Atividade 2 - base de dados Hotel Reservations kAGGLE

Summary

1. **Visualização de Dados**
   - Apresentar um mapa de calor (heatmap) entre todas as variáveis numéricas.
2. **Transformação de Dados**
   - Efetuar as devidas transformações nos atributos categóricos.
3. **Normalização de Dados**
   - Normalizar por Z-Score (Standard Scaler).

Modelos de machine learning

4. **RandomForestClassifier DecisionTreeClassifier KNeighborsClassifier**
   - evaluation metrics
5. **Pipeline Sklearn**
6. **Feature Selection techniques**
   - "Mutual Information": SelectKBest(mutual_info_classif, k=10)
   - "ANOVA": SelectKBest(f_classif)
   - "Smart Correlated": SmartCorrelatedSelection
7. **Diagrama de Venn**

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#pre-processing
from sklearn.preprocessing  import LabelEncoder
from sklearn.preprocessing import StandardScaler
## Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
## Model evaluators
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, accuracy_score, recall_score
```

In [ ]:
```
# df = pd.read_csv('train.csv')
df = pd.read_csv('Hotel Reservations.csv')
df.head(5)
```

Out[ ]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_night
0	INN00001	2	0	1	
1	INN00002	2	0	2	
2	INN00003	1	0	2	
3	INN00004	2	0	0	
4	INN00005	2	0	1	

In [ ]:
```
df.shape
```

Out[ ]: (36275, 19)

In [ ]:
```
df.info()
```

```
<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
```

Limpeza dos Dados

Dados Faltantes

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Booking_ID                              0
no_of_adults                            0
no_of_children                          0
no_of_weekend_nights                    0
no_of_week_nights                       0
type_of_meal_plan                       0
required_car_parking_space              0
room_type_reserved                      0
lead_time                               0
arrival_year                            0
arrival_month                           0
arrival_date                            0
market_segment_type                     0
repeated_guest                          0
no_of_previous_cancellations            0
no_of_previous_bookings_not_canceled    0
avg_price_per_room                      0
no_of_special_requests                  0
booking_status                          0
dtype: int64
```

pré-processing

Vou dropar o Booking ID e o ano da reserva para evitar overfitting, já que o ID poderia acabar indicando uma relação entre as reservas e o ano é uma feature que não vai se repetir nos anos seguintes, não sendo útil para a identificação de cancelamentos futuros

In [ ]:
```
df = df.drop('Booking_ID', axis =1)
df = df.drop('arrival_year', axis = 1)
```

Transformação dados categóricos

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
for i in df.columns:
  if(df[i].dtype=='object'):
    print(f'{i}: {df[i].unique()}')
```

```
type_of_meal_plan: ['Meal Plan 1' 'Not Selected' 'Meal Plan 2' 'Meal Plan
3']
room_type_reserved: ['Room_Type 1' 'Room_Type 4' 'Room_Type 2' 'Room_Type
6' 'Room_Type 5'
 'Room_Type 7' 'Room_Type 3']
market_segment_type: ['Offline' 'Online' 'Corporate' 'Aviation' 'Complemen
tary']
booking_status: ['Not_Canceled' 'Canceled']
```

In [ ]:
```
columns_to_encode = [ 'market_segment_type', 'type_of_meal_plan', 'room_t
encoder = LabelEncoder()
for column in columns_to_encode:
    df[column] = encoder.fit_transform(df[column])

