dataxlab-internship-task5

November 7, 2025

[1]:
```
# 0. Imports & settings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
```

[2]:
```
# plotting settings
%matplotlib inline
sns.set(style="whitegrid")
pd.set_option('display.max_columns', 200)
```

[3]:
```
import pandas as pd
file_path = r"D:\intern\Superstore.csv"

df = pd.read_csv(file_path, encoding='latin1')
print(df.head())
```

```
   Row ID        Order ID  Order Date   Ship Date       Ship Mode Customer ID  \
0       1  CA-2016-152156   11/8/2016  11/11/2016    Second Class    CG-12520
1       2  CA-2016-152156   11/8/2016  11/11/2016    Second Class    CG-12520
2       3  CA-2016-138688   6/12/2016   6/16/2016    Second Class    DV-13045
3       4  US-2015-108966  10/11/2015  10/18/2015  Standard Class    SO-20335
4       5  US-2015-108966  10/11/2015  10/18/2015  Standard Class    SO-20335

     Customer Name    Segment        Country             City       State  \
0      Claire Gute   Consumer  United States        Henderson    Kentucky
1      Claire Gute   Consumer  United States        Henderson    Kentucky
2  Darrin Van Huff  Corporate  United States      Los Angeles  California
3   Sean O'Donnell   Consumer  United States  Fort Lauderdale     Florida
4   Sean O'Donnell   Consumer  United States  Fort Lauderdale     Florida

   Postal Code Region       Product ID         Category Sub-Category  \
0        42420  South  FUR-BO-10001798        Furniture    Bookcases
1        42420  South  FUR-CH-10000454        Furniture       Chairs
2        90036   West  OFF-LA-10000240  Office Supplies       Labels
```

```
3        33311  South  FUR-TA-10000577        Furniture       Tables
4        33311  South  OFF-ST-10000760  Office Supplies      Storage

                                        Product Name     Sales  Quantity  \
0                    Bush Somerset Collection Bookcase  261.9600         2
1  Hon Deluxe Fabric Upholstered Stacking Chairs,…  731.9400         3
2  Self-Adhesive Address Labels for Typewriters b…   14.6200         2
3      Bretford CR4500 Series Slim Rectangular Table  957.5775         5
4                       Eldon Fold 'N Roll Cart System   22.3680         2

   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164
```

[4]:
```
# 2. Quick overview
df.info()
df.head()
df.describe(include='all').T
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   object
 3   Ship Date      9994 non-null   object
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
```

```
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

[4]:

	count	unique	top	freq	mean	\
Row ID	9994.0	NaN	NaN	NaN	4997.5	
Order ID	9994	5009	CA-2017-100111	14	NaN	
Order Date	9994	1237	9/5/2016	38	NaN	
Ship Date	9994	1334	12/16/2015	35	NaN	
Ship Mode	9994	4	Standard Class	5968	NaN	
Customer ID	9994	793	WB-21850	37	NaN	
Customer Name	9994	793	William Brown	37	NaN	
Segment	9994	3	Consumer	5191	NaN	
Country	9994	1	United States	9994	NaN	
City	9994	531	New York City	915	NaN	
State	9994	49	California	2001	NaN	
Postal Code	9994.0	NaN	NaN	NaN	55190.379428	
Region	9994	4	West	3203	NaN	
Product ID	9994	1862	OFF-PA-10001970	19	NaN	
Category	9994	3	Office Supplies	6026	NaN	
Sub-Category	9994	17	Binders	1523	NaN	
Product Name	9994	1850	Staple envelope	48	NaN	
Sales	9994.0	NaN	NaN	NaN	229.858001	
Quantity	9994.0	NaN	NaN	NaN	3.789574	
Discount	9994.0	NaN	NaN	NaN	0.156203	
Profit	9994.0	NaN	NaN	NaN	28.656896	

	std	min	25%	50%	75%	max
Row ID	2885.163629	1.0	2499.25	4997.5	7495.75	9994.0
Order ID	NaN	NaN	NaN	NaN	NaN	NaN
Order Date	NaN	NaN	NaN	NaN	NaN	NaN
Ship Date	NaN	NaN	NaN	NaN	NaN	NaN
Ship Mode	NaN	NaN	NaN	NaN	NaN	NaN
Customer ID	NaN	NaN	NaN	NaN	NaN	NaN
Customer Name	NaN	NaN	NaN	NaN	NaN	NaN
Segment	NaN	NaN	NaN	NaN	NaN	NaN
Country	NaN	NaN	NaN	NaN	NaN	NaN
City	NaN	NaN	NaN	NaN	NaN	NaN
State	NaN	NaN	NaN	NaN	NaN	NaN
Postal Code	32063.69335	1040.0	23223.0	56430.5	90008.0	99301.0
Region	NaN	NaN	NaN	NaN	NaN	NaN
Product ID	NaN	NaN	NaN	NaN	NaN	NaN
Category	NaN	NaN	NaN	NaN	NaN	NaN
Sub-Category	NaN	NaN	NaN	NaN	NaN	NaN
Product Name	NaN	NaN	NaN	NaN	NaN	NaN
Sales	623.245101	0.444	17.28	54.49	209.94	22638.48
Quantity	2.22511	1.0	2.0	3.0	5.0	14.0
Discount	0.206452	0.0	0.0	0.2	0.2	0.8

