eda

May 13, 2023

[1]:
```
import os
import pandas as pd
import altair as alt
from exif import Image
import altair_saver
from altair_saver import save
import selenium

# prof joel's code to save altair plots
import vl_convert as vlc

def save_chart(chart, filename, scale_factor=1):
    '''
    Save an Altair chart using vl-convert

    Parameters
    ----------
    chart : altair.Chart
        Altair chart to save
    filename : str
        The path to save the chart to
    scale_factor: int or float
        The factor to scale the image resolution by.
        E.g. A value of `2` means two times the default resolution.
    '''
    if filename.split('.')[-1] == 'svg':
        with open(filename, "w") as f:
            f.write(vlc.vegalite_to_svg(chart.to_dict()))
    elif filename.split('.')[-1] == 'png':
        with open(filename, "wb") as f:
            f.write(vlc.vegalite_to_png(chart.to_dict(), scale=scale_factor))
    else:
        raise ValueError("Only svg and png formats are supported")

# make outputs folder
output_path = r'../tmp/proposal/'
if not os.path.exists(output_path):
```

```
    os.makedirs(output_path)
```

[2]:
```
## creating list of metadata
dir_path = "../tmp/ICBC_imgs"

metadata_list = []
filename_list = []

for folders in os.listdir(dir_path):
    if folders != '.DS_Store':
        next_dir = os.path.join(dir_path, folders)
        for filename in os.listdir(next_dir):
            file_path = os.path.join(next_dir, filename)
            with open(file_path, 'rb') as img_file:
                img = Image(img_file)
                if img.has_exif==True:
                    metadata = img.get('user_comment')
        # Add the metadata to the list and filename
                filename_list.append(filename)
                metadata_list.append(metadata)
```

[3]:
```
file_path = os.path.join(dir_path, '2023 Photos', "processed_April 2023_243.
 ↪jpg")
img = Image(open(file_path, 'rb') )
img.has_exif==True
# metadata# = img.get('user_comment')
```

[3]:
```
True
```

[4]:
```
images_df = pd.DataFrame({'file_name': filename_list, 'metadata' :␣
 ↪metadata_list})
images_df.head()
```

[4]:
```
                          file_name  \
0  processed_December 2019_2329.jpg
1   processed_December 2019_351.jpg
2  processed_December 2019_3751.jpg
3  processed_December 2019_3989.jpg
4  processed_December 2019_2301.jpg

                                        metadata
0      {make:'PONTIAC', model:'VIBE', year:2007}
1       {make:'HONDA', model:'CIVIC', year:2012}
2         {make:'HONDA', model:'FIT', year:2015}
3    {make:'MERCEDES', model:'GL350', year:2012}
4  {make:'VOLKSWAGEN', model:'RABIT', year:2007}
```

[5]:
```
images_df[images_df['file_name'].str.contains('1247')]
```

[5]:
```
                              file_name  \
933       processed_April 2019_1247.jpg
2953      processed_April 2021_1247.jpg
5256   processed_February 2022_1247.jpg
7413      processed_April 2019_1247.jpg
18041  processed_BEU Completed_1247.jpg

                                       metadata
933    {make:'LEXUS', model:'RX350', year:2016}
2953    {make:'JEEP', model:'LBRTY', year:2003}
5256    {make:'FORD', model:'MUSTG', year:1997}
7413   {make:'LEXUS', model:'RX350', year:2016}
18041   {make:'DODGE', model:'DART', year:1965}
```

[6]:
```
# create a data frame with make, model year as separate columns

data_dicts = []
for info in metadata_list:
    info = info.strip('{}')
    items = info.split(', ')
    info_dict = {}
    for item in items:
        key, value = item.split(':')
        key = key.strip("'")
        if key == 'year':
            value = int(value)
        else:
            value = value.strip("'")
        info_dict[key] = value
    data_dicts.append(info_dict)

metadata_df = pd.DataFrame(data_dicts)
metadata_df['make_model'] = metadata_df['make'] + ' ' + metadata_df['model']
```