# Applying get_dummies to encode
#df = pd.get_dummies(df, columns=['type_of_meal_plan', 'room_type_reserve
```

In [ ]:
```
# Transforma os dados categóricos
labelencoder = LabelEncoder()
df['booking_status'] = labelencoder.fit_transform(df['booking_status'])
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 17 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   no_of_adults                          36275 non-null  int64
 1   no_of_children                        36275 non-null  int64
 2   no_of_weekend_nights                  36275 non-null  int64
 3   no_of_week_nights                     36275 non-null  int64
 4   type_of_meal_plan                     36275 non-null  int64
 5   required_car_parking_space            36275 non-null  int64
 6   room_type_reserved                    36275 non-null  int64
 7   lead_time                             36275 non-null  int64
 8   arrival_month                         36275 non-null  int64
 9   arrival_date                          36275 non-null  int64
 10  market_segment_type                   36275 non-null  int64
 11  repeated_guest                        36275 non-null  int64
 12  no_of_previous_cancellations          36275 non-null  int64
 13  no_of_previous_bookings_not_canceled  36275 non-null  int64
 14  avg_price_per_room                    36275 non-null  float64
 15  no_of_special_requests                36275 non-null  int64
 16  booking_status                        36275 non-null  int64
dtypes: float64(1), int64(16)
memory usage: 4.7 MB
```

Normalização de atributos numéricos

In [ ]:
```
df.head()
```

Out[ ]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_m
0	2	0	1	2	
1	2	0	2	3	
2	1	0	2	1	
3	2	0	0	2	
4	2	0	1	1	

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['no_of_adults', 'no_of_children', 'no_of_weekend_nights',
       'no_of_week_nights', 'type_of_meal_plan', 'required_car_parking_s
pace',
       'room_type_reserved', 'lead_time', 'arrival_month', 'arrival_dat
e',
       'market_segment_type', 'repeated_guest', 'no_of_previous_cancella
tions',
       'no_of_previous_bookings_not_canceled', 'avg_price_per_room',
       'no_of_special_requests', 'booking_status'],
      dtype='object')
```

In [ ]:
```
scaler_cols = ['no_of_adults', 'no_of_children', 'no_of_weekend_nights',
       'no_of_week_nights', 'required_car_parking_space', 'lead_time',
       'arrival_month', 'arrival_date', 'repeated_guest',
       'no_of_previous_cancellations', 'no_of_previous_bookings_not_cance
       'avg_price_per_room', 'no_of_special_requests']
#print(scaler_cols)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
ajuste = scaler.fit(df[scaler_cols])
df[scaler_cols] = ajuste.transform(df[scaler_cols])
```

In [ ]:
```
df.head()
```

Out[ ]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_m
0	0.298893	-0.26147	0.217401	-0.144803	
1	0.298893	-0.26147	1.365993	0.563972	
2	-1.628975	-0.26147	1.365993	-0.853578	
3	0.298893	-0.26147	-0.931190	-0.144803	
4	0.298893	-0.26147	0.217401	-0.853578	

Modelos de Machine Learning

Separação entre treino e teste

In [ ]:
```
# Everything except target variable
X = df.drop("booking_status", axis=1)

# Target variable
y = df['booking_status']
```

In [ ]:
```
# Random seed for reproducibility
np.random.seed(55)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Models

In [ ]:
```
forest = RandomForestClassifier()
```

```
decision_tree = DecisionTreeClassifier()
knn = KNeighborsClassifier()
```

In [ ]:
```
# Fit models
forest.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
knn.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier i ?

KNeighborsClassifier()
```

In [ ]:
```
# Predict and evaluate RandomForest
forest_y_pred = forest.predict(X_test)
print("RandomForest Training Score:", forest.score(X_train, y_train))
print("RandomForest Testing Score:", forest.score(X_test, y_test))
```

```
RandomForest Training Score: 0.9941764300482426
RandomForest Testing Score: 0.9051688490696072
```

In [ ]:
```
# Predict and evaluate DecisionTree
dt_y_pred = decision_tree.predict(X_test)
print("DecisionTree Training Score:", decision_tree.score(X_train, y_trai
print("DecisionTree Testing Score:", decision_tree.score(X_test, y_test))
```

```
DecisionTree Training Score: 0.99421088904204
DecisionTree Testing Score: 0.8693314955203308
```

In [ ]:
```
# Predict and evaluate KNN
knn_y_pred = knn.predict(X_test)
print("KNN Training Score:", knn.score(X_train, y_train))
print("KNN Testing Score:", knn.score(X_test, y_test))
```

