Machine Learning Project Report

Shreyan Gupta IMT2021039
Vidhu Arora IMT2021082
Mayank Sharma IMT2021086

December 17, 2023

1 Introduction

The provided Python code conducts an extensive analysis of a dataset focused on traveler satisfaction. This includes thorough exploratory data analysis (EDA) and the implementation of various machine learning models, with a particular emphasis on neural networks.

2 Data Preprocessing

Data preprocessing is a crucial step in the machine learning pipeline that involves cleaning and transforming raw data into a format suitable for training machine learning models. In the provided code, the data preprocessing is performed on the training and testing datasets. Here's a breakdown of the key data preprocessing steps:

2.1 Handling Missing Values

```
def datapreprocessing(data):
    data['Arrival Delay in Minutes'] =
        data['Arrival Delay in Minutes'].fillna(
        data['Arrival Delay in Minutes'].mode().iloc[0])
```

The `datapreprocessing` function is defined to handle missing values in the 'Arrival Delay in Minutes' column. Missing values are replaced with the mode (the most frequently occurring value) of the same column. This ensures that the missing values are imputed with a representative value, maintaining the integrity of the dataset.

2.2 Loading Data

```
training_data = pd.read_csv('/kaggle/input/sdatasets/train.csv')
testing_data = pd.read_csv('/kaggle/input/sdatasets/test.csv')
```

The training and testing datasets are loaded from CSV files. This step is crucial for obtaining the raw data that will be used to train and evaluate machine learning models.

2.3 Further Preprocessing

```
# Data Preprocessing
datapreprocessing(training_data)
datapreprocessing(testing_data)
```

The `datapreprocessing` function is applied to both the training and testing datasets. This ensures consistency in handling missing values across both datasets.

2.4 Checking for Null Values

```
# Check for null values in training data
training_data.isnull().sum()
```

Checking for null values in the training data is an essential step to identify and address any missing values that might still exist after the initial preprocessing.

2.5 Extracting Features and Target Variable

```
# Extract features and target variable
X = training_data.drop('contentment', axis=1)
X = pd.get_dummies(X)
Y = pd.get_dummies(training_data['contentment'], drop_first=True)
```

The features (`X`) and target variable (`Y`) are extracted from the training data. Categorical variables are one-hot encoded using `pd.get_dummies` to convert them into a format suitable for machine learning models.

2.6 Standardizing the Data

```
# Further preprocessing
X = X.iloc[:, 2:]
X_test = pd.get_dummies(testing_data).iloc[:, 2:]

# Standardize the data
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)
```

The data is further preprocessed by selecting relevant columns and standardizing the numerical features. Standardization ensures that numerical features have zero mean and unit variance, which can be important for the performance of certain machine learning algorithms.

These data preprocessing steps collectively ensure that the dataset is cleaned, missing values are handled appropriately, and features are prepared for training machine learning models. The standardized features are then ready to be used as input for the subsequent machine learning model training and evaluation.

