UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Helen Vitória Brandão de Sousa e Ingrid Melo de Oliveira

1. Introdução

O relatório visa analisar dois conjuntos de dados distintos, ambos de domínio público, e prever comportamentos ou custos com base em dados históricos. O primeiro conjunto de dados refere-se a reservas de hotéis, onde o objetivo é prever se o cliente vai honrar a reserva ou cancelá-la, levando em consideração fatores como mudança de planos ou conflitos de agendamento. Já o segundo conjunto de dados envolve informações sobre custos médicos e busca-se prever os custos do seguro com base em dados pessoais e históricos médicos. Ambos os conjuntos exigem um processo de limpeza e recodificação dos dados para adequá-los à análise. Este trabalho faz parte de uma avaliação na disciplina de Inteligência Artificial, com foco no desenvolvimento de modelos preditivos. Logo abaixo, estão mais informações sobre os datasets, que estão separados devido à natureza das tarefas: o conjunto de dados de reservas de hotel é destinado a uma tarefa de classificação, enquanto o conjunto de dados de custos médicos é uma tarefa de regressão.

Dataset de Classificação

O uso de plataformas online para fazer reservas em hotéis, tem afetado significativamente a indústria hoteleira, pois tem mudado muito a possibilidade de reserva e o comportamento dos clientes. Os principais problemas com relação a isso são o alto índice de cancelamentos e ausências (o cliente não aparece na data reservada), afetando diretamente a receita dos hotéis e as ocupações do mesmo. As razões típicas para os cancelamentos incluem mudança de planos, conflitos de agenda, etc. Isso geralmente é facilitado pela opção de fazer o cancelamento sem custos ou com um custo

baixo, o que é benéfico para os hóspedes do hotel, mas é um fator possivelmente prejudicial à receita para os hotéis.

O dataset em questão contém dados sobre as reservas realizadas por clientes em hotéis, incluindo informações sobre a natureza dessas reservas e o comportamento dos clientes. Entre as variáveis registradas, estão informações sobre o cliente como idade dos clientes, gênero, massa corporal(IMC), quantidade de dependentes, região em que moram e suas rendas.

A análise desse conjunto de dados irá proporcionar a previsão da probabilidade de cancelamento de uma reserva, permitindo que a indústria hoteleira tenha insights valiosos, fazendo com que antecipem cancelamentos e adotem estratégias para minimizar o impacto desses eventos, consequentemente otimizando suas operações e as suas receitas.

Dicionário de Dados - Classificação

- ***Booking_ID:*** *identificador único de cada reserva*
- ***no_of_adults:*** *Número de adultos*
- ***no_of_children:*** *Número de filhos*
- ***no_of_weekend_nights:*** *Número de noites de fim de semana (sábado ou domingo) em que o hóspede ficou ou reservou para ficar no hotel*
- ***no_of_week_nights:*** *Número de noites da semana (segunda a sexta) em que o hóspede ficou ou reservou para ficar no hotel*
- ***type_of_meal_plan:*** *Tipo de plano de refeições reservado pelo cliente:*
- ***required_car_parking_space:*** *O cliente precisa de uma vaga de estacionamento? (0 - Não, 1 - Sim)*
- ***room_type_reserved:*** *Tipo de quarto reservado pelo cliente. Os valores são cifrados (codificados) pelo INN Hotels.*
- ***lead_time:*** *Número de dias entre a data da reserva e a data de chegada*
- ***arrival_year:*** *Ano da data de chegada*

- ***arrival_month:*** *Mês da data de chegada*
- ***arrival_date:*** *Data do mês*
- ***market_segment_type:*** *Designação do segmento de mercado.*
- ***repeated_guest:*** *O cliente é um hóspede repetido? (0 - Não, 1 - Sim)*
- ***no_of_previous_cancellations:*** *Número de reservas anteriores que foram canceladas pelo cliente antes da reserva atual*
- ***no_of_previous_bookings_not_canceled:*** *Número de reservas anteriores não canceladas pelo cliente antes da reserva atual*
- ***avg_price_per_room:*** *Preço médio por dia da reserva; os preços dos quartos são dinâmicos. (em euros)*
- ***no_of_special_requests:*** *Número total de solicitações especiais feitas pelo cliente (por exemplo, andar alto, vista do quarto, etc.)*
- ***booking_status:*** *Indicador que indica se a reserva foi cancelada ou não.*

Dataset de Regressão

Atualmente pode-se notar um crescente interesse em aplicar técnicas de machine learning em diversas áreas, para melhorar as performances e decisões. Um dos tópicos importantes é a previsão de custos de seguro de saúde, o livro Machine Learning with R de Brett Lantz introduz o conceito de Machine Learning usando a linguagem de programação R, e o conjunto de dados que acompanha o livro é utilizado para demonstrar como prever os custos de seguro com base em diferentes fatores relacionados aos beneficiários, porém não é disponibilizado seus conjuntos de dados online, a menos que você compre o livro e crie uma conta de usuário, o que pode ser um problema caso você esteja pegando o livro na biblioteca ou emprestado de um amigo. Todos esses conjuntos de dados estão disponibilizados publicamente, mas precisam de uma limpeza e recodificação para corresponder ao formato do livro.

O dataset utilizado neste trabalho é um exemplo prático dessa aplicação, com o objetivo de prever os custos individuais médicos cobrados pelo seguro com base em características pessoais dos beneficiários. Esse tipo de análise permite entender como

variáveis demográficas e de comportamento influenciam nos custos dos serviços de saúde e pode ser útil tanto para empresas de seguros quanto para a formulação de políticas públicas de saúde. O conjunto de dados contém informações sobre diversos atributos dos beneficiários do seguro, como idade, gênero, índice de massa corporal (IMC), número de dependentes, status de fumante e a região de residência.

Neste trabalho, serão analisadas as relações entre essas variáveis e os custos com o seguro de saúde, utilizando métodos de Machine Learning para desenvolver um modelo preditivo. O objetivo é identificar padrões e correlacionar os atributos mais relevantes que impactam os custos, permitindo, assim, melhorar a capacidade de previsão e tomada de decisão por parte das seguradoras. A análise desse dataset fornece insights valiosos para entender as dinâmicas de precificação de seguros de saúde e os fatores de risco associados.

Dicionário de Dados - Regressão

- ***Idade:*** *idade do beneficiário principal.*
- ***Sexo:*** *gênero do contratante de seguros, feminino, masculino.*
- ***IMC:*** *Índice de massa corporal, que fornece uma compreensão do corpo, pesos que são relativamente altos ou baixos em relação à altura, índice objetivo do peso corporal (kg / m ^ 2) usando a proporção entre altura e peso, idealmente 18,5 a 24,9.*
- ***Crianças:*** *Número de crianças cobertas pelo seguro de saúde / Número de dependentes.*
- ***Fumante:*** *Se o contratante fuma.*
- ***Região:*** *área residencial do beneficiário nos EUA, nordeste, sudeste, sudoeste, noroeste.*
- ***Encargos:*** *Custos médicos individuais cobrados pelo seguro de saúde*

2. Observações

Ao decorrer da execução do projeto, tiveram vários problemas com relação ao GitHub, mais por conta de não sabermos utilizar muito bem. Também houve problemas em questão de nossos computadores não suportarem algumas implementações que tentamos fazer então tivemos que reduzir os códigos e as informações.

