Setup

```
# pip install pandas numpy openpyxl statsmodels
import pandas as pd, numpy as np
from openpyxl import Workbook
from openpyxl.utils import get_column_letter
from openpyxl.chart import ScatterChart, Reference,
Series
import statsmodels.api as sm
```

How to use: Place your data files in the same folder as your script, copy a snippet into a .py file, edit input paths as needed, and run: python your_script.py.

Contents

1) Core EDA Summary to Excel

Read a file, compute types, missing, sample rows, and describe() and export to Excel.

```
# --- Core EDA Summary to Excel ---
import pandas as pd
```

```
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "YOUR_DATA.xlsx"   # or .csv
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)

dtypes = pd.DataFrame("column": df.columns, "dtype":
df.dtypes.astype(str))
missing =
df.isna().mean().mul(100).round(2).rename("missing_pct").
reset_index().rename(columns="index":"column")
sample = df.head(10)
describe = df.describe(include="all",
datetime_is_numeric=True).T.reset_index().rename(columns=
"index":"column")

wb = Workbook(); wb.remove(wb.active)
def to_sheet(name, frame):
    ws = wb.create_sheet(name)
    ws.append(list(frame.columns))
    for r in frame.itertuples(index=False):
        ws.append(list(r))
    for j in range(1, frame.shape[1]+1):
        ws.column_dimensions[get_column_letter(j)].width
= 18

to_sheet("dtypes", dtypes)
to_sheet("missing_pct", missing)
to_sheet("sample_head", sample)
to_sheet("describe", describe)
wb.save("EDA_Output.xlsx")



2) Columns with >5 Missing (answer cell + detail)
# --- Columns with >5% Missing ---
```

```
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "YOUR_DATA.xlsx"
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)
missing = df.isna().mean().mul(100)
detail =
missing.round(2).rename("missing_pct").reset_index().rena
me(columns="index":"column")
count_gt5 = int((missing > 5).sum())

wb = Workbook(); ws = wb.active; ws.title = "Summary"
ws.append(["Columns > 5% Missing", count_gt5])
for j in range(1, 3):
ws.column_dimensions[get_column_letter(j)].width = 24
ws2 = wb.create_sheet("Detail");
ws2.append(list(detail.columns))
for r in detail.itertuples(index=False):
ws2.append(list(r))
for j in range(1, detail.shape[1]+1):
ws2.column_dimensions[get_column_letter(j)].width = 18
wb.save("Missing_Over5pct.xlsx")
```

3) Numeric Profiling (skew, kurtosis, outliers)

```
# --- Numeric Profiling: skew/kurtosis and outlier flags
(IQR rule) ---
import pandas as pd, numpy as np
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "YOUR_DATA.xlsx"
```

```
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)
num = df.select_dtypes(include="number").copy()
prof = pd.DataFrame(
    "col": num.columns,
    "mean": num.mean().values,
    "std": num.std(ddof=1).values,
    "skew": num.skew().values,
    "kurt": num.kurt().values,
)

# Outlier count by IQR
def iqr_outliers(s):
    q1, q3 = s.quantile(0.25), s.quantile(0.75)
    iqr = q3 - q1
    lo, hi = q1 - 1.5*iqr, q3 + 1.5*iqr
    return int(((s<lo) | (s>hi)).sum())

prof["outliers_IQR"] = [iqr_outliers(num[c]) for c in
num.columns]

wb = Workbook(); ws = wb.active; ws.title =
"Numeric_Profile"
ws.append(list(prof.columns))
for r in prof.itertuples(index=False): ws.append(list(r))
for j in range(1, prof.shape[1]+1):
ws.column_dimensions[get_column_letter(j)].width = 16
wb.save("Numeric_Profile.xlsx")



4) Correlation Matrix to Excel (optional scatter pairs)
# --- Correlation Matrix to Excel ---
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter
```

```
infile = "YOUR_DATA.xlsx"
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)
num = df.select_dtypes(include="number")
corr = num.corr().round(3)

wb = Workbook(); wb.remove(wb.active)
ws = wb.create_sheet("Correlation")
ws.append([""] + list(corr.columns))
for rname, row in corr.iterrows():
    ws.append([rname] + list(row.values))
for j in range(1, corr.shape[1]+2):
ws.column_dimensions[get_column_letter(j)].width = 16
wb.save("Correlation_Matrix.xlsx")
```

