Task – 4

Topic – **Complex Data Munging & Statistical Modeling in Pandas**

Created by – Kirti Bala

Data- prep notebook (data_prep.ipynb) – structure + runnable codes with various subtitles

1) Top – level imports

 # Markdown: "Imports & helper functions"

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pathlib import Path

import json

# Ensure reproducibility where needed

np.random.seed(0)


# small helper to write an audit/log entry (append to a list then persist at end)

pipeline_audit = []
```

```
def audit(step, col=None, details=None):
    pipeline_audit.append({'step': step, 'col': col, 'details': details, 'ts': pd.Timestamp.utcnow().isoformat()}).
```

2) Loading Raw CSV (s) and an inspection:

 # Markdown: "Load raw CSV (safe options for mixed types). Inspect basic properties"

```
raw_path = Path('raw.csv')

df = pd.read_csv(raw_path, low_memory=False)

# low_memory=False helps avoid mixed-type columns

df.info(memory_usage='deep')

df.head()

df.describe(include='all')
```

missingness summary

missing_pct =
df.isna().mean().sort_values(ascending =
False)

missing_pct[missing_pct > 0].head(30)

audit("load_and_inspect",
details=f"{len(df)} rows, {len(df.columns)}
columns")

3) Initial type inference strategy (read as object sample)

Markdown: "If dataset is huge, sample to infer dtypes then cast full dataframe"

sample = pd.read_csv(raw_path,
nrows=5000, dtype=str)

Heuristic conversions from sample:

cand_num = [c for c in sample.columns if
sample[c].str.contains(na=False).mean() >
0.8]

cand_dt = [c for c in sample.columns if pd.to_datetime(sample[c], errors='coerce').notna().mean() > 0.6]

print("numeric candidates:", cand_num)

print("datetime candidates:", cand_dt)

audit("infer_dtypes", details={'num_candidates': cand_num, 'dt_candidates': cand_dt})

4) Robust numeric and datetime parsing (coerce errors)

 # Markdown: "Clean numeric columns (strip currency, percent), parse datetimes"

```
def parse_numeric(s):
    if s.dtype.name == 'category':
        s = s.astype(str)
```

```
    return
pd.to_numeric(s.astype(str).str.replace(r'[,\
$%]', '', regex=True).str.strip(),
errors='coerce')


for col in ['amount', 'price', 'volume']:   #
replace with your actual columns
    if col in df.columns:
        df[col] = parse_numeric(df[col])
        audit("parse_numeric", col,
details=f"parsed {col}")


# parse datetime columns
for col in ['timestamp', 'date']:   # replace
with actual
    if col in df.columns:
        df[col] = pd.to_datetime(df[col],
errors='coerce', utc=False)  # choose tz-
aware if needed
```

```
    audit("parse_datetime", col, details=f"parsed {col}")
```

5) Categorical dtypes & ordered categories

```
# Markdown: "Using Categorical dtype for memory & semantics; set explicit orders for ordinal variables"
if 'region' in df.columns:
    df['region'] = df['region'].astype('category')

# ordinal example
if 'risk' in df.columns:
    risk_order = ['low', 'medium', 'high']
    df['risk'] = pd.Categorical(df['risk'].str.lower().map(lambda x: x if x in risk_order else np.nan),
```

categories=risk_order, ordered=True)

audit("set_categorical", details="region, risk")

6) Missing- value strategy- group- aware imputation

Using the group medians for the numeric and the group- mode for categorical. Use groupby().transform for efficient vectorized fills.

```
# Markdown: "Numeric imputation: group median then global median fallback"

num_cols = ['score', 'amount']  # replace with real numeric features

for col in num_cols:

    if col in df.columns:

        # compute group median (example grouped by 'customer_id' if present)
```

```
    group_key = 'customer_id' if
'customer_id' in df.columns else None
    if group_key:
      group_med =
df.groupby(group_key)[col].transform('med
ian')
      df[f'{col}_imputed'] =
df[col].fillna(group_med)
      df[f'{col}_imputed'] =
df[f'{col}_imputed'].fillna(df[f'{col}_imputed'
].median())
      audit("impute_num_group_median",
col, details=f"grouped by {group_key}")
    else:
      df[f'{col}_imputed'] =
df[col].fillna(df[col].median())
      audit("impute_num_global_median",
col)
```

```
# Markdown: "Categorical imputation: group-mode using custom apply"
def group_mode_map(group, col):
   m = group[col].mode()
   return m.iloc[0] if not m.empty else np.nan


if 'product_category' in df.columns:
   if 'customer_segment' in df.columns:
      # build mapping per segment
      mode_map = df.groupby('customer_segment')['product_category'].agg(lambda s: s.mode().iloc[0] if not s.mode().empty else np.nan)
      df['product_category'] = df['product_category'].fillna(df['customer_segment'].map(mode_map))
   df['product_category'] = df['product_category'].fillna('UNKNOWN')
```

df['product_category'] = df['product_category'].astype('category')

audit("impute_cat_group_mode", "product_category")

7) Outlier detection and handling

Markdown: "Mark outliers per group using IQR and optionally winsorize or flag"

