# Decision Tree

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Decision Tree**

**Guidelines:**

**1. An assignment submission is considered complete only when the correct and executable code(s) and documentation explaining the method and results are submitted. Failing to submit either of those will be considered an invalid submission and not a correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to the keys provided. (will be available only post the submission).**

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**



**2.1 Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

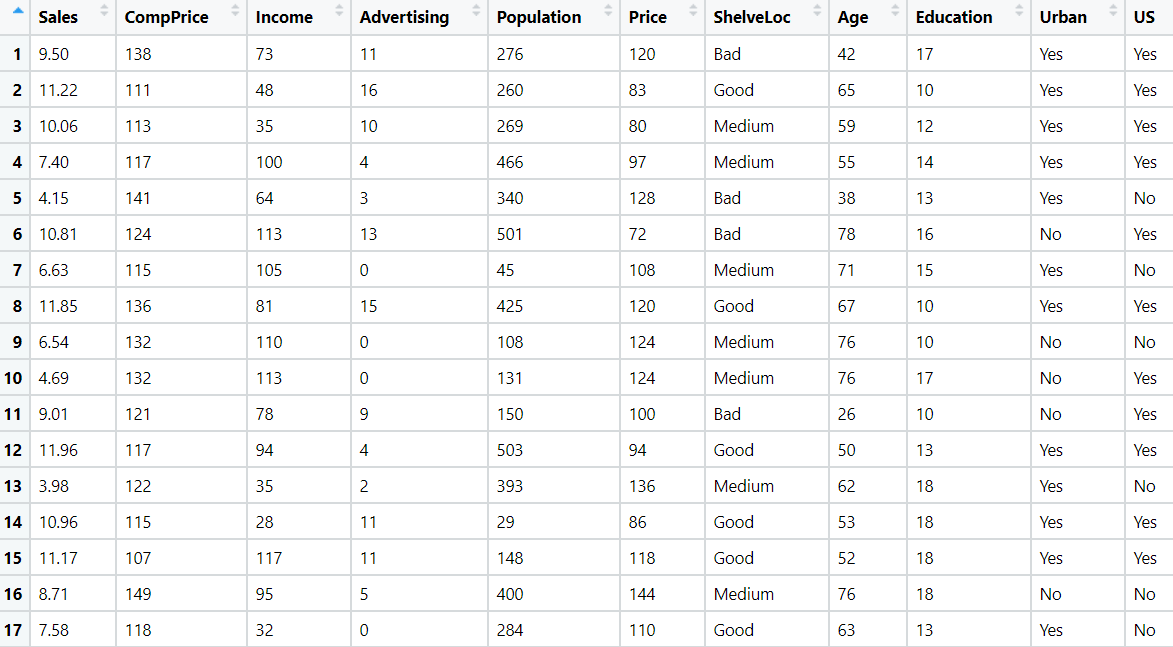
1. **Data Pre-processing**

**3.1 Data Cleaning, Feature Engineering, etc.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the cleaned data.**
   2. **Perform Decision Tree on the given dataset.**
   3. **Train and Test the data and perform cross-validation techniques, compare accuracies, and precision, and recall and explain them.**
   4. **Briefly explain the model output in the documentation.**
3. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

**Problem Statements:**

1. A cloth manufacturing company is interested to know about the different attributes contributing to high sales. Build a decision tree & random forest model with Sales as the target variable (first convert it into categorical variable).





**Code:**

'''CRISP-ML(Q):

1.a. Business problem: A cloth manufacturing company is interested to know about the different attributes contributing to high sales.

i. Business Objectives: Maximize the sales

ii. Business Constraints: Minimize the advertising

Success Criteria:

i. Business success criteria: Increase the efficient of marketing by 10%

ii. ML success criteria: Achieve an accuracy of at least 70%

iii. Economic success criteria: Increase the sales by at least by 20%

1.b. Data Collection: Bank -> 400 sales data, 11 variables (10 Inputs and 1 Ouput)