```
Profit          234.260108 -6599.978  1.72875   8.6665   29.364  8399.976
```

[5]:
```
# 3. Missing values & duplicates
missing = df.isnull().sum().sort_values(ascending=False)
missing[missing>0]
print("Duplicates:", df.duplicated().sum())
```

```
Duplicates: 0
```

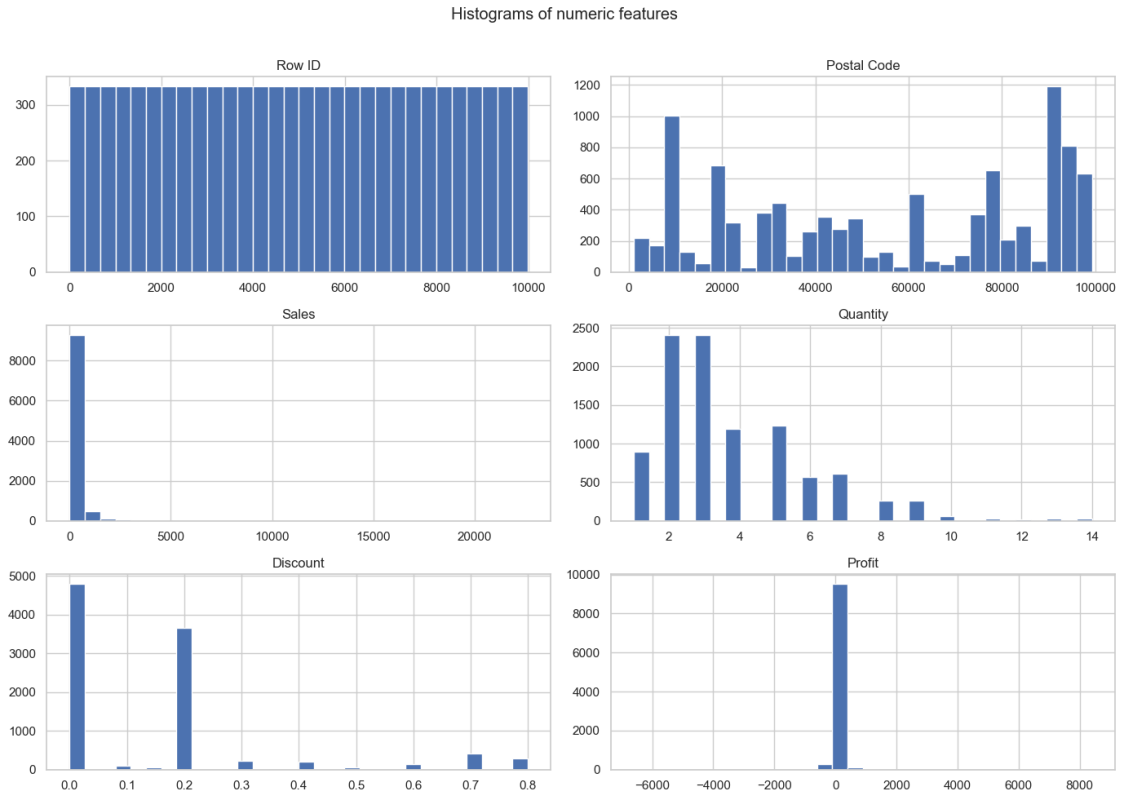
[6]:
```
# 4. Value counts for categorical fields (pick a few important ones)
for col in df.select_dtypes(include='object').columns[:8]:
    print("----", col, "----")
    print(df[col].value_counts(dropna=False).head(10))
    print()
```

```
---- Order ID ----
Order ID
CA-2017-100111    14
CA-2017-157987    12
CA-2016-165330    11
US-2016-108504    11
US-2015-126977    10
CA-2016-105732    10
CA-2015-131338    10
CA-2015-158421     9
CA-2014-106439     9
US-2015-163433     9
Name: count, dtype: int64

---- Order Date ----
Order Date
9/5/2016      38
9/2/2017      36
11/10/2016    35
12/1/2017     34
12/2/2017     34
12/9/2017     33
11/12/2017    30
12/8/2017     30
9/9/2017      29
9/4/2017      28
Name: count, dtype: int64

---- Ship Date ----
Ship Date
12/16/2015    35
9/26/2017     34
11/21/2017    32
```

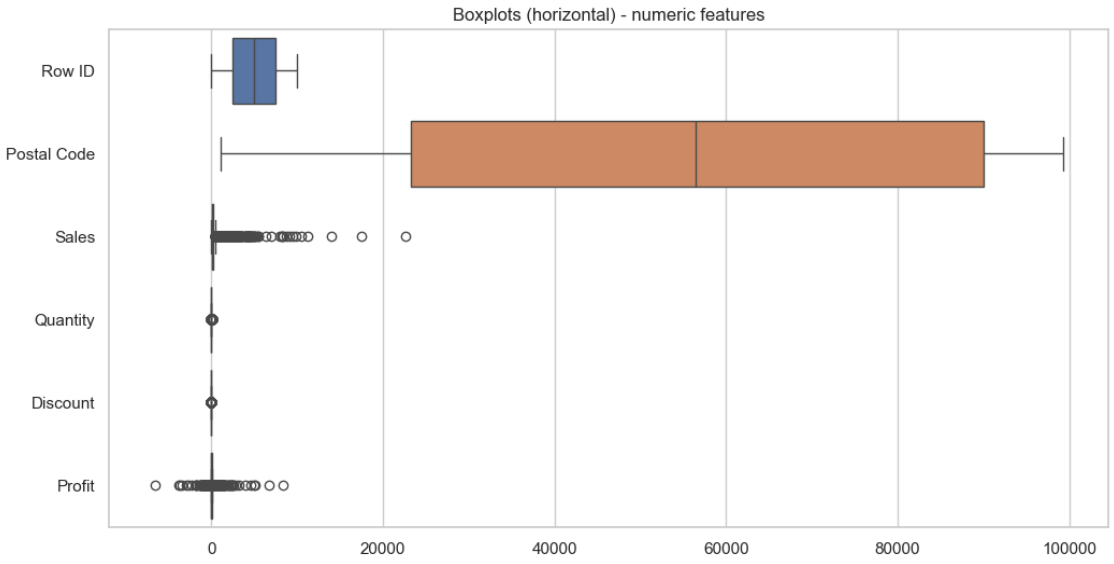
```
12/6/2017     32
9/6/2017      30
12/12/2017    30
9/15/2017     30
9/13/2014     27
9/8/2017      27
9/26/2015     26
Name: count, dtype: int64

