DTS201TC_Tutorial_3

October 13, 2023

1 DTS201TC Tutorial 3: Data Preprocessing

1.1 Preprocessing of tables

PREPROCESS DATA IN PYTHON STEP-BY-STEP

1. Load data in Pandas.
2. Drop columns that aren't useful.
3. Drop rows with missing values (or fill Na).
4. Convert the data frame to NumPy.
5. Divide the data set into training data and test data.

About the dataset: Only over 700 passengers survived in the sinking of the Titanic. Although luck played a role, perhaps certain groups had a higher chance of survival. We can establish a predictive model and test our hypothesis using their information, such as age, gender, economic class, etc.

Ref: https://www.kaggle.com/competitions/titanic/overview

1.1.1 Load csv with pandas

To work on the data, you can either load the CSV in Excel or in Pandas. For the purposes of this tutorial, we'll load the CSV data in Pandas.

[4]:
```
import numpy as np
import pandas as pd
```

[7]:
```
# load the dataset
df = pd.read_csv('train.csv')

# take a look at the data format
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
```

```
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[8]:
```
# basic infomration
df.describe()
```

[8]:

	PassengerId	Survived	Pclass	Age	SibSp	\
count	891.000000	891.000000	891.000000	714.000000	891.000000	
mean	446.000000	0.383838	2.308642	29.699118	0.523008	
std	257.353842	0.486592	0.836071	14.526497	1.102743	
min	1.000000	0.000000	1.000000	0.420000	0.000000	
25%	223.500000	0.000000	2.000000	20.125000	0.000000	
50%	446.000000	0.000000	3.000000	28.000000	0.000000	
75%	668.500000	1.000000	3.000000	38.000000	1.000000	
max	891.000000	1.000000	3.000000	80.000000	8.000000	

	Parch	Fare
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200
75%	0.000000	31.000000
max	6.000000	512.329200

1.1.2 Drop unuseful columns

In the second step, we can drop some of the columns which won't contribute much to our machine learning model. We'll start with Name, Ticket and Cabin.

[9]:
```
# drop unuseful columns
cols = ['Name', 'Ticket', 'Cabin']
df = df.drop(cols, axis=1)
```

[10]:
```
# re-check the data format
# three columns have been dropped
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
```

1.1.3 Drop rows with missing data

If you carefully observe the above summary of Pandas, there are 891 total rows but Age shows only 714 (which means we're missing some data), Embarked is missing two rows and Cabin is missing a lot as well.

[11]:
```
df = pd.read_csv('train.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[12]:
```
df.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 183 entries, 1 to 889
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  183 non-null    int64
 1   Survived     183 non-null    int64
 2   Pclass       183 non-null    int64
 3   Name         183 non-null    object
 4   Sex          183 non-null    object
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64
 7   Parch        183 non-null    int64
 8   Ticket       183 non-null    object
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object
 11  Embarked     183 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

After dropping rows with missing values, we find the data set is reduced to 183 rows from 891, which means we are wasting data.

1.1.4 Fill Na

[8]:
```
df = pd.read_csv('train.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[9]:
```
# delete unecessary columns
cols = ['Name', 'Ticket', 'Cabin']
df = df.drop(cols, axis=1)
# fill Na with the average of that row
df.fillna(df.mean()).info()
# any other better values?
# any other better mehtods?
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    object
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
```

```
/var/folders/qc/tbxsysmj1v31__b2v69tf34m0000gn/T/ipykernel_18977/3988535515.py:5
: FutureWarning: The default value of numeric_only in DataFrame.mean is
deprecated. In a future version, it will default to False. In addition,
specifying 'numeric_only=None' is deprecated. Select only valid columns or
specify the value of numeric_only to silence this warning.
  df.fillna(df.mean()).info()
```

1.1.5 Convert the data frame to numpy

Now that we've converted all the data to integers, it's time to prepare the data for machine learning models. This is where scikit-learn and NumPy come into play:

x = input set with 8 arrtributes

y = whether survived

Now we convert our data frame from Pandas to NumPy and we assign input and output:

[10]:
```
X = df.values
y = df['Survived'].values
```

X still has Survived values in it, which should not be there. So we drop in the NumPy column, which is the first column.