2. Data Preprocessing - Cleansing & EDA / Descriptive Analytics

3. Model Building - Experiment with different models alongside Hyperparameter tuning

4. Evaluation - Not just model evaluation based on accuracy but we also need

to evaluate business & economic success criteria

5. Model Deployment (Flask)

6. Monitoring & Maintenance (Prediction results to the database - MySQL / MS SQL)'''

# pip install sklearn\_pandas

# conda install graphviz

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import LabelEncoder

# from sklearn.preprocessing import OrdinalEncoder

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from feature\_engine.outliers import Winsorizer

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier as DT

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import GridSearchCV

import joblib

import pickle

from sqlalchemy import create\_engine, text

# MySQL Database connection

from urllib.parse import quote

# Creating engine which connect to MySQL

user = 'root' # user name

pw = '1234' # password

db = 'sales\_db' # database

# creating engine to connect database

engine = create\_engine(f"mysql+pymysql://{user}:{quote(pw)}@localhost/{db}")

'''

# MS SQL Database connection

engine = create\_engine("mssql://@{server}/{database}?driver={driver}"

.format(server = "360DIGITMG\SQLEXPRESS", # server name

database = "loanSales\_Category", # database

driver = "ODBC Driver 17 for SQL Server")) # driver name

'''

# Load the data into Python dataframe for bulk load to SQL

sales = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/DT Model with Flask/Assignment/Decision Tree/ClothCompany\_Data.csv")

# Load the dataframe into database

sales.to\_sql('sales', con = engine, if\_exists = 'replace', chunksize = 1000, index = False )

###########

# Read the data from the database

sql = 'select \* from sales'

data = pd.read\_sql\_query(text(sql), con = engine.connect())

data.columns

data.info()

# Checking for Null values

data.isnull().sum()

# Define the numerical ranges and corresponding labels for the categories

sales\_bins = [0, 5, 10, float('inf')]

sales\_labels = [ 'Low', 'Medium', 'High']

# Convert numerical "Sales" data to categorical using cut function

data['Sales\_Category'] = pd.cut(data['Sales'], bins=sales\_bins, labels=sales\_labels, right=False)

# Drop the original "Sales" column if needed

data.drop(columns=['Sales'], inplace=True)

# ### AutoEDA

##############

# sweetviz

##########

# pip install sweetviz

import sweetviz

my\_report = sweetviz.analyze([data, "data"])

my\_report.show\_html('Report1.html')

# D-Tale

########

# pip install dtale

# import dtale

# d = dtale.show(data)

# d.open\_browser()

###################

# Target variable categories

data['Sales\_Category'].unique()

data['Sales\_Category'].value\_counts()

# Data split into Input and Output

X = data.iloc[:, :10] # Predictors

y = data['Sales\_Category'] # Target

data.info()

# #### Separating Numeric and Non-Numeric columns

numeric\_features = X.select\_dtypes(exclude = ['object']).columns

numeric\_features

categorical\_features = X.select\_dtypes(include=['object']).columns

categorical\_features

# ### Data Preprocessing

# Numeric\_features

# ### Imputation to handle missing values

# ### MinMaxScaler to convert the magnitude of the columns to a range of 0 to 1

num\_pipeline = Pipeline(steps = [('impute', SimpleImputer(strategy = 'mean')),

('scale', MinMaxScaler())])

# ### Encoding - One Hot Encoder to convert Categorical data to Numeric values

# Categorical features

encoding\_pipeline = Pipeline([('onehot', OneHotEncoder(drop = 'first'))])

# Creating a transformation of variable with ColumnTransformer()

preprocessor = ColumnTransformer(transformers = [('num', num\_pipeline, numeric\_features),

('categorical', encoding\_pipeline, categorical\_features)])

imp\_enc\_scale = preprocessor.fit(X)

# #### Save the pipeline model using joblib

joblib.dump(imp\_enc\_scale, 'imp\_enc\_scale')

import os

os.getcwd()

cleandata = pd.DataFrame(imp\_enc\_scale.transform(X),

columns = imp\_enc\_scale.get\_feature\_names\_out())

cleandata

# Note: If you get any error then update the scikit-learn library version & restart the kernel to fix it

# ### Outlier Analysis

# Multiple boxplots in a single visualization.

# Columns with larger scales affect other columns.

# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

cleandata.iloc[:, 0:7].plot(kind = 'box', subplots = True, sharey = False, figsize = (10, 6))

'''sharey True or 'all': x- or y-axis will be shared among all subplots.