---- Ship Mode ----
Ship Mode
Standard Class    5968
Second Class      1945
First Class       1538
Same Day           543
Name: count, dtype: int64

---- Customer ID ----
Customer ID
WB-21850    37
MA-17560    34
JL-15835    34
PP-18955    34
CK-12205    32
JD-15895    32
EH-13765    32
SV-20365    32
ZC-21910    31
EP-13915    31
Name: count, dtype: int64

---- Customer Name ----
Customer Name
William Brown          37
Matt Abelman           34
John Lee               34
Paul Prost             34
Chloris Kastensmidt    32
Jonathan Doherty       32
Edward Hooks           32
Seth Vernon            32
Zuschuss Carroll       31
Emily Phan             31
Name: count, dtype: int64

---- Segment ----
Segment
Consumer       5191
```

```
Corporate      3020
Home Office    1783
Name: count, dtype: int64

---- Country ----
Country
United States    9994
Name: count, dtype: int64
```

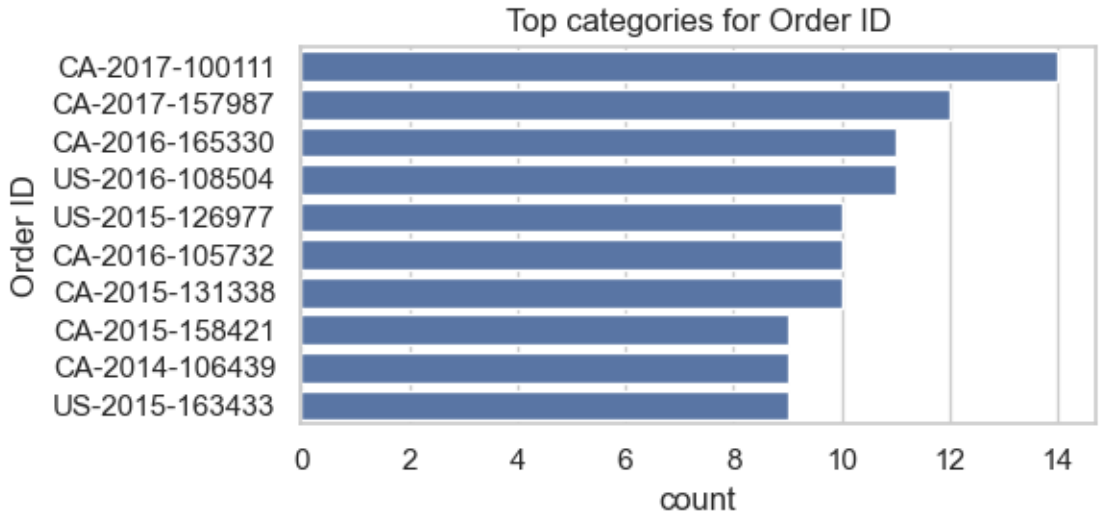
[7]:
```
# 5. Univariate analysis - numeric features: histograms + boxplots
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
# histograms
df[num_cols].hist(figsize=(14, 10), bins=30)
plt.suptitle("Histograms of numeric features")
plt.tight_layout(rect=[0, 0, 1, 0.97])
```

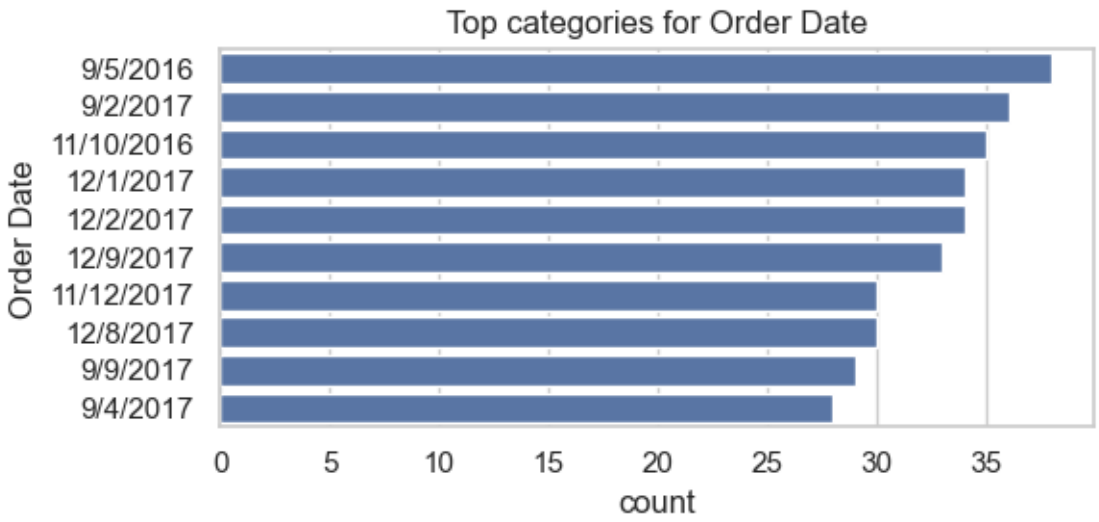


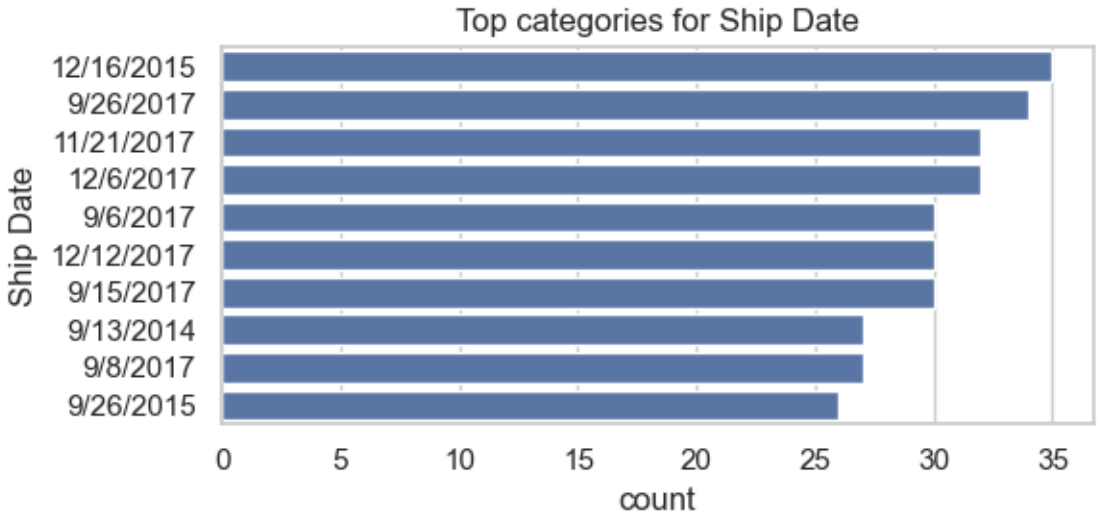
[8]:
```
# boxplots (one per numeric col)
plt.figure(figsize=(12,6))
sns.boxplot(data=df[num_cols], orient='h')
plt.title("Boxplots (horizontal) - numeric features")
plt.show()