[11]:
```
X = np.delete(X, 1, axis=1)
```

1.1.6 Divide the dataset into training set and test data

Now that we're ready with X and y, let's split the data set: we'll allocate 70 percent for training and 30 percent for tests using scikit model_selection.

You can start to build your own models and making predictions.

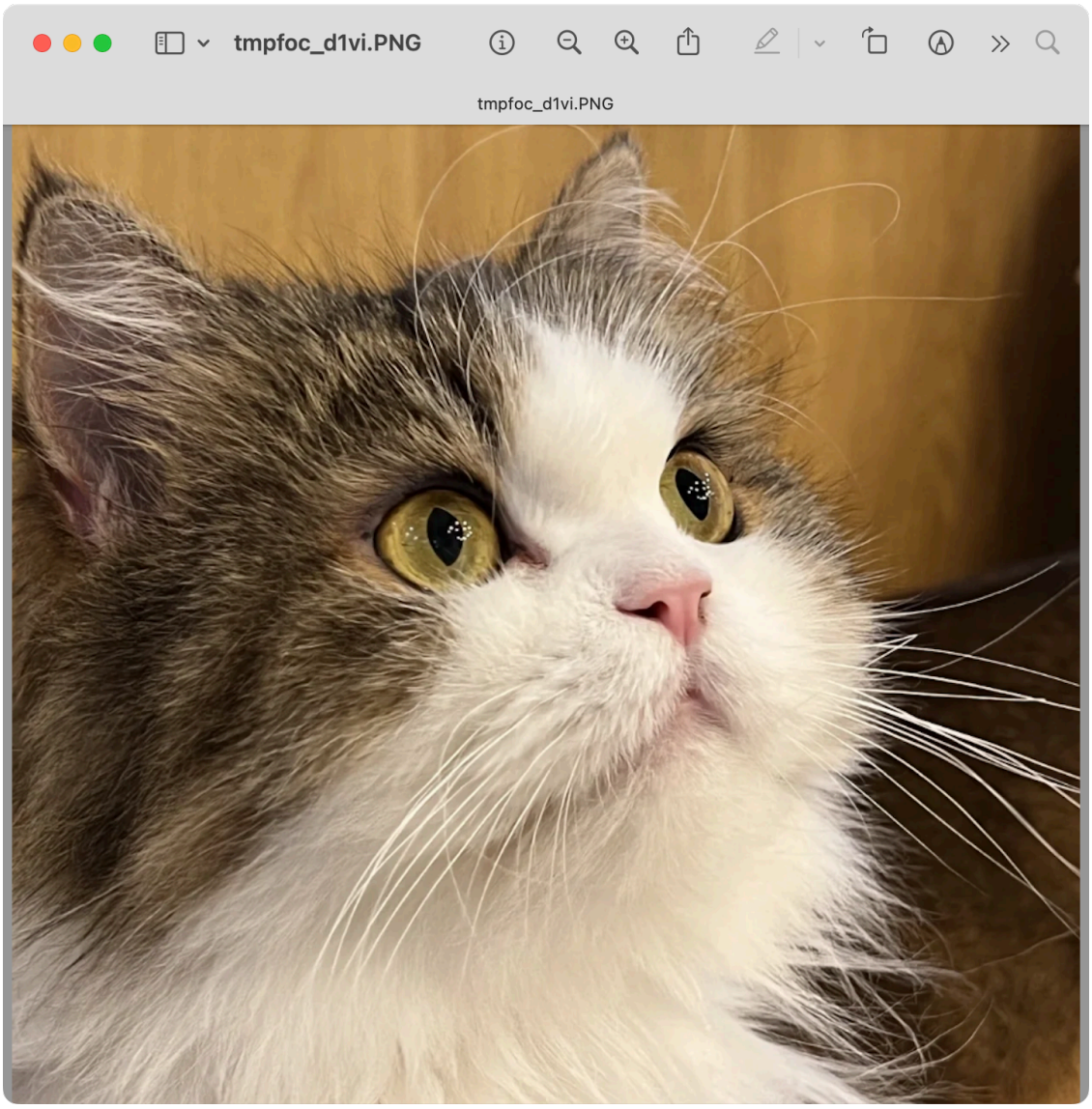
[19]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=0)
```

1.2 Preprocessing of images

1.2.1 Read image

The second part of today's lab is to learn the preprocessing skills of images. There are two methods we can use to load an image.

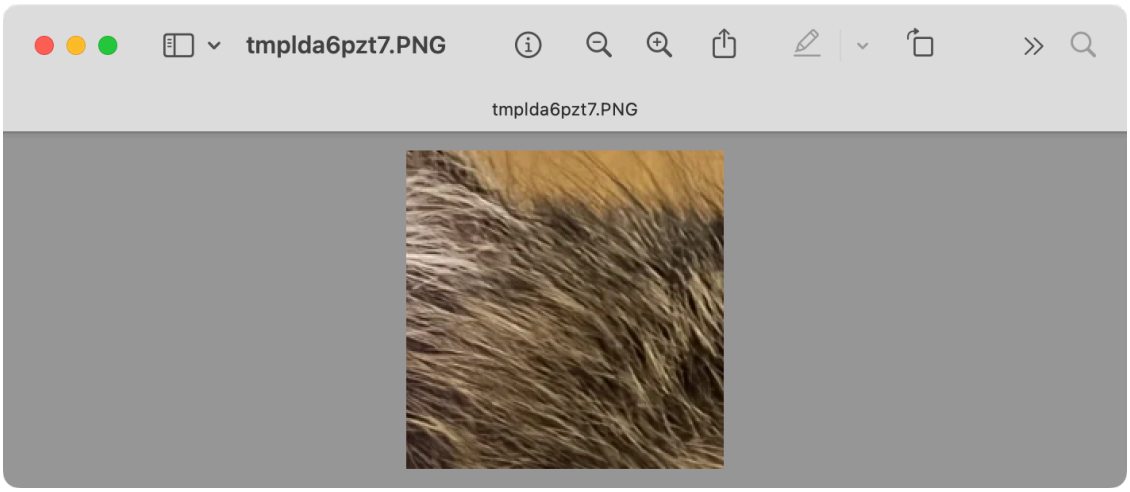
[20]:
```
# pip install Pillow

