(accuracy_score(Y_test, predict_test),6)", 0, 1)
    elif Score == 'f1':
        lmod_acc_treino.append(round(f1_score(Y_train, predict_train),6))
        lmod_acc_teste.append(round(f1_score(Y_test, predict_test),6))
        gera_codigo("score_treino = round(f1_score(Y_train, predict_train),6)", 0, 1)
        gera_codigo("score_teste = round(f1_score(Y_test, predict_test),6)", 0, 1)
    if Score == 'precision':
        lmod_acc_treino.append(round(precision_score(Y_train, predict_train),6))
        lmod_acc_teste.append(round(precision_score(Y_test, predict_test),6))
        gera_codigo("score_treino = round(precision_score(Y_train, predict_train),6)", 0, 1)
        gera_codigo("score_teste = round(precision_score(Y_test, predict_test),6)", 0, 1)
    if Score == 'recall':
        lmod_acc_treino.append(round(recall_score(Y_train, predict_train),6))
        lmod_acc_teste.append(round(recall_score(Y_test, predict_test),6))
        gera_codigo("score_treino = round(recall_score(Y_train, predict_train),6)", 0, 1)
        gera_codigo("score_teste = round(recall_score(Y_test, predict_test),6)", 0, 1)
    if Score == 'roc_auc':
        lmod_acc_treino.append(round(roc_auc_score(Y_train, predict_train),6))
        lmod_acc_teste.append(round(roc_auc_score(Y_test, predict_test),6))
        gera_codigo("score_treino = round(roc_auc_score(Y_train, predict_train),6)", 0, 1)
        gera_codigo("score_teste = round(roc_auc_score(Y_test, predict_test),6)", 0, 1)
```

```
    gera_codigo("# ------------------------------------------------", 1, 1)

    print("Finalizado após: " + str(round(DataX.total_seconds(),2)) + " segundos")
    print("--------------------")


# ## Importação de Modulos e Pacotes


import warnings
warnings.filterwarnings('ignore')

gera_codigo("# Importando Modulos e Pacotes", 0, 1)
import numpy as np
import pandas as pd
from tabulate import tabulate
import sklearn.model_selection as model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import datetime

gera_codigo("import numpy as np", 0, 1)
gera_codigo("import pandas as pd", 0, 1)
gera_codigo("from tabulate import tabulate", 0, 1)
gera_codigo("import sklearn.model_selection as model_selection", 0, 1)
gera_codigo("from sklearn.tree import DecisionTreeClassifier", 0, 1)
gera_codigo("from sklearn.linear_model import LogisticRegression", 0, 1)
gera_codigo("from sklearn.neighbors import KNeighborsClassifier", 0, 1)
gera_codigo("from sklearn.naive_bayes import GaussianNB", 0, 1)
gera_codigo("from sklearn.svm import SVC, LinearSVC", 0, 1)
gera_codigo("from sklearn.ensemble import RandomForestClassifier", 0, 1)
gera_codigo("from sklearn.ensemble import GradientBoostingClassifier", 0, 1)
gera_codigo("from sklearn.metrics import accuracy_score", 0, 1)
gera_codigo("from sklearn.metrics import roc_auc_score", 0, 1)
gera_codigo("from sklearn.metrics import f1_score", 0, 1)
gera_codigo("from sklearn.metrics import precision_score", 0, 1)
gera_codigo("from sklearn.metrics import recall_score", 0, 1)
gera_codigo("from sklearn.metrics import auc", 0, 1)
gera_codigo("from sklearn.model_selection import GridSearchCV", 0, 1)
gera_codigo("from xgboost import XGBClassifier", 0, 1)
gera_codigo("import datetime", 0, 2)


# ## Criando Data Frame inicial


gera_codigo("# Atribuindo parametros de entrada", 0, 1)

gera_codigo("target = '" + target +"'", 0, 2)

gera_codigo("# Importando os dados de entrada", 0, 1)

df = pd.read_csv(dados, sep = delimitador, decimal = separador_decimal)
gera_codigo("df = pd.read_csv('"+ dados +"', sep = '"+ delimitador \
            +"', decimal = '"+ separador_decimal +"')", 0, 2)


# ## Seleção de Variáveis e Identificação de tratamento


# Criacao de variaveis que serao utilizadas no Dataframe
# responsavel por consolidar todos os dados para
# selecao de variaveis
lcol = []
```

```
ltip = []
ldesvio = []
lmedia = []
lcoefvar = []

lmediana = []
lcoecentral = []

ldistper = []
ldistval = []
lnull = []

lacao = []
ldescacao = []

for x in df.columns :

    if x != target:

        lcol.append(x)

        cmd = 'df.' + x + '.dtype'
        ltip.append(str(eval(cmd)))

        if str(eval(cmd)) in ('float64','int64'):
            cmd = 'round( df.' + x + '.std() , 2)'
            ldesvio.append(str(eval(cmd)))

            cmd = 'round( df.' + x + '.mean() , 2)'
            lmedia.append(str(eval(cmd)))

            if abs(float(lmedia[-1])) == 0:
                cv = 0
            else:
                cv = round((float(ldesvio[-1]) / float(lmedia[-1])) * 100 ,2)

            lcoefvar.append(str(cv))

            cmd = 'round( df.' + x + '.median() , 2)'
            lmediana.append(str(eval(cmd)))

            if abs(float(lmedia[-1])) == 0:
                central = 0
            else:
                central = round((float(lmediana[-1]) / float(lmedia[-1])) * 100 ,2)

            lcoecentral.append(str(central))

        else:
            ldesvio.append('-')
            lmedia.append('-')
            lcoefvar.append('-')
            lmediana.append('-')
            lcoecentral.append('-')

        cmd = 'round( df.' + x + '.value_counts().count() / len(df.index) * 100 ,2)'
        ldistper.append(str(eval(cmd)))

        cmd = 'df.' + x + '.value_counts().count()'
        ldistval.append(str(eval(cmd)))

        cmd = 'round( df.' + x + '.isna().sum() / len(df.index) * 100 ,2)'
        lnull.append(str(eval(cmd)))

        if ltip[-1] in ('float64','int64') and float(ldistper[-1]) > 70 and \
            float(lcoefvar[-1]) > 50 and float(lcoecentral[-1]) > 80:
            lacao.append('excluir')
            ldescacao.append('alta dispersao')

        elif float(lnull[-1]) > 75:
            lacao.append('excluir')
            ldescacao.append('muito nulo')

        elif float(ldistval[-1]) < 30 and float(lnull[-1]) > 0 :
            lacao.append('dummy_com_null')
            ldescacao.append('-')
```

```
        elif float(ldistval[-1]) < 30 and float(lnull[-1]) == 0 :
            lacao.append('dummy_sem_null')
            ldescacao.append('-')

        elif ltip[-1] in ('float64','int64') and float(lnull[-1]) > 0 :
            lacao.append('continua_com_null')
            ldescacao.append('-')

        elif ltip[-1] in ('float64','int64') and float(lnull[-1]) == 0 :
            lacao.append('continua_sem_null')
            ldescacao.append('-')

        elif ltip[-1] in ('object') and float(ldistper[-1]) > 90:
            lacao.append('excluir')
            ldescacao.append('categorica com muitas categorias')

        elif ltip[-1] in ('object') :
            lacao.append('filtrar')
            ldescacao.append('possivel Feature Engineering')

        else:
            lacao.append('nao especificado')
            ldescacao.append('-')

lfinal =  list(zip(lcol, ltip, ldesvio, lmedia, lcoefvar, lmediana, lcoecentral, \
                   ldistper, ldistval, lnull, lacao, ldescacao))

df_Sel_Variaveis = pd.DataFrame(lfinal, columns = ['Coluna' , 'Tipo', 'Desvio Padrao', \
                                                   'Media', 'Coef de Variacao', \
                                                   'Mediana',  'Coef de Centralidade', \
                                                   'Perc_Distintos', 'Valores_Distintos', \
                                                   'Perc_Null', 'Acao', 'Desc Acao'])

gera_codigo("# Analise de variaveis e identificacao de tratamento:", 0, 1)
gera_codigo_df(df_Sel_Variaveis,False,0,2)

gera_codigo("# Correlacao entre as variaveis:", 0, 1)
gera_codigo_df(df.corr().round(3),True,0,2)