```
KNN Training Score: 0.8932115782219159
KNN Testing Score: 0.8548587181254307
```

In [ ]:
```
# Function to print evaluation metrics
def print_evaluation_metrics(y_true, y_pred, model_name):
    print(f"\n{model_name} Evaluation Metrics")
    print("ACC: {:.3f}".format(accuracy_score(y_true, y_pred)))
    print("Recall: {:.2f}".format(recall_score(y_true, y_pred)))
    print("Precision: {:.2f}".format(precision_score(y_true, y_pred)))
    print("F1-score: {:.2f}".format(f1_score(y_true, y_pred)))
    print(classification_report(y_true, y_pred))
```

In [ ]:
```
# Print evaluation metrics for each model
print_evaluation_metrics(y_test, forest_y_pred, "RandomForest")
print_evaluation_metrics(y_test, dt_y_pred, "DecisionTree")
print_evaluation_metrics(y_test, knn_y_pred, "KNN")
```

```
RandomForest Evaluation Metrics
ACC: 0.905
Recall: 0.94
Precision: 0.92
F1-score: 0.93
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      2266
           1       0.92      0.94      0.93      4989

    accuracy                           0.91      7255
   macro avg       0.89      0.88      0.89      7255
weighted avg       0.90      0.91      0.90      7255


DecisionTree Evaluation Metrics
ACC: 0.869
Recall: 0.89
Precision: 0.91
F1-score: 0.90
              precision    recall  f1-score   support

           0       0.78      0.82      0.80      2266
           1       0.91      0.89      0.90      4989

    accuracy                           0.87      7255
   macro avg       0.85      0.85      0.85      7255
weighted avg       0.87      0.87      0.87      7255


KNN Evaluation Metrics
ACC: 0.855
Recall: 0.90
Precision: 0.89
F1-score: 0.90
              precision    recall  f1-score   support

           0       0.78      0.75      0.76      2266
           1       0.89      0.90      0.90      4989

    accuracy                           0.85      7255
   macro avg       0.83      0.83      0.83      7255
weighted avg       0.85      0.85      0.85      7255
```

Pipeline Sklearn

- Permite a criação de diferentes combinações de técnicas
- https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, mutual_info_classif, f
from sklearn.feature_selection import SequentialFeatureSelector as SFS, R
```

In [ ]:
```
#!pip install feature-engine
```

In [ ]:
```
from feature_engine.selection import SmartCorrelatedSelection
```

Separação entre treino e teste

In [ ]:
```
# Everything except target variable
X = df.drop("booking_status", axis=1)
# Target variable
y = df['booking_status']
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Configuração das técnicas que serão utilizadas

In [ ]:
```
# Define classifiers
classifiers = {
    "DecisionTree": DecisionTreeClassifier(),
    "KNN": KNeighborsClassifier(),
    "RandomForest": RandomForestClassifier(n_estimators=100)
}
```

In [ ]:
```
# Define feature selection techniques
feature_selection_techniques = {
    "Mutual Information": SelectKBest(mutual_info_classif, k=11),
    "ANOVA": SelectKBest(f_classif, k=11),
    'SmartCorrelatedGroups': SmartCorrelatedSelection(variables=None, met
}
```

In [ ]:
```
# Armazenamento dos pipelines ajustados
pipelines = {}
results = {}

# Loop through classifiers and feature selection techniques
for clf_name, clf in classifiers.items():
    for fs_name, fs in feature_selection_techniques.items():
        pipeline_name = f"{clf_name} with {fs_name}"
        # Define and fit pipeline
        pipeline = Pipeline([('feature_selection', fs), ('classifier', cl
        pipeline.fit(X_train, y_train)
        pred = pipeline.predict(X_test)

        # Calculate accuracy and print results
        acc = accuracy_score(y_test, pred)
        print(f"{pipeline_name} Accuracy: {acc}")
        results[pipeline_name] = acc

        # Store the fitted pipeline for later analysis
        pipelines[pipeline_name] = pipeline
```

```
DecisionTree with Mutual Information Accuracy: 0.8639161995773225
DecisionTree with ANOVA Accuracy: 0.8482955067536525
DecisionTree with SmartCorrelatedGroups Accuracy: 0.8598731967288431
KNN with Mutual Information Accuracy: 0.8496738031792704
KNN with ANOVA Accuracy: 0.8465496646145364
KNN with SmartCorrelatedGroups Accuracy: 0.8441606174767987
RandomForest with Mutual Information Accuracy: 0.8951575852246623
RandomForest with ANOVA Accuracy: 0.8800882109712396
RandomForest with SmartCorrelatedGroups Accuracy: 0.9009464302122576
```

Métricas de avaliação

In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import (recall_score,
                             accuracy_score,
                             precision_score,
                             f1_score)
```

