```
# -*- coding: utf-8 -*-
"""Untitled5.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1pSWperL85Gt2eM1ZFre0iWWQjmbpkXyy
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# read the dataset and assign variable to it
booking = pd.read_csv("booking.csv")

# removing the ID column from the dataset
booking.drop(["Booking_ID"], axis=1, inplace=True)
booking.index = booking.index + 1

# list of items and columns
print(booking.shape)

# first five rows that are present in the dataset
booking.head(5)

# provides information about the type of values in the dataset
booking.info()

# check for null values in the dataset
print(booking.isnull().sum().sort_values(ascending=False))

# dropping outlier variables
outliers_cols = ["lead time", "average price"]
for column in outliers_cols:
    if booking[column].dtype in ["int64", "float64"]:
```

```
        q1 = booking[column].quantile(0.25)
        q3 = booking[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        booking = booking[
            (booking[column] >= lower_bound) & (booking[column] <= upper_bound)
        ]

# displaying box plot of all variables after dropping previously mentioned columsn
plt.figure(figsize=(12, 8))
sns.set(style="darkgrid")
booking_boxplot = sns.boxplot(data=booking, orient="h")
plt.title("Box Plot for Every Variable")
plt.show()
print(booking.shape)

# encoding the categorical variables into a binary numeric system
booking["booking status"] = booking["booking status"].replace("Canceled", 1)
booking["booking status"] = booking["booking status"].replace("Not_Canceled", 0)
booking.head()

# split the data into (day-month-year) & drop previous date format
booking = booking[~booking["date of reservation"].str.contains("-")]
booking["date of reservation"] = pd.to_datetime(booking["date of reservation"])

booking["day"] = booking["date of reservation"].dt.day
booking["month"] = booking["date of reservation"].dt.month
booking["year"] = booking["date of reservation"].dt.year

booking = booking.drop(columns=["date of reservation"])

# rounding it from col to int
booking["average price"] = booking["average price"].round().astype(int)

# applying one-hot encoding on variables with 'object' datatype in binary format
object_columns = booking.select_dtypes(include=["object"]).columns
booking = pd.get_dummies(booking, columns=object_columns)
booking = booking.replace({True: 1, False: 0})

# displays the correlation between variables
plt.figure(figsize=(12, 8))
sns.heatmap(booking.corr(), cmap="icefire", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()

# selecting the best features and dropping the rest
```

```
features = booking.drop(["booking status"], axis=1)
target = booking["booking status"]

k_best = SelectKBest(score_func=f_classif, k=10)

X = k_best.fit_transform(features, target)
y = target

# Get the indices of the selected features
selected_features_indices = k_best.get_support(indices=True)

# Get the scores associated with each feature
feature_scores = k_best.scores_

# Create a list of tuples containing feature names and scores
feature_info = list(zip(features.columns, feature_scores))

# Sort the feature info in descending order based on scores
sorted_feature_info = sorted(feature_info, key=lambda x: x[1], reverse=True)

for feature_name, feature_score in sorted_feature_info[:10]:
    print(f"{feature_name}: {feature_score:.2f}")

selected_features_df = features.iloc[:, selected_features_indices]
selected_features_df.head()

# hold the scores
scores = {}

# splitting the dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, target, test_size=0.2, random_state=5
)

# perform knn algorithm and print its performance metrics
knn = KNeighborsClassifier()

params = {"n_neighbors": np.arange(1, 10)}

grid_search = GridSearchCV(knn, param_grid=params, cv=5)
grid_search.fit(X_train, y_train)
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Score: {grid_search.best_score_}")

best_knn = grid_search.best_estimator_

y_pred = best_knn.predict(X_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
scores["KNN"] = accuracy_score(y_test, y_pred)
print("-----------------------------------------------------------")
print(f"Confusion Matrix: \n{confusion_matrix(y_test, y_pred)}")
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt=".0f")
print("-----------------------------------------------------------")
print(f"Classification Report: \n{classification_report(y_test, y_pred)}")

# perform decision tree classification and print its peformance metrics
dt = DecisionTreeClassifier()

params = {"max_depth": np.arange(0, 30, 5), "criterion": ["gini", "entropy"]}

grid_search = GridSearchCV(dt, param_grid=params, cv=5)
grid_search.fit(X_train, y_train)
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Score: {grid_search.best_score_}")

best_dt = grid_search.best_estimator_

y_pred = best_dt.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
scores["Decision Tree"] = accuracy_score(y_test, y_pred)
print("-----------------------------------------------------------")
print(f"Confusion Matrix: \n{confusion_matrix(y_test, y_pred)}")
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt=".0f")
print("-----------------------------------------------------------")
print(f"Classification Report: \n{classification_report(y_test, y_pred)}")

# print the scores for each model

for model, score in scores.items():
    print(f"{model}: {score:.4f}")
```