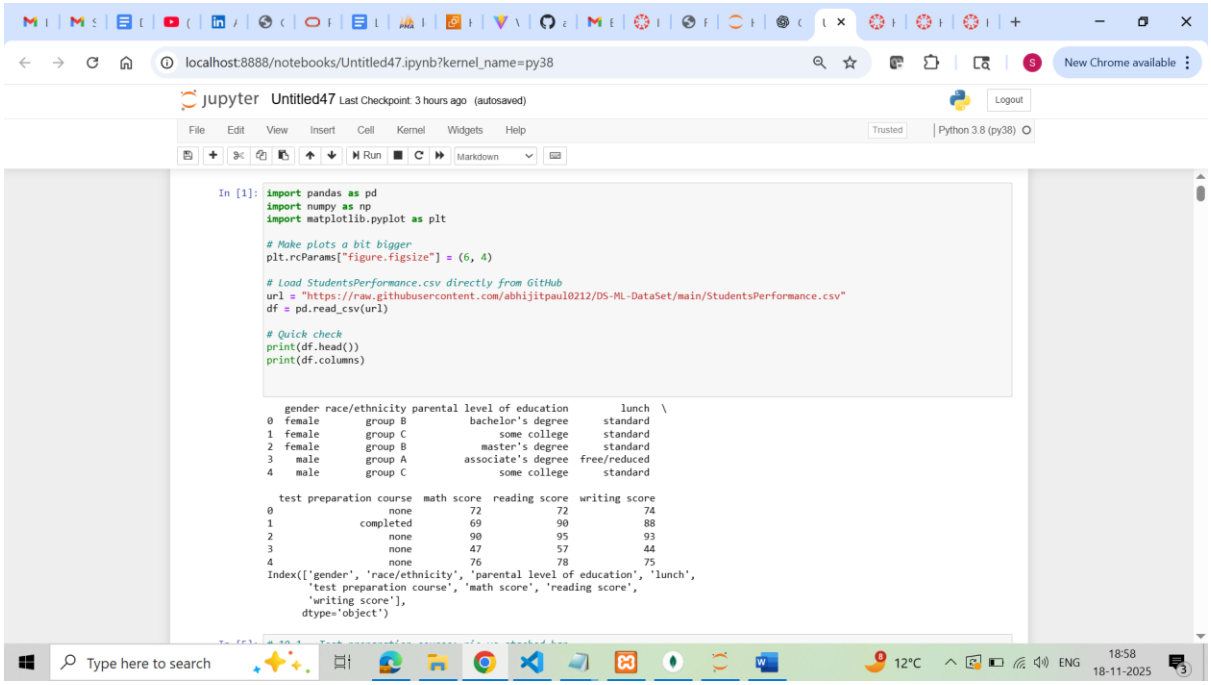
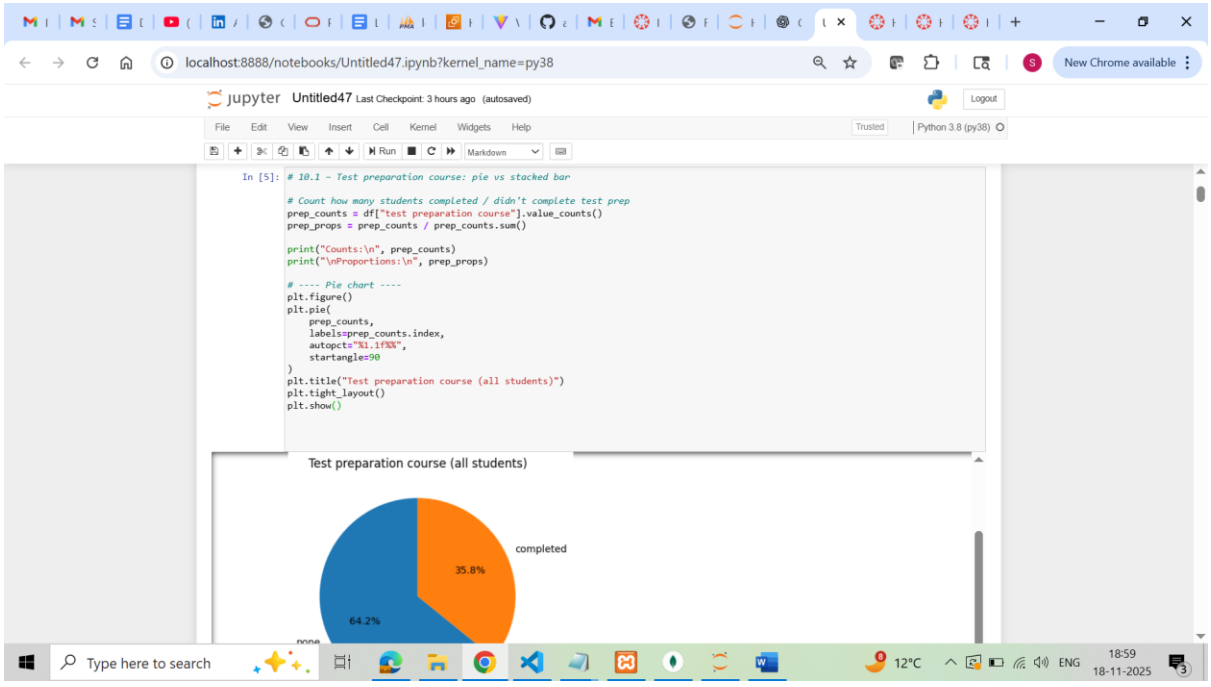
CHAPTER 10


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3.8 (py38)
Run
Markdown
In [1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Make plots a bit bigger
plt.rcParams["figure.figsize"] = (6, 4)
# Load StudentsPerformance.csv directly from GitHub
url = "https://raw.githubusercontent.com/abhijitpaul0212/DS-ML-DataSet/main/StudentsPerformance.csv"
df = pd.read_csv(url)
# Quick check
print(df.head())
print(df.columns)
gender race/ethnicity parental level of education lunch \
0 female group B bachelor's degree standard
1 female group C some college standard
2 female group B master's degree standard
3 male group A associate's degree free/reduced
4 male group C some college standard
test preparation course math score reading score writing score
0 none 72 72 74
1 completed 69 90 88
2 none 90 95 93
3 none 47 57 44
4 none 76 78 75
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
'test preparation course', 'math score', 'reading score',
'writing score'],
dtype='object')
Type here to search
12°C
ENG
18:58
18-11-2025


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3.8 (py38)
Run
Markdown
In [5]:
# 10.1 - Test preparation course: pie vs stacked bar
# Count how many students completed / didn't complete test prep
prep_counts = df["test preparation course"].value_counts()
prep_props = prep_counts / prep_counts.sum()
print("Counts:\n", prep_counts)
print("\nProportions:\n", prep_props)
# ---- Pie chart ----
plt.figure()
plt.pie(
prep_counts,
labels=prep_counts.index,
autopct="%1.1f%%",
startangle=90
)
plt.title("Test preparation course (all students)")
plt.tight_layout()
plt.show()
Test preparation course (all students)
completed
35.8%
64.2%
none
Type here to search
12°C
ENG
18:59
18-11-2025

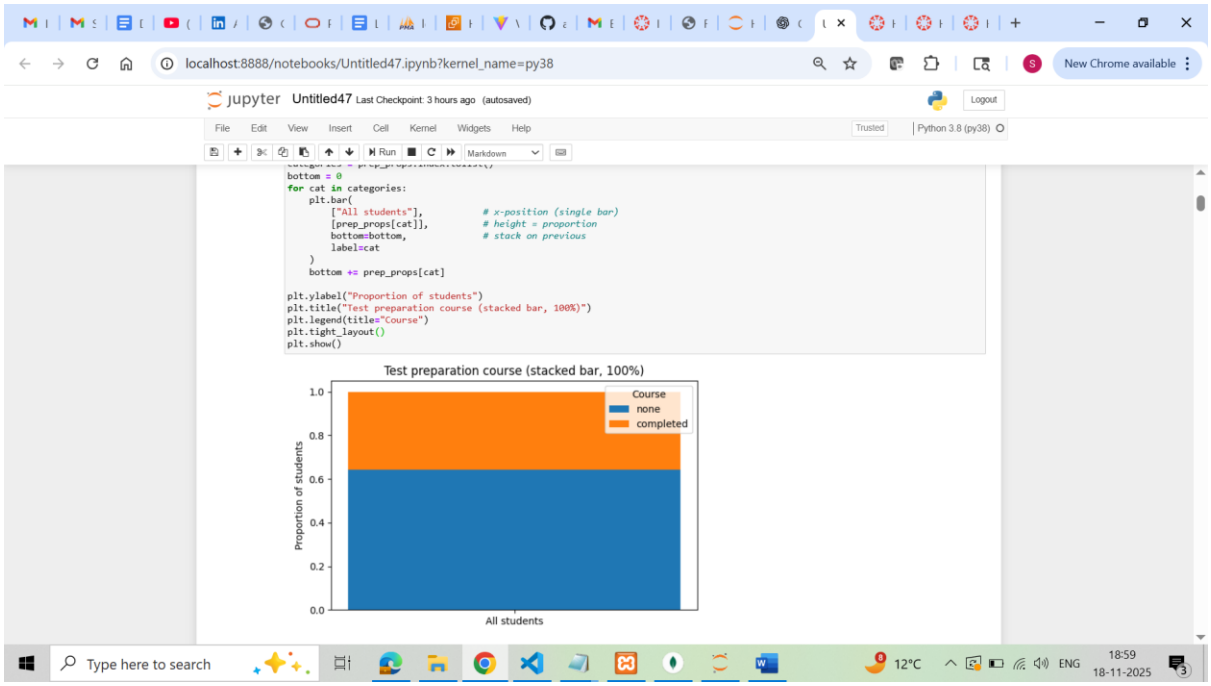
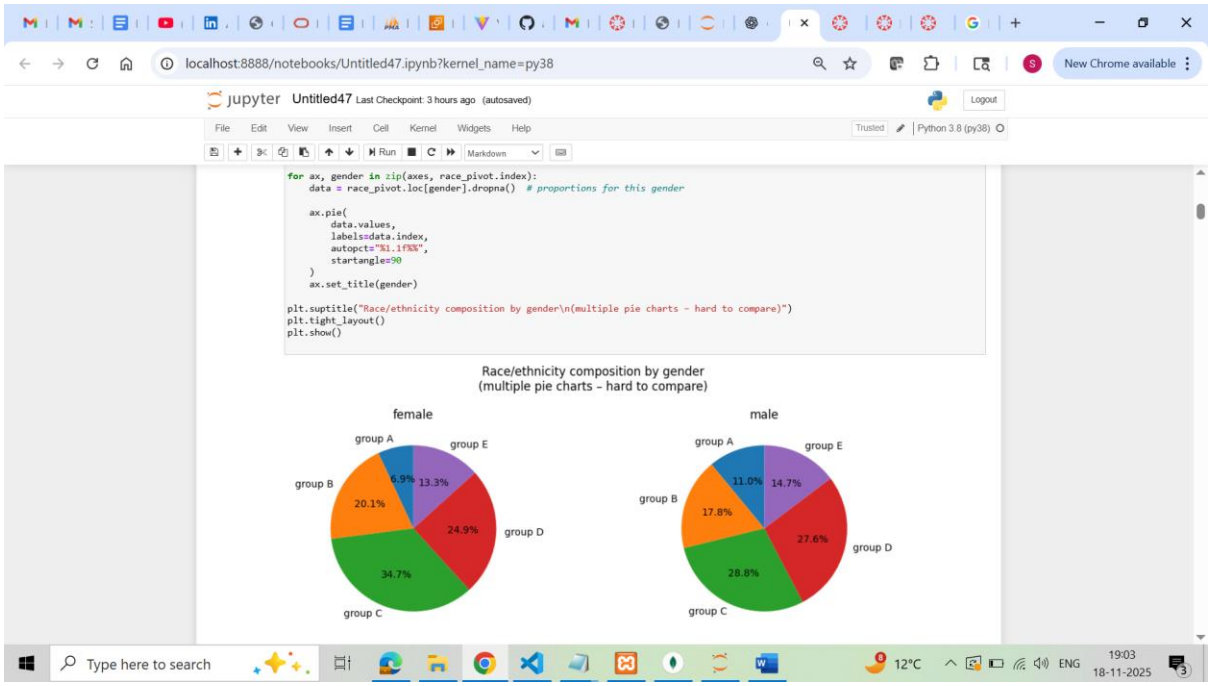


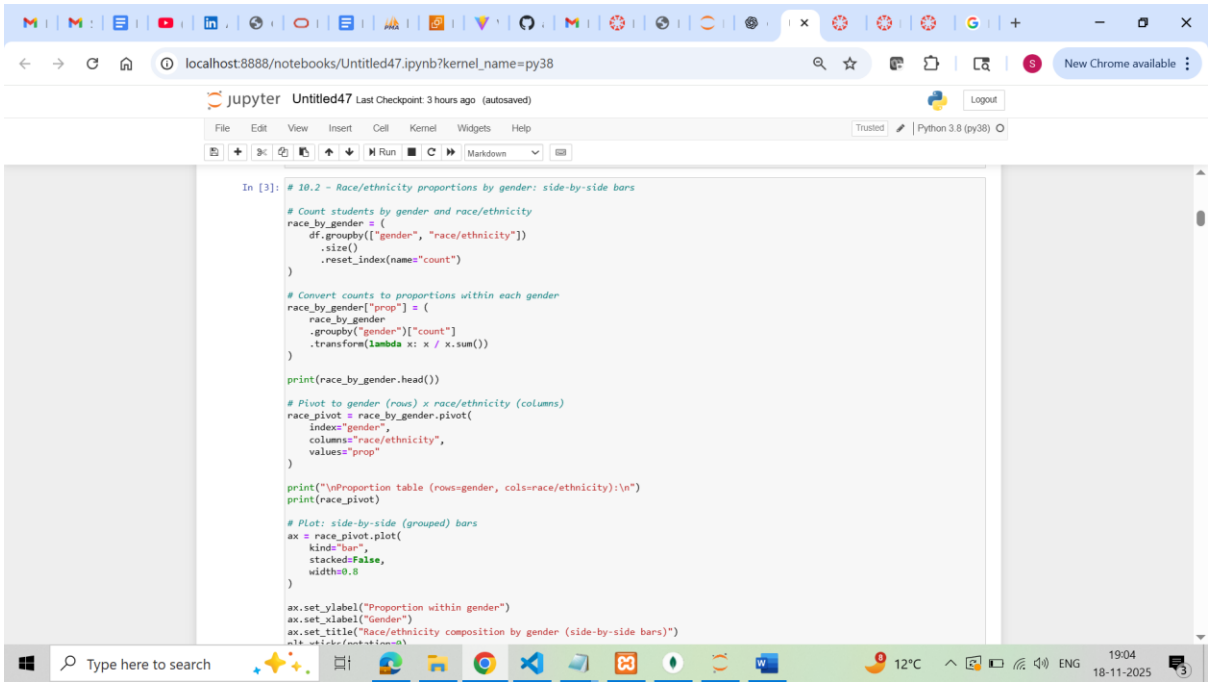
Figure 10.1a – Pie chart of test preparation course

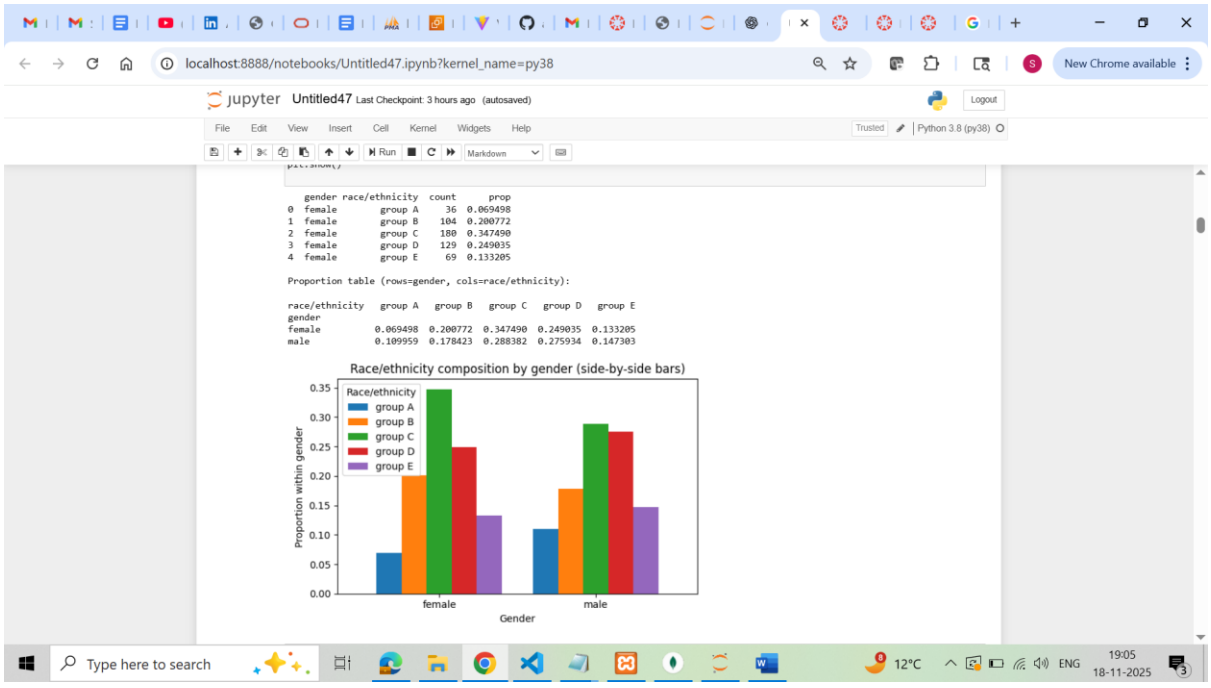
This pie chart shows the overall proportion of students who completed the test-preparation course versus those who did not. Most students did not complete test prep, while a smaller slice completed it. With only two categories and a single set of proportions, the pie chart works reasonably well and emphasizes simple fractions of the whole.

