MANREEN KAUR

The codes are provided below, and the links to the datasets. Just follow below to build a machine learning model.

Contents

[Install datasets from Kaggle 2](#_Toc195730963)

[Import libraries 2](#_Toc195730964)

[Classification 3](#_Toc195730965)

[Classification Glass Model 3](#_Toc195730966)

[Classification Black Model 4](#_Toc195730967)

[Regression 5](#_Toc195730968)

[Regression Glass Model 5](#_Toc195730969)

[Regression Black Model 7](#_Toc195730970)

[Counterfactual 9](#_Toc195730971)

[Lime 11](#_Toc195730972)

# Install datasets from Kaggle

<https://www.kaggle.com/datasets/nikhil7280/student-performance-multiple-linear-regression>

<https://www.kaggle.com/datasets/l3llff/banana>

# Import libraries

|  |
| --- |
| # Import necessary libraries  import os  import pandas as pd  import numpy as np  import tensorflow as tf  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error |

# Classification

## Classification Glass Model

|  |
| --- |
| class\_df = pd.read\_csv('/kaggle/input/banana/banana\_quality.csv')  print(class\_df.head()) |
| *# This dataset has no missing values*  *# class\_df.dropna(inplace=True)*  *# Separate Features and Target Variables*  class\_X = class\_df.drop(columns='Quality')  class\_y = class\_df['Quality']  class\_num\_cols = list(class\_X.select\_dtypes(include=[np.number]).columns.values)  class\_cat\_cols = list(class\_X.select\_dtypes(exclude=[np.number]).columns.values)  *# Create Train & Test Data*  class\_X\_train, class\_X\_test, class\_y\_train, class\_y\_test = train\_test\_split(class\_X, class\_y,test\_size=0.3) |
| from sklearn.tree import DecisionTreeClassifier, plot\_tree  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report  class\_dt = DecisionTreeClassifier(max\_depth = 3, min\_samples\_leaf = 2)  class\_dt.fit(class\_X\_train, class\_y\_train)  # Predict on the test data and evaluate the model  class\_y\_pred = class\_dt.predict(class\_X\_test)  # Classification Report  print(classification\_report(class\_y\_pred, class\_y\_test)) |
| fig = plt.figure(figsize=(25,20))  \_ = plot\_tree(class\_dt,  feature\_names = list(class\_X\_train.columns),  class\_names = ['Bad', 'Good'],  filled=True, proportion = True) |

## Classification Black Model

|  |
| --- |
| reg\_df = pd.read\_csv('/kaggle/input/student-performance-multiple-linear-regression/Student\_Performance.csv')  print(reg\_df.head()) |
| from sklearn.ensemble import RandomForestClassifier  # Build model  class\_rf = RandomForestClassifier(max\_features=2, n\_estimators =100 ,bootstrap = True)  class\_rf.fit(class\_X\_train, class\_y\_train)  # Predict on the test data and evaluate the model  class\_y\_pred = class\_rf.predict(class\_X\_test)  # Classification Report  print(classification\_report(class\_y\_pred, class\_y\_test)) |

# Regression

## Regression Glass Model

|  |
| --- |
| reg\_df = pd.read\_csv('/kaggle/input/student-performance-multiple-linear-regression/Student\_Performance.csv') print(reg\_df.head()) |
| *# This dataset has no missing values*  *# reg\_df.dropna(inplace=True)*  *# Split Features (X) and Target (y)*  *reg\_X = reg\_df.drop(columns=["Performance Index"]) # Replace 'Target' with actual target column name*  *reg\_y = reg\_df["Performance Index"]*  *reg\_num\_cols = list(reg\_X.select\_dtypes(include=[np.number]).columns.values)*  *reg\_cat\_cols = list(reg\_X.select\_dtypes(exclude=[np.number]).columns.values)* |
| from sklearn.preprocessing import OneHotEncoder  *# Convert Categorical Data into Numerical (One-Hot Encoding)*  encoder = OneHotEncoder(sparse\_output=False, drop="first")  encoded\_df = pd.DataFrame(  encoder.fit\_transform(reg\_X[reg\_cat\_cols]),  columns=encoder.get\_feature\_names\_out(reg\_cat\_cols),  )  reg\_X = reg\_X.drop(columns=reg\_cat\_cols).reset\_index(drop=True)  reg\_X = pd.concat([reg\_X, encoded\_df], axis=1)  *# Split Dataset into Train and Test Sets*  reg\_X\_train, reg\_X\_test, reg\_y\_train, reg\_y\_test = train\_test\_split(reg\_X, reg\_y, test\_size=0.2, random\_state=42) |

|  |
| --- |
| from sklearn.linear\_model import LinearRegression  *# Buid & train model*  reg\_linear = LinearRegression()  reg\_linear.fit(reg\_X\_train, reg\_y\_train)  *# Evaluate model*  r2\_score = reg\_linear.score(reg\_X\_test, reg\_y\_test)  print(f"Final model R² Score: **{**r2\_score**}**") |
| print("Model coefficients:\n")  for i in range(reg\_X.shape[1]):  print(reg\_X.columns[i], "=", reg\_linear.coef\_[i].round(5)) |
| from sklearn.linear\_model import LinearRegression  *# Buid & train model*  reg\_linear = LinearRegression()  reg\_linear.fit(reg\_X\_train, reg\_y\_train)  *# Evaluate model*  r2\_score = reg\_linear.score(reg\_X\_test, reg\_y\_test)  print(f"Final model R² Score: **{**r2\_score**}**") |