False or 'none': each subplot x- or y-axis will be independent.'''

# Increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

cleandata.iloc[:, 0:7].columns

# #### Outlier analysis: Columns 'months\_loan\_duration', 'amount', and 'age' are continuous, hence outliers are treated

winsor = Winsorizer(capping\_method = 'iqr', # choose IQR rule boundaries or gaussian for mean and std

tail = 'both', # cap left, right or both tails

fold = 1.5,

variables = list(cleandata.iloc[:, 0:7].columns))

outlier = winsor.fit(cleandata.iloc[:, 0:7])

# Save the winsorizer model

joblib.dump(outlier, 'winsor')

cleandata.iloc[:, 0:7] = outlier.transform(cleandata.iloc[:, 0:7])

# Clean data

cleandata

# Verify for outliers

cleandata.iloc[:, 0:7].plot(kind = 'box', subplots = True, sharey = False, figsize = (10, 6))

# increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

# Split data into train and test with Stratified Sample technique

# from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(cleandata, y,

test\_size = 0.2,

stratify = y, random\_state = 0)

# Proportion of Target variable categories are consistent across train and test

print(Y\_train.value\_counts()/320)

print("\n")

print(Y\_test.value\_counts()/80)

### Decision Tree Model

model = DT(criterion = 'entropy')

model.fit(X\_train, Y\_train)

# Prediction on Test Data

preds = model.predict(X\_test)

preds

# Accuracy

print(accuracy\_score(Y\_test, preds))

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames = ['Predictions'])

### Hyperparameter Optimization

# create a dictionary of all hyperparameters to be experimented

param\_grid = { 'criterion':['gini', 'entropy'], 'max\_depth': np.arange(3, 15)}

# Decision tree model

dtree\_model = DT()

# GridsearchCV with cross-validation to perform experiments with parameters set

dtree\_gscv = GridSearchCV(dtree\_model, param\_grid, cv = 5, scoring = 'accuracy',

return\_train\_score = False, verbose = 1)

# Train the model with Grid search optimization technique

dtree\_gscv.fit(X\_train, Y\_train)

# The best set of parameter values

dtree\_gscv.best\_params\_

# Model with best parameter values

DT\_best = dtree\_gscv.best\_estimator\_

DT\_best

# Prediction on Test Data

preds1 = DT\_best.predict(X\_test)

preds1

# Model evaluation

# Cross Table (Confusion Matrix)

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames= ['Predictions'])

# Accuracy

print(accuracy\_score(Y\_test, preds))

#####################

# Generate Tree visualization

'''Steps to install Graphviz tool

# conda install python-graphviz

# Note: If you use pip install graphviz, the graphviz executable sit on a

different path from your conda directory.

'''

import os

import graphviz

from sklearn import tree

predictors = list(cleandata.columns)

type(predictors)

dot\_data = tree.export\_graphviz(DT\_best, filled = True,

rounded = True,

feature\_names = predictors,

class\_names = ['Low\_Sales', 'Medium\_Sales', 'High\_Sales'],

out\_file = None)

# os.environ["PATH"] += os.pathsep + 'C:/Users/Lenovo?anaconda3/envs/python\_10/Lib/site-packages/Graphviz-10.0.1-win64/bin'

graph = graphviz.Source(dot\_data)

graph

#####################

# Prediction on Train Data

preds\_train = DT\_best.predict(X\_train)

preds\_train

# Confusion Matrix

pd.crosstab(Y\_train, preds\_train, rownames = ['Actual'], colnames = ['Predictions'])

# Accuracy

print(accuracy\_score(Y\_train, preds\_train))

# ### Save the Best Model with pickel library

pickle.dump(DT\_best, open('DT.pkl', 'wb'))

## Model Training with Cross Validation

from sklearn.model\_selection import cross\_validate

def cross\_validation(model, \_X, \_y, \_cv=5):

'''

Function to perform 5 Folds Cross-Validation

Parameters

----------

model: Python Class, Sales\_Category=None

This is the machine learning algorithm to be used for training.