3 Exploratory Data Analysis (EDA)

3.1 Target Variable Distribution

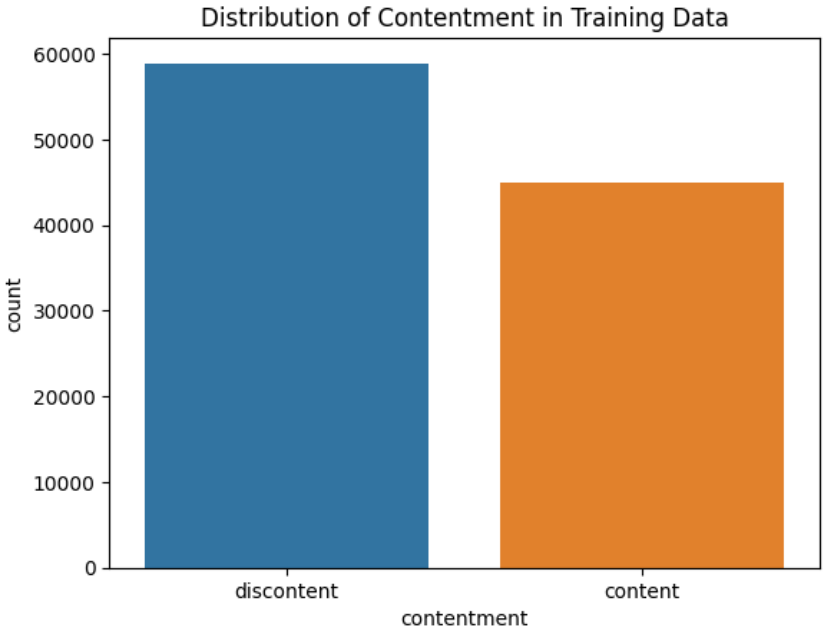
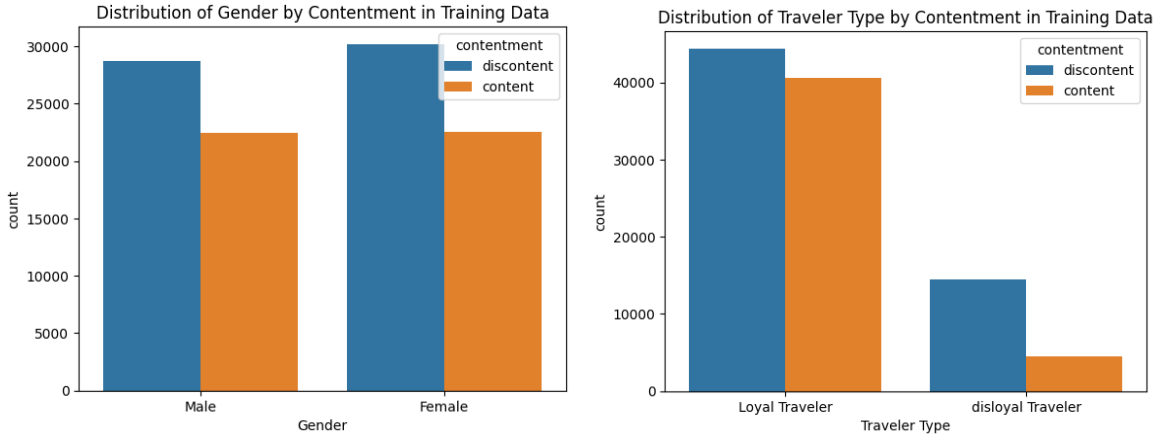


Figure 1: Distribution of Contentment in Training Data

The bar plot illustrating the distribution of the target variable, 'contentment,' provides a foundational understanding of the dataset. A balanced distribution indicates that both classes (content and discontent) are represented, which is crucial for training a machine learning model without bias.

3.2 Categorical Feature Distributions



(a) Distribution of Gender by Contentment

(b) Distribution of Traveler Type by Contentment

Figure 2: Distribution of Categorical Features by Contentment in Training Data

These count plots visualize how contentment varies across different categories such as gender and traveler type. They offer insights into potential patterns, showing if certain groups are more likely to be content or discontent. This information is valuable for feature importance and model interpretation.