from PIL import Image
img = Image.open('cat.png')

img.show()

# or,
# pip install opencv-python
# import cv2
# img = cv2.imread('cat.png')
```



1.2.2 Crop

You can use `crop()` to crop unnecessary areas, reduce noise and increase the clarity of the image.

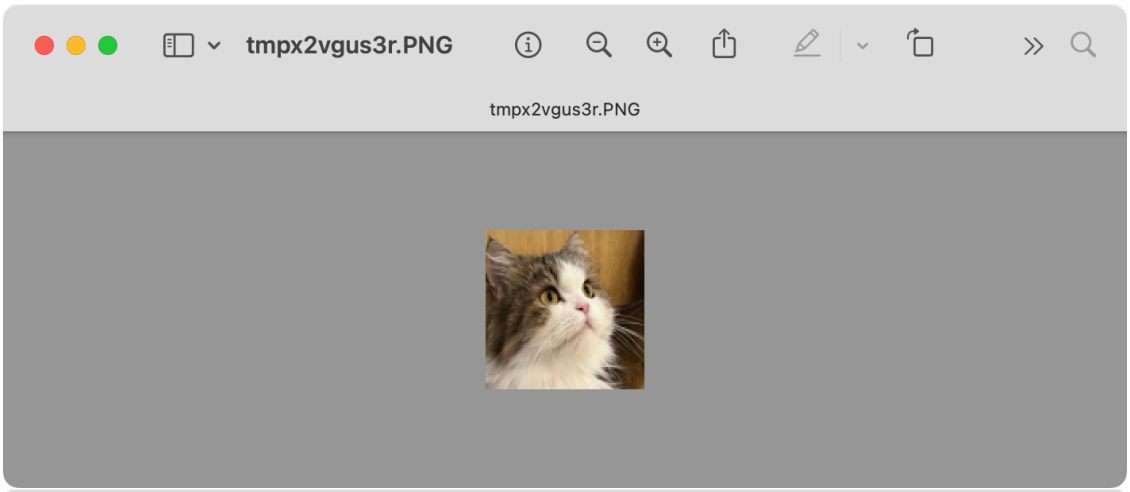
[21]:
```
left = 100
upper = 100
right = 300
lower = 300
