Project 3, APS1070 Fall 2023

PCA [11 marks]

Deadline: Nov 17th, 23:00

Academic Integrity

This project is individual - it is to be completed on your own. If you have questions, please post your query in the APS1070 Piazza Q&A forums (the answer might be useful to others!).

Do not share your code with others, or post your work online. Do not submit code that you have not written yourself. Students suspected of plagiarism on a project, midterm or exam will be referred to the department for formal discipline for breaches of the Student Code of Conduct.

In this project we work on a temperature dataset that reports the average earth surface temperature for different cities for each month over the years 1992-2006.

Please fill out the following:

- **Name**: Yi-Fan, Hung
- **Student number**: 1010807799

How to submit **(HTML + IPYNB)**

1. Download your notebook: `File -> Download .ipynb`
2. Click on the Files icon on the far left menu of Colab
3. Select & upload your `.ipynb` file you just downloaded, and then obtain its path (right click) (you might need to hit the Refresh button before your file shows up)
1. execute the following in a Colab cell:

   ```
   %%shell
   jupyter nbconvert --to html /PATH/TO/YOUR/NOTEBOOKFILE.ipynb
   ```
2. An HTML version of your notebook will appear in the files, so you can download it.
3. Submit **both** `HTML` and `IPYNB` files on Quercus for grading.

Ref: https://stackoverflow.com/a/64487858

In [83]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (6.
5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbc
onvert) (4.9.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from n
bconvert) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (fr
om nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-pack
ages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (f
rom nbconvert) (3.1.2)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert) (5.5.0)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-pac
kages (from nbconvert) (0.2.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-package
s (from nbconvert) (2.1.3)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-package
s (from nbconvert) (0.9.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (5.9.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
m nbconvert) (23.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-pa
ckages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.10/dist-package
s (from nbconvert) (2.16.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from
nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (5.7.1)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packa
ges (from jupyter-core>=4.7->nbconvert) (3.11.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.10/dist-
packages (from nbclient>=0.5.0->nbconvert) (6.1.12)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-packages
(from nbformat>=5.1->nbconvert) (2.18.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-package
s (from nbformat>=5.1->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages
(from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (fr
om bleach->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages
(from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages
(from jsonschema>=2.6->nbformat>=5.1->nbconvert) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/py
thon3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-pac
kages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages
(from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.12.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (fro
m jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-pa
ckages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-packages
(from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.3.2)
```

Part 1: Getting started with GitHub [1 Mark + 1 Mark Git Submission]

This first part of the project assignment is to be completed independently from Parts 2 - 5. In this part you will be completing some coding tasks and submitting your results on Github. To access this part of the assignment and upload your answers, you will need to use Github. Please complete the following step-by-step instructions:

1. Create a Github account and install git for Windows or Mac:

   - https://git-scm.com/download/win
   - https://git-scm.com/download/mac

2. Open this link: https://classroom.github.com/a/BWpQKQJt to create your assignment repository in GitHub. You should get a link similar to:

   `https://github.com/APS1070-UofT/project-3-part-1-*********`
   This your private repository to get this part questions and upload your answers. **Copy this link to the text box below to be graded for this part.**

1. Open `Git Bash`, the app you downloaded in step `0`, and set your Email and username by:

   ```
   git config --global user.email “<your-GitHub-email>”
   git config --global user.name “<your-GitHub-username>”
   ```

2. Create a folder for the course on your computer and `cd` to that. `cd` means `Change Directory`. For example, on a Windows machine, where I have a folder on "C:\aps1070":

   ```
   cd c:aps1070
   ```

3. Get your assignment by the link you got in step 1:

   ```
   git clone https://github.com/APS1070-UofT/project-3-part-1-*********
   ```

4. A new folder should be created in your directory similar to:

   `C:\aps1070\project-3-part-1-********`
   This folder has an `ipynb` notebook which you need to manually upload to colab and answer its questions.

5. After you finished working on this notebook, download the notebook from colab and move it to the directory in step 5.

6. Replace the old notebook with the new one that has your answers. Make sure your completed notebook has the same name as the original notebook you downloaded.

7. To submit your work, follow:

   ```
   cd <your assignment folder>
   git add F23_Project_3_Part_1_git.ipynb
   git commit -m “Final Submission”
   git push
   ```
   If you have any problem with pushing your work on GitHub you can try one of following commands:

   ```
   git push --force
   or
   ```

`git push origin HEAD:main`

8. Make sure your submission is ready for grading. Open the private repository link in your browser and make sure you can see your final submission with your latest changes there. **Only you and the teaching team can open that link.**

assignment repository

https://github.com/APS1070-UofT/project-3-part-1-Yifan-Hung

Part 2: Applying PCA [2 Marks]

1. Compute the covariance matrix of the dataframe. *Hint: The dimensions of your covariance matrix should be (180, 180).* **[0.25]**
2. Write a function `get_sorted_eigen(df_cov)` that gets the covariance matrix of dataframe `df` (from step 1), and returns sorted eigenvalues and eigenvectors using `np.linalg.eigh`. **[0.25]**
3. Show the effectiveness of your principal components in covering the variance of the dataset with a `scree plot`. **[0.25]**
4. How many PCs do you need to cover 99% of the dataset's variance? **9 PCs are needed to cover 99% of the data's varience. [0.25]**
5. Plot the first 16 principal components (Eigenvectors) as a time series (16 subplots, on the x-axis you have dates and on the y-axis you have the value of the PC element) . **[0.5]**
6. Compare the first two PCs with the rest of them. Do you see any difference in their trend? **[0.5] In comparison, the first two pcs have a more regular pattern. The first few eigenvectors represents a certain trends of the data.This is because the latter pcs contains more noise.**

In [84]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

In [85]:
```
import pandas as pd
data_raw = pd.read_csv(
    filepath_or_buffer='https://raw.githubusercontent.com/Sabaae/Dataset/main/Temperatur
    index_col=0
)
```

In [86]:
```
#check the dataframe
data_raw
```

Out[86]:

City	1992-01-01	1992-02-01	1992-03-01	1992-04-01	1992-05-01	1992-06-01	1992-07-01	1992-08-01	1992-09-01	1992-10-01	...	2006-03-01	2006-04-01	2
A Coruña	7.053	9.548	11.154	12.322	15.951	15.312	19.686	18.823	16.474	12.558	...	11.524	13.307	15
Aachen	1.492	3.867	6.179	8.231	14.977	16.608	18.689	18.689	14.240	6.996	...	3.135	8.432	13
Aalborg	3.148	3.891	4.578	5.868	13.822	18.477	17.836	16.108	13.398	6.338	...	-0.591	6.229	11
Aba	26.090	28.293	28.755	28.334	27.305	26.121	25.289	25.287	25.681	25.933	...	29.319	28.716	27
Abadan	10.283	13.018	15.789	23.541	29.505	34.483	35.475	35.706	33.093	26.394	...	21.011	25.701	31
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Århus	3.148	3.891	4.578	5.868	13.822	18.477	17.836	16.108	13.398	6.338	...	-0.591	6.229	11
Çorlu	3.563	2.850	6.930	11.577	14.750	20.510	21.534	24.625	19.128	17.421	...	8.176	12.514	17
Çorum	-3.209	-2.546	3.744	9.299	12.780	17.550	18.765	21.050	15.027	12.990	...	6.441	10.458	14
Öskemen	-9.365	-12.908	-9.790	3.892	12.488	15.539	19.302	15.734	6.899	3.337	...	-4.790	3.547	1
Ürümqi	-12.445	-9.708	-1.082	11.655	16.550	21.596	23.253	20.733	12.910	6.112	...	-0.234	11.193	10

3448 rows × 180 columns

In [87]:
```
#standardize the data in dataframe
scaler = StandardScaler()
data_std = scaler.fit_transform(data_raw)
data_std = pd.DataFrame(data_std)
```

In [88]:
```
#the covarience matrix
cov_matrix = data_std.cov()
cov_matrix
```

Out[88]:

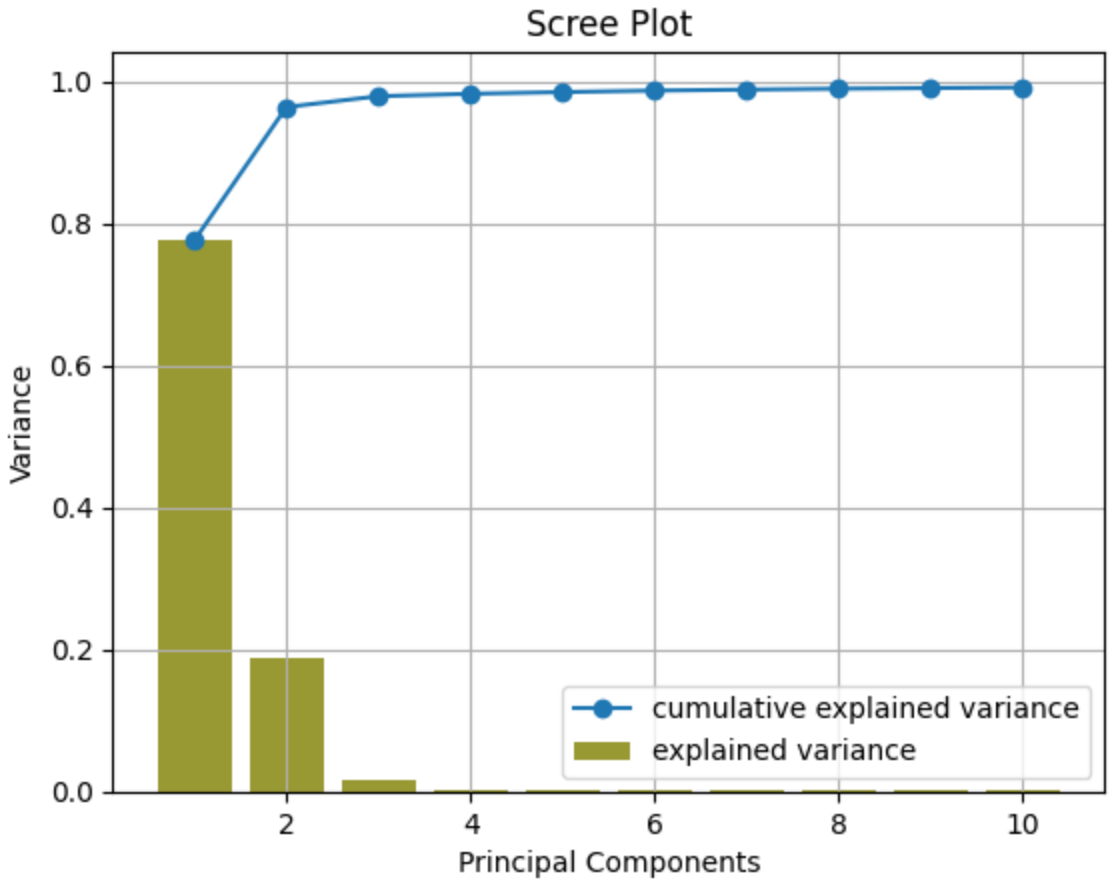
	0	1	2	3	4	5	6	7	8	9	...	
0	1.000290	0.992689	0.967891	0.879597	0.725127	0.494916	0.221077	0.288143	0.676715	0.880453	...	0.9
1	0.992689	1.000290	0.979553	0.901575	0.756487	0.527662	0.259778	0.315561	0.698849	0.889444	...	0.9
2	0.967891	0.979553	1.000290	0.953353	0.838400	0.631373	0.356323	0.402918	0.758520	0.922532	...	0.9
3	0.879597	0.901575	0.953353	1.000290	0.943871	0.800682	0.573898	0.596212	0.873071	0.955424	...	0.9
4	0.725127	0.756487	0.838400	0.943871	1.000290	0.930550	0.758156	0.747936	0.905713	0.891266	...	0.8
...	...	...	...	...	...	...	...	...	...	...	...	
175	0.270622	0.295929	0.379692	0.580734	0.710600	0.859851	0.945990	0.942821	0.849477	0.641363	...	0.4
176	0.655434	0.677469	0.744127	0.859897	0.920423	0.923898	0.850414	0.880087	0.977256	0.896314	...	0.7
177	0.868570	0.881036	0.911410	0.950932	0.905407	0.798641	0.623024	0.674717	0.924272	0.972031	...	0.9
178	0.967887	0.970153	0.966417	0.933824	0.825230	0.642957	0.413718	0.480100	0.814401	0.945799	...	0.9
179	0.992822	0.983977	0.962946	0.874866	0.731462	0.510398	0.238312	0.309715	0.690464	0.877376	...	0.9

180 rows × 180 columns

In [89]:
```
#function returns sorted eigenvalues and eigenvectors
def get_sorted_eigen(df_cov):
  #compute eigenvalues and eigenvectors
  eigenvalues, eigenvectors = np.linalg.eigh(df_cov)
  #sort in descending order
                                    envalues[::-1])
```

```
  eigenvalues = eigenvalues[sorted_index]
  eigenvectors = eigenvectors[:, sorted_index]
  return (eigenvalues, eigenvectors)
eigenvalues, eigenvectors = get_sorted_eigen(cov_matrix)
```

In [90]:
```
explained_variance = eigenvalues / np.sum(eigenvalues)
cum_variance = np.cumsum(explained_variance)
#scree plot
plt.bar([i for i in range(1, 11)], explained_variance[0:10], label='explained variance',
plt.plot([i for i in range(1, 11)], cum_variance[0:10], marker='o', label='cumulative ex
plt.grid()
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('Variance')
plt.legend()
plt.show()
```



In [91]:
```
#check how many PCs are needed to cover over 99% of the data's variance
for i in range (0,10):
  print('The cumulative varience is ' + str(cum_variance[i]*100) + '% for PC' + str(i+1)
```

```
The cumulative varience is 77.6268397128714% for PC1
The cumulative varience is 96.39971500997855% for PC2
The cumulative varience is 97.9311538583977% for PC3
The cumulative varience is 98.26840156820809% for PC4
The cumulative varience is 98.51119605990233% for PC5
The cumulative varience is 98.72003638089035% for PC6
The cumulative varience is 98.85719764768076% for PC7
The cumulative varience is 98.9837014441261% for PC8
The cumulative varience is 99.07822084457248% for PC9
The cumulative varience is 99.16159750026593% for PC10
```

In [92]:
```
eigenvectors_df = pd.DataFrame(eigenvectors)
eigenvectors_df
```

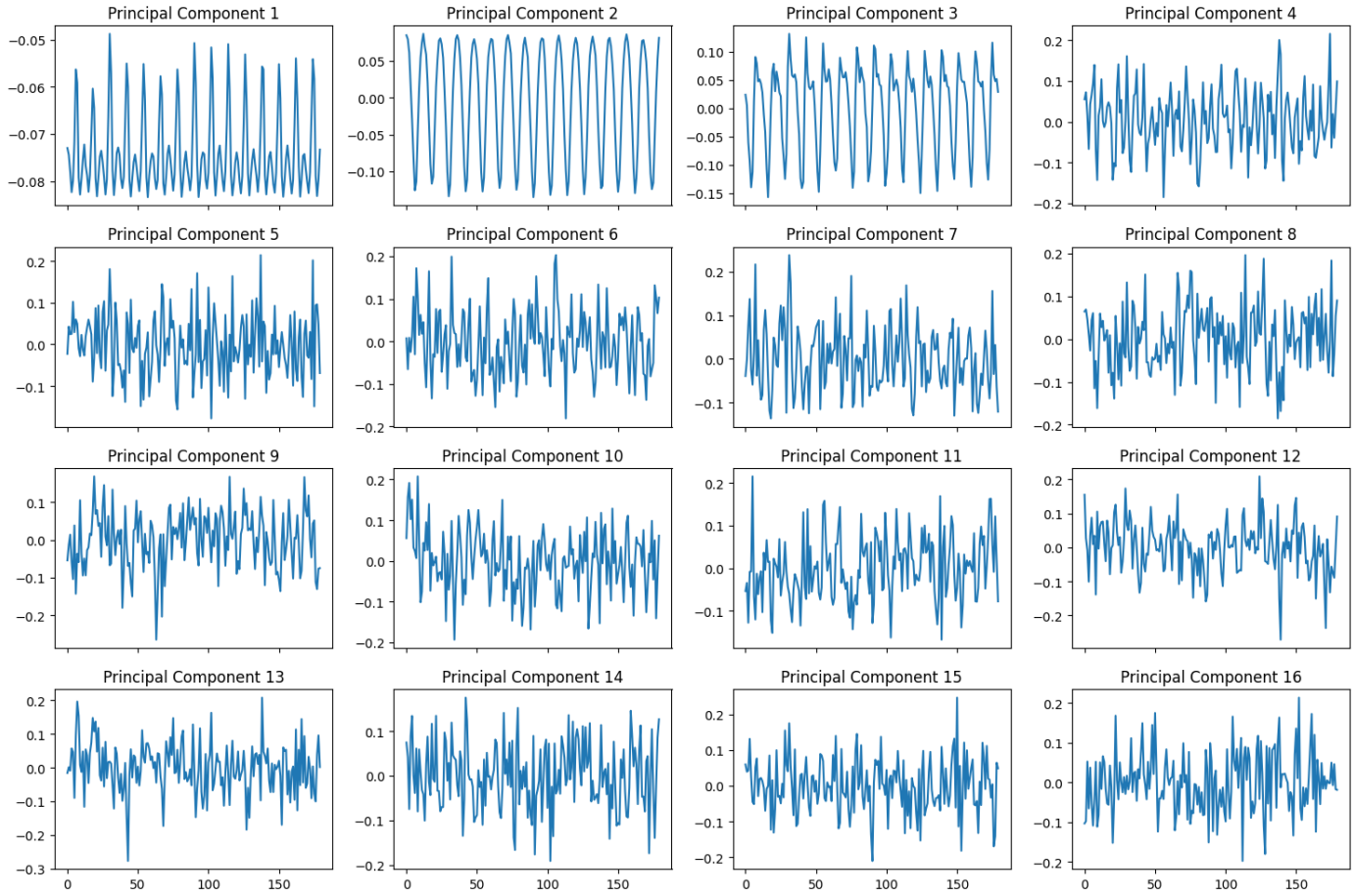
Out[92]:

	0	1	2	3	4	5	6	7	8	9
0	-0.072987	0.084422	0.024057	0.054976	-0.022836	0.007092	-0.039102	0.064410	-0.054374	0.055841
1	-0.074428	0.078845	0.005642	0.071981	0.042425	-0.065741	-0.002046	0.069130	-0.012981	0.157189
2	-0.078034	0.060746	-0.059711	0.015650	0.024127	0.008309	0.076832	0.045670	0.013909	0.191662
3	-0.082367	0.016739	-0.093610	-0.066883	0.025246	-0.025007	0.137429	0.011082	-0.058413	0.102452
4	-0.079972	-0.029733	-0.139857	0.043136	0.102165	0.011112	-0.025791	-0.026837	-0.104089	0.150637
...	...	...	...	...	...	...	...	...	...	...
175	-0.058237	-0.115572	0.116166	-0.063405	-0.148870	-0.050482	0.155827	0.183545	0.051915	-0.045686
176	-0.078886	-0.052892	0.059558	0.018598	0.093686	0.131657	-0.034497	-0.086759	-0.111766	0.050396
177	-0.083191	0.007887	0.048053	-0.039539	0.095908	0.103026	0.032207	-0.033440	-0.130837	-0.141285
178	-0.079772	0.051838	0.051952	0.029141	0.048180	0.066110	-0.062648	0.054147	-0.076395	-0.032876
179	-0.073308	0.080917	0.029073	0.098774	-0.068896	0.102801	-0.120649	0.090062	-0.074937	0.061841

180 rows × 180 columns

In [93]:

```
# Plot the first 16 principal components as time series
fig, axes = plt.subplots(4, 4, figsize=(15, 10), sharex=True)
for i in range(4):
    for j in range(4):
        index = i * 4 + j + 1
        axes[i, j].plot(eigenvectors_df.iloc[:, index-1])
        axes[i, j].set_title('Principal Component ' + str(index))
plt.tight_layout()
plt.show()
```



Part 3: Data reconstruction [3 Marks]

Create a function that:

- Accepts a city and the original dataset as inputs.
- Calls useful functions that you designed in previous parts to compute eigenvectors and eigenvalues.
- Plots 4 figures:

    1. The original time-series for the specified city. **[0.5]**
    2. The incremental reconstruction of the **original** (not standardized) time-series for the specified city in a single plot. **[1.5]**

        - You should at least show 5 curves in a figure for incremental reconstruction. For example, you can pick the following (or any other combination that you think is reasonable):

            - Reconstruction with only PC1
            - Reconstruction with both PC1 and PC2
            - Reconstruction with PC1 to PC4 (First 4 PCs)
            - Reconstruction with PC1 to PC8 (First 8 PCs)
            - Reconstruction with PC1 to PC16 (First 16 PCs)
        - Hint: you need to compute the reconstruction for the standardized time-series first, and then scale it back to the original (non-standardized form) using the StandardScaler `inverse_transform` help...

    3. The residual error for your best reconstruction with respect to the original time-series. **[0.5]**
        - Hint: You are plotting the error that we have for reconstructing each month `(df - df_reconstructed)`. On the x-axis, you have dates, and on the y-axis, the residual error.
    4. The RMSE of the reconstruction as a function of the number of included components (x-axis is the number of components and y-axis is the RMSE). Sweep x-axis from 1 to 10 (this part is independent from part 3.2.) **[1]**

Test your function using the `Yakeshi`, `Zamboanga`, `Norilsk`, `Juliaca`, and `Doha` as inputs. **[0.5]**

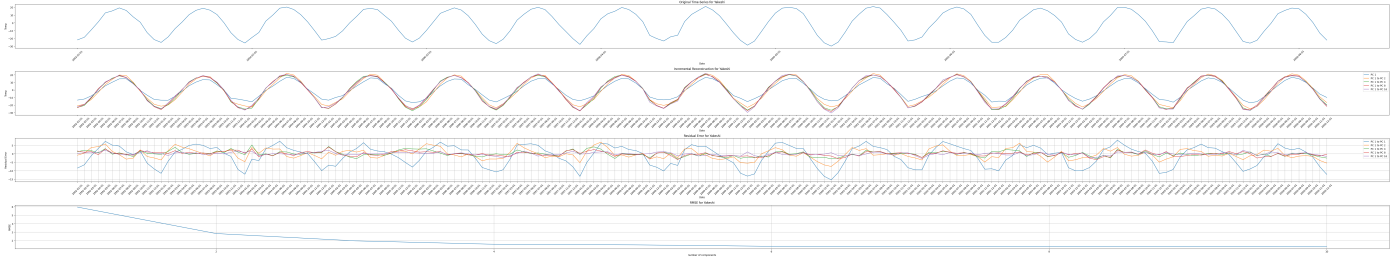
In [94]:
```
def plot_city_figures(original_df, city_name):
  print('---------PCA---------')
  ##1
  #plot the original time-series
  plt.subplot(4, 1, 1)
  original_df.loc[city_name,:].plot(figsize=(80,15))
  plt.title('Original Time-Series for '+ city_name)
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Temp')

  ##2
  #the incremental reconstruction of the original time-series
  #standarization
  plt.subplot(4, 1, 2)
  scaler = StandardScaler()
  data_std = scaler.fit_transform(original_df)
  data_std = pd.DataFrame(data_std, columns = original_df.columns, index = original_df.i
  #cov matrix
  cov_matrix = data_std.cov()
```

```
  eigenvalues, eigenvectors = get_sorted_eigen(cov_matrix)
  #PC1
  rmse = []
  re = []
  W = eigenvectors[:, 0:1]
  project = np.dot(data_std.loc[city_name,:].values, eigenvectors[:, 0:1])
  reconstruction = scaler.inverse_transform(np.dot(project.reshape(-1, 1), W.T))
  plt.plot(original_df.columns, reconstruction.reshape(data_std.shape[1]), label='PC 1')
  re.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[1]
  rmse.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[
  #PC1 and PC2 / PC1 to PC4 / PC1 to PC8 / PC1 to PC16
  pcs = [2, 4, 8, 16]
  for pc in pcs:
    W = eigenvectors[:, 0:pc]
    project = np.dot(data_std.loc[city_name,:].values.reshape(1, -1), eigenvectors[:, 0:
    reconstruction = scaler.inverse_transform(np.dot(project, W.T))
    plt.plot(original_df.columns, reconstruction.reshape(data_std.shape[1]), label='PC 1
    plt.legend()
    re.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Temp')
  plt.title('Incremental Reconstruction for '+ city_name)
  #for rmse PC2 to PC9
  for i in range(2, 16):
    W = eigenvectors[:, 0:i]
    project = np.dot(data_std.loc[city_name,:].values.reshape(1, -1), eigenvectors[:, 0:
    reconstruction = scaler.inverse_transform(np.dot(project, W.T))
    rmse.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shap
  ##3
  plt.subplot(4, 1, 3)
  pcs = [1, 2, 4, 8, 16]
  for i in range(0, 5):
    plt.plot(original_df.columns, re[i], label='PC 1 to PC ' + str(pcs[i]))
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Residual Error')
  plt.title('Residual Error for '+ city_name)
  plt.grid()
  plt.legend()
  ##4
  temp = [] #(y - y')^2
  rmse_plt = []
  # formula for root mean square error sqrt(sum((y - y')^2)/N)
  plt.subplot(4, 1, 4)
  for i in range(0, 10):
    temp.append(np.power(rmse[i], 2))
    rmse_plt.append(np.sqrt(np.sum(temp[i])/len(temp[i])))
  plt.plot([i for i in range(1, 11)],rmse_plt)
  plt.xlabel('number of components')
  plt.ylabel('RMSE')
  plt.title('RMSE for '+ city_name)
  plt.grid()
  plt.tight_layout()
  plt.show()
  return rmse_plt
```

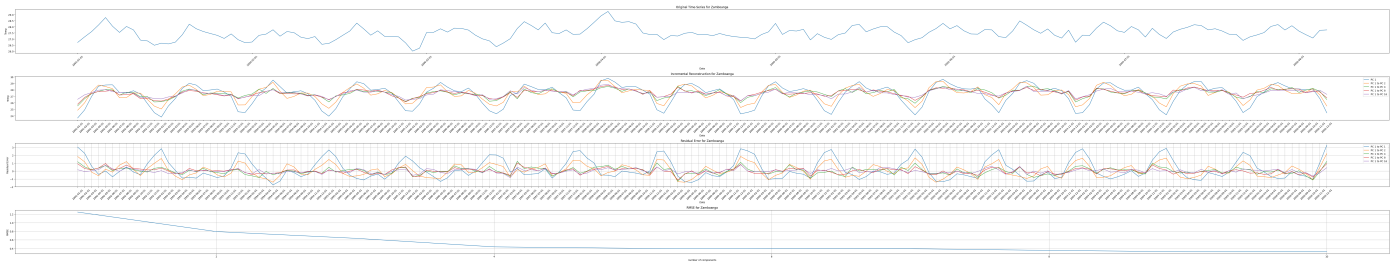
In [95]:

```
#test function using Yakeshi, Zamboanga, Norilsk, Juliaca, and Doha as input
cities = ['Yakeshi', 'Zamboanga', 'Norilsk', 'Juliaca', 'Doha']
for city in cities:
  plot_city_figures(data_raw, city)
```

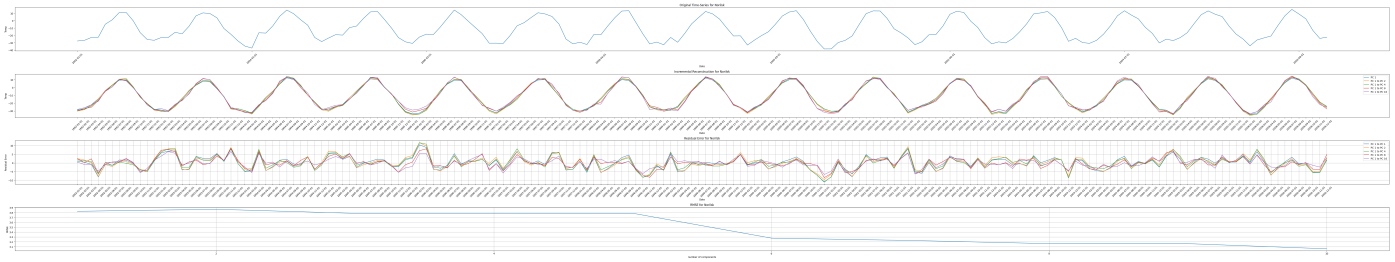
```
---------PCA---------
```



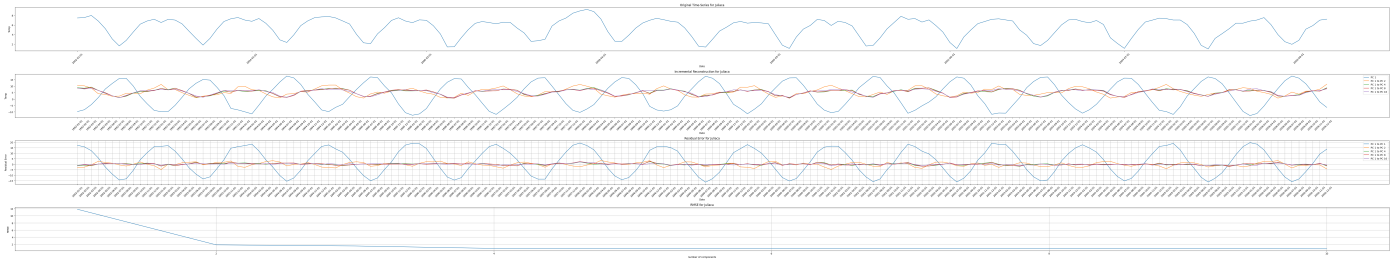
---------PCA---------



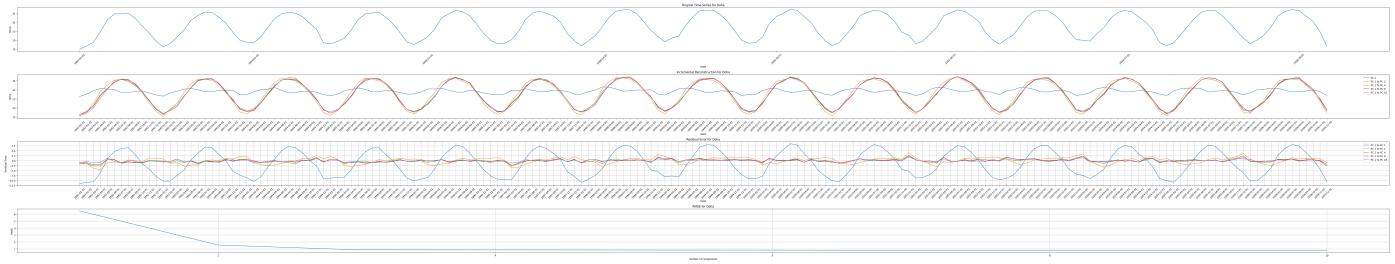
---------PCA---------



---------PCA---------



---------PCA---------



Part 4: SVD [2 Marks]

Modify your code in part 3 to use SVD instead of PCA for extracting the eigenvectors. **[1]**

Explain if standardization or covariance computation is required for this part. Repeat part 3 and compare your PCA and SVD results. Write a function to make this comparison **[0.5]**, and comment on the results. **[0.5]**.

If standardization or covariance computation required: Standardization or covariance computation is not a strict requirement for SVD. However, standarization may be beneficial to ensure that all features are on a similar scale.Also, it might be helpful if the ultimate goal dimensionality reduction or obtaining principal components.

Comparison: As shown in the figure, we compared the RMSE we got from SVD and PCA. As a result, the curves have no significant difference. For PCA, it computes eigenvectors of the covariance matrix. As to SVD, it provides right singular vectors that are related to the eigenvectors. While the values may not match exactly due to differences in normalization and scaling, the directions are often very similar. Moverover, here we have chosen to standarlized the dataset while using both PCA and SVD. Therefore, the result was pretty similiar.

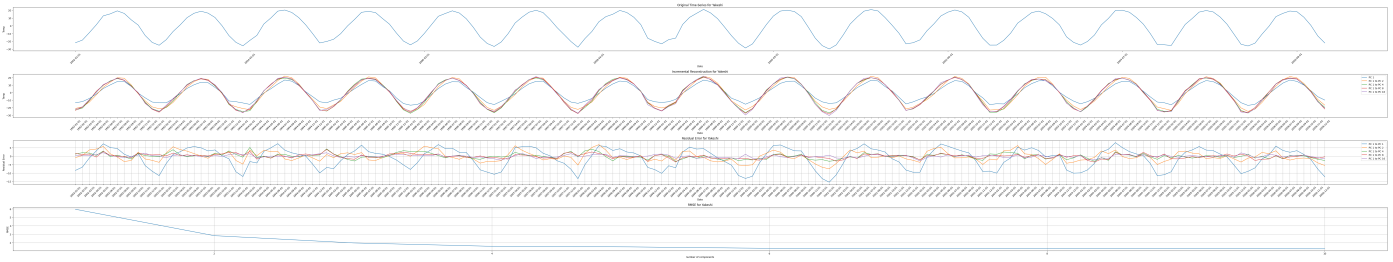
In [96]:
```
def svd(original_df, city_name):
  print('---------SVD---------')
  #center the data
  scaler = StandardScaler()
  data_std = scaler.fit_transform(original_df)
  data_std = pd.DataFrame(data_std, columns = original_df.columns, index = original_df.i
  #perform svd
  U, S, Vt = np.linalg.svd(data_std, full_matrices=False)
  eigenvectors = Vt.T
  eigenvalues = S * S

  ##1
  #plot the original time-series
  plt.subplot(4, 1, 1)
  original_df.loc[city_name,:].plot(figsize=(80,15))
  plt.title('Original Time-Series for '+ city_name)
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Temp')

  ##2
  plt.subplot(4, 1, 2)
  #the incremental reconstruction of the original time-series
  #PC1
  rmse = []
  re = []
  W = eigenvectors[:, 0:1]
  project = np.dot(data_std.loc[city_name,:].values, eigenvectors[:, 0:1])
  reconstruction = scaler.inverse_transform(np.dot(project.reshape(-1, 1), W.T))
  plt.plot(original_df.columns, reconstruction.reshape(data_std.shape[1]), label='PC 1')
  re.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[1]
  rmse.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[
  #PC1 and PC2 / PC1 to PC4 / PC1 to PC8 / PC1 to PC16
  pcs = [2, 4, 8, 16]
  for pc in pcs:
    W = eigenvectors[:, 0:pc]
    project = np.dot(data_std.loc[city_name,:].values.reshape(1, -1), eigenvectors[:, 0:
    reconstruction = scaler.inverse_transform(np.dot(project, W.T))
    plt.plot(original_df.columns, reconstruction.reshape(data_std.shape[1]), label='PC 1
    plt.legend()
    re.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shape[
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Temp')
  plt.title('Incremental Reconstruction for '+ city_name)
  #for rmse PC2 to PC9
  for i in range(2, 16):
    W = eigenvectors[:, 0:i]
    project = np.dot(data_std.loc[city_name,:].values.reshape(1, -1), eigenvectors[:, 0:
    reconstruction = scaler.inverse_transform(np.dot(project, W.T))
    rmse.append(original_df.loc[city_name,:].values-reconstruction.reshape(data_std.shap
  ##3
  plt.subplot(4, 1, 3)
  pcs = [1, 2, 4, 8, 16]
```

```
    plt.plot(original_df.columns, re[i], label='PC 1 to PC ' + str(pcs[i]))
  plt.xlabel('Date')
  plt.xticks(rotation=45)
  plt.ylabel('Residual Error')
  plt.title('Residual Error for '+ city_name)
  plt.grid()
  plt.legend()
  ##4
  temp = [] #(y - y')^2
  rmse_plt = []
  # formula for root mean square error sqrt(sum((y - y')^2)/N)
  plt.subplot(4, 1, 4)
  for i in range(0, 10):
    temp.append(np.power(rmse[i], 2))
    rmse_plt.append(np.sqrt(np.sum(temp[i])/len(temp[i])))
  plt.plot([i for i in range(1, 11)],rmse_plt)
  plt.xlabel('number of components')
  plt.ylabel('RMSE')
  plt.title('RMSE for '+ city_name)
  plt.grid()
  plt.tight_layout()
  plt.show()
  return rmse_plt
```

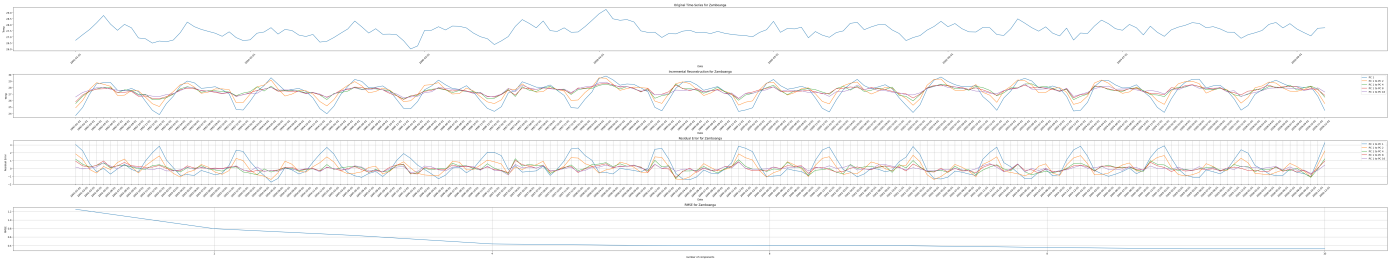
In [97]:

```
for city in cities:
  svd(data_raw, city)
```

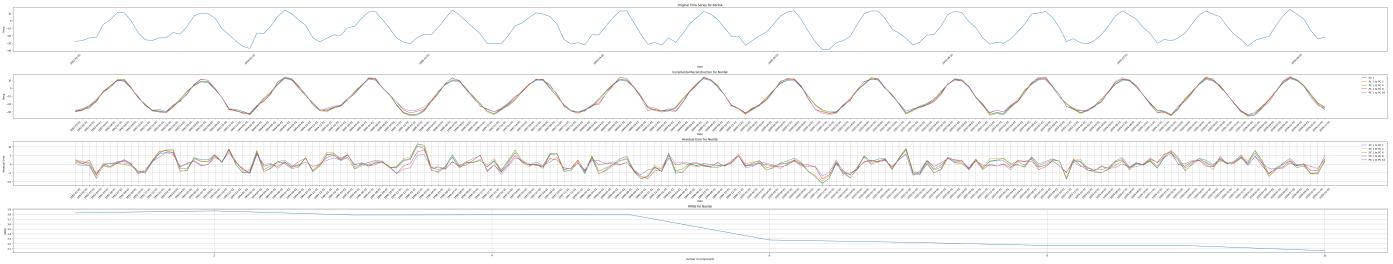
```
---------SVD---------
```



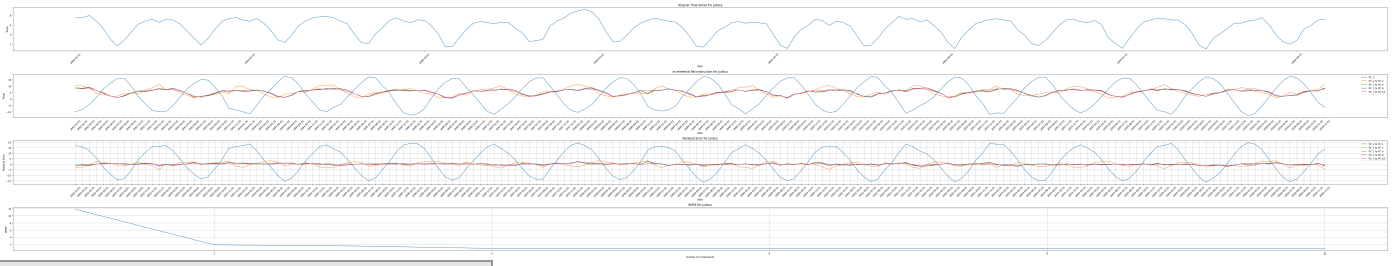
```
---------SVD---------
```

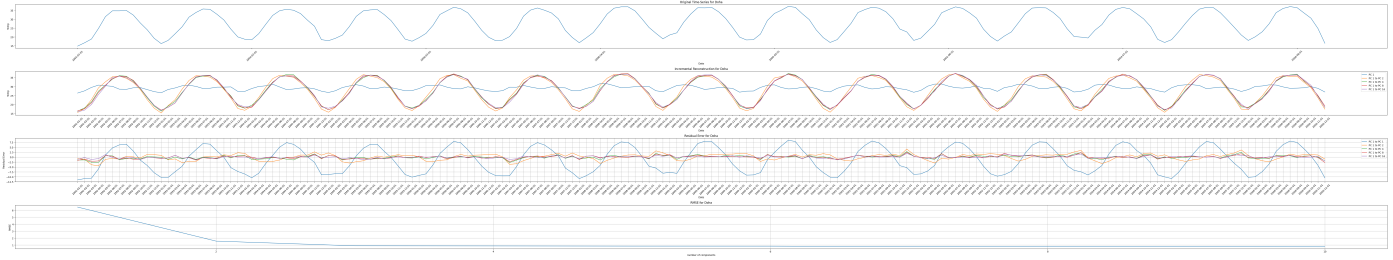


```
---------SVD---------
```



```
---------SVD---------
```

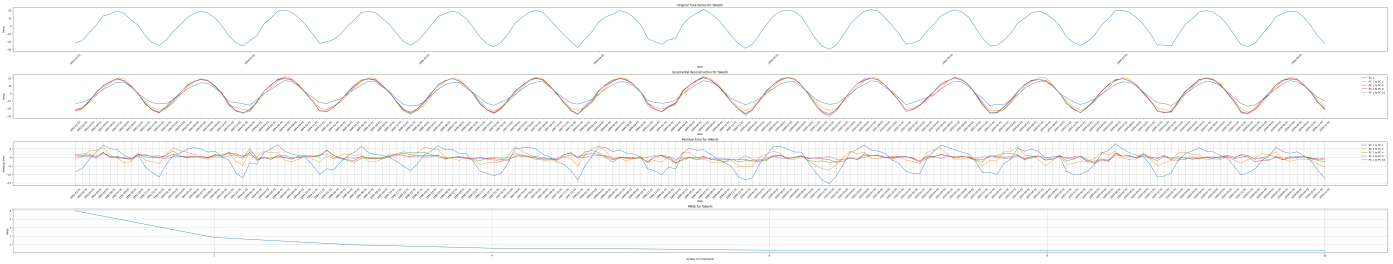




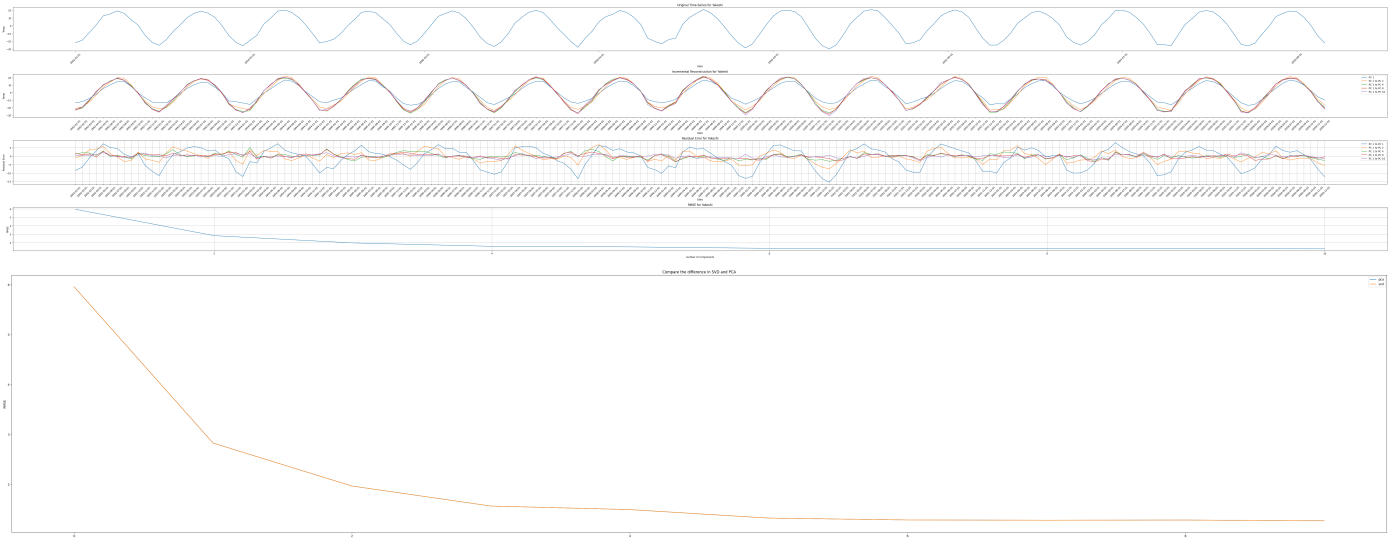
In [98]:
```
def compare_pca_svd(original_df, city_name):
  pca_rmse = plot_city_figures(original_df, city_name)
  svd_rmse = svd(original_df, city_name)
  plt.figure(figsize=(80,15))
  plt.plot(pca_rmse, label='pca')
  plt.plot(svd_rmse, label='svd')
  plt.title('Compare the difference in SVD and PCA')
  plt.ylabel('RMSE')
  plt.legend()
  plt.show()
```

In [99]:
```
for city in cities:
  compare_pca_svd(data_raw, city)
```

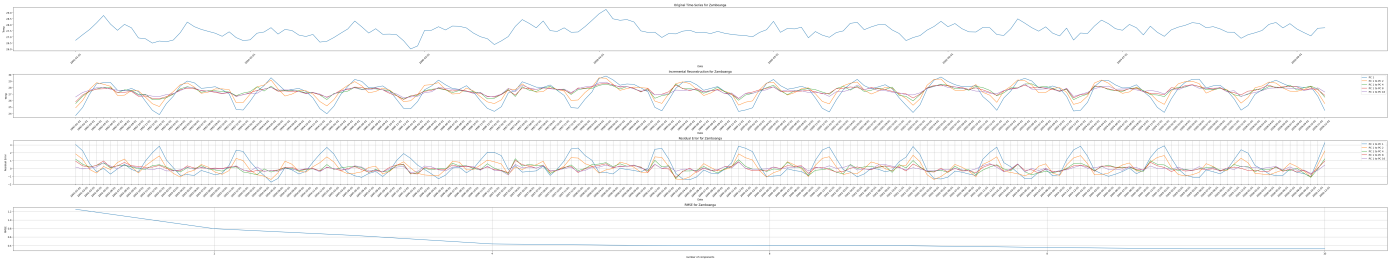
---------PCA---------



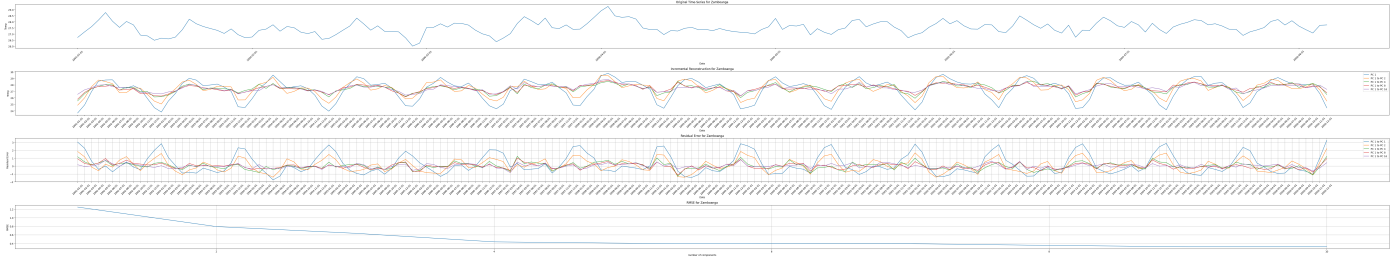
---------SVD---------



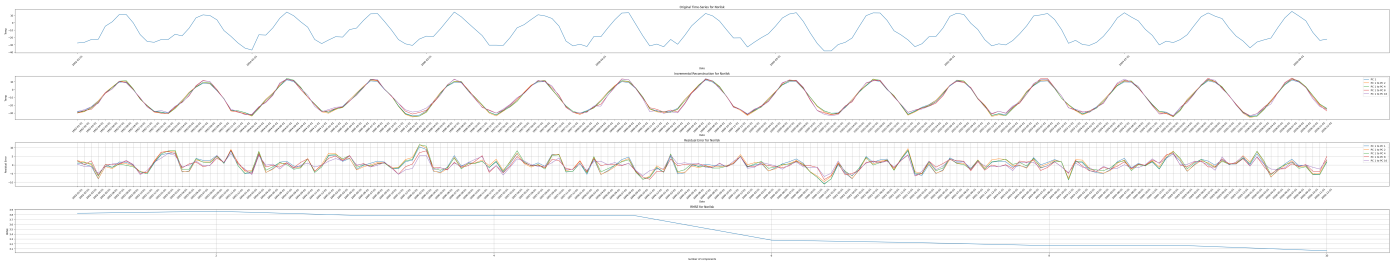
---------PCA---------



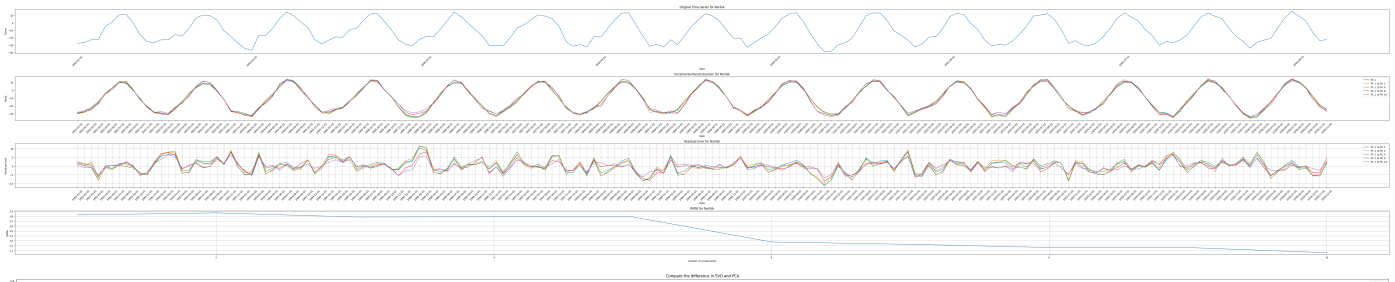
---------SVD---------



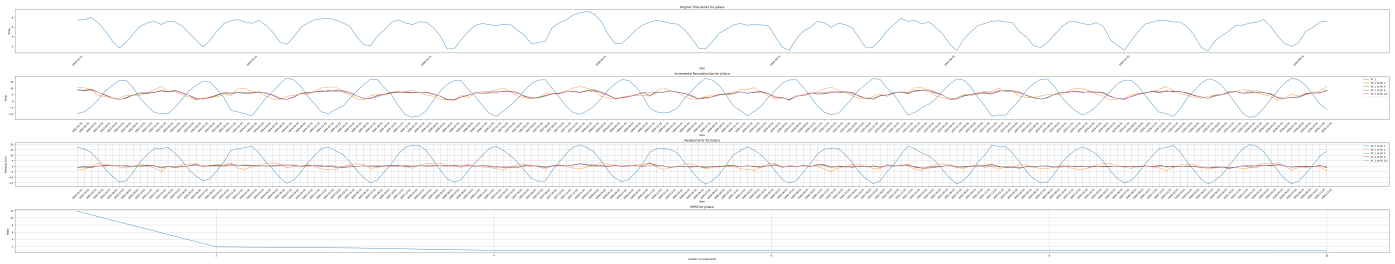
---------PCA---------



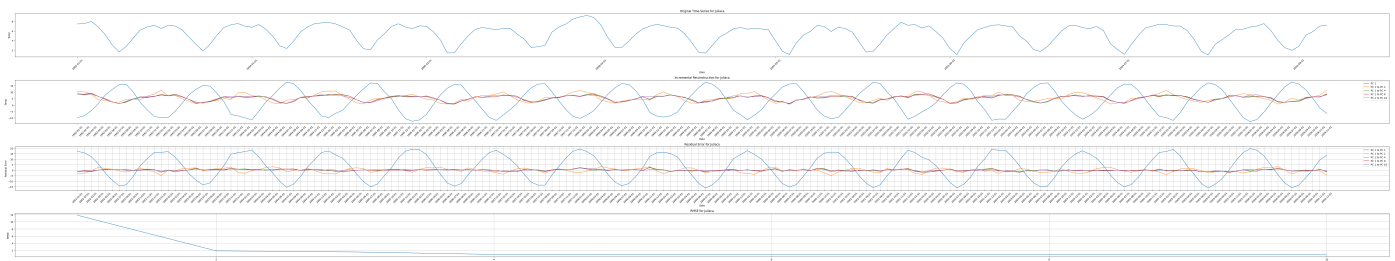
---------SVD---------

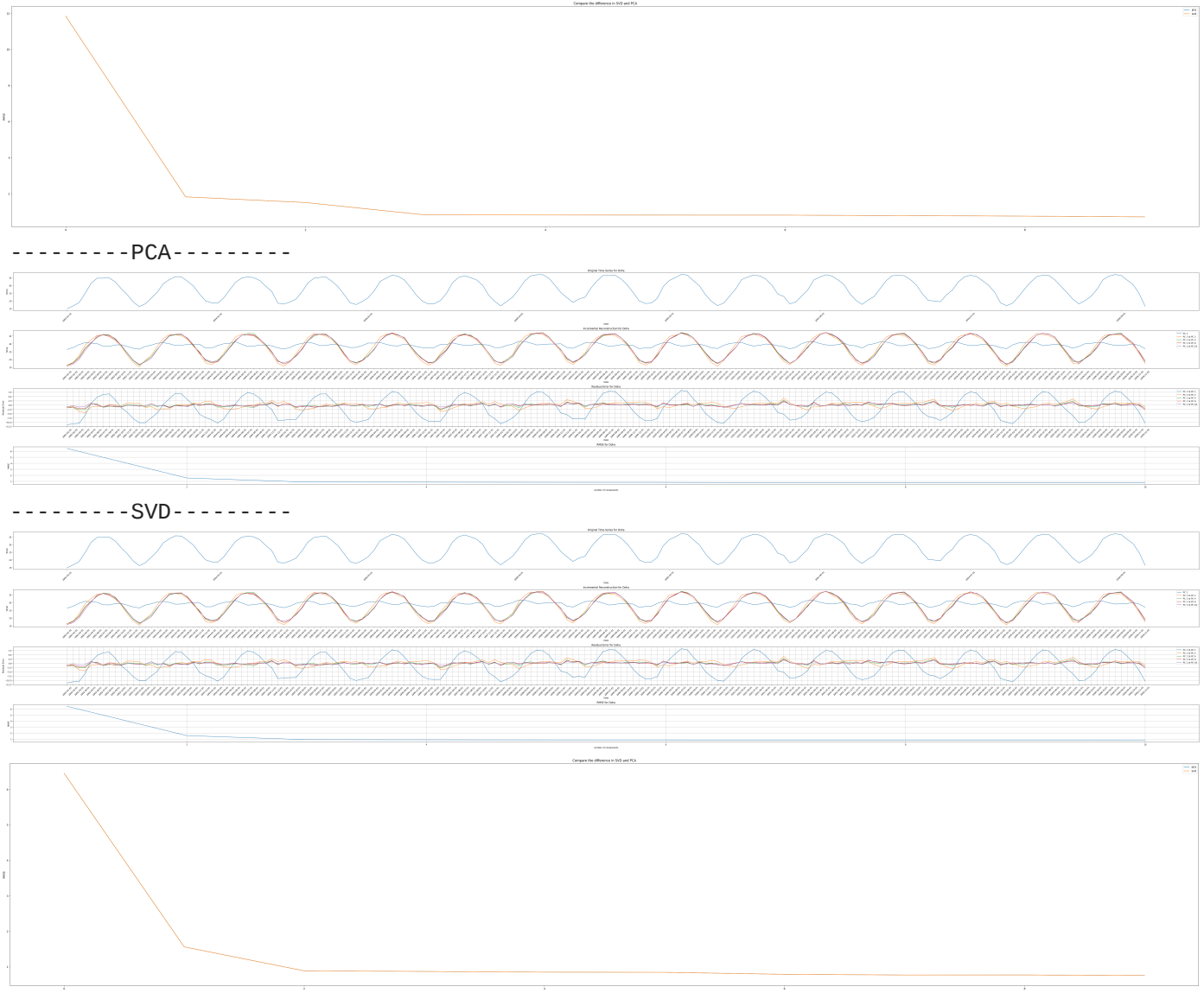


---------PCA---------



---------SVD---------





---------PCA---------

---------SVD---------

Part 5: Let's collect another dataset! [2 Marks]

Create another dataset similar to the one provided in your handout using the raw information on average temperatures per states (not cities) provided here. **[1]**

You need to manipulate the data to organize it in the desired format (i.e., the same format that was in previous parts). If there is a missing value for the average temperature of a particular state at a given date, make sure to remove that date completely from the dataset, even if the data of that specific date exists for other states.

You are free to use any tools you like, from Excel to Python! In the end, you should have a new CSV file with more dates (features) compared to the provided dataset. How many features does the final dataset have? How many states are there? **There are 241 states and 608 features.**

Upload your new dataset (in CSV format) to your colab notebook and repeat part 4. When analyzing the states, you may use `Jilin`, `Nunavut`, `Rio Grande Do Norte`, `Louisiana`, and `Tasmania`. **[1]**

The code below helps you to upload your new CSV file to your colab session.

In [100…

```
# load train.csv to Google Colab
from google.colab import files
```

```
uploaded = files.upload()
#the original file downloaded from 'https://github.com/Sabaae/Dataset/blob/main/Temperat
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving TemperaturesbyState.csv to TemperaturesbyState (2).csv
```

In [101…

```
#visualize the dataframe
data = pd.read_csv('/content/TemperaturesbyState.csv')
data
```

Out[101]:

	dt	AverageTemperature	AverageTemperatureUncertainty	State	Country
0	1963-01-01	26.037	0.315	Acre	Brazil
1	1963-01-01	-1.260	0.551	Adygey	Russia
2	1963-01-01	-22.297	0.563	Aga Buryat	Russia
3	1963-01-01	4.234	0.134	Alabama	United States
4	1963-01-01	26.389	0.540	Alagoas	Brazil
...	...	...	...	...	...
146764	2013-09-01	NaN	NaN	Yaroslavl'	Russia
146765	2013-09-01	NaN	NaN	Yevrey	Russia
146766	2013-09-01	5.267	4.786	Yukon	Canada
146767	2013-09-01	NaN	NaN	Yunnan	China
146768	2013-09-01	NaN	NaN	Zhejiang	China

146769 rows × 5 columns

In [102…

```
a = data[data['dt']=='1963-01-01']
a
```

Out[102]:

	dt	AverageTemperature	AverageTemperatureUncertainty	State	Country
0	1963-01-01	26.037	0.315	Acre	Brazil
1	1963-01-01	-1.260	0.551	Adygey	Russia
2	1963-01-01	-22.297	0.563	Aga Buryat	Russia
3	1963-01-01	4.234	0.134	Alabama	United States
4	1963-01-01	26.389	0.540	Alagoas	Brazil
...	...	...	...	...	...
236	1963-01-01	-18.650	0.213	Yaroslavl'	Russia
237	1963-01-01	-20.697	0.595	Yevrey	Russia
238	1963-01-01	-24.129	0.559	Yukon	Canada
239	1963-01-01	7.286	0.220	Yunnan	China
240	1963-01-01	2.765	0.312	Zhejiang	China

241 rows × 5 columns

In [103…

```
b = data[data['dt']=='1963-02-01']
b
```

Out[103]:

	dt	AverageTemperature	AverageTemperatureUncertainty	State	Country
241	1963-02-01	25.659	0.341	Acre	Brazil
242	1963-02-01	2.380	0.581	Adygey	Russia
243	1963-02-01	-16.490	0.385	Aga Buryat	Russia
244	1963-02-01	5.685	0.158	Alabama	United States
245	1963-02-01	26.215	0.739	Alagoas	Brazil
...	...	...	...	...	...
477	1963-02-01	-12.679	0.315	Yaroslavl'	Russia
478	1963-02-01	-17.472	0.306	Yevrey	Russia
479	1963-02-01	-19.783	0.519	Yukon	Canada
480	1963-02-01	10.041	0.229	Yunnan	China
481	1963-02-01	5.534	0.242	Zhejiang	China

241 rows × 5 columns

In [104…

```
#get the unique states
state = b['State']
state
```

Out[104]:

```
241          Acre
242        Adygey
243    Aga Buryat
244       Alabama
245       Alagoas
           ...
477    Yaroslavl'
478        Yevrey
479         Yukon
480        Yunnan
481      Zhejiang
Name: State, Length: 241, dtype: object
```

In [105…

```
#get the unique dates
date = data['dt'].unique()
```

In [106…

```
temp = []
for d in date:
  temp.append(data[data['dt']==d]['AverageTemperature'])
temp = np.array(temp)
```

In [107…

```
#put the corresponding date, state, and average temperature in a dataframe
df = {i: t for i, t in zip(date, temp)}
df = pd.DataFrame(df)
df.columns = date
df.index = state
df
```

Out[107]:

State	1963-01-01	1963-02-01	1963-03-01	1963-04-01	1963-05-01	1963-06-01	1963-07-01	1963-08-01	1963-09-01	1963-10-01	...	2012-12-01	2013-01-01
Acre	26.037	25.659	25.506	26.019	25.675	25.097	25.382	27.432	27.706	27.393	...	26.318	26.587
Adygey	-1.260	2.380	1.506	8.992	15.204	17.843	22.220	21.487	17.756	10.716	...	0.191	0.621
Aga Buryat	-22.297	-16.490	-6.900	-1.146	8.558	13.960	17.899	16.316	8.011	-1.363	...	-26.094	-26.128
Alabama	4.234	5.685	14.853	18.815	22.172	25.254	25.863	26.514	23.083	18.871	...	10.822	10.284
Alagoas	26.389	26.215	25.830	25.340	24.060	22.792	22.103	22.641	23.853	25.479	...	27.021	27.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Yaroslavl'	-18.650	-12.679	-12.197	2.414	15.048	12.505	17.616	16.227	12.858	4.163	...	-11.730	-10.928
Yevrey	-20.697	-17.472	-6.148	2.557	10.782	16.184	20.655	18.919	11.894	2.443	...	-22.980	-24.428
Yukon	-24.129	-19.783	-17.878	-6.328	3.664	7.780	11.404	10.530	3.584	-4.766	...	-28.727	-22.899
Yunnan	7.286	10.041	13.768	17.398	20.858	20.698	21.497	20.927	20.737	15.778	...	10.254	9.053
Zhejiang	2.765	5.534	11.229	16.736	22.544	23.874	27.800	27.806	24.726	17.307	...	6.633	5.120

241 rows × 609 columns

In [108…
```
#drop a column if it contains a nan value
df.dropna(axis=1, inplace=True)
```

In [109…
```
#visualize the datafram
df
```

Out[109]:

State	1963-01-01	1963-02-01	1963-03-01	1963-04-01	1963-05-01	1963-06-01	1963-07-01	1963-08-01	1963-09-01	1963-10-01	...	2012-11-01	2012-12-01
Acre	26.037	25.659	25.506	26.019	25.675	25.097	25.382	27.432	27.706	27.393	...	27.393	26.318
Adygey	-1.260	2.380	1.506	8.992	15.204	17.843	22.220	21.487	17.756	10.716	...	7.324	0.191
Aga Buryat	-22.297	-16.490	-6.900	-1.146	8.558	13.960	17.899	16.316	8.011	-1.363	...	-12.900	-26.094
Alabama	4.234	5.685	14.853	18.815	22.172	25.254	25.863	26.514	23.083	18.871	...	11.216	10.822
Alagoas	26.389	26.215	25.830	25.340	24.060	22.792	22.103	22.641	23.853	25.479	...	27.074	27.021
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Yaroslavl'	-18.650	-12.679	-12.197	2.414	15.048	12.505	17.616	16.227	12.858	4.163	...	-0.254	-11.730
Yevrey	-20.697	-17.472	-6.148	2.557	10.782	16.184	20.655	18.919	11.894	2.443	...	-8.168	-22.980
Yukon	-24.129	-19.783	-17.878	-6.328	3.664	7.780	11.404	10.530	3.584	-4.766	...	-22.696	-28.727
Yunnan	7.286	10.041	13.768	17.398	20.858	20.698	21.497	20.927	20.737	15.778	...	14.213	10.254
Zhejiang	2.765	5.534	11.229	16.736	22.544	23.874	27.800	27.806	24.726	17.307	...	12.306	6.633

241 rows × 608 columns

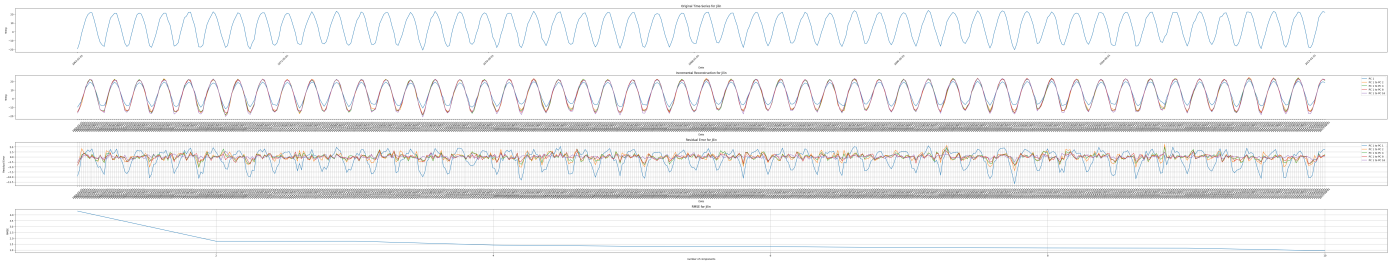
In [110…
```
print('There are '+str(df.shape[0])+' states and '+str(df.shape[1])+' features.')
```

```
There are 241 states and 608 features.
```

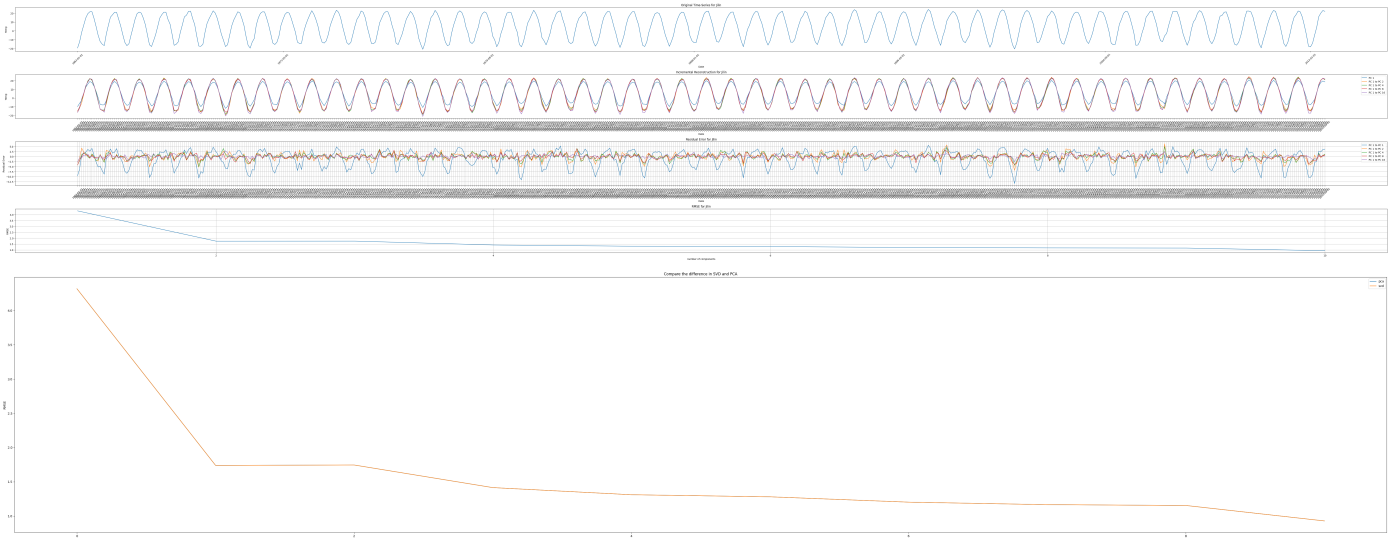
In [111…

```
#analyse the states
c = ['Jilin', 'Nunavut', 'Rio Grande Do Norte', 'Louisiana', 'Tasmania']
for city in c:
  compare_pca_svd(df, city)
```

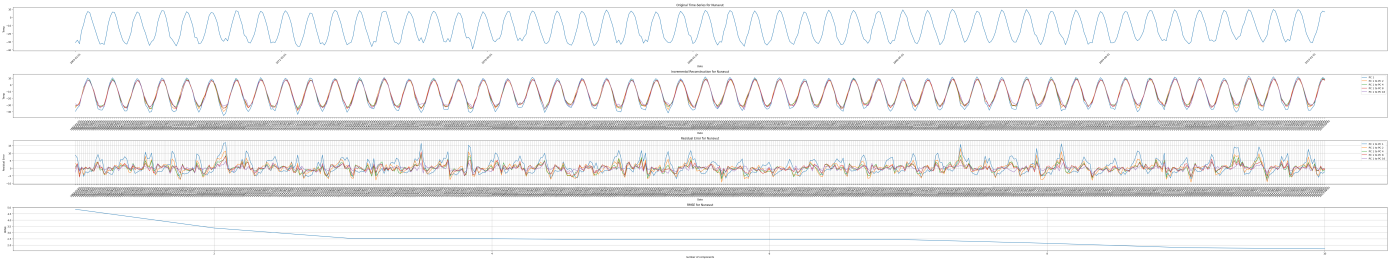
---------PCA---------



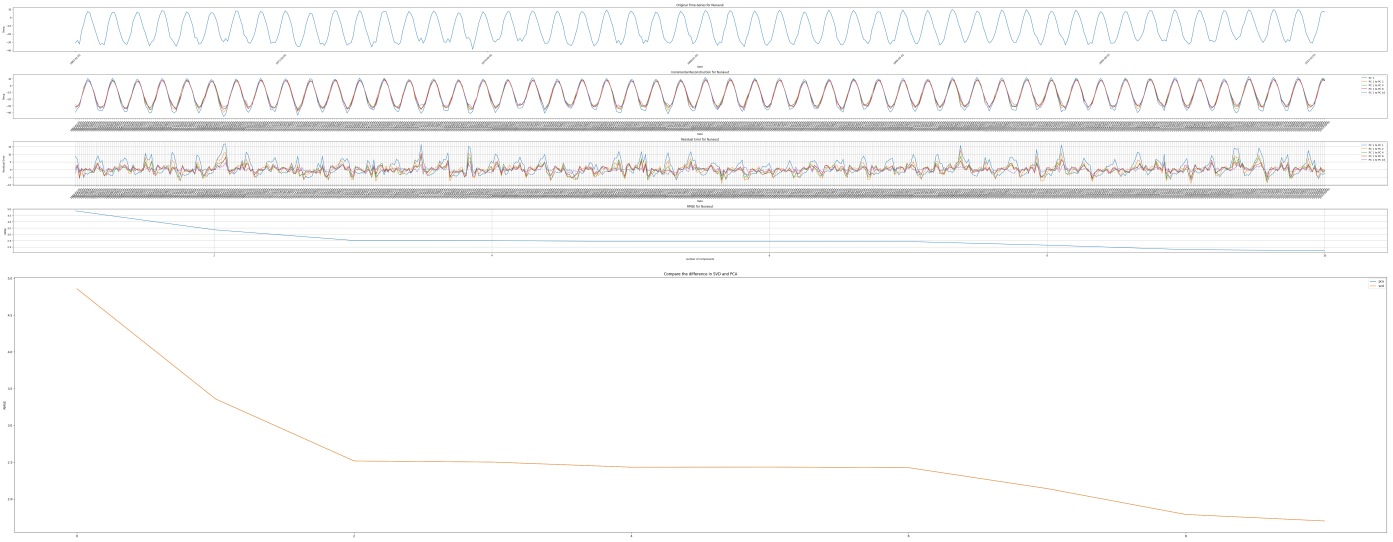
---------SVD---------



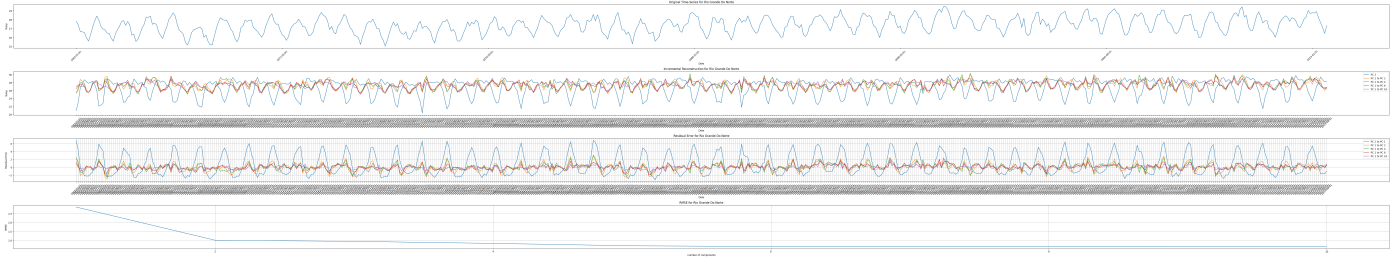
---------PCA---------



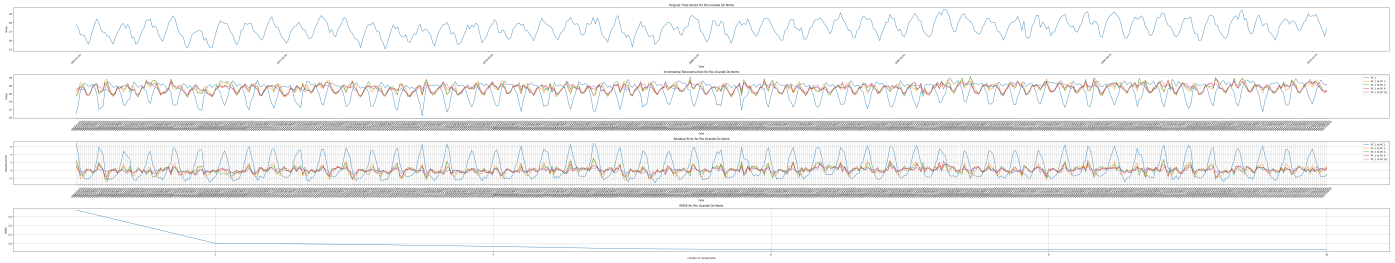
---------SVD---------

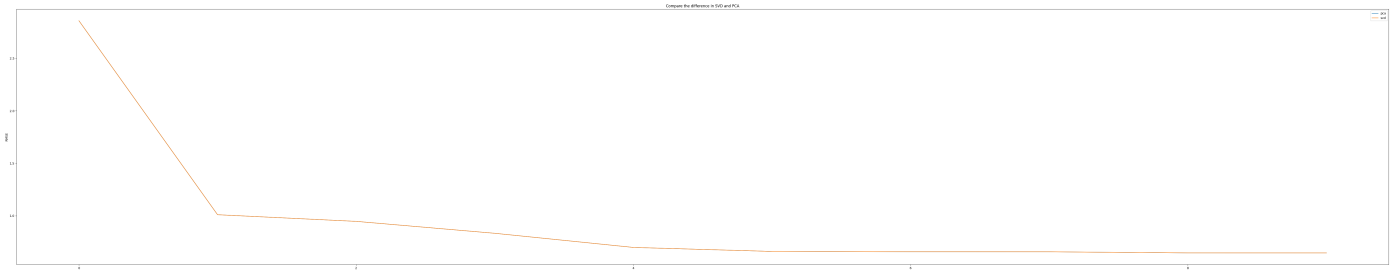


---------PCA---------

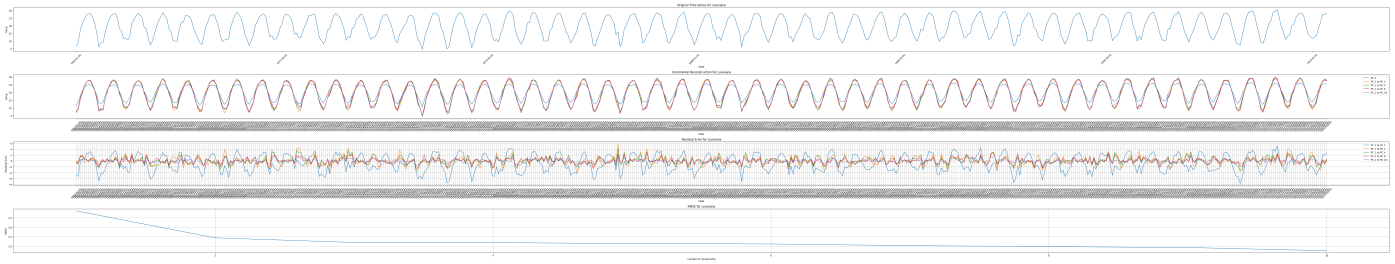


---------SVD---------

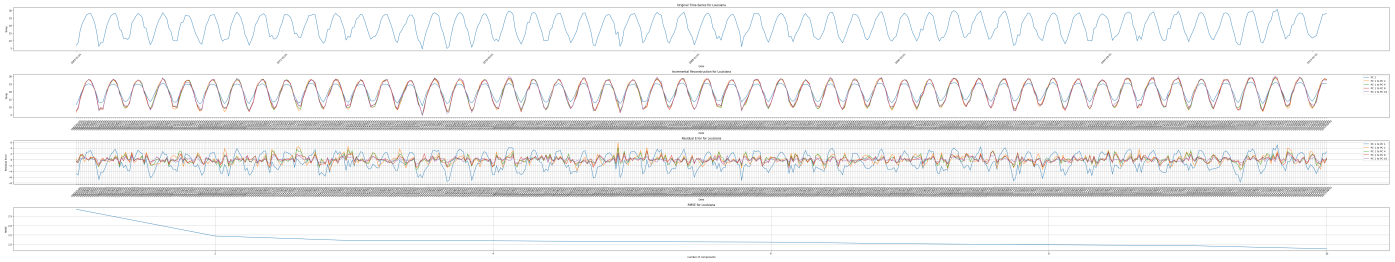


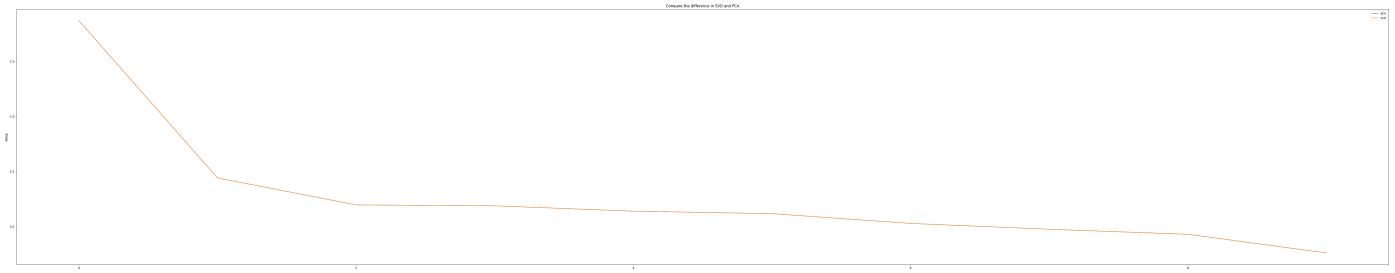


---------PCA---------

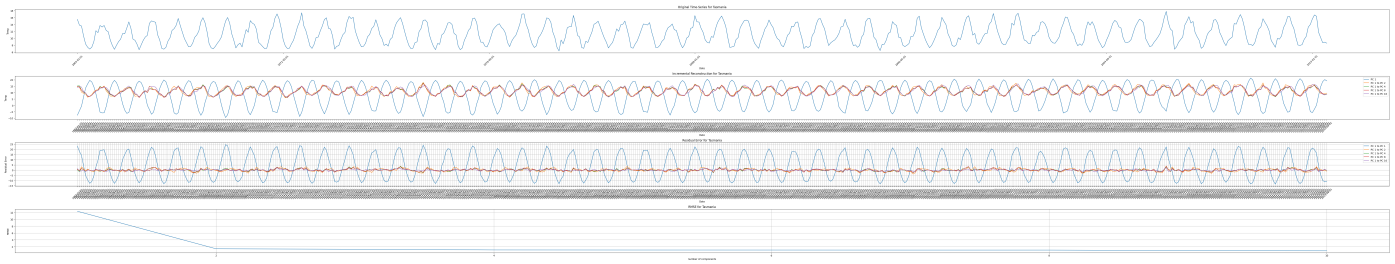


---------SVD---------

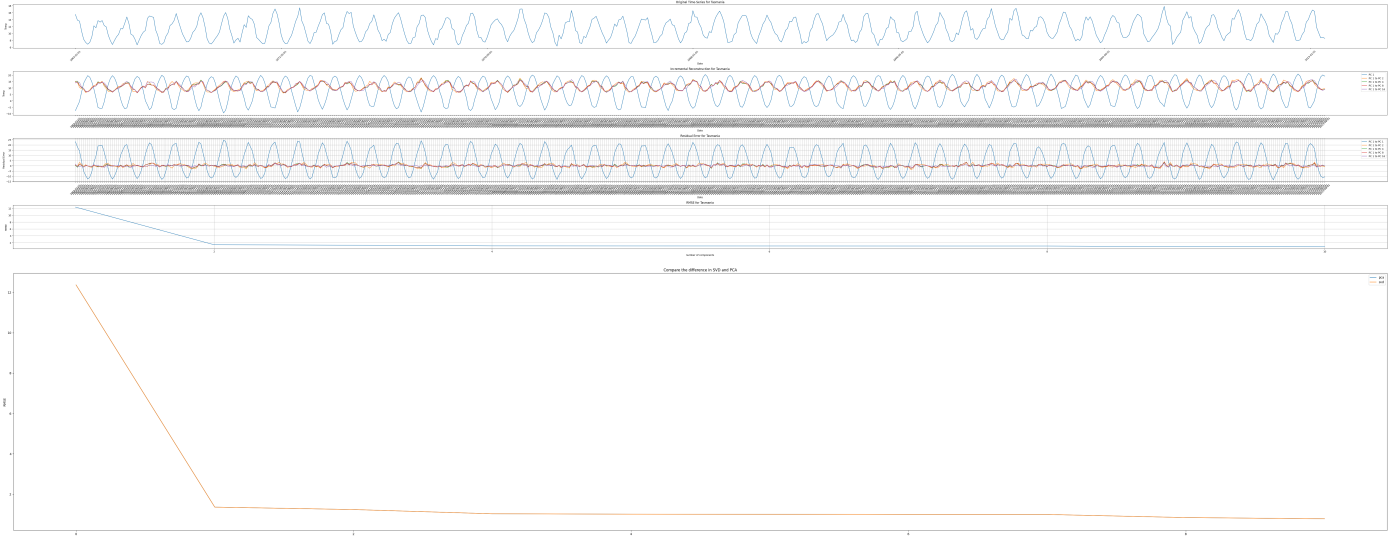




---------PCA---------



---------SVD---------



References

Understanding PCA and SVD:

1. https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8
2. https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca
3. https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues
4. https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.8-Singular-Value-Decomposition/

PCA:

1. Snippets from: https://plot.ly/ipython-notebooks/principal-component-analysis/
2. https://www.value-at-risk.net/principal-component-analysis/

Temperature Data:

1. https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth-surface-temperature-data
2. https://berkeleyearth.org/data/