Web usage mining

Aim:

Program:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import pandas as pd
import re

# Define the log file path
log_file_path = '/kaggle/input/web-server-access-logs/access.log'

# Define the regex pattern to extract information from log lines
regex_pattern = r'^(?P<client>\S+) \S+ (?P<userid>\S+) \[(?P<datetime>[\w:/]+\s[+\-]\d{4}
)\] "(?P<method>[A-Z]+) (?P<request>[^ "]+)? HTTP/[0-9.]+" (?P<status>[0-9]{3}) (?P<siz
e>[0-9]+|-) "(?P<referer>[^"]*)" "(?P<user_agent>[^"]*)"'

# Define the column names
columns = ['client', 'userid', 'datetime', 'method', 'request', 'status', 'size', 'referer', 'user
_agent']

# Read the first 10000 rows of the log file into a list of dictionaries using regex pattern matc
hing
log_data = []
```

```
with open(log_file_path, 'r') as file:
    for i, line in enumerate(file):
        if i >= 10000:
            break
        match = re.match(regex_pattern, line)
        if match:
            log_data.append({
                'client': match.group('client'),
```

```
                        'userid': match.group('userid'),
                        'datetime': match.group('datetime'),
                        'method': match.group('method'),
                        'request': match.group('request'),
                        'status': match.group('status'),
                        'size': match.group('size'),
                        'referer': match.group('referer'),
                        'user_agent': match.group('user_agent')
                    })
            else:
                    print("Error: Line does not match regex pattern:", line)

# Create DataFrame from the list of dictionaries
logs_df = pd.DataFrame(log_data, columns=columns)
```

```
from datetime import datetime
import pytz
```

```
def parse_datetime(x):
    try:
        dt = datetime.strptime(x[1:-7], '%d/%b/%Y:%H:%M:%S')
        dt_tz = int(x[-6:-3])*60+int(x[-3:-1])
        return dt.replace(tzinfo=pytz.FixedOffset(dt_tz))
    except ValueError:
        return '-'
```

```
logs_df['status'] = logs_df['status'].astype(int)
logs_df['size'] = logs_df['size'].astype(int)
logs_df['datetime'] = logs_df['datetime'].apply(parse_datetime)
```

```
logs_df.drop(columns=['userid'], inplace=True)
```

```
duplicate_count = logs_df.duplicated().sum()
# Display the count of duplicates
print("Number of duplicates:", duplicate_count)
# Drop the duplicates
logs_df = logs_df.drop_duplicates()
```

```
logs_df.head()
```

	client	userid	datetime	method	request	status	siz
0	54.36.149.41	-	22/Jan/2019:03:56:14 +0330	GET	/filter/27|13%20%D9%85%DA%AF%D8%A7%D9%BE%DB%8C...	200	30
1	31.56.96.51	-	22/Jan/2019:03:56:16 +0330	GET	/image/60844/productModel/200x200	200	56
2	31.56.96.51	-	22/Jan/2019:03:56:16 +0330	GET	/image/61474/productModel/200x200	200	53
3	40.77.167.129	-	22/Jan/2019:03:56:17 +0330	GET	/image/14925/productModel/100x100	200	16
4	91.99.72.15	-	22/Jan/2019:03:56:17 +0330	GET	/product/31893/62100/%D8%B3%D8%B4%D9%88%D8%A7%...	200	41

Q1. 10 people who visited the site frequently

```
frequent_visitors = logs_df.groupby(['client', 'user_agent']).size().reset_index(name='count
').sort_values(by='count', ascending=False)

# Select the top 10 frequent visitors
top_10 = frequent_visitors.head(10)

index = 0
# Display the top 3 frequent visitors
for i, row in top_3.iterrows():
    print(f"{index + 1}. Client: {row['client']}, User Agent: {row['user_agent']}, Count: {ro
w['count']}\n")
    index += 1
```

```
1. Client: 66.249.66.194, User Agent: Mozilla/5.0 (Linux; Android 6.0.1; N
exus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2
272.96 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google
.com/bot.html), Count: 778

2. Client: 66.249.66.91, User Agent: Mozilla/5.0 (compatible; Googlebot/2.
1; +http://www.google.com/bot.html), Count: 739

3. Client: 130.185.74.243, User Agent: Mozilla/5.0 (Windows NT 6.1; rv:42.
0) Gecko/20100101 Firefox/42.0, Count: 660
```

Q2. Sessions and the page views per each session

```
sessions = logs_df.groupby(['client', 'user_agent'])

session_info = []
for (client, user_agent), session_data in sessions:
    # Extract timestamps and page views for the session
    timestamps = session_data['datetime'].tolist()
    pages = session_data['request'].tolist()

    # Store session information in a tuple
    session_info.append((client, user_agent, timestamps, pages))
```

```
# Display at least 3 sessions and their page views per session
for i, (client, user_agent, timestamps, pages) in enumerate(session_info[:3], start=1):
    print(f"Session {i} - Client: {client}, User Agent: {user_agent}")
    for timestamp, page in zip(timestamps, pages):
        print(f"    Timestamp: {timestamp}, Page: {page}")
    print()
```

```
Session 1 - Client: 104.156.210.196, User Agent: Dalvik/2.1.0 (Linux; U; A
ndroid 8.0.0; SM-A720F Build/R16NW)
    Timestamp: 2019-01-02 04:20:00+00:33,

Session 2 - Client: 104.194.24.33, User Agent: Mozilla/5.0 (Linux; Android
8.0.0; SM-G955F) AppleWebKit/
    Timestamp: 2019-01-02 03:57:00+00:33,

Session 3 - Client: 104.194.24.54, User Agent: Dalvik/2.1.0 (Linux; U; And
roid 6.0.1; SM-G900H Build/MMB29K)
    Timestamp: 2019-01-02 04:24:00+00:33,
```

```
SESSION_THRESHOLD_SECONDS = 10 * 60

# Sort the logs_df by client, user_agent, and datetime
logs_df_sorted = logs_df.sort_values(by=['client', 'user_agent', 'datetime'])

# Initialize empty lists to store session information
session_info = []

# Initialize variables for tracking sessions
current_client = None
current_user_agent = None
current_session_start = None
current_session_end = None
current_session_pages = []
```

```
for index, row in logs_df_sorted.iterrows():
    # Check if the client or user_agent has changed, or if the time gap exceeds the threshold
    if (row['client'] != current_client or row['user_agent'] != current_user_agent or
            (current_session_start and (row['datetime'] - current_session_end).seconds > SESSION_THRESHOLD_SECONDS)):
        # If so, store the current session information
        if current_session_start:
            session_info.append((current_client, current_user_agent, current_session_start, current_session_end, current_session_pages))
            # Check if we have at least five sessions, if so, break the loop
            if len(session_info) >= 5:
                break