img_crop = img.crop((left,upper,right,lower))
img_crop.show()
```

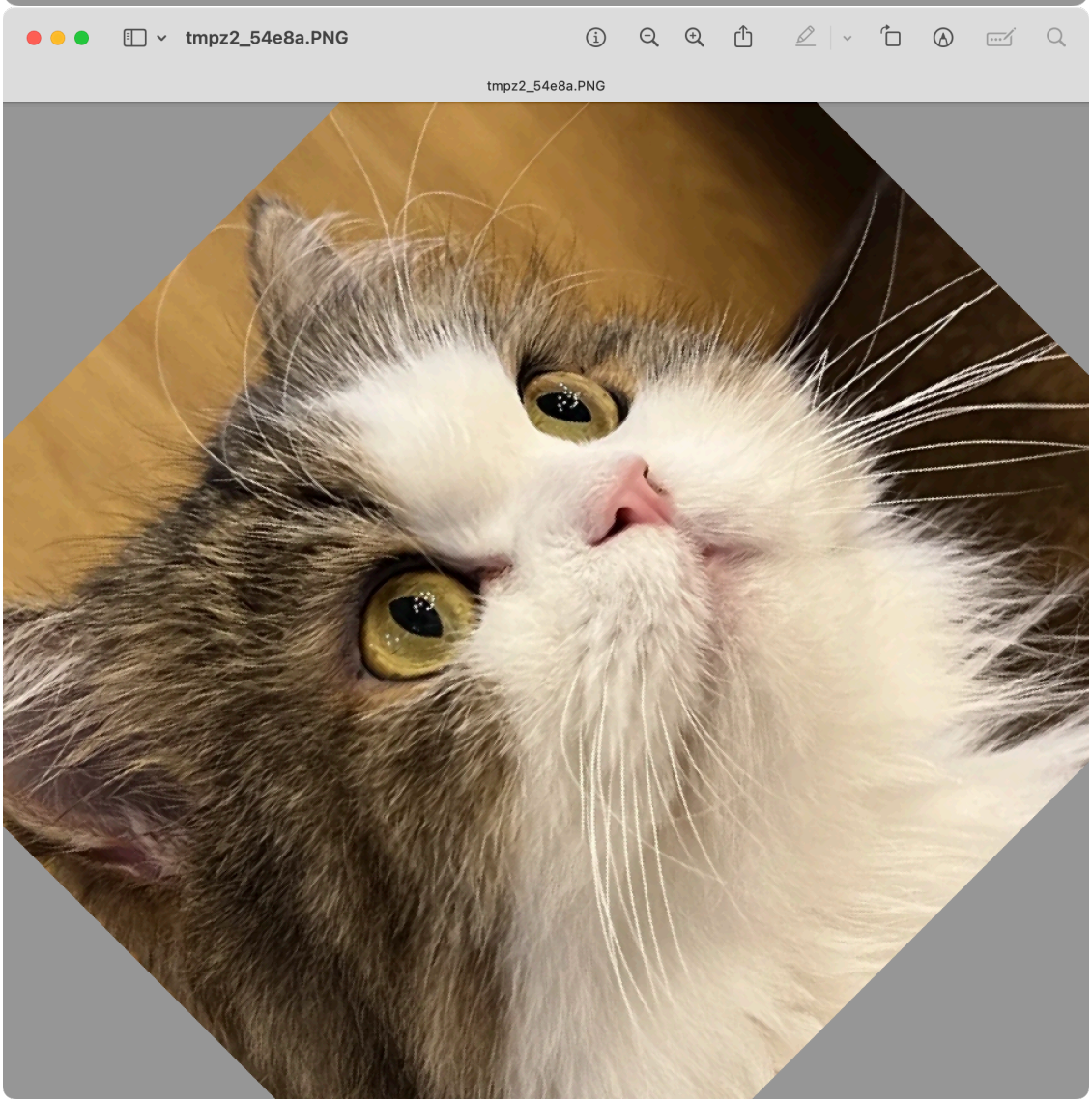


1.2.3 Transformation: Resize & Rotate

`resize()` and `rotate()` are used to transform the images. The former uses a tuple to specify the new size of images, while the latter rotates images in a counterclockwise direction.

[22]:
```
img_resize = img.resize((100,100))
img_resize.show()