```
def mark_iqr_outliers(s):
    q1 = s.quantile(0.25)
    q3 = s.quantile(0.75)
    iqr = q3 - q1
    if pd.isna(iqr) or iqr == 0:
        return pd.Series(False, index=s.index)
    return ~s.between(q1 - 1.5*iqr, q3 + 1.5*iqr)
```

```
if 'amount' in df.columns:
    group_key = 'region' if 'region' in
df.columns else None
    if group_key:
        df['is_outlier_amount'] =
df.groupby(group_key)['amount'].transform
(lambda s: mark_iqr_outliers(s))
    else:
        df['is_outlier_amount'] =
mark_iqr_outliers(df['amount'])

    # Example handling: create winsorized
column (cap at 1st/99th percentiles)
    lower = df['amount'].quantile(0.01)
    upper = df['amount'].quantile(0.99)
    df['amount_winsor'] =
df['amount'].clip(lower, upper)
```

audit("outlier_detection", "amount", details="IQR flag + winsorized")

8) Schema normalization: pivoting and multi- index

Markdown: "If data is long (measurements per row), pivot to wide"

Example: long table with columns ['id', 'date', 'metric', 'value']

if set(['id','date','metric','value']).issubset(df.columns):

df_wide = df.pivot_table(index=['id','date'], columns='metric', values='value', aggfunc='first')

df_wide.columns = [str(c) for c in df_wide.columns] # flatten df_wide = df_wide.reset_index().set_index(['id','date']).sort_index()

continue cleaning on df_wide...

audit("pivot_to_wide", details=f"pivoted metrics into columns; shape {df_wide.shape}")

Multi-index examples

"Set a multi-index for time-series operations"

if set(['entity_id','date']).issubset(df.columns):

df = df.set_index(['entity_id','date']).sort_index()

audit("set_multiindex", details="entity_id/date index set")

to go back:

df.reset_index(inplace=True)

9) Time-series alignment & merge

Merge_ asof is extremely useful when **matching** events with the nearest measurnments.

Markdown: "Align irregular time-series with merge_asof (e.g., trades <- nearest prior quotes)"

Example assumes trades and quotes dataframes exist and have 'ticker' and 'time' cols

trades = trades.sort_values(['ticker','time']); quotes = quotes.sort_values(['ticker','time'])

merged = pd.merge_asof(trades, quotes, by='ticker', left_on='time', right_on='time', direction='backward', tolerance=pd.Timedelta('1s'))

Explanation: each trade takes the most recent quote at-or-before the trade time (within tolerance)

10) Group- based transforms, lags & rolling features

It is used to transform when you need an aligned series of same length; it is applied for group-level returns or custom aggregations.

```
# "Create lags, rolling means, and group-normalized features"
if set(['ticker','time','price']).issubset(df.columns):
 df = df.sort_values(['ticker','time'])
 df['price_lag1'] = df.groupby('ticker')['price'].shift(1)
 df['price_rolling_7'] = df.groupby('ticker')['price'].rolling(window)7
min_periods=1).mean().reset_index(level=0, drop=True)
```

df['price_roll_diff'] = df['price'] - df['price_rolling_7']

audit("lags_rolls", details="lag & rolling features created for price")

11) Engineering Feature (polynomial features, interactions, dummies) by using pandas only

```
# "Polynomial & interaction features using pandas"
num_feats = ['x1', 'x2']  # replace with your numeric columns
for f in num_feats:
    if f in df.columns:
        df[f'{f}_sq'] = df[f] ** 2
        df[f'{f}_cube'] = df[f] ** 3
# interactions
if set(['x1','x2']).issubset(df.columns):
    df['x1_x2'] = df['x1'] * df['x2']
```

One-hot dummies (model-ready)

cat_cols = ['product_category']

df = pd.get_dummies(df, columns=[c for c in cat_cols if c in df.columns], dummy_na=True, drop_first=False)

audit("feature_engineering", details="polynomial, interaction, dummies")

12) Final checks, dedupe, export cleaned dataset

"Final sanity checks: duplicates, NaN proportions, type checks"

e.g., check duplicates for key columns

if set(['id','date']).issubset(df.columns):

  dups = df.duplicated(subset=['id','date']).sum()

  print("duplicate id-date rows:", dups)

inspect remaining missingness

```
remaining_missing =
df.isna().mean().sort_values(ascending =
False)

print(remaining_missing.head(20))

audit("final_checks", details={'n_rows':
len(df), 'missing_summary':
remaining_missing.head(10).to_dict()})