In [ ]:
```
# Initialize a dictionary to store predictions
predictions = {}

# Assuming the loop and pipeline setup from the previous response here
# After fitting each pipeline, store predictions
for clf_name, clf in classifiers.items():
    for fs_name, fs in feature_selection_techniques.items():
        pipeline_name = f"{clf_name} with {fs_name}"
        # Fit and predict inside the loop as before
        pipeline.fit(X_train, y_train)
        pred = pipeline.predict(X_test)
        predictions[pipeline_name] = pred  # Store predictions

# Now, calculate and print metrics for each set of predictions
for name, pred in predictions.items():
    print(f">> Metrics for: {name}")
    print("ACC: {:.3f}".format(accuracy_score(y_test, pred)))
    print("Recall: {:.2f}".format(recall_score(y_test, pred, average='bin
    print("Precision: {:.2f}".format(precision_score(y_test, pred, averag
    print("F1-score: {:.2f}".format(f1_score(y_test, pred, average='binar
    print()  # Print a blank line for readability
```

```
>> Metrics for: DecisionTree with Mutual Information
ACC: 0.897
Recall: 0.94
Precision: 0.91
F1-score: 0.92

>> Metrics for: DecisionTree with ANOVA
ACC: 0.898
Recall: 0.94
Precision: 0.91
F1-score: 0.92

>> Metrics for: DecisionTree with SmartCorrelatedGroups
ACC: 0.897
Recall: 0.94
Precision: 0.91
F1-score: 0.92

>> Metrics for: KNN with Mutual Information
ACC: 0.897
Recall: 0.94
Precision: 0.91
F1-score: 0.92

>> Metrics for: KNN with ANOVA
ACC: 0.898
Recall: 0.94
Precision: 0.91
F1-score: 0.92

>> Metrics for: KNN with SmartCorrelatedGroups
ACC: 0.899
Recall: 0.94
Precision: 0.91
F1-score: 0.93

>> Metrics for: RandomForest with Mutual Information
ACC: 0.897
Recall: 0.95
Precision: 0.90
F1-score: 0.92

>> Metrics for: RandomForest with ANOVA
ACC: 0.898
Recall: 0.95
Precision: 0.90
F1-score: 0.92

>> Metrics for: RandomForest with SmartCorrelatedGroups
ACC: 0.898
Recall: 0.94
Precision: 0.91
F1-score: 0.92
```

Similaridade das Features

In [ ]:
```
features_by_selector = {}
```

```
for name, pipeline in pipelines.items():
    print(f"Processing pipeline: {name}")  # Debug print to show which pi
    feature_selection_step = pipeline.named_steps['feature_selection']
    if hasattr(feature_selection_step, 'get_feature_names_out'):
        # For methods that directly support
        feature_names = feature_selection_step.get_feature_names_out(inpu
    elif hasattr(feature_selection_step, 'get_support'):
        # For methods that provide a boolean mask
        selected_mask = feature_selection_step.get_support()
        feature_names = X_train.columns[selected_mask].tolist()
    else:
        feature_names = None

    features_by_selector[name] = feature_names
```

```
Processing pipeline: DecisionTree with Mutual Information
Processing pipeline: DecisionTree with ANOVA
Processing pipeline: DecisionTree with SmartCorrelatedGroups
Processing pipeline: KNN with Mutual Information
Processing pipeline: KNN with ANOVA
Processing pipeline: KNN with SmartCorrelatedGroups
Processing pipeline: RandomForest with Mutual Information
Processing pipeline: RandomForest with ANOVA
Processing pipeline: RandomForest with SmartCorrelatedGroups
```

Diagrama de Venn (Até 4 conjuntos)

