**Steps to Process the Cuisine Column**

**1. Split the Values into Multiple Categories**

The first step is to split the cuisines into individual components and then handle them as separate features.

python

CopyEdit

# Split the 'Cuisine' column into individual cuisines

cuisine\_split = df\_train['Cuisine'].str.split(',', expand=True)

# Rename columns for clarity (if cuisine\_split results in multiple columns)

cuisine\_split.columns = ['Cuisine\_1', 'Cuisine\_2']

# Add the split columns back to the original dataframe

df\_train = pd.concat([df\_train, cuisine\_split], axis=1)

Now, the dataset will have Cuisine\_1 and Cuisine\_2 as separate columns.

**2. Create Binary Features for Each Cuisine**

One-hot encode the unique cuisines across both Cuisine\_1 and Cuisine\_2 to create binary indicators for each cuisine.

python

CopyEdit

# Combine Cuisine\_1 and Cuisine\_2 into a single column to extract unique cuisines

all\_cuisines = pd.concat([df\_train['Cuisine\_1'], df\_train['Cuisine\_2']])

# Get unique cuisines

unique\_cuisines = all\_cuisines.unique()

# Create binary columns for each unique cuisine

for cuisine in unique\_cuisines:

df\_train[cuisine] = df\_train[['Cuisine\_1', 'Cuisine\_2']].apply(lambda x: cuisine in x.values, axis=1).astype(int)

This will create new columns, each representing a unique cuisine (e.g., indian, irish, tibetan, etc.), with binary values indicating the presence (1) or absence (0) of that cuisine.

**3. Drop the Original Cuisine Column**

After encoding, drop the original Cuisine, Cuisine\_1, and Cuisine\_2 columns since their information is now encoded into binary features.

python

CopyEdit

df\_train.drop(columns=['Cuisine', 'Cuisine\_1', 'Cuisine\_2'], inplace=True)

**Advantages of This Approach**

1. **Preserves All Information**:
   * Instead of treating combined cuisines as separate categories, the one-hot encoding approach allows the model to understand the presence of each individual cuisine.
2. **Avoids Cardinality Issues**:
   * Direct one-hot encoding of combined cuisines (e.g., indian,irish as a single category) would result in high cardinality, making the model less generalizable.
3. **Better for Tree-Based Models**:
   * Models like Random Forest or XGBoost can handle the binary indicators effectively.

**Alternative: Frequency Encoding**

If the number of unique cuisines is too large, you could use **frequency encoding** instead of one-hot encoding:

python

CopyEdit

# Combine and count frequency of each unique cuisine

cuisine\_frequency = all\_cuisines.value\_counts()

# Map frequencies back to the original dataframe

df\_train['Cuisine\_1\_Freq'] = df\_train['Cuisine\_1'].map(cuisine\_frequency)

df\_train['Cuisine\_2\_Freq'] = df\_train['Cuisine\_2'].map(cuisine\_frequency)

This approach assigns a frequency count to each cuisine, reducing dimensionality but potentially losing some nuance.

**Choosing the Right Approach**

* Use **one-hot encoding** if:
  + The number of unique cuisines is manageable, and you want to preserve all detail.
* Use **frequency encoding** if:
  + The dataset contains many unique cuisines, leading to high dimensionality with one-hot encoding.

Let me know if you’d like help implementing these steps or exploring alternative strategies!

**Alternate Aproach**

To ensure consistency between the training and testing datasets—where both datasets have the same set of binary columns for all cuisines—we need to derive the unique cuisines from both datasets combined. Here's how you can handle this situation step-by-step:

**Steps to Ensure Consistency**

1. **Combine Unique Cuisines from Both Datasets**:
   * Extract unique cuisines from both training and testing datasets.
   * Create a union of these unique cuisines.
2. **Add Missing Cuisine Columns**:
   * Ensure that both datasets have binary columns for every cuisine in the combined list.
   * For cuisines not present in a dataset, create a binary column filled with 0.
3. **Drop Temporary Columns**:
   * After encoding, drop the temporary columns like Cuisine\_1 and Cuisine\_2 from both datasets.

**Code Implementation**

python

CopyEdit

# Split 'Cuisine' into individual cuisines for both train and test datasets

cuisine\_split\_train = df\_train['Cuisine'].str.split(',', expand=True)

cuisine\_split\_test = df\_test['Cuisine'].str.split(',', expand=True)

# Rename columns for clarity

cuisine\_split\_train.columns = ['Cuisine\_1', 'Cuisine\_2']

cuisine\_split\_test.columns = ['Cuisine\_1', 'Cuisine\_2']

# Add the split columns back to the original dataframes

df\_train = pd.concat([df\_train.reset\_index(drop=True), cuisine\_split\_train], axis=1)

df\_test = pd.concat([df\_test.reset\_index(drop=True), cuisine\_split\_test], axis=1)

# Combine Cuisine\_1 and Cuisine\_2 across both datasets to get all unique cuisines

all\_cuisines\_train = pd.concat([df\_train['Cuisine\_1'], df\_train['Cuisine\_2']], ignore\_index=True)

all\_cuisines\_test = pd.concat([df\_test['Cuisine\_1'], df\_test['Cuisine\_2']], ignore\_index=True)

all\_cuisines = pd.concat([all\_cuisines\_train, all\_cuisines\_test], ignore\_index=True).dropna().unique()

# Create binary columns for each unique cuisine in both datasets

for cuisine in all\_cuisines:

# For train dataset

df\_train[cuisine] = df\_train[['Cuisine\_1', 'Cuisine\_2']].apply(lambda x: cuisine in x.values, axis=1).astype(int)

# For test dataset

df\_test[cuisine] = df\_test[['Cuisine\_1', 'Cuisine\_2']].apply(lambda x: cuisine in x.values, axis=1).astype(int)

# Drop temporary columns from both datasets

df\_train.drop(columns=['Cuisine', 'Cuisine\_1', 'Cuisine\_2'], inplace=True)

df\_test.drop(columns=['Cuisine', 'Cuisine\_1', 'Cuisine\_2'], inplace=True)

**Explanation of the Code**

1. **Splitting the Cuisine Column**:
   * The str.split(',', expand=True) splits the Cuisine column into Cuisine\_1 and Cuisine\_2.
2. **Combine Unique Cuisines**:
   * Use pd.concat() across both datasets (train and test) to gather all possible cuisines.
   * Use .unique() to extract distinct cuisines.
3. **Binary Encoding**:
   * For each unique cuisine, create a binary column in both datasets.
   * Use a lambda function to check if the cuisine exists in Cuisine\_1 or Cuisine\_2 for each row.
4. **Drop Temporary Columns**:
   * Remove Cuisine, Cuisine\_1, and Cuisine\_2 after the encoding is complete.

**Ensuring Consistency**

* Both df\_train and df\_test will now have the exact same set of binary columns for all cuisines.
* If a cuisine exists in the training set but not in the testing set (or vice versa), its binary column in the other dataset will be filled with 0s.

**Verification**

After running the above code, you can verify consistency:

python

CopyEdit

# Check if train and test have the same columns

assert set(df\_train.columns) == set(df\_test.columns), "Columns mismatch between train and test datasets"

# Display first few rows of both datasets

print(df\_train.head())

print(df\_test.head())

**Steps to Fix the Column Mismatch**

**1. Ensure All Cuisines Are Identified from Both Datasets**

Make sure the list of unique cuisines (all\_cuisines) is derived from both training and testing datasets combined.

python

CopyEdit

# Combine all unique cuisines from both train and test datasets

all\_cuisines\_train = pd.concat([df\_train['Cuisine\_1'], df\_train['Cuisine\_2']], ignore\_index=True)

all\_cuisines\_test = pd.concat([df\_test['Cuisine\_1'], df\_test['Cuisine\_2']], ignore\_index=True)

all\_cuisines = pd.concat([all\_cuisines\_train, all\_cuisines\_test], ignore\_index=True).dropna().unique()

This ensures that all\_cuisines contains every unique cuisine present in **both** datasets.

**2. Add Missing Columns**

When creating binary columns, ensure both datasets have the same columns. If a column (cuisine) exists in one dataset but not the other, create it and fill with 0.

python

CopyEdit

# Ensure both train and test datasets have the same columns for cuisines

for cuisine in all\_cuisines:

if cuisine not in df\_train.columns:

df\_train[cuisine] = 0

if cuisine not in df\_test.columns:

df\_test[cuisine] = 0

**3. Reorder Columns**

Even if both datasets have the same columns, their order might differ. Reorder the columns in the test dataset to match the training dataset.

python

CopyEdit

# Reorder test columns to match train columns

df\_test = df\_test[df\_train.columns]

**4. Verify Column Consistency**

After applying the above fixes, verify again that both datasets have identical columns.

python

CopyEdit

assert set(df\_train.columns) == set(df\_test.columns), "Columns mismatch between train and test datasets"

**Full Corrected Code**

Here’s the full implementation:

python

CopyEdit

# Split 'Cuisine' into individual cuisines for both train and test datasets

cuisine\_split\_train = df\_train['Cuisine'].str.split(',', expand=True)

cuisine\_split\_test = df\_test['Cuisine'].str.split(',', expand=True)

# Rename columns for clarity

cuisine\_split\_train.columns = ['Cuisine\_1', 'Cuisine\_2']

cuisine\_split\_test.columns = ['Cuisine\_1', 'Cuisine\_2']

# Add the split columns back to the original dataframes

df\_train = pd.concat([df\_train.reset\_index(drop=True), cuisine\_split\_train], axis=1)

df\_test = pd.concat([df\_test.reset\_index(drop=True), cuisine\_split\_test], axis=1)

# Combine all unique cuisines from both train and test datasets

all\_cuisines\_train = pd.concat([df\_train['Cuisine\_1'], df\_train['Cuisine\_2']], ignore\_index=True)

all\_cuisines\_test = pd.concat([df\_test['Cuisine\_1'], df\_test['Cuisine\_2']], ignore\_index=True)

all\_cuisines = pd.concat([all\_cuisines\_train, all\_cuisines\_test], ignore\_index=True).dropna().unique()

# Create binary columns for each unique cuisine in both datasets

for cuisine in all\_cuisines:

df\_train[cuisine] = df\_train[['Cuisine\_1', 'Cuisine\_2']].apply(lambda x: cuisine in x.values, axis=1).astype(int)

df\_test[cuisine] = df\_test[['Cuisine\_1', 'Cuisine\_2']].apply(lambda x: cuisine in x.values, axis=1).astype(int)

# Ensure both train and test datasets have all cuisine columns

for cuisine in all\_cuisines:

if cuisine not in df\_train.columns:

df\_train[cuisine] = 0

if cuisine not in df\_test.columns:

df\_test[cuisine] = 0

# Reorder test columns to match train columns

df\_test = df\_test[df\_train.columns]

# Drop temporary columns from both datasets

df\_train.drop(columns=['Cuisine', 'Cuisine\_1', 'Cuisine\_2'], inplace=True)

df\_test.drop(columns=['Cuisine', 'Cuisine\_1', 'Cuisine\_2'], inplace=True)

# Verify column consistency

assert set(df\_train.columns) == set(df\_test.columns), "Columns mismatch between train and test datasets"

**Key Points**

1. **Handling Missing Columns**:
   * Missing columns are filled with 0 to ensure alignment between the datasets.
2. **Reordering Columns**:
   * The test dataset is reordered to match the column order of the train dataset.
3. **Final Verification**:
   * The assert statement ensures column consistency after processing.