3. Resultados e discussão

3.1 Primeira Questão

Neste primeiro exercício foi realizado a manipulação de um dataset com a biblioteca pandas e realizado o pré-processamento deste.

Instruções:

1. ***Importando as bibliotecas necessárias.***

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
```

2. ***Exploração do Dataset.***

```
#Importando o dataset:
classificacao = pd.read_csv('./dataset/data_classificacao.csv')

#Exploração do dataset:
print('Exploração do dataset')
print(classificacao.head(100))

# Verificação do número de linhas e colunas:
print('Verificação do número de linhas e colunas')
print(classificacao.shape)

#Todas as informações do dataset:
print('Informações do dataset')
print(classificacao.info())
```

```
#Estatísticas do dataset:
print('Estatísticas do dataset')
print(classificacao.describe().T)

#Nome das colunas:
print('Colunas do dataset')
print(classificacao.columns)

#Tipo de dados de cada coluna:
print('Tipos de dados das colunas')
print(classificacao.dtypes)
```

3. ***Verificação de valores nulos.***

```
#Verificação de valores nulos:
print('Verificação de valores nulos')
print(classificacao.isna().sum())

#Verificação de duplicatas:
print('Verificação de duplicatas')
print(classificacao.duplicated().sum())
```

4. ***Verificação das colunas mais relevantes e criação de um novo data frame com somente as colunas necessárias.***

```
#Separação de colunas com dados categóricos:
print('Colunas com dados categoricos')
categoricos_col = [cat for cat in classificacao.columns if
classificacao[cat].dtype=='O']
print('Existem {} colunas
categóricas'.format(len(categoricos_col)))
print(categoricos_col)

#Separação de colunas com dados numéricos:
print('Colunas com dados numericos')
numericos_col = [num for num in classificacao.columns if
classificacao[num].dtype!='O']
print('Existem {} colunas numericas'.format(len(numericos_col)))
print(numericos_col)
```

```
        #Selecionar colunas relevantes
        print('Colunas relevantes')
        colunas_relevantes = ['type_of_meal_plan', 'room_type_reserved',
'market_segment_type', 'booking_status', 'lead_time',
'avg_price_per_room', 'no_of_special_requests',
'no_of_previous_cancellations']
        classificacao = classificacao[colunas_relevantes]
```

5. ***A coluna de classes precisa ser renomeada para atributos numéricos, e foi feita a conversão.***

```
        #Codificação de rótulo em colunas categóricas que são:
type_of_meal_plan, room_type_reserved, market_segment_type,
booking_status
        label_enconder=LabelEncoder()
        classificacao['type_of_meal_plan']=label_enconder.fit_transform(
classificacao['type_of_meal_plan'])
        classificacao['room_type_reserved']=label_enconder.fit_transform
(classificacao['room_type_reserved'])
        classificacao['market_segment_type']=label_enconder.fit_transfor
m(classificacao['market_segment_type'])
        classificacao['booking_status']=label_enconder.fit_transform(cla
ssificacao['booking_status'])
        print(classificacao)
```

6. ***A distribuição de classes que vão ser classificadas.***

```
        # Mostrar a distribuição de classes
        print("Distribuição de classes:")
        print(classificacao['booking_status'].value_counts())

        #DataFrame final
        print("DataFrame final após pré-processamento:")
        print(classificacao.head())
```

7. ***Gráfico com as ocorrências da coluna 'booking_status'.***

```
        # Contar ocorrências
```

```
        contagem = classificacao['booking_status'].value_counts()

        # Gráfico
        plt.figure(figsize=(10, 5))
        plt.bar(contagem.index, contagem.values, color=['skyblue',
'salmon'])
        plt.xticks([0, 1], ['Canceled', 'Not_Canceled'])
        plt.xlabel('Status de Reserva')
        plt.ylabel('Contagem')
        plt.title('Status de Reservas')
        plt.s
```

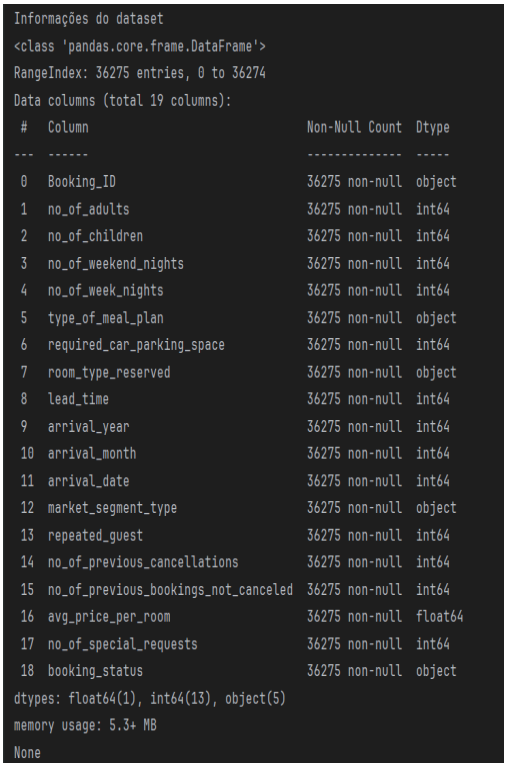
8. ***O Dataset foi atualizado e salvo como dataset_classificacao_ajustado.csvhow()***

```
        # Salvar o dataset atualizado
        classificacao.to_csv('dataset_classificacao_ajustado.csv',
index=False)
```

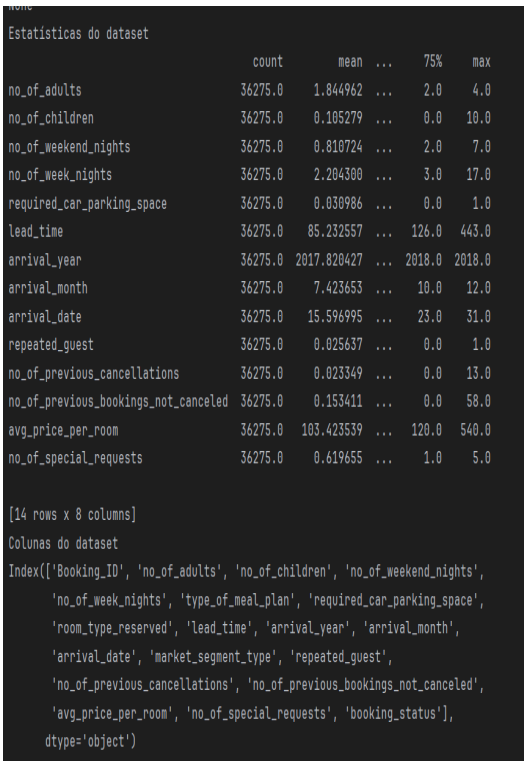
3.1.3 Resultados

Registro dos resultados obtidos durante a exploração do dataset.

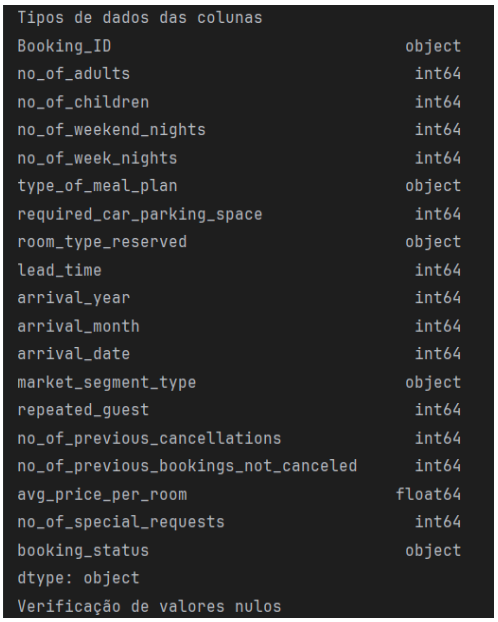
-
```
C:\Users\win11\anaconda3\envs\IA--SI\python.exe C:\Users\win11\Desktop\IA-SI\IA--SI\AV1\questao1.py
Exploração do dataset
   Booking_ID  no_of_adults  ...  no_of_special_requests  booking_status
0    INN00001             2  ...                       0    Not_Canceled
1    INN00002             2  ...                       1    Not_Canceled
2    INN00003             1  ...                       0        Canceled
3    INN00004             2  ...                       0        Canceled
4    INN00005             2  ...                       0        Canceled
..        ...           ...  ...                     ...             ...
95   INN00096             2  ...                       2    Not_Canceled
96   INN00097             2  ...                       1        Canceled
97   INN00098             2  ...                       1    Not_Canceled
98   INN00099             2  ...                       1    Not_Canceled
99   INN00100             2  ...                       0    Not_Canceled