Figure 10.1b – 100% stacked bar of test preparation course

This stacked bar encodes the same information as the pie chart: the bar height is 100% of students and the segments show the share with and without test prep. It reinforces that stacked bars and pies both show “part of a whole,” but for one single group the stacked bar feels less natural than the pie.







10.2 – Multiple pies (bad) vs side-by-side bars (good)

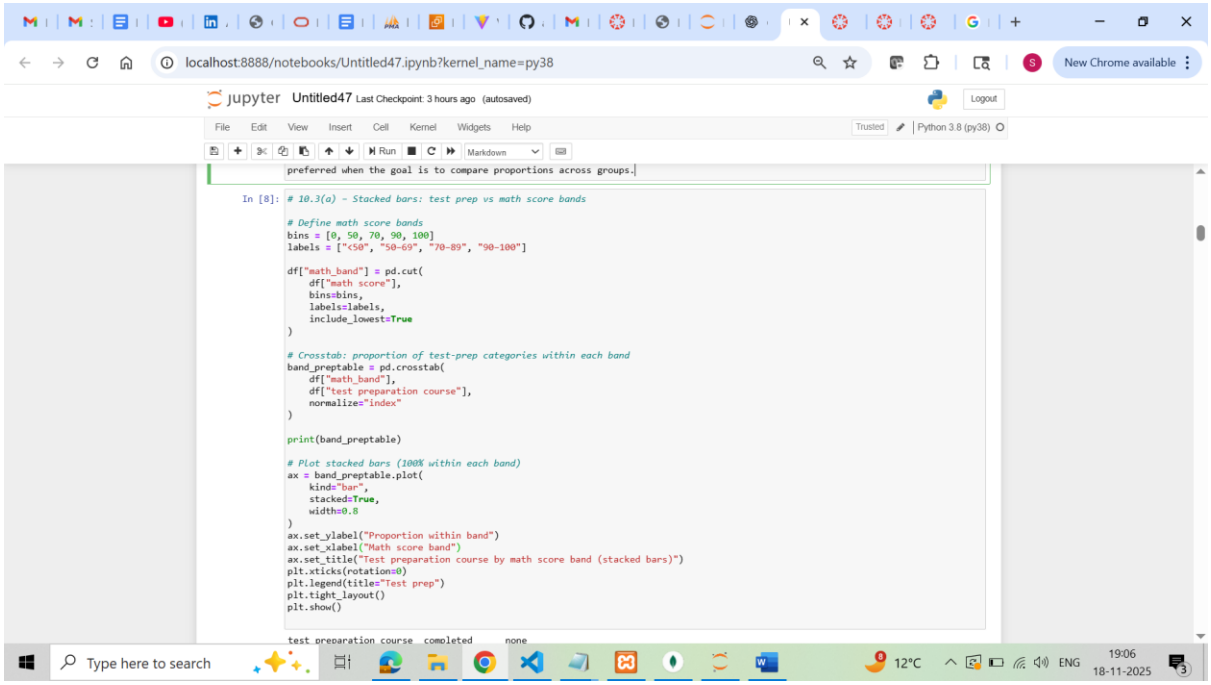
Figure 10.2a – Multiple pie charts by gender (bad design)

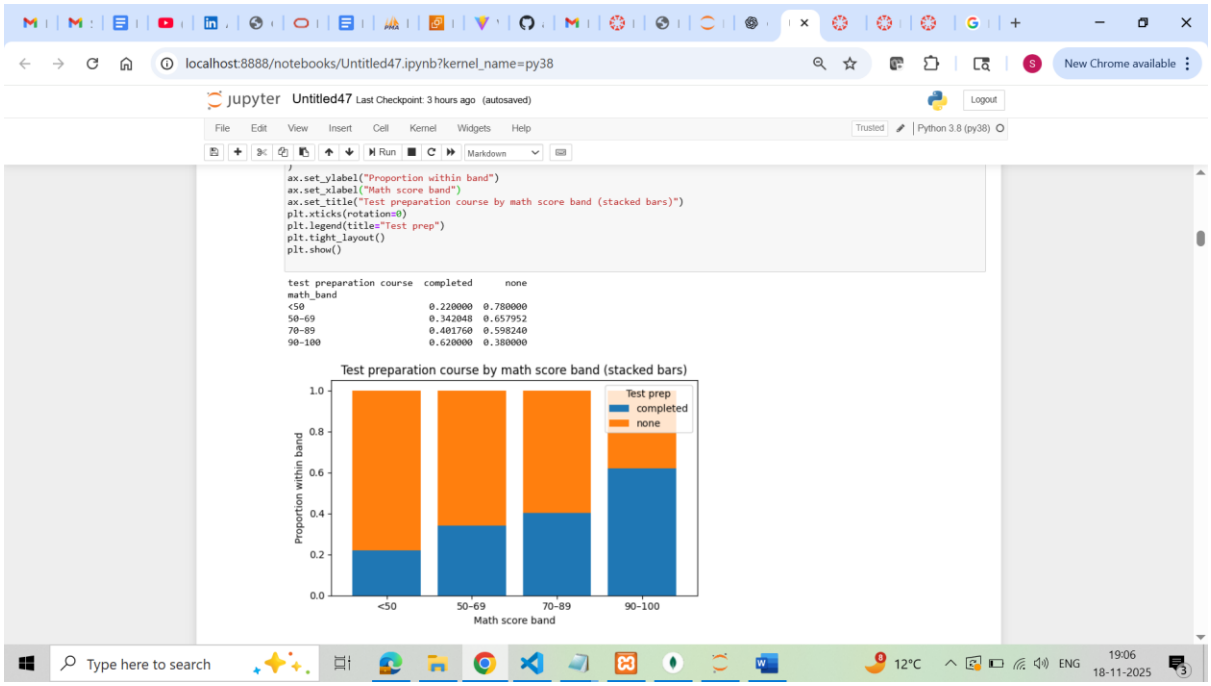
Here each pie shows the race/ethnicity composition within one gender. While each individual pie is readable, it is very hard to compare slices across pies (e.g., “Is group B more common for males or females?”). This matches the lecture example where multiple pies over years are labeled as a poor design for comparisons.

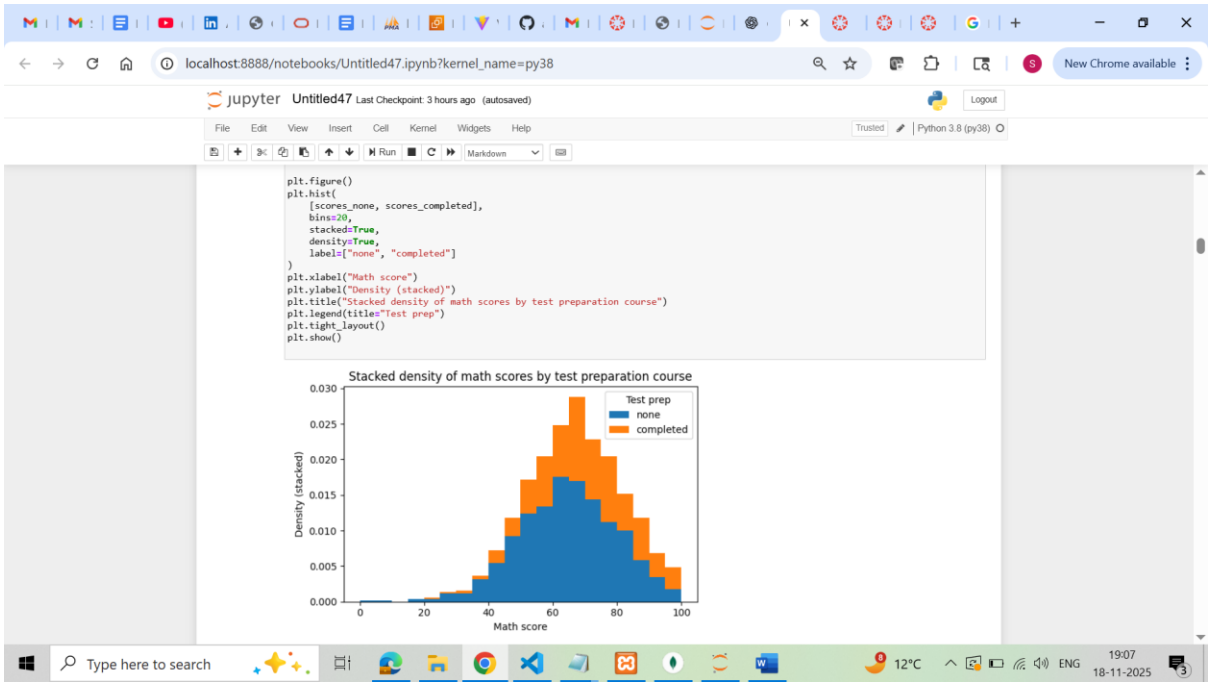
Figure 10.2b – Side-by-side bars of race/ethnicity by gender (good design)

This grouped bar chart shows the same information, but now the race/ethnicity categories are bars on a common scale within each gender. Differences between genders are easy to compare by looking

at bar heights. This illustrates why side-by-side bars are preferred when the goal is to compare proportions across groups.







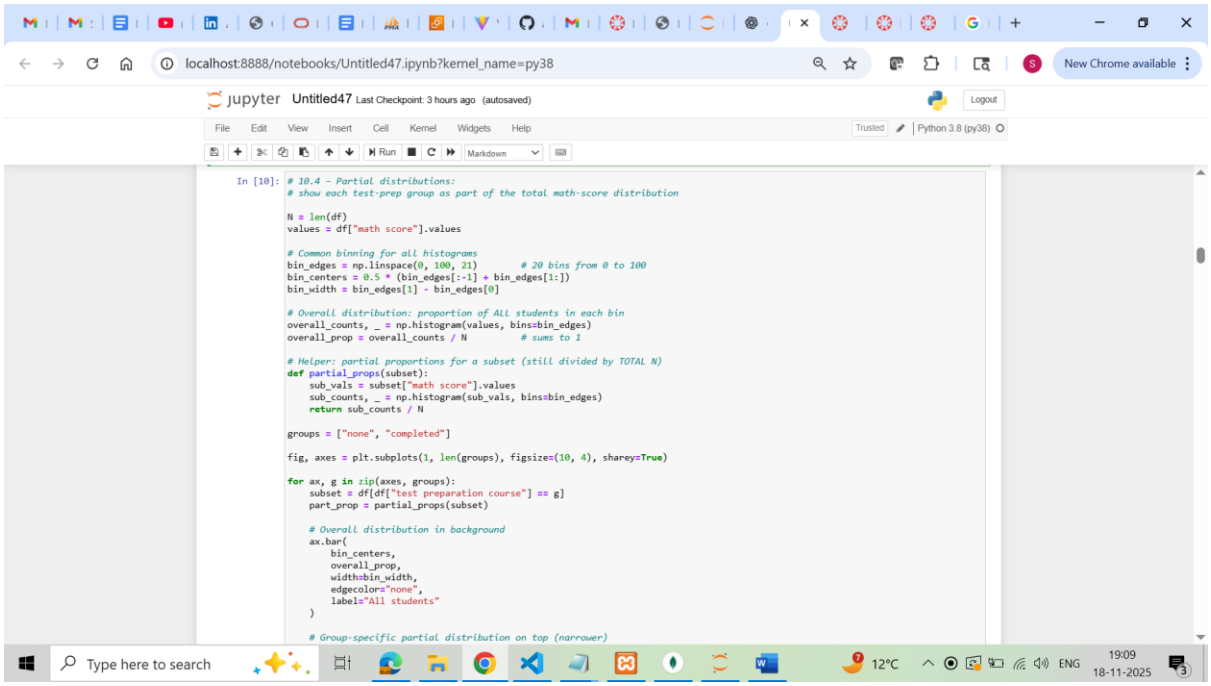
10.3 – Stacked bars and stacked densities

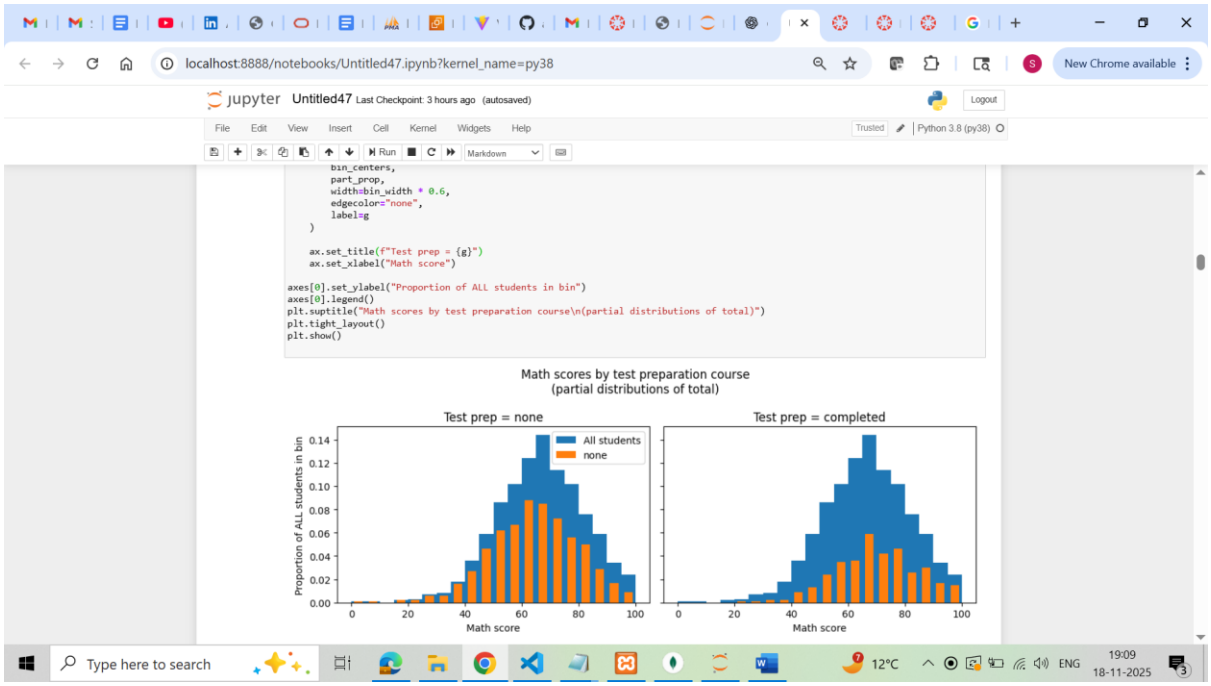
Figure 10.3a – 100% stacked bars of test prep by math score band

Math scores are split into bands (<50, 50–69, 70–89, 90–100), and within each band the bar is stacked into “none” vs “completed” test prep. Each bar sums to 100% of students in that band. Higher score bands have a larger share of students who completed test prep, while lower bands have more students with no prep. This is similar to the textbook’s example of stacked bars showing composition over conditions.

Figure 10.3b – Stacked histogram (stacked density) of math scores

The stacked histogram shows the overall density of math scores, with contributions from the “none” and “completed” test-prep groups stacked on top of each other. At low scores the density is dominated by students with no test prep, whereas at high scores it is dominated by students who completed test prep. This is the stacked-density idea from the chapter, showing how category composition changes along a continuous axis.