3.3 Type of Travel Distribution

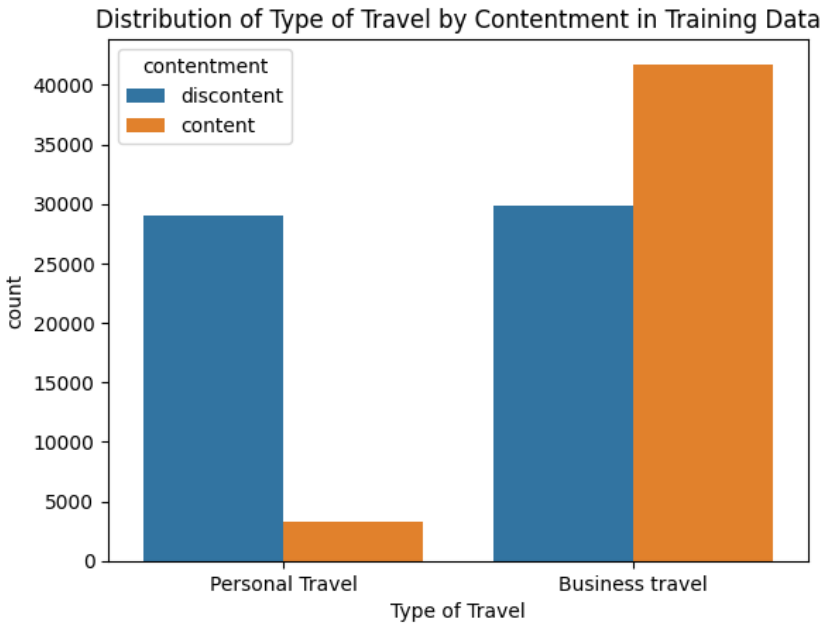
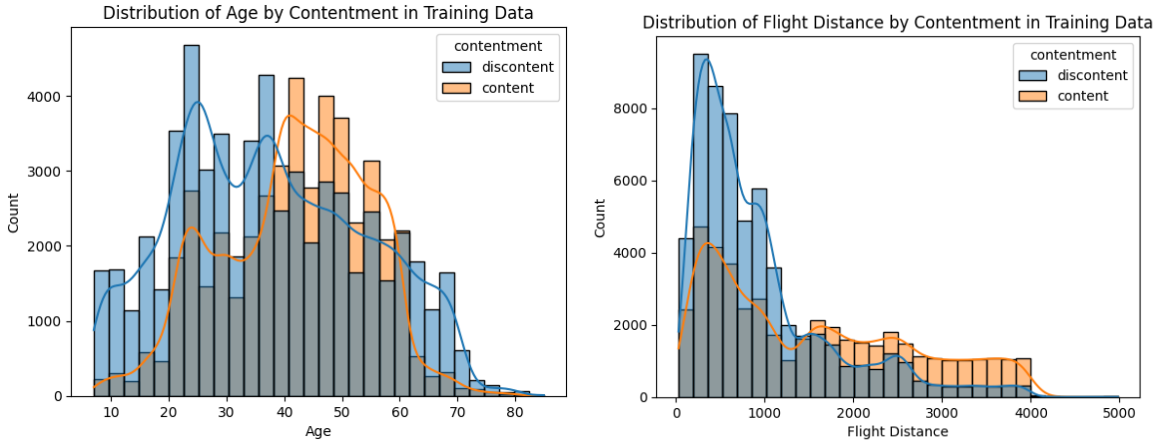


Figure 3: Distribution of Type of Travel by Contentment

This plot reveals the distribution of the type of travel for different contentment levels. It helps identify whether certain types of travel are associated with higher or lower satisfaction among travelers.

3.4 Numerical Feature Distributions



(a) Distribution of Age by Contentment

(b) Distribution of Flight Distance by Contentment

Figure 4: Distribution of Numerical Features by Contentment in Training Data

Histograms with kernel density estimates illustrate how numerical features (e.g., age, flight distance) are distributed for different contentment levels. Understanding these distributions helps identify potential age or distance ranges associated with higher contentment.