img_rotate = img.rotate(45)
img_rotate.show()
```

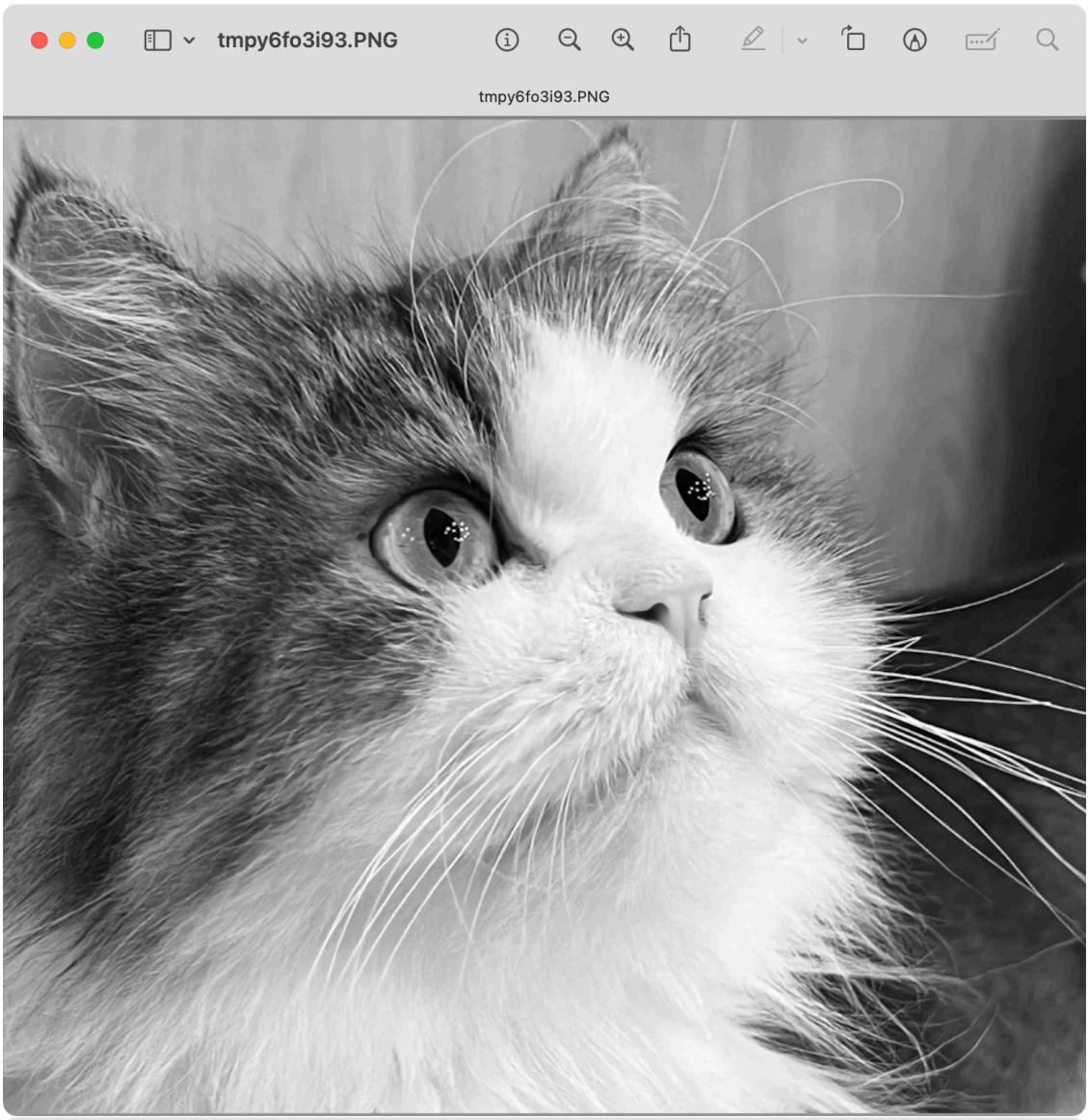

tmpx2vgus3r.PNG
tmpx2vgus3r.PNG

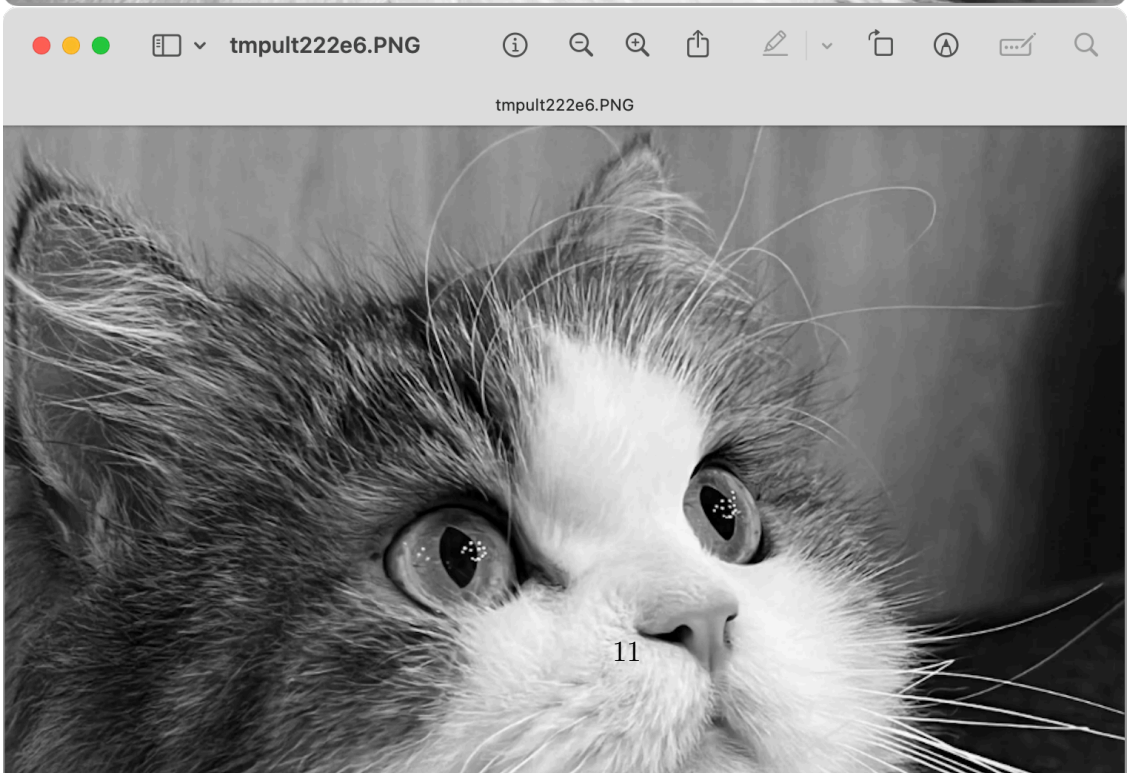

tmpz2_54e8a.PNG
tmpz2_54e8a.PNG