5) Text Wrangling: Split and Combine Columns

```
# --- Split 'FullName' into 'First','Last' and combine
back ---
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "YOUR_DATA.xlsx"
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)

if "FullName" in df.columns:
    sp = df["FullName"].astype(str).str.split("\\s+",
n=1, expand=True)
    df["First"] = sp[0]
    df["Last"] = sp[1] if sp.shape[1]>1 else ""
    df["Recombined"] = df["First"].fillna("") + " " +
df["Last"].fillna("")
```

```
df.to_excel("Text_Wrangling_Output.xlsx", index=False)
```

6) Stack “long” →nstack “wide” (and back)

```
# --- Long to Wide (pivot) and back ---
import pandas as pd

infile = "YOUR_STACKED.xlsx"  # columns like: Key,
SubKey, Value
df = pd.read_excel(infile)
wide = df.pivot_table(index="Key", columns="SubKey",
values="Value", aggfunc="first").reset_index()
wide.to_excel("Unstacked_Wide.xlsx", index=False)
# Back to long
long2 = wide.melt(id_vars=["Key"], var_name="SubKey",
value_name="Value")
long2.to_excel("Restacked_Long.xlsx", index=False)
```

7) Pivot/Groupby Summaries (Store rand KPIs)

```
# --- Groupby: Avg profit by store; totals by brand ---
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "Tissue_CleanedBrands.xlsx"
df = pd.read_excel(infile)

by_store = df.groupby("Store",
dropna=False)["ProfitPerCase"].mean().reset_index().renam
e(columns="ProfitPerCase":"AvgProfitPerCase")
by_brand_pkgs = df.groupby("Brand",
dropna=False)["PackagesSold"].sum().reset_index().rename(
columns="PackagesSold":"TotalPackages")
```

```
by_brand_rev = df.groupby("Brand",
dropna=False)["Revenue"].sum().reset_index().rename(colum
ns="Revenue":"TotalRevenue")

wb = Workbook(); wb.remove(wb.active)
def to_sheet(name, frame):
    ws = wb.create_sheet(name);
ws.append(list(frame.columns))
    for r in frame.itertuples(index=False):
ws.append(list(r))
    for j in range(1, frame.shape[1]+1):
ws.column_dimensions[get_column_letter(j)].width = 18

to_sheet("AvgProfit_ByStore", by_store)
to_sheet("Brand_TotalPackages", by_brand_pkgs)
to_sheet("Brand_TotalRevenue", by_brand_rev)
wb.save("Tissue_Summaries.xlsx")



8) One-Way What-If Table Writer (Sensitivity)
# --- One-way sensitivity table: Profit vs Demand ---
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

fixed = 160000; var_cost = 6; price = 46
demands = list(range(1000, 6001, 200))
rows = []
for d in demands:
    revenue = price*d
    cost = fixed + var_cost*d
    profit = revenue - cost
    rows.append([d, revenue, cost, profit])

df = pd.DataFrame(rows,
columns=["Demand","Revenue","Cost","Profit"])
```

```
wb = Workbook(); ws = wb.active; ws.title =
"OneWay_Sensitivity"
ws.append(list(df.columns))
for r in df.itertuples(index=False): ws.append(list(r))
for j in range(1, df.shape[1]+1):
ws.column_dimensions[get_column_letter(j)].width = 16
wb.save("OneWay_WhatIf.xlsx")
```

9) Two-Way What-If Table Writer (Grid)

```
# --- Two-way table: Profit vs Demand 7  Price ---
import pandas as pd, numpy as np
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

fixed = 160000; var_cost = 6
demands = list(range(2000, 6001, 500))
prices = [35, 40, 45, 50, 55]

grid = []
for d in demands:
    row = [d]
    for p in prices:
        profit = p*d - (fixed + var_cost*d)
        row.append(profit)
    grid.append(row)

cols = ["Demand"] + [f"Price_p" for p in prices]
df = pd.DataFrame(grid, columns=cols)

wb = Workbook(); ws = wb.active; ws.title =
"TwoWay_Sensitivity"
ws.append(cols)
for r in df.itertuples(index=False): ws.append(list(r))
for j in range(1, len(cols)+1):
ws.column_dimensions[get_column_letter(j)].width = 16
```

```
wb.save("TwoWay_WhatIf.xlsx")
```