[100 rows x 19 columns]
Verificação do numero de linhas e colunas
(36275, 19)
```

- 

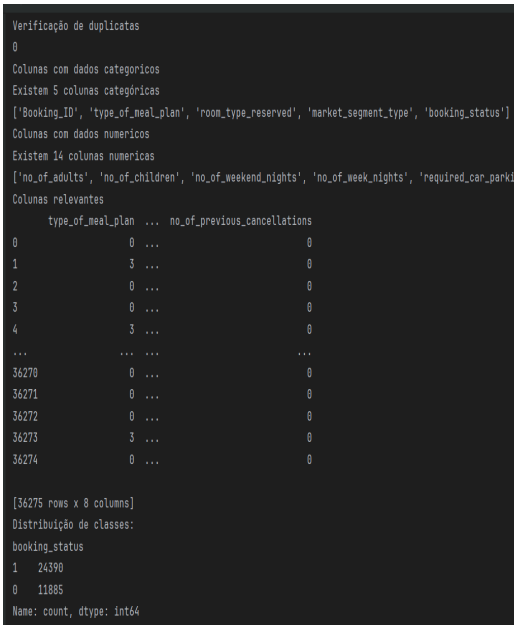
```
Informações do dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Booking_ID                            36275 non-null  object
 1   no_of_adults                          36275 non-null  int64
 2   no_of_children                        36275 non-null  int64
 3   no_of_weekend_nights                  36275 non-null  int64
 4   no_of_week_nights                     36275 non-null  int64
 5   type_of_meal_plan                     36275 non-null  object
 6   required_car_parking_space            36275 non-null  int64
 7   room_type_reserved                    36275 non-null  object
 8   lead_time                             36275 non-null  int64
 9   arrival_year                          36275 non-null  int64
 10  arrival_month                         36275 non-null  int64
 11  arrival_date                          36275 non-null  int64
 12  market_segment_type                   36275 non-null  object
 13  repeated_guest                        36275 non-null  int64
 14  no_of_previous_cancellations          36275 non-null  int64
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64
 18  booking_status                        36275 non-null  object
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
None
```



```
Estatísticas do dataset
                                       count        mean  ...     75%     max
no_of_adults                         36275.0    1.844962  ...     2.0     4.0
no_of_children                       36275.0    0.105279  ...     0.0    10.0
no_of_weekend_nights                 36275.0    0.810724  ...     2.0     7.0
no_of_week_nights                    36275.0    2.204300  ...     3.0    17.0
required_car_parking_space           36275.0    0.030986  ...     0.0     1.0
lead_time                            36275.0   85.232557  ...   126.0   443.0
arrival_year                         36275.0 2017.820427  ...  2018.0  2018.0
arrival_month                        36275.0    7.423653  ...    10.0    12.0
arrival_date                         36275.0   15.596995  ...    23.0    31.0
repeated_guest                       36275.0    0.025637  ...     0.0     1.0
no_of_previous_cancellations         36275.0    0.023349  ...     0.0    13.0
no_of_previous_bookings_not_canceled 36275.0    0.153411  ...     0.0    58.0
avg_price_per_room                   36275.0  103.423539  ...   120.0   540.0
no_of_special_requests               36275.0    0.619655  ...     1.0     5.0

[14 rows x 8 columns]
Colunas do dataset
Index(['Booking_ID', 'no_of_adults', 'no_of_children', 'no_of_weekend_nights',
       'no_of_week_nights', 'type_of_meal_plan', 'required_car_parking_space',
       'room_type_reserved', 'lead_time', 'arrival_year', 'arrival_month',
       'arrival_date', 'market_segment_type', 'repeated_guest',
       'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled',
       'avg_price_per_room', 'no_of_special_requests', 'booking_status'],
      dtype='object')
```



```
Tipos de dados das colunas
Booking_ID                              object
no_of_adults                             int64
no_of_children                           int64
no_of_weekend_nights                     int64
no_of_week_nights                        int64
type_of_meal_plan                       object
required_car_parking_space               int64
room_type_reserved                      object
lead_time                                int64
arrival_year                             int64
arrival_month                            int64
arrival_date                             int64
market_segment_type                     object
repeated_guest                           int64
no_of_previous_cancellations             int64
no_of_previous_bookings_not_canceled     int64
avg_price_per_room                     float64
no_of_special_requests                   int64
booking_status                          object
dtype: object
Verificação de valores nulos
```



```
Verificação de duplicatas
0
Colunas com dados categoricos
Existem 5 colunas categóricas
['Booking_ID', 'type_of_meal_plan', 'room_type_reserved', 'market_segment_type', 'booking_status']
Colunas com dados numericos
Existem 14 colunas numericas
['no_of_adults', 'no_of_children', 'no_of_weekend_nights', 'no_of_week_nights', 'required_car_parki
Colunas relevantes
      type_of_meal_plan  ...  no_of_previous_cancellations
0                     0  ...                             0
1                     3  ...                             0
2                     0  ...                             0
3                     0  ...                             0
4                     3  ...                             0
...                 ...  ...                           ...
36270                 0  ...                             0
36271                 0  ...                             0
36272                 0  ...                             0
36273                 3  ...                             0
36274                 0  ...                             0

