# Categorical pitfalls

WORKING WITH CATEGORICAL DATA IN PYTHON

**Kasey Jones**
Research Data Scientist

datacamp

# Used cars: the final dataset

```
import pandas as pd

used_cars = pd.read_csv("used_cars.csv")
used_cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38531 entries, 0 to 38530
Data columns (total 30 columns):
 #   Column             Non-Null Count  Dtype
 --  ------             --------------  -----
 0   manufacturer_name  38531 non-null  object
 1   model_name         38531 non-null  object
 2   transmission       38531 non-null  object
...
```

# Huge memory savings

```
used_cars['manufacturer_name'].describe()
```

```
count           38531
unique             55
top        Volkswagen
freq             4243
Name: manufacturer_name, dtype: object
```

```
print("As object: ", used_cars['manufacturer_name'].nbytes)
print("As category: ", used_cars['manufacturer_name'].astype('category').nbytes)
```

```
As object: 308248
As category: 38971
```

[1] https://pandas.pydata.org/pandas-docs/stable/user_guide/categorical.html

# Little memory savings

```
used_cars['odometer_value'].astype('object').describe()
```

```
count      38531
unique      6063
top       300000
freq         1794
Name: odometer_value, dtype: int64
```

```
print(f"As float: {used_cars['odometer_value'].nbytes}")
print(f"As category: {used_cars['odometer_value'].astype('category').nbytes}")
```

```
As float: 308248
As category: 125566
```

# Using categories can be frustrating

- Using the `.str` accessor object to manipulate data converts the Series to an object.
- The `.apply()` method outputs a new Series as an object.
- The common methods of adding, removing, replacing, or setting categories do not all handle missing categories the same way.
- NumPy functions generally do not work with categorical Series.

# Check and convert

## Check

```
used_cars["color"] = used_cars["color"].astype("category")
used_cars["color"] = used_cars["color"].str.upper()
print(used_cars["color"].dtype)
```

```
object
```

## Convert

```
used_cars["color"] = used_cars["color"].astype("category")
print(used_cars["color"].dtype)
```

```
category
```

# Look for missing values

Set categories

```
used_cars["color"] = used_cars["color"].astype("category")
used_cars["color"].cat.set_categories(["black", "silver", "blue"], inplace=True)
used_cars["color"].value_counts(dropna=False)
```

```
NaN       18172
black      7705
silver     6852
blue       5802
Name: color, dtype: int64
```

# Using numpy arrays

```
used_cars['number_of_photos'] = used_cars['number_of_photos'].astype("category")
used_cars['number_of_photos'].sum()  # <--- Gives an Error
```

```
TypeError: Categorical cannot perform the operation sum
```

```
used_cars['number_of_photos'].astype(int).sum()
```

Note:

```
# .str converts the column to an array
used_cars["color"].str.contains("red")
```

```
0        False
1        False
...
```

# Pitfall practice

WORKING WITH CATEGORICAL DATA IN PYTHON

# Label encoding

WORKING WITH CATEGORICAL DATA IN PYTHON

**Kasey Jones**
Research Data Scientist

datacamp

# What is label encoding?

**The basics:**

- Codes each category as an integer from 0 through `n` - 1, where `n` is the number of categories
- A `-1` code is reserved for any missing values
- Can save on memory
- Often used in surveys

**The drawback:**

- Is not the best encoding method for machine learning (see next lesson)

# Creating codes

Convert to categorical and sort by manufacturer name

```
used_cars['manufacturer_name'] = used_cars['manufacturer_name'].astype("category")
```

Use `.cat.codes`

```
used_cars['manufacturer_code'] = used_cars['manufacturer_name'].cat.codes
```

# Check output

```
print(used_cars[['manufacturer_name', 'manufacturer_code']])
```

```
      manufacturer_name  manufacturer_code
0                Subaru                 45
1                Subaru                 45
2                Subaru                 45
...                 ...                ...
38526          Chrysler                  8
38527          Chrysler                  8
```

# Code books / data dictionaries

**Survey Year(s): 2013**

| | |
|---|---|
| **Topic** | Admin |
| **Description** | New construction in last 4 years |
| **Table Name** | NEWHOUSE |
| **Type** | Character |
| **Edit Flag Variable** | NA |
| **Imputation Strategy** | |
| **Response Codes** | 1: Yes<br>2: No |