3.5 Departure Delay Distribution

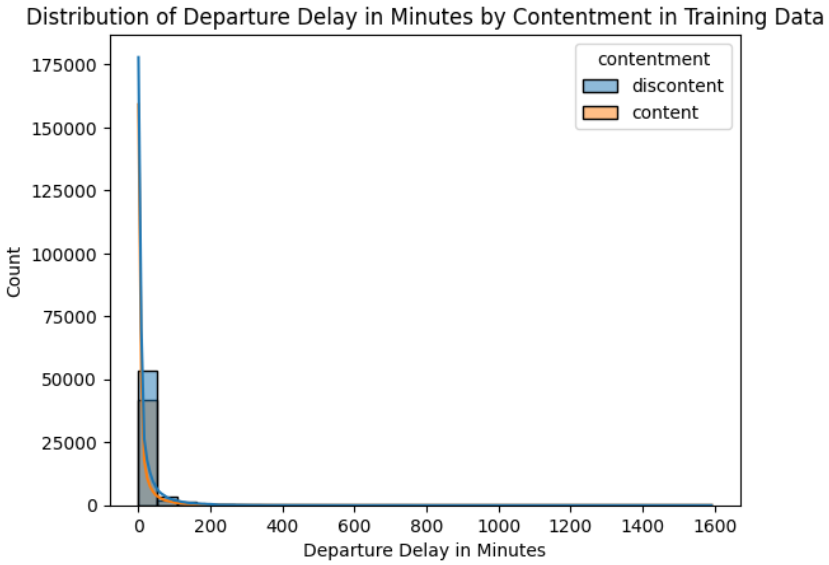


Figure 5: Distribution of Departure Delay by Contentment

This plot shows the distribution of departure delay times for different levels of contentment. It provides insights into whether delayed departures correlate with lower satisfaction levels among travelers.

3.6 Arrival Delay Distribution

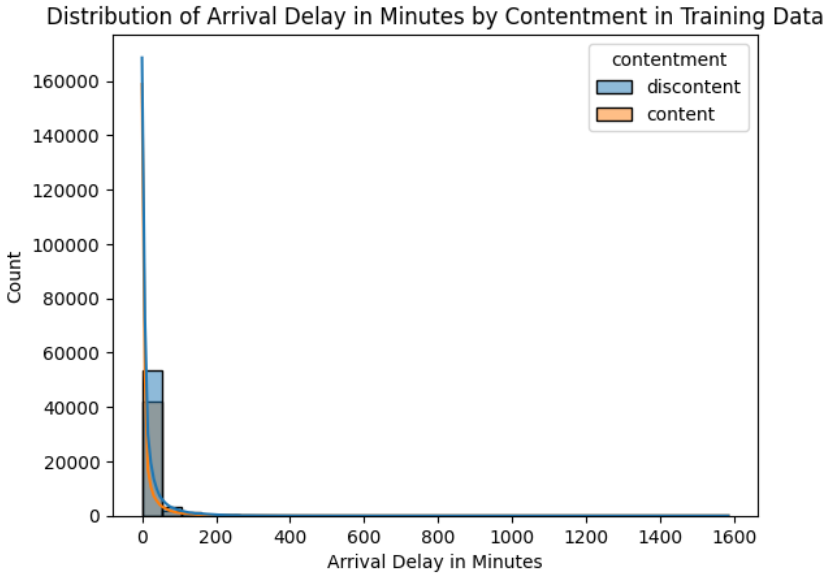


Figure 6: Distribution of Arrival Delay by Contentment

Similar to departure delay, this plot visualizes the distribution of arrival delay times for different contentment levels. Understanding how delays impact satisfaction is crucial for addressing potential pain points in the travel experience.