        # Start a new session
        current_client = row['client']
        current_user_agent = row['user_agent']
        current_session_start = row['datetime']
        current_session_end = row['datetime']
        current_session_pages = [(row['datetime'], row['request'])]
    else:
        # Otherwise, add the page to the current session
        current_session_pages.append((row['datetime'], row['request']))
        # Update session end time
        current_session_end = row['datetime']

# Display session information for at least 3 sessions
index = 0
for session in session_info:
    print(f"Session {index+1}")
    print("Client:", session[0])
    print("User Agent:", session[1])
    print("Session Start Time:", session[2])
    print("\n\n")
```

```
Session 1
Client: 104.156.210.196
User Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; SM-A720F Build/R16NW)
Session Start Time: 2019-01-02 04:20:00+00:33

Session 2
Client: 104.194.24.33
User Agent: Mozilla/5.0 (Linux; Android 8.0.0; SM-G955F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Mobile Safari/537.36
Session Start Time: 2019-01-02 03:57:00+00:33
```

```
session_df = pd.DataFrame(session_info)
# Set the columns
session_df.columns = ['client', 'user_agent', 'start_time', 'end_time', 'pages']
session_df['pages'] = session_df['pages'].apply(lambda x: [page[1] for page in x])
session_df.head(2)
```

	client	user_agent	start_time	end_time	pages
0	104.156.210.196	Dalvik/2.1.0 (Linux; U; Android 8.0.0; SM-A720...	2019-01-02 04:20:00+00:33	2019-01-02 04:20:00+00:33	[/image/32768?name=24xs450-33.jpg&wh=200x200]
1	104.194.24.33	Mozilla/5.0 (Linux; Android 8.0.0; SM-G955F) A...	2019-01-02 03:57:00+00:33	2019-01-02 03:57:00+00:33	[/amp-helper-frame.html?appId=a624a1c1-0c93-46...

Q3. **Pages that are frequently visited together with a support ratio not less than 25%**

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

pages_accessed = session_df['pages'].tolist()
te = TransactionEncoder()
onehot = te.fit_transform(pages_accessed)

# Convert the one-hot encoded DataFrame into a DataFrame
df = pd.DataFrame(onehot, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.25, use_colnames=True)
```

```
filter_frequent_itemsets = frequent_itemsets[frequent_itemsets['itemsets'].apply(lambda x: len(x)) > 1]
# Display frequent itemsets
print("Frequent Itemsets:")
filter_frequent_itemsets
```

	support	itemsets
2	0.4	(/image/33888?name=model-b2048u-1-.jpg&wh=200x200, /image/11947?name=11947-1-fw.jpg&wh=200x200)

Q5. Association rules with lift values not less than 2.05

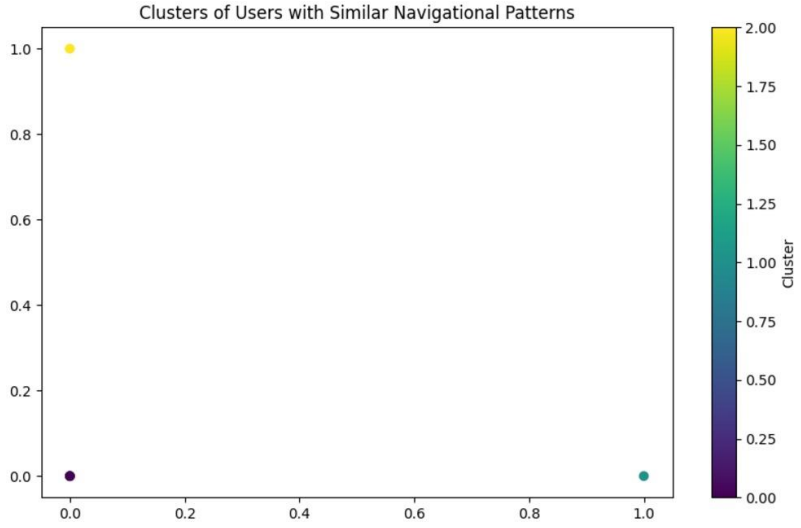
```
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=2.05)
print("Association Rules with Lift > 2.05:\n")
for index, rule in rules.iterrows():
    antecedents = ', '.join(list(rule['antecedents']))
    consequents = ', '.join(list(rule['consequents']))
    support = rule['support']
```

```
    confidence = rule['confidence']
    lift = rule['lift']
    print(f"Rule {index+1}: {antecedents} -> {consequents}")
    print(f"Support: {support:.4f}, Confidence: {confidence:.4f}, Lift: {lift:.4f}\n")
```

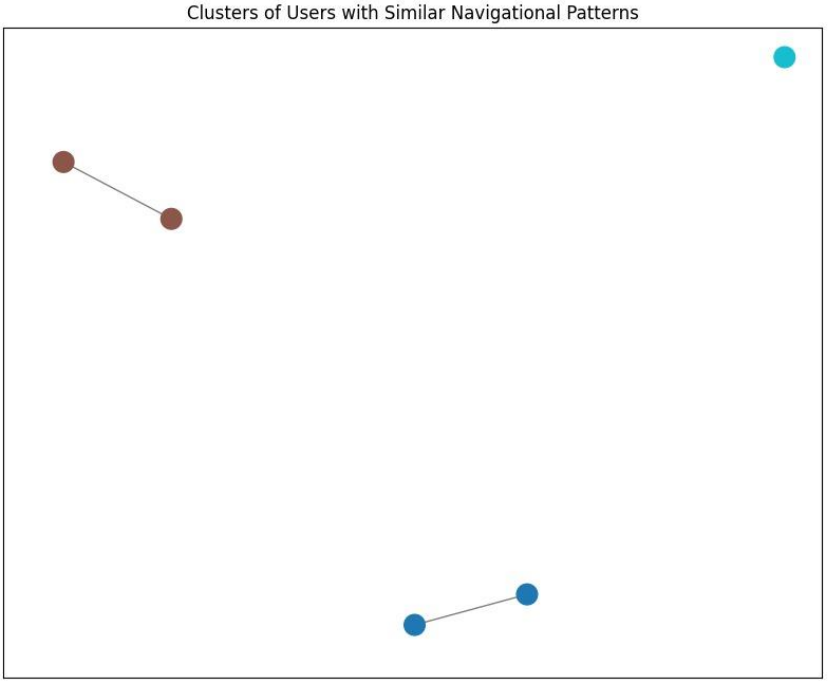
```
Association Rules with Lift > 2.05:

Rule 1: /image/33888?name=model-b2048u-1-.jpg&wh=200x200 -> /image/11947?n
ame=11947-1-fw.jpg&wh=200x200
Support: 0.4000, Confidence: 1.0000, Lift: 2.5000

Rule 2: /image/11947?name=11947-1-fw.jpg&wh=200x200 -> /image/33888?name=m
odel-b2048u-1-.jpg&wh=200x200
Support: 0.4000, Confidence: 1.0000, Lift: 2.5000
```