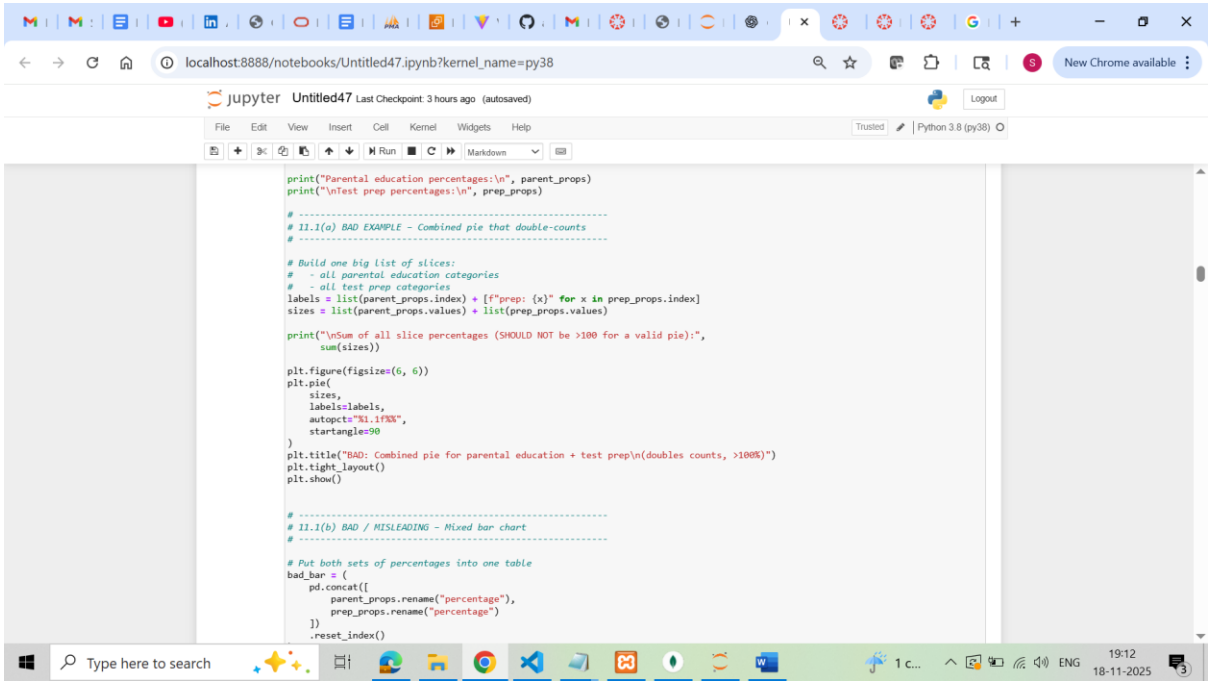


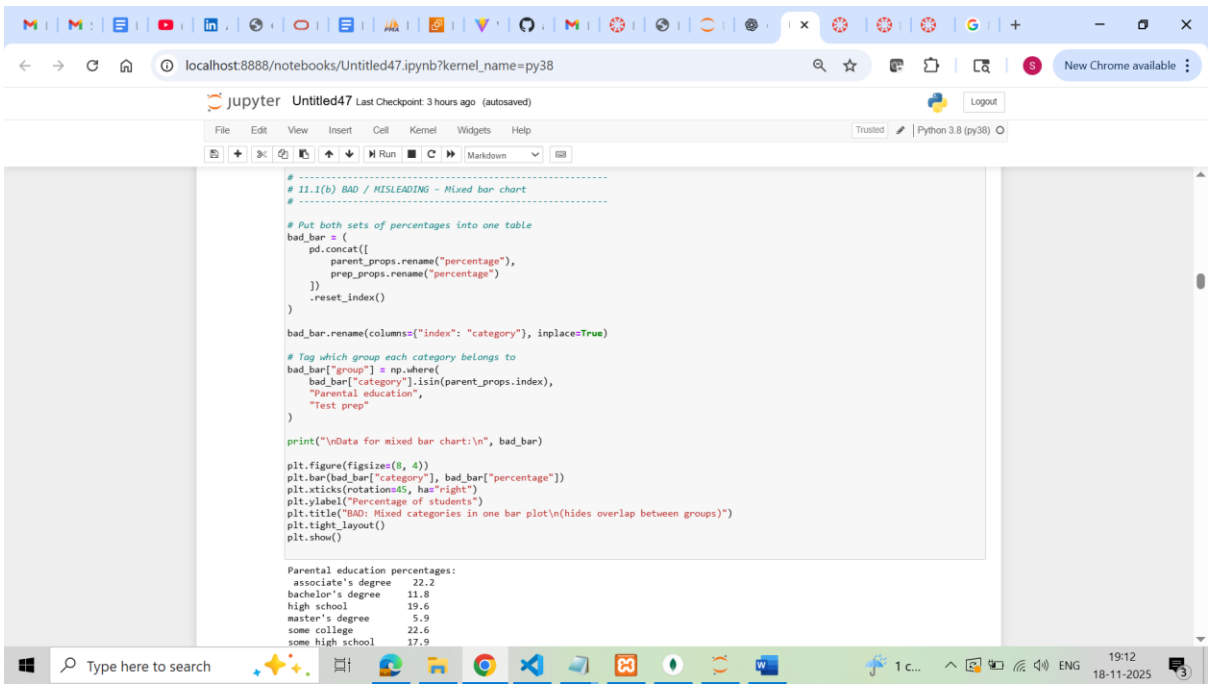
10.4 – Partial distributions as parts of the total

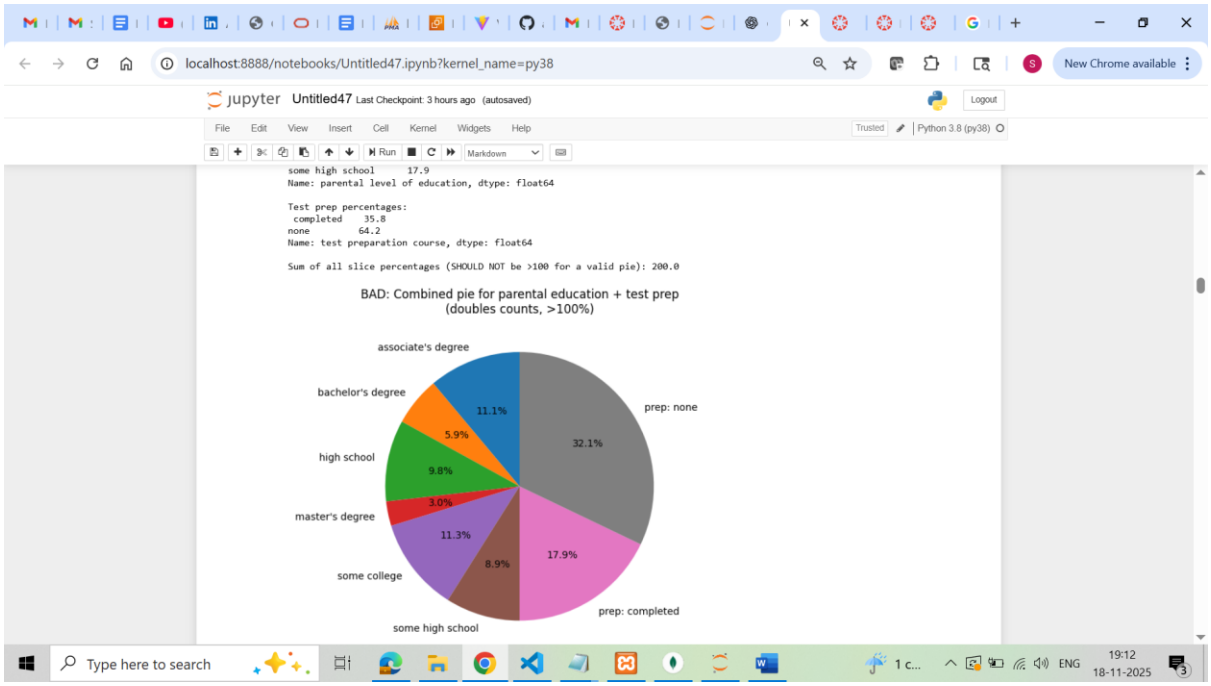
Figure 10.4 – Partial distributions of math scores by test preparation course

These two panels show partial distributions for the two test-prep groups. In each panel, the gray bars represent the overall math-score distribution (all students), and the colored bars show the proportion of all students who are in that score bin and in that specific test-prep group. For the “completed” group, the colored bars are concentrated at higher scores; for the “none” group they are shifted toward lower and mid scores. This follows the chapter’s idea of viewing each group as a part of the total distribution, rather than stacking everything in a single figure.

CHAPTER 11


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Markdown
print("Parental education percentages:\n", parent_props)
print("\nTest prep percentages:\n", prep_props)
# --
# 11.1(a) BAD EXAMPLE – Combined pie that double-counts
# --
# Build one big list of slices:
# - all parental education categories
# - all test prep categories
labels = list(parent_props.index) + [f"prep: {x}" for x in prep_props.index]
sizes = list(parent_props.values) + list(prep_props.values)
print("\nSum of all slice percentages (SHOULD NOT be >100 for a valid pie):",
sum(sizes))
plt.figure(figsize=(6, 6))
plt.pie(
sizes,
labels=labels,
autopct="%1.1f%%",
startangle=90
)
plt.title("BAD: Combined pie for parental education + test prep\n(doubles counts, >100%)")
plt.tight_layout()
plt.show()
# --
# 11.1(b) BAD / MISLEADING – Mixed bar chart
# --
# Put both sets of percentages into one table
bad_bar = (
pd.concat([
parent_props.rename("percentage"),
prep_props.rename("percentage")
])
.reset_index()
Type here to search
1 c...
ENG
19:12
18-11-2025


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Markdown
# --
# 11.1(b) BAD / MISLEADING – Mixed bar chart
# --
# Put both sets of percentages into one table
bad_bar = (
pd.concat([
parent_props.rename("percentage"),
prep_props.rename("percentage")
])
.reset_index()
)
bad_bar.rename(columns={"index": "category"}, inplace=True)
# Tag which group each category belongs to
bad_bar["group"] = np.where(
bad_bar["category"].isin(parent_props.index),
"Parental education",
"Test prep"
)
print("\nData for mixed bar chart:\n", bad_bar)
plt.figure(figsize=(8, 4))
plt.bar(bad_bar["category"], bad_bar["percentage"])
plt.xticks(rotation=45, ha="right")
plt.ylabel("Percentage of students")
plt.title("BAD: Mixed categories in one bar plot\n(hides overlap between groups)")
plt.tight_layout()
plt.show()
Parental education percentages:
associate's degree 22.2
bachelor's degree 11.8
high school 19.6
master's degree 5.9
some college 22.6
some high school 17.9
Type here to search
1 c...
ENG
19:12
18-11-2025



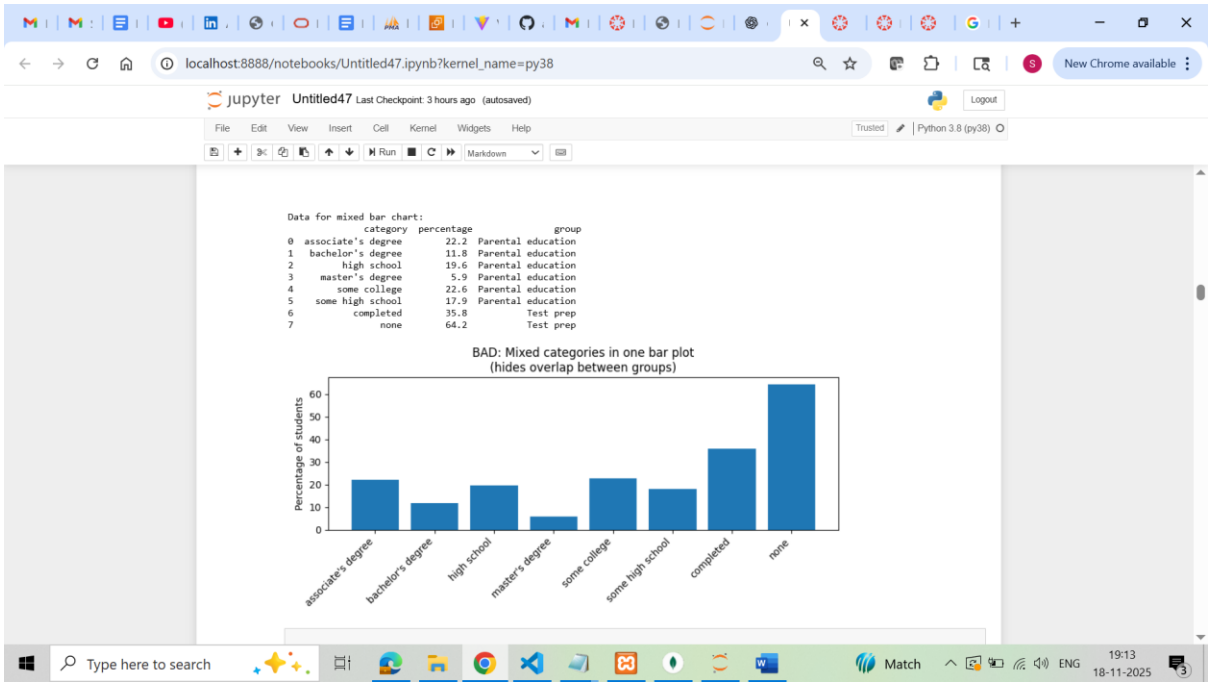
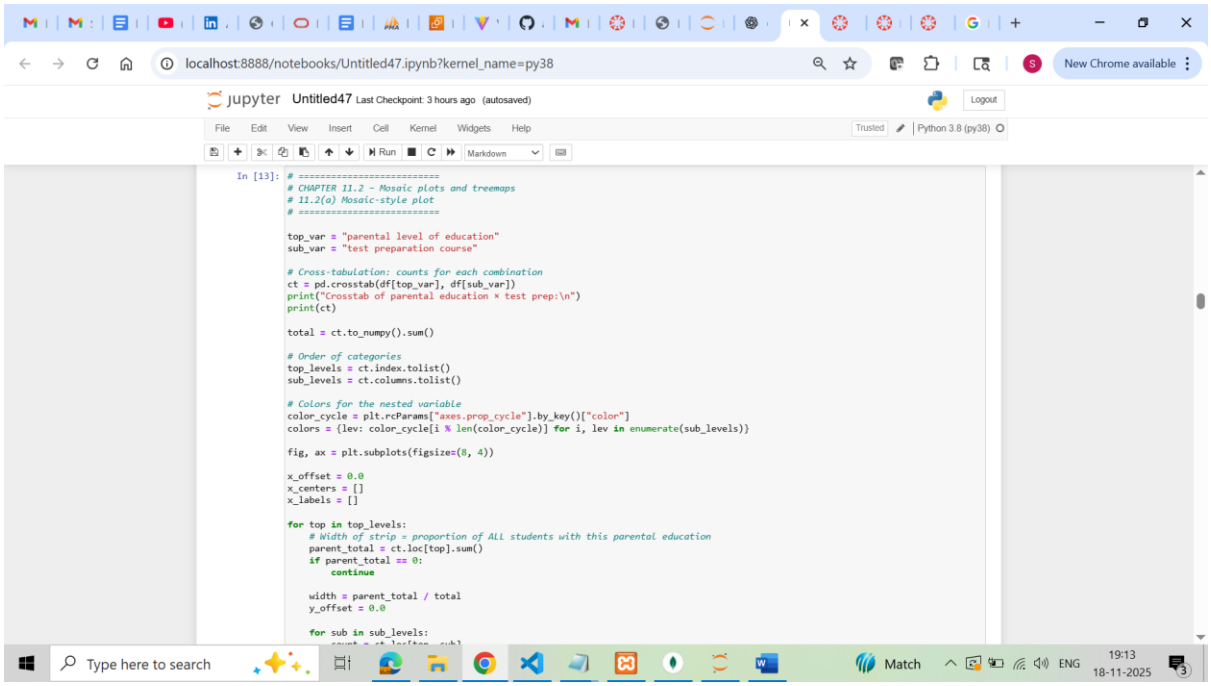


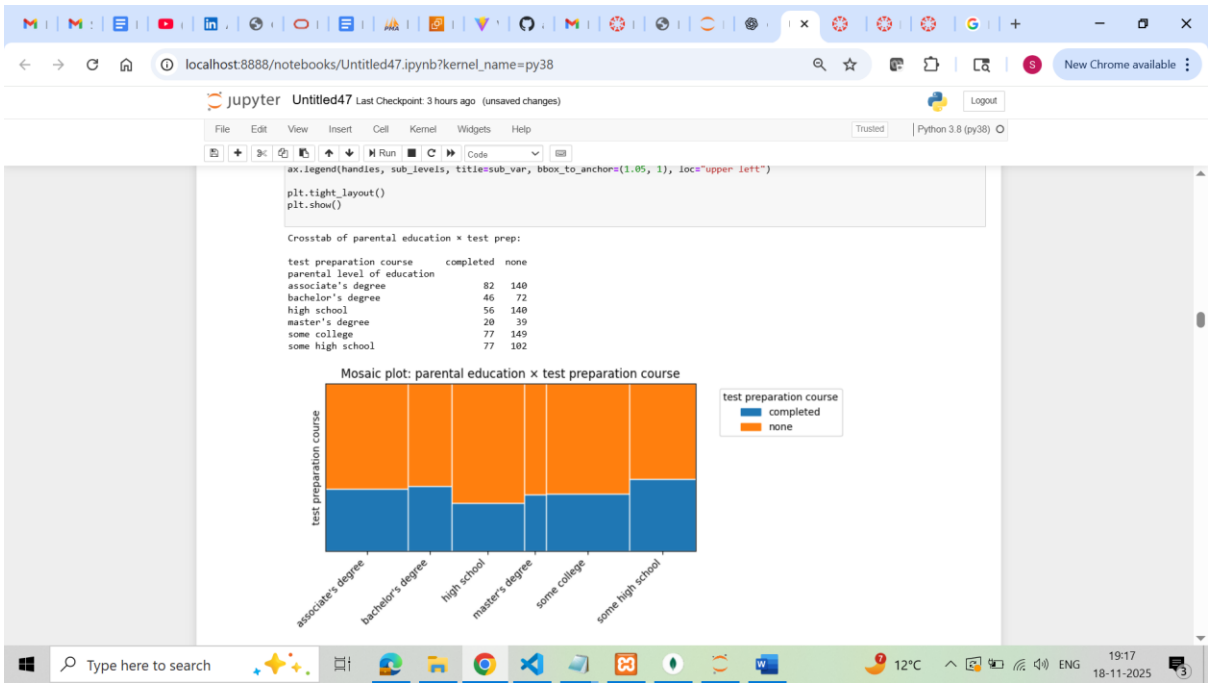
Figure 11.1(a) – BAD combined pie

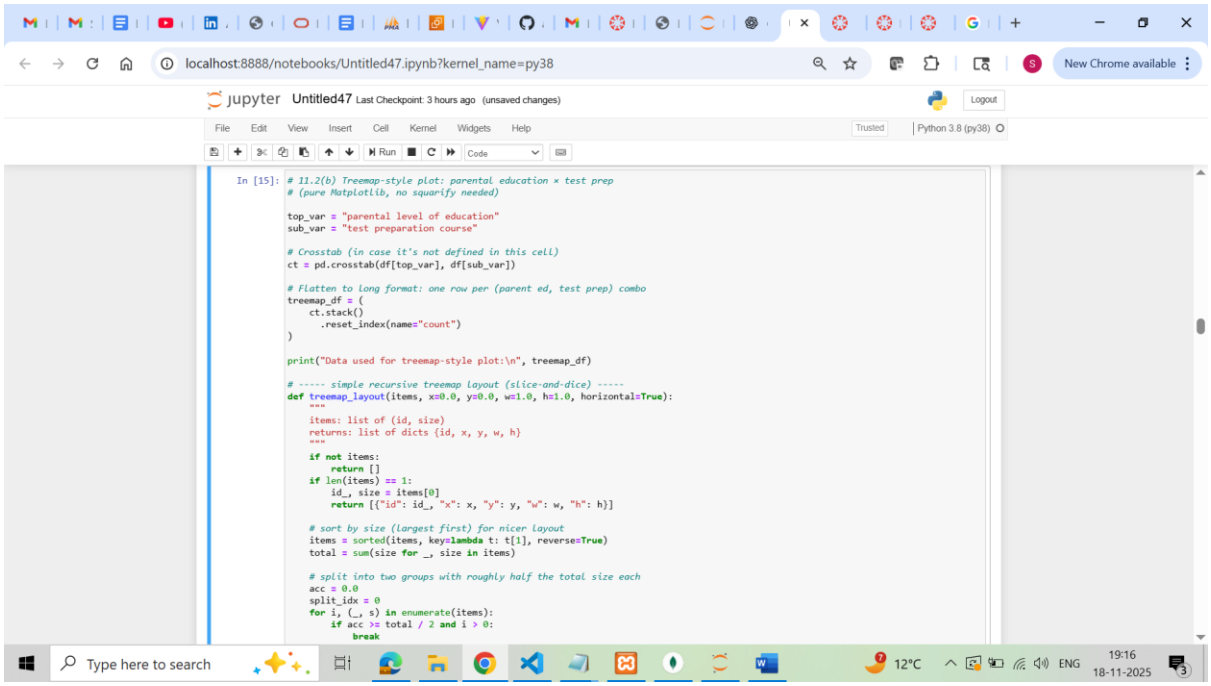
I combined the percentages of parental education levels and test-preparation completion into a single pie chart. The slices sum to more than 100%, because each student is counted once in a parental-education category and again in a test-prep category. A pie chart implies mutually exclusive categories that add up to one whole, which is violated here, so this is a “nested proportions gone wrong” example.