3.7 Class Distribution

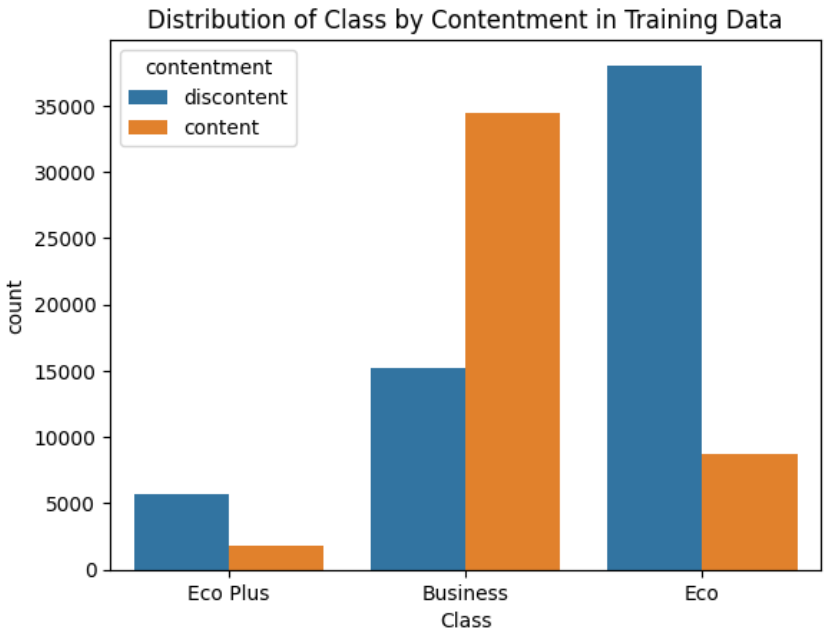


Figure 7: Distribution of Class by Contentment

This plot illustrates the distribution of travel class for different contentment levels. It helps identify whether certain classes are associated with higher or lower satisfaction among travelers.

3.8 Correlation Matrix

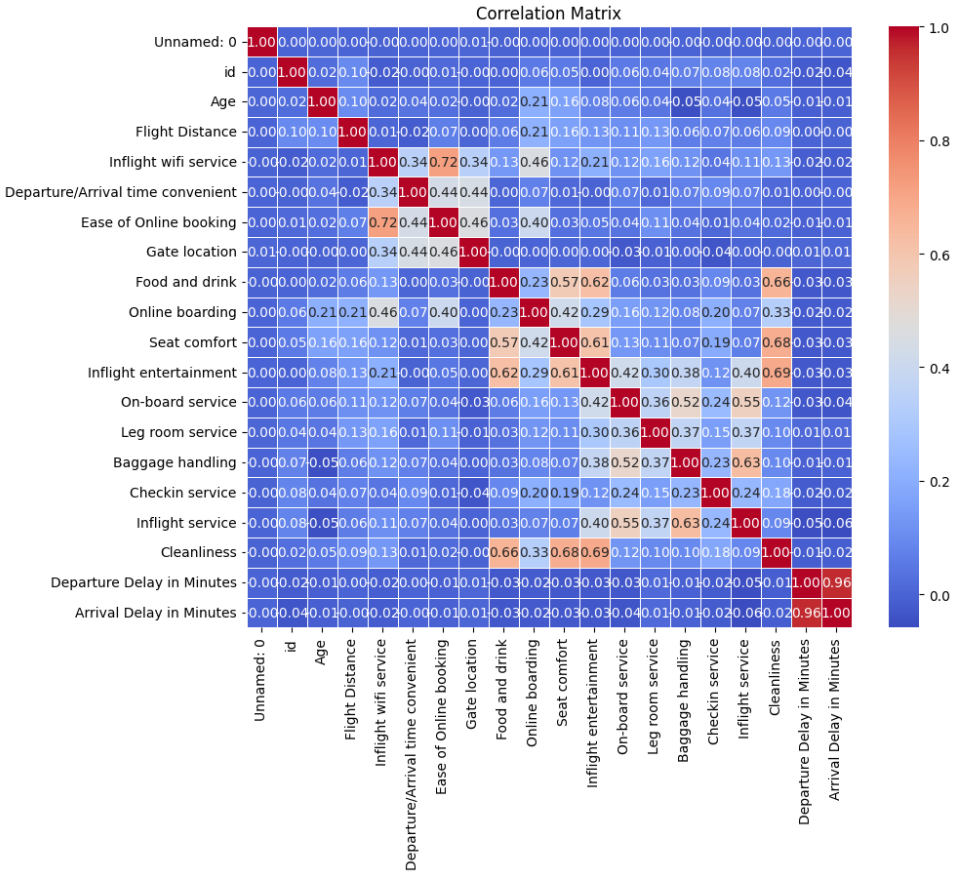


Figure 8: Correlation Matrix of Numerical Features

The correlation matrix heatmap provides insights into the relationships between numerical features. High correlations may indicate multicollinearity, which can impact model performance. Understanding these relationships guides feature selection and model-building decisions.