## Regression Black Model

|  |
| --- |
| from tensorflow import keras  from tensorflow.keras import layers  from tensorflow.keras import backend as K  from keras.models import Sequential  from keras.layers import Dense, Dropout  from tensorflow.keras import Input  def r2\_score(y\_true, y\_pred):  SS\_res = K.sum(K.square(y\_true - y\_pred)) # Residual sum of squares  SS\_tot = K.sum(K.square(y\_true - K.mean(y\_true))) # Total sum of squares  return 1 - (SS\_res / (SS\_tot + K.epsilon())) # Add epsilon to avoid division by zero  # Define model  reg\_neural = model = Sequential([  Input(shape=(reg\_X\_train.shape[1],)), # Define input shape explicitly  Dense(1000, activation='relu'),# Input layer  Dropout(0.5),  Dense(500, activation='relu'), # Hidden layer with 500 neurons and ReLU activation  Dropout(0.5),  Dense(250, activation='relu'),  Dropout(0.5),  Dense(1, activation='linear') # Output layer with a single neuron (for regression)  ])  reg\_neural.compile(optimizer="adam", loss="mse", metrics=[r2\_score]) |
| # Train model  reg\_neural.fit(  reg\_X\_train, reg\_y\_train,  validation\_data=(reg\_X\_test, reg\_y\_test), # Track performance on test set  epochs=50, # Adjust epochs as needed  batch\_size=32, # Common batch size  verbose=1 # Show training progress  ) |
| # Evaluate model  loss, r2 = reg\_neural.evaluate(reg\_X\_test, reg\_y\_test)  print(f"Test Loss (MSE): {loss:.4f}")  print(f"Final model R² Score: {r2}") |

# Counterfactual

|  |
| --- |
| !pip install dice-ml |
| import dice\_ml  from dice\_ml.utils import helpers # helper functions  class\_dice\_train = pd.concat([class\_X\_train, class\_y\_train], axis=1)  print(class\_dice\_train)  # Dataset for training an ML model  # Continuous features need to be specified as they are perturbed differently  class\_d = dice\_ml.Data(dataframe=class\_dice\_train,  continuous\_features=['Size', 'Weight', 'Sweetness', 'Softness', 'HarvestTime', 'Ripeness', 'Acidity'],  outcome\_name='Quality')  # Pre-trained ML model  class\_m = dice\_ml.Model(model=class\_rf, backend='sklearn', model\_type='classifier')  # DiCE explanation instance  class\_exp = dice\_ml.Dice(class\_d,class\_m, method='random') |
| # Generate counterfactual examples  query\_instance = class\_X\_test[0:1]  dice\_exp = class\_exp.generate\_counterfactuals(query\_instance, total\_CFs=3, desired\_class="opposite")  # Visualize counterfactual explanation  dice\_exp.visualize\_as\_dataframe(show\_only\_changes=True) |
| query\_instance = class\_X\_test[0:1]  dice\_exp = class\_exp.generate\_counterfactuals(  query\_instance,  total\_CFs=3,  desired\_class="opposite",  features\_to\_vary=['Sweetness', 'Size'],  permitted\_range = {'Size': [-5, 5]}  )  dice\_exp.visualize\_as\_dataframe(show\_only\_changes=True) |

# Lime

|  |
| --- |
|  |
|  |
| # Calling the explain\_instance method by passing in the:  # 1) with test vector (fourth in this case)  # 2) prediction function used by our prediction model('reg\_neural' in this case)  # 3) the top features which we want to see, denoted by k    reg\_exp\_lime = reg\_explainer\_lime.explain\_instance(  reg\_X\_test.values[3], reg\_neural.predict, num\_features=5)    # Visualize the explanations  reg\_exp\_lime.show\_in\_notebook() #can import and show the visual representations of explanation |
| from lime import lime\_tabular  # Fit the Explainer on the training data set using the LimeTabularExplainer  reg\_explainer\_lime = lime\_tabular.LimeTabularExplainer(reg\_X\_train.values,  feature\_names=list(reg\_X\_train.columns),  class\_names=['Performance Index'],  verbose=True,  mode='regression' # “classification” or “regression”  ) |
| class\_explainer\_lime = lime\_tabular.LimeTabularExplainer(class\_X\_train.values,  feature\_names = list(class\_X\_train.columns),  class\_names = ['Bad', 'Good'],  verbose=True,  mode='classification') |
| class\_exp\_lime = class\_explainer\_lime.explain\_instance(  class\_X\_test.iloc[0], class\_rf.predict\_proba,  num\_features=5  )    class\_exp\_lime.show\_in\_notebook() |