Q7. Graph that shows clusters of users with similar navigational patterns

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

patterns = onehot
kmeans = KMeans(n_clusters=3, n_init=10)
clusters = kmeans.fit_predict(patterns)
plt.figure(figsize=(10, 6))
plt.scatter(patterns[:, 0], patterns[:, 1], c=clusters, cmap='viridis')
plt.title('Clusters of Users with Similar Navigational Patterns')
plt.colorbar(label='Cluster')
plt.show()
```



```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
for i in range(len(patterns)):
    # Assign cluster as a node attribute
    G.add_node(i, label=f"User {i}", cluster=clusters[i])
for i in range(len(patterns)):
    for j in range(i + 1, len(patterns)):
        if clusters[i] == clusters[j]:
            G.add_edge(i, j)


plt.figure(figsize=(10, 8))
pos = nx.spring_layout(G)

node_color = [clusters[node] for node in G.nodes()]
nx.draw_networkx_nodes(G, pos, node_color=node_color, cmap=plt.cm.tab10, node_size=200)

# Draw edges
nx.draw_networkx_edges(G, pos, alpha=0.5)

plt.title('Clusters of Users with Similar Navigational Patterns')
# plt.axis('off')
plt.show()
```



Access log Analysis

Aim:

Program:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import re
import os
import time
from tqdm import tqdm
common_regex = '^(?P<client>\S+) \S+ (?P<userid>\S+) \[(?P<datetime>[^\]]+)\] "(?P<method>[A-Z]+) (?P<request>[^ "]+)? HTTP/[0-9.]+" (?P<status>[0-9]{3}) (?P<size>[0-9]+|-)'
combined_regex = '^(?P<client>\S+) \S+ (?P<userid>\S+) \[(?P<datetime>[^\]]+)\] "(?P<method>[A-Z]+) (?P<request>[^ "]+)? HTTP/[0-9.]+" (?P<status>[0-9]{3}) (?P<size>[0-9]+|-) "(?P<referrer>[^"]*)" "(?P<useragent>[^"]*)'
columns = ['client', 'userid', 'datetime', 'method', 'request', 'status', 'size', 'referer', 'user_agent']
```

```
def logs_to_df(logfile, output_dir, errors_file):
    with open(logfile) as source_file:
        linenumber = 0
        parsed_lines = []
        for line in tqdm(source_file):
            try:
                log_line = re.findall(combined_regex, line)[0]
                parsed_lines.append(log_line)
            except Exception as e:
                with open(errors_file, 'at') as errfile:
                    print((line, str(e)), file=errfile)
                continue
            linenumber += 1
            if linenumber % 250_000 == 0:
```

```
                    df = pd.DataFrame(parsed_lines, columns=columns)
                    df.to_parquet(f'{output_dir}/file_{linenumber}.parquet')
                    parsed_lines.clear()
            else:
                df = pd.DataFrame(parsed_lines, columns=columns)
                df.to_parquet(f'{output_dir}/file_{linenumber}.parquet')
                parsed_lines.clear()
mkdir parquet_dir
logs_to_df(logfile='/kaggle/input/web-server-access-logs/access.log', output_dir='parquet_dir/', errors_file='errors.txt')
```

```
10365152it [02:26, 70811.95it/s]
```

```
logs_df = pd.read_parquet('parquet_dir/')
logs_df
```

	client	userid	datetime	method	request	status	size	referer
0	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/image/29314?name=%D8%AF%DB%8C%D8%A8%D8%A7-7.j...	200	1105	https://www.zanbil.ir/
1	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/zanbil-kharid.png	200	358	https://www.zanbil.ir/
2	85.9.73.119	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/next.png	200	3045	https://znbl.ir/static/b
3	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/image/29314?name=%D8%AF%DB%8C%D8%A8%D8%A7-4.j...	200	1457	https://www.zanbil.ir/
4	85.9.73.119	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/checked.png	200	1083	https://znbl.ir/static/b
...	...	...	...	...	...	...	...	...
10364860	86.104.110.254	-	26/Jan/2019:16:01:31 +0330	GET	/settings/logo	200	4120	https://www.zanbil.ir/
10364861	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/5/brand	200	2171	https://www.zanbil.ir/
10364862	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/64646/productModel/150x150	200	5318	https://www.zanbil.ir/

```
df = logs_df.query("request.str.contains('.css') == False and request.str.contains('.png') == False and request.str.contains('.jpg') == False and request.str.contains('.jpeg') == False and request.str.contains('.mp3') == False and request.str.contains('.js') == False")
df
```

	client	userid	datetime	method	request	status	size	referer
5	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/loading.gif	200	7370	https://www.zanbil.ir/prod
6	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/11082/productType/240x180	200	12458	https://www.zanbil.ir/brov
7	37.27.128.139	-	22/Jan/2019:12:38:27 +0330	GET	/browse/Tablet-Arm-Chair/%D8%B5%D9%86%D8%AF%D9...	200	30604	https://www.zanbil.ir/brov
8	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/851/mainSlide	200	89859	https://www.zanbil.ir/brov
9	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/848/mainSlide	200	93168	https://www.zanbil.ir/brov
...	...	...	...	...	...	...	...	...
10364860	86.104.110.254	-	26/Jan/2019:16:01:31 +0330	GET	/settings/logo	200	4120	https://www.zanbil.ir/m/br
10364861	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/5/brand	200	2171	https://www.zanbil.ir/m/fil
10364862	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/64646/productModel/150x150	200	5318	https://www.zanbil.ir/brov
10364863	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/1/brand	200	3924	https://www.zanbil.ir/m/fil
10364864	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/56698/productModel/150x150	200	3570	https://www.zanbil.ir/brov

```
df2 = df[df['method'].str.contains("POST") == False]
df2
```

	client	userid	datetime	method	request	status	size	referer
5	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/loading.gif	200	7370	https://www.zanbil.ir/prod
6	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/11082/productType/240x180	200	12458	https://www.zanbil.ir/brov
7	37.27.128.139	-	22/Jan/2019:12:38:27 +0330	GET	/browse/Tablet-Arm-Chair/%D8%B5%D9%86%D8%AF%D9...	200	30604	https://www.zanbil.ir/brov
8	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/851/mainSlide	200	89859	https://www.zanbil.ir/brov
9	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/848/mainSlide	200	93168	https://www.zanbil.ir/brov
...	...	...	...	...	...	...	...	...
10364860	86.104.110.254	-	26/Jan/2019:16:01:31 +0330	GET	/settings/logo	200	4120	https://www.zanbil.ir/m/br
10364861	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/5/brand	200	2171	https://www.zanbil.ir/m/fil
10364862	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/64646/productModel/150x150	200	5318	https://www.zanbil.ir/brov
10364863	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/1/brand	200	3924	https://www.zanbil.ir/m/fil
10364864	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/56698/productModel/150x150	200	3570	https://www.zanbil.ir/brov

```
df3 = df2.query("status.str.contains('200') == True")
df3
```