# persist cleaned data & audit log

df.to_parquet('cleaned_data.parquet',
index=False)

df.to_csv('cleaned_data.csv', index=False)

with open('pipeline_audit.json','w') as
f:json.dump(pipeline_audit, f, indent=2)
```

Modeling notebook-(modelling.ipynb)

Imports & load cleaned data

 # "Imports & load cleaned data"

import pandas as pd

import numpy as np

import statsmodels.api as sm

import statsmodels.formula.api as smf

from stats models.stats.outliers_influence import variance_inflation_factor

from statsmodels.stats.diagnostic import het_breuschpagan

from statsmodels.stats.stattools

import durbin_watson

import matplotlib.pyplot as plt

df =
pd.read_parquet('cleaned_data.parquet')   # or read_csv

df.shape

df.head()

1) Define target and predictor set (drop Nas used by the model)

"Define y and X; choose features created during data_prep"

target = 'target'   # replace with actual target column

features = ['x1','x2','x1_sq','x1_x2','price_rolling_7','is_promo'] # example

Ensure all features present

features = [f for f in features if f in df.columns]

```
# drop rows with NA in model variables
model_df = df[[target] + features].dropna()
y = model_df[target].astype(float)
X = model_df[features]
X = sm.add_constant(X)   # add intercept
```

OLS regression and Summary

```
#"Fit OLS, print summary, get CI and p-values"
ols = sm.OLS(y, X).fit()
print(ols.summary())

# point estimates & CI
params = ols.params
```

```
conf_int = ols.conf_int(alpha=0.05)  # 95% CI
pvals = ols.pvalues

result_df = pd.DataFrame({'coef': params, 'ci_lower': conf_int[0], 'ci_upper': conf_int[1], 'pval': pvals})
result_df
```

Robust standard errors, clustered Ses and hypothesis

 # "Robust (HC3) standard errors"

```
ols_hc3 = ols.get_robustcov_results(cov_type='HC3')
print(ols_hc3.summary())
```

```
# Clustered SEs example (cluster by 'group_col' if available)
if 'group_col' in model_df.columns:
    ols_cluster = ols.get_robustcov_results(cov_type='cluster', groups=df.loc[model_df.index, 'group_col'])
    print(ols_cluster.summary())
    audit("clustered_se", details="clustered by group_col")

# Hypothesis tests: single & joint
print("t-test x1 = 0 ->", ols.t_test("x1 = 0"))
print("joint test x1 = 0, x2 = 0 ->", ols.f_test("x1 = 0, x2 = 0"))
```

Odds ratios examples

```
# "If binary target, use Logit and report odds ratios and CIs"
if df[target].dropna().nunique() == 2:
    y_bin = model_df[target].astype(int)
    logit = sm.Logit(y_bin, X).fit(disp=False)
    print(logit.summary())
    or_df = pd.DataFrame({
        'OR': np.exp(logit.params),
        'OR_ci_lower': np.exp(logit.conf_int()[0]),
        'OR_ci_upper': np.exp(logit.conf_int()[1]),
        'pval': logit.pvalues
    })
    or_df
```

Diagnostics:

If VIF> 5-10 , consider dropping / reducing correlated features or by using PCA

```
# Residuals vs fitted

resid = ols.resid

fitted = ols.fittedvalues

plt.figure()

plt.scatter(fitted, resid, alpha=0.4)

plt.axhline(0, color='black', linewidth=0.8)

plt.xlabel('Fitted'); plt.ylabel('Residuals'); plt.title('Residuals vs Fitted')


# Q-Q plot

sm.qqplot(resid, line='45', fit=True)
```

```
plt.title("Q-Q plot of residuals")

# Breusch-Pagan test for heteroskedasticity
bp_test = het_breuschpagan(resid, ols.model.exog)
# Returns (lm_stat, lm_pvalue, fvalue, f_pvalue)
bp_results = {'lm_stat': bp_test[0], 'lm_pvalue': bp_test[1], 'fvalue': bp_test[2], 'f_pvalue': bp_test[3]}
print("Breusch-Pagan:", bp_results)

# Durbin-Watson for autocorrelation (useful if time series)
print("Durbin-Watson:", durbin_watson(resid))
```

VIF for multicollinearity (exclude constant)

X_no_const = X.drop(columns='const', errors='ignore')

vif = pd.Series([variance_inflation_factor(X_no_const.values, i)

for i in range(X_no_const.shape[1])], index=X_no_const.columns)

print("VIF:\n", vif)

Prediction

"Prediction + intervals on new data"

X_new = X.iloc[:10] # replace with new observation(s)

pred = ols.get_prediction(X_new)

pred_df =
pred.summary_frame(alpha=0.05) # contains mean, mean_ci_lower/upper, obs_ci_lower/upper

pred_df.head()

Error checks & alternative specs

- ✓ Re-fit excluding extreme outliers.
- ✓ Re-fit using np.log() transform on skewed vars.
- ✓ Usemodel.get_robustcov_results (cov_type='HC3').
- ✓ If clustered data, use clustered SEs.