```

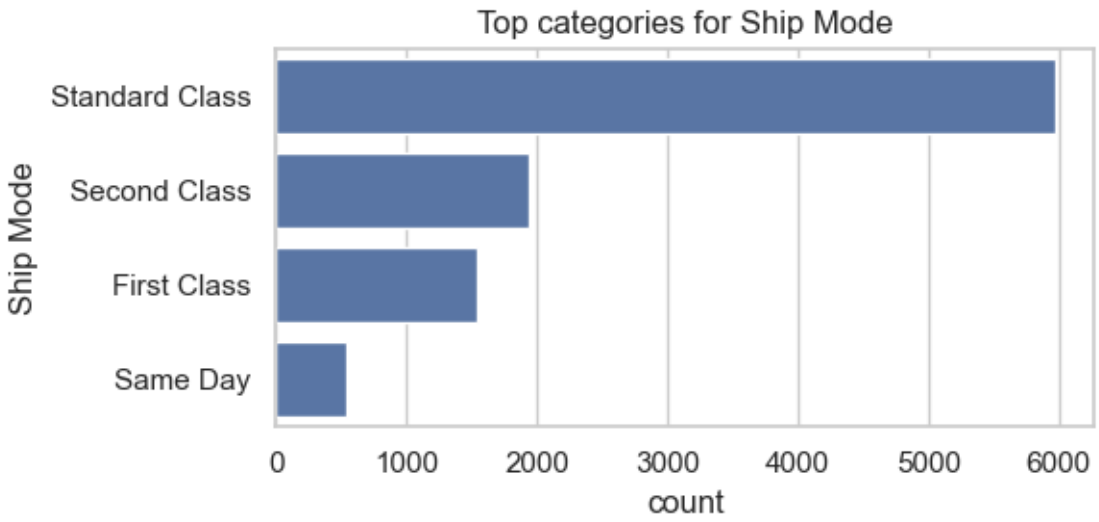


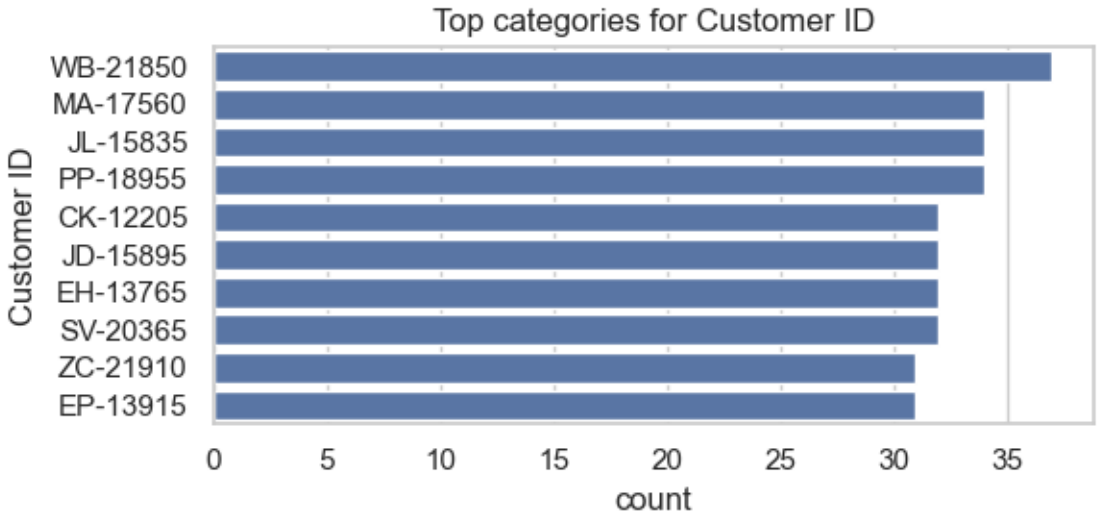
[9]:
```
# 6. Categorical distributions - bar plots for top categories
cat_cols = df.select_dtypes(include='object').columns.tolist()
for c in cat_cols[:6]:
    plt.figure(figsize=(6,3))
    sns.countplot(y=c, data=df, order=df[c].value_counts().index[:10])
    plt.title(f"Top categories for {c}")
    plt.tight_layout()
    plt.show()
```

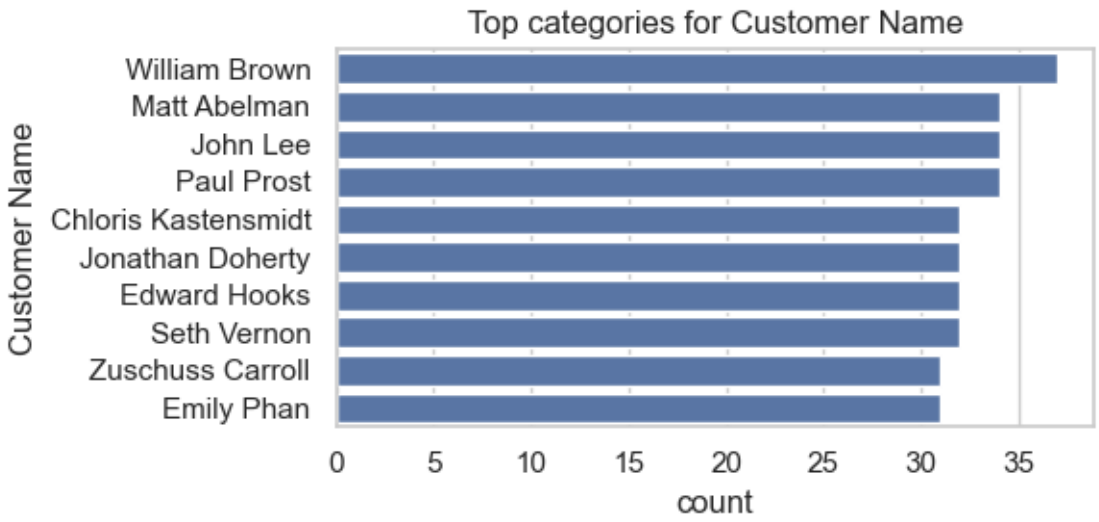



Top categories for Order Date
Order Date
9/5/2016
9/2/2017
11/10/2016
12/1/2017
12/2/2017
12/9/2017
11/12/2017
12/8/2017
9/9/2017
9/4/2017
0
5
10
15
20
25
30
35
count

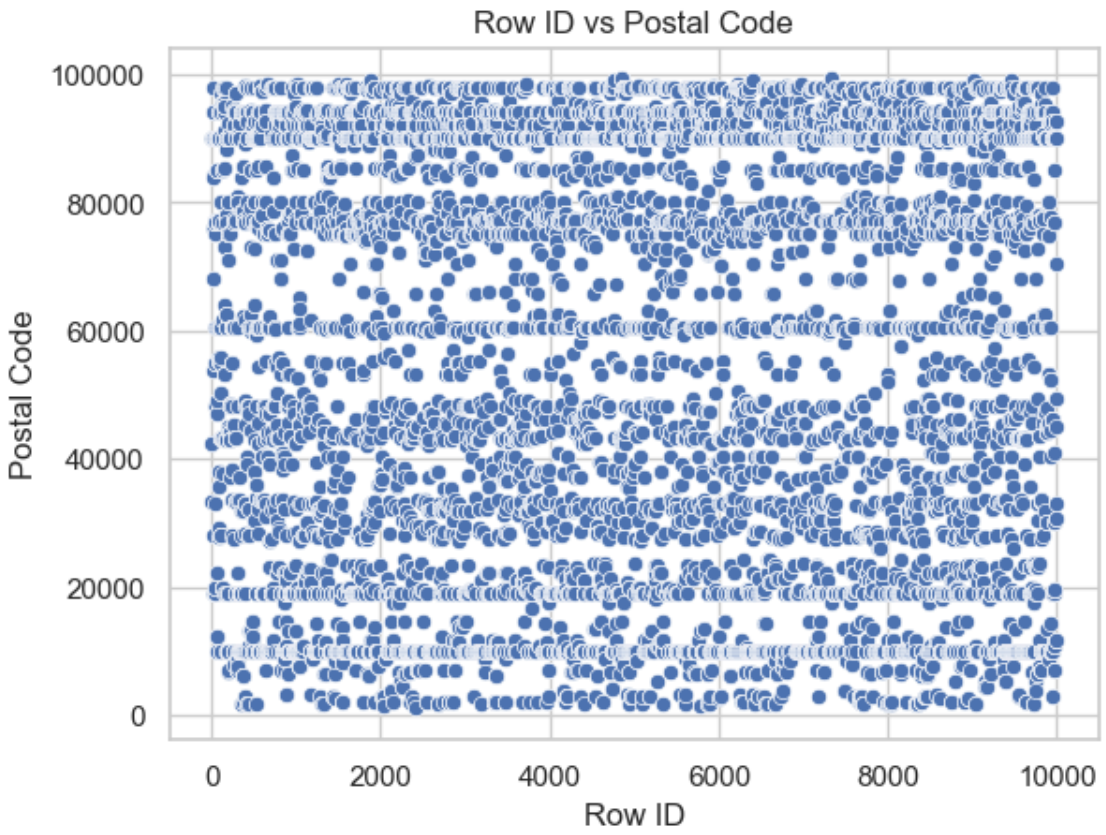

Top categories for Ship Date
Ship Date