	client	userid	datetime	method	request	status	size	referer
5	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/loading.gif	200	7370	https://www.zanbil.ir/prod
6	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/11082/productType/240x180	200	12458	https://www.zanbil.ir/brov
7	37.27.128.139	-	22/Jan/2019:12:38:27 +0330	GET	/browse/Tablet-Arm-Chair/%D8%B5%D9%86%D8%AF%D9...	200	30604	https://www.zanbil.ir/brov
8	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/851/mainSlide	200	89859	https://www.zanbil.ir/brov
9	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/848/mainSlide	200	93168	https://www.zanbil.ir/brov
...	...	...	...	...	...	...	...	...
10364860	86.104.110.254	-	26/Jan/2019:16:01:31 +0330	GET	/settings/logo	200	4120	https://www.zanbil.ir/m/br
10364861	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/5/brand	200	2171	https://www.zanbil.ir/m/fil
10364862	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/64646/productModel/150x150	200	5318	https://www.zanbil.ir/brov
10364863	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/1/brand	200	3924	https://www.zanbil.ir/m/fil
10364864	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/56698/productModel/150x150	200	3570	https://www.zanbil.ir/brov

```
df3.client
```

```
5             37.152.163.59
6             77.245.233.52
7             37.27.128.139
8             77.245.233.52
9             77.245.233.52
                 ...
10364862       65.49.68.192
10364863      5.125.254.169
10364864       65.49.68.192
```

```
df3.client.nunique()
```

```
147899
```

```
print(df3.groupby('client').get_group('37.152.163.59'))
```

```
                client userid                      datetime method  \
5        37.152.163.59      -  22/Jan/2019:12:38:27 +0330    GET
56       37.152.163.59      -  22/Jan/2019:12:38:27 +0330    GET
186      37.152.163.59      -  22/Jan/2019:12:38:31 +0330    GET
7863700  37.152.163.59      -  22/Jan/2019:12:38:01 +0330    GET
7863704  37.152.163.59      -  22/Jan/2019:12:38:01 +0330    GET
...                ...    ...                         ...    ...
9828922  37.152.163.59      -  26/Jan/2019:12:57:13 +0330    GET
9828925  37.152.163.59      -  26/Jan/2019:12:57:14 +0330    GET
9828928  37.152.163.59      -  26/Jan/2019:12:57:14 +0330    GET
9828991  37.152.163.59      -  26/Jan/2019:12:57:15 +0330    GET
9828993  37.152.163.59      -  26/Jan/2019:12:57:15 +0330    GET

                                                   request status   size
\
5                                  /static/images/loading.gif    200   7370
56                                   /site/alexaGooleAnalitic    200    323
186                                /static/images/favicon.ico    200    152
7863700  /product/29314/%DA%A9%D8%A7%D9%84%D8%B3%DA%A9%...    200  41580
7863704  /image/%7B%7BbasketItem.id%7D%7D?type=productM...    200      5
...                                                    ...    ...    ...
9828922                                      /filter/p62,b5    200  34238
9828993                           /site/alexaGooleAnalitic    200    323
```

```
df3
```

	client	userid	datetime	method	request	status	size	referer
5	37.152.163.59	-	22/Jan/2019:12:38:27 +0330	GET	/static/images/loading.gif	200	7370	https://www.zanbil.ir/prod
6	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/11082/productType/240x180	200	12458	https://www.zanbil.ir/brov
7	37.27.128.139	-	22/Jan/2019:12:38:27 +0330	GET	/browse/Tablet-Arm-Chair/%D8%B5%D9%86%D8%AF%D9...	200	30604	https://www.zanbil.ir/brov
8	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/851/mainSlide	200	89859	https://www.zanbil.ir/brov
9	77.245.233.52	-	22/Jan/2019:12:38:27 +0330	GET	/image/848/mainSlide	200	93168	https://www.zanbil.ir/brov
...	...	...	...	...	...	...	...	...
10364860	86.104.110.254	-	26/Jan/2019:16:01:31 +0330	GET	/settings/logo	200	4120	https://www.zanbil.ir/m/br
10364861	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/5/brand	200	2171	https://www.zanbil.ir/m/fil
10364862	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/64646/productModel/150x150	200	5318	https://www.zanbil.ir/brov
10364863	5.125.254.169	-	26/Jan/2019:16:01:31 +0330	GET	/image/1/brand	200	3924	https://www.zanbil.ir/m/fil
10364864	65.49.68.192	-	26/Jan/2019:16:01:31 +0330	GET	/image/56698/productModel/150x150	200	3570	https://www.zanbil.ir/brov

Clickstream Sales Analysis

Aim:

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
csdata = pd.read_csv('../input/clickstream-data-for-online-shopping/e-shop clothing 2008.csv',
                     delimiter = ';')
print(csdata.shape)
csdata.head(3)
```

```
(165474, 14)
```

	year	month	day	order	country	session ID	page 1 (main category)	page 2 (clothing model)	colour	location	model photography	price	price 2	page
0	2008	4	1	1	29	1	1	A13	1	5	1	28	2	1
1	2008	4	1	2	29	1	1	A16	1	6	1	33	2	1
2	2008	4	1	3	29	1	2	B4	10	2	1	52	1	1

Defining Goals and Variables¶

Questions / Goals

1. When do sales peak?
2. What type of clothing sells most? What type of clothing sells most per month?
3. Does a correlation exist between price and page, and, if so, how strongly are price and product placement related?

Predictions

1. I expect sales to peak in June, as buyers purchase clothing for vacation months / outdoor months.
2. No strong feelings / expectations of what to find
3. I believe higher priced items will be located towards the front page, in order to maximize profits.

Chosen Variables

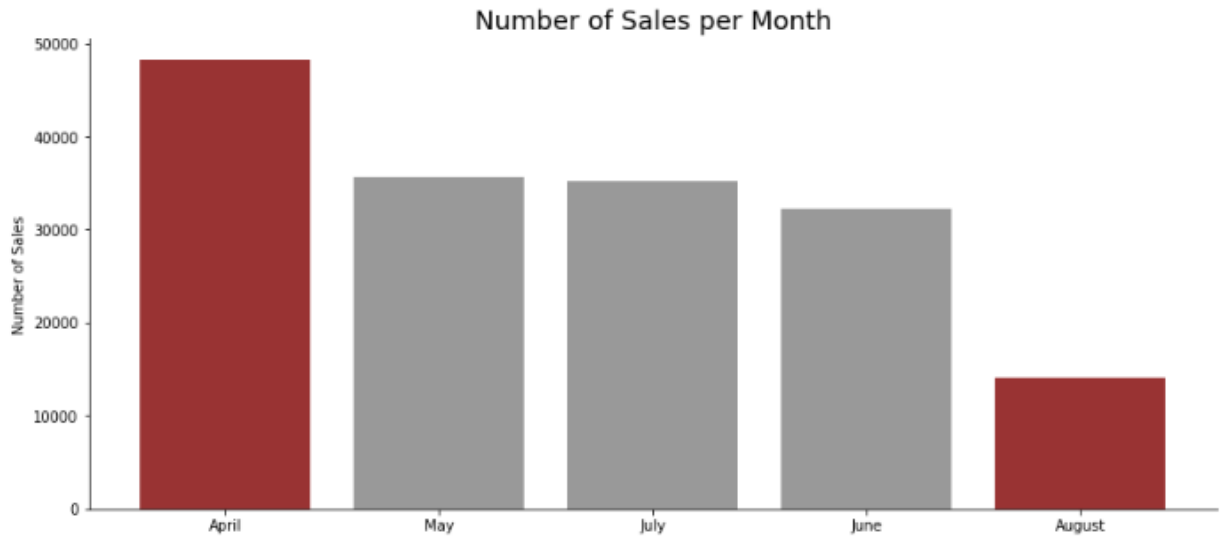
The columns which will be relevant for this analysis are (as defined in the uploaded data):

- MONTH -> from April (4) to August (8)
- DAY -> day number of the month
- PAGE 1 (MAIN CATEGORY) -> concerns the main product category:
    - 1-trousers
    - 2-skirts
    - 3-blouses
    - 4-sale
- PRICE -> price in US dollars
- PAGE -> page number within the e-store website (from 1 to 5)