# ## Tratamento de Dados


# Criacao de variaveis que serao utilizadas no Dataframe
# responsavel por consolidar todos os dados para
# selecao de variaveis
lcol = []
ltip = []
ldesvio = []
lmedia = []
lcoefvar = []

lmediana = []
lcoecentral = []

ldistper = []
ldistval = []
lnull = []

lacao = []
ldescacao = []

for x in df.columns :

    if x != target:

        lcol.append(x)

        cmd = 'df.' + x + '.dtype'
        ltip.append(str(eval(cmd)))

        if str(eval(cmd)) in ('float64','int64'):
            cmd = 'round( df.' + x + '.std() , 2)'
            ldesvio.append(str(eval(cmd)))

            cmd = 'round( df.' + x + '.mean() , 2)'
            lmedia.append(str(eval(cmd)))
```

```
            if abs(float(lmedia[-1])) == 0:
                cv = 0
            else:
                cv = round((float(ldesvio[-1]) / float(lmedia[-1])) * 100 ,2)

            lcoefvar.append(str(cv))

            cmd = 'round( df.' + x + '.median() , 2)'
            lmediana.append(str(eval(cmd)))

            if abs(float(lmedia[-1])) == 0:
                central = 0
            else:
                central = round((float(lmediana[-1]) / float(lmedia[-1])) * 100 ,2)

            lcoecentral.append(str(central))

        else:
            ldesvio.append('-')
            lmedia.append('-')
            lcoefvar.append('-')
            lmediana.append('-')
            lcoecentral.append('-')

        cmd = 'round( df.' + x + '.value_counts().count() / len(df.index) * 100 ,2)'
        ldistper.append(str(eval(cmd)))

        cmd = 'df.' + x + '.value_counts().count()'
        ldistval.append(str(eval(cmd)))

        cmd = 'round( df.' + x + '.isna().sum() / len(df.index) * 100 ,2)'
        lnull.append(str(eval(cmd)))

        if ltip[-1] in ('float64','int64') and float(ldistper[-1]) > 70 and \
            float(lcoefvar[-1]) > 50 and float(lcoecentral[-1]) > 80:
            lacao.append('excluir')
            ldescacao.append('alta dispersao')

        elif float(lnull[-1]) > 75:
            lacao.append('excluir')
            ldescacao.append('muito nulo')

        elif float(ldistval[-1]) < 30 and float(lnull[-1]) > 0 :
            lacao.append('dummy_com_null')
            ldescacao.append('-')

        elif float(ldistval[-1]) < 30 and float(lnull[-1]) == 0 :
            lacao.append('dummy_sem_null')
            ldescacao.append('-')

        elif ltip[-1] in ('float64','int64') and float(lnull[-1]) > 0 :
            lacao.append('continua_com_null')
            ldescacao.append('-')

        elif ltip[-1] in ('float64','int64') and float(lnull[-1]) == 0 :
            lacao.append('continua_sem_null')
            ldescacao.append('-')

        elif ltip[-1] in ('object') and float(ldistper[-1]) > 90:
            lacao.append('excluir')
            ldescacao.append('categorica com muitas categorias')

        elif ltip[-1] in ('object') :
            lacao.append('filtrar')
            ldescacao.append('possivel Feature Engineering')

        else:
            lacao.append('nao especificado')
            ldescacao.append('-')

lfinal =  list(zip(lcol, ltip, ldesvio, lmedia, lcoefvar, lmediana, lcoecentral, \
                   ldistper, ldistval, lnull, lacao, ldescacao))

df_Sel_Variaveis = pd.DataFrame(lfinal, columns = ['Coluna' , 'Tipo', 'Desvio Padrao', \
                                                   'Media', 'Coef de Variacao', \
                                                   'Mediana',  'Coef de Centralidade', \
```

```
                                                    'Perc_Distintos', 'Valores_Distintos', \
                                                    'Perc_Null', 'Acao', 'Desc Acao'])

gera_codigo("# Analise de variaveis e identificacao de tratamento:", 0, 1)
gera_codigo_df(df_Sel_Variaveis,False,0,2)

gera_codigo("# Correlacao entre as variaveis:", 0, 1)
gera_codigo_df(df.corr().round(3),True,0,2)


# ## Separando bases de teste e treino


gera_codigo("# Dividindo a base entre variaveis explicativas e variavel resposta", 2, 1)
X = df2.loc[:, df2.columns != target]
Y = eval("df2." + target)
gera_codigo("X = df2.loc[:, df2.columns !='"+ target +"']", 0, 1)
gera_codigo("Y = df2."+ target , 0, 1)

gera_codigo("# Separando a Base entre Teste e Treino", 1, 1)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, train_size=0.75,
test_size=0.25, random_state=7)
gera_codigo("X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
train_size=0.75, test_size=0.25, random_state=7)", 0, 1)



# ## Aplicando e Ajustando Modelos


for s in score:

    print('--------\nScore: ' + s + '\n--------')

    for m in model_param.split('\n'):


        mod = str(m.split(';')[0])
        par = eval(m.split(';')[1])

        executa_modelo(s, mod , par)


# ## Comparativo de Modelos


gera_codigo("# Comparativo de Modelos: ",1,1)

l_modelos =  list(zip(lmod_nome, lmod_score, lmod_tempo, lmod_acc_treino, lmod_acc_teste,\
                      lmod acc hiperparam))
df_modelos = pd.DataFrame(l_modelos, columns = ['Modelo' , 'Metrica_Score', 'Tempo_Exec(s)',
'Score_Treino',\
                                                'Score_Teste', 'Melhor_hiper_param'])

gera_codigo_df(df_modelos,False,1,1)