10) Data Dictionary Generator to Excel

```
# --- Data Dictionary: column name, dtype, sample,
missing% ---
import pandas as pd
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

infile = "YOUR_DATA.xlsx"
df = pd.read_excel(infile) if
infile.lower().endswith((".xlsx",".xls")) else
pd.read_csv(infile)

rows = []
for c in df.columns:
    dtype = str(df[c].dtype)
    miss = float(df[c].isna().mean()*100)
    sample = df[c].dropna().iloc[0] if
df[c].notna().any() else ""
    rows.append([c, dtype, round(miss,2),
str(sample)[:80]])

dd = pd.DataFrame(rows,
columns=["column","dtype","missing_pct","sample_value"])

wb = Workbook(); ws = wb.active; ws.title =
"DataDictionary"
ws.append(list(dd.columns))
for r in dd.itertuples(index=False): ws.append(list(r))
for j in range(1, dd.shape[1]+1):
ws.column_dimensions[get_column_letter(j)].width = 22
wb.save("Data_Dictionary.xlsx")
```

```
This collection covers the most common Module 1â€“2 tasks
and exports results as Excel workbooks using openpyxl.
Adjust paths, column names, and parameters as needed.
```

EDA → Excel (matches your Module 2 sheets)

```
def write_sheet(ws, df):
    ws.append(list(df.columns))
    for r in df.itertuples(index=False):
        ws.append(list(r))
    for j in range(1, df.shape[1]+1):
        ws.column_dimensions[get_column_letter(j)].width
= 18


def eda_to_excel(infile, outfile="EDA_Output.xlsx"):
    df = pd.read_excel(infile) if
infile.endswith((".xlsx",".xls")) else
pd.read_csv(infile)
    dtypes = pd.DataFrame({"column": df.columns, "dtype":
df.dtypes.astype(str)})
    missing =
df.isna().mean().mul(100).round(2).rename("missing_pct").
reset_index().rename(columns={"index":"column"})
    sample = df.head(10)
    describe = df.describe(include="all",
datetime_is_numeric=True).T.reset_index().rename(columns=
{"index":"column"})


    wb = Workbook(); wb.remove(wb.active)
    ws = wb.create_sheet("dtypes");   write_sheet(ws,
dtypes)
    ws = wb.create_sheet("missing");  write_sheet(ws,
missing)
```

```
    ws = wb.create_sheet("sample");   write_sheet(ws,
sample)
    ws = wb.create_sheet("describe"); write_sheet(ws,
describe)
    wb.save(outfile)

# EXAMPLE: eda_to_excel("Housing_Data.xlsx")  # or any
exam dataset
```

Simple Linear Regression → Excel (e.g., Bus Age → AnnualCost)

```
def simple_lr_to_excel(infile, ycol, xcol,
outfile="SLR_Output.xlsx", sheet="SLR"):
    df = pd.read_excel(infile).dropna(subset=[ycol,
xcol])
    X = sm.add_constant(df[[xcol]].astype(float))
    y = df[ycol].astype(float)
    m = sm.OLS(y, X).fit()

    # Excel book
    wb = Workbook(); ws = wb.active; ws.title = sheet
    ws.append(["Term","Coef","StdErr","t","pval"])
    ws.append(["Intercept", float(m.params['const']),
float(m.bse['const']), float(m.tvalues['const']),
float(m.pvalues['const'])])
    ws.append([xcol, float(m.params[xcol]),
float(m.bse[xcol]), float(m.tvalues[xcol]),
float(m.pvalues[xcol])])
    ws.append([]); ws.append(["R2", float(m.rsquared)])

    # predictions & residuals
    pred = m.predict(X)
    resid = y - pred
    ws.append([]); ws.append(["i","y","yhat","resid"])
```

```
    start = ws.max_row+1
    for i,(ya,yp,re) in enumerate(zip(y.values,
pred.values, resid.values)):
        ws.append([i, float(ya), float(yp), float(re)])

    # chart
    chart = ScatterChart(); chart.title="Actual vs
Predicted"; chart.x_axis.title="Predicted";
chart.y_axis.title="Actual"
    n = len(y)
    xref = Reference(ws, min_col=3, min_row=start+1,
max_row=start+n)
    yref = Reference(ws, min_col=2, min_row=start+1,
max_row=start+n)
    chart.series.append(Series(yref, xref, title="y"))
    ws.add_chart(chart, "H5")
    wb.save(outfile)

# EXAMPLE: simple_lr_to_excel("busagecost.xlsx",
ycol="AnnualCost", xcol="Age", outfile="Bus_SLR.xlsx")
```