[36275 rows x 8 columns]
Distribuição de classes:
booking_status
1    24390
0    11885
Name: count, dtype: int64
```

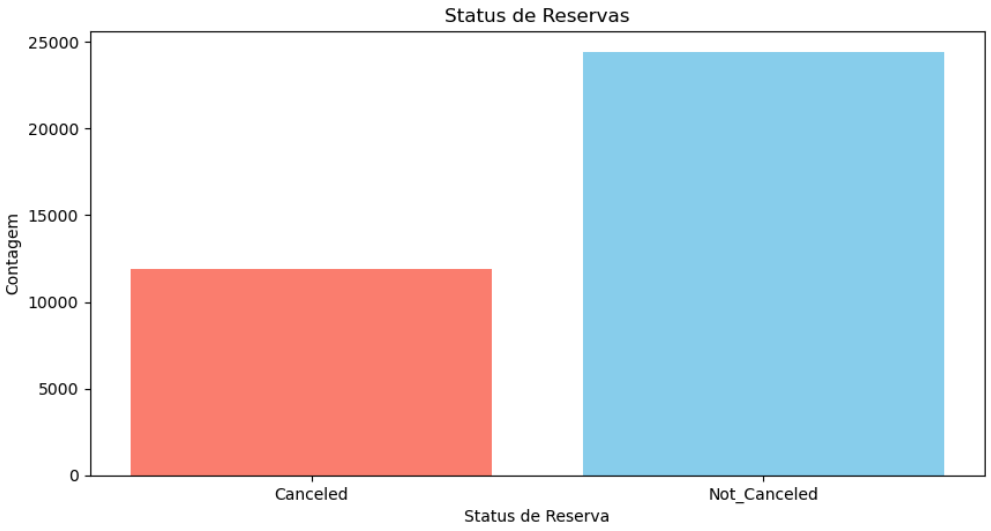
```
DataFrame final após pré-processamento:
   type_of_meal_plan  ...  no_of_previous_cancellations
0                  0  ...                            0
1                  3  ...                            0
2                  0  ...                            0
3                  0  ...                            0
4                  3  ...                            0

[5 rows x 8 columns]
O pré-processamento atual cobre todas as etapas necessárias para este exercício.

Process finished with exit code 0
```

- ***Registro do gráfico:***

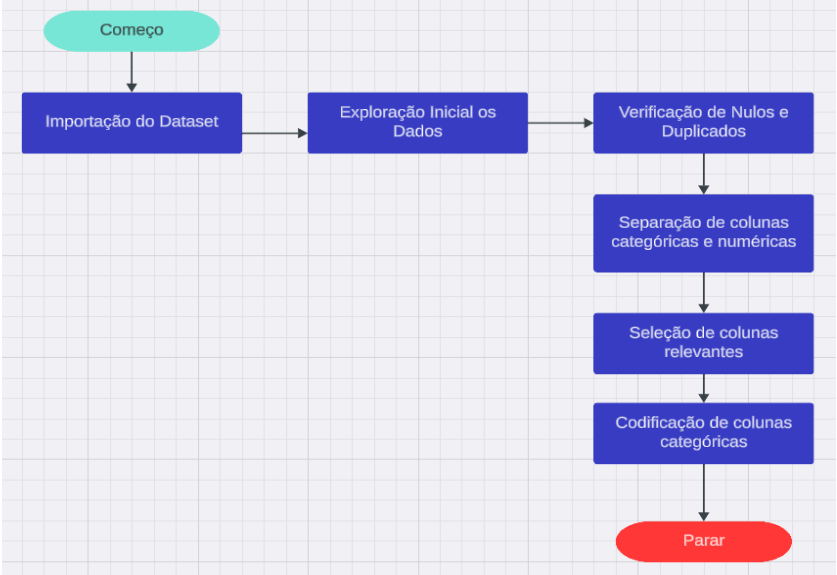
  Após realizar as etapas de pré-processamento, a distribuição das classes de reserva foi visualizada por meio de um gráfico de barras. O gráfico revelou a seguinte distribuição entre as classes "Canceled" e "Not_Canceled":

  Cancelamentos (Canceled): 11885

  Não Cancelados (Not_Canceled): 24390



- ***Fluxograma do processo:***



3.2.1 Segunda Questão

Neste segundo exercício realizamos uma classificação utilizando KNN implementado de forma manual.

Instruções

1. ***Importando as bibliotecas necessárias.***

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from scipy.spatial.distance import mahalanobis,chebyshev, cityblock, euclidean
```

2. ***Carregando o dataset atualizado***

```
# Carregar o dataset
classificacao=pd.read_csv('./dataset/dataset_classificacao_ajustado.csv')
```

3. ***Sem normalizar o conjunto de dados divida o dataset em treino e teste.***

```
# Dividir em X (entradas) e y (saídas)
X = classificacao.drop(columns=['booking_status'], axis=1).values
y = classificacao['booking_status'].values

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, stratify=y, random_state=42)
```

4. ***Implementando o Knn exibindo sua acurácia nos dados de teste, utilizando um valor de k= 7***

```
#KNN
def knn_manual_optimized(X_train, y_train, X_test, k,
distance_func, inv_cov_matrix=None):
y_pred = []
  # Limitar o tamanho do conjunto de treino para otimizar o cálculo
  sample_size = min(len(X_train), 5000)  # Usar no máximo 5000
  amostras
  X_train_sample = X_train[:sample_size]
  y_train_sample = y_train[:sample_size]

  for test_point in X_test:
    distances = []
    for i, train_point in enumerate(X_train_sample):
        if distance_func == mahalanobis_distance:
        dist = distance_func(test_point, train_point,
inv_cov_matrix)
        else:
            dist = distance_func(test_point, train_point)
        distances.append((dist, y_train_sample[i]))
    distances = sorted(distances, key=lambda x: x[0])[:k]
    classes = [neighbor[1] for neighbor in distances]
    y_pred.append(max(set(classes), key=classes.count))
  return np.array(y_pred)
```

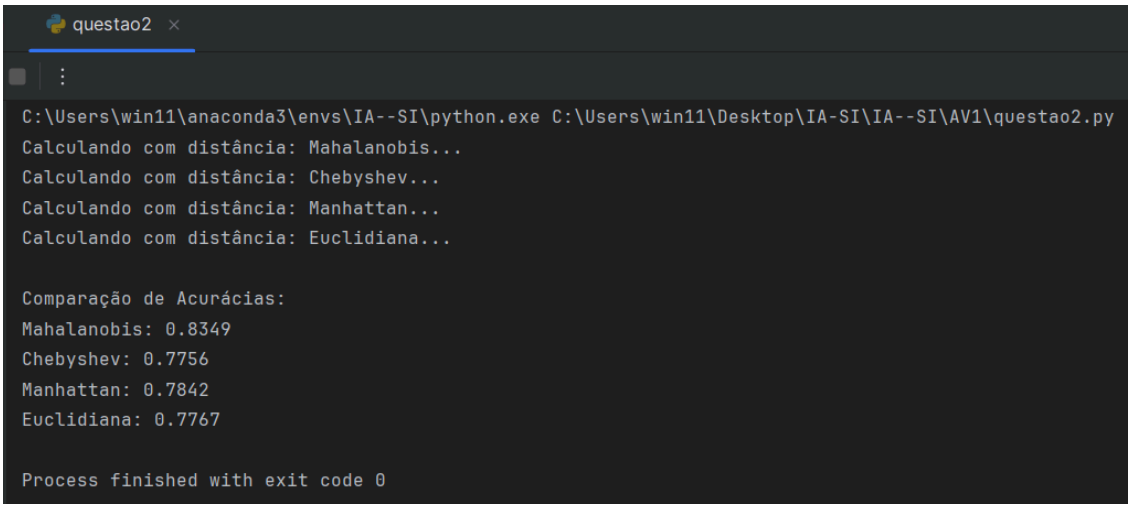
5. ***Comparação entre as acurácias considerando os 4 possíveis cálculos de distâncias diferentes: a) distância de mahalanobis. b) distância de chebyshev c) distância de manhattan d) distância euclidiana***

```
# Funções de distância
def mahalanobis_distance(x, y, inv_cov_matrix):
return mahalanobis(x, y, inv_cov_matrix)

def chebyshev_distance(x, y):
return chebyshev(x, y)

def manhattan_distance(x, y):
return cityblock(x, y)

def euclidean_distance(x, y):
return euclidean(x, y)

# Calcular a matriz inversa da covariância para Mahalanobis
cov_matrix = np.cov(X_train, rowvar=False)
inv_cov_matrix = np.linalg.inv(cov_matrix)

# Valor de k
k = 7

# Distâncias para teste
distances = {
"Mahalanobis": lambda x, y: mahalanobis_distance(x, y,
inv_cov_matrix),
"Chebyshev": chebyshev_distance,
"Manhattan": manhattan_distance,
"Euclidiana": euclidean_distance
}
# Avaliar e comparar
results = {}
for name, func in distances.items():
print(f"Calculando com distância: {name}...")
y_pred = knn_manual_optimized(X_train, y_train, X_test, k, func,
inv_cov_matrix)
```

