Py\_Predictions

Coffy Andrews-Guo

2022-07-16

# PY PREDICTIONS

**Decision Tree**

**KNN Algorithm**

# R packages  
suppressPackageStartupMessages(library(tidyverse)) #data transformations  
suppressPackageStartupMessages(library(reticulate)) #bridges Python and R

# Python libraries/modules  
import warnings  
warnings.simplefilter(action = 'ignore', category = FutureWarning)  
warnings.filterwarnings('ignore')  
def ignore\_warn(\*args, \*\*kwargs):  
 pass  
  
warnings.warn = ignore\_warn #ignore annoying warning (from sklearn and seaborn)  
  
import pandas as pd #data processing  
import numpy as np #linear algebra  
import openpyxl  
import pyreadr #read and write R RData and Rds files into/from pandas dataframes  
import matplotlib.pyplot as plt #create graphs  
from IPython import get\_ipython #to see the graphs in the output  
import seaborn as sns

### 3. Load Data and Save Original Data files

pydf = pd.read\_excel("StudentData - TO MODEL.xlsx")  
pydf.head(3)  
  
#remove the space from column name

## Brand Code Carb Volume Fill Ounces ... Alch Rel Carb Rel Balling Lvl  
## 0 B 5.340000 23.966667 ... 6.58 5.32 1.48  
## 1 A 5.426667 24.006667 ... 6.56 5.30 1.56  
## 2 B 5.286667 24.060000 ... 7.66 5.84 3.28  
##   
## [3 rows x 33 columns]

pydf.columns = pydf.columns.str.replace(' ', '')  
print("\n\nRemoved Space between Header Strings\n", pydf.head(3))

##   
##   
## Removed Space between Header Strings  
## BrandCode CarbVolume FillOunces ... AlchRel CarbRel BallingLvl  
## 0 B 5.340000 23.966667 ... 6.58 5.32 1.48  
## 1 A 5.426667 24.006667 ... 6.56 5.30 1.56  
## 2 B 5.286667 24.060000 ... 7.66 5.84 3.28  
##   
## [3 rows x 33 columns]

pyde = pd.read\_excel("StudentEvaluation- TO PREDICT.xlsx")  
pyde.head(3)  
  
#remove the space from column name

## Brand Code Carb Volume Fill Ounces ... Alch Rel Carb Rel Balling Lvl  
## 0 D 5.480000 24.033333 ... 6.56 5.34 1.48  
## 1 A 5.393333 23.953333 ... 7.14 5.58 3.04  
## 2 B 5.293333 23.920000 ... 6.52 5.34 1.46  
##   
## [3 rows x 33 columns]

pyde.columns = pyde.columns.str.replace(' ', '')  
print("\n\n Removed Space between Header Strings\n", pyde.head(3))

##   
##   
## Removed Space between Header Strings  
## BrandCode CarbVolume FillOunces ... AlchRel CarbRel BallingLvl  
## 0 D 5.480000 24.033333 ... 6.56 5.34 1.48  
## 1 A 5.393333 23.953333 ... 7.14 5.58 3.04  
## 2 B 5.293333 23.920000 ... 6.52 5.34 1.46  
##   
## [3 rows x 33 columns]

**View the dataset size:**

print("\n Model dataframe shape\n", pydf.shape, "\n\nPredict dataframe shape\n", pyde.shape)

##   
## Model dataframe shape  
## (2571, 33)   
##   
## Predict dataframe shape  
## (267, 33)

#### 4.2 Formatting Data / Filtering Data / Cleaning Data

*Missing Values - Categorical Variable*

#replace NaN values with string in specific column  
pydf[['BrandCode']] = pydf[["BrandCode"]].fillna("Missing")   
pyde[['BrandCode']] = pyde[["BrandCode"]].fillna("Missing")

*Grouped by Brand Code*

#view dataframe first 6 columns based on 'BrandCode` summary grouping  
g2 = pyde.groupby('BrandCode').sum()  
g2.iloc[:, :6].round(3)

## CarbVolume FillOunces PCVolume CarbPressure CarbTemp PSC  
## BrandCode   
## A 189.827 814.407 9.831 2407.6 4930.0 3.020  
## B 684.473 2973.067 34.719 8667.0 18077.2 11.122  
## C 158.717 743.407 9.596 2066.8 4378.6 2.776  
## D 352.850 1533.627 16.933 4563.8 9067.4 5.016  
## Missing 42.227 191.727 1.953 518.8 1112.8 0.454

**Removing NaN values in the Numerical Variables**

from sklearn.impute import SimpleImputer

The data set is relatively small and the missing values will remain. The NaN’s will be replaced with a constant: zero.