# ## Gerando codigo.py


text_file = open(codigo_out, "w")
text_file.write(codigo)
text_file.close()
```

## APÊNDICE C - Código final gerado pela solução após processamento da base titanic

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
# Importando Modulos e Pacotes
import numpy as np
import pandas as pd
from tabulate import tabulate
import sklearn.model_selection as model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import datetime

# Atribuindo parametros de entrada
target = 'Survived'

# Importando os dados de entrada
df = pd.read_csv('C:/Users/rcosin/Desktop/FIA/TCC/Titanic/train.csv', sep = ',', decimal =
'.')

# Analise de variaveis e identificacao de tratamento:
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
-----------------+------------------+---------------------+-------------+--------------------+-
---------------------------------+
#| Coluna      | Tipo    | Desvio Padrao   | Media   | Coef de Variacao   | Mediana   | Coef
de Centralidade   |   Perc_Distintos |   Valores_Distintos |   Perc_Null | Acao               |
Desc Acao                       |
#+=============+=========+=================+=========+====================+===========+=======
=================+==================+=====================+=============+====================+=
=================================+
#| PassengerId | int64   | 257.35          | 446.0   | 57.7               | 446.0     | 100.0
|           100    |                891 |        0    | excluir            | alta dispersao
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
-----------------+------------------+---------------------+-------------+--------------------+-
---------------------------------+
#| Pclass      | int64   | 0.84            | 2.31    | 36.36              | 3.0       | 129.87
|             0.34 |                  3 |        0    | dummy_sem_null     | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
-----------------+------------------+---------------------+-------------+--------------------+-
---------------------------------+
#| Name        | object  | -               | -       | -                  | -         | -
|           100    |                891 |        0    | excluir            | categorica com
muitas categorias |
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
-----------------+------------------+---------------------+-------------+--------------------+-
---------------------------------+
#| Sex         | object  | -               | -       | -                  | -         | -
|             0.22 |                  2 |        0    | dummy_sem_null     | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
-----------------+------------------+---------------------+-------------+--------------------+-
---------------------------------+
#| Age         | float64 | 14.53           | 29.7    | 48.92              | 28.0      | 94.28
|             9.88 |                 88 |       19.87 | continua_com_null | -
|
```

```
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| SibSp       | int64   | 1.1             | 0.52    | 211.54             | 0.0       | 0.0
|             0.79 |                   7 |        0    | dummy_sem_null    | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| Parch       | int64   | 0.81            | 0.38    | 213.16             | 0.0       | 0.0
|             0.79 |                   7 |        0    | dummy_sem_null    | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| Ticket      | object  | -               | -       | -                  | -         | -
|            76.43 |                 681 |        0    | filtrar           | possivel Feature
Engineering     |
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| Fare        | float64 | 49.69           | 32.2    | 154.32             | 14.45     | 44.88
|            27.83 |                 248 |        0    | continua_sem_null | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| Cabin       | object  | -               | -       | -                  | -         | -
|            16.5  |                 147 |       77.1  | excluir           | muito nulo
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+
#| Embarked    | object  | -               | -       | -                  | -         | -
|             0.34 |                   3 |        0.22 | dummy_com_null    | -
|
#+-------------+---------+-----------------+---------+--------------------+-----------+-------
----------------+------------------+---------------------+-------------+-------------------+-
---------------------------------+

# Correlacao entre as variaveis:
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#|             |   PassengerId |   Survived |   Pclass |    Age |   SibSp |   Parch |   Fare |
#+=============+===============+============+==========+========+=========+=========+========+
#| PassengerId |         1     |     -0.005 |   -0.035 |  0.037 |  -0.058 |  -0.002 |  0.013 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Survived    |        -0.005 |      1     |   -0.338 | -0.077 |  -0.035 |   0.082 |  0.257 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Pclass      |        -0.035 |     -0.338 |    1     | -0.369 |   0.083 |   0.018 | -0.549 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Age         |         0.037 |     -0.077 |   -0.369 |  1     |  -0.308 |  -0.189 |  0.096 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| SibSp       |        -0.058 |     -0.035 |    0.083 | -0.308 |   1     |   0.415 |  0.16  |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Parch       |        -0.002 |      0.082 |    0.018 | -0.189 |   0.415 |   1     |  0.216 |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+
#| Fare        |         0.013 |      0.257 |   -0.549 |  0.096 |   0.16  |   0.216 |  1     |
#+-------------+---------------+------------+----------+--------+---------+---------+--------+


# Tratamento de Dados
df2 = pd.concat([df['Survived']])

# Criando Dataframe auxiliar
df_temp = pd.DataFrame()

# Adicionando colunas continuas sem nulos
df2 = pd.concat([df['Fare'], df2], axis=1)

# Adicionando colunas continuas com nulos
df_temp = df.Age.fillna( df.Age.mean() )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies sem tratamento de NULL
df_temp = pd.get_dummies( df.Pclass, prefix='Pclass' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Sex, prefix='Sex' )
```

```
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.SibSp, prefix='SibSp' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Parch, prefix='Parch' )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies com tratamento de NULL
df_temp = df.Embarked.fillna( 'NULL' )
df_temp = pd.get_dummies( df_temp, prefix='Embarked' )
df2 = pd.concat([df_temp, df2], axis=1)


# Dividindo a base entre variaveis explicativas e variavel resposta
X = df2.loc[:, df2.columns !='Survived']
Y = df2.Survived

# Separando a Base entre Teste e Treino
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, train_size=0.75,
test_size=0.25, random_state=7)


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 12, 'max_features': 18, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()
```

```
# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 12,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 8, 'max_features': 18,
'min_samples_leaf': 0.1, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
```

```
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 8, 'max_features': 18, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
```

```
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': True, 'max_depth': 8, 'max_features': 18,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score treino = round(f1 score(Y train, predict train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 6,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()
```

```
# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)
```

```
# Melhores Parametros: {'max_depth': 4, 'max_features': 12, 'min_samples_leaf': 0.1,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n estimators': [10, 50, 100], 'max depth': [2, 8, 16], 'min samples split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 6,
'min_samples_leaf': 5, 'min_samples_split': 1.0, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
```

```
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 2, 'max_features': 25,
'min_samples_leaf': 0.1, 'min_samples_split': 0.1, 'n_estimators': 16}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.01}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
```

```
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1, 'degree': 0, 'gamma': 5}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 4, 'max_features': 6, 'min_samples_leaf': 0.1,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 5, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
```

```
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 8, 'max_features': 25,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 18,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 16}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()
```

```
# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 12, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
```

```
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 6,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning rate': [1, 0.25, 0.05, 0.01], 'n estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [6, 12, 18, 25]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 2, 'max_features': 18,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------
```

```
# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ---------------------------------------------

# Comparativo de Modelos:

#+---------------------------+----------------+----------------+---------------+----------
-----+-------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------
-----+
#| Modelo                      | Metrica_Score   |   Tempo_Exec(s) |   Score_Treino |
Score_Teste | Melhor_hiper_param
|
#+=============================+=================+=================+================+==========
=====+=========================================================================================
=========================================================================================
=====+
#| DecisionTreeClassifier     | accuracy        |            9.49 |       0.845808 |
0.753363 | {'max_depth': 12, 'max_features': 18, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | accuracy        |            7.38 |       0.80988  |
0.753363 | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6} |
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | accuracy        |           84.66 |       0.844311 |
0.802691 | {'bootstrap': True, 'max_depth': 16, 'max_features': 12, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 10}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | accuracy        |           78.82 |       0.898204 |
0.789238 | {'learning_rate': 0.25, 'max_depth': 8, 'max_features': 18, 'min_samples_leaf':
0.1, 'min_samples_split': 10, 'n_estimators': 100}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | accuracy        |            1.09 |       0.782934 |
0.721973 | {'n_neighbors': 8, 'p': 1}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | accuracy        |            0.3  |       0.823353 |
0.748879 | {'C': 0.1}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | accuracy        |           13.23 |       0.932635 |
0.690583 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | f1              |            2.17 |       0.785863 |
0.654088 | {'max_depth': 8, 'max_features': 18, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | f1              |            9.26 |       0.728051 |
0.645161 | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6} |
#+-----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | f1              |           79.82 |       0.779874 |
0.675    | {'bootstrap': True, 'max_depth': 8, 'max_features': 18, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 10}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | f1              |           80.08 |       0.852391 |
0.696203 | {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 6, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | f1              |            1.07 |       0.662005 |
0.550725 | {'n_neighbors': 8, 'p': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | f1              |            0.21 |       0.757202 |
0.658537 | {'C': 0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | f1              |           14.47 |       0.909457 |
0.596491 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | precision       |            2.18 |       0.790055 |
0.775862 | {'max_depth': 4, 'max_features': 12, 'min_samples_leaf': 0.1, 'min_samples_split':
0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | precision       |            8.17 |       0.798122 |
0.746269 | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min child weight': 12, 'n estimators': 10, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.6} |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | precision       |           77.9  |       0.975    |
0.933333 | {'bootstrap': False, 'max_depth': 16, 'max_features': 6, 'min_samples_leaf': 5,
'min_samples_split': 1.0, 'n_estimators': 10}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | precision       |           86.42 |       0.943548 |
0.956522 | {'learning_rate': 0.05, 'max_depth': 2, 'max_features': 25, 'min_samples_leaf':
0.1, 'min_samples_split': 0.1, 'n_estimators': 16}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | precision       |            1.04 |       0.811429 |      0.76
| {'n_neighbors': 8, 'p': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | precision       |            0.17 |       0.801325 |
0.877551 | {'C': 0.01}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| SVC                        | precision       |           13.68 |       0.995868 |
0.411765 | {'C': 1, 'degree': 0, 'gamma': 5}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | recall          |            2.45 |       0.255906 |
0.272727 | {'max_depth': 4, 'max_features': 6, 'min_samples_leaf': 0.1, 'min_samples_split':
0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | recall          |            8.15 |       0.606299 |
0.545455 | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 5, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5}  |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | recall          |           78.14 |       0.748031 |
0.613636 | {'bootstrap': False, 'max_depth': 8, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 10}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | recall          |           83.62 |       0.866142 |
0.647727 | {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 18, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 16}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | recall          |            1.41 |       0.559055 |
0.431818 | {'n_neighbors': 8, 'p': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | recall          |            0.25 |       0.724409 |
0.613636 | {'C': 0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| SVC                        | recall          |           13.2  |       0.889764 |
0.579545 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | roc_auc         |            2.64 |       0.826914 |
0.72904  | {'max_depth': 12, 'max_features': 25, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+
```

```
#| XGBClassifier              | roc_auc         |            8.18 |       0.787459 |
0.706566 | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | roc_auc         |           78.42 |       0.797834 |
0.736195 | {'bootstrap': False, 'max_depth': 16, 'max_features': 6, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 10}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | roc_auc         |           83.74 |       0.830984 |
0.73447  | {'learning_rate': 0.05, 'max_depth': 2, 'max_features': 18, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | roc_auc         |            0.68 |       0.739672 |
0.671465 | {'n_neighbors': 8, 'p': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | roc_auc         |            0.18 |       0.818907 |
0.740404 | {'C': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | roc_auc         |           13.28 |       0.92435  |
0.671254 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
```