```
### New dataframe relevant columns
csdf = csdata[['month', 'day', 'page 1 (main category)', 'price', 'page']]
csdf = csdf.rename(columns={'month':'Month', 'day':'Day', 'page 1 (main category)':'Type',
                            'price':'Price', 'page':'Page'})
csdf.Type = csdf.Type.replace({1: 'Trousers', 2: 'Skirts', 3: 'Blouses', 4: 'Sale'})
csdf.Month = csdf.Month.replace({4: 'April', 5: 'May', 6: 'June', 7:'July', 8: 'August'})
csdf.head()
```

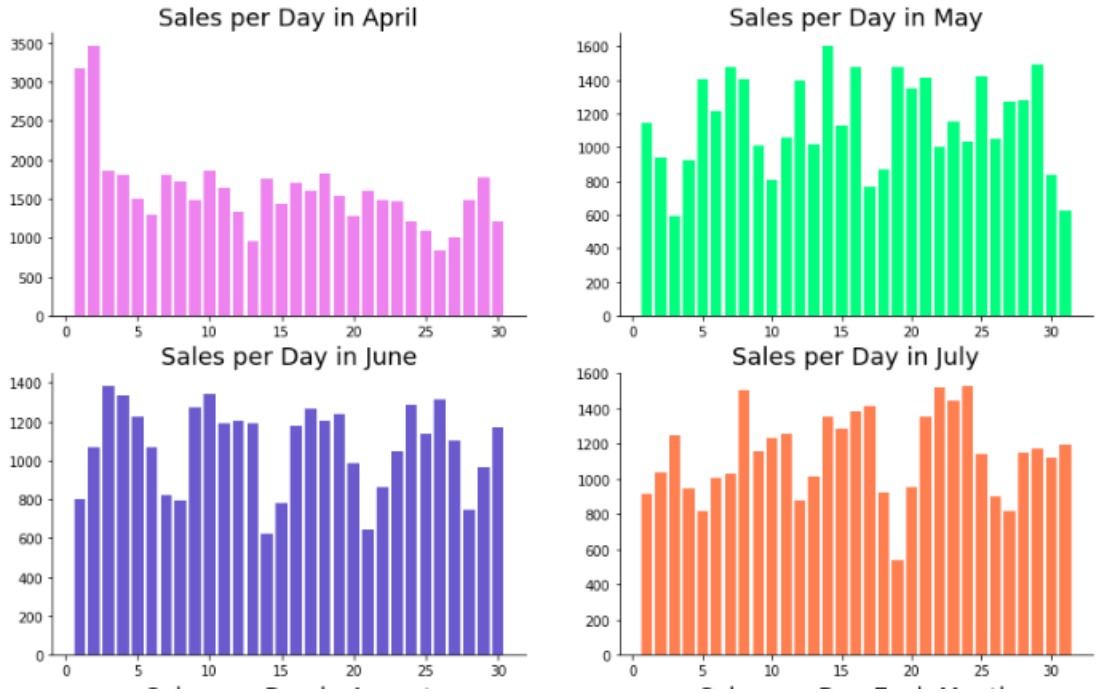
	Month	Day	Type	Price	Page
0	April	1	Trousers	28	1
1	April	1	Trousers	33	1
2	April	1	Skirts	52	1
3	April	1	Skirts	38	1
4	April	1	Skirts	52	1

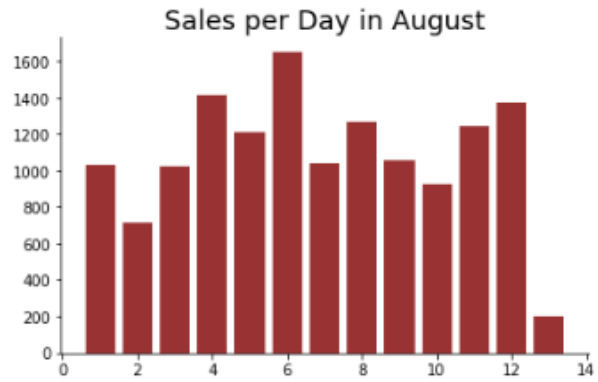
```
### Number of goods sold each month
csmsm = csdf.Month.value_counts()

fig , ax = plt.subplots(figsize = [14,6])

ax.bar(csmsm.keys(), csmsm.values, color=['maroon','gray','gray','gray','maroon'], alpha=.8)
ax.set_title('Number of Sales per Month', fontsize = 18)
ax.set_ylabel('Number of Sales')
ax.spines[['right', 'top']].set_visible(False)

plt.show()
```

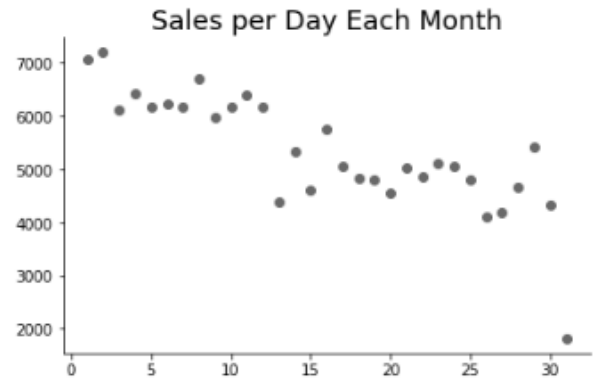


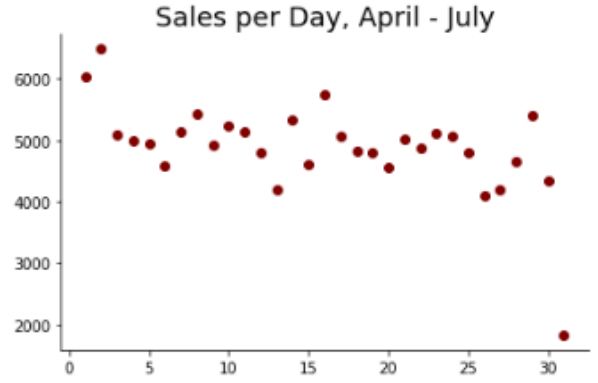