# 'np.nan' signifies that we are targeting missing values  
# and the strategy we are choosing is replacing it with a 'constant'  
imputer = SimpleImputer(missing\_values=np.nan, strategy='constant', fill\_value = 0)  
  
imputer.fit(pydf.iloc[:, 1:])

## SimpleImputer(fill\_value=0, strategy='constant')

pydf.iloc[:, 1:] = imputer.transform(pydf.iloc[:, 1:])   
  
imputer.fit(pyde.iloc[:, 1:])

## SimpleImputer(fill\_value=0, strategy='constant')

pyde.iloc[:, 1:] = imputer.transform(pyde.iloc[:, 1:])  
# print the dataset  
#pydf

**Creating Dummies**

#created new data frame with `BrandCode` dummy variables  
modeldf2 = pd.get\_dummies(pydf, columns=['BrandCode'])  
predictdf2 = pd.get\_dummies(pyde, columns=['BrandCode'])

**Creating Train / Test data**

#train set  
x\_model = modeldf2["PH"] #target variable / dependent variable  
modeldf2.drop("PH", inplace = True, axis = 1) #independent variables  
modeldf2.shape

## (2571, 36)

x\_model.shape

## (2571,)

Removing PH column from Predict data set because the predicted values are not present.

#test set  
y\_pred = predictdf2["PH"]  
predictdf2.drop("PH", inplace = True, axis = 1)  
predictdf2.shape

## (267, 36)

y\_pred.shape

## (267,)

X\_train = modeldf2  
y\_train = x\_model  
X\_test = predictdf2  
y\_test = y\_pred

### PREDICTIONS

**Decision Tree Model**

#Building the Decision Tree Model on our dataset  
from sklearn.tree import DecisionTreeRegressor  
DT\_model = DecisionTreeRegressor(max\_depth=5).fit(X\_train, y\_train)  
DT\_predict = DT\_model.predict(X\_test) #Predictions on Testing data  
print(DT\_predict)

## [8.67422028 8.52699187 8.52699187 8.52699187 8.52699187 8.52699187  
## 8.52699187 8.52699187 8.52699187 8.52699187 8.52699187 8.52699187  
## 8.52699187 8.67422028 8.40734694 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.40734694 8.40734694 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.40734694 8.40734694 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.40734694 8.40734694 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.40734694 8.40734694  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.57409091 8.57409091 8.57409091  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.58 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.57409091 8.57409091  
## 8.57409091 8.40734694 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028 8.67422028  
## 8.67422028 8.67422028 8.67422028 8.50310127 8.50310127 8.50310127  
## 8.50310127 8.50310127 8.48866242 8.48866242 8.50310127 8.50310127  
## 8.48866242 8.50310127 8.50310127 8.50310127 8.48866242 8.61098901  
## 8.48866242 8.48866242 8.50310127 8.50310127 8.50310127 8.50310127  
## 8.41818681 8.50310127 8.50310127 8.50310127 8.41818681 8.50310127  
## 8.50310127 8.50310127 8.50310127 8.50310127 8.50310127 8.50310127  
## 8.50310127 8.41818681 8.50310127 8.50310127 8.50310127 8.48866242  
## 8.50310127 8.50310127 8.50310127 8.50310127 8.50310127 8.61098901  
## 8.61098901 8.61098901 8.50310127 8.50310127 8.41818681 8.50310127  
## 8.50310127 8.50310127 8.50310127 8.50310127 8.41818681 8.41818681  
## 8.41818681 8.61098901 8.41818681 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.48866242 8.41818681 8.41818681 8.41818681 8.41818681  
## 8.48866242 8.48866242 8.48866242 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.41818681 8.48866242 8.48866242 8.41818681 8.41818681  
## 8.41818681 8.61098901 8.61098901 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.41818681 8.41818681 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.41818681 8.48866242 8.61098901 8.41818681 8.41818681  
## 8.41818681 8.41818681 8.61098901 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.41818681 8.41818681 8.41818681 8.41818681 8.61098901  
## 8.61098901 8.41818681 8.41818681 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.48866242 8.48866242 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.61098901 8.48866242 8.41818681 8.41818681 8.41818681  
## 8.41818681 8.61098901 8.61098901 0. 8.41818681 8.61098901  
## 8.41818681 8.41818681 8.61098901 8.61098901 8.41818681 8.61098901  
## 8.41818681 8.41818681 8.41818681]

