Predictive Risk assessment model for Household Income below $2

Using Random Forest Model

Step 1: Data Collection

I downloaded the dataset from the email sent to me from raising the village.

Step 2: Importing Necessary Dependencies

#Import Dependencies

import pandas as pd

import numpy as np #making numpy arrays

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.ensemble import RandomForestRegressor

from sklearn import metrics

Step 3. Loading the data set

#Loading downloaded dataset into the pandas data frame

interview\_data = pd.read\_csv(r'C:\Users\USER\Desktop\interview\_package\interview\_dataset.csv')

Step 4: Data Pre processing

1. Displaying 5 row in the dataset

#Display the first five rows of the Interview dataset

interview\_data.head()

1. Viewing last five rows in the dataset

#Print the last five rows of the interview dataset

interview\_data.tail()

1. Identifying number of rows and columns in dataset

#Print the number of rows and columns

interview\_data.shape

1. Knowing data types in the dataset

#Getting basic information about the data

interview\_data.info()

1. Selecting key variables to be used from dataset

# Selecting columns to be used in model training (Key variables)

selected\_columns = [

'HH Income + Production/Day (USD)',

'most\_recommend\_rtv\_program',

'least\_recommend\_rtv\_program',

'most\_recommend\_rtv\_program\_reason',

'least\_recommend\_rtv\_program\_reason'

]

selected\_data = interview\_data.loc[:, selected\_columns]

1. Discovering datatypes of selected data

#Identifying datatypes of selected data

selected\_data.info()

1. Discovering missing values in the dataset

#checking missing values

selected\_data.isnull().sum()

1. Statistically describing the selected data in the dataset

#Understanding the statistical distribution of selected data for the model

selected\_data.describe()

1. Handling Missing values

#Checking the distribution data in most\_recommend\_rtv\_program to identify if to impute using mean or median

# Plotting distribution of 'most\_recommend\_rtv\_program'

sns.histplot(selected\_data['most\_recommend\_rtv\_program'].dropna(), kde=True)

plt.title("Distribution of most\_recommend\_rtv\_program")

plt.show()

#Checking the distribution data in least\_recommend\_rtv\_program to identify if to impute using mean or median

# Plotting distribution of 'least\_recommend\_rtv\_program'

sns.histplot(selected\_data['least\_recommend\_rtv\_program'].dropna(), kde=True)

plt.title("Distribution of least\_recommend\_rtv\_program")

plt.show()

1. Filling in missing values

# Filling missing values in selected\_data

selected\_data['most\_recommend\_rtv\_program'].fillna(selected\_data['most\_recommend\_rtv\_program'].median(), inplace=True)

selected\_data['least\_recommend\_rtv\_program'].fillna(selected\_data['least\_recommend\_rtv\_program'].median(), inplace=True)

1. Encoding Categorical Data

#Encoding categorical data for most\_recommend\_rtv\_program\_reason

# Fill missing text values with a placeholder

selected\_data['most\_recommend\_rtv\_program\_reason']=selected\_data['most\_recommend\_rtv\_program\_reason'].fillna("No Reason Provided")

# Using TF-IDF tranform text reasons to numeric data computable for machine learning

tfidf = TfidfVectorizer(max\_features=100, stop\_words='english')

tfidf\_matrix = tfidf.fit\_transform(selected\_data['most\_recommend\_rtv\_program\_reason'])

tfidf\_df = pd.DataFrame(tfidf\_matrix.toarray(), columns=tfidf.get\_feature\_names\_out())

selected\_data = pd.concat([selected\_data, tfidf\_df], axis=1)

selected\_data.drop(columns=['most\_recommend\_rtv\_program\_reason'], inplace=True)

#Encoding categorical data for least\_recommend\_rtv\_program\_reason

# Fill missing text values with a placeholder

selected\_data['least\_recommend\_rtv\_program\_reason']=selected\_data['least\_recommend\_rtv\_program\_reason'].fillna("No Reason Provided")

# Using TF-IDF tranform text reasons to numeric data computable for machine learning

tfidf = TfidfVectorizer(max\_features=100, stop\_words='english')

tfidf\_matrix = tfidf.fit\_transform(selected\_data['least\_recommend\_rtv\_program\_reason'])

tfidf\_df = pd.DataFrame(tfidf\_matrix.toarray(), columns=tfidf.get\_feature\_names\_out())

selected\_data = pd.concat([selected\_data, tfidf\_df], axis=1)

selected\_data.drop(columns=['least\_recommend\_rtv\_program\_reason'], inplace=True)

Data Analysis

Train Test Split

1. Seperating target feature from other features

#Seperating the Target Feature (Column) from the Other Features Columns

X = gold\_data.drop(['Close/Last'], axis=1)

Y = gold\_data['Close/Last']

1. Splitting dataset for training

#Spliting our dataset into training and Test data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,Y, test\_size=0.2, random\_state=2)

Actual Model Training

1. Loading Model

#Model Training

#Random Forest Model Regressor

#Loading the Model

fst\_model = RandomForestRegressor(n\_estimators=100)

1. Training the model

#Training Our Model

fst\_model.fit(x\_train, y\_train)

1. Model predictions

#After model has been trained, we give it test data and se see its predictions

model\_price\_predictions = fst\_model.predict(x\_test)

Evaluation

1. Calculating R squared value

#Obtaining R Squarded value for the predictions  
 error\_score = metrics.r2\_score(y\_test, model\_price\_predictions)

1. Printing Results

print('R Squared Value: ', error\_score)

1. Plotting Visualizations

#Visualising the Actual values and Predicted Values in a plot.

plt.scatter(y\_test, model\_price\_predictions)

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual Prices vs Prediceted Prices')

plt.show()

plt.plot(y\_test, color='Red', label='Actual Prices')

plt.plot(model\_price\_predictions, color='Blue', label='Predicted Prices')

plt.title('Actual Prices vs Predicted Prices')

plt.xlabel('Number of Values')

plt.ylabel('Close/Last Prices')

plt.legend()