```
### Dataframes for each month
csau = csdf.loc[csdf['Month'] == 'August']
csjn =csdf.loc[csdf['Month'] == 'June']
csjl = csdf.loc[csdf['Month'] == 'July']
csmy = csdf.loc[csdf['Month'] == 'May']
csap = csdf.loc[csdf['Month'] == 'April']

fig, axs = plt.subplots(nrows=4, ncols = 2, figsize=[14,18])

axs[0,0].bar(csap.Day.value_counts().keys(), csap.Day.value_counts().values, color='violet'
)
axs[0,0].set_title('Sales per Day in April', fontsize=18)
axs[0,0].spines[['right', 'top']].set_visible(False)

axs[0,1].bar(csmy.Day.value_counts().keys(), csmy.Day.value_counts().values, color='spri
nggreen')
axs[0,1].set_title('Sales per Day in May', fontsize=18)
axs[0,1].spines[['right', 'top']].set_visible(False)

axs[1,0].bar(csjn.Day.value_counts().keys(), csjn.Day.value_counts().values, color='slatebl
ue')
axs[1,0].set_title('Sales per Day in June', fontsize=18)
axs[1,0].spines[['right', 'top']].set_visible(False)


axs[1,1].bar(csjl.Day.value_counts().keys(), csjl.Day.value_counts().values, color='coral')
axs[1,1].set_title('Sales per Day in July', fontsize=18)
axs[1,1].spines[['right', 'top']].set_visible(False)
```

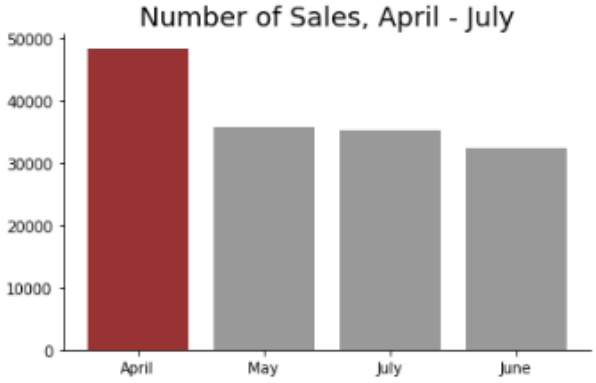
```
axs[2,0].bar(csau.Day.value_counts().keys(), csau.Day.value_counts().values, color='maroon', alpha=.8)
axs[2,0].set_title('Sales per Day in August', fontsize=18)
axs[2,0].spines[['right', 'top']].set_visible(False)


axs[2,1].scatter(csdf.Day.value_counts().keys(), csdf.Day.value_counts().values, color='dimgray')
axs[2,1].set_title('Sales per Day Each Month', fontsize = 18)
axs[2,1].spines[['right', 'top']].set_visible(False)

csna = csdf.loc[csdf['Month'] != 'August']
csnac = csna.Month.value_counts()

axs[3,0].scatter(csna.Day.value_counts().keys(), csna.Day.value_counts().values, color='maroon')
axs[3,0].set_title('Sales per Day, April - July', fontsize = 18)
axs[3,0].spines[['right', 'top']].set_visible(False)

axs[3,1].bar(csnac.keys(), csnac.values, color=['maroon','gray','gray','gray'], alpha=.8)
axs[3,1].set_title('Number of Sales, April - July', fontsize = 18)
axs[3,1].spines[['right', 'top']].set_visible(False)

plt.show()
```



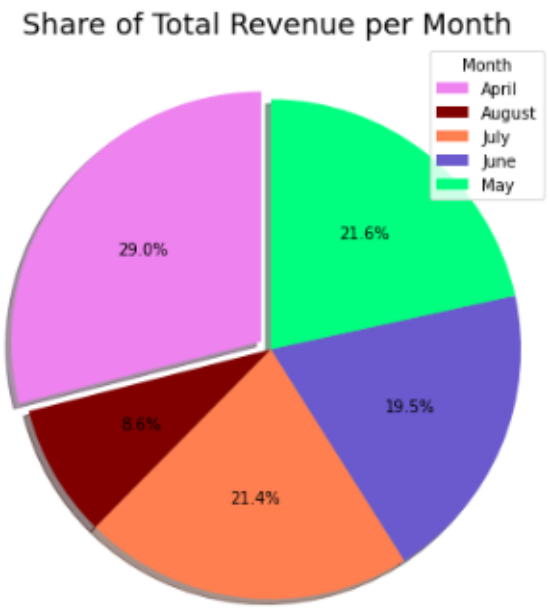


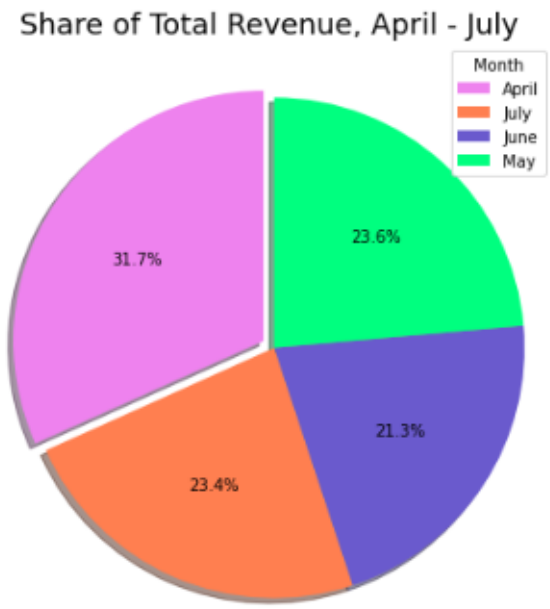




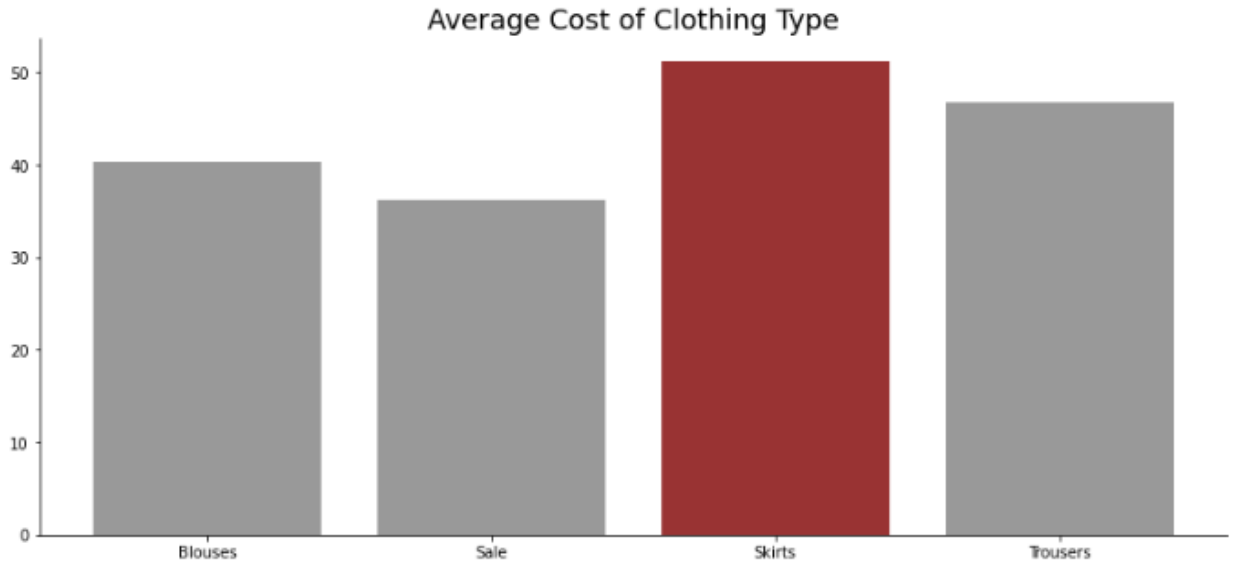