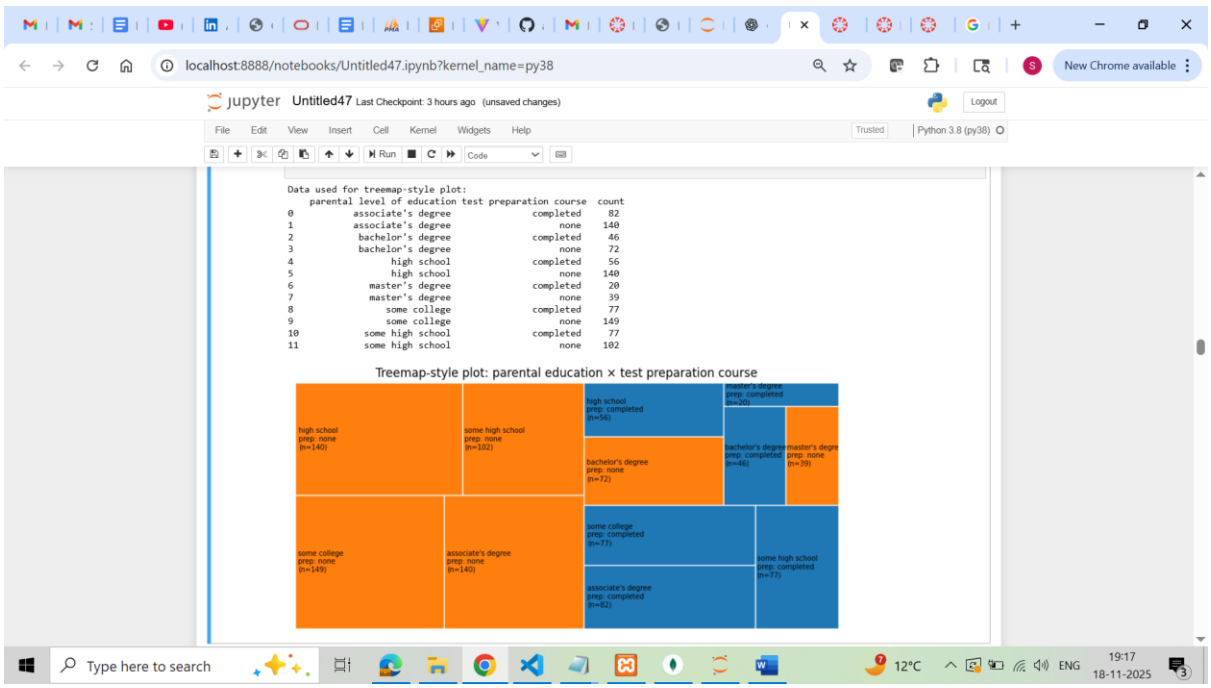
Figure 11.1(b) – BAD / misleading bar chart

The bar chart shows the same percentages as separate bars. This plot is not mathematically wrong, since bar heights don't need to add to 100%. However, it is conceptually misleading: it suggests many independent categories, even though all test-prep bars are just another way of slicing the same students that already appear in the parental-education bars. The overlap between groups is completely hidden.

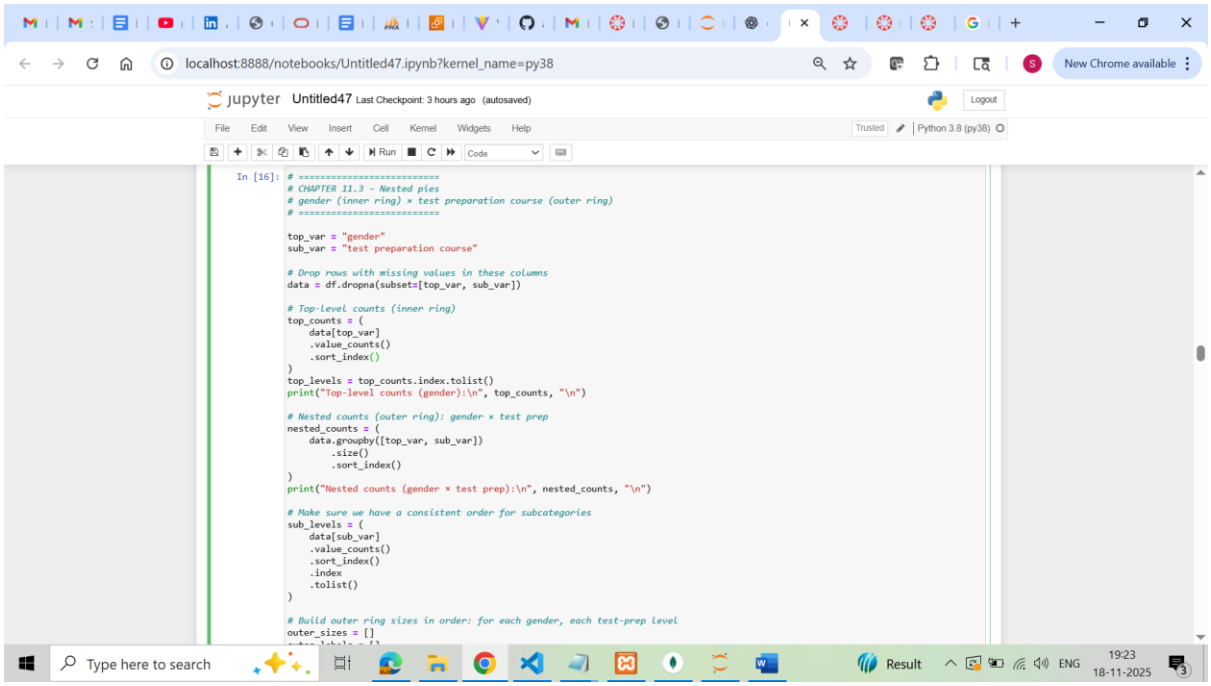








The treemap-style plot divides the full plotting area into rectangles whose area is proportional to the number of students in each combination of parental education level and test-preparation status. All rectangles together represent the entire sample. Colors encode test-preparation status, while text labels show the parent-education category and the group size. Like the mosaic plot, this correctly visualizes nested proportions, but here the rectangles are arranged in a free-form tiling rather than a strict grid.



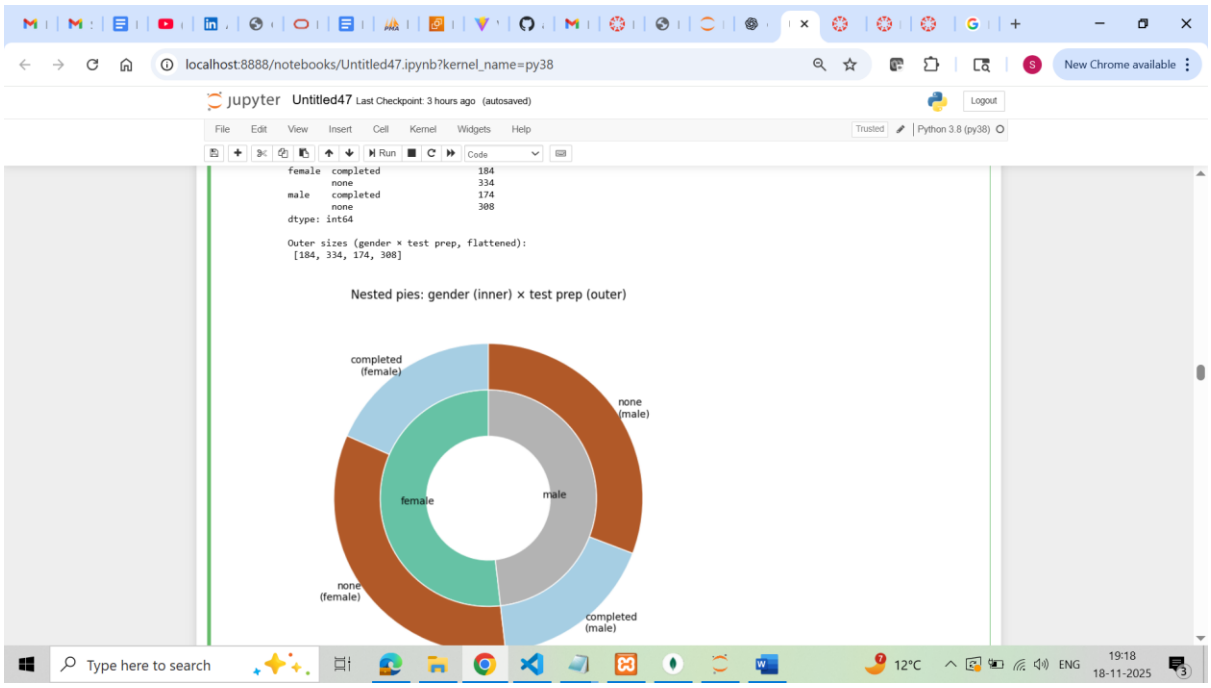
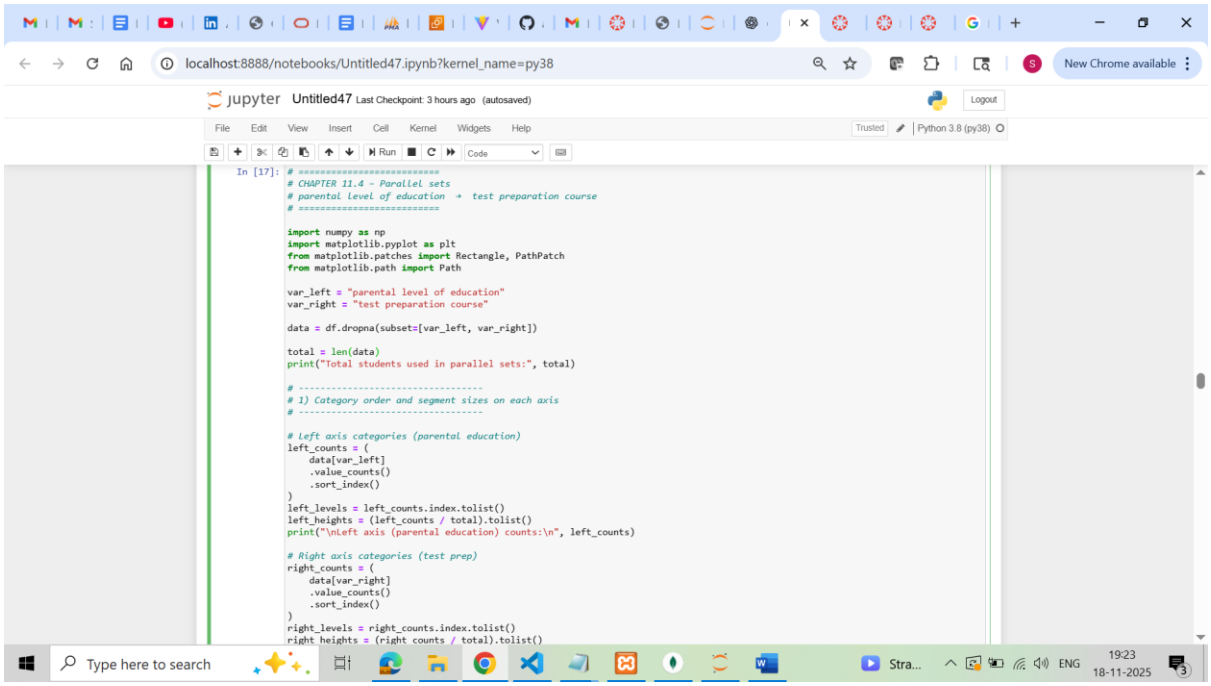
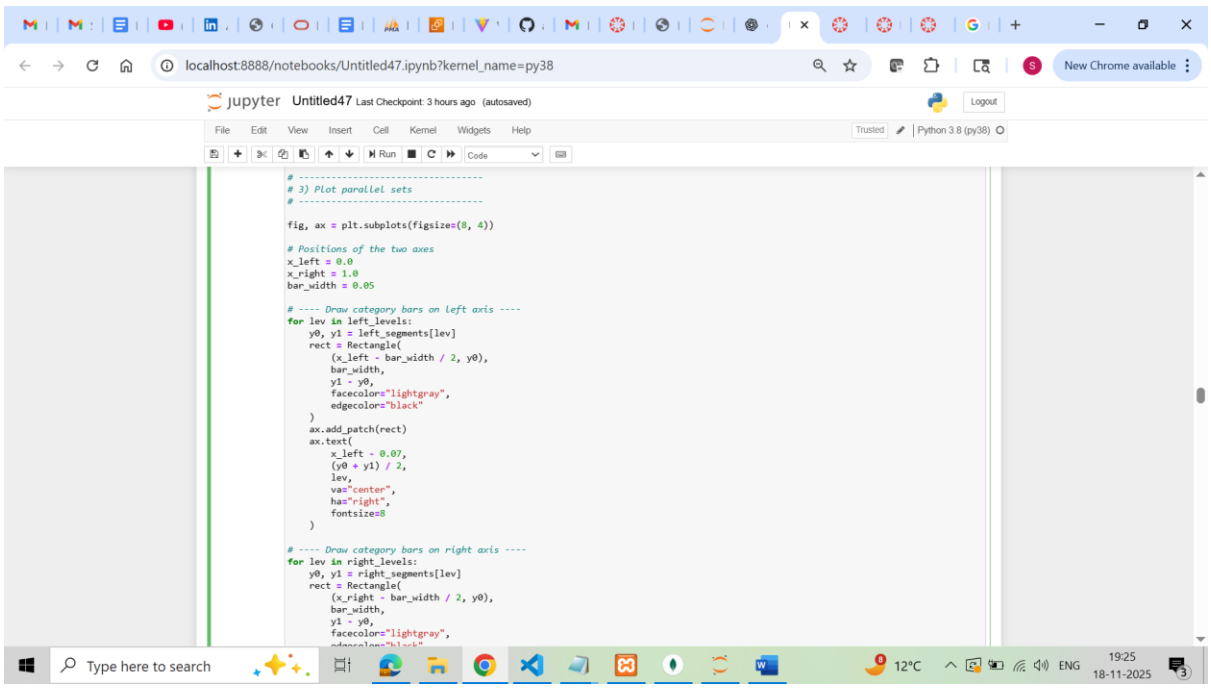


Figure 11.3 – Nested pies for gender and test preparation course.

The inner ring shows the proportion of female versus male students in the sample. The outer ring breaks each gender slice into students who completed the test-preparation course and those who did not. The area and angle of each outer wedge represent the fraction of all students in that gender–test-prep combination. This is a nested-pies visualization of the same nested proportions we showed earlier with mosaic and treemap plots.