# APÊNDICE D - Arquivo de configurações para processamento da base investimento

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
dados = 'C:/Users/Rodrigo/Desktop/TCC/bank_customer_survey/bank_customer_survey.csv'
delimitador = ','
target = 'y'
separador_decimal = '.'
codigo_out = 'C:/Users/Rodrigo/Desktop/TCC/bank_customer_survey/ML_bank_customer_survey.py'

score = ['accuracy', 'f1', 'precision', 'recall', 'roc_auc']

Modelo; parametros
DecisionTreeClassifier;{'max_depth':[2,4,8,12,16], 'min_samples_split':[0.1, 1.0, 10],
'min_samples_leaf':[0.1, 0.5, 5], 'max_features':[int(X_train.shape[1]*0.25),
int(X_train.shape[1]*0.50), int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
XGBClassifier;{'nthread':[4],'objective':['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8,9,10], 'min_child_weight': [12,20,30], 'silent': [1],'subsample':
[0.5,0.6,0.7], 'colsample_bytree': [0.6,0.7,0.8], 'n_estimators': [5,10,20] }
RandomForestClassifier;{'n_estimators':[10,50,100], 'max_depth':[2,8,16],
'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5, 5], 'bootstrap':[True,
False], 'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
GradientBoostingClassifier;{'learning rate':[1, 0.25, 0.01], 'n estimators':[4, 30],
'max_depth':[2,8,16], 'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5, 5],
'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])] }
KNeighborsClassifier;{'n_neighbors':[2,8,16,32], 'p':[1, 3, 5] }
LogisticRegression;{'C':[0.001,0.01,0.1,1,10,100] }
```

# APÊNDICE E - Código final gerado pela solução após processamento da base investimento

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
# Importando Modulos e Pacotes
import numpy as np
import pandas as pd
from tabulate import tabulate
import sklearn.model_selection as model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc auc score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import datetime

# Atribuindo parametros de entrada
target = 'y'

# Importando os dados de entrada
df = pd.read_csv('C:/Users/Rodrigo/Desktop/TCC/bank_customer_survey/bank_customer_survey.csv',
sep = ',', decimal = '.')

# Analise de variaveis e identificacao de tratamento:
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
#| Coluna    | Tipo   | Desvio Padrao   | Media   | Coef de Variacao   | Mediana   | Coef de
Centralidade   |   Perc_Distintos |   Valores_Distintos |   Perc_Null | Acao               |
Desc Acao   |
#+===========+========+=================+=========+====================+===========+==========
==============+==================+=====================+=============+====================+====
=========+
#| age       | int64  | 10.62           | 40.94   | 25.94              | 39.0      | 95.26
|             0.17 |                77 |           0 | continua_sem_null | -          |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
#| job       | object | -               | -       | -                  | -         | -
|             0.03 |                12 |           0 | dummy_sem_null    | -          |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
#| marital   | object | -               | -       | -                  | -         | -
|             0.01 |                 3 |           0 | dummy_sem_null    | -          |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
#| education | object | -               | -       | -                  | -         | -
|             0.01 |                 4 |           0 | dummy_sem_null    | -          |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
#| default   | object | -               | -       | -                  | -         | -
|             0    |                 2 |           0 | dummy_sem_null    | -          |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
--------------+------------------+---------------------+-------------+--------------------+----
---------+
```

```
#| balance   | int64  | 3044.77         | 1362.27 | 223.51             | 448.0     | 32.89
|           15.85 |                7168 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| housing   | object | -               | -       | -                  | -         | -
|            0    |                   2 |           0 | dummy_sem_null    | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| loan      | object | -               | -       | -                  | -         | -
|            0    |                   2 |           0 | dummy_sem_null    | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| contact   | object | -               | -       | -                  | -         | -
|            0.01 |                   3 |           0 | dummy_sem_null    | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| day       | int64  | 8.32            | 15.81   | 52.62              | 16.0      | 101.2
|            0.07 |                  31 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| month     | object | -               | -       | -                  | -         | -
|            0.03 |                  12 |           0 | dummy_sem_null    | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| duration  | int64  | 257.53          | 258.16  | 99.76              | 180.0     | 69.72
|            3.48 |                1573 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| campaign  | int64  | 3.1             | 2.76    | 112.32             | 2.0       | 72.46
|            0.11 |                  48 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| pdays     | int64  | 100.13          | 40.2    | 249.08             | -1.0      | -2.49
|            1.24 |                 559 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| previous  | int64  | 2.3             | 0.58    | 396.55             | 0.0       | 0.0
|            0.09 |                  41 |           0 | continua_sem_null | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+
#| poutcome  | object | -               | -       | -                  | -         | -
|            0.01 |                   4 |           0 | dummy_sem_null    | -           |
#+-----------+--------+-----------------+---------+--------------------+-----------+----------
-------------+------------------+---------------------+-------------+------------------+----
---------+

# Correlacao entre as variaveis:
#+----------+--------+-----------+--------+------------+------------+---------+------------+--
------+
#|          |    age |   balance |    day |   duration |   campaign |   pdays |   previous |
y |
#+==========+========+===========+========+============+============+=========+============+==
======+
#| age      |  1     |     0.098 | -0.009 |     -0.005 |      0.005 |  -0.024 |      0.001 |
0.025 |
#+----------+--------+-----------+--------+------------+------------+---------+------------+--
------+
#| balance  |  0.098 |     1     |  0.005 |      0.022 |     -0.015 |   0.003 |      0.017 |
0.053 |
#+----------+--------+-----------+--------+------------+------------+---------+------------+--
------+
#| day      | -0.009 |     0.005 |  1     |     -0.03  |      0.162 |  -0.093 |     -0.052 | -
0.028 |
#+----------+--------+-----------+--------+------------+------------+---------+------------+--
------+
#| duration | -0.005 |     0.022 | -0.03  |      1     |     -0.085 |  -0.002 |      0.001 |
0.395 |
```

```
#+----------+--------+----------+--------+-----------+-----------+---------+-----------+--
------+
#| campaign |  0.005 |    -0.015 |  0.162 |    -0.085 |      1    |  -0.089 |    -0.033 | -
0.073 |
#+----------+--------+----------+--------+-----------+-----------+---------+-----------+--
------+
#| pdays    | -0.024 |     0.003 | -0.093 |    -0.002 |    -0.089 |   1     |     0.455 |
0.104 |
#+----------+--------+----------+--------+-----------+-----------+---------+-----------+--
------+
#| previous |  0.001 |     0.017 | -0.052 |     0.001 |    -0.033 |   0.455 |     1     |
0.093 |
#+----------+--------+----------+--------+-----------+-----------+---------+-----------+--
------+
#| y        |  0.025 |     0.053 | -0.028 |     0.395 |    -0.073 |   0.104 |     0.093 |
1     |
#+----------+--------+----------+--------+-----------+-----------+---------+-----------+--
------+


# Tratamento de Dados
df2 = pd.concat([df['y']])

# Criando Dataframe auxiliar
df_temp = pd.DataFrame()

# Adicionando colunas continuas sem nulos
df2 = pd.concat([df['age'], df2], axis=1)
df2 = pd.concat([df['balance'], df2], axis=1)
df2 = pd.concat([df['day'], df2], axis=1)
df2 = pd.concat([df['duration'], df2], axis=1)
df2 = pd.concat([df['campaign'], df2], axis=1)
df2 = pd.concat([df['pdays'], df2], axis=1)
df2 = pd.concat([df['previous'], df2], axis=1)

# Adicionando colunas continuas com nulos

# Adicionando e preparando colunas com dummies sem tratamento de NULL
df_temp = pd.get_dummies( df.job, prefix='job' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.marital, prefix='marital' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.education, prefix='education' )
df2 = pd.concat([df temp, df2], axis=1)
df_temp = pd.get_dummies( df.default, prefix='default' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.housing, prefix='housing' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.loan, prefix='loan' )
df2 = pd.concat([df temp, df2], axis=1)
df_temp = pd.get_dummies( df.contact, prefix='contact' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.month, prefix='month' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.poutcome, prefix='poutcome' )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies com tratamento de NULL


# Dividindo a base entre variaveis explicativas e variavel resposta
X = df2.loc[:, df2.columns !='y']
Y = df2.y

# Separando a Base entre Teste e Treino
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, train_size=0.75,
test_size=0.25, random_state=7)


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
```

```
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 8, 'max_features': 51, 'min_samples_leaf': 5,
'min_samples_split': 10}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 12,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