[7]:
```
metadata_df.head()
```

[7]:
```
         make  model  year        make_model
0     PONTIAC   VIBE  2007      PONTIAC VIBE
1       HONDA  CIVIC  2012       HONDA CIVIC
2       HONDA    FIT  2015         HONDA FIT
3    MERCEDES  GL350  2012    MERCEDES GL350
4  VOLKSWAGEN  RABIT  2007  VOLKSWAGEN RABIT
```

[8]:
```
# top 3 brands
sum(metadata_df['make'].isin(['HONDA', 'TOYOTA', 'FORD']))/len(metadata_df)
```

[8]:
```
0.36957663620128167
```

[9]:
```
metadata_df.to_csv(os.path.join(output_path,'metadata_df.csv'), index=False)
```

[10]:
```
# summaries

#unique make, model and year
unique_counts = metadata_df.nunique()
print(unique_counts,'\n')

#unique combinations of make, model and year
unique_counts2 = images_df.nunique()[0]
print("There are", unique_counts2, " unique cars",'\n')

#number of images in the first photo folder
print("There are", len(metadata_list)," images in total",'\n')

#the count of each unique model
model_counts = metadata_df['make_model'].value_counts()
print(model_counts,'\n')
```

```
make           94
model         802
year           68
make_model    885
dtype: int64

There are 18908  unique cars

There are 19038  images in total

FORD F150          833
HONDA CIVIC        674
TOYOTA CROLA       499
HONDA CR-V         469
DODGE/RAM 1500     446
                  …
BMW 318ti            1
MERCEDES CLK63       1
SATURN SC1           1
INFINITI FX45        1
MG MGB               1
Name: make_model, Length: 885, dtype: int64
```

[11]:
```
# Count the occurrences of each unique make and convert it to a DataFrame
year_counts = metadata_df['year'].value_counts().reset_index()
year_counts.columns = ['year', 'count']
```

```
all_years = pd.DataFrame({'year': range(1934, 2022)})
year_counts = all_years.merge(year_counts, how='left', on='year').fillna(0)
```

[12]:
```
# Create the bar chart
chart = alt.Chart(year_counts).mark_bar().encode(
    x=alt.X('year:O', title='Year', axis=alt.Axis(
        labelExpr='datum.label % 5 == 0 ? datum.label : ""',  # show label only␣
 ↪for every 5th year
        tickSize=alt.condition(
            'datum.label % 5 == 0',
            alt.value(10),  # set tick size to 10 for every 5th year
            alt.value(5)  # set tick size to 5 otherwise
        )
    )),
    # x=alt.X('year:O', title='Year', axis=alt.Axis(labelExpr='datum.label % 5␣
 ↪== 0 ? datum.label : "",)),
    y=alt.Y('count:Q', title='Count'),
    color=alt.Color('count:Q', scale=alt.Scale(scheme='warmgreys')),
    tooltip=['year', 'count']
).properties(
    title='Distribution of Manufacturing Years',
    width=500,
    height=200)

rule = alt.Chart(pd.DataFrame({'year': [2001]})).mark_rule(color='red', size=3.
 ↪5).encode(x='year:O')
combined_chart = (chart + rule).configure_axis(
    titleFontSize=20,
    labelFontSize=15,
    titleFontWeight='normal').configure_axisX(
    labelAngle=-45).configure_legend(
    disable=True).configure_title(
    fontSize=20)

combined_chart
```

[12]:
```
alt.LayerChart(…)
```

[13]:
```
save_chart(combined_chart, os.path.join(output_path, "report_Fig2.png"))
```