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Code
In [17]:
# ========================
# CHAPTER 11.4 – Parallel sets
# parental level of education → test preparation course
# ========================
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle, PathPatch
from matplotlib.path import Path
var_left = "parental level of education"
var_right = "test preparation course"
data = df.dropna(subset=[var_left, var_right])
total = len(data)
print("Total students used in parallel sets:", total)
# ----------------------------------
# 1) Category order and segment sizes on each axis
# ----------------------------------
# Left axis categories (parental education)
left_counts = (
data[var_left]
.value_counts()
.sort_index()
)
left_levels = left_counts.index.tolist()
left_heights = (left_counts / total).tolist()
print("\nLeft axis (parental education) counts:\n", left_counts)
# Right axis categories (test prep)
right_counts = (
data[var_right]
.value_counts()
.sort_index()
)
right_levels = right_counts.index.tolist()
right_heights = (right_counts / total).tolist()
Type here to search
Stra...
ENG
19:23
18-11-2025


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Code
----
3) Plot parallel sets
----
fig, ax = plt.subplots(figsize=(8, 4))
# Positions of the two axes
x_left = 0.0
x_right = 1.0
bar_width = 0.05
# ---- Draw category bars on left axis ----
for lev in left_levels:
y0, y1 = left_segments[lev]
rect = Rectangle(
(x_left - bar_width / 2, y0),
bar_width,
y1 - y0,
facecolor="lightgray",
edgecolor="black"
)
ax.add_patch(rect)
ax.text(
x_left - 0.07,
(y0 + y1) / 2,
lev,
va="center",
ha="right",
fontsize=8
)
# ---- Draw category bars on right axis ----
for lev in right_levels:
y0, y1 = right_segments[lev]
rect = Rectangle(
(x_right - bar_width / 2, y0),
bar_width,
y1 - y0,
facecolor="lightgray",
Type here to search
12°C
ENG
19:25
18-11-2025

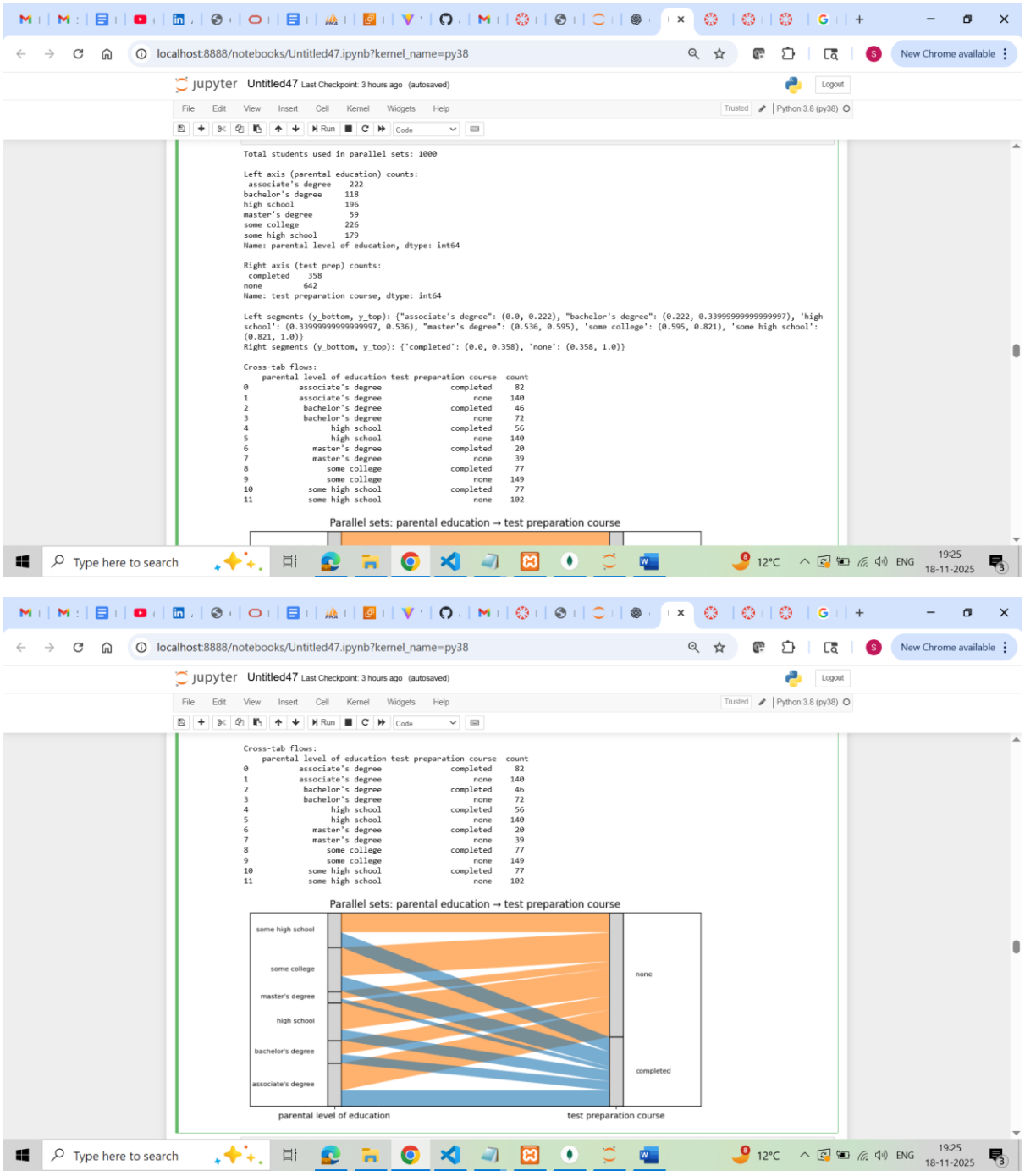


Figure 11.4 – Parallel sets for parental education and test preparation course.

The left axis shows how all students are distributed across parental education levels, and the right axis shows how they are distributed across test-preparation status. The shaded bands connect categories on the left to categories on the right. The thickness of each band is proportional to the number of students in that combination (e.g., “some college” + “completed test prep”). Coloring the bands by test-preparation status highlights where most of the “completed” students come from. This is a simple two-axis parallel sets plot; the same idea can be extended to three or more categorical variables, as in the textbook example with bridge material, length, era, and river.

localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)

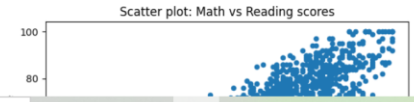
File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

CHAPTER 12

In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (6, 4)

# df already loaded from URL earlier
# df = pd.read_csv(url)
```

In [19]:
```
# ==========================
# CHAPTER 12.1 - Scatter plots
# 12.1(a) Basic scatter: math vs reading
# ==========================

plt.figure()
plt.scatter(
    df["math score"],
    df["reading score"],
    s=20  # marker size
)
plt.xlabel("Math score")
plt.ylabel("Reading score")
plt.title("Scatter plot: Math vs Reading scores")
plt.tight_layout()
plt.show()
```



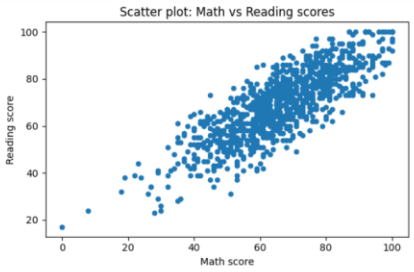
Type here to search 12°C ENG 19:26 18-11-2025

localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)

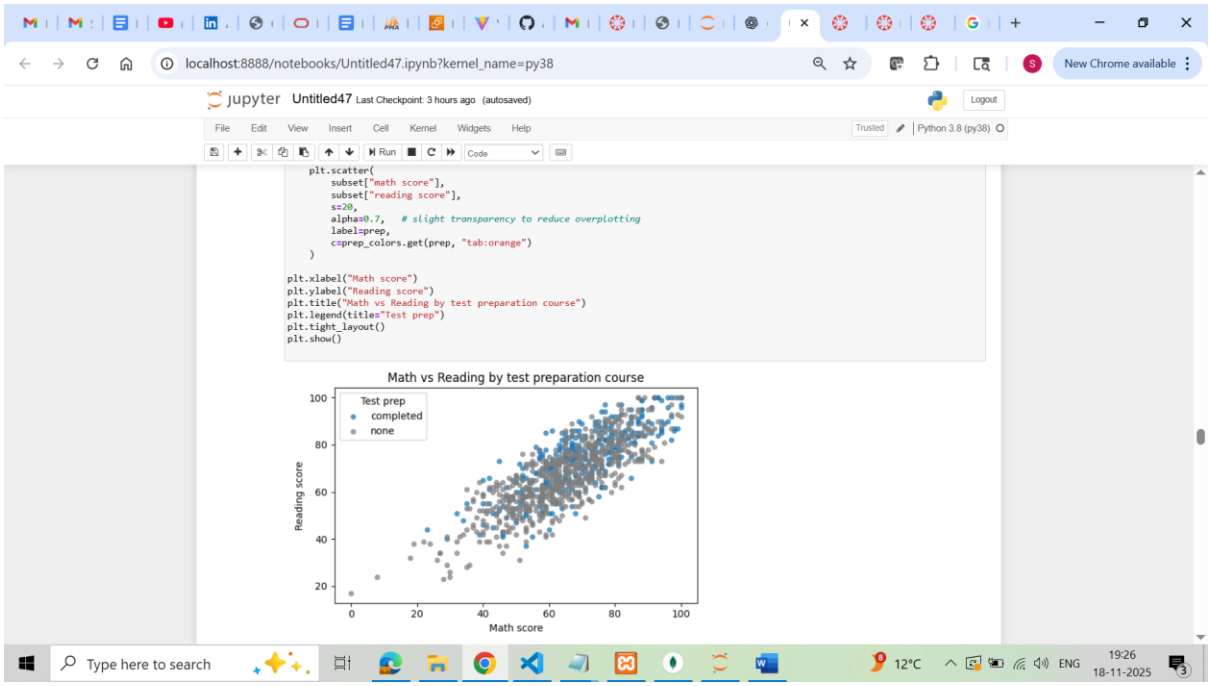
File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)



In [20]:
```
# 12.1(b) Scatter: Math vs Reading, colored by test prep

