3.High-Performance Time Series Transformation (NumPy + pandas)

(i)timeseries_utils.py

```
import numpy as np
import pandas as pd
import time
from scipy.signal import butter, filtfilt

def rolling_stats_numpy(data, window):
    cumsum = np.cumsum(np.insert(data, 0, 0))
    means = (cumsum[window:] - cumsum[:-window]) / window
    return means

def ewma_numpy(data, alpha):
    result = np.zeros_like(data)
    result[0] = data[0]
    for t in range(1, len(data)):
        result[t] = alpha * data[t] + (1 - alpha) * result[t - 1]
    return result

def fft_bandpass(data, low, high, fs):
    fft_vals = np.fft.fft(data)
    freqs = np.fft.fftfreq(len(data), d=1/fs)
    fft_filtered = fft_vals.copy()
    fft_filtered[(np.abs(freqs) < low) | (np.abs(freqs) > high)] = 0
    return np.fft.ifft(fft_filtered).real
```

(ii)benchmark.py

```
%%writefile text_analyzer.py
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag
from nltk.probability import FreqDist
from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder
from textblob import TextBlob
import matplotlib.pyplot as plt

def analyze_text(text):
    # Tokenization
    words = word_tokenize(text)
    sentences = sent_tokenize(text)

    # POS Tagging
    pos_tags = pos_tag(words)

    # Frequency Distribution
    fdist = FreqDist(words)

    # Bigrams
    finder = BigramCollocationFinder.from_words(words)
    bigram_measures = BigramAssocMeasures()
    scored = finder.score_ngrams(bigram_measures.pmi)
```

```
    # Sentiment Analysis
    sentiment = TextBlob(text).sentiment

    return {
        "words": words,
        "sentences": sentences,
        "pos_tags": pos_tags,
        "fdist": fdist,
        "bigrams": scored,
        "sentiment": sentiment
    }
```

```
Writing text_analyzer.py
```

```
%%writefile streamlit_app.py
import streamlit as st
import pandas as pd
from text_analyzer import analyze_text
import matplotlib.pyplot as plt

st.title("Text Analytics Web App")

uploaded_file = st.file_uploader("Upload a text file", type=["txt"])

if uploaded_file is not None:
    text = uploaded_file.read().decode("utf-8")
    st.text_area("Your Text", text, height=200)

    analysis = analyze_text(text)

    st.header("Tokenization")
    st.write("Number of words:", len(analysis["words"]))
    st.write("Number of sentences:", len(analysis["sentences"]))

    st.header("POS Tagging")
    st.dataframe(pd.DataFrame(analysis["pos_tags"], columns=["Word", "POS"]))

    st.header("Frequency Distribution")
    st.bar_chart(analysis["fdist"])

    st.header("Bigrams")
    st.dataframe(pd.DataFrame(analysis["bigrams"], columns=["Bigram", "PMI"]))

    st.header("Sentiment Analysis")
    st.write("Polarity:", analysis["sentiment"].polarity)
    st.write("Subjectivity:", analysis["sentiment"].subjectivity)
```

```
Overwriting streamlit_app.py
```

(iii)sample_data.npy

```
np.save("sample_data.npy", data)
```

(iv)report.md

```
numpy as np
pandas as pd
ipy.fftpack import fft, ifft

e large time-series data
ze = 10**6
np.random.randn(data_size)

ling_stats_numpy(data, window):

putes rolling mean and standard deviation using NumPy's stride tricks for speed.

pe = (data.size - window + 1, window)
ides = (data.strides[0], data.strides[0])
ling_data = np.lib.stride_tricks.as_strided(data, shape=shape, strides=strides)
n = np.mean(rolling_data, axis=1)
 = np.std(rolling_data, axis=1)
urn mean, std

ling_stats_pandas(data, window):

putes rolling mean and standard deviation using Pandas.

= pd.Series(data)
n = df.rolling(window=window).mean()
 = df.rolling(window=window).std()
urn mean.dropna().values, std.dropna().values

a_numpy(data, span):

putes Exponentially Weighted Moving Average (EWMA) using NumPy.

ha = 2 / (span + 1)
a = np.zeros_like(data)
a[0] = data[0]
 i in range(1, len(data)):
 ewma[i] = alpha * data[i] + (1 - alpha) * ewma[i-1]
urn ewma

a_pandas(data, span):

putes Exponentially Weighted Moving Average (EWMA) using Pandas.

urn pd.Series(data).ewm(span=span).mean().values

_filter(data, min_freq, max_freq, samplerate):

lies a frequency filter using Fast Fourier Transform (FFT).

 len(data)
= fft(data)
= np.fft.fftfreq(n, 1 / samplerate)

reate a mask to zero out frequencies outside the desired range
filtered = yf.copy()
filtered[(np.abs(xf) < min_freq) | (np.abs(xf) > max_freq)] = 0

urn ifft(yf_filtered)
```

```
part of the script
me__ == '__main__':
dow_size = 100
a_span = 100
plerate = 1000

nt("--- Running Rolling Stats ---")

olling Stats with NumPy
mean, np_std = rolling_stats_numpy(data, window_size)
nt(f"NumPy Rolling Mean computed. Shape: {np_mean.shape}")

olling Stats with Pandas
mean, pd_std = rolling_stats_pandas(data, window_size)
nt(f"Pandas Rolling Mean computed. Shape: {pd_mean.shape}")

heck if results are similar
nt(f"Are NumPy and Pandas rolling means close? {np.allclose(np_mean, pd_mean, atol=1e-5)}")

nt("\n--- Running EWMA ---")

WMA with NumPy
ewma = ewma_numpy(data, ewma_span)
nt(f"NumPy EWMA computed. Shape: {np_ewma.shape}")

WMA with Pandas
ewma = ewma_pandas(data, ewma_span)
nt(f"Pandas EWMA computed. Shape: {pd_ewma.shape}")

heck if results are similar
nt(f"Are NumPy and Pandas EWMA close? {np.allclose(np_ewma, pd_ewma, atol=1e-5)}")

nt("\n--- Running FFT Filtering ---")

FT Filtering
_f = 10
_f = 50
tered_data = fft_filter(data, min_f, max_f, samplerate)
nt(f"FFT Filtered data computed. Shape: {filtered_data.shape}")
nt("FFT filtering is effective for isolating specific frequency bands.")
```

```
--- Running Rolling Stats ---
NumPy Rolling Mean computed. Shape: (999901,)
Pandas Rolling Mean computed. Shape: (999901,)
Are NumPy and Pandas rolling means close? True

--- Running EWMA ---
NumPy EWMA computed. Shape: (1000000,)
Pandas EWMA computed. Shape: (1000000,)
Are NumPy and Pandas EWMA close? False

--- Running FFT Filtering ---
FFT Filtered data computed. Shape: (1000000,)
FFT filtering is effective for isolating specific frequency bands.
```