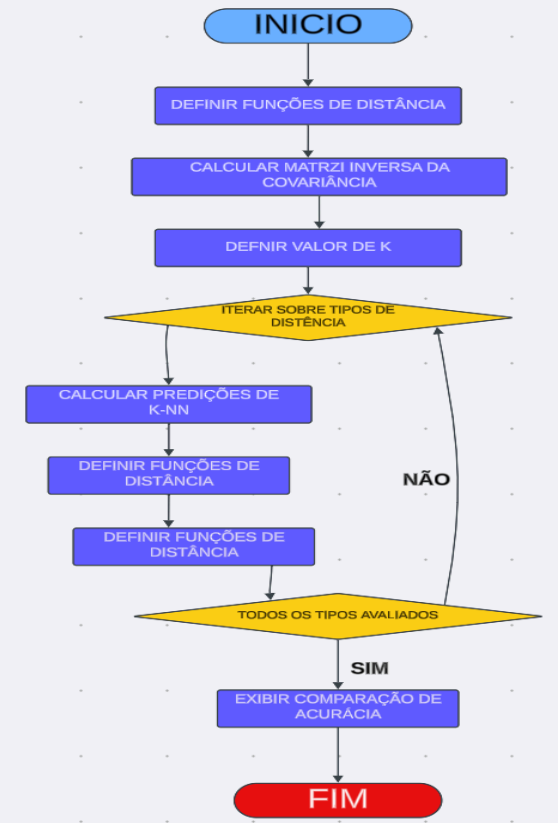
```
acc = accuracy_score(y_test, y_pred)
results[name] = acc

# Mostrar resultados
print("\nComparação de Acurácias:")
for name, acc in results.items():
print(f"{name}: {acc:.4f}")
```

3.2.2 Resultados

- *Resultado da comparação entre as acurácias considerando os 4 possíveis cálculos de distâncias.*
- *Depois de analisar as acurácias, é possível perceber que a distância que teve melhor resultado foi a Mahalanobis, que teve 0.8349 de acurácia.*



- ***Fluxograma do processo:***



3.3 Questão 3

Consideramos a melhor distância observada no exercício anterior e verificamos se a normalização interfere nos resultados da classificação.

Seguindo as instruções:

1. ***Importando as bibliotecas necessárias.***

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

2. ***Carregue o dataset atualizado***

```
# Carregar o dataset
```

```
classificacao =
pd.read_csv('./dataset/dataset_classificacao_ajustado.csv')

# Dividir em X (entradas) e y (saídas)
X = classificacao.drop(columns=['booking_status'], axis=1).values
y = classificacao['booking_status'].values

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, stratify=y, random_state=42)
```

3. ***Normalizando o conjunto de dados com normalização logarítmica e verificando a acurácia do knn.***

```
# Normalização logarítmica
log_transformer = FunctionTransformer(np.log1p, validate=True)
X_train_log = log_transformer.fit_transform(X_train)
X_test_log = log_transformer.transform(X_test)
```

4. ***Normalizando o conjunto de dados com normalização de média zero e variância unitária e verificando a acurácia do knn.***

```
# Normalização z-score
scaler = StandardScaler()
X_train_zscore = scaler.fit_transform(X_train)
X_test_zscore = scaler.transform(X_test)

# Limitar o número de amostras para 5000
X_train_limited = X_train[:5000]
y_train_limited = y_train[:5000]
X_test_limited = X_test[:5000]
y_test_limited = y_test[:5000]

# Instanciando o KNeighborsClassifier com Mahalanobis como a
métrica
```

```
knn = KNeighborsClassifier(n_neighbors=7, metric='mahalanobis',
metric_params={'VI': np.linalg.inv(np.cov(X_train_log,
rowvar=False))})

# Avaliar com normalização logarítmica
knn.fit(X_train_log, y_train)
y_pred_log = knn.predict(X_test_log)
accuracy_log = accuracy_score(y_test, y_pred_log)

# Avaliar com normalização z-score
knn.fit(X_train_zscore, y_train)
y_pred_zscore = knn.predict(X_test_zscore)
accuracy_zscore = accuracy_score(y_test, y_pred_zscore)
```

5. ***Print das duas acurácias lado a lado para comparar.***

```
# Comparar resultados
print("Acurácias:")
print(f"Normalização Logarítmica: {accuracy_log:.4f}")
print(f"Normalização Z-score: {accuracy_zscore:.4f}")
```

3.3.2 Resultados

- *Com base nesse resultado é possível observar que a melhor normalização foi a Logarítmica que retorna uma valor de 0.8682.*

3.4 Questão 4

Com base nas parametrizações vistas anteriormente, neste exercício buscamos saber a melhor parametrização do knn implementado na questão anterior.

Seguindo as instruções:

1. ***Importando as bibliotecas necessárias.***

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

2. ***Carregue o dataset atualizado***

```
# Carregar o dataset
classificacao =
pd.read_csv('./dataset/dataset_classificacao_ajustado.csv')
```

3. ***Normalizando com a melhor normalização o conjunto de dados, que no caso teve melhoria com a normalização logarítmica.***

```
# Dividir em X (entradas) e y (saídas)
X = classificacao.drop(columns=['booking_status'], axis=1).values
y = classificacao['booking_status'].values

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, stratify=y, random_state=42)

# Normalização logarítmica
X_train_log = np.log1p(X_train)
X_test_log = np.log1p(X_test)

# Treinando e avaliando para diferentes valores de k
accuracies = []
k_values = range(1, 21)  # Testando k de 1 a 20
```

```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k,
metric='mahalanobis', metric_params={'VI':
np.linalg.inv(np.cov(X_train_log, rowvar=False))})
    knn.fit(X_train_log, y_train)
    y_pred = knn.predict(X_test_log)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
```

