Predictive Risk assessment model for Household Income below $2

Using RandomForestRegressor Model

Step 1: Data Collection

I obtained this dataset through download from an email sent to me from the hiring team of Raising   
 the village. I maintained the name of this csv file as interview\_dataset.csv throughout this project.

Step 2: Importing Necessary Dependencies

#Import Dependencies

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.ensemble import RandomForestRegressor

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

Step 3. Loading the data set

#loading the downloaded dataset into the pandas dataframe

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].copy(deep=True)

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

#Fill missing values in the most\_recommend\_rtv\_program with median value

column\_median = selected\_data['most\_recommend\_rtv\_program'].median()

selected\_data['most\_recommend\_rtv\_program'] = selected\_data['most\_recommend\_rtv\_program'].fillna(column\_median)

selected\_data['most\_recommend\_rtv\_program'].isnull().sum()

#Fill missing values in the least\_recommend\_rtv\_program with median value

column\_median = selected\_data['least\_recommend\_rtv\_program'].median()

selected\_data['least\_recommend\_rtv\_program'] = selected\_data['least\_recommend\_rtv\_program'].fillna(column\_median)

selected\_data['least\_recommend\_rtv\_program'].isnull().sum()

# Fill missing values in least\_recommend\_rtv\_program\_reason with the mode

column\_mode = selected\_data['least\_recommend\_rtv\_program\_reason'].mode()[0]

selected\_data['least\_recommend\_rtv\_program\_reason'] = selected\_data['least\_recommend\_rtv\_program\_reason'].fillna(column\_mode)

# Fill missing values in most\_recommend\_rtv\_program\_reason with the mode

column\_mode = selected\_data['most\_recommend\_rtv\_program\_reason'].mode()[0]

selected\_data['most\_recommend\_rtv\_program\_reason'] = selected\_data['most\_recommend\_rtv\_program\_reason'].fillna(column\_mode)

1. Encoding Categorical Data

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

# 1. Calculate unique values BEFORE encoding

unique\_values = selected\_data['least\_recommend\_rtv\_program\_reason'].nunique()

# 2. Label Encoding

label\_encoder = LabelEncoder()

selected\_data['least\_recommend\_rtv\_program\_reason\_encoded'] = label\_encoder.fit\_transform(selected\_data['least\_recommend\_rtv\_program\_reason'])

# 3. One-Hot Encoding (Conditional)

if unique\_values <= 10:

one\_hot\_encoded = pd.get\_dummies(selected\_data['least\_recommend\_rtv\_program\_reason'], prefix='least\_recommend\_reason')

selected\_data = pd.concat([selected\_data, one\_hot\_encoded], axis=1)

# 4. Drop the original categorical column

selected\_data = selected\_data.drop('least\_recommend\_rtv\_program\_reason', axis=1)

# 5. Display encoded data

print(selected\_data['least\_recommend\_rtv\_program\_reason\_encoded'].head())

if unique\_values <= 10:

print(selected\_data.filter(like='least\_recommend\_reason').head())

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

# Fill missing text values with a placeholder

# 1. Calculate unique values BEFORE encoding (using most\_recommend\_rtv\_program\_reason)

unique\_values = selected\_data['most\_recommend\_rtv\_program\_reason'].nunique()

# 2. Label Encoding (on most\_recommend\_rtv\_program\_reason)

label\_encoder = LabelEncoder()

selected\_data['most\_recommend\_rtv\_program\_reason\_encoded'] = label\_encoder.fit\_transform(selected\_data['most\_recommend\_rtv\_program\_reason'])

# 3. One-Hot Encoding (Conditional, on most\_recommend\_rtv\_program\_reason)

if unique\_values <= 10:

one\_hot\_encoded = pd.get\_dummies(selected\_data['most\_recommend\_rtv\_program\_reason'], prefix='most\_recommend\_reason') #correct prefix

selected\_data = pd.concat([selected\_data, one\_hot\_encoded], axis=1)

# 4. Drop the original categorical column (most\_recommend\_rtv\_program\_reason)

selected\_data = selected\_data.drop('most\_recommend\_rtv\_program\_reason', axis=1)

# 5. Display encoded data

print(selected\_data['most\_recommend\_rtv\_program\_reason\_encoded'].head())

if unique\_values <= 10:

print(selected\_data.filter(like='most\_recommend\_reason').head())

1. Visualising the target feature

# Visualize the target variable distribution

sns.countplot(x=selected\_data['HH Income + Production/Day (USD)'])

plt.title('Distribution of HH Income + Production/Day (USD)')

plt.show()

Step 4: Data Analysis

Train Test Split

1. Seperating target feature from other features

#Seperating target feature from other features

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

X = selected\_data.drop(['HH Income + Production/Day (USD)'], axis=1)

Y = selected\_data['HH Income + Production/Day (USD)']

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=42)

print(x\_train.shape, x\_test.shape, y\_train.shape, y\_test.shape)

Actual Model Training

1. Loading Model

#Model Training

#RandomForestRegressor

#Loading the Model

model = RandomForestRegressor(random\_state=42)

1. Training the model

#Training Our Model

model.fit(x\_train, y\_train)

1. Model predictions

#Model Predictions both on train and test data

#Model predictions on training data

y\_train\_pred = model.predict(x\_train)

#Model predictions on test data

y\_test\_pred = model.predict(x\_test)