```
csmp = csdf[['Month', 'Price']]
csmpna = csna[['Month', 'Price']]
csmp2 = csmp.groupby('Month').sum()
csmpna2 = csmpna.groupby('Month').sum()

l1 = csmp2.index
l2 = csmpna2.index

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=[14,7])
ax[0].pie(csmp2.Price, explode=(0.05, 0, 0, 0, 0), autopct='%1.1f%%',
          shadow=True, startangle=90, colors=['violet', 'maroon', 'coral', 'slateblue', 'springgree
n'])
ax[0].axis('equal')
ax[0].set_title("Share of Total Revenue per Month", fontsize=18)
ax[0].legend(l1, title="Month", loc="upper right")

ax[1].pie(csmpna2.Price, explode=(0.05, 0, 0, 0), autopct='%1.1f%%',
          shadow=True, startangle=90, colors=['violet', 'coral', 'slateblue', 'springgreen'])
ax[1].axis('equal')
ax[1].set_title("Share of Total Revenue, April - July", fontsize=18)
ax[1].legend(l2, title="Month", loc="upper right")

plt.show()
```

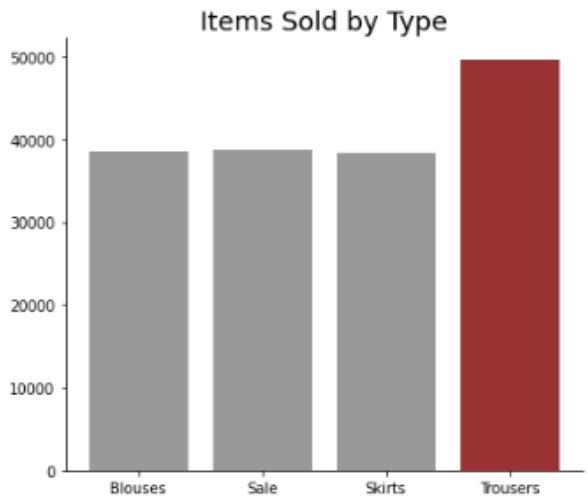


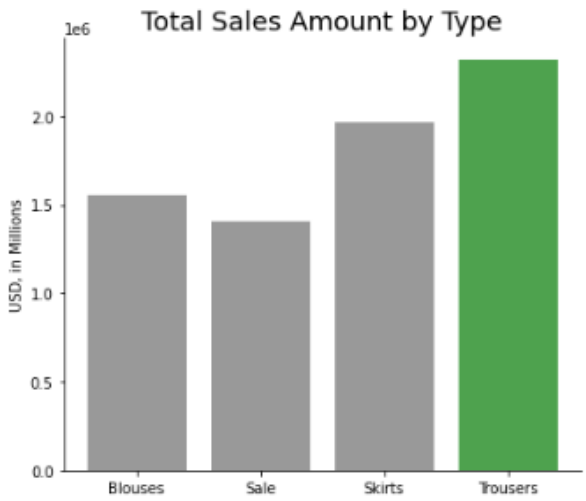


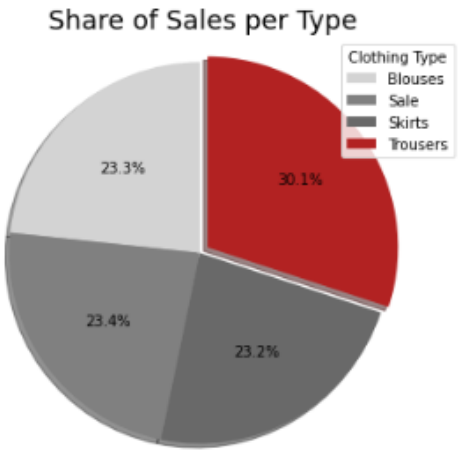

```
### Number of types of clothing sold
csctc = csdf.Type.value_counts()
### Monetary amount sold per type of clothing
csctr = csdf[['Type', 'Price']]
cscts = csctr.groupby('Type').sum()
cscta = csctr.groupby('Type').mean()

csct = cscts
csct['Total'] = csctc
csct['Average'] = cscta['Price']
csct = csct.rename(columns={'Price' : 'Value'})

fig, ax = plt.subplots(figsize = [14,6])

ax.bar(csct.index, csct.Average, color = ['gray', 'gray', 'maroon', 'gray'], alpha = .8)
ax.set_title('Average Cost of Clothing Type', fontsize=18)
ax.spines[['right', 'top']].set_visible(False)