prep_colors = {
    "none": "tab:gray",
    "completed": "tab:blue"
}

plt.figure()
for prep, subset in df.groupby("test preparation course"):
    plt.scatter(
        subset["math score"],
        subset["reading score"],
        s=20,
        alpha=0.7,   # slight transparency to reduce overplotting
        label=prep,
```

Type here to search 12°C ENG 19:26 18-11-2025



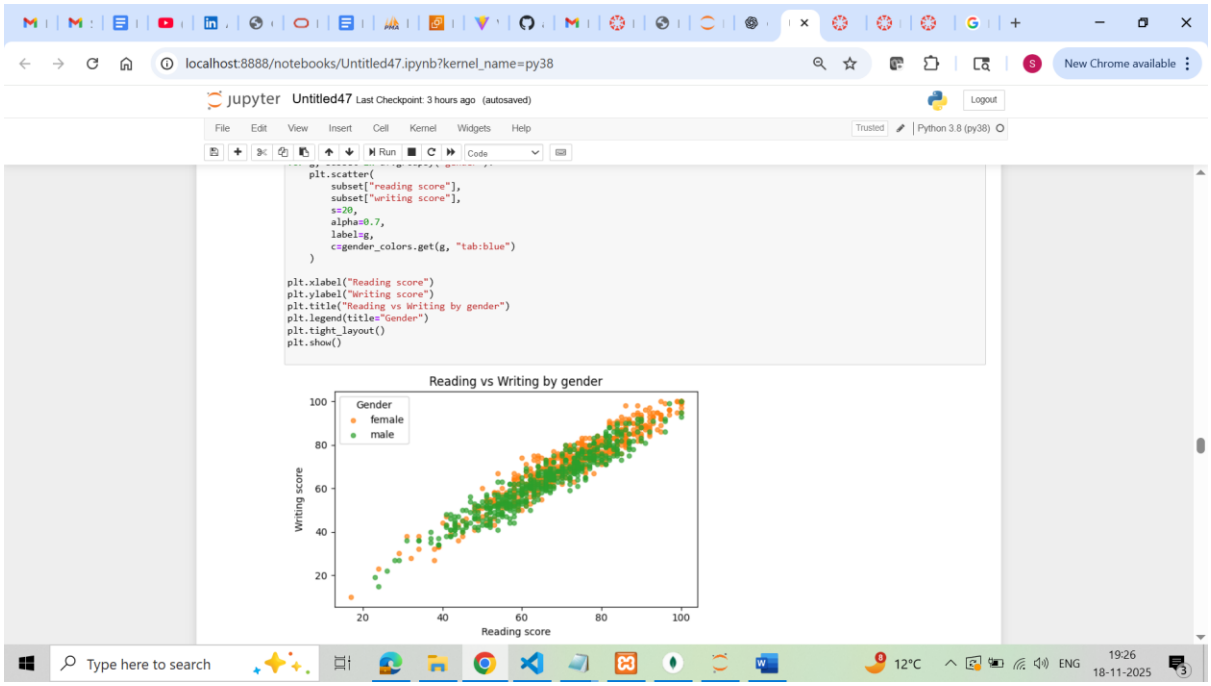


Figure 12.1(a) shows a basic scatter plot of math score versus reading score.

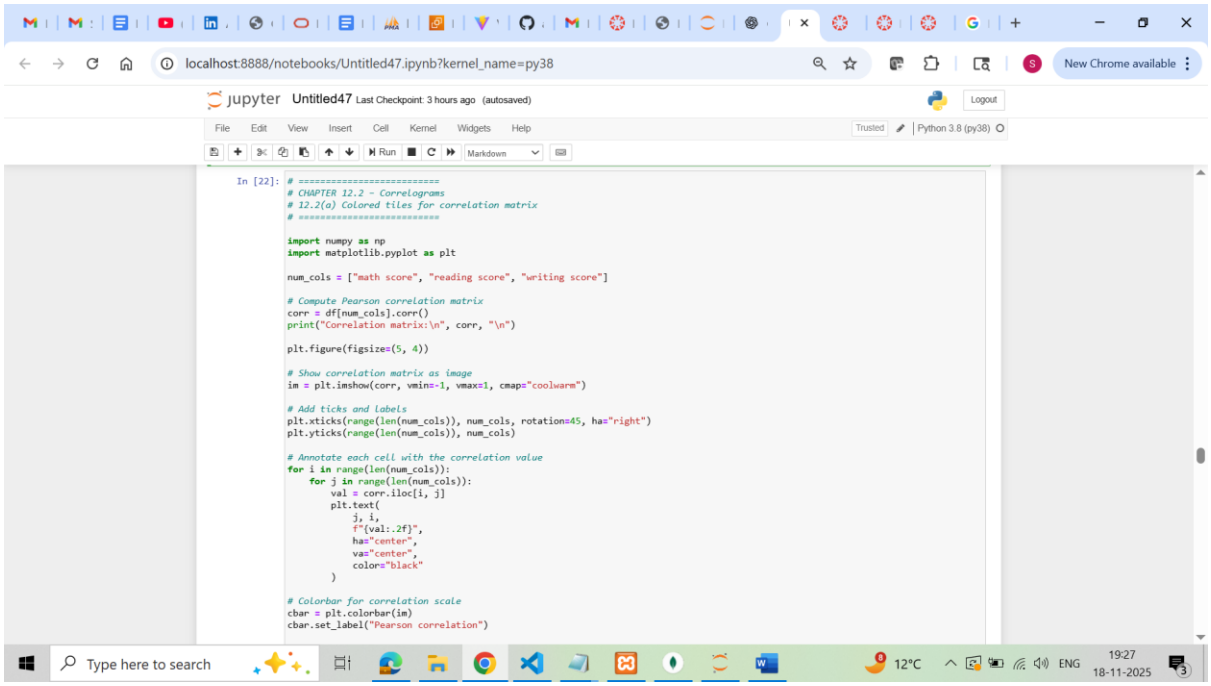
The points form a clear upward cloud, indicating a strong positive association: students with higher math scores also tend to have higher reading scores.

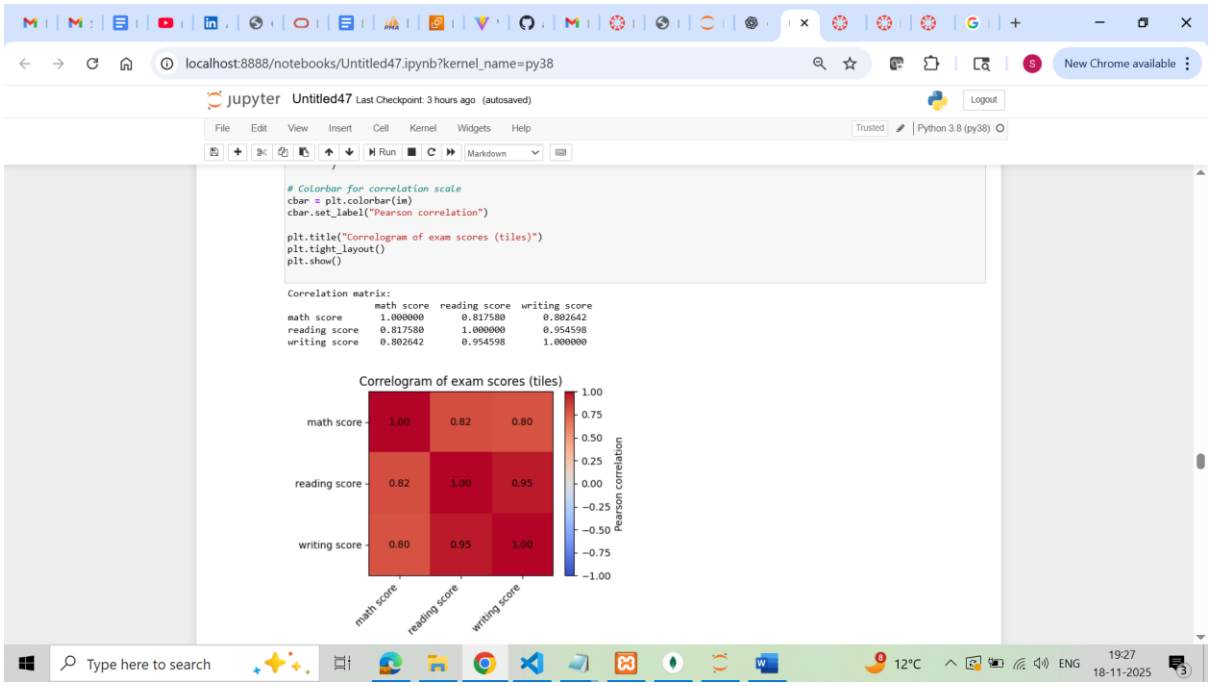
Figure 12.1(b) colors the same scatter plot by test-preparation course.

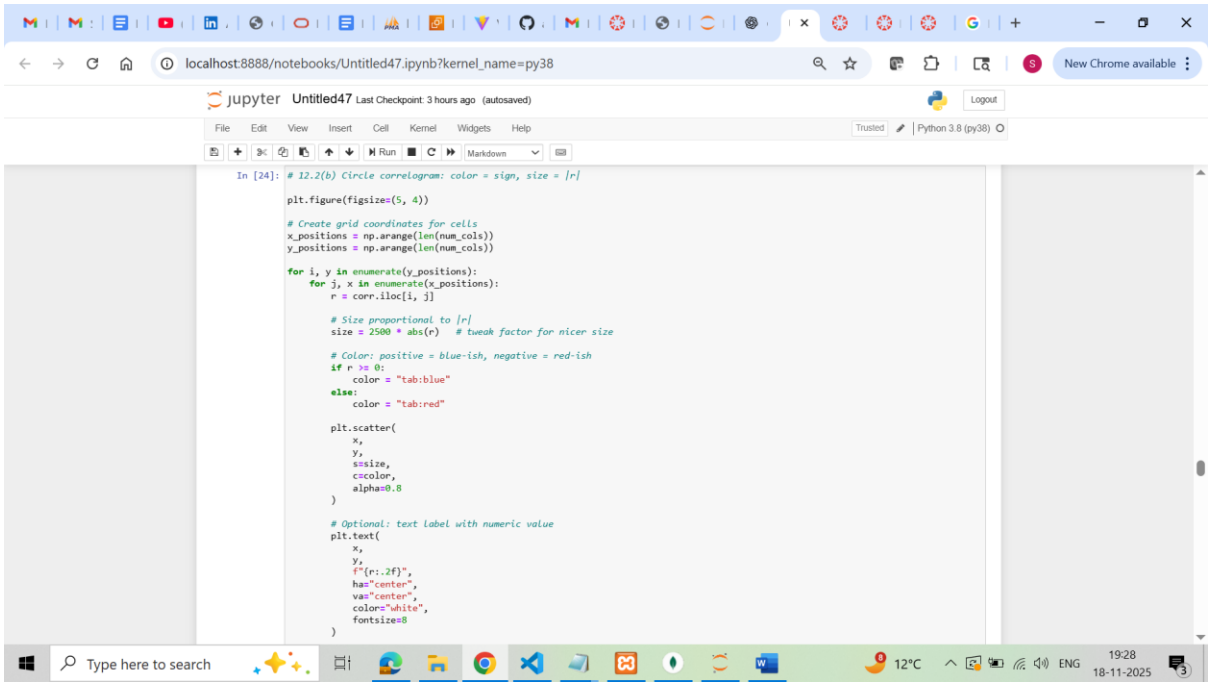
Students who completed the test-preparation course (blue) tend to cluster in the upper-right region (high math and reading scores), while students with no prep (gray) are more spread out and more common at lower scores. This shows how scatter plots can incorporate a third variable via color to reveal group differences.

Figure 12.1(c) shows reading score versus writing score, colored by gender.

The relationship between reading and writing scores is almost perfectly linear and positive, with both genders following the same general pattern. Any gender differences are subtle compared to the strong overall association between the two quantitative variables.







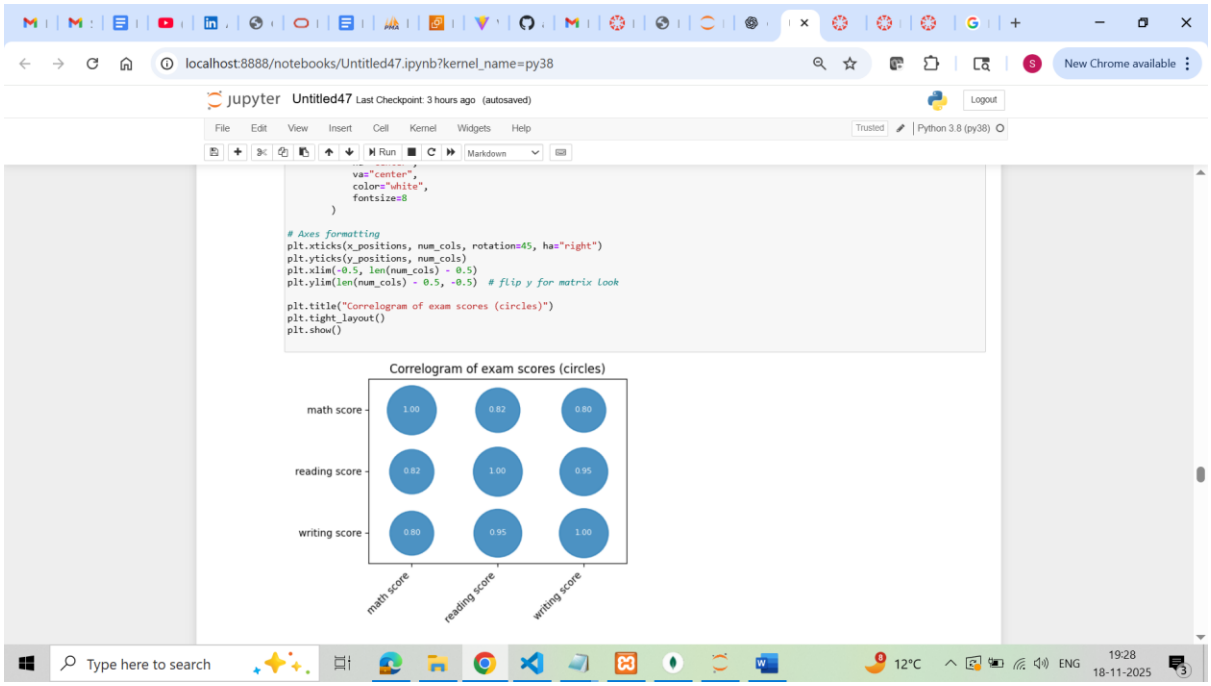


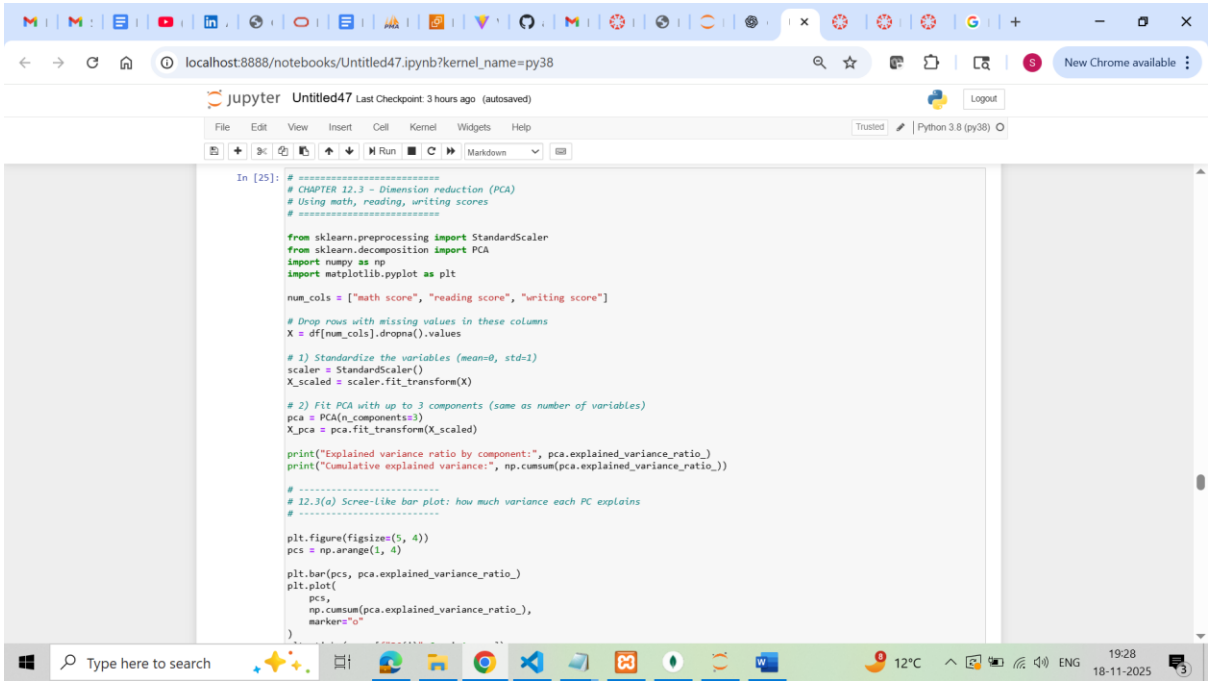
Figure 12.2(a) – Correlogram with colored tiles.

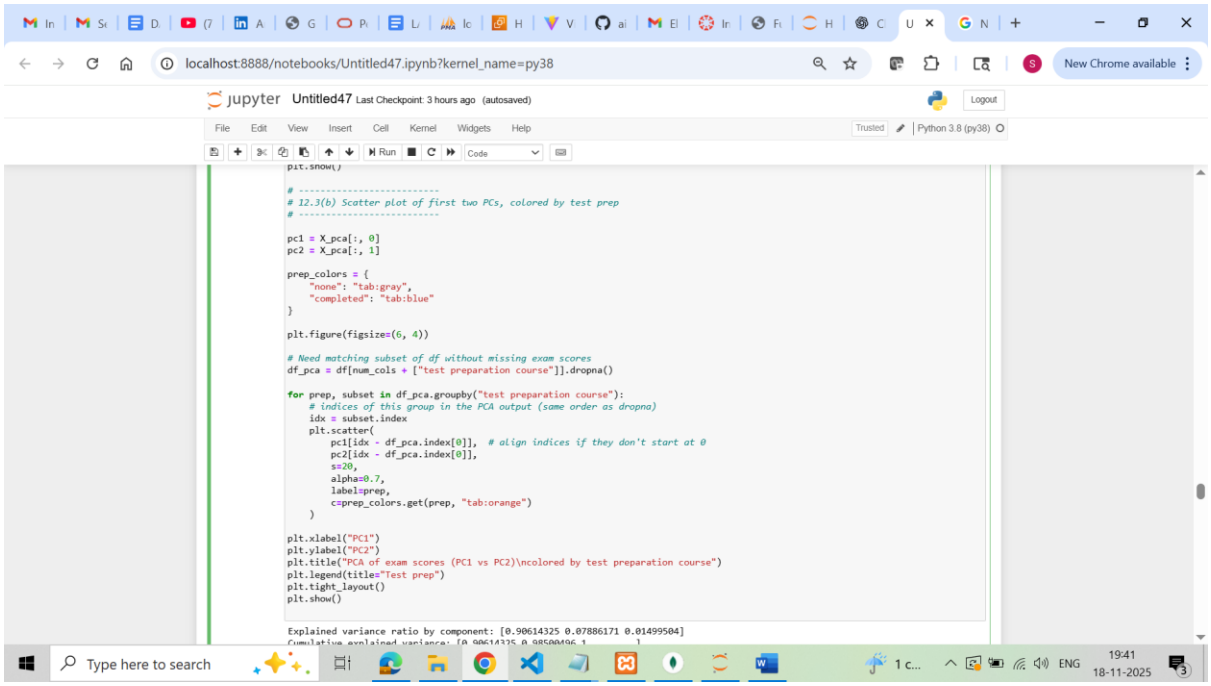
I computed the Pearson correlation matrix for math, reading, and writing scores and displayed it as a tile plot. All off-diagonal correlations are strongly positive (around 0.8–0.95), meaning that students who do well on one exam tend to do well on the others. The color scale from blue to red encodes correlation strength and sign, similar to the correlogram example in the textbook.

Figure 12.2(b) – Circle correlogram.

This plot shows the same correlation matrix, but each cell is represented by a circle whose color indicates the sign of the correlation and whose size scales with the absolute value of the correlation coefficient. Large blue circles highlight strong positive relationships (e.g., reading vs. writing), while

any correlations near zero would appear as very small circles, visually deemphasizing weak associations as recommended in the chapter





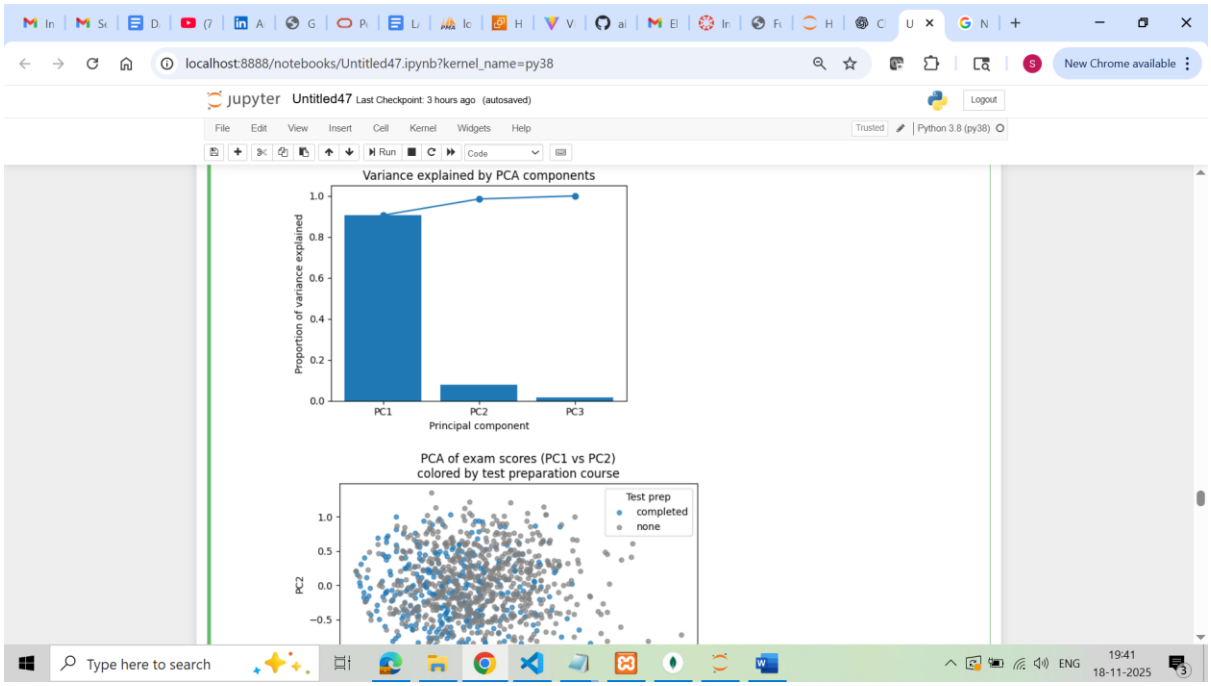


Figure 12.3(a) – Variance explained by PCA.

I applied principal component analysis (PCA) to the three standardized exam scores (math, reading, writing). The bar/line plot shows how much variance each principal component explains. The first component (PC1) explains most of the total variance, while PC2 and PC3 contribute much less. This indicates that a single underlying dimension—roughly “overall exam performance”—captures most of the variation in these three scores.

Figure 12.3(b) – PC1 vs PC2 scatter plot colored by test-preparation course.

The scatter plot shows the first two principal components. Each point is a student; color indicates whether they completed the test-preparation course. PC1 separates students largely by overall performance: higher PC1 values correspond to higher scores on all three exams. Students who completed test prep tend to lie toward the higher-PC1 region, reinforcing what we saw in the original scatter plots. PC2 mostly captures smaller differences between the individual exams. PCA thus reduces the three correlated exam scores to two new variables that summarize performance while retaining most of the structure in the data.

localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

In [26]:
```
# ============================
# CHAPTER 12.4 – Paired data
# 12.4(a) Scatter of reading vs writing with x = y line
# ============================

import numpy as np
import matplotlib.pyplot as plt

# Keep only rows with both scores
paired = df.dropna(subset=["reading score", "writing score"])

x = paired["reading score"]
y = paired["writing score"]

plt.figure()

# Scatter of paired measurements (one point per student)
plt.scatter(x, y, s=20, alpha=0.7)

# Diagonal line x = y (no difference between reading and writing)
line_min = 0
line_max = 100
plt.plot(
    [line_min, line_max],
    [line_min, line_max],
    linestyle="--"
)

plt.xlabel("Reading score")
plt.ylabel("Writing score")
plt.title("Paired data: Reading vs Writing scores\nwith x = y reference line")
plt.xlim(line_min, line_max)
plt.ylim(line_min, line_max)
plt.tight_layout()
plt.show()
```

Paired data: Reading vs Writing scores

Type here to search

Match ENG 19:42 18-11-2025

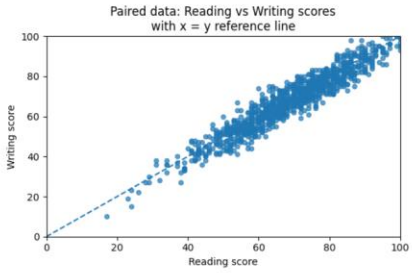
localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