```

```
parametros = {'learning_rate': [1, 0.25, 0.01], 'n_estimators': [4, 30], 'max_depth': [2, 8,
16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 12,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 30}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 32, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [12, 25, 38, 51]}
```

```
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 12, 'max_features': 38, 'min_samples_leaf': 5,
'min_samples_split': 10}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': True, 'max_depth': 16, 'max_features': 51,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 50}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
```

```
parametros = {'learning_rate': [1, 0.25, 0.01], 'n_estimators': [4, 30], 'max_depth': [2, 8,
16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 8, 'max_features': 25,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 30}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)
```

```
# Melhores Parametros: {'max_depth': 2, 'max_features': 38, 'min_samples_leaf': 5,
'min_samples_split': 10}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ----------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 20, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ----------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n estimators': [10, 50, 100], 'max depth': [2, 8, 16], 'min samples split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 2, 'max_features': 12,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ----------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
```

```
parametros = {'learning_rate': [1, 0.25, 0.01], 'n_estimators': [4, 30], 'max_depth': [2, 8,
16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 2, 'max_features': 38,
'min_samples_leaf': 5, 'min_samples_split': 1.0, 'n_estimators': 4}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 32, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.01}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [12, 25, 38, 51]}
```

```
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 16, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 10}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 25,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()
```

```
# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.01], 'n_estimators': [4, 30], 'max_depth': [2, 8,
16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 1, 'max_depth': 16, 'max_features': 25,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 30}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
```

```
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 16, 'max_features': 51, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': True, 'max_depth': 16, 'max_features': 12,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------
```

```
# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.01], 'n_estimators': [4, 30], 'max_depth': [2, 8,
16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'max_features':
[12, 25, 38, 51]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 25,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 30}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 32, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------

# Comparativo de Modelos:
```

```
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| Modelo                    | Metrica_Score   |   Tempo_Exec(s) |   Score_Treino |
Score_Teste | Melhor_hiper_param
|
#+===========================+=================+=================+================+==========
=====+=========================================================================================
===============================================================================================
======+
#| DecisionTreeClassifier    | accuracy        |          147.06 |       0.914121 |
0.9033   | {'max_depth': 8, 'max_features': 51, 'min_samples_leaf': 5, 'min_samples_split':
10}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| XGBClassifier             | accuracy        |         1130.71 |       0.906541 |
0.905246 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5} |
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| RandomForestClassifier    | accuracy        |         1647.49 |       0.95939  |
0.910909 | {'bootstrap': False, 'max_depth': 16, 'max_features': 12, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| GradientBoostingClassifier | accuracy        |         1179.07 |       0.918338 |
0.910201 | {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 12, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 30}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| KNeighborsClassifier      | accuracy        |          161.14 |       0.890114 |
0.895249 | {'n_neighbors': 32, 'p': 5}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| LogisticRegression        | accuracy        |           10.9  |       0.899906 |
0.907724 | {'C': 1}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| DecisionTreeClassifier    | f1              |           29.16 |       0.666196 |
0.471384 | {'max_depth': 12, 'max_features': 38, 'min_samples_leaf': 5, 'min_samples_split':
10}
|
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| XGBClassifier             | f1              |          751.78 |       0.4952   |
0.421452 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+---------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------+
#| RandomForestClassifier    | f1              |         1671.34 |       0.792691 |
0.543342 | {'bootstrap': True, 'max_depth': 16, 'max_features': 51, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 50}
|
```

```
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| GradientBoostingClassifier | f1              |         1206.35 |       0.778455 |
0.542643 | {'learning_rate': 0.25, 'max_depth': 8, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 30}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| KNeighborsClassifier       | f1              |          151.45 |       0.38512  |
0.282876 | {'n_neighbors': 8, 'p': 1}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| LogisticRegression         | f1              |           10.84 |       0.447771 |
0.467586 | {'C': 1}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| DecisionTreeClassifier     | precision       |           28.62 |       0.653612 |
0.627072 | {'max_depth': 2, 'max_features': 38, 'min_samples_leaf': 5, 'min_samples_split':
10}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| XGBClassifier              | precision       |          750.57 |       0.781785 |
0.71875  | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 20, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| RandomForestClassifier     | precision       |         1690.59 |       0.848485 |
0.666667 | {'bootstrap': False, 'max depth': 2, 'max_features': 12, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| GradientBoostingClassifier | precision       |         1193.19 |       0.822086 |
0.764706 | {'learning_rate': 0.25, 'max_depth': 2, 'max_features': 38, 'min_samples_leaf': 5,
'min_samples_split': 1.0, 'n_estimators': 4}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| KNeighborsClassifier       | precision       |          161.41 |       0.612491 |
0.596618 | {'n_neighbors': 32, 'p': 5}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| LogisticRegression         | precision       |           10.52 |       0.654853 |
0.665568 | {'C': 0.01}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| DecisionTreeClassifier     | recall          |           27.05 |       0.675528 |
0.486551 | {'max_depth': 16, 'max_features': 25, 'min_samples_leaf': 5, 'min_samples_split':
10}
|
```

```
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| XGBClassifier             | recall         |          759.83 |       0.365217 |
0.307753 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| RandomForestClassifier    | recall         |         1658.37 |       0.768199 |
0.47943  | {'bootstrap': False, 'max_depth': 16, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| GradientBoostingClassifier | recall         |         1190.39 |       0.622857 |
0.502373 | {'learning_rate': 1, 'max_depth': 16, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 30}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| KNeighborsClassifier      | recall         |          155.94 |       0.261366 |
0.205696 | {'n_neighbors': 8, 'p': 5}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| LogisticRegression        | recall         |           11.93 |       0.341863 |
0.362342 | {'C': 1}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| DecisionTreeClassifier    | roc_auc        |           28.48 |       0.703954 |
0.700603 | {'max_depth': 16, 'max_features': 51, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| XGBClassifier             | roc_auc        |          760.96 |       0.675213 |
0.644264 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 10,
'min_child_weight': 12, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| RandomForestClassifier    | roc_auc        |         1650.14 |       0.798659 |
0.698223 | {'bootstrap': True, 'max_depth': 16, 'max_features': 12, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| GradientBoostingClassifier | roc_auc        |         1191.08 |       0.763064 |
0.726684 | {'learning_rate': 0.25, 'max_depth': 16, 'max_features': 25, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 30}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| KNeighborsClassifier      | roc_auc        |          129.29 |       0.585165 |
0.585637 | {'n_neighbors': 32, 'p': 1}
|
```

```
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
#| LogisticRegression         | roc_auc         |           10.77 |       0.655175 |
0.665564 | {'C': 0.1}
|
#+---------------------------+----------------+----------------+---------------+----------
-----+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------+
```