\_X: array

This is the matrix of features.

\_y: array

This is the target variable.

\_cv: int, Sales\_Category=5

Determines the number of folds for cross-validation.

Returns

-------

The function returns a dictionary containing the metrics 'accuracy', 'precision',

'recall', 'f1' for both training set and validation set.

'''

\_scoring = ['accuracy', 'precision', 'recall', 'f1']

results = cross\_validate(estimator = model, X = \_X, y = \_y,

cv = \_cv, scoring = \_scoring,

return\_train\_score = True)

return pd.DataFrame({"Training Accuracy scores": results['train\_accuracy'],

"Mean Training Accuracy": results['train\_accuracy'].mean()\*100,

"Training Precision scores": results['train\_precision'],

"Mean Training Precision": results['train\_precision'].mean(),

"Training Recall scores": results['train\_recall'],

"Mean Training Recall": results['train\_recall'].mean(),

"Training F1 scores": results['train\_f1'],

"Mean Training F1 Score": results['train\_f1'].mean(),

"Validation Accuracy scores": results['test\_accuracy'],

"Mean Validation Accuracy": results['test\_accuracy'].mean()\*100,

"Validation Precision scores": results['test\_precision'],

"Mean Validation Precision": results['test\_precision'].mean(),

"Validation Recall scores": results['test\_recall'],

"Mean Validation Recall": results['test\_recall'].mean(),

"Validation F1 scores": results['test\_f1'],

"Mean Validation F1 Score": results['test\_f1'].mean()

})

# Alternate approach for Encoding categorical data - required to encode target variable

# from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

encoded\_y = label\_encoder.fit\_transform(Y\_train)

'''label\_encoder\_name\_mapping = dict(zip(label\_encoder.classes\_,

label\_encoder.transform(label\_encoder.classes\_)))

print("Mapping of Label Encoded Classes", label\_encoder\_name\_mapping, sep = "\n")

print("Label Encoded Target Variable", encoded\_y, sep = "\n")'''

# from sklearn.tree import DecisionTreeClassifier

decision\_tree\_result = cross\_validation(DT\_best, X\_train, encoded\_y, 5)

decision\_tree\_result

def plot\_result(x\_label, y\_label, plot\_title, train\_data, val\_data):

'''Function to plot a grouped bar chart showing the training and validation

results of the ML model in each fold after applying K-fold cross-validation.

Parameters

----------

x\_label: str,

Name of the algorithm used for training e.g 'Decision Tree'

y\_label: str,

Name of metric being visualized e.g 'Accuracy'

plot\_title: str,

This is the title of the plot e.g 'Accuracy Plot'

train\_result: list, array

This is the list containing either training precision, accuracy, or f1 score.

val\_result: list, array

This is the list containing either validation precision, accuracy, or f1 score.

Returns

-------

The function returns a Grouped Barchart showing the training and validation result

in each fold.

'''

# Set size of plot

plt.figure(figsize=(12, 6))

labels = ["1st Fold", "2nd Fold", "3rd Fold", "4th Fold", "5th Fold"]

X\_axis = np.arange(len(labels))

ax = plt.gca()

plt.ylim(0.40000, 1)

plt.bar(X\_axis - 0.2, train\_data, 0.4, color = 'blue', label = 'Training')

plt.bar(X\_axis + 0.2, val\_data, 0.4, color = 'red', label = 'Validation')

plt.title(plot\_title, fontsize = 30)

plt.xticks(X\_axis, labels)

plt.xlabel(x\_label, fontsize = 14)

plt.ylabel(y\_label, fontsize = 14)

plt.legend()

plt.grid(True)

plt.show()

# import matplotlib

# matplotlib.use('Qt5Agg')

# import logging

# logging.getLogger("matplotlib.font\_manager").setLevel(logging.ERROR)

model\_name = "Decision Tree"

plot\_result(model\_name,

"Accuracy",

"Accuracy scores in 5 Folds",

decision\_tree\_result["Training Accuracy scores"],

decision\_tree\_result["Validation Accuracy scores"])

**Output:**

data['Sales\_Category'].value\_counts()