```
plt.xlabel("Reading score")
plt.ylabel("Writing score")
plt.title("Paired data: Reading vs Writing scores\nwith x = y reference line")
plt.xlim(line_min, line_max)
plt.ylim(line_min, line_max)
plt.tight_layout()
plt.show()
```

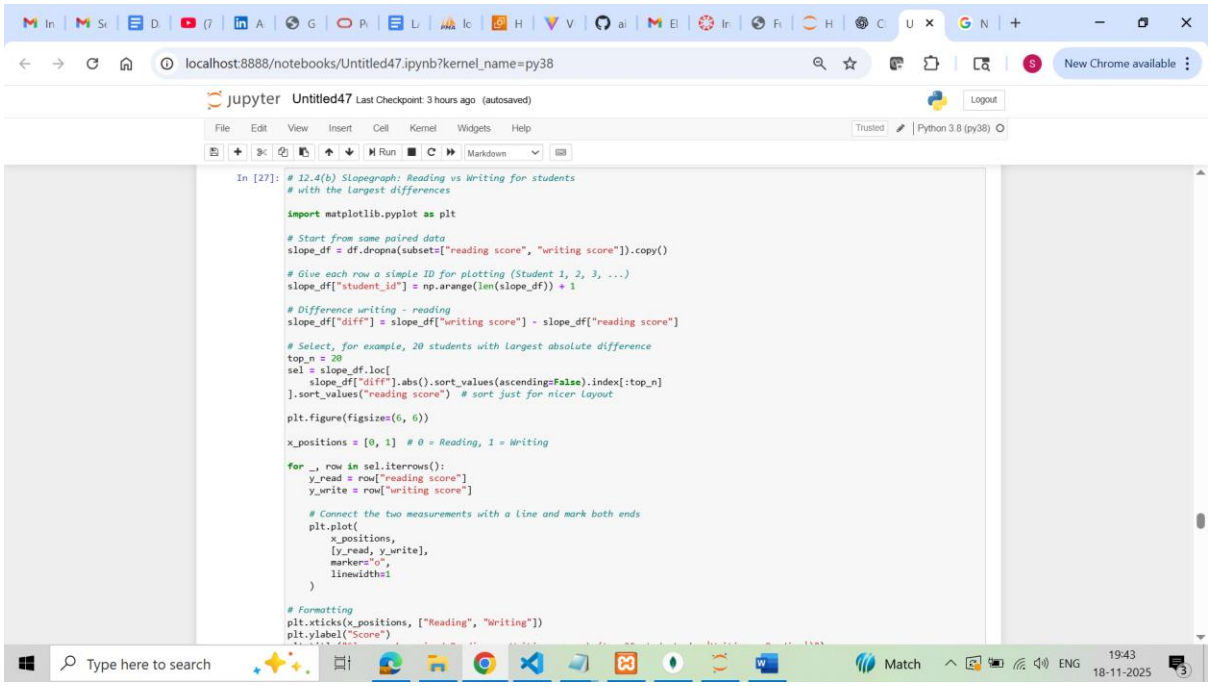


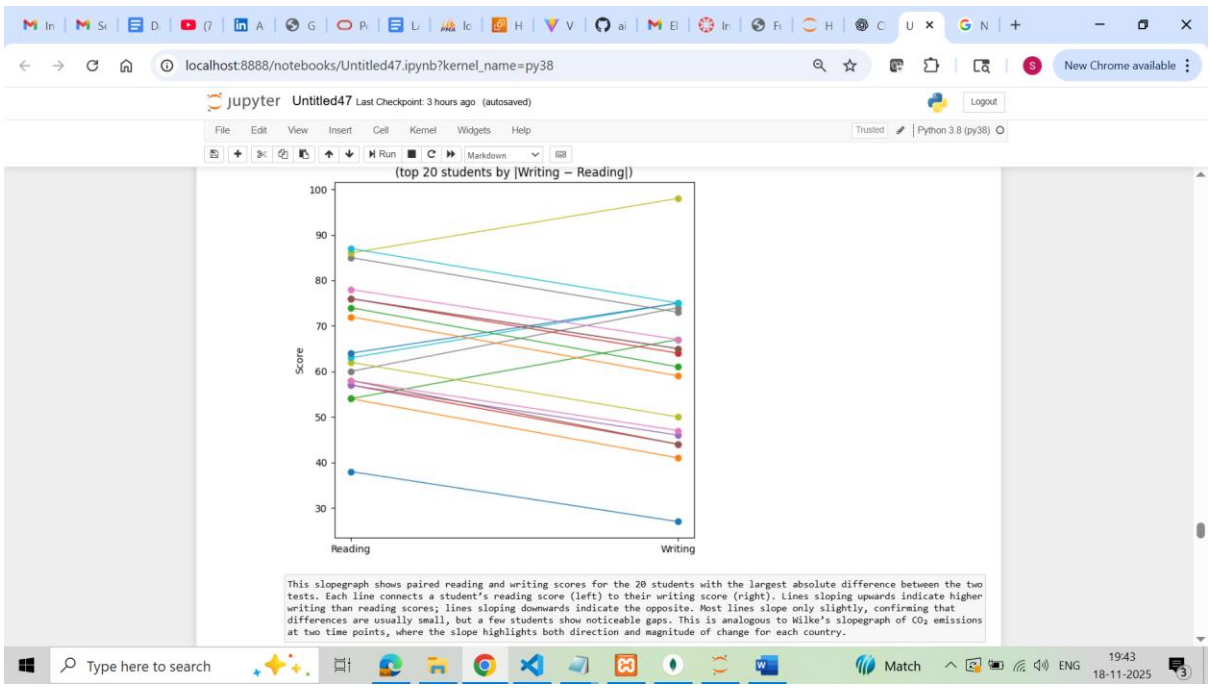
This scatter plot treats reading and writing scores as paired measurements for each student. The dashed diagonal shows where reading and writing scores would be equal. Most points lie close to the diagonal, indicating that the two tests measure very similar ability. Small deviations above or below the line show students who are slightly stronger in writing or reading, respectively. This mirrors the textbook's example where paired CO_2 measurements for countries are plotted against a diagonal x = y line to highlight systematic differences

In [27]: # 12.4(b) Slopegraph: Reading vs Writing for students

Type here to search

Match ENG 19:42 18-11-2025


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Markdown
In [27]:
# 12.4(b) Slopegraph: Reading vs Writing for students
# with the largest differences
import matplotlib.pyplot as plt
# Start from same paired data
slope_df = df.dropna(subset=["reading score", "writing score"]).copy()
# Give each row a simple ID for plotting (Student 1, 2, 3, ...)
slope_df["student_id"] = np.arange(len(slope_df)) + 1
Difference writing - reading
slope_df["diff"] = slope_df["writing score"] - slope_df["reading score"]
# Select, for example, 20 students with largest absolute difference
top_n = 20
sel = slope_df.loc[
slope_df["diff"].abs().sort_values(ascending=False).index[:top_n]
].sort_values("reading score") # sort just for nicer layout
plt.figure(figsize=(6, 6))
x_positions = [0, 1] # 0 = Reading, 1 = Writing
for _, row in sel.iterrows():
y_read = row["reading score"]
y_write = row["writing score"]
# Connect the two measurements with a line and mark both ends
plt.plot(
x_positions,
[y_read, y_write],
marker="o",
linewidth=1
)
# Formatting
plt.xticks(x_positions, ["Reading", "Writing"])
plt.ylabel("Score")
Type here to search
Match
ENG
19:43
18-11-2025


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
New Chrome available
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)
Run Markdown
(top 20 students by |Writing − Reading|)
Score
Reading
Writing
This slopegraph shows paired reading and writing scores for the 20 students with the largest absolute difference between the two tests. Each line connects a student's reading score (left) to their writing score (right). Lines sloping upwards indicate higher writing than reading scores; lines sloping downwards indicate the opposite. Most lines slope only slightly, confirming that differences are usually small, but a few students show noticeable gaps. This is analogous to Wilke's slopegraph of CO_2 emissions at two time points, where the slope highlights both direction and magnitude of change for each country.
Type here to search
Match
ENG
19:43
18-11-2025

CHAPTER 13

CHAPTER 13

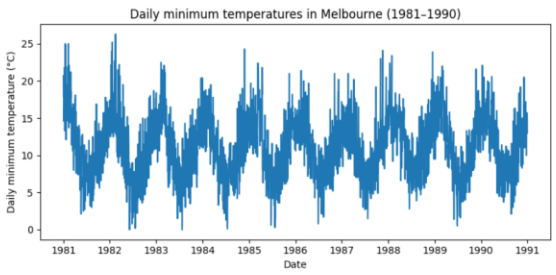
In [28]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (8, 4)

# Daily minimum temperatures in Melbourne (1981-1990)
url_temp = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv"
temp_df = pd.read_csv(url_temp)

# Parse dates
temp_df["Date"] = pd.to_datetime(temp_df["Date"])
temp_df = temp_df.rename(columns={"Temp": "temp"})
temp_df = temp_df.set_index("Date").sort_index()

print(temp_df.head())
print(temp_df.index.min(), "→", temp_df.index.max())
```

```
            temp
Date
1981-01-01  20.7
1981-01-02  17.9
1981-01-03  18.8
1981-01-04  14.6
1981-01-05  15.8
1981-01-01 00:00:00 → 1990-12-31 00:00:00
```

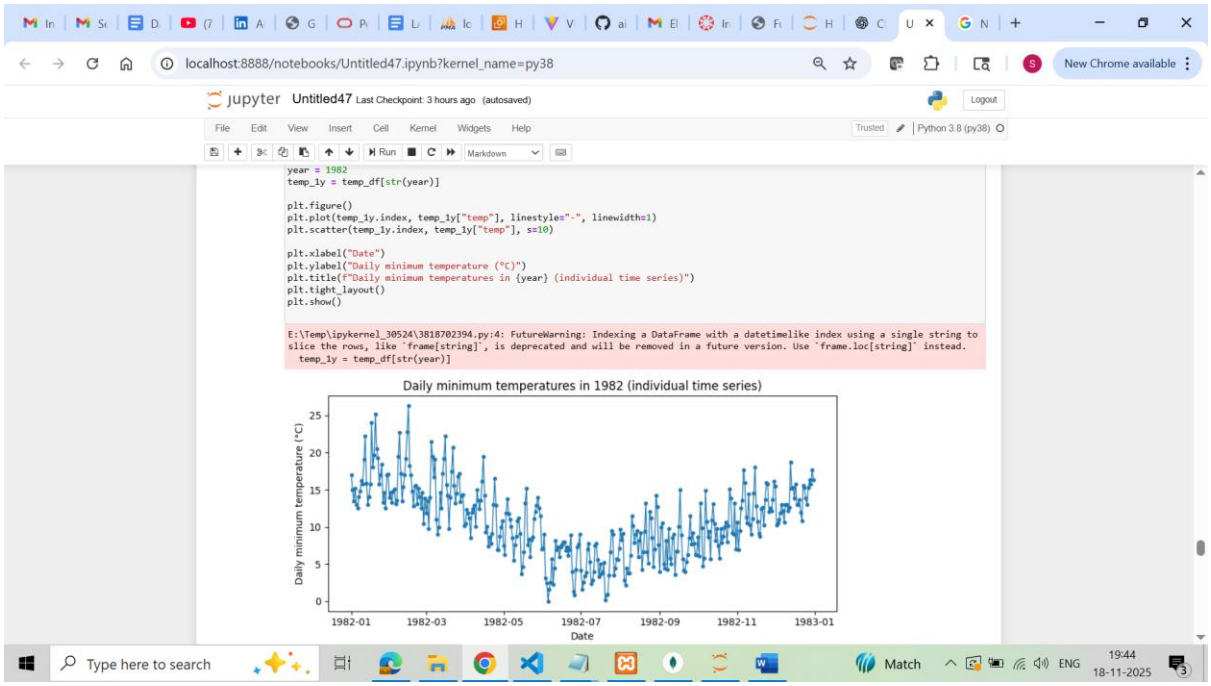
In [29]:
```
# 13.1(a) Individual time series: full daily temperature line

plt.figure()
plt.plot(temp_df.index, temp_df["temp"])
plt.xlabel("Date")
plt.ylabel("Daily minimum temperature (°C)")
plt.title("Daily minimum temperatures in Melbourne (1981-1990)")
plt.tight_layout()
```

localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
jupyter Untitled47 Last Checkpoint: 3 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3.8 (py38)

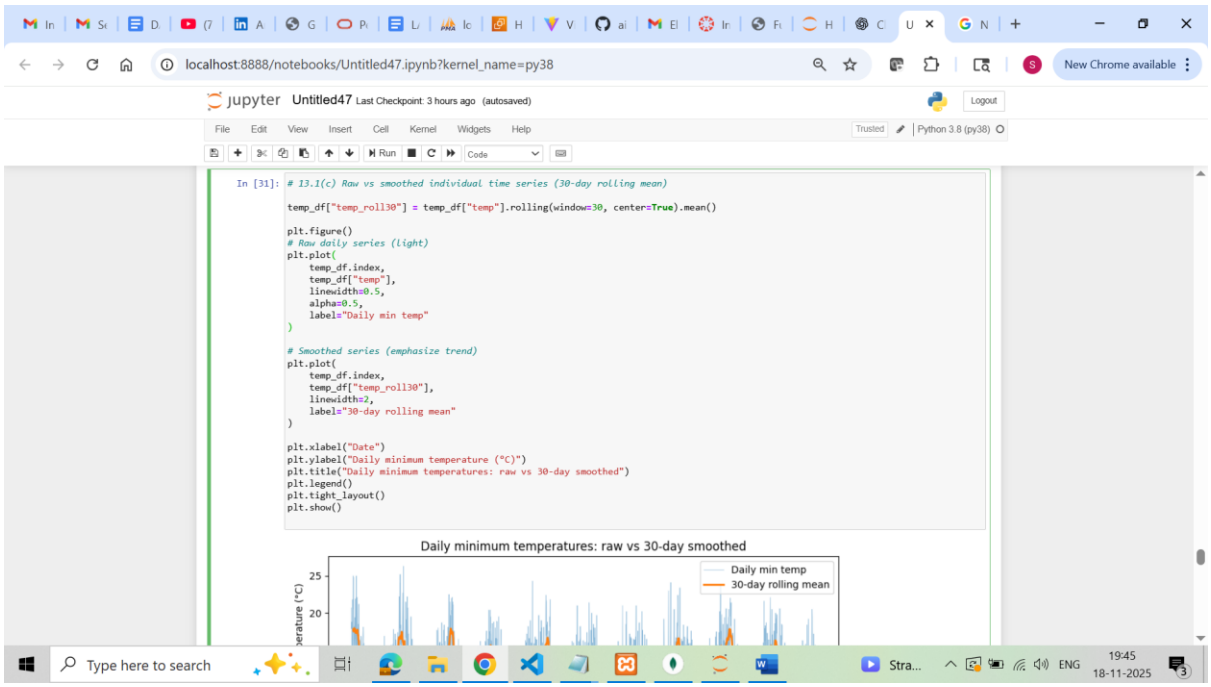
```
plt.figure()
plt.plot(temp_df.index, temp_df["temp"])
plt.xlabel("Date")
plt.ylabel("Daily minimum temperature (°C)")
plt.title("Daily minimum temperatures in Melbourne (1981-1990)")
plt.tight_layout()
plt.show()
```

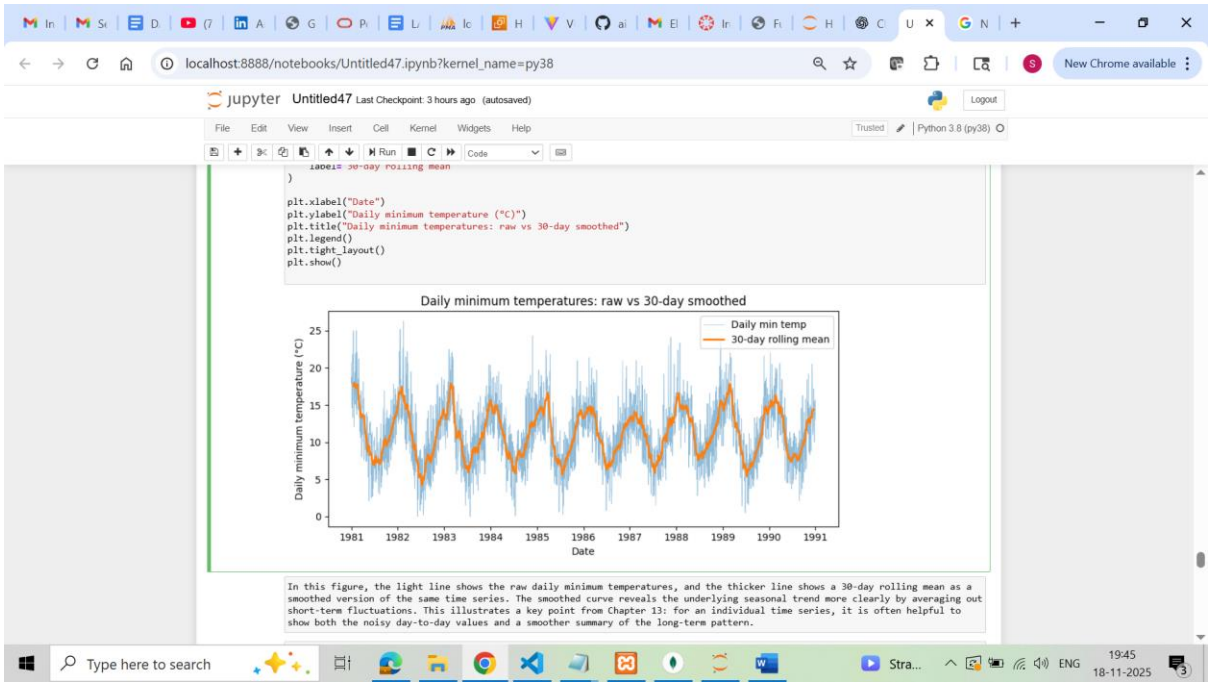


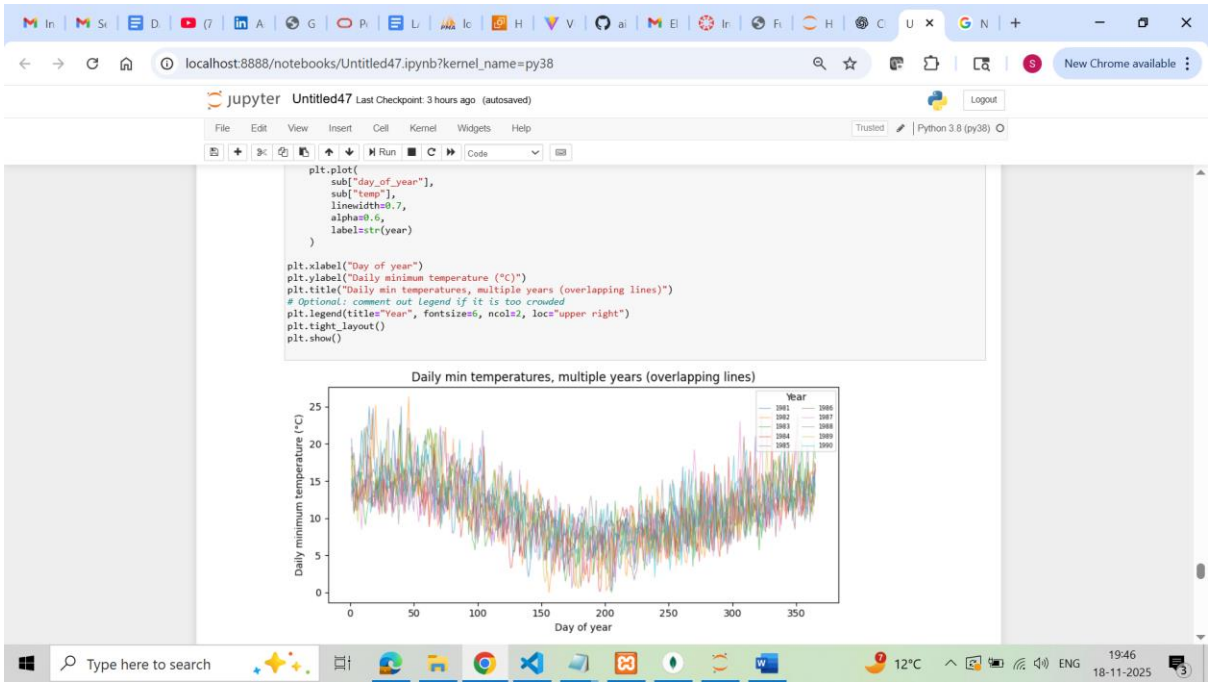
This line plot shows an individual time series of daily minimum temperatures in Melbourne from 1981 to 1990. The seasonal pattern is visible as regular up-and-down cycles: higher minima in the Australian summer and lower minima in winter. Short-term day-to-day fluctuations ride on top of the longer seasonal trend, matching the chapter's idea of using a simple line to show how a single quantity evolves over time.