Multiple Regression + Dummies → Excel (e.g., Fueldata, Hotels)

```
def mlr_with_dummies_to_excel(infile, ycol, num_cols,
cat_cols, drop_first=True,
                              outfile="MLR_Output.xlsx",
sheet="MLR"):
    df = pd.read_excel(infile).copy()
    # build dummies
    X = pd.get_dummies(df[num_cols + cat_cols],
columns=cat_cols, drop_first=drop_first)
    data = pd.concat([df[ycol], X], axis=1).dropna()
    y = data[ycol].astype(float)
```

```
    X =
sm.add_constant(data.drop(columns=[ycol]).astype(float))
    m = sm.OLS(y, X).fit()

    wb = Workbook(); ws = wb.active; ws.title = sheet
    ws.append(["Term","Coef","StdErr","t","pval"])
    for term in m.params.index:
        ws.append([term, float(m.params[term]),
float(m.bse[term]), float(m.tvalues[term]),
float(m.pvalues[term])])
    ws.append([]); ws.append(["R2", float(m.rsquared)])
    wb.save(outfile)
    return m, X.columns.tolist()

# EXAMPLES:
# Fuel MPG (~ displacement + class dummies + fuel
premium)
# m, cols = mlr_with_dummies_to_excel("fueldata.xlsx",
#     ycol="HwyMPG",
#     num_cols=["Displacement"],
#     cat_cols=["Class","FuelType"],
#     outfile="Fuel_MLR.xlsx",
#     sheet="Fuel_MLR")

# Hotels Overall (~ Comfort + Amenities + InHouseDining)
[no dummies]
# m, cols =
mlr_with_dummies_to_excel("beachfronthotels.xlsx",
#     ycol="Overall",
#     num_cols=["Comfort","Amenities","InHouseDining"],
#     cat_cols=[],  # none
#     outfile="Hotels_MLR.xlsx",
#     sheet="Hotels_MLR")
```

Prediction from the fitted multiple regression

```
def mlr_predict(m, colnames, values_dict):
    # values_dict must include 'const':1 and every
regressor column from colnames
    x = np.array([values_dict.get(c, 0.0) for c in
colnames], dtype=float)
    return float(np.dot(x, m.params[colnames]))

# EXAMPLE for fuel model:
# Suppose base had drop_first=True so Class_Midsize and
Class_Large appear; compact is base
# colnames might look like:
['const','Displacement','Class_Large','Class_Midsize','Fu
elType_P']
# pred = mlr_predict(m, colnames, {"const":1,
"Displacement":3.5, "Class_Midsize":1, "Class_Large":0,
"FuelType_P":1})
# print(round(pred,2))
```

Quadratic Regression (e.g., corporate bonds yield ~ years + years^2)

```
def quadratic_lr_to_excel(infile, ycol, xcol,
outfile="Quadratic_Output.xlsx", sheet="Quad"):
    df = pd.read_excel(infile).dropna(subset=[ycol,
xcol])
    df["x2"] = df[xcol]**2
    X = sm.add_constant(df[[xcol,"x2"]].astype(float))
    y = df[ycol].astype(float)
    m = sm.OLS(y, X).fit()

    wb = Workbook(); ws = wb.active; ws.title = sheet
    ws.append(["Term","Coef","StdErr","t","pval"])
    for term in m.params.index:
```

```
        ws.append([term, float(m.params[term]),
float(m.bse[term]), float(m.tvalues[term]),
float(m.pvalues[term])])
    ws.append([]); ws.append(["R2", float(m.rsquared)])
    wb.save(outfile)
    return m

# EXAMPLE: quadratic_lr_to_excel("corporatebonds.xlsx",
ycol="Yield", xcol="Years")
```

Manual-numbers → Excel `def manual_line_to_excel(b0, b1, predict_x=None, outfile="Manual_SLR.xlsx"):`

```
    wb = Workbook(); ws = wb.active;
ws.title="Manual_SLR"
    ws.append(["Term","Coef"]); ws.append(["Intercept",
b0]); ws.append(["Slope", b1])
    if predict_x is not None:
        ws.append([]); ws.append(["Predict at x",
predict_x]); ws.append(["yhat", b0 + b1*predict_x])
    wb.save(outfile)

# EXAMPLE: manual_line_to_excel(1200.0, 480.5,
predict_x=3.5)
```