Out[30]:

Sales\_Category

Medium 244

High 79

Low 77

Name: count, dtype: int64

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 400 entries, 0 to 399

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 CompPrice 400 non-null int64

1 Income 400 non-null int64

2 Advertising 400 non-null int64

3 Population 400 non-null int64

4 Price 400 non-null int64

5 ShelveLoc 400 non-null object

6 Age 400 non-null int64

7 Education 400 non-null int64

8 Urban 400 non-null object

9 US 400 non-null object

10 Sales\_Category 400 non-null category

dtypes: category(1), int64(7), object(3)

memory usage: 31.9+ KB

cleandata = pd.DataFrame(imp\_enc\_scale.transform(X),

columns = imp\_enc\_scale.get\_feature\_names\_out())

cleandata

Out[61]:

num\_\_CompPrice num\_\_Income ... categorical\_\_Urban\_Yes categorical\_\_US\_Yes

0 0.622449 0.525253 ... 1.0 1.0

1 0.346939 0.272727 ... 1.0 1.0

2 0.367347 0.141414 ... 1.0 1.0

3 0.408163 0.797980 ... 1.0 1.0

4 0.653061 0.434343 ... 1.0 0.0

.. ... ... ... ... ...

395 0.622449 0.878788 ... 1.0 1.0

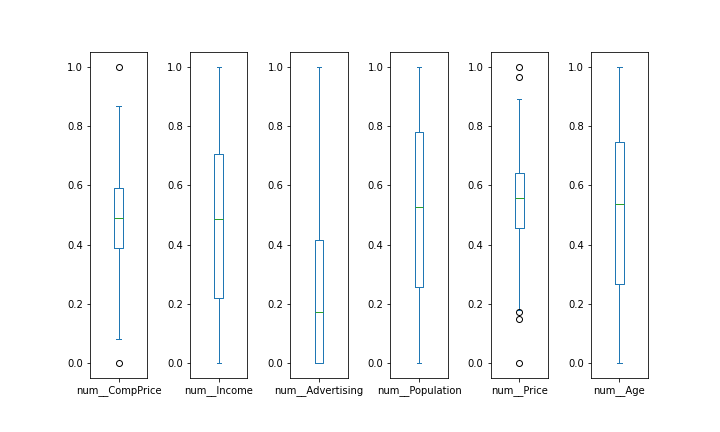
396 0.632653 0.020202 ... 0.0 1.0

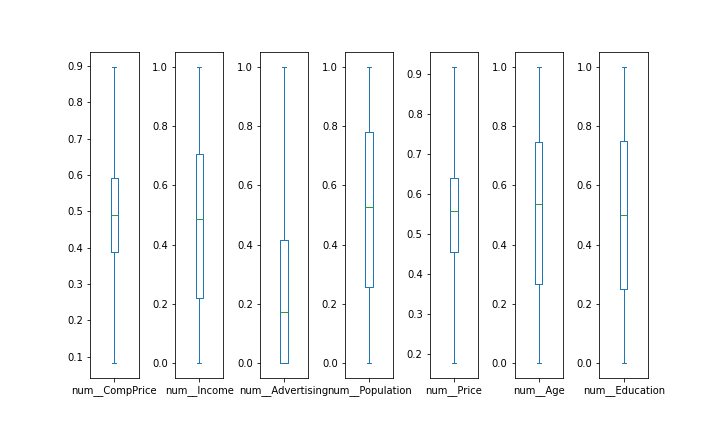
397 0.867347 0.050505 ... 1.0 1.0

398 0.234694 0.585859 ... 1.0 1.0

399 0.581633 0.161616 ... 1.0 1.0

[400 rows x 11 columns]





print(Y\_train.value\_counts()/320)

Sales\_Category

Medium 0.609375

High 0.196875

Low 0.193750

Name: count, dtype: float64

print("\n")

print(Y\_test.value\_counts()/80)

Sales\_Category

Medium 0.6125

High 0.2000

Low 0.1875

Name: count, dtype: float64

print(accuracy\_score(Y\_test, preds))

0.5875

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames = ['Predictions'])

Out[84]:

Predictions High Low Medium

Actual

Low 0 4 11

Medium 6 5 38

High 5 1 10

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames= ['Predictions'])

Out[98]:

Predictions High Low Medium

Actual

Low 0 4 11

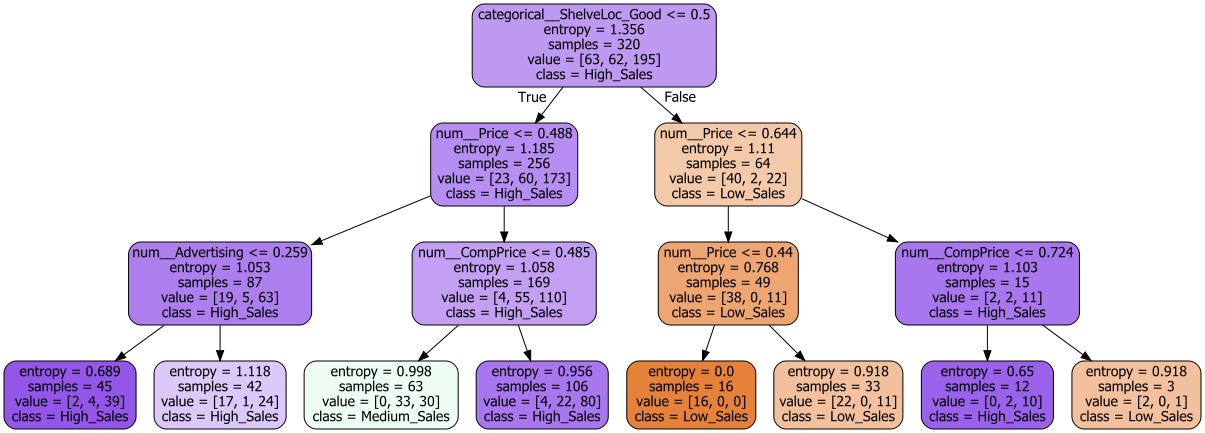
Medium 6 5 38

High 5 1 10

# Accuracy

print(accuracy\_score(Y\_test, preds))