4 Model Training and Evaluation

4.1 Neural Network Architectures and Differences

4.1.1 Model 1

This neural network architecture represents a relatively simple configuration. It begins with a flattening layer, converting the input data into a one-dimensional array. It is followed by a hidden layer with 64 neurons, another hidden layer with 32 neurons, and a final output layer with a sigmoid activation function. The sigmoid activation in the output layer makes it suitable for binary classification, with the objective of predicting traveler contentment.

4.1.2 Model 2

Model 2 introduces a more complex architecture. It includes a flattening layer followed by two hidden layers with 128 and 64 neurons, respectively. The final output layer retains the sigmoid activation function. The increased number of neurons and additional hidden layers allows the model to capture more intricate patterns and relationships within the data.

4.1.3 Model 3

This neural network configuration differs from the previous ones. It includes a flattening layer, followed by three hidden layers with 32, 16, and 8 neurons, respectively. The final output layer with a sigmoid activation function remains consistent with binary classification requirements. The structure of Model 3 introduces a different level of complexity, potentially capturing features in a unique way.

4.1.4 Model 4

Model 4 is an extended neural network with four hidden layers having 64, 48, 32, and 16 neurons, respectively. The final output layer continues to use the sigmoid activation function. This architecture adds further complexity to the model, allowing it to potentially learn intricate patterns in the dataset.

4.2 Training Process

The training process is a critical stage in machine learning where the model learns to adjust its parameters (weights and biases) based on the training data. The process involves the following steps:

4.2.1 Compilation

Before training, the model needs to be compiled. In this code, the Nadam optimizer is used, which is an extension of the Adam optimizer. The loss function chosen is binary cross-entropy, suitable for binary classification problems. Additionally, the accuracy metric is monitored during training.

```
model.compile(optimizer=Nadam(learning_rate=1e-3),
              loss='binary_crossentropy', metrics=['accuracy'])
```

4.2.2 Training

The model is then trained on the preprocessed data. The 'fit' method is employed, specifying the input features ('X') and target variable ('Y'). The '$validation_split$' parameter allocates a portion of the training data for validation, allowing the model's performance to be monitored on a separate dataset during training. The 'epochs' parameter determines the number of times the model will iterate over the entire training dataset.

```
model.fit(X, Y, validation_split=0.1, epochs=20)
```

4.3 Model Evaluation

The evaluation phase assesses the model's performance on a separate dataset, often referred to as the testing dataset. In this code, each trained model is evaluated on the testing data, and the accuracy is reported. Additionally, predictions are made on the testing set:

```
y_pred = model.predict(X_test)
```

The model's predictions are then converted to binary form based on a threshold of 0.5:

```
target = (y_pred > 0.5).astype(int)
```

These binary predictions are used to create a DataFrame for submission. The results, along with the 'id' column from the testing data, are

5 Model Training and Evaluation

5.1 Neural Network Architectures and Differences

5.1.1 Model 1

This neural network architecture represents a relatively simple configuration. It begins with a flattening layer, converting the input data into a one-dimensional array. It is followed by a hidden layer with 64 neurons, another hidden layer with 32 neurons, and a final output layer with a sigmoid activation function. The sigmoid activation in the output layer makes it suitable for binary classification, with the objective of predicting traveler contentment.

5.1.2 Model 2

Model 2 introduces a more complex architecture. It includes a flattening layer followed by two hidden layers with 128 and 64 neurons, respectively. The final output layer retains the sigmoid activation function. The increased number of neurons and additional hidden layers allows the model to capture more intricate patterns and relationships within the data.

5.1.3 Model 3

This neural network configuration differs from the previous ones. It includes a flattening layer, followed by three hidden layers with 32, 16, and 8 neurons, respectively. The final output layer with a sigmoid activation function remains consistent with binary classification requirements. The structure of Model 3 introduces a different level of complexity, potentially capturing features in a unique way.