Focusing on 1982 alone, I plotted the daily minimum temperatures as both a line and individual points. The line reveals the smooth seasonal rise and fall across the year, while the points emphasize that each dot is a separate daily observation. This matches the “individual time series” examples in the textbook, where both the trajectory and the discreteness of the data are important.







This figure plots a separate time series for each year (1981–1990) on the same axes. While it technically shows multiple time series, the result is a “spaghetti plot” where lines overlap heavily. It is hard to follow any single year, and comparing years is difficult. This is a typical “bad” multiple time series design mentioned in the book, where adding many lines to one panel makes the visualization cluttered instead of informative.

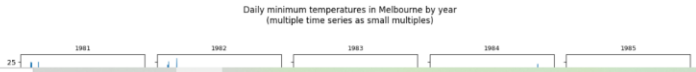
localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter **Untitled47** Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

In [35]:
```
# 13.2(b) BETTER: small multiples (one time series per panel, same scales)

unique_years = sorted(temp_df["year"].unique())
n_years = len(unique_years)

# We have 10 years; 2 rows x 5 columns works nicely
n_rows, n_cols = 2, 5

fig, axes = plt.subplots(n_rows, n_cols, figsize=(14, 5), sharex=True, sharey=True)

for ax, year in zip(axes.flatten(), unique_years):
    sub = temp_df[temp_df["year"] == year]
    ax.plot(
        sub["day_of_year"],
        sub["temp"],
        linewidth=0.8
    )
    ax.set_title(str(year), fontsize=9)
    ax.set_xticks([1, 91, 182, 274, 365])
    ax.set_xticklabels(["Jan", "Apr", "Jul", "Oct", "Dec"])

# Hide any unused axes (if any)
for ax in axes.flatten()[n_years:]:
    ax.set_visible(False)

# Common labels
fig.text(0.5, 0.04, "Day of year", ha="center")
fig.text(0.04, 0.5, "Daily minimum temperature (°C)", va="center", rotation="vertical")
fig.suptitle("Daily minimum temperatures in Melbourne by year\n(multiple time series as small multiples)", y=1.02)

plt.tight_layout()
plt.show()
```



Type here to search | 12°C | ENG | 19:47 | 18-11-2025

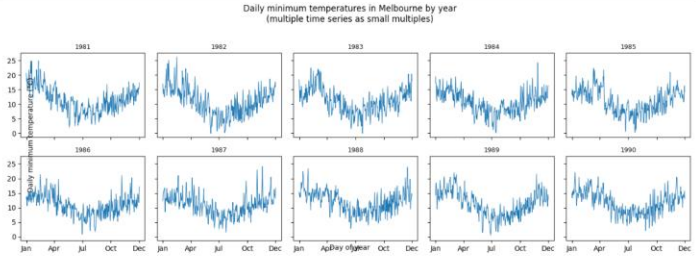
localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter **Untitled47** Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

```
# Common labels
fig.text(0.5, 0.04, "Day of year", ha="center")
fig.text(0.04, 0.5, "Daily minimum temperature (°C)", va="center", rotation="vertical")
fig.suptitle("Daily minimum temperatures in Melbourne by year\n(multiple time series as small multiples)", y=1.02)

plt.tight_layout()
plt.show()
```



Here I show the same ten yearly temperature series using small multiples: each panel contains one year, and all panels share the same x and y axes. This design makes it much easier to read the pattern for a single year and to compare years visually. For example, we can see that every year has a similar seasonal cycle, but some winters are slightly colder than others. This follows the chapter's recommendation to use small multiples or grouped displays for multiple time series instead of overlaid "spaghetti" plots.

In [ ]:

Type here to search | 12°C | ENG | 19:48 | 18-11-2025

localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter **Untitled47** Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

In [42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection

plt.rcParams["figure.figsize"] = (8, 4)

# Daily minimum temperatures (same dataset as 13.1)
url_temp = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv"
temp_df = pd.read_csv(url_temp)

temp_df["Date"] = pd.to_datetime(temp_df["Date"])
temp_df = (
    temp_df.rename(columns={"Temp": "temp"})
           .set_index("Date")
           .sort_index()
)

print("temp_df columns:", temp_df.columns)

# Monthly mean of daily minimum temps
monthly_ts = temp_df["temp"].resample("M").mean().to_frame("monthly_min_temp")

# Second response variable: 12-month rolling mean (trend)
monthly_ts["trend_12m"] = monthly_ts["monthly_min_temp"].rolling(12, center=True).mean()

print("monthly_ts columns:", monthly_ts.columns)
print(monthly_ts.head())
```

```
temp_df columns: Index(['temp'], dtype='object')
monthly_ts columns: Index(['monthly_min_temp', 'trend_12m'], dtype='object')
            monthly_min_temp  trend_12m
Date
1981-01-31         17.712903        NaN
1981-02-28         17.678571        NaN
1981-03-31         13.500000        NaN
1981-04-30         12.356667        NaN
1981-05-31          9.490323        NaN
```

Type here to search 12°C ENG 19:48 18-11-2025

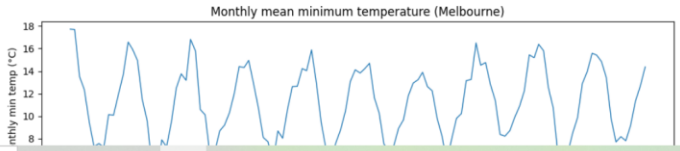
localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38

jupyter **Untitled47** Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3.8 (py38)

In [43]:
```
# 13.3(a) Two stacked line graphs:
# response 1 = monthly_min_temp, response 2 = trend_12m

fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

# Top: raw monthly min temp
axes[0].plot(
    monthly_ts.index,
    monthly_ts["monthly_min_temp"],
    linewidth=1.0
)
axes[0].set_ylabel("Monthly min temp (°C)")
axes[0].set_title("Monthly mean minimum temperature (Melbourne)")

# Bottom: 12-month rolling trend
axes[1].plot(
    monthly_ts.index,
    monthly_ts["trend_12m"],
    linewidth=1.5
)
axes[1].set_ylabel("12-month trend (°C)")
axes[1].set_xlabel("Year")
axes[1].set_title("Smoothed 12-month trend of minimum temperature")

plt.tight_layout()
plt.show()
```



Type here to search 12°C ENG 19:49 18-11-2025

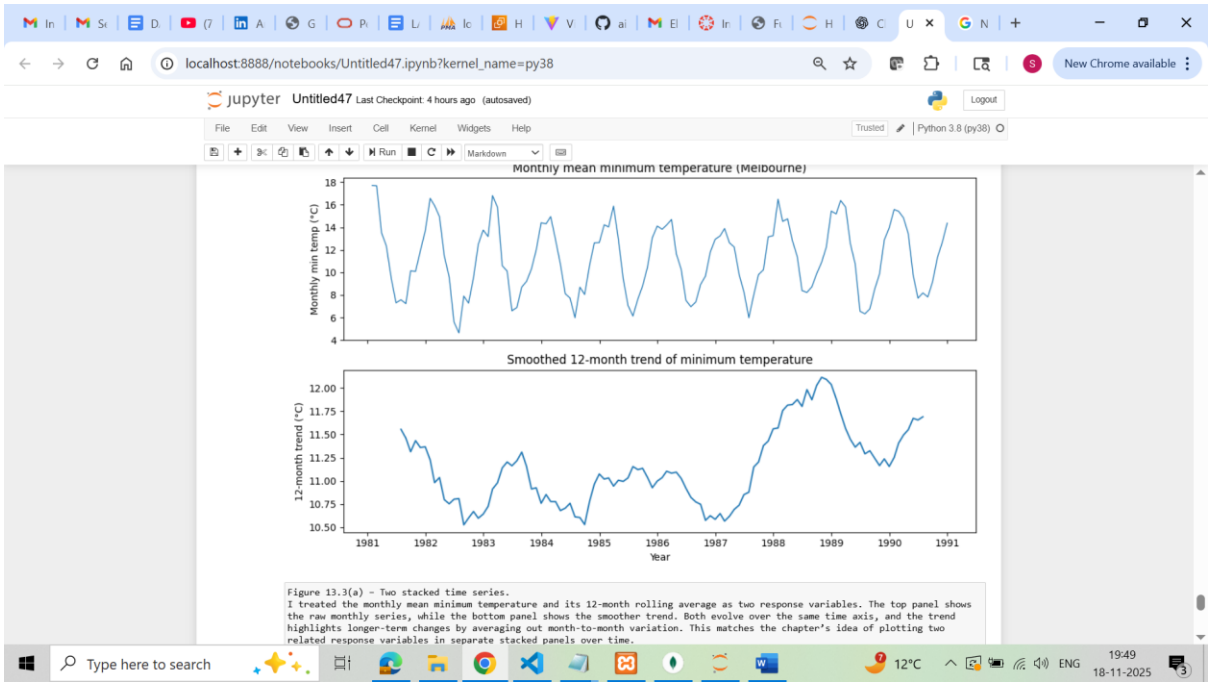
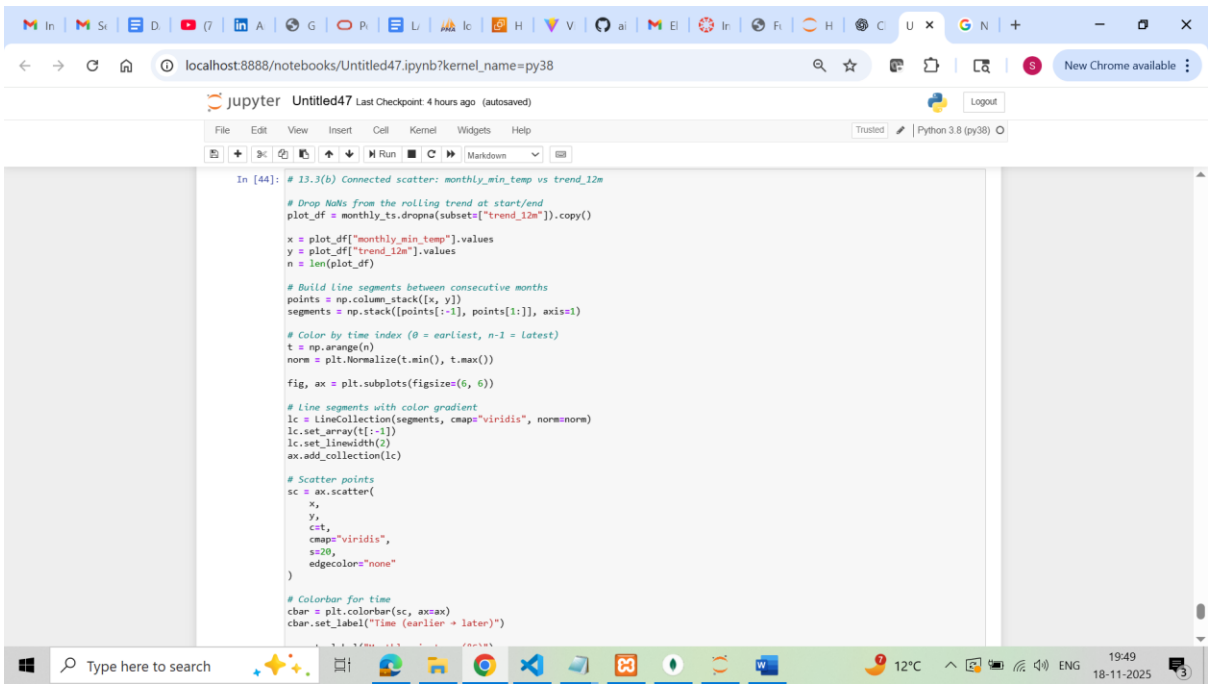
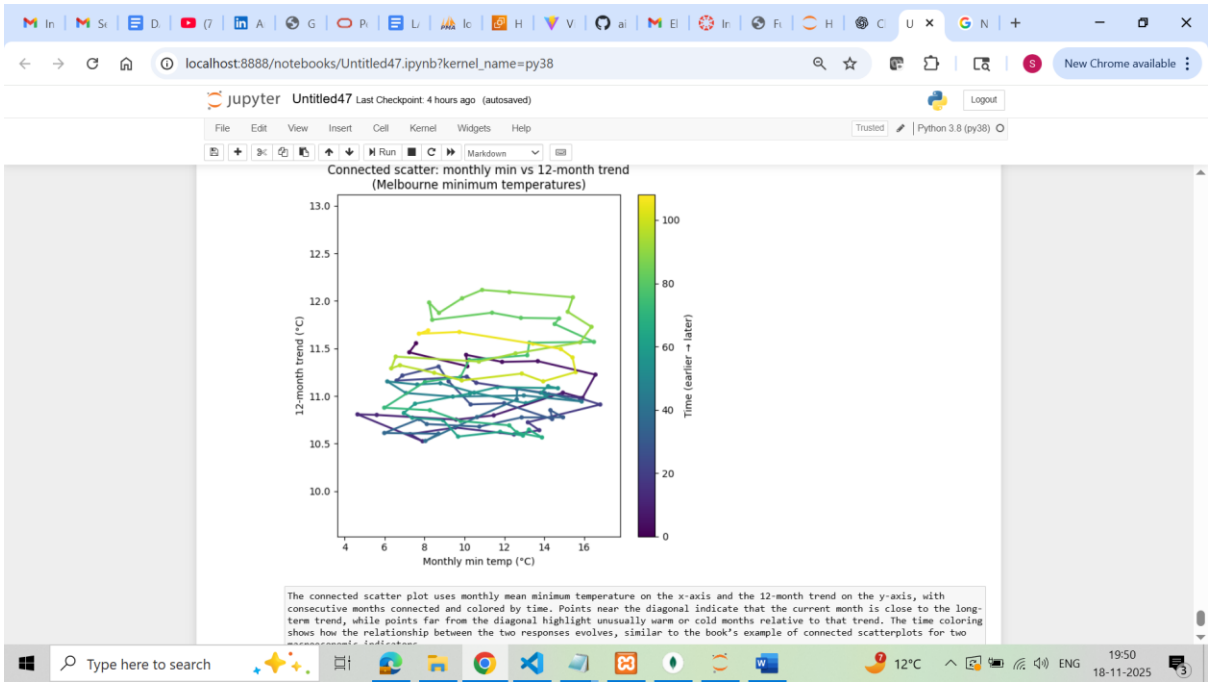


Figure 13.3(a) – Two stacked time series.
I treated the monthly mean minimum temperature and its 12-month rolling average as two response variables. The top panel shows the raw monthly series, while the bottom panel shows the smoother trend. Both evolve over the same time axis, and the trend highlights longer-term changes by averaging out month-to-month variation. This matches the chapter's idea of plotting two related response variables in separate stacked panels over time.



```
In [44]: # 13.3(b) Connected scatter: monthly_min_temp vs trend_12m

# Drop NaNs from the rolling trend at start/end
plot_df = monthly_ts.dropna(subset=["trend_12m"]).copy()

x = plot_df["monthly_min_temp"].values
y = plot_df["trend_12m"].values
n = len(plot_df)

# Build line segments between consecutive months
points = np.column_stack([x, y])
segments = np.stack([points[:-1], points[1:]], axis=1)

# Color by time index (0 = earliest, n-1 = latest)
t = np.arange(n)
norm = plt.Normalize(t.min(), t.max())

fig, ax = plt.subplots(figsize=(6, 6))

# Line segments with color gradient
lc = LineCollection(segments, cmap="viridis", norm=norm)
lc.set_array(t[:-1])
lc.set_linewidth(2)
ax.add_collection(lc)

# Scatter points
sc = ax.scatter(
    x,
    y,
    c=t,
    cmap="viridis",
    s=20,
    edgecolor="none"
)

# Colorbar for time
cbar = plt.colorbar(sc, ax=ax)
cbar.set_label("Time (earlier → later)")
```


localhost:8888/notebooks/Untitled47.ipynb?kernel_name=py38
jupyter Untitled47 Last Checkpoint: 4 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Connected scatter: monthly min vs 12-month trend
(Melbourne minimum temperatures)
12-month trend (°C)
Monthly min temp (°C)
Time (earlier → later)
The connected scatter plot uses monthly mean minimum temperature on the x-axis and the 12-month trend on the y-axis, with consecutive months connected and colored by time. Points near the diagonal indicate that the current month is close to the long-term trend, while points far from the diagonal highlight unusually warm or cold months relative to that trend. The time coloring shows how the relationship between the two responses evolves, similar to the book's example of connected scatterplots for two
Type here to search
12°C
ENG
19:50
18-11-2025