```
# Unemployment Rate Data Analysis - Complete Reproducible Code
# Save this entire block into a Jupyter cell (or multiple cells) and run.
# This script expects the uploaded zip to be at '/mnt/data/ziptask1.zip'
# It will:
#  - unzip and load CSV(s)
#  - clean data
#  - perform EDA, COVID impact analysis, seasonality decomposition
#  - save figures in /mnt/data/figures and generate simple report text
# NOTE: run this in a Python 3 environment with required packages installed:
# pandas, numpy, matplotlib, seaborn, scipy, statsmodels, openpyxl (optional), pyarrow (optiona
l)
# You can install missing packages with: pip install pandas numpy matplotlib seaborn scipy stat
smodels

# ---------- 1. Imports ----------
import zipfile
import io
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind
from statsmodels.tsa.seasonal import STL
import warnings
warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (10,5)

# ---------- 2. Paths & unzip ----------
DATA_ZIP = '/mnt/data/ziptask1.zip'   # <--- update if different
OUT_DIR = Path('/mnt/data/unemployment_analysis_output')
FIG_DIR = OUT_DIR / 'figures'
OUT_DIR.mkdir(parents=True, exist_ok=True)
FIG_DIR.mkdir(parents=True, exist_ok=True)

with zipfile.ZipFile(DATA_ZIP) as z:
    print("Files in zip:", z.namelist())
    # Try to find likely CSV files
    csv_files = [n for n in z.namelist() if n.lower().endswith('.csv')]
    if len(csv_files) == 0:
        raise FileNotFoundError("No CSV found inside the zip. Please ensure CSV is included.")
    # We'll load the first CSV by default
    name = csv_files[0]
    print("Loading:", name)
    df = pd.read_csv(z.open(name))

# ---------- 3. Initial inspection ----------
print("\n--- Initial dataframe info ---")
print(df.shape)
print(df.columns)
print(df.head(5))

# Standardize column names (lowercase, strip)
df.columns = [c.strip().lower() for c in df.columns]

# Identify likely date and unemployment columns
date_col = None
rate_col = None
for c in df.columns:
    if 'date' in c:
        date_col = c
    if 'unemploy' in c or 'unemployment' in c or 'unemployment_rate' in c or 'rate'==c:
        rate_col = c
# If not found heuristically, fallback to first two columns
if date_col is None:
    # try first column
    date_col = df.columns[0]
if rate_col is None:
    # try second column if exists
    if len(df.columns) > 1:
        rate_col = df.columns[1]
    else:
        raise ValueError("Could not find an unemployment rate column automatically. Please rena
me columns to include 'date' and 'unemployment_rate'.")
```

```
print(f"\nDetected date column: {date_col}")
print(f"Detected rate column: {rate_col}")

# ---------- 4. Parse dates & set index ----------
df[date_col] = pd.to_datetime(df[date_col], errors='coerce')
# If time portion present, you may want to normalize to period start/end
df = df.sort_values(by=date_col).reset_index(drop=True)
df = df.set_index(date_col)
print("\nAfter setting index:")
print(df.index.min(), df.index.max())

# ---------- 5. Convert rate to numeric ----------
df[rate_col] = pd.to_numeric(df[rate_col], errors='coerce')

# ---------- 6. Inspect missing values ----------
print("\nMissing values per column:")
print(df.isnull().sum())

# Document assumption: we'll interpolate unemployment rate over time (time-based interpolation)
# but keep subgroup columns untouched.
if df[rate_col].isnull().any():
    df[rate_col] = df[rate_col].interpolate(method='time', limit_direction='both')

# Drop exact duplicate rows
before = len(df)
df = df[~df.index.duplicated(keep='first')]
print(f"\nDropped {before - len(df)} duplicate-index rows (if any). Now {len(df)} rows remain."
)

# ---------- 7. Optional subgroup columns ----------
# Detect possible subgroup columns like 'region', 'age_group', 'sex', etc.
subgroups = [c for c in df.columns if c not in [rate_col]]
print("\nPossible subgroup columns (non-rate):", subgroups)

# ---------- 8. Resampling & smoothing ----------
# Ensure monthly frequency if data is monthly
# If daily or irregular, use the existing frequency but for many operations we'll resample to m
onthly mean.