12/16/2015
9/26/2017
11/21/2017
12/6/2017
9/6/2017
12/12/2017
9/15/2017
9/13/2014
9/8/2017
9/26/2015
0
5
10
15
20
25
30
35
count

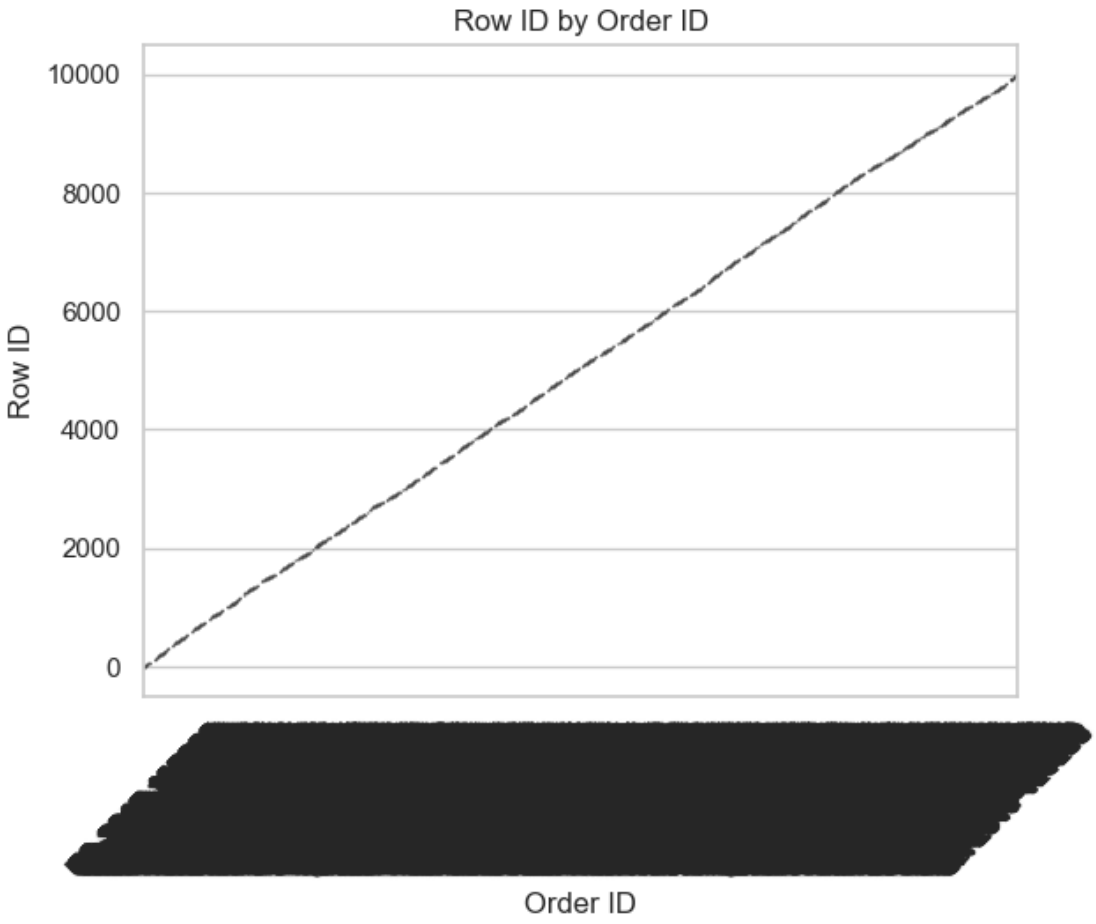

Top categories for Ship Mode
Ship Mode
Standard Class
Second Class
First Class
Same Day
0
1000
2000
3000
4000
5000
6000
count


Top categories for Customer ID
Customer ID
WB-21850
MA-17560
JL-15835
PP-18955
CK-12205
JD-15895
EH-13765
SV-20365
ZC-21910
EP-13915
0
5
10
15
20
25
30
35
count



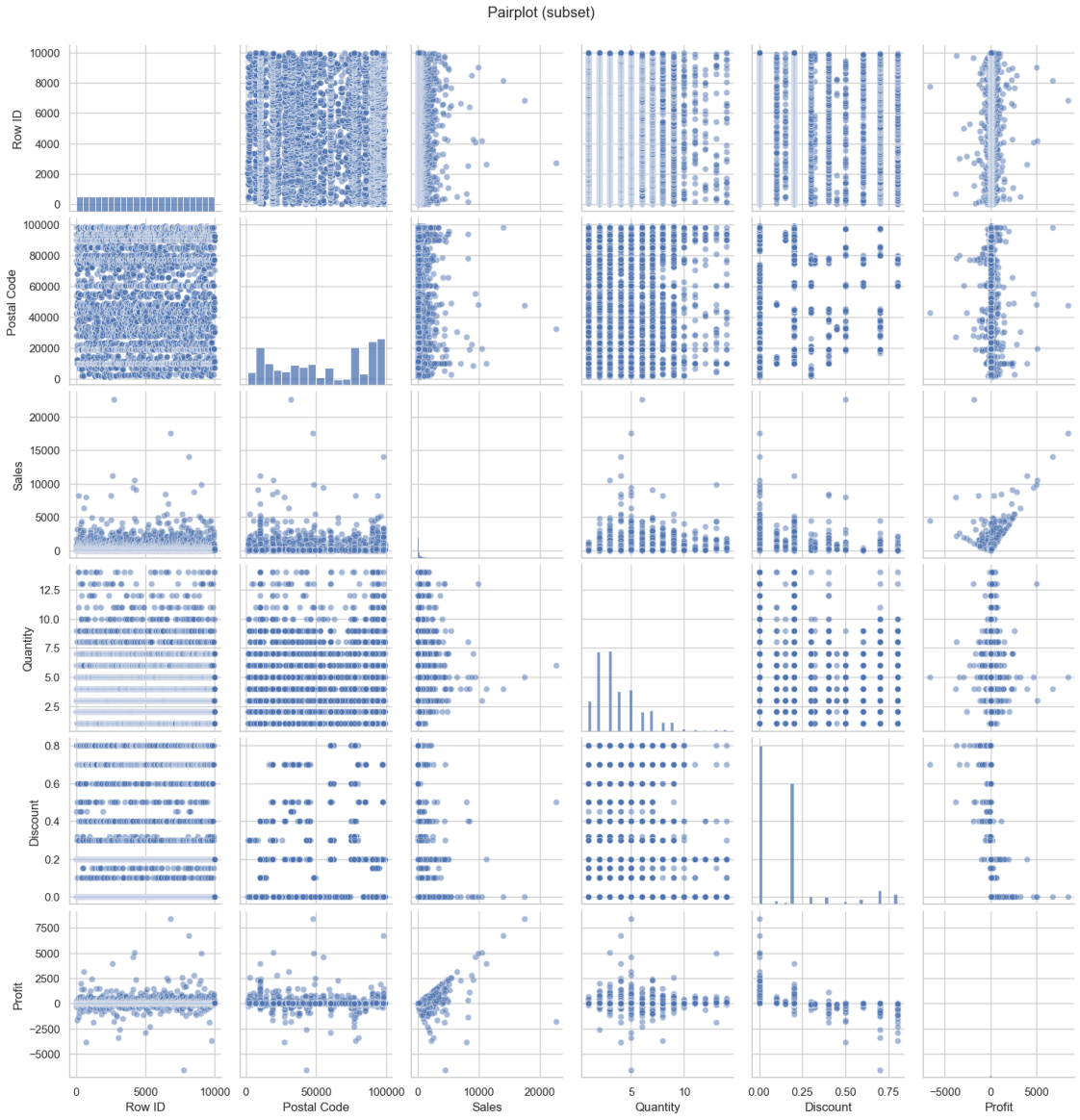
[10]:
```
# 7. Bivariate analysis - scatter / boxplots
#numeric vs numeric scatter
if len(num_cols) >= 2:
    sns.scatterplot(x=num_cols[0], y=num_cols[1], data=df)
    plt.title(f"{num_cols[0]} vs {num_cols[1]}")
    plt.show()