#DTpredict = pd.DataFrame(DT\_predict)  
#DTpredict.to\_csv("pypredict.csv")

**KNN Algorithm**

#Building the KNN Model on our dataset  
from sklearn.neighbors import KNeighborsRegressor  
KNN\_model = KNeighborsRegressor(n\_neighbors=3).fit(X\_train,y\_train)  
KNN\_predict = KNN\_model.predict(X\_test) #Predictions on Testing data  
print(KNN\_predict)

## [8.63333333 8.56 8.36666667 8.71333333 8.65333333 8.53333333  
## 8.63333333 8.36666667 8.70666667 8.7 8.56 8.55333333  
## 8.71333333 8.78666667 8.58 8.75333333 8.66 8.43333333  
## 8.62 8.52666667 8.43333333 8.6 8.48666667 8.6  
## 8.74 8.38666667 8.64 8.74 8.61333333 8.74666667  
## 8.68 8.7 8.67333333 8.45333333 8.40666667 8.64  
## 8.4 8.64666667 8.62666667 8.70666667 8.76666667 8.77333333  
## 8.44666667 8.38 8.76666667 8.76 8.72 8.52  
## 8.63333333 8.47333333 8.64666667 8.67333333 8.58 8.39333333  
## 8.42666667 8.64 8.56 8.61333333 8.71333333 8.67333333  
## 8.64666667 8.78666667 8.62 8.74 8.81333333 8.49333333  
## 8.63333333 8.72 8.72666667 8.56666667 8.7 8.27333333  
## 8.64 8.51333333 8.78 8.64666667 8.68 8.68666667  
## 8.76 8.78 8.72666667 8.76666667 8.85333333 8.54666667  
## 8.5 8.60666667 8.69333333 8.67333333 8.6 8.67333333  
## 8.82 3.12 8.76 8.67333333 8.46666667 8.52  
## 8.64666667 8.70666667 8.70666667 8.58 8.52 8.66  
## 8.67333333 8.74666667 8.78666667 8.72666667 8.63333333 8.48666667  
## 8.52 8.63333333 8.74666667 8.7 8.62666667 8.74  
## 8.36666667 8.69333333 8.70666667 8.64 8.7 8.7  
## 8.64666667 8.72 8.73333333 8.56 8.68666667 8.6  
## 8.54 8.26666667 8.46 8.49333333 8.54 8.52666667  
## 8.59333333 8.46666667 8.46666667 8.50666667 8.5 8.34  
## 8.4 8.41333333 8.44666667 8.61333333 8.50666667 8.61333333  
## 8.39333333 8.65333333 8.46 8.40666667 8.37333333 8.58  
## 8.59333333 8.64 8.62 8.72 8.48 8.56  
## 8.38 8.55333333 8.54 8.60666667 8.52666667 8.70666667  
## 8.52666667 8.56 8.53333333 8.61333333 8.54 8.71333333  
## 8.64 8.77333333 8.34666667 8.59333333 8.4 8.39333333  
## 8.32 8.45333333 8.4 8.42666667 8.45333333 8.43333333  
## 8.48 8.52 8.48 8.49333333 8.42 8.42666667  
## 8.5 8.55333333 8.52 8.44666667 8.5 8.3  
## 8.41333333 8.32666667 8.35333333 8.42666667 8.34 8.24666667  
## 8.37333333 8.45333333 8.36666667 8.41333333 8.46666667 8.53333333  
## 8.40666667 8.49333333 8.49333333 8.34666667 8.5 8.22  
## 8.26666667 8.38666667 8.34 8.56666667 8.40666667 8.5  
## 8.33333333 8.33333333 8.49333333 8.46 8.4 8.48  
## 8.5 8.43333333 8.65333333 8.43333333 8.48666667 8.54  
## 8.49333333 8.51333333 8.53333333 8.51333333 8.42666667 8.46  
## 8.51333333 8.56666667 8.54666667 8.64666667 8.49333333 8.49333333  
## 8.49333333 8.58 8.52 8.43333333 8.40666667 8.55333333  
## 8.37333333 8.48 8.52 8.48 8.48 8.38666667  
## 8.48666667 8.6 8.58 8.52 8.47333333 8.62666667  
## 8.50666667 8.49333333 8.66666667 8.64 8.58 8.58  
## 8.28 8.47333333 8.21333333]

#KNNpredict = pd.DataFrame(KNN\_predict)  
#KNNpredict.to\_csv("KNNpypredict.csv")