# APÊNDICE F - Arquivo de configurações para processamento da base compras online

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
dados =
'C:/Users/rcosin/Desktop/FIA/TCC/online_shoppers_intention/online_shoppers_intention.csv'
delimitador = ','
target = 'Revenue'
separador_decimal = '.'
codigo_out =
'C:/Users/rcosin/Desktop/FIA/TCC/online_shoppers_intention/ML_online_shoppers_intention.py'

score = ['accuracy', 'f1', 'precision', 'recall', 'roc_auc']

Modelo; parametros
DecisionTreeClassifier;{'max_depth':[2,4,8,12,16], 'min_samples_split':[0.1, 1.0, 10],
'min_samples_leaf':[0.1, 0.5, 5], 'max_features':[int(X_train.shape[1]*0.25),
int(X_train.shape[1]*0.50), int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
XGBClassifier;{'nthread':[4],'objective':['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8,9,10], 'min_child_weight': [12,20,30], 'silent': [1],'subsample':
[0.5,0.6,0.7], 'colsample_bytree': [0.6,0.7,0.8], 'n_estimators': [5,10,20] }
RandomForestClassifier;{'n_estimators':[10,50,100], 'max_depth':[2,8,16],
'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5, 5], 'bootstrap':[True,
False], 'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])]}
GradientBoostingClassifier;{'learning_rate':[1, 0.25, 0.05, 0.01], 'n_estimators':[4, 16,
100], 'max_depth':[2,8,16], 'min_samples_split':[0.1, 1.0, 10], 'min_samples_leaf':[0.1, 0.5,
5], 'max_features':[int(X_train.shape[1]*0.25), int(X_train.shape[1]*0.50),
int(X_train.shape[1]*0.75), int(X_train.shape[1])] }
KNeighborsClassifier;{'n_neighbors':[2,8,16,32], 'p':[1, 3, 5] }
LogisticRegression;{'C':[0.001,0.01,0.1,1,10,100] }
SVC;{'gamma':[0.05, 0.1, 1, 5, 10], 'C':[0.1, 1, 10, 100, 1000], 'degree':[0, 1, 2, 3, 5, 7] }
```

## APÊNDICE G - Código final gerado pela solução após processamento da base compras online

Código disponível na íntegra em: https://github.com/rodrigocosin/ML_automator.git

```
# Importando Modulos e Pacotes
import numpy as np
import pandas as pd
from tabulate import tabulate
import sklearn.model_selection as model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc auc score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import datetime

# Atribuindo parametros de entrada
target = 'Revenue'

# Importando os dados de entrada
df =
pd.read_csv('C:/Users/rcosin/Desktop/FIA/TCC/online_shoppers_intention/online_shoppers_intenti
on.csv', sep = ',', decimal = '.')

# Analise de variaveis e identificacao de tratamento:
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
----------+-------------+
#| Coluna                  | Tipo    | Desvio Padrao   | Media   | Coef de Variacao   |
Mediana   | Coef de Centralidade   |   Perc_Distintos |   Valores_Distintos |   Perc_Null |
Acao              | Desc Acao   |
#+=========================+=========+=================+=========+====================+=======
====+=========================+==================+======================+=============+=========
==========+=============+
#| Administrative          | float64 | 3.32            | 2.32    | 143.1              | 1.0
| 43.1                  |            0.22 |                  27 |        0.11 |
dummy_com_null    | -           |
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
----------+-------------+
#| Administrative_Duration | float64 | 176.86          | 80.91   | 218.59             | 8.0
| 9.89                  |           27.06 |                3336 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
----------+-------------+
#| Informational           | float64 | 1.27            | 0.5     | 254.0              | 0.0
| 0.0                   |            0.14 |                  17 |        0.11 |
dummy_com_null    | -           |
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
----------+-------------+
#| Informational_Duration  | float64 | 140.83          | 34.51   | 408.08             | 0.0
| 0.0                   |           10.21 |                1259 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
----------+-------------+
```

```
#| ProductRelated          | float64 | 44.49           | 31.76   | 140.08            | 18.0
| 56.68                  |            2.52 |                311 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| ProductRelated_Duration | float64 | 1914.37         | 1196.04 | 160.06            | 599.77
| 50.15                  |            77.47 |               9552 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| BounceRates             | float64 | 0.05            | 0.02    | 250.0             | 0.0
| 0.0                    |            15.18 |               1872 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| ExitRates               | float64 | 0.05            | 0.04    | 125.0             | 0.03
| 75.0                   |            38.74 |               4777 |        0.11 |
continua_com_null | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| PageValues              | float64 | 18.57           | 5.89    | 315.28            | 0.0
| 0.0                    |            21.93 |               2704 |        0    |
continua_sem_null | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| SpecialDay              | float64 | 0.2             | 0.06    | 333.33            | 0.0
| 0.0                    |             0.05 |                  6 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| Month                   | object  | -               | -       | -                 | -
| -                      |             0.08 |                 10 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| OperatingSystems        | int64   | 0.91            | 2.12    | 42.92             | 2.0
| 94.34                  |             0.06 |                  8 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| Browser                 | int64   | 1.72            | 2.36    | 72.88             | 2.0
| 84.75                  |             0.11 |                 13 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| Region                  | int64   | 2.4             | 3.15    | 76.19             | 3.0
| 95.24                  |             0.07 |                  9 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| TrafficType             | int64   | 4.03            | 4.07    | 99.02             | 2.0
| 49.14                  |             0.16 |                 20 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| VisitorType             | object  | -               | -       | -                 | -
| -                      |             0.02 |                  3 |        0    |
dummy_sem_null    | -           |
#+-------------------------+---------+-----------------+---------+-------------------+-------
----+-------------------------+-----------------+--------------------+-------------+---------
----------+-------------+
#| Weekend                 | bool    | -               | -       | -                 | -
| -                      |             0.02 |                  2 |        0    |
dummy_sem_null    | -           |
```

```
#+-------------------------+---------+-----------------+---------+--------------------+-------
----+-------------------------+------------------+----------------------+-------------+---------
---------+-------------+

# Correlacao entre as variaveis:
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#|                         |   Administrative |   Administrative_Duration |   Informational |
Informational_Duration |   ProductRelated |   ProductRelated_Duration |   BounceRates |
ExitRates |   PageValues |   SpecialDay |   OperatingSystems |   Browser |   Region |
TrafficType |   Weekend |   Revenue |
#+=========================+==================+============================+=================+=
=========================+==================+============================+===============+=====
========+==============+==============+====================+===========+==========+===========
====+===========+===========+
#| Administrative          |            1     |                     0.601 |           0.377 |
0.256 |           0.431 |                    0.374 |        -0.223 |      -0.316 |
0.099 |       -0.095 |             -0.006 |    -0.025 |   -0.006 |        -0.034 |     0.026 |
0.139 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| Administrative_Duration |            0.601 |                     1     |           0.303 |
0.238 |           0.289 |                    0.355 |        -0.144 |      -0.206 |
0.067 |       -0.073 |             -0.007 |    -0.016 |   -0.006 |        -0.014 |     0.015 |
0.093 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| Informational           |            0.377 |                     0.303 |           1     |
0.619 |           0.374 |                    0.387 |        -0.116 |      -0.164 |
0.049 |       -0.048 |             -0.009 |    -0.038 |   -0.029 |        -0.035 |     0.036 |
0.095 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| Informational_Duration  |            0.256 |                     0.238 |           0.619 |
1     |           0.28  |                    0.347 |        -0.074 |      -0.105 |
0.031 |       -0.031 |             -0.01  |    -0.019 |   -0.027 |        -0.025 |     0.024 |
0.07  |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| ProductRelated          |            0.431 |                     0.289 |           0.374 |
0.28  |           1     |                    0.861 |        -0.204 |      -0.292 |
0.056 |       -0.024 |              0.004 |    -0.013 |   -0.038 |        -0.043 |     0.016 |
0.158 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| ProductRelated_Duration |            0.374 |                     0.355 |           0.387 |
0.347 |           0.861 |                    1     |        -0.184 |      -0.252 |
0.053 |       -0.037 |              0.003 |    -0.008 |   -0.033 |        -0.037 |     0.007 |
0.152 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| BounceRates             |           -0.223 |                    -0.144 |          -0.116 |
-0.074 |          -0.204 |                   -0.184 |         1     |       0.913 |       -
0.119 |        0.073 |              0.024 |    -0.016 |   -0.007 |         0.079 |    -0.047 |
-0.151 |
#+-------------------------+------------------+----------------------------+-----------------+-
-------------------------+------------------+----------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+-----------
----+-----------+-----------+
#| ExitRates               |           -0.316 |                    -0.206 |          -0.164 |
-0.105 |          -0.292 |                   -0.252 |         0.913 |       1     |       -
0.174 |        0.103 |              0.015 |    -0.004 |   -0.009 |         0.079 |    -0.063 |
-0.207 |
```