# numeric vs categorical: use boxplot
if cat_cols:
    sns.boxplot(x=cat_cols[0], y=num_cols[0], data=df)
    plt.title(f"{num_cols[0]} by {cat_cols[0]}")
    plt.xticks(rotation=45)
    plt.show()
```


Row ID vs Postal Code
Postal Code
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000
10000
Row ID



[11]:
```
# 8. Correlation matrix + heatmap
corr = df[num_cols].corr()
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title("Correlation matrix")
plt.show()
```

[12]:
```
# 9. Pairplot (careful with many columns; select subset)
subset = num_cols[:6]
sns.pairplot(df[subset].dropna(), diag_kind='hist', plot_kws={'alpha':0.5})
plt.suptitle("Pairplot (subset)", y=1.02)
plt.show()
```



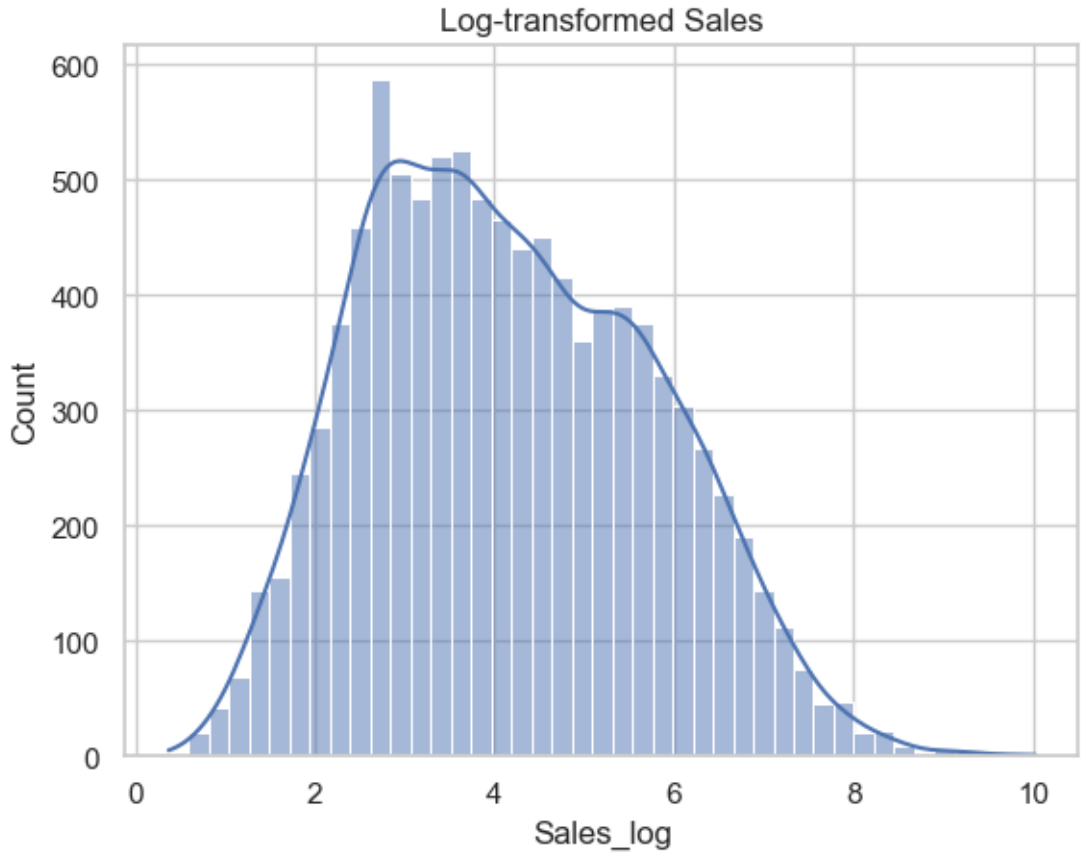
[13]:
```
# 10. Detect multicollinearity using VIF (for numeric features)
X = df[num_cols].dropna()  # drop rows with NA in numeric columns for VIF calc
X_const = sm.add_constant(X)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X_const.values, i+1) for i in␣
 ↪range(len(X.columns))]
vif_data.sort_values("VIF", ascending=False)
```

[13]:

	feature	VIF
5	Profit	1.375543
2	Sales	1.358932