Chapter 8, Problem 15 (Fueldata – Multiple Regression with Dummy Variables)

Python (save as `fuel_regression_to_excel.py` and run)

```
import pandas as pd, numpy as np
from openpyxl import Workbook
from openpyxl.utils import get_column_letter
```

```
from openpyxl.chart import ScatterChart, Reference,
Series

# ---------- Load data ----------
path = "fueldata.xlsx"                  # put your file
next to this script
df = pd.read_excel(path)

# ---------- Build dummies ----------
df["Class"] = df["Class"].astype(str)
df["FuelType"] = df["FuelType"].astype(str)
df["ClassMidsize"] =
(df["Class"].str.lower()=="midsize").astype(int)
df["ClassLarge"]   =
(df["Class"].str.lower()=="large").astype(int)
df["FuelPremium"]  =
(df["FuelType"].str.upper()=="P").astype(int)

need =
["HwyMPG","Displacement","ClassMidsize","ClassLarge","Fue
lPremium"]
df = df.dropna(subset=need)

y = df[["HwyMPG"]].to_numpy(float)

def fit_ols(X, y):
    X = np.asarray(X, float)
    n, k = X.shape
    X_i = np.c_[np.ones((n,1)), X]
    XtX = X_i.T @ X_i
    try: XtX_inv = np.linalg.inv(XtX)
    except np.linalg.LinAlgError: XtX_inv =
np.linalg.pinv(XtX)
    beta = XtX_inv @ (X_i.T @ y)
    yhat = X_i @ beta
    resid = y - yhat
    ss_res = float((resid.T @ resid).ravel())
```

```
    ss_tot = float(((y - y.mean())**2).sum())
    dof = max(n - (k+1), 1)
    s2 = ss_res / dof
    se = np.sqrt(np.diag(s2 * XtX_inv)).reshape(-1,1)
    tstats = (beta / se).ravel()
    # p-values if SciPy present; else NaN
    try:
        from scipy.stats import t as student_t
        pvals = 2*(1 - student_t.cdf(np.abs(tstats),
df=dof))
    except Exception:
        pvals = np.full_like(tstats, np.nan, dtype=float)
    r2 = 1 - ss_res/ss_tot if ss_tot>0 else np.nan
    return dict(beta=beta.ravel(), yhat=yhat.ravel(),
resid=resid.ravel(),
                r2=float(r2), se=se.ravel(), t=tstats,
p=pvals, dof=int(dof))

def write_model(ws, model_name, target, features, res):
    ws.append([model_name, "", "Target", target])
    ws.append(["n", len(res["yhat"]), "k (features)",
len(features), "R^2", res["r2"], "df", res["dof"]])
    ws.append([])

ws.append(["Term","Coefficient","StdErr","t-Stat","p-Valu
e"])
    terms = ["Intercept"] + features
    for i,term in enumerate(terms):
        ws.append([term, float(res["beta"][i]),
float(res["se"][i]), float(res["t"][i]),
float(res["p"][i])])
    ws.append([])
    ws.append(["Index","y_actual","y_pred","Residual"])
    start = ws.max_row+1
    for i,(ya,yp,re) in enumerate(zip(y.ravel(),
res["yhat"], res["resid"])):
        ws.append([i, float(ya), float(yp), float(re)])
```

```
    for j in range(1, 8):
        ws.column_dimensions[get_column_letter(j)].width
= 18
    # Chart: Actual vs Predicted
    chart = ScatterChart()
    chart.title=f"{model_name}: Actual vs Predicted"
    chart.x_axis.title="Predicted";
chart.y_axis.title="Actual"
    n = len(res["yhat"])
    xref = Reference(ws, min_col=3, min_row=start+1,
max_row=start+n)  # y_pred
    yref = Reference(ws, min_col=2, min_row=start+1,
max_row=start+n)  # y_actual
    chart.series.append(Series(yref, xref,
title="HwyMPG"))
    ws.add_chart(chart, "H5")

# ---------- Fit models ----------
# (a)
Xa = df[["Displacement"]].to_numpy(float)
resa = fit_ols(Xa, y)
# (c)
Xc =
df[["Displacement","ClassMidsize","ClassLarge"]].to_numpy
(float)
resc = fit_ols(Xc, y)
# (e)
Xe =
df[["Displacement","ClassMidsize","ClassLarge","FuelPremi
um"]].to_numpy(float)
rese = fit_ols(Xe, y)

# ---------- Write to Excel ----------
wb = Workbook(); wb.remove(wb.active)
wsa = wb.create_sheet("Part_a_SLR"); write_model(wsa,
"Part (a) HwyMPG ~ Displacement", "HwyMPG",
["Displacement"], resa)
```

```
wsc = wb.create_sheet("Part_c_with_Class");
write_model(wsc, "Part (c) + Class Dummies", "HwyMPG",
["Displacement","ClassMidsize","ClassLarge"], resc)
wse = wb.create_sheet("Part_e_Class_Fuel");
write_model(wse, "Part (e) + Class + FuelPremium",
"HwyMPG",
["Displacement","ClassMidsize","ClassLarge","FuelPremium"
], rese)

# (g) Example prediction (edit here if your prompt
specifies different inputs)
wsg = wb.create_sheet("Part_g_Prediction")
wsg.append(["Model used", "Part (e) + Class +
FuelPremium"])
wsg.append(["Inputs", "Value"])
disp = 2.9; midsize=0; large=0; premium=1   # Compact
base, 2.9L, Premium fuel
wsg.append(["Displacement (L)", disp])
wsg.append(["ClassMidsize", midsize])
wsg.append(["ClassLarge", large])
wsg.append(["FuelPremium", premium])

b = rese["beta"]
xvec = np.array([1, disp, midsize, large, premium],
float)
yhat = float(np.dot(b, xvec))
wsg.append([]); wsg.append(["Predicted HwyMPG", yhat])

for s in wb.sheetnames:
    ws = wb[s]
    for j in range(1, 7):
        ws.column_dimensions[get_column_letter(j)].width
= 20

wb.save("Ch8_Problem15_FuelRegression.xlsx")
print("Wrote: Ch8_Problem15_FuelRegression.xlsx")
```