try:
    freq = pd.infer_freq(df.index)
except Exception:
    freq = None
print("Inferred freq:", freq)
monthly = df[rate_col].resample('M').mean()
monthly.name = 'unemployment_rate'

# Add rolling averages
monthly = monthly.to_frame()
monthly['rate_3m'] = monthly['unemployment_rate'].rolling(window=3, min_periods=1).mean()
monthly['rate_12m'] = monthly['unemployment_rate'].rolling(window=12, min_periods=1).mean()

# Save cleaned monthly series for analysis
monthly.to_csv(OUT_DIR / 'monthly_unemployment_rate.csv')

# ---------- 9. Summary statistics by year ----------
monthly['year'] = monthly.index.year
summary_by_year = monthly.groupby('year')['unemployment_rate'].agg(['count','mean','median','mi
n','max','std']).reset_index()
summary_by_year.to_csv(OUT_DIR / 'summary_by_year.csv', index=False)
print("\nSummary by year saved to", OUT_DIR / 'summary_by_year.csv')
print(summary_by_year.head())

# ---------- 10. Time-series plots ----------
plt.figure()
plt.plot(monthly.index, monthly['unemployment_rate'], label='Monthly rate')
plt.plot(monthly.index, monthly['rate_3m'], label='3-month MA', linestyle='--')
plt.plot(monthly.index, monthly['rate_12m'], label='12-month MA', linestyle=':')
plt.title('Unemployment Rate - Time Series')
plt.ylabel('Unemployment Rate (percent)')
plt.xlabel('Date')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(FIG_DIR / 'ts_unemployment_rate.png', dpi=150)
plt.close()
```

```
# ---------- 11. COVID-19 impact analysis ----------
# Define periods
pre_period = ('2017-01-01', '2019-12-31')
covid_period = ('2020-03-01', '2021-12-31')

pre_vals = monthly.loc[pre_period[0]:pre_period[1], 'unemployment_rate'].dropna()
covid_vals = monthly.loc[covid_period[0]:covid_period[1], 'unemployment_rate'].dropna()

pre_mean = pre_vals.mean() if len(pre_vals)>0 else np.nan
covid_mean = covid_vals.mean() if len(covid_vals)>0 else np.nan
change_abs = covid_mean - pre_mean
change_pct = (change_abs / pre_mean)*100 if pre_mean!=0 else np.nan

ttest_res = None
if len(pre_vals) >= 3 and len(covid_vals) >= 3:
    tstat, pval = ttest_ind(pre_vals, covid_vals, equal_var=False, nan_policy='omit')
    ttest_res = (tstat, pval)

covid_summary = {
    'pre_mean': pre_mean,
    'covid_mean': covid_mean,
    'change_abs': change_abs,
    'change_pct': change_pct,
    'ttest': ttest_res
}
import json
with open(OUT_DIR / 'covid_summary.json','w') as f:
    json.dump({k: (v.tolist() if hasattr(v,'tolist') else v) for k,v in covid_summary.items()},
 f, default=str)
print("\nCOVID summary:", covid_summary)

# Plot with COVID annotation
plt.figure()
plt.plot(monthly.index, monthly['unemployment_rate'], label='Monthly')
plt.axvspan(pd.to_datetime(covid_period[0]), pd.to_datetime(covid_period[1]), color='red', alph
a=0.12, label='COVID period')
plt.title('Unemployment Rate with COVID-19 period')
plt.ylabel('Unemployment Rate')
plt.xlabel('Date')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(FIG_DIR / 'ts_with_covid_annotation.png', dpi=150)
plt.close()

# ---------- 12. Seasonality decomposition (STL) ----------
# STL expects no NA and at least two seasonal cycles (ideal)
series_for_stl = monthly['unemployment_rate'].dropna()
period = 12  # monthly seasonality
if len(series_for_stl) >= (period*2):
    stl = STL(series_for_stl, period=period, robust=True)
    res = stl.fit()
    # Save components plot
    fig = res.plot()
    fig.set_size_inches(10,8)
    fig.savefig(FIG_DIR / 'stl_decomposition.png', dpi=150)
    plt.close(fig)
    # Save components to CSV
    comp = pd.DataFrame({'trend': res.trend, 'seasonal': res.seasonal, 'resid': res.resid})
    comp.to_csv(OUT_DIR / 'stl_components.csv')
else:
    print("Not enough data points for robust STL decomposition. Need at least 2*period rows.")