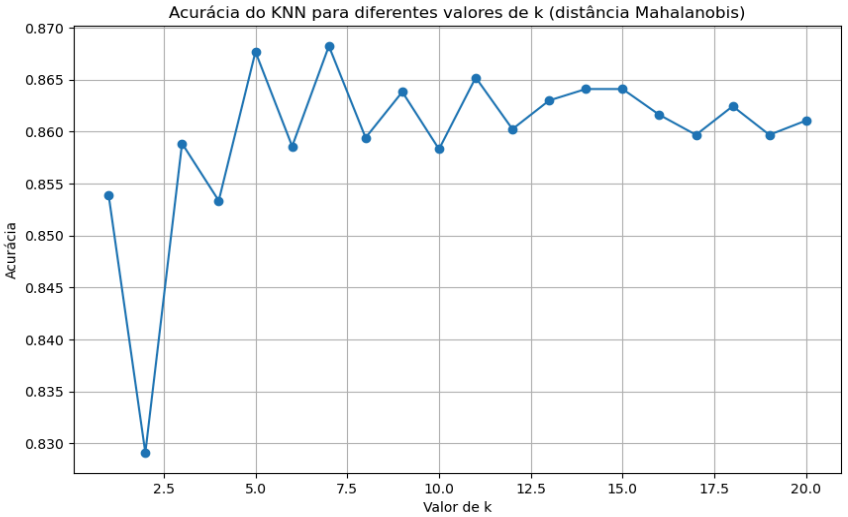
4. ***Plote do gráfico com a indicação do melhor k.***

```
# Plotando o gráfico com o melhor k
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.title('Acurácia do KNN para diferentes valores de k (distância
Mahalanobis)')
plt.xlabel('Valor de k')
plt.ylabel('Acurácia')
plt.grid(True)
plt.show()

# Indicar o melhor k
best_k = k_values[np.argmax(accuracies)]
print(f'O melhor valor de k é: {best_k} com uma acurácia de
{max(accuracies):.4f}')
```

3.4.2 Resultados

- *Para obter esse resultado de k=7, utilizamos a distância mahalanobis, que foi a melhor e também usamos a normalização logarítmica que teve melhor desempenho.*

- ***Registro do Gráfico***
- *Esse gráfico mostra a representação visual do melhor valor de K, que nesse caso seria 7, pois quando K=7 o valor fica 0.865.*



-

3.5 Questão 5

Observamos o dataset de regressão e realizamos o pré-processamento. Verificamos qual atributo será o alvo para regressão no dataset e fizemos uma análise de qual atributo é mais relevante para realizar a regressão do alvo escolhido. E fizemos a comprovação via gráfico. Não foi necessário remover colunas insignificantes e não tinha valores nulos.

- ***charges: Custos médicos individuais cobrados pelo seguro de saúde***

 # Importantando as bibliotecas necessárias

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

 # Importando o dataset

```
regressao = pd.read_csv('./dataset/data_regressao.csv')
```

 # Exploração inicial do dataset

```
print('Exploração do dataset')
print(regressao.head(10))
print('\nVerificação do número de linhas e colunas:',
regressao.shape)
print('\nInformações do dataset:')
print(regressao.info())
print('\nEstatísticas do dataset:')
print(regressao.describe().T)
```

 # Verificação de valores nulos e duplicatas

```
print('\nVerificação de valores nulos:')
print(regressao.isna().sum())
print('\nVerificação de duplicatas:')
print(regressao.duplicated().sum())
```

 # Verificação de valores categóricos

```
print('\nValores únicos por categoria:')
print('Sex:', regressao['sex'].value_counts())
print('Smoker:', regressao['smoker'].value_counts())
print('Region:', regressao['region'].value_counts())
```

 # Visualizações de distribuição

```
sns.countplot(x='sex', data=regressao)
plt.title('Distribuição por Sexo')
plt.show()

sns.countplot(x='smoker', data=regressao)
plt.title('Distribuição por Status de Fumante')
```

```
plt.show()

sns.countplot(x='region', data=regressao)
plt.title('Distribuição por Região')
plt.show()
```

 # Transformar colunas categóricas em numéricas

```
label_encoder = LabelEncoder()
regressao['sex'] = label_encoder.fit_transform(regressao['sex'])
regressao['smoker'] =
label_encoder.fit_transform(regressao['smoker'])
regressao['region'] =
label_encoder.fit_transform(regressao['region'])
```

 # Verificar correlação

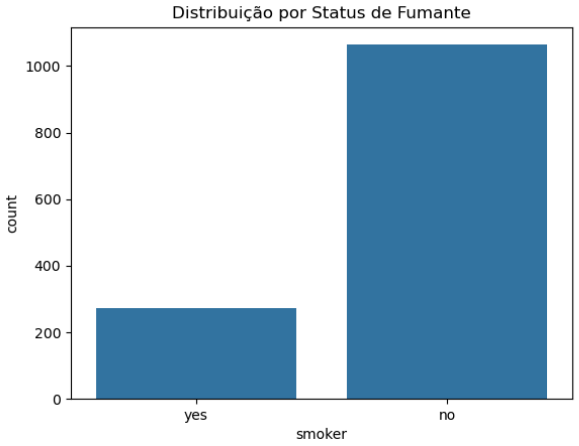
```
plt.figure(figsize=(10, 6))
sns.heatmap(regressao.corr(), annot=True, fmt=".2f",
cmap="coolwarm")
plt.title('Matriz de Correlação')
plt.show()

# Análise gráfica de variáveis relevantes
sns.scatterplot(x='bmi', y='charges', data=regressao)
plt.title('Relação entre BMI e Charges')
plt.show()

sns.scatterplot(x='age', y='charges', data=regressao)
plt.title('Relação entre Idade e Charges')
plt.show()

sns.boxplot(x='smoker', y='charges', data=regressao)
plt.title('Impacto do Status de Fumante em Charges')
plt.show()
```

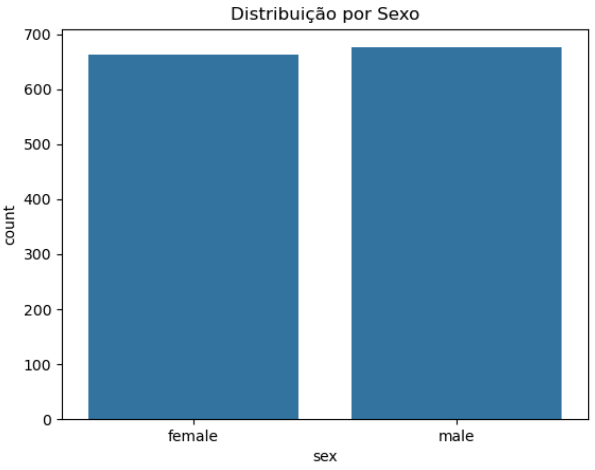
 # Salvar o Dataset atualizado:

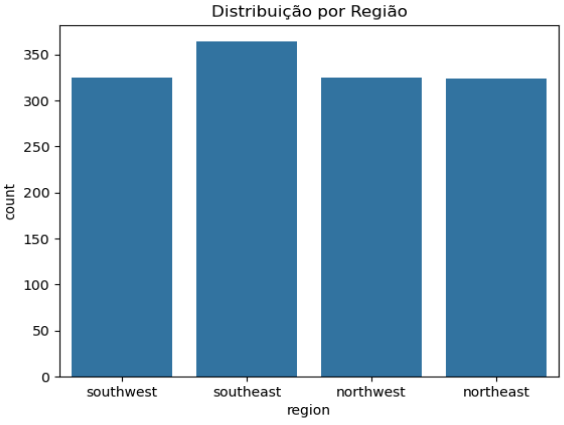
```
regressao.to_csv('./dataset_regressao_ajustado.csv', index=False)
```