[14]:
```
# Count the occurrences of each unique make and convert it to a DataFrame
make_counts = metadata_df['make'].value_counts().reset_index()
make_counts.columns = ['make', 'count']

# Get the top 10 makes (makes that are over 200) and their counts
top_makes = make_counts.nlargest(10, 'count')
```

```
# Calculate the 'others' count
others_count = make_counts[~make_counts['make'].
 ↪isin(top_makes['make'])]['count'].sum()

# Add the 'others' row to the top_makes DataFrame
top_makes = top_makes.append({'make': 'others', 'count': others_count},␣
 ↪ignore_index=True)

# Calculate the total count for percentage calculation
total_count = top_makes['count'].sum()

# Create the pie chart
pie_chart = alt.Chart(top_makes).transform_calculate(
    percentage='datum.count / %d' % total_count
).transform_window(
    rank='rank(count)',
    sort=[alt.SortField('count', order='descending')]
).mark_arc(innerRadius=50, outerRadius=100, cornerRadius=5).encode(
    theta=alt.Theta('count:Q', stack=True),
    color=alt.Color('make:N',  scale=alt.Scale(scheme='tableau20'),
                    legend=alt.Legend(title='Make'),
                    sort=alt.EncodingSortField(field='theta',␣
 ↪order='descending')),
    tooltip=['make', 'count', alt.Tooltip('percentage:Q', format='.2%')],
    order=alt.Order('count:Q', sort='descending'),
).properties(
    title={
        "text": "Distribution of IMDB ratings by genre",
        "fontSize": 20, # Set title font size
    },
    width=350,
    height=300
).configure_legend(
    labelFontSize=15, # Set legend font size
    titleFontSize=14, # Set legend title font size
).configure_axis(
    titleFontSize=15, # Set axis title font size
)

# .properties(
#     title={
#         "text": "Distribution of IMDB ratings by genre",
#         "fontSize": 16, # Set title font size
#     }
# ).configure_legend(
#     labelFontSize=12, # Set legend font size
```

```
#     titleFontSize=14, # Set legend title font size
# ).configure_axis(
#     titleFontSize=14, # Set axis title font size
# )


# Display the pie chart
pie_chart.display()
```

```
/var/folders/2f/jgypjqv54pdd8_f2_jcv_b300000gn/T/ipykernel_9335/3093433952.py:12
: FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  top_makes = top_makes.append({'make': 'others', 'count': others_count},
ignore_index=True)
```

```
alt.Chart(…)
```

[15]:
```
save_chart(pie_chart, os.path.join(output_path, "report_Fig4.png"))
```

[16]:
```
from PIL import Image, ImageEnhance
import numpy as np
from multiprocessing import Pool, cpu_count
import os

from process_image import *

folder_path = "../tmp/ICBC_imgs/"
image_files = [os.path.join(dp, f) for dp, dn, fn in os.walk(folder_path) for f␣
 ↪in fn if f.endswith('.jpg')]

# Create a pool of workers
with Pool(cpu_count()) as p:
    quality_data = p.map(process_image, image_files)
```

```
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
```

```
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
/opt/miniconda3/envs/icbc/lib/python3.10/site-packages/PIL/Image.py:3176:
DecompressionBombWarning: Image size (108000000 pixels) exceeds limit of
89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
```

[17]:

```
# Create a DataFrame with the results
image_quality_df = pd.DataFrame(quality_data)

# save the dataframe as a CSV file
image_quality_df.to_csv(os.path.join(output_path, 'image_quality_df.csv'),␣
↪index=False)
```

[18]:

```
alt.data_transformers.disable_max_rows()
# Create a histogram for the average contrast
avg_contrast_chart = alt.Chart(image_quality_df).mark_bar().encode(
    alt.X('avg_contrast:Q', bin=alt.Bin(maxbins=30), title='Contrast'),
    alt.Y('count()', title='Count'),
    tooltip=['count()']
).properties(
    title='Image Contrast',
    width=300,
    height=200
)



# Display the charts
# avg_contrast_chart.display()

save_chart(avg_contrast_chart, os.path.join(output_path, "report_Fig5a.png"))
```

[19]:

```
# Create new columns for width and height in the DataFrame
image_quality_df['width'] = image_quality_df['resolution'].apply(lambda x: x[0])
image_quality_df['height'] = image_quality_df['resolution'].apply(lambda x:␣
↪x[1])
# Create a scatter plot for the resolution
resolution_chart = alt.Chart(image_quality_df).mark_circle(size=20).encode(
    alt.X('width:Q', title='Width'),
```

```
    alt.Y('height:Q', title='Height'),
    tooltip=['file_name', 'width', 'height']
).properties(
    title='Image Resolutions',
    width=300,
    height=200
)

# Display the chart
# resolution_chart.display()

save_chart(resolution_chart, os.path.join(output_path, "report_Fig5b.png"))
```