```
4     Discount  1.062648
3     Quantity  1.043633
1  Postal Code  1.004354
0       Row ID  1.000625
```

[14]:
```python
# 11. Skewness - show skew and example transform
skews = df[num_cols].skew().sort_values(key=lambda x: x.abs(), ascending=False)
skews.head(20)
#transform for a skewed feature
skewed_col = skews.index[0]
df[skewed_col + "_log"] = np.log1p(df[skewed_col])
sns.histplot(df[skewed_col + "_log"].dropna(), kde=True)
plt.title(f"Log-transformed {skewed_col}")
plt.show()
```



[15]:
```python
# 12. Outlier detection (IQR method) - count outliers per numeric col
outlier_counts = {}
for c in num_cols:
    Q1 = df[c].quantile(0.25)
    Q3 = df[c].quantile(0.75)
```

```
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR
    outlier_counts[c] = ((df[c] < lower) | (df[c] > upper)).sum()
pd.Series(outlier_counts).sort_values(ascending=False).head(10)
```

[15]:
```
Profit         1881
Sales          1167
Discount        856
Quantity        170
Postal Code       0
Row ID            0
dtype: int64
```

[16]:
```
# 13. Statistical test examples (e.g., t-test / ANOVA) depending on problem
#t-test for numeric across binary categorical variable
if len(cat_cols)>0 and df[cat_cols[0]].nunique() == 2:
    grp = df[df[cat_cols[0]] == df[cat_cols[0]].unique()[0]][num_cols[0]].
 ↪dropna()
    grp2 = df[df[cat_cols[0]] == df[cat_cols[0]].unique()[1]][num_cols[0]].
 ↪dropna()
    tstat, pval = stats.ttest_ind(grp, grp2, equal_var=False)
    print("t-stat:", tstat, "p-value:", pval)
```

[17]:
```
# 14. Save cleaned sample or notebook outputs if needed
df.to_csv("cleaned_dataset_sample.csv", index=False)
```

[18]:
```
# 15. Export notebook to PDF (run from terminal or Jupyter):
!jupyter nbconvert --to pdf "Dataxlab_internship_task5.ipynb"
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
```

```
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
```

```
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
```

```
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook.
To recover
                                    previous default behaviour (outputting to the
current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
```

```
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]
```

```
            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.

[NbConvertApp] WARNING | pattern 'Dataxlab_internship_task5.ipynb' matched no
files
```