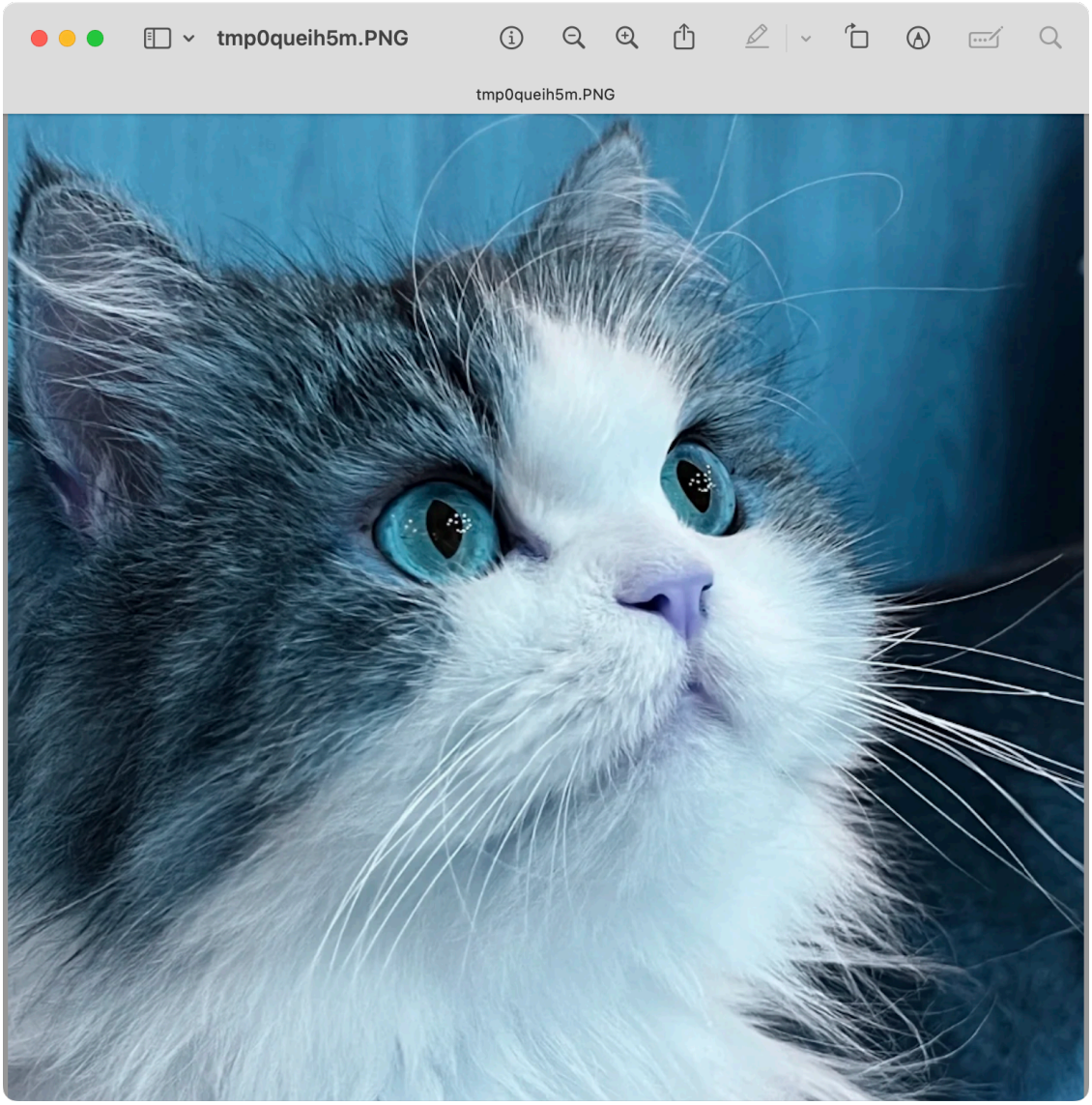
1.2.4 Image color channel separation

Pillow allows processing of various channels in an image, which may have four channels: R, G, B, A. 'A' is the first letter of Alpha which refers to the transparency and translucency of an image. You may also use `merge()` to combine a new image.

[23]:
```
r,g,b,a = img.split()
r.show()
g.show()

