```
import pandas as pd  # 1.1

tsv_path = r"C:\Users\□□□□□□□□\Desktop\earthquakes-2025-11-10_18-18-40_+0800.tsv"
Sig_Eqs = pd.read_csv(tsv_path, sep="\t", low_memory=False, encoding="utf-8")
import re

# Sig_Eqs = pd.read_csv(r"C:\Users\10147\Desktop\earthquakes-2025-11-10_18-18-40_+0800.tsv", sep="\t", low_memory=False, encoding="utf-8")

df = Sig_Eqs.copy()

# □□□□□
required = {"Year", "Total Deaths", "Location Name"}
missing = required - set(df.columns)
if missing:
    raise ValueError(f"□□□□□: {missing}")

# □□□□□
df["Year"] = pd.to_numeric(df["Year"], errors="coerce")
df["Total Deaths"] = pd.to_numeric(df["Total Deaths"], errors="coerce")
df = df[df["Year"] >= -2150]

# □□□□□□□□□□□□□□□□□□
country = df["Location Name"].astype(str).str.split(":", n=1, expand=True)[0].str.strip()
df = df.assign(Country=country)

# □□□□□
df = df[df["Country"].notna() & (df["Country"] != "")]

# □□□□□ sum □□□□□□□□□□□□□□□□
top10 = (
    df.groupby("Country", as_index=False)["Total Deaths"].sum()
        .rename(columns={"Total Deaths": "
Total_Deaths_Since_2150BC"})

        .sort_values("Total_Deaths_Since_2150BC", ascending=False)
        .head(10)
)

print("⊠⊠⊠⊠2150⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠")
print(top10.to_string(index=False))



import matplotlib.pyplot as plt        #1.2
for col in ["Year", "Mag"]:
    if col not in df.columns:
        raise ValueError(f"⊠⊠⊠⊠⊠: {col}")

# ⊠⊠⊠⊠
df["Year"] = pd.to_numeric(df["Year"], errors="coerce")
df["Mag"]  = pd.to_numeric(df["Mag"],  errors="coerce")

# ⊠⊠⊠Mag > 6.0⊠⊠ Year ⊠⊠⊠
df6 = df[(df["Mag"] > 6.0) & df["Year"].notna()]

# ⊠⊠⊠⊠
counts = (
    df6.groupby("Year")
       .size()
       .reset_index(name="count_m_gt_6")
       .sort_values("Year")
)

# ⊠⊠⊠⊠⊠⊠⊠
plt.figure(figsize=(10, 4))
plt.plot(counts["Year"], counts["count_m_gt_6"], color="#1f77b4", lw=1.5)
plt.title("Global count of earthquakes with Mag > 6.0 by Year")
plt.xlabel("Year (BCE ⊠⊠⊠)")
```

```python
plt.ylabel("Count (Mag > 6.0)")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# □□□□□□□□□□□ >= 1900□□□□□□□□□□□□□
modern = counts[counts["Year"] >= 1900]
if len(modern) > 0:
    plt.figure(figsize=(10, 4))
    plt.plot(modern["Year"], modern["count_m_gt_6"], color="#d62728", lw=1.5)
    plt.title("Global count of earthquakes with Mag > 6.0 by Year (>= 1900)")
    plt.xlabel("Year")
    plt.ylabel("Count (Mag > 6.0)")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# □□□□□□□□□
print(counts.head(10))
print(counts.tail(10))


from typing import Dict, Any
df0 = Sig_Eqs.copy()             #1.3

# □□□□□
need = {"Year", "Location Name", "Mag"}
missing = need - set(df0.columns)
if missing:
    raise ValueError(f"□□□□□: {missing}")

# □□□□
df0["Year"] = pd.to_numeric(df0["Year"], errors="coerce")
df0["Mag"]  = pd.to_numeric(df0["Mag"],  errors="coerce")

# □□□□□□□□□□□□□□□□
df0["Country"] = df0["Location Name"].astype(str).
```

```
str.split(":", n=1, expand=True)[0].str.strip()

# ▯▯▯▯▯▯▯▯▯ 2150 ▯▯▯
df0 = df0[df0["Year"] >= -2150].copy()

# ▯▯▯▯▯▯▯
def _pick(df, candidates):
    for c in candidates:
        if c in df.columns:
            return c
    return None

col_date  = _pick(df0, ["Date", "date"])
col_month = _pick(df0, ["Mo", "Month"])
col_day   = _pick(df0, ["Dy", "Day"])

# ▯▯▯▯▯▯▯▯▯
def compose_date_str(row) -> str:
    if col_date and pd.notna(row.get(col_date)):
        return str(row[col_date]).strip()
    y = row.get("Year")
    if pd.isna(y):
        return ""
    try:
        yi = int(y)
    except Exception:
        return ""
    def to_int(v):
        try:
            return int(v)
        except Exception:
            return None
    m = to_int(row.get(col_month)) if col_month else None
    d = to_int(row.get(col_day))   if col_day   else None
    s_year = f"{abs(yi):04d}"
    s_md = ""
    if m is not None:
        s_md += f"-{m:02d}"
        if d is not None:
```

```python
            s_md += f"-{d:02d}"
    return f"{s_year}{s_md} BCE" if yi < 0 else f"{s_year}{s_md}"

df0["Date_str"] = df0.apply(compose_date_str, axis=1)

# □□□□
def CountEq_LargestEq(df: pd.DataFrame, country: str) -> Dict[str, Any]:
    g = df[df["Country"] == country]
    total_count = int(len(g))
    if total_count == 0:
        return {"total_count": 0, "largest_eq_date": "", "largest_mag": float("nan")}
    g_valid = g[pd.notna(g["Mag"])]
    if g_valid.empty:
        return {"total_count": total_count, "largest_eq_date": "", "largest_mag": float("nan")}
    max_mag = float(g_valid["Mag"].max())
    g_max = g_valid[g_valid["Mag"] == max_mag].copy()
    # □ Year, Month, Day □□□□□
    sort_cols = ["Year"]
    if col_month: sort_cols.append(col_month)
    if col_day:   sort_cols.append(col_day)
    for c in sort_cols:
        g_max[c] = pd.to_numeric(g_max[c], errors="coerce")
    g_max = g_max.sort_values(sort_cols, na_position="last")
    date_str = str(g_max.iloc[0]["Date_str"])
    return {"total_count": total_count, "largest_eq_date": date_str, "largest_mag": max_mag}

# □□□□□□□□□□□□□□□□□□□□□□□□□□□□
countries = df0["Country"].dropna().unique().tolist()
rows = []
for c in countries:
    out = CountEq_LargestEq(df0, c)
    rows.append({"Country": c, **out})

result = (
    pd.DataFrame(rows)
      .sort_values(["total_count", "largest_mag"], ascending=[False, False])
      .reset_index(drop=True)
)

print(result.head(20))


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from scipy.stats import linregress
import calendar

csv_path = Path(r"C:\Users\10147\Desktop\2281305.csv")  # ▯▯▯▯▯▯
peek_cols = pd.read_csv(csv_path, nrows=0).columns
usecols = ["DATE", "WND"] if set(["DATE", "WND"]).issubset(peek_cols) else None
df = pd.read_csv(csv_path, low_memory=False, usecols=usecols)

df["DATE"] = pd.to_datetime(df["DATE"], errors="coerce", utc=True)
df = df.dropna(subset=["DATE"])
df["dt_local"] = df["DATE"].dt.tz_convert("Asia/Shanghai")

def extract_wspd_ms(df: pd.DataFrame) -> pd.Series:
    # WND: dir, dir_qc, type, speed, spd_qc
    if "WND" in df.columns:
        parts = df["WND"].astype(str).str.split(",", expand=True)
        if parts.shape[1] < 5:
            raise ValueError("WND ▯▯▯▯▯▯▯▯▯▯▯▯ 5 ▯▯")
        spd = pd.to_numeric(parts[3], errors="coerce")  # m/s
        qc  = parts[4].astype(str).str.strip()
        # □□□□□□□□
        spd[(spd >= 999) | (spd < 0)] = np.nan
        spd[~qc.isin(["1", "5"])] = np.nan
        return spd
    for c in ["WIND_SPEED", "wind_speed", "WSPD", "wspd"]:
        if c in df.columns:
            return pd.to_numeric(df[c], errors="coerce")

    raise ValueError("□□□ WND □□□□□□□□□")

df["wspd_ms"] = extract_wspd_ms(df)
start = pd.Timestamp("2010-01-01 00:00:00", tz="Asia/Shanghai")
end   = pd.Timestamp("2020-12-31 23:59:59", tz="Asia/Shanghai")
df = df[(df["dt_local"] >= start) & (df["dt_local"] <= end)].copy()
def expected_hours(ts: pd.Timestamp) -> int:
    y, m = ts.year, ts.month
    return calendar.monthrange(y, m)[1] * 24  # Asia/Shanghai □□□□

# □□□□□□□□□□□
s = df.set_index("dt_local")["wspd_ms"]
monthly = s.resample("MS").agg(mean="mean", count="count").dropna(subset=["mean"], how="any")

# □□□□□□□□□□□□
exp_hours = pd.Series({idx: expected_hours(idx) for idx in monthly.index})
monthly["coverage"] = monthly["count"] / exp_hours
monthly_f = monthly[monthly["coverage"] >= 0.70].copy()
if len(monthly_f) < 12:
    print("□□□□□□□□□□□□□□□□□")
    slope_per_year = np.nan
    slope_per_decade = np.nan
    p_value = np.nan
else:
    # □□“□”□□□□□□□□□□□□□□□□□□□□
    t_years = (monthly_f.index - monthly_f.index[0
]).days / 365.2425
    y = monthly_f["mean"].to_numpy()
    # □□ NaN
    mask = ~np.isnan(t_years) & ~np.isnan(y)
    res = linregress(t_years[mask], y[mask])
    slope_per_year = res.slope
    slope_per_decade = res.slope * 10.0
    p_value = res.pvalue
    # 95% □□□□□□□□□□□□□□□□ ± 1.96*stderr
    ci95 = (res.slope - 1.96*res.stderr, res.slope
 + 1.96*res.stderr)
    ci95_decade = (ci95[0]*10, ci95[1]*10)

plt.figure(figsize=(11, 5))
plt.plot(monthly.index, monthly["mean"], label="
Monthly mean (raw)", color="#9ecae1")
plt.plot(monthly_f.index, monthly_f["mean"], label="
Monthly mean (>=70% coverage)", color="#1f77b4")
# 12 □□□□□□□□□□□□□□□□□□□□□□□
roll12 = monthly_f["mean"].rolling(12, min_periods=6
).mean()
plt.plot(monthly_f.index, roll12, label="12-mo
rolling mean", color="#2ca02c", linewidth=2)

title = "Shenzhen Bao'an Intl (Station 2281305)\n
Monthly averaged wind speed (2010-2020, local time)"
plt.title(title)
plt.xlabel("Observation time (month)")
plt.ylabel("Wind speed (m/s)")
plt.grid(alpha=0.3)
plt.legend()

# □□□□□□□□□□□□□
if not np.isnan(slope_per_year):
    ann = (f"Trend (OLS, {len(monthly_f)} months):\n
"
```

```
           f"{slope_per_year:.3f} m/s per year "
           f"({slope_per_decade:.3f} per decade)\n"
           f"p = {p_value:.3g}")
    plt.gca().text(0.01, 0.02, ann, transform=plt.gca().transAxes,
                   fontsize=10, va="bottom", ha="left",
                   bbox=dict(facecolor="white", alpha=0.7, edgecolor="none"))

plt.tight_layout()
plt.show()


print("□□□□□", start.strftime("%Y-%m-%d"), "—", end.strftime("%Y-%m-%d"))
print(f"□□□□□□□□□□□□{len(monthly_f)} / {len(monthly)}")
if not np.isnan(slope_per_year):
    print(f"□□□□□□□{slope_per_year:.4f} m/s/□  □□□□{slope_per_decade:.4f} m/s□")
    print(f"p □□{p_value:.4g} □□□□□ 0.05□")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings

warnings.filterwarnings('ignore')

# □□□□□□
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


# 3.1 □□□□□□□
def load_and_clean_data():
    """
    □□Excel□□□□□□□
    """
    # ⧄⧄Excel⧄⧄ - ⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄
    df = pd.read_excel(r'E:\learning\ESE5023_Assignments_12532745\assignments2\⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄.xlsx', sheet_name='09 ⧄⧄⧄',
                       header=1)

    # ⧄⧄⧄⧄⧄⧄⧄⧄
    print("⧄⧄⧄⧄⧄⧄:", df.shape)
    print("\n⧄⧄⧄⧄⧄⧄:")
    print(df.columns.tolist())

    # ⧄⧄⧄⧄ - ⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄
    # ⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄
    population_data = {}

    # ⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄ (⧄2⧄)
    resident_pop = df.iloc[1, 1:].reset_index(drop=True)
    # ⧄⧄⧄⧄⧄⧄⧄⧄ (⧄3⧄)
    urban_pop = df.iloc[2, 1:].reset_index(drop=True)
    # ⧄⧄⧄⧄⧄⧄⧄⧄ (⧄4⧄)
    rural_pop = df.iloc[3, 1:].reset_index(drop=True)
    # ⧄⧄⧄⧄⧄⧄⧄⧄⧄⧄ (⧄6⧄)
    registered_pop = df.iloc[5, 1:].reset_index(drop=True)

    # ⧄⧄⧄⧄⧄
    years = df.columns[1:].values

    # ⧄⧄⧄⧄⧄⧄⧄⧄⧄
    clean_df = pd.DataFrame({
        '⧄⧄': years,
        '⧄⧄⧄⧄⧄⧄': resident_pop.values,
        '⧄⧄⧄⧄': urban_pop.values,
        '⧄⧄⧄⧄': rural_pop.values,
        '⧄⧄⧄⧄⧄⧄': registered_pop.values
    })

    # □□□□□□
    print("\n=== □□□□□□ ===")

    # 1. □□□□□
    print("1. □□□□□:")
    print(clean_df.isnull().sum())

    # 2. □□□□□ - □□□□□□□
    clean_df = clean_df.fillna(method='ffill').fillna(method='bfill')

    # 3. □□□□□□
    for col in clean_df.columns[1:]:
        clean_df[col] = pd.to_numeric(clean_df[col], errors='coerce')

    # 4. □□□□□□□
    print("\n2. □□□□□□□□:")
    print(clean_df.isnull().sum())

    # 5. □□□□□□□□
    clean_df = clean_df.dropna(how='all')

    # 6. □□□□
    clean_df = clean_df.reset_index(drop=True)

    print(f"\n3. □□□□□□□: {clean_df.shape}")
    print("\n4. □□□5□:")
    print(clean_df.head())

    return clean_df


# 3.2 □□□□□□□
def plot_time_series(clean_df, variable='□□□□□□'):
    """
    □□□□□□□□□□□□
    """
    plt.figure(figsize=(12, 8))

    # □□□□□□□
```

```python
    plt.subplot(2, 1, 1)
    plt.plot(clean_df['□□'], clean_df[variable], marker='o', linewidth=2, markersize=6)
    plt.title(f'□□□{variable}□□□□□□ (1988-2021)', fontsize=14, fontweight='bold')
    plt.xlabel('□□', fontsize=12)
    plt.ylabel(f'{variable}(□)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.xticks(rotation=45)

    # □□□□□
    x_numeric = range(len(clean_df))
    z = np.polyfit(x_numeric, clean_df[variable], 1)
    p = np.poly1d(z)
    plt.plot(clean_df['□□'], p(x_numeric), "r--", alpha=0.8, label='□□□')
    plt.legend()

    # □□□□□□□□□
    plt.subplot(2, 1, 2)
    plt.plot(clean_df['□□'], clean_df['□□□□'], marker='s', label='□□□□', linewidth=2)
    plt.plot(clean_df['□□'], clean_df['□□□□'], marker='^', label='□□□□', linewidth=2)
    plt.title('□□□□□□□□□□□ (1988-2021)', fontsize=14, fontweight='bold')
    plt.xlabel('□□', fontsize=12)
    plt.ylabel('□□□□(□)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.xticks(rotation=45)

    plt.tight_layout()
    plt.show()

    # □□□□
    plt.savefig('□□□□□□□□□□□.png', dpi=300, bbox_inches='tight')


# 3.3 □□□□
```

```
def statistical_analysis(clean_df, variable='▯▯▯▯▯▯'):
    """
    ▯▯▯▯▯▯▯▯▯▯▯
    """
    print(f"\n=== {variable}▯▯▯▯▯▯ ===\n")

    data = clean_df[variable]

    # 1. ▯▯▯▯▯▯▯
    print("1. ▯▯▯▯▯▯▯:")
    desc_stats = data.describe()
    print(desc_stats)

    # 2. ▯▯▯▯
    print("\n2. ▯▯▯▯▯▯:")
    growth_rates = data.pct_change() * 100
    print(f"▯▯▯▯▯: {growth_rates.mean():.2f}%")
    print(f"▯▯▯▯▯: {growth_rates.max():.2f}% (▯▯: {clean_df.loc[growth_rates.idxmax(), '▯▯']})")
    print(f"▯▯▯▯▯: {growth_rates.min():.2f}% (▯▯: {clean_df.loc[growth_rates.idxmin(), '▯▯']})")

    # 3. ▯▯▯▯
    print("\n3. ▯▯▯▯:")
    # ▯▯▯▯▯▯
    x = np.arange(len(data))
    slope, intercept = np.polyfit(x, data, 1)
    print(f"▯▯▯▯▯▯: {slope:.2f} (▯▯▯▯▯▯▯▯)")

    # 4. ▯▯▯▯▯
    print("\n4. ▯▯▯▯▯:")
    rolling_std = data.rolling(window=5).std()
    print(f"▯▯▯▯▯: {data.std():.2f}")
    print(f"▯▯▯▯: {(data.std() / data.mean() * 100):.2f}%")

    # 5. ▯▯▯▯
    print("\n5. ▯▯▯▯:")
    skewness = data.skew()
    kurtosis = data.kurtosis()
    print(f"□□: {skewness:.2f} ({'□□' if skewness > 0 else '□□' if skewness < 0 else '□□'})")
    print(f"□□: {kurtosis:.2f} ({'□□' if kurtosis > 0 else '□□' if kurtosis < 0 else '□□'})")

    # 6. □□□□□
    print("\n6. □□□□□:")
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    print(f"□□□□□: {len(outliers)}")
    if len(outliers) > 0:
        print(f"□□□□□: {[clean_df.loc[data[data == out].index[0], '□□'] for out in outliers]}")


# □□□□□□
def main():
    print("=" * 50)
    print("□□□□□□□□□")
    print("=" * 50)

    # 3.1 □□□□□□□
    print("\n□□3.1: □□□□□□□")
    clean_df = load_and_clean_data()

    # 3.2 □□□□□□
    print("\n□□3.2: □□□□□□")
    plot_time_series(clean_df, '□□□□□□')

    # 3.3 □□□□
    print("\n□□3.3: □□□□")
    statistical_analysis(clean_df, '□□□□□□')

    # □□□□□□□□□
    print("\n=== □□□□: □□□□ ===")
    clean_df['□□□□'] = (clean_df['□□□□'] / clean_df[
'□□□□□□']) * 100
    print(f"2021□□□□□: {clean_df['□□□□'].iloc[-1]:.2f}%")
    print(f"□□□□□□: {clean_df['□□□□'].iloc[0]:.2f}% → {clean_df['□□□□'].iloc[-1]:.2f}%")

    # □□□□□□□□
    clean_df.to_csv('□□□□□□□_□□□.csv', index=False, encoding='utf-8-sig')
    print(f"\n□□□□□□□□□□: □□□□□□□_□□□.csv")


# □□□□□
if __name__ == "__main__":
    main()
```