More Random Scripts

```
import os
import numpy as np
import pandas as pd
from openpyxl import Workbook, load_workbook
from openpyxl.utils import get_column_letter
from openpyxl.chart import ScatterChart, Reference, Series

# ---------- helpers ----------
def _read_any(path):
    low = path.lower()
    if low.endswith((".xlsx", ".xls")):
        return pd.read_excel(path)
    elif low.endswith(".csv"):
        return pd.read_csv(path)
    else:
        raise ValueError("Unsupported file type. Use .xlsx, .xls, or .csv")

def _to_sheet(ws, frame: pd.DataFrame, autosize=True):
    ws.append(list(frame.columns))
    for row in frame.itertuples(index=False):
        ws.append(list(row))
    if autosize:
        for j in range(1, frame.shape[1]+1):
            ws.column_dimensions[get_column_letter(j)].width = 18

def _safe_float(x):
    try: return float(x)
    except Exception: return None

# ---------- 1) Housing EDA ----------
def build_housing_eda(input_path, output_path="Housing_EDA_Output.xlsx"):
```

```
    df = _read_any(input_path)
    # Coerce numerics where possible (non-destructive)
    co = df.copy()
    for c in co.columns:
        if not pd.api.types.is_numeric_dtype(co[c]):
            co[c] = pd.to_numeric(co[c], errors="ignore")

    dtypes = pd.DataFrame({"column": co.columns, "dtype": [str(t) for t in
co.dtypes]})
    miss =
co.isna().mean().mul(100).round(2).rename("missing_pct").reset_index().rename(
columns={"index":"column"})
    sample = co.head(10)
    desc = co.describe(include="all",
datetime_is_numeric=True).T.reset_index().rename(columns={"index":"column"})

    # summary answers often requested
    total_cols = co.shape[1]
    num_missing_gt5 = int((miss["missing_pct"] > 5).sum())

    wb = Workbook(); wb.remove(wb.active)
    ws0 = wb.create_sheet("Summary")
    ws0.append(["Metric","Value"])
    ws0.append(["Num Columns", total_cols])
    ws0.append([">5% Missing Columns", num_missing_gt5])
    for j in range(1, 3): ws0.column_dimensions[get_column_letter(j)].width = 24

    ws1 = wb.create_sheet("dtypes"); _to_sheet(ws1, dtypes)
    ws2 = wb.create_sheet("missing_pct"); _to_sheet(ws2, miss)
    ws3 = wb.create_sheet("sample_head"); _to_sheet(ws3, sample)
    ws4 = wb.create_sheet("describe"); _to_sheet(ws4, desc)

    wb.save(output_path)
    return output_path

# ---------- 2) Tissue Analysis ----------
```

```
def build_tissue_analysis(input_path, output_path="Tissue_Analysis.xlsx"):
    """
    Expects columns like: Store, Brand, ProfitPerCase, Revenue, PackagesSold,
Week (names may differ).
    We will try to infer typical names and fallback heuristically.
    """
    df = _read_any(input_path)

    # Heuristic column mapping
    cols = {c.lower(): c for c in df.columns}
    def pick(*names):
        for n in names:
            if n.lower() in cols: return cols[n.lower()]
        return None

    store_col   = pick("Store","store_name","StoreName","Outlet")
    brand_col   = pick("Brand","brand","BrandName")
    profit_col  = pick("ProfitPerCase","Profit per Case","AvgProfitPerCase","Profit")
    revenue_col = pick("Revenue","TotalRevenue","Sales")