```
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| PageValues              |           0.099 |                    0.067 |           0.049 |
0.031 |           0.056 |                  0.053 |        -0.119 |      -0.174 |        1
|       -0.064 |              0.019 |     0.046 |    0.011 |         0.013 |     0.012 |
0.493 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| SpecialDay              |          -0.095 |                   -0.073 |          -0.048 |
-0.031 |          -0.024 |                 -0.037 |         0.073 |       0.103 |       -
0.064 |        1     |              0.013 |     0.003 |   -0.016 |         0.052 |    -0.017 |
-0.082 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| OperatingSystems        |          -0.006 |                   -0.007 |          -0.009 |
-0.01  |            0.004 |                  0.003 |         0.024 |       0.015 |
0.019 |        0.013 |              1     |     0.223 |    0.077 |         0.189 |     0     |
-0.015 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| Browser                 |          -0.025 |                   -0.016 |          -0.038 |
-0.019 |          -0.013 |                 -0.008 |        -0.016 |      -0.004 |
0.046 |        0.003 |              0.223 |     1     |    0.097 |         0.112 |    -0.04  |
0.024 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| Region                  |          -0.006 |                   -0.006 |          -0.029 |
-0.027 |          -0.038 |                 -0.033 |        -0.007 |      -0.009 |
0.011 |       -0.016 |              0.077 |     0.097 |    1     |         0.048 |    -0.001 |
-0.012 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| TrafficType             |          -0.034 |                   -0.014 |          -0.035 |
-0.025 |          -0.043 |                 -0.037 |         0.079 |       0.079 |
0.013 |        0.052 |              0.189 |     0.112 |    0.048 |         1     |    -0.002 |
-0.005 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| Weekend                 |           0.026 |                    0.015 |           0.036 |
0.024 |           0.016 |                  0.007 |        -0.047 |      -0.063 |
0.012 |       -0.017 |              0     |    -0.04  |   -0.001 |        -0.002 |     1     |
0.029 |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+
#| Revenue                 |           0.139 |                    0.093 |           0.095 |
0.07  |           0.158 |                  0.152 |        -0.151 |      -0.207 |
0.493 |       -0.082 |             -0.015 |     0.024 |   -0.012 |        -0.005 |     0.029 |
1     |
#+-------------------------+-----------------+--------------------------+-----------------+-
-------------------------+-----------------+--------------------------+---------------+-----
--------+--------------+--------------+--------------------+-----------+----------+----------
----+-----------+-----------+


# Tratamento de Dados
df2 = pd.concat([df['Revenue']])

# Criando Dataframe auxiliar
df_temp = pd.DataFrame()

# Adicionando colunas continuas sem nulos
```

```
df2 = pd.concat([df['PageValues'], df2], axis=1)

# Adicionando colunas continuas com nulos
df_temp = df.Administrative_Duration.fillna( df.Administrative_Duration.mean() )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.Informational_Duration.fillna( df.Informational_Duration.mean() )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.ProductRelated.fillna( df.ProductRelated.mean() )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.ProductRelated_Duration.fillna( df.ProductRelated_Duration.mean() )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.BounceRates.fillna( df.BounceRates.mean() )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.ExitRates.fillna( df.ExitRates.mean() )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies sem tratamento de NULL
df_temp = pd.get_dummies( df.SpecialDay, prefix='SpecialDay' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Month, prefix='Month' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.OperatingSystems, prefix='OperatingSystems' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Browser, prefix='Browser' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Region, prefix='Region' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.TrafficType, prefix='TrafficType' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.VisitorType, prefix='VisitorType' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = pd.get_dummies( df.Weekend, prefix='Weekend' )
df2 = pd.concat([df_temp, df2], axis=1)

# Adicionando e preparando colunas com dummies com tratamento de NULL
df_temp = df.Administrative.fillna( 'NULL' )
df_temp = pd.get_dummies( df_temp, prefix='Administrative' )
df2 = pd.concat([df_temp, df2], axis=1)
df_temp = df.Informational.fillna( 'NULL' )
df_temp = pd.get_dummies( df_temp, prefix='Informational' )
df2 = pd.concat([df_temp, df2], axis=1)


# Dividindo a base entre variaveis explicativas e variavel resposta
X = df2.loc[:, df2.columns !='Revenue']
Y = df2.Revenue

# Separando a Base entre Teste e Treino
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, train_size=0.75,
test_size=0.25, random_state=7)


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 4, 'max_features': 124, 'min_samples_leaf': 5,
'min_samples_split': 10}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------
```

```
# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': True, 'max_depth': 16, 'max_features': 93,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 31,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
```

```
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.001}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X train, Y train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='accuracy', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(accuracy_score(Y_train, predict_train),6)
score_teste = round(accuracy_score(Y_test, predict_test),6)

# ------------------------------------------------
```

```
# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 2, 'max_features': 93, 'min_samples_leaf': 5,
'min_samples_split': 1.0}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X train, Y train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 16, 'max_features': 62,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)
```

```
# ---------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.25, 'max_depth': 2, 'max_features': 124,
'min_samples_leaf': 0.1, 'min_samples_split': 0.1, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 8, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()
```

```
# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='f1', verbose=10, refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(f1_score(Y_train, predict_train),6)
score_teste = round(f1_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 2, 'max_features': 124, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 9,
'min_child_weight': 20, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
```

```
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': True, 'max_depth': 2, 'max_features': 62,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 31,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 16}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict train = model.predict(X train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 32, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ------------------------------------------------
```

```
# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.001}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='precision', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(precision_score(Y_train, predict_train),6)
score_teste = round(precision_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: DecisionTreeClassifier
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 2, 'max_features': 93, 'min_samples_leaf': 0.1,
'min_samples_split': 1.0}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()
```

```
# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 2, 'max_features': 124,
'min_samples_leaf': 0.1, 'min_samples_split': 1.0, 'n_estimators': 10}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict train = model.predict(X train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 1, 'max_depth': 16, 'max_features': 93,
'min_samples_leaf': 0.1, 'min_samples_split': 1.0, 'n_estimators': 4}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)
```

```
# ---------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 2, 'p': 5}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='recall', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 10, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(recall_score(Y_train, predict_train),6)
score_teste = round(recall_score(Y_test, predict_test),6)

# ---------------------------------------------


# Modelo: DecisionTreeClassifier
```

```
modelo = DecisionTreeClassifier()

# Tunning:
parametros = {'max_depth': [2, 4, 8, 12, 16], 'min_samples_split': [0.1, 1.0, 10],
'min_samples_leaf': [0.1, 0.5, 5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'max_depth': 12, 'max_features': 124, 'min_samples_leaf': 5,
'min_samples_split': 0.1}

# Aplicando o Modelo
model = DecisionTreeClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: XGBClassifier
modelo = XGBClassifier()

# Tunning:
parametros = {'nthread': [4], 'objective': ['binary:logistic'], 'learning_rate': [0.05],
'max_depth': [8, 9, 10], 'min_child_weight': [12, 20, 30], 'silent': [1], 'subsample': [0.5,
0.6, 0.7], 'colsample_bytree': [0.6, 0.7, 0.8], 'n_estimators': [5, 10, 20]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 9,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7}