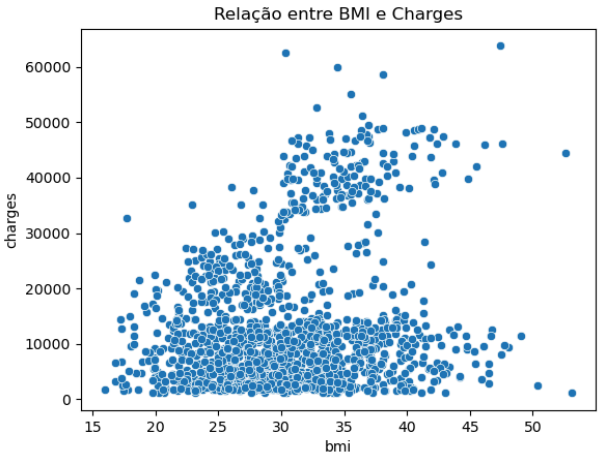
3.5.3 Resultados

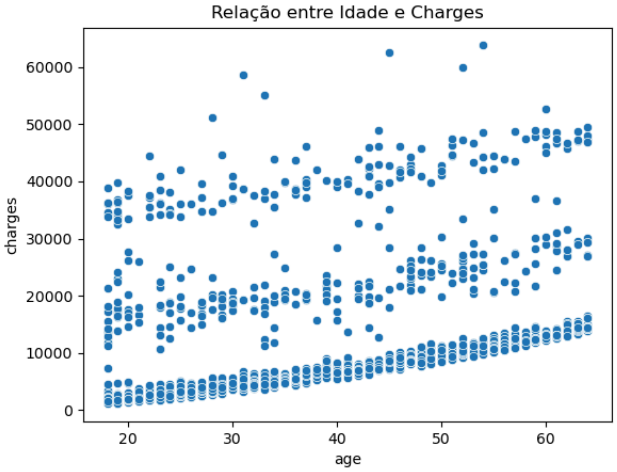
- *Esses gráficos representam de forma visual, os dados das colunas do dataset pré-processado. E também representam as relações entre a coluna alvo - 'charges', com as outras colunas.*

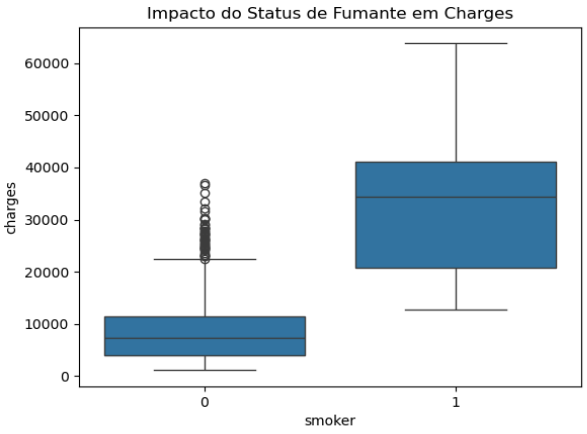












- *E através dessas análises gráficas foi possível ver que o atributo 'fumantes' é a categoria que tem mais influência sobre a coluna alvo - 'charges'.*

3.6 Questão 6

Utilizando o atributo mais relevante calculado na questão 5, implementamos uma regressão linear utilizando somente este atributo mais relevante, para predição do atributo alvo determinado na questão 5 também. Mostramos o gráfico da reta de regressão em conjunto com a nuvem de atributo e determinamos também os valores: RSS, MSE, RMSE e R Squared para esta regressão baseada somente no atributo mais relevante.

1. ***Importe as bibliotecas necessárias.***

```
import pandas as pd
import numpy as np
import

matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

2. ***Carregando o dataset atualizado***

```
# Importando o dataset
regressao                                                    =
pd.read_csv('./dataset/dataset_regressao_ajustado.csv')

# Escolhendo o atributo mais relevante baseado na análise de
correlação
X = regressao[['smoker']]    # Usando 'smoker' como variável
independente
y = regressao['charges']  # Variável dependente
```

3. ***Normalizando o conjunto de dados com a melhor normalização, que nesse caso foi a StandarScaler, já que houve uma melhoria.***

```
# Normalização (aplicando a melhor normalização, StandardScaler)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Dividir os dados em treino e teste
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# 5. Treinar o modelo de regressão linear
modelo = LinearRegression()
modelo.fit(X_train, y_train)

# 6. Previsões
y_pred = modelo.predict(X_test)

# 7. Calcular as métricas
# RSS (Residual Sum of Squares)
RSS = np.sum((y_test - y_pred) ** 2)

# MSE (Mean Squared Error)
MSE = mean_squared_error(y_test, y_pred)

# RMSE (Root Mean Squared Error)
RMSE = np.sqrt(MSE)

# R² (Coeficiente de Determinação)
R2 = r2_score(y_test, y_pred)

# Exibir as métricas
print(f"\nRSS: {RSS}")
print(f"MSE: {MSE}")
print(f"RMSE: {RMSE}")
print(f"R²: {R2}")
```

4. ***Plote do gráfico com a indicação do melhor k.***

```
# Plotar o gráfico com a reta de regressão
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Nuvem de pontos')
plt.plot(X_test, y_pred, color='red', label='Reta de regressão')
plt.title('Reta de Regressão Linear')
plt.xlabel('Smoker (Fumantes)')
plt.ylabel('Charges')
plt.legend()
```

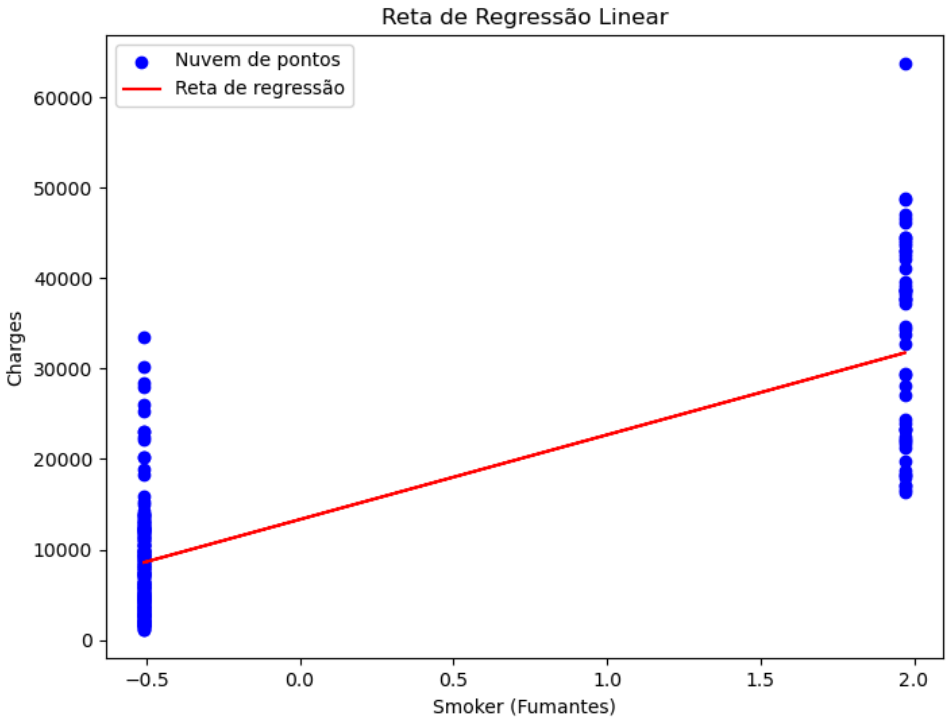
```
plt.show()
```

3.6.2 Resultados

- *O atributo mais relevante foi o 'smoker' (fumante).*
- *Então foi feito o cálculo com base nesse atributo e na coluna alvo, 'charges'.*
- *Os resultados obtidos indicam que a melhor categoria foi realmente o atributo 'smoker'.*
- *E que o valor que obteve melhor desempenho foi o RSS, já que apresentou um valor relativamente bom, comparados às outras, e também cabe destacar que foi feito um teste com as outras categorias do dataset, e o resultado menos pior foi o da categoria 'smoker'.*