plt.show()
```

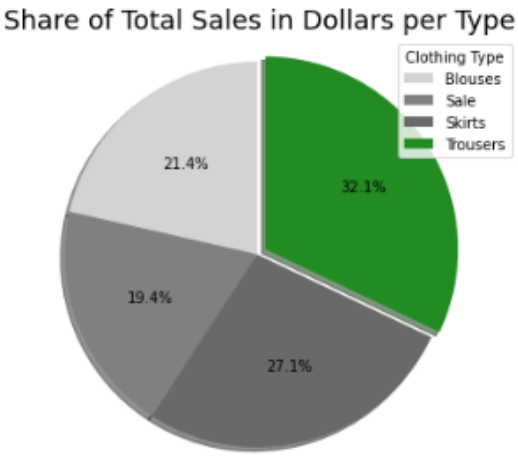



```
fig, axs = plt.subplots(nrows=2, ncols = 2, figsize=[14, 12])

axs[0,0].bar(csct.index, csct.Total,
                color=['gray', 'gray', 'gray', 'maroon'], alpha=.8)
axs[0,0].set_title('Items Sold by Type', fontsize=18)
axs[0,0].spines[['right', 'top']].set_visible(False)

axs[0,1].bar(csct.index, csct.Value,
                color=['gray', 'gray', 'gray', 'forestgreen'], alpha=.8)
axs[0,1].set_title('Total Sales Amount by Type', fontsize=18)
axs[0,1].set_ylabel('USD, in Millions')
axs[0,1].spines[['right', 'top']].set_visible(False)

axs[1,0].pie(csct.Total, explode=(0, 0, 0, 0.05), autopct='%1.1f%%',
            shadow=True, startangle=90,
            colors=['lightgray', 'gray', 'dimgray', 'firebrick'])
axs[1,0].axis('equal')
axs[1,0].set_title("Share of Sales per Type", fontsize=18)
axs[1,0].legend(csct.index, title="Clothing Type", loc="upper right")

axs[1,1].pie(csct.Value, explode=(0, 0, 0, 0.05), autopct='%1.1f%%',
            shadow=True, startangle=90,
            colors=['lightgray', 'gray', 'dimgray', 'forestgreen'])
axs[1,1].axis('equal')
axs[1,1].set_title("Share of Total Sales in Dollars per Type", fontsize=18)
axs[1,1].legend(csct.index, title="Clothing Type", loc="upper right")

plt.show()
```







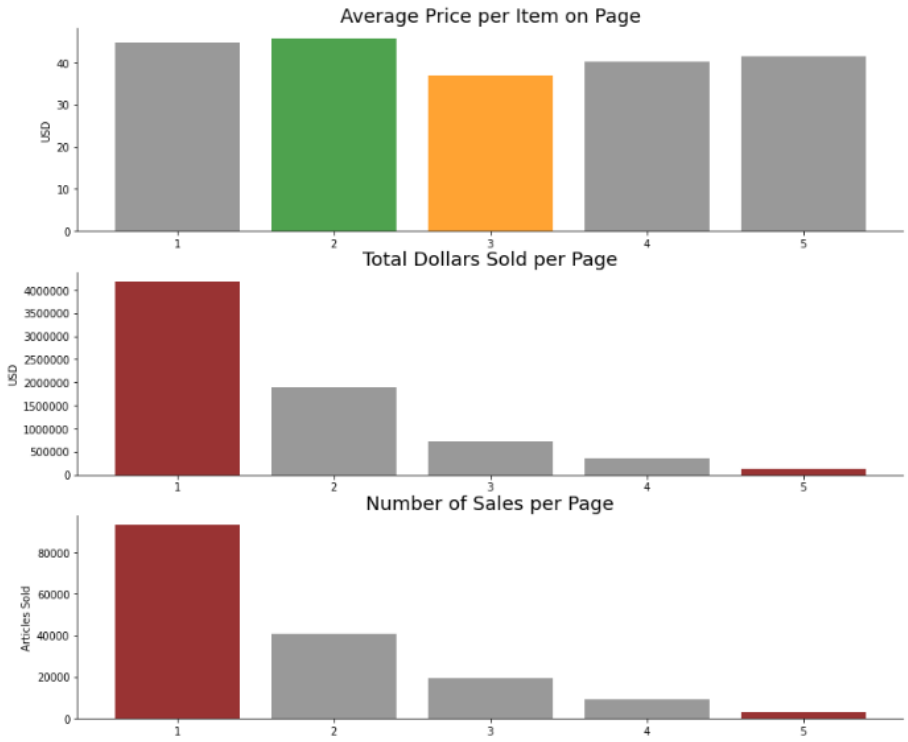



```
csdf.corr()
```

	Day	Price	Page
Day	1.000000	-0.002818	0.011125
Price	-0.002818	1.000000	-0.150455
Page	0.011125	-0.150455	1.000000

```
cspp = csdf[['Price', 'Page']]

ppavg = cspp.groupby('Page').mean()
pptot = cspp.groupby('Page').sum()
ppcnt = cspp.Page.value_counts()

ppdf = ppavg
ppdf['Total'] = pptot.Price
ppdf['Count'] = ppcnt
```

```
ppdf = ppdf.rename(columns={'Price':'Average'})

fig, axs = plt.subplots(nrows=3, ncols = 1, figsize=[14, 12])

axs[0].bar(ppdf.index, ppdf.Average,
                color=['gray', 'forestgreen', 'darkorange', 'gray', 'gray'], alpha=.8)
axs[0].set_title('Average Price per Item on Page', fontsize=18)
axs[0].set_ylabel('USD')

axs[1].bar(ppdf.index, ppdf.Total,
                color=['maroon', 'gray', 'gray', 'gray', 'maroon'], alpha=.8)
axs[1].set_title('Total Dollars Sold per Page', fontsize=18)
axs[1].set_ylabel('USD')
axs[1].ticklabel_format(useOffset=False, style='plain')

axs[2].bar(ppdf.index, ppdf.Count,
                color=['maroon', 'gray', 'gray', 'gray', 'maroon'], alpha=.8)
axs[2].set_title('Number of Sales per Page', fontsize=18)
axs[2].set_ylabel('Articles Sold')

for ax in axs:
    ax.yaxis.grid(False)
    ax.spines[['right', 'top']].set_visible(False)

plt.show()
```



```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=[14,7])
ax[0].pie(ppdf.Count, explode=(0.1, 0, 0, 0, 0), autopct='%1.1f%%',
          shadow=True, startangle=90,
          colors=['firebrick', 'lightgray', 'gray', 'dimgray', 'slateblue'])
ax[0].axis('equal')
ax[0].set_title("Share of Number of Sales per Page", fontsize=18)
ax[0].legend(ppdf.index, title="Page", loc="upper right")

ax[1].pie(ppdf.Total, explode=(0.1, 0, 0, 0, 0), autopct='%1.1f%%',
          shadow=True, startangle=90,
          colors=['firebrick', 'lightgray', 'gray', 'dimgray', 'slateblue'])
ax[1].axis('equal')
ax[1].set_title("Share of Total Sales Dollars per Page", fontsize=18)
ax[1].legend(ppdf.index, title="Page", loc="upper right")

plt.show()
```