img_new = Image.merge("RGBA", (b,g,r,a))
img_new.show()
```


tmpy6fo3i93.PNG
tmpy6fo3i93.PNG

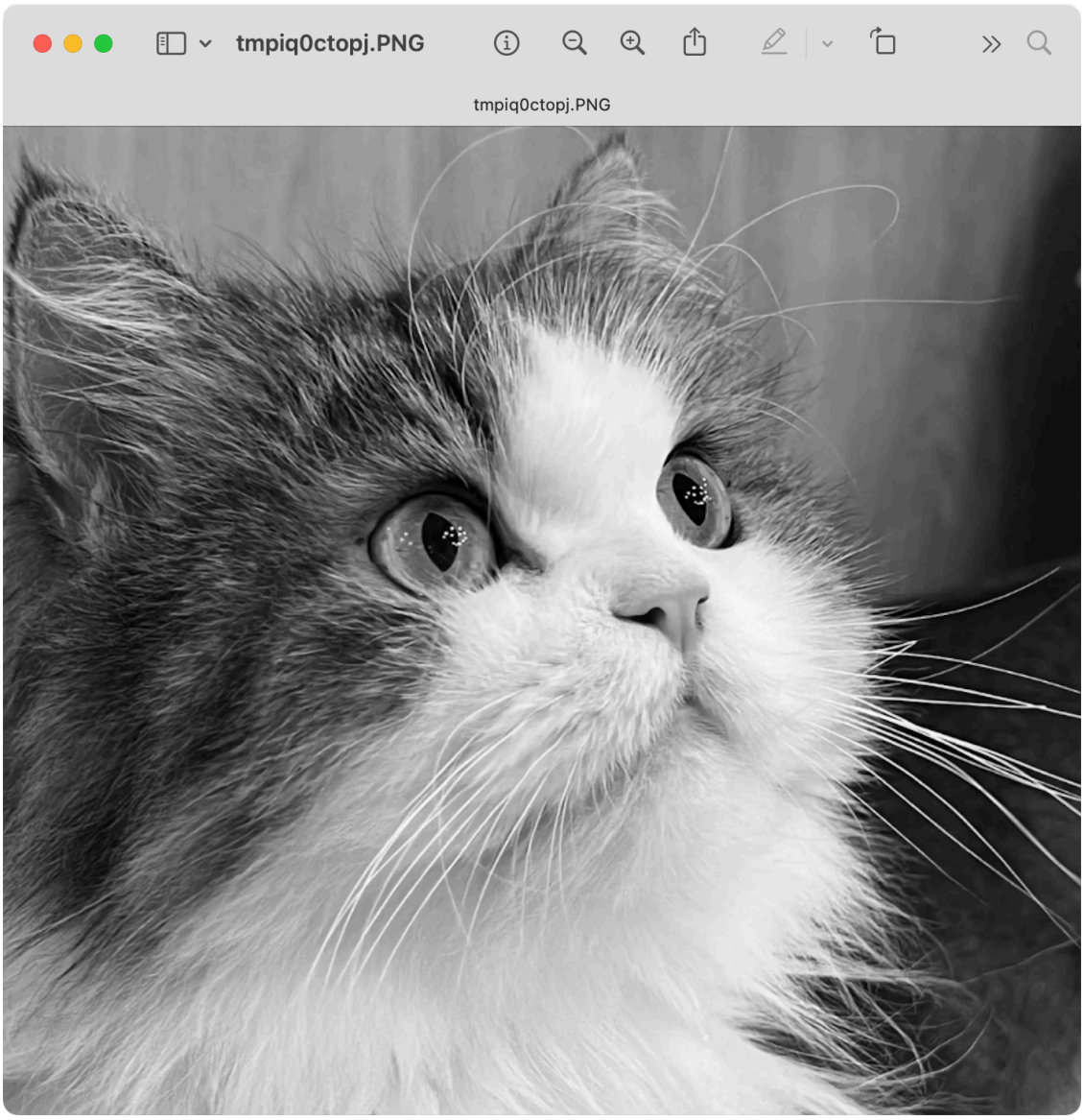

tmpult222e6.PNG
tmpult222e6.PNG



1.2.5 Convert to grayscale image

At last, the function of `convert()` transforms images into a grayscale version, where colors are replaced with shades of gray.

[24]:
```
img_convert = img.convert("L")
img_convert.show()
```



1.3 More functions and practices:

opencv: https://docs.opencv.org/4.8.0/d2/d96/tutorial_py_table_of_contents_imgproc.html

Pillow: https://pillow.readthedocs.io/en/latest/reference/Image.html#image-processing