5.1.4 Model 4

Model 4 is an extended neural network with four hidden layers having 64, 48, 32, and 16 neurons, respectively. The final output layer continues to use the sigmoid activation function. This architecture adds further complexity to the model, allowing it to potentially learn intricate patterns in the dataset.

5.2 Training Process

The training process is a critical stage in machine learning where the model learns to adjust its parameters (weights and biases) based on the training data. The process involves the following steps:

5.2.1 Compilation

Before training, the model needs to be compiled. In this code, the Nadam optimizer is used, which is an extension of the Adam optimizer. The loss function chosen is binary cross-entropy, suitable for binary classification problems. Additionally, the accuracy metric is monitored during training.

```
model.compile(optimizer=Nadam(learning_rate=1e-3),
              loss='binary_crossentropy', metrics=['accuracy'])
```

5.2.2 Training

The model is then trained on the preprocessed data. The 'fit' method is employed, specifying the input features ('X') and target variable ('Y'). The '$validation_split$' parameter allocates a portion of the training data for validation, allowing the model's performance to be monitored on a separate dataset during training. The 'epochs' parameter determines the number of times the model will iterate over the entire training dataset.

```
model.fit(X, Y, validation_split=0.1, epochs=20)
```

5.3 Model Evaluation

The evaluation phase assesses the model's performance on a separate dataset, often referred to as the testing dataset. In this code, each trained model is evaluated on the testing data, and the accuracy is reported. Additionally, predictions are made on the testing set:

```
y_pred = model.predict(X_test)
```

The model's predictions are then converted to binary form based on a threshold of 0.5:

```
target = (y_pred > 0.5).astype(int)
```

These binary predictions are used to create a DataFrame for submission. The results, along with the 'id' column from the testing data, are saved to CSV files:

```
result = pd.DataFrame({'id': testing_data['id'], 'Target': target.flatten()})
result['contentment'] = result['Target'].map({1: 'neutral or discontent', 0: 'content'})
result = result.drop(['Target'], axis=1)
result.to_csv(f'sample_submission_model_{i}.csv', index=False)
```

This process allows for a comprehensive assessment of each model's ability to generalize to unseen data. The choice of evaluation metric, accuracy in this case, provides a measure of the model's overall correctness in predicting traveler contentment. The saved CSV files can be further analyzed or submitted for evaluation in a relevant competition or task.

6 Conclusion

In conclusion, the provided code conducts a comprehensive analysis of a traveler satisfaction dataset. Key aspects of the analysis include thorough exploratory data analysis (EDA) and the implementation of various machine learning models, with a focus on neural networks. The following key points summarize the findings and outcomes:

- **Exploratory Data Analysis (EDA):** The EDA phase revealed insights into the distribution of the target variable ('contentment') as well as the relationships between contentment and various categorical and numerical features. Visualization techniques such as count plots and histograms were employed to provide a clear understanding of the dataset.
- **Data Preprocessing:** The dataset underwent preprocessing steps to handle missing values and ensure its readiness for model training. Categorical variables were one-hot encoded, and numerical features were standardized to facilitate the training of machine learning models.
- **Neural Network Models:** Four neural network models (Model 1, Model 2, Model 3, and Model 4) with different architectures were implemented. These models varied in complexity, featuring different numbers of hidden layers and neurons. The models were trained and evaluated on the provided dataset, with accuracy metrics reported for each.
- **Model Training and Evaluation:** The training process involved compiling the models with appropriate optimizers and loss functions, followed by training on the preprocessed data. The models were evaluated on a separate testing dataset, and predictions were made using a threshold of 0.5 for binary classification. The results were saved to CSV files for further analysis or submission.

In summary, the code combines effective data exploration, preprocessing, and the implementation of neural network models to address the task of predicting traveler contentment. The insights gained from this analysis and the trained models can be valuable in understanding factors influencing satisfaction and making predictions on new data.