```
C:\Users\win11\anaconda3\envs\IA--SI\python.exe C:\Users\win11\Desktop\IA-SI\IA--SI\AV1\questao6.py

RSS: 14135918546.97704
MSE: 52745964.72752627
RMSE: 7262.6417182404275
R²: 0.6602486589056528

Process finished with exit code 0
```

- ***Registro do Gráfico***



3.7 Questão 7

Implementamos um modelo de regressão linear para resolver o problema a seguir.

Seguimos os passos abaixo:

1. ***Divisão de Dados: Implementamos manualmente o método k-fold e a validação (cross-validation).***

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Importar o dataset
regressao =
pd.read_csv('./dataset/dataset_regressao_ajustado.csv')

# Escolher o atributo mais relevante para o modelo
X = regressao[['smoker']]  # Variável independente
y = regressao['charges']  # Variável dependente

# Normalização dos dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Implementação do K-Fold Cross-Validation manualmente
k = 5  # Definir o número de folds
fold_size = len(X) // k

# Inicializando o armazenamento dos resultados das métricas
results = {'LinearRegression': {'RSS': [], 'MSE': [], 'RMSE': [],
'R2': []},
           'Ridge': {'RSS': [], 'MSE': [], 'RMSE': [], 'R2': []},
           'Lasso': {'RSS': [], 'MSE': [], 'RMSE': [], 'R2': []}}

for i in range(k):
    # Dividir os dados em treino e validação
    validation_start = i * fold_size
```

```
    validation_end = (i + 1) * fold_size if i != k - 1 else len(X)

X_train = np.concatenate([X_scaled[:validation_start],
X_scaled[validation_end:]], axis=0)
    X_valid = X_scaled[validation_start:validation_end]

    y_train = np.concatenate([y[:validation_start],
y[validation_end:]], axis=0)
    y_valid = y[validation_start:validation_end]
```

2. ***Modelos a Serem Testados: Realizamos uma regressão linear clássica, Ridge e Lasso.***

```
   # Modelos a serem testados
(Regressão Linear, Ridge, Lasso)

models = {
        'LinearRegression': LinearRegression(),
        'Ridge': Ridge(alpha=1),
        'Lasso': Lasso(alpha=1)
    }

    for model_name, model in models.items():
        # Treinar o modelo
        model.fit(X_train, y_train)

        # Fazer previsões
        y_pred = model.predict(X_valid)


        # Calcular as métricas
        RSS, MSE, RMSE, R2 = calculate_metrics(y_valid, y_pred)

        # Armazenar os resultados nas listas correspondentes
        results[model_name]['RSS'].append(RSS)
        results[model_name]['MSE'].append(MSE)
        results[model_name]['RMSE'].append(RMSE)
```

```
        results[model_name]['R2'].append(R2)
```

3. ***Métricas de Avaliação: Implementamos manualmente as seguintes métricas e calculamos para cada modelo:***

   RSS (Soma dos Quadrados Residuais);

   MSE (Erro Quadrático Médio);

   RMSE (Erro Quadrático Médio);

   R^2 (Coeficiente de Determinação).

```
# Função para calcular as métricas manualmente
def calculate_metrics(y_true, y_pred):
    # RSS (Residual Sum of Squares)
    RSS = np.sum((y_true - y_pred) ** 2)

    # MSE (Mean Squared Error)
    MSE = mean_squared_error(y_true, y_pred)

    # RMSE (Root Mean Squared Error)
    RMSE = np.sqrt(MSE)

    # R² (Coeficiente de Determinação)
    R2 = r2_score(y_true, y_pred)

    return RSS, MSE, RMSE, R2

# Calcular a média das métricas para cada modelo
averages = {}
for model_name, metrics in results.items():
    averages[model_name] = {metric: np.mean(values) for metric,
values in metrics.items()}

# Exibir os resultados
for model_name, avg_metrics in averages.items():
    print(f"\n{model_name}:")
    for metric, avg_value in avg_metrics.items():
        print(f"Average {metric}: {avg_value}")
```

4. ***Análise de Desempenho: Comparamos os resultados das métricas para os três tipos de regressão linear e identificamos qual deles apresentou o melhor desempenho.***

```
# 6. Comparar os modelos
best_model = min(averages, key=lambda model:
averages[model]['RMSE'])  # Escolher o modelo com o menor RMSE
print(f"\nMelhor modelo: {best_model}")
```

3.7.2 Resultados

- *Com base nos resultados, a melhor métrica para os três tipos de regressão linear foi a Ridge.*
- *Apesar da diferença ser mínima, as outras métricas ficaram muito próximas.*

```
C:\Users\win11\anaconda3\envs\IA--SI\python.exe C:\Users\win11\Desktop\IA-SI\IA--SI\AV1\questao7.py

LinearRegression:
Average RSS: 14942646648.894917
Average MSE: 55844365.17821733
Average RMSE: 7470.655619239505
Average R2: 0.6176787336254209

Ridge:
Average RSS: 14942563100.516687
Average MSE: 55844041.74214192
Average RMSE: 7470.642739353636
Average R2: 0.6176811511319622

Lasso:
Average RSS: 14942636646.21928
Average MSE: 55844326.54745485
Average RMSE: 7470.654024994988
Average R2: 0.6176790206033634

Melhor modelo: Ridge
```

4. Conclusões

Após a aplicação de machine learning em ambos os projetos, foi perceptível a sua grande colaboração para todos. Consideramos sim os resultados satisfatórios, pois com essas aplicações conseguimos alcançar o objetivo que foi proposto, além do aprendizado que conseguimos, que permitiu melhorar a capacidade de previsão e tomadas de decisões. Os resultados obtidos atenderam plenamente às expectativas definidas para o projeto. Todos os processos realizados, desde o pré-processamento dos datasets até a aplicação dos modelos de classificação e regressão, foram bem-sucedidos e apresentaram resultados coerentes com os objetivos iniciais.

5. Próximos passos

Para os próximos passos, sugere-se o aprimoramento dos modelos com o teste de árvores de decisão e redes neurais, além do ajuste fino de hiperparâmetros. Também é recomendada a exploração de métricas adicionais, como F1-score, precisão e recall, para uma análise mais completa, bem como a aplicação do pipeline a datasets maiores, utilizando técnicas como PCA para verificar escalabilidade. A automatização do fluxo de trabalho e a documentação detalhada dos resultados em relatórios e apresentações visuais também são passos fundamentais para consolidar os avanços e facilitar futuras implementações.