    pkgs_col    = pick("PackagesSold","Units","Qty","Quantity","Packages")
    week_col    = pick("Week","WeekNum","WeekNumber")

    # Average profit per case by store
    by_store = None
    if store_col and profit_col:
        by_store = df.groupby(store_col,
dropna=False)[profit_col].mean().reset_index().rename(columns={profit_col:"Av
gProfitPerCase"})
    else:
        by_store = pd.DataFrame({"Note":["Store or ProfitPerCase column not
found"]})

    # Market leader by total packages
    market_leader = None
    if brand_col and pkgs_col:
```

```
        pk = df.groupby(brand_col,
dropna=False)[pkgs_col].sum().reset_index().rename(columns={pkgs_col:"Total
Packages"})
        pk_sorted = pk.sort_values("TotalPackages", ascending=False)
        market_leader = pk_sorted
    else:
        market_leader = pd.DataFrame({"Note":["Brand or PackagesSold column
not found"]})

    # Lowest total revenue by brand
    low_rev = None
    if brand_col and revenue_col:
        br = df.groupby(brand_col,
dropna=False)[revenue_col].sum().reset_index().rename(columns={revenue_col
:"TotalRevenue"})
        low_rev = br.sort_values("TotalRevenue", ascending=True)
    else:
        low_rev = pd.DataFrame({"Note":["Brand or Revenue column not found"]})

    # Weekly Scott anomaly check (if present)
    scott_week = pd.DataFrame({"Note":["Scott-week check unavailable (need
Brand, Week, Packages/Revenue)"]})
    if brand_col and week_col and (pkgs_col or revenue_col):
        is_scott = df[brand_col].astype(str).str.contains("scott", case=False,
na=False)
        if pkgs_col:
            tab = df[is_scott].groupby(week_col)[pkgs_col].sum().reset_index()
            scott_week = tab.rename(columns={pkgs_col:"Scott_TotalPackages"})
        elif revenue_col:
            tab = df[is_scott].groupby(week_col)[revenue_col].sum().reset_index()
            scott_week =
tab.rename(columns={revenue_col:"Scott_TotalRevenue"})

    wb = Workbook(); wb.remove(wb.active)
    ws1 = wb.create_sheet("AvgProfit_ByStore"); _to_sheet(ws1, by_store)
    ws2 = wb.create_sheet("MarketLeader_Packages"); _to_sheet(ws2,
market_leader)
```

```
    ws3 = wb.create_sheet("LowestRevenue_Brand"); _to_sheet(ws3, low_rev)
    ws4 = wb.create_sheet("Scott_ByWeek"); _to_sheet(ws4, scott_week)

    wb.save(output_path)
    return output_path

# ------------- 3) Bus SLR -------------
def build_bus_slr(input_path, output_path="Bus_SLR.xlsx",
                  y_col="AnnualCost", x_col="Age"):
    df = _read_any(input_path)
    df = df.dropna(subset=[y_col, x_col])
    x = df[[x_col]].to_numpy(float)
    y = df[[y_col]].to_numpy(float)

    X = np.c_[np.ones((len(x),1)), x]
    beta = np.linalg.pinv(X.T @ X) @ (X.T @ y)
    yhat = (X @ beta).ravel()
    resid = y.ravel() - yhat
    r2 = 1 - (resid @ resid) / (((y - y.mean())**2).sum())

    wb = Workbook(); wb.remove(wb.active)
    ws = wb.create_sheet("SLR")
    ws.append(["Term","Coefficient"])
    ws.append(["Intercept", float(beta[0])])
    ws.append([x_col,     float(beta[1])])
    ws.append([]); ws.append(["R^2", float(r2)])