# ---------- 13. Seasonality heatmap by month ----------
monthly['month'] = monthly.index.month
pivot = monthly.reset_index().pivot_table(values='unemployment_rate', index='year', columns='mo
nth', aggfunc='mean')
plt.figure(figsize=(10,6))
sns.heatmap(pivot, annot=True, fmt='.2f', cmap='viridis')
plt.title('Average unemployment rate by Year (rows) and Month (columns)')
plt.tight_layout()
plt.savefig(FIG_DIR / 'seasonality_heatmap.png', dpi=150)
plt.close()

# ---------- 14. Subgroup analyses (if subgroup columns exist) ----------
# If original df had subgroup columns like 'region' etc, attempt groupby
```

```
# We'll reload raw CSV to use subgroup columns (if any)
with zipfile.ZipFile(DATA_ZIP) as z:
    raw = pd.read_csv(z.open(name))
raw.columns = [c.strip().lower() for c in raw.columns]
if date_col not in raw.columns:
    raw[date_col] = pd.to_datetime(raw[raw.columns[0]], errors='coerce')
else:
    raw[date_col] = pd.to_datetime(raw[date_col], errors='coerce')
raw = raw.sort_values(by=date_col)
# Try common subgroup columns
possible_subgroup_cols = [c for c in raw.columns if c not in [rate_col, date_col]]
print("\nPossible subgroup cols in raw data:", possible_subgroup_cols)

# Example subgroup plot for up to 4 groups if a 'region' or similar column exists
for col in ['region','age_group','sex','gender']:
    if col in raw.columns:
        # compute monthly mean by subgroup
        raw[col] = raw[col].astype(str)
        raw[rate_col] = pd.to_numeric(raw[rate_col], errors='coerce')
        grp = raw.set_index(date_col).groupby([pd.Grouper(freq='M'), col])[rate_col].mean().uns
tack(col)
        # plot top 6 groups
        top_cols = list(grp.columns)[:6]
        plt.figure(figsize=(10,5))
        for c in top_cols:
            plt.plot(grp.index, grp[c], label=str(c))
        plt.legend()
        plt.title(f'Unemployment rate by {col}')
        plt.tight_layout()
        plt.savefig(FIG_DIR / f'unemployment_by_{col}.png', dpi=150)
        plt.close()

# ---------- 15. Generate a short report (text) ----------
report_lines = []
report_lines.append("Unemployment Rate Analysis - Short Report")
report_lines.append("==============================")
report_lines.append(f"Data file used: {name}")
report_lines.append("")
report_lines.append("Key findings (automatically generated):")
report_lines.append(f"- Data range: {monthly.index.min().date()} to {monthly.index.max().date()
}")
report_lines.append(f"- Pre-pandemic mean (2017-2019): {pre_mean:.3f}")
report_lines.append(f"- Pandemic mean (Mar 2020-Dec 2021): {covid_mean:.3f}")
report_lines.append(f"- Absolute change: {change_abs:.3f}")
report_lines.append(f"- Percent change: {change_pct:.2f}%")
if ttest_res:
    report_lines.append(f"- t-test stat: {ttest_res[0]:.4f}, p-value: {ttest_res[1]:.4f}")
report_lines.append("")
report_lines.append("Suggested policy recommendations (auto-generated):")
report_lines.append("- Targeted job training programs in regions/groups with largest increases.
")
report_lines.append("- Strengthen unemployment benefits during shock periods; consider extensio
n mechanisms.")
report_lines.append("- Support sectors with persistent higher unemployment via subsidies or inc
entives.")
report_lines.append("")
report_lines.append("Saved figures: see the 'figures' directory in the output folder.")
report_text = "\\n".join(report_lines)
with open(OUT_DIR / 'short_report.txt','w') as f:
    f.write(report_text.replace('\\n','\n'))
print("\nShort report saved to", OUT_DIR / 'short_report.txt')

# ---------- 16. Save code as a .py for convenience ----------
code_text = open(__file__, 'r').read() if '__file__' in globals() else None
# We'll save the code we provided above into a file for the user
with open(OUT_DIR / 'unemployment_analysis_script.py', 'w') as f:
    f.write(code)

# ---------- 17. Try to create a PDF containing the code for easy copy-paste ----------
pdf_path = OUT_DIR / 'unemployment_analysis_code.pdf'
try:
    from reportlab.lib.pagesizes import A4
    from reportlab.pdfgen import canvas
    from reportlab.lib.units import mm
    c = canvas.Canvas(str(pdf_path), pagesize=A4)
    width, height = A4
```

```
    left_margin = 15*mm
    top = height - 15*mm
    lines = code.splitlines()
    font_size = 8
    leading = font_size + 2
    c.setFont("Courier", font_size)
    y = top
    for line in lines:
        if y < 15*mm:
            c.showPage()
            c.setFont("Courier", font_size)
            y = top
        # Ensure long lines wrap - simple wrap at 95 chars
        max_chars = 95
        while len(line) > max_chars:
            c.drawString(left_margin, y, line[:max_chars])
            line = line[max_chars:]
            y -= leading
            if y < 15*mm:
                c.showPage()
                c.setFont("Courier", font_size)
                y = top
        c.drawString(left_margin, y, line)
        y -= leading
    c.save()
    print("PDF created at", pdf_path)
except Exception as e:
    # fallback: save plain text file
    fallback = OUT_DIR / 'unemployment_analysis_code.txt'
    with open(fallback, 'w') as f:
        f.write(code)
    print("reportlab not available or failed to create PDF. Saved code as", fallback)

# ---------- Done ----------
print("\nAll outputs saved in:", OUT_DIR)
print("Figures in:", FIG_DIR)
```