[1] https://www.census.gov/data-tools/demo/codebook/ahs/ahsdict.html

# Creating a code book

```
codes = used_cars['manufacturer_name'].cat.codes
categories = used_cars['manufacturer_name']
```

```
name_map = dict(zip(codes, categories))
print(name_map)
```

```
{45: 'Subaru',
 24: 'LADA',
 12: 'Dodge',
 ...
}
```

# Using a code book

Creating the codes:

```
used_cars['manufacturer_code'] = used_cars['manufacturer_name'].cat.codes
```

Reverting to previous values:

```
used_cars['manufacturer_code'].map(name_map)
```

```
0         Acura
1         Acura
2         Acura
...
```

[1] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.map.html

# Boolean coding

Find all body types that have "van" in them:

```
# Code from previous lesson:
used_cars["body_type"].str.contains("van", regex=False)
```

Create a boolean coding:

```
used_cars["van_code"] = np.where(
  used_cars["body_type"].str.contains("van", regex=False), 1, 0)
used_cars["van_code"].value_counts()
```

```
0    34115
1     4416
Name: van_code, dtype: int64
```

# Encoding practice

WORKING WITH CATEGORICAL DATA IN PYTHON

# One-hot encoding

## WORKING WITH CATEGORICAL DATA IN PYTHON

**Kasey Jones**
Research Data Scientist

datacamp

# Why not just label encoding?

```
used_cars["engine_fuel"] = used_cars["engine_fuel"].astype("category")
codes = used_cars["engine_fuel"].cat.codes
categories = used_cars["engine_fuel"]
dict(zip(codes, categories))
```

```
{3: 'gasoline',
 2: 'gas',
 0: 'diesel',
 5: 'hybrid-petrol',
 4: 'hybrid-diesel',
 1: 'electric'}
```

# One-hot encoding with pandas

`pd.get_dummies()`

- `data` : a `pandas` DataFrame
- `columns` : a list-like object of column names
- `prefix` : a string to add to the beginning of each category

# One-hot encoding on a DataFrame

```
used_cars[["odometer_value", "color"]].head()
```

Example output:

```
   odometer_value   color
0          190000  silver
1          290000    blue
2          402000     red
3           10000    blue
4          280000   black
...
```

# One-hot encoding on a DataFrame continued

```
used_cars_onehot = pd.get_dummies(used_cars[["odometer_value", "color"]])
used_cars_onehot.head()
```

```
   odometer_value  color_black  color_brown  color_green ...
0          190000            0            0            0 ...
1          290000            0            0            0 ...
2          402000            0            0            0 ...
3           10000            0            0            0 ...
4          280000            1            0            0 ...
```

```
print(used_cars_onehot.shape)
```

```
(38531, 13)
```

# Specifying columns to use

```
used_cars_onehot = pd.get_dummies(used_cars, columns=["color"], prefix="")
used_cars_onehot.head()
```

```
      manufacturer_name ...  _black  _blue  _brown
0                Subaru ...       0      0       0
1                Subaru ...       0      1       0
2                Subaru ...       0      0       0
3                Subaru ...       0      1       0
4                Subaru ...       1      0       0
```

```
print(used_cars_onehot.shape)
```

```
(38531, 41)
```

# A few quick notes

- Might create too many features

```
used_cars_onehot = pd.get_dummies(used_cars)
print(used_cars_onehot.shape)
```

```
(38531, 1240)
```

- `NaN` values do not get their own column

# One-hot encoding practice

WORKING WITH CATEGORICAL DATA IN PYTHON

# Wrap-up video

## WORKING WITH CATEGORICAL DATA IN PYTHON

**Kasey Jones**
Research Data Scientist

datacamp

# Categorical columns

Our Datasets:

- Incomes
- Trip Reviews
- Shelter Dogs
- Used Cars

# Chapter 1

Topics covered:

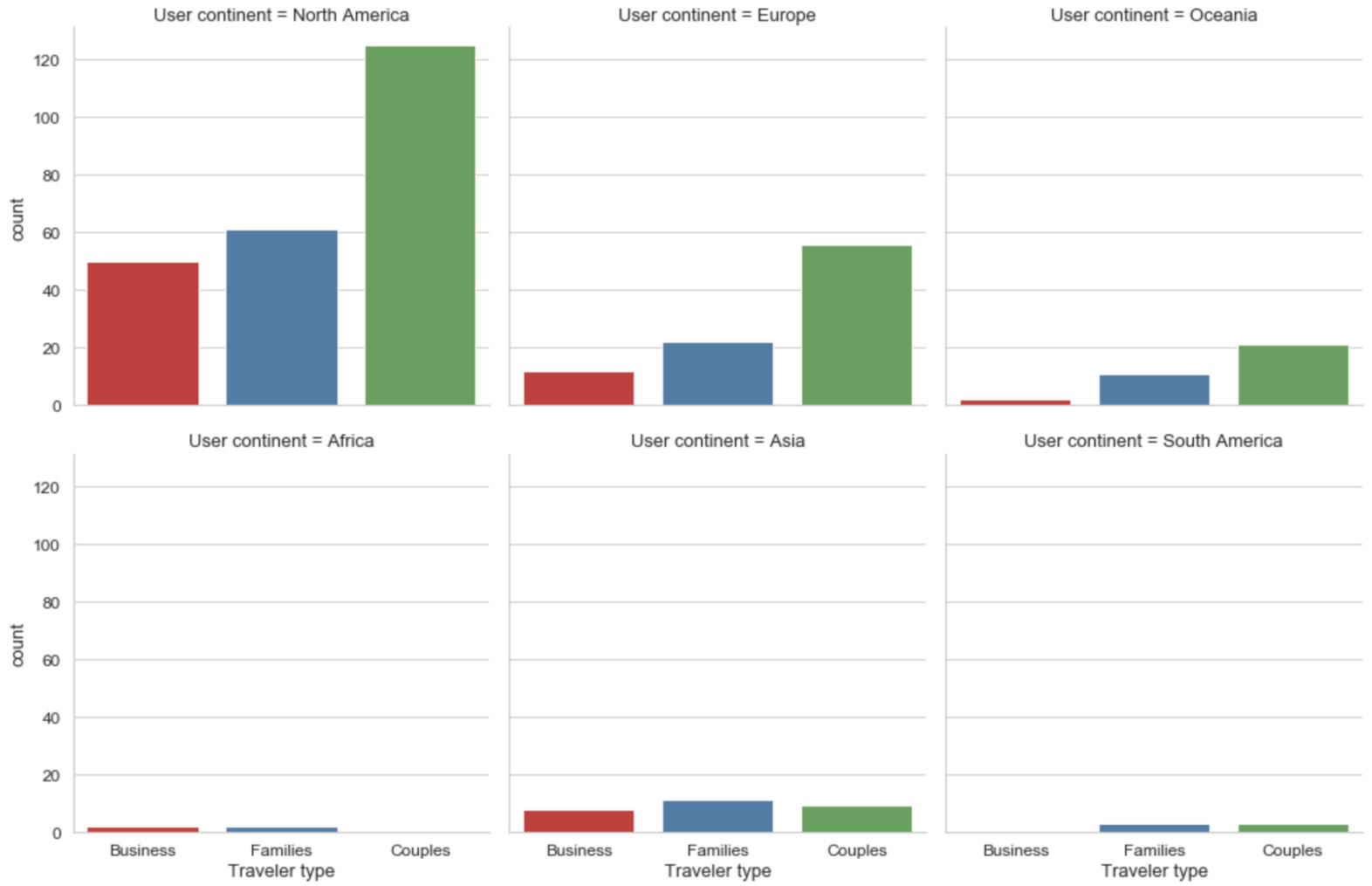
- Nominal vs ordinal columns
- Creating our first categorical column
- `.value_counts()`, as well as `.groupby()`

# Chapter 2

Methods for categorical columns:

- Setting
- Adding
- Removing
- Updating
- Reordering

# Chapter 3


User continent = North America
User continent = Europe
User continent = Oceania
User continent = Africa
User continent = Asia
User continent = South America
count
0
20
40
60
80
100
120
Business
Families
Couples
Traveler type

# Chapter 4

Pitfalls

Encoding examples:

- Label encoding
- One-hot encoding

# Great job!

## WORKING WITH CATEGORICAL DATA IN PYTHON