# Aplicando o Modelo
model = XGBClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict train = model.predict(X train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------


# Modelo: RandomForestClassifier
modelo = RandomForestClassifier()

# Tunning:
parametros = {'n_estimators': [10, 50, 100], 'max_depth': [2, 8, 16], 'min_samples_split':
[0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5, 5], 'bootstrap': [True, False], 'max_features':
[31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'bootstrap': False, 'max_depth': 8, 'max_features': 62,
'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 100}

# Aplicando o Modelo
model = RandomForestClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)
```

```
# ----------------------------------------------


# Modelo: GradientBoostingClassifier
modelo = GradientBoostingClassifier()

# Tunning:
parametros = {'learning_rate': [1, 0.25, 0.05, 0.01], 'n_estimators': [4, 16, 100],
'max_depth': [2, 8, 16], 'min_samples_split': [0.1, 1.0, 10], 'min_samples_leaf': [0.1, 0.5,
5], 'max_features': [31, 62, 93, 124]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'learning_rate': 0.05, 'max_depth': 8, 'max_features': 31,
'min_samples_leaf': 5, 'min_samples_split': 0.1, 'n_estimators': 100}

# Aplicando o Modelo
model = GradientBoostingClassifier(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ----------------------------------------------


# Modelo: KNeighborsClassifier
modelo = KNeighborsClassifier()

# Tunning:
parametros = {'n_neighbors': [2, 8, 16, 32], 'p': [1, 3, 5]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'n_neighbors': 32, 'p': 1}

# Aplicando o Modelo
model = KNeighborsClassifier(**mod.best_params_)
model.fit(X train, Y train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ----------------------------------------------


# Modelo: LogisticRegression
modelo = LogisticRegression()

# Tunning:
parametros = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 0.01}

# Aplicando o Modelo
model = LogisticRegression(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ----------------------------------------------
```

```
# Modelo: SVC
modelo = SVC()

# Tunning:
parametros = {'gamma': [0.05, 0.1, 1, 5, 10], 'C': [0.1, 1, 10, 100, 1000], 'degree': [0, 1,
2, 3, 5, 7]}
mod = GridSearchCV(modelo, parametros, n_jobs=4,  cv=4, scoring='roc_auc', verbose=10,
refit=True)
mod.fit(X_train, Y_train)

# Melhores Parametros: {'C': 1, 'degree': 0, 'gamma': 0.05}

# Aplicando o Modelo
model = SVC(**mod.best_params_)
model.fit(X_train, Y_train)
predict_train = model.predict(X_train)
predict_test = model.predict(X_test)

# Calculando o Score
score_treino = round(roc_auc_score(Y_train, predict_train),6)
score_teste = round(roc_auc_score(Y_test, predict_test),6)

# ------------------------------------------------

# Comparativo de Modelos:

#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
#| Modelo                     | Metrica_Score   |   Tempo_Exec(s) |   Score_Treino |
Score_Teste | Melhor_hiper_param
|
#+============================+=================+=================+================+==========
=====+=========================================================================================
===============================================================================================
=====+
#| DecisionTreeClassifier     | accuracy        |           20.43 |       0.906889 |
0.903665 | {'max_depth': 4, 'max_features': 124, 'min_samples_leaf': 5, 'min_samples_split':
10}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | accuracy        |          331.95 |       0.908619 |
0.904314 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | accuracy        |          644.96 |       0.955445 |
0.907233 | {'bootstrap': True, 'max depth': 16, 'max_features': 93, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | accuracy        |         2149.68 |       1        |
0.90626  | {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 31, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | accuracy        |           90.83 |       0.879637 |
0.869283 | {'n_neighbors': 8, 'p': 5}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+
```

```
#| LogisticRegression         | accuracy        |            2.32 |       0.883097 |
0.885825 | {'C': 0.001}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | accuracy        |         6249.53 |       0.994485 |
0.842361 | {'C': 1, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | f1              |           16.27 |       0.657526 |
0.688153 | {'max_depth': 2, 'max_features': 93, 'min_samples_leaf': 5, 'min_samples_split':
1.0}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | f1              |          328.78 |       0.670051 |
0.658169 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | f1              |          662.88 |       0.668624 |
0.655629 | {'bootstrap': False, 'max_depth': 16, 'max_features': 62, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | f1              |         1947.83 |       0.679502 |
0.674888 | {'learning_rate': 0.25, 'max_depth': 2, 'max_features': 124, 'min_samples_leaf':
0.1, 'min_samples_split': 0.1, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | f1              |           87.61 |       0.417582 |
0.379045 | {'n_neighbors': 8, 'p': 5}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | f1              |            2.35 |       0.507186 |
0.508796 | {'C': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | f1              |         6286.89 |       0.999648 |
0.008016 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | precision       |           13.67 |       0.729761 |
0.748387 | {'max_depth': 2, 'max_features': 124, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | precision       |          336.98 |       0.807735 |
0.83274  | {'colsample_bytree': 0.6, 'learning_rate': 0.05, 'max_depth': 9,
```

```
'min_child_weight': 20, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.5} |
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | precision      |          651.44 |       0.825832 |
0.834225 | {'bootstrap': True, 'max_depth': 2, 'max_features': 62, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 10}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | precision      |         1978.6  |       0.953271 |
0.942857 | {'learning_rate': 0.05, 'max_depth': 16, 'max_features': 31, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 16}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | precision      |          105.54 |       0.878788 |
0.894737 | {'n_neighbors': 32, 'p': 5}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | precision      |            2.96 |       0.753343 |
0.768    | {'C': 0.001}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | precision      |         6270.54 |       1        |
0.153846 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | recall         |           13.01 |       0.792546 |
0.812757 | {'max_depth': 2, 'max_features': 93, 'min_samples_leaf': 0.1, 'min_samples_split':
1.0}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | recall         |          332.63 |       0.603376 |
0.584362 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 8,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | recall         |          668.3  |       0.792546 |
0.812757 | {'bootstrap': False, 'max_depth': 2, 'max_features': 124, 'min_samples_leaf': 0.1,
'min_samples_split': 1.0, 'n_estimators': 10}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | recall         |         1909.52 |       0.555556 |
0.555556 | {'learning_rate': 1, 'max_depth': 16, 'max_features': 93, 'min_samples_leaf': 0.1,
'min_samples_split': 1.0, 'n_estimators': 4}
|
#+----------------------------+----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
```

```
#| KNeighborsClassifier       | recall          |           91.23 |       0.396624 |
0.246914 | {'n_neighbors': 2, 'p': 5}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | recall          |            2.36 |       0.384669 |
0.386831 | {'C': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | recall          |         6369.5  |       0.999297 |
0.004115 | {'C': 10, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| DecisionTreeClassifier     | roc_auc         |           17.36 |       0.782617 |
0.781164 | {'max_depth': 12, 'max_features': 124, 'min_samples_leaf': 5, 'min_samples_split':
0.1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| XGBClassifier              | roc_auc         |          336.74 |       0.783796 |
0.774276 | {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 9,
'min_child_weight': 30, 'n_estimators': 20, 'nthread': 4, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.7} |
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| RandomForestClassifier     | roc_auc         |          663.34 |       0.835673 |
0.782181 | {'bootstrap': False, 'max_depth': 8, 'max_features': 62, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| GradientBoostingClassifier | roc_auc         |         1960.67 |       0.808569 |
0.78688  | {'learning_rate': 0.05, 'max_depth': 8, 'max_features': 31, 'min_samples_leaf': 5,
'min_samples_split': 0.1, 'n_estimators': 100}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| KNeighborsClassifier       | roc_auc         |           83.11 |       0.532381 |
0.525786 | {'n_neighbors': 32, 'p': 1}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| LogisticRegression         | roc_auc         |            2.43 |       0.672554 |
0.688037 | {'C': 0.01}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
#| SVC                        | roc_auc         |         6593.62 |       0.982068 |
0.500836 | {'C': 1, 'degree': 0, 'gamma': 0.05}
|
#+----------------------------+-----------------+-----------------+----------------+----------
-----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+
```