0.5875



Predictions High Low Medium

Actual

Low 0 33 29

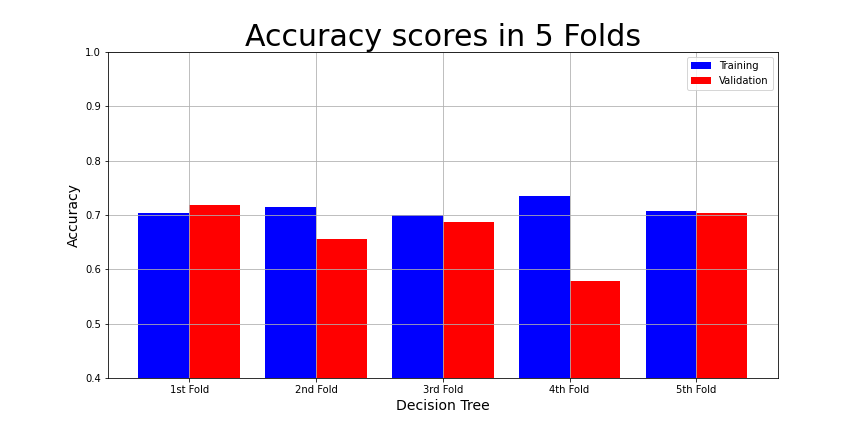
Medium 12 30 153

High 40 0 23

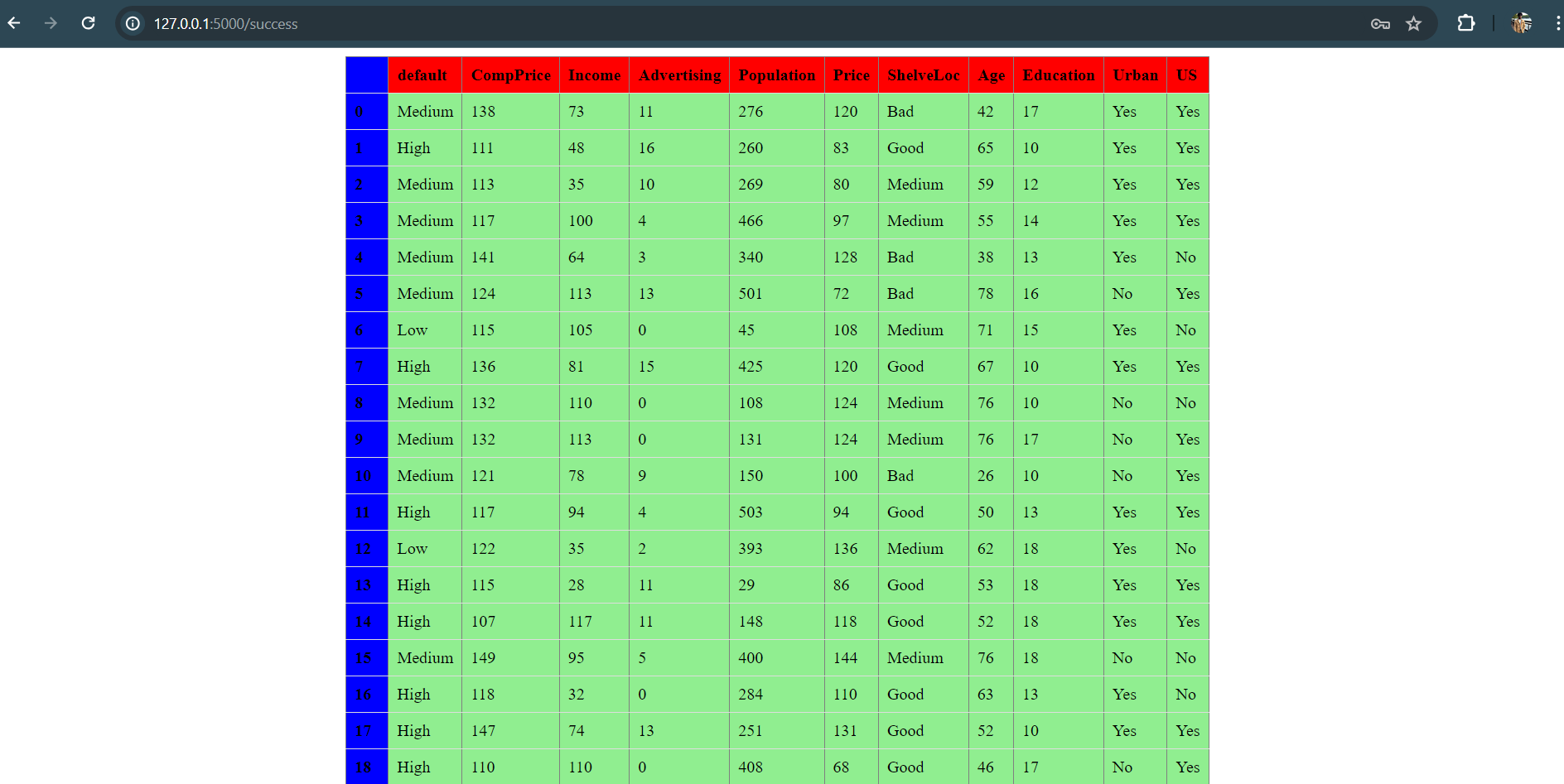
# Accuracy

print(accuracy\_score(Y\_train, preds\_train))

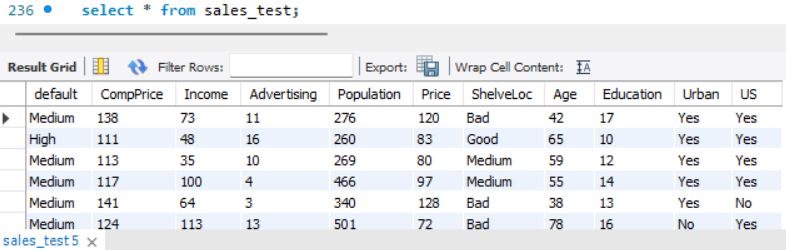
0.70625



**Deployment of Sales Prediction with Decision Tree model using Flask**

****

**Saving predicted Sales category in MySQL for Monitoring**

****