    ws.append([]); ws.append(["Index","y","yhat","resid"])
    start = ws.max_row+1
    for i,(ya,yp,re) in enumerate(zip(y.ravel(), yhat, resid)):
        ws.append([i, float(ya), float(yp), float(re)])

    # Chart
    chart = ScatterChart(); chart.title="Actual vs Predicted";
chart.x_axis.title="Predicted"; chart.y_axis.title="Actual"
    n = len(yhat)
```

```
    xref = Reference(ws, min_col=3, min_row=start+1, max_row=start+n)
    yref = Reference(ws, min_col=2, min_row=start+1, max_row=start+n)
    chart.series.append(Series(yref, xref, title=y_col))
    ws.add_chart(chart, "H5")

    for j in range(1, 7): ws.column_dimensions[get_column_letter(j)].width = 18
    wb.save(output_path)
    return output_path

# ---------- 4) Beachfront MLR ----------
def build_beachfront_mlr(input_path, output_path="Beachfront_MLR.xlsx",
                         y_col="Overall",
X_cols=("Comfort","Amenities","InHouseDining")):
    df = _read_any(input_path)
    df = df.dropna(subset=[y_col] + list(X_cols))
    Y = df[[y_col]].to_numpy(float)
    X = np.c_[np.ones((len(df),1)), df[list(X_cols)].to_numpy(float)]

    # OLS
    XtX = X.T @ X
    try:
        XtX_inv = np.linalg.inv(XtX)
    except np.linalg.LinAlgError:
        XtX_inv = np.linalg.pinv(XtX)
    beta = XtX_inv @ (X.T @ Y)
    yhat = (X @ beta).ravel()
    resid = Y.ravel() - yhat
    n = len(Y); k = len(X_cols)
    ss_res = float(resid @ resid)
    ss_tot = float(((Y - Y.mean())**2).sum())
    r2 = 1 - ss_res/ss_tot if ss_tot>0 else np.nan
    dof = max(n - (k+1), 1)
    s2 = ss_res / dof
    se = np.sqrt(np.diag(s2 * XtX_inv))
```

```
    tstats = beta.ravel()/se
    try:
        from scipy.stats import t as student_t
        pvals = 2*(1 - student_t.cdf(np.abs(tstats), df=dof))
    except Exception:
        pvals = np.full_like(tstats, np.nan, dtype=float)

    # Write Excel
    wb = Workbook(); wb.remove(wb.active)
    ws = wb.create_sheet("Hotels_MLR")
    ws.append(["Model", f"{y_col} ~ " + " + ".join(X_cols)])
    ws.append(["n", n, "k", k, "R^2", float(r2), "df", int(dof)])
    ws.append([])
    ws.append(["Term","Coefficient","StdErr","t-Stat","p-Value"])
    terms = ["Intercept"] + list(X_cols)
    for i,term in enumerate(terms):
        ws.append([term, float(beta.ravel()[i]), float(se[i]), float(tstats[i]),
float(pvals[i])])

    ws.append([]); ws.append(["Index","y","yhat","resid"])
    for i,(ya,yp,re) in enumerate(zip(Y.ravel(), yhat, resid)):
        ws.append([i, float(ya), float(yp), float(re)])

    for j in range(1, 7): ws.column_dimensions[get_column_letter(j)].width = 20
    wb.save(output_path)
    return output_path

if __name__ == "__main__":
    # EXAMPLES (edit paths as needed, then run: python exam1_generators.py)
    # 1) Housing EDA
    # print(build_housing_eda("Housing_Data.xlsx",
"Housing_EDA_Output.xlsx"))
    # 2) Tissue
    # print(build_tissue_analysis("Tissue_CleanedBrands.xlsx",
"Tissue_Analysis.xlsx"))
    # 3) Bus SLR
```

```
    # print(build_bus_slr("busagecost.xlsx", "Bus_SLR.xlsx"))
    # 4) Beachfront MLR
    # print(build_beachfront_mlr("beachfronthotels.xlsx",
"Beachfront_MLR.xlsx"))
    pass
```