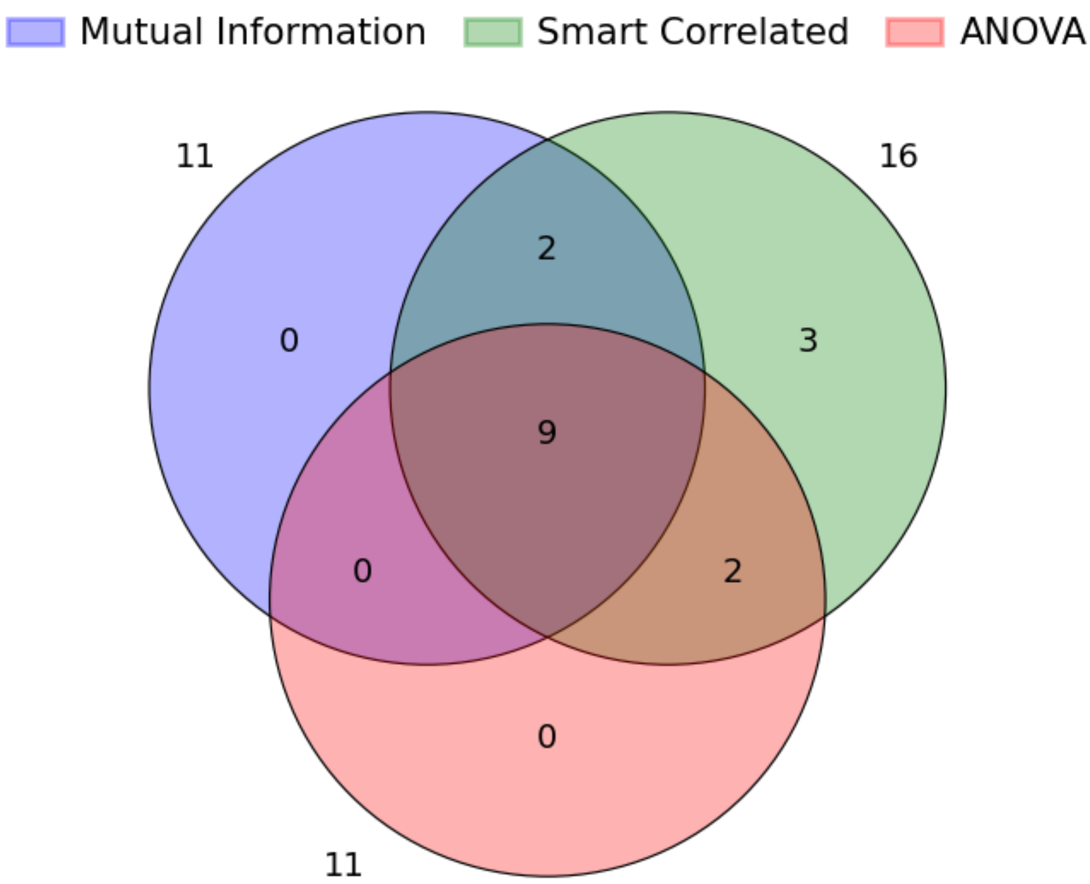
- Biblioteca Vennforest4Py
- https://pypi.org/project/venny4py/

In [ ]:
```
# Extrair os conjuntos de features de "Mutual Information", "Drop Correla
mutual_information_features = features_by_selector.get("RandomForest with
smart_correlated_features = features_by_selector.get("RandomForest with S
anova_correlated_features = features_by_selector.get("RandomForest with A
```

In [ ]:
```
# !pip install milkviz
#!pip install venny4py
```

In [ ]:
```
# import matplotlib_venn as venn
# import milkviz as mv
from venny4py.venny4py import *
```

In [ ]:
```
sets = {
    'Mutual Information': set(mutual_information_features),
    'Smart Correlated': set(smart_correlated_features),
    'ANOVA': set(anova_correlated_features)
}
# Gerar o diagrama de Venn
venny4py(sets=sets)
```



Features similares entre as 4 abordagens

In [ ]:
```
set(mutual_information_features).intersection(smart_correlated_features,a
```

Out[ ]:
```
{'avg_price_per_room',
 'lead_time',
 'market_segment_type',
 'no_of_adults',
 'no_of_previous_bookings_not_canceled',
 'no_of_special_requests',
 'no_of_week_nights',
 'no_of_weekend_nights',
 'required_car_parking_space'}
```

In [ ]: