Cheat Sheet for Data Analysis #4Digitizing Categorical Features

Basic Setup and Data Loading

Import essential libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.feature_extraction.text import CountVectorizer
```

Meaning: Imports necessary libraries for data analysis and categorical feature encoding

Load a CSV file into a DataFrame

```
df = pd.read_csv("Diamond.csv")
```

Meaning: Reads data from a CSV file and stores it in a variable called `df`

Identify Categorical Variables

Display data types of each column

```
print(df.dtypes)
```

Meaning: `.dtypes` shows the data type of each column; object types are typically categorical variables

Get concise summary of the DataFrame

```
print(df.info())
```

Meaning: `.info()` helps identify categorical columns by showing data types and non-null counts

Check unique values in categorical columns

```
for col in ['colour', 'clarity', 'certification']:
    print(f"Unique values in {col}: {df[col].unique()}")
    print(f"Value counts:\n{df[col].value_counts()}\n")
```

Meaning: Reveals all distinct categories and their frequencies; essential for understanding the structure before encoding

---

Label Encoding (Ordinal/Integer Encoding)

Manual label mapping

```python
# Define mapping for clarity (ordered from best to worst)
clarity_mapping = {'IF': 6, 'VVS1': 5, 'VVS2': 4, 'VS1': 3, 'VS2': 2}
df['clarity_encoded'] = df['clarity'].map(clarity_mapping)
```

Meaning: Assigns numerical values based on logical order; appropriate when categories have natural ranking (e.g., clarity grades)

Using scikit-learn's LabelEncoder

```python
le = LabelEncoder()
df['colour_encoded'] = le.fit_transform(df['colour'])
# To reverse the encoding later:
# original_colours = le.inverse_transform(df['colour_encoded'])
```

Meaning: Automatically assigns integers to categories alphabetically; useful for tree-based models but creates artificial ordering

When to use: For ordinal data or when memory efficiency is critical; avoid for nominal data with no natural order

---

One-Hot Encoding (Dummy Variables)

Using pandas get_dummies()

```python
# Create dummy variables for certification
dummies_cert = pd.get_dummies(df['certification'], prefix='cert')
df_encoded = pd.concat([df, dummies_cert], axis=1)
print(dummies_cert.head())
```

Meaning: Creates binary columns for each category; avoids implying ordinal relationships between nominal categories

Drop first category to avoid multicollinearity

```python
dummies_drop_first = pd.get_dummies(df['certification'],
                                    prefix='cert',
                                    drop_first=True)
```

Meaning: Prevents perfect multicollinearity in regression models by dropping one reference category

Apply to multiple categorical columns

```python
# One-hot encode all categorical variables
categorical_cols = ['colour', 'clarity', 'certification']
```

```
df_with_dummies = pd.get_dummies(df,
                                 columns=categorical_cols,
                                 prefix=categorical_cols,
                                 drop_first=True)
```

Meaning: Efficiently encodes multiple categorical variables simultaneously; essential preprocessing step for many ML algorithms

---

Advanced Text Feature Extraction

Binary presence/absence features

```
# Create binary features based on text patterns
df['is_high_colour'] = df['colour'].isin(['D', 'E', 'F']).astype(int)
df['is_premium_clarity'] = df['clarity'].isin(['IF', 'VVS1', 'VVS2']).astype(int)
```

Meaning: Converts complex categorical logic into simple binary indicators; captures domain-specific knowledge

Frequency-based encoding

```
# Replace categories with their frequency in dataset
freq_encoding = df['certification'].value_counts()
df['certification_freq'] = df['certification'].map(freq_encoding)
```

Meaning: Encodes categories by their prevalence; can be more informative than arbitrary labels

---

Practical Tips for Categorical Feature Encoding

1. **Choose encoding method based on variable type:**
    - Use **one-hot encoding** for nominal variables (no natural order)
    - Use **label encoding** only for truly ordinal variables
    - Consider **target encoding** for high-cardinality categorical variables
2. **Handle high-cardinality variables carefully:**
    - Group rare categories into "Other" or "Rare" category
    - Use frequency encoding or target encoding instead of one-hot encoding
    - Consider embedding techniques for very high cardinality
3. **Avoid the Dummy Variable Trap:**
    - Always use `drop_first=True` in regression models
    - Keep all dummies for tree-based models if needed
4. **Preserve encoding mappings:**
    - Save LabelEncoder objects or mapping dictionaries
    - Ensure consistent encoding between training and test datasets

5. **Memory considerations:**
    - One-hot encoding increases dimensionality significantly
    - Use sparse matrices for datasets with many categorical variables
    - Consider feature hashing for extremely large categorical spaces
6. **Domain knowledge matters:**
    - Some categorizations may have meaningful order even if not obvious
    - Combine related categories based on business logic
    - Create hierarchical features when appropriate
7. **Always validate your encoding:**
    - Check that encoded features make sense
    - Verify no information loss during transformation
    - Test different encoding strategies through cross-validation

Remember: The goal of digitizing categorical features is to represent them in a way that machine learning algorithms can effectively use while preserving their semantic meaning as much as possible.