1. Model Evaluations

#Model Evaluations both on train data

mse\_train = mean\_squared\_error(y\_train, y\_train\_pred)

mae\_train = mean\_absolute\_error(y\_train, y\_train\_pred)

r2\_train = r2\_score(y\_train, y\_train\_pred)

print(f"Mean Squared Error (MSE): {mse\_train}")

print(f"Mean Absolute Error (MAE): {mae\_train}")

print(f"R-squared (R2): {r2\_train}")

#Model Evaluations both on test data

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

r2\_test = r2\_score(y\_test, y\_test\_pred)

print(f"Mean Squared Error (MSE): {mse\_test}")

print(f"Mean Absolute Error (MAE): {mae\_test}")

print(f"R-squared (R2): {r2\_test}")

Step 5: Model Performance Report

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Training Data** | **Test Data** | **Interpretation** |
| Mean Squared Error(MSE) | 0.536 | 1.476 | Higher MSE on test data indicates overfitting (model fits training data too well) |
| Mean Absolute Error(MAE) | 0.531 | 0.915 | Higher MAE on test data confirms poor generalization to unseen data. |
| R Squared(R2) | 0.780 | 0.183 | Low R2 on test data (18.3%) means the model explains little variance in test data. |

**Key observations on the current model**

1. **Overfitting:**

The model performs well on the training data (low MSE, high R2) but poorly on the test data (high MSE, low R2).

This indicates that the model is too complex and is capturing noise in the training data instead of generalizing to new data.

1. **Poor Generalization:**

The large gap between training and test performance (e.g., R2: 0.780 vs. 0.183) suggests the model is not generalizing well.

Step 6: Making necessary adjustments in the model

1. Adjusting Model Parameters

#Adjusting model parameters to better performnce

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [5, 10, 15],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 5],

'max\_features': ['sqrt', 'log2']

}

# Perform grid search with cross-validation

model = RandomForestRegressor(random\_state=42)

grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(x\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

1. Model Predictions after adjusting parameters

#Model Predictions both on train and test data

#Model predictions on training data

y\_train\_pred = best\_model.predict(x\_train)

#Model predictions on test data

y\_test\_pred = best\_model.predict(x\_test)

1. Model Evaluations after adjusting parameters

#Model Evaluations both on train data after parameter adjustments

mse\_train = mean\_squared\_error(y\_train, y\_train\_pred)

mae\_train = mean\_absolute\_error(y\_train, y\_train\_pred)

r2\_train = r2\_score(y\_train, y\_train\_pred)

print(f"Mean Squared Error (MSE): {mse\_train}")

print(f"Mean Absolute Error (MAE): {mae\_train}")

print(f"R-squared (R2): {r2\_train}")

#Model Evaluations both on test data

mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)

mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)

r2\_test = r2\_score(y\_test, y\_test\_pred)

print(f"Mean Squared Error (MSE): {mse\_test}")

print(f"Mean Absolute Error (MAE): {mae\_test}")

print(f"R-squared (R2): {r2\_test}")

**Performance Summary of Model with adjusted parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Training Data** | **Test Data** | **Interpretation** |
| Mean Squared Error(MSE) | 1.670 | 1.312 | Training: High MSE indicates poor fit to training data.  Test: Slightly lower MSE than training, but still high, indicating poor generalization. |
| Mean Absolute Error(MAE) | 0.946 | 0.880 | Training: High MAE indicates poor fit to training data.  Test: Slightly lower MAE than training, but still high, indicating poor generalization. |
| R Squared(R2) | 0.316 | 0.274 | Training: Low R2 indicates the model explains only 31.6% of variance.  Test: Slightly lower R2 than training, indicating poor generalization |

**Key Observations**

Underfitting:

The model is underfitting the training data, as evidenced by:

High MSE (1.670) and MAE (0.946) on the training data.

Low R-squared (0.316) on the training data, meaning the model explains only 31.6% of the variance.

This suggests the model is too simple to capture the underlying patterns in the data.

Poor Generalization:

The model's performance on the test data is only slightly better than on the training data:

MSE (1.312) and MAE (0.880) are lower than training metrics but still high.

R-squared (0.274) is slightly lower than training R-squared, meaning the model explains only 27.4% of the variance in the test data.

This indicates that the model is not generalizing well to unseen data.

**Comparison with Initial Model:**

Before Adjustments:

The model was overfitting the training data (low training MSE, high test MSE).

Training R-squared was 0.780, but test R-squared was 0.183.

**After Adjustments:**

The model is now underfitting the training data (high training MSE, low R-squared).

Test performance improved slightly but is still poor.

**Conclusion**

Before Adjustments:

The model was overfitting the training data, meaning it performed well on the training data but poorly on the test data.

**Strengths:**

Good fit to training data (low MSE, high R-squared).

**Weaknesses:**

Poor generalization to test data (high MSE, low R-squared).

**After Adjustments:**

The model is now underfitting the training data, meaning it is too simple to capture the underlying patterns.

**Strengths:**

Slight improvement in generalization to test data (lower MSE, higher R-squared than before).

**Weaknesses:**

Poor fit to training data (high MSE, low R-squared).

Overall performance is worse than before.

**Final Verdict:**

The model is worse after adjusting the parameters because it is now underfitting the data. While the generalization to the test data improved slightly, the overall performance (both training and test) is worse than before.

The adjustments made the model too simple, reducing its ability to learn from the data.

**Future Recommendations continue.**