In [5]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [45]:

```
!pip install lucifer-ml mlfoundry servicefoundry gradio pydantic==1.10.1
```

```
Collecting lucifer-ml
  Downloading lucifer_ml-0.0.79-py3-none-any.whl (43 kB)
                                              0.0/43.5 kB ? eta -:--:--
     ---------------------------------------- 43.5/43.5 kB 2.2 MB/s eta
0:00:00
Collecting mlfoundry
  Downloading mlfoundry-0.9.8-py3-none-any.whl (130 kB)
                                              0.0/130.9 kB ? eta -:--:-
-
     -------------------------------------- 130.9/130.9 kB 7.5 MB/s eta
0:00:00
Collecting servicefoundry
  Downloading servicefoundry-0.9.20-py3-none-any.whl (127 kB)
                                              0.0/127.2 kB ? eta -:--:-
-
     -------------------------------------- 127.2/127.2 kB 7.3 MB/s eta
0:00:00
Collecting gradio
  Downloading gradio-3.45.1-py3-none-any.whl (20.2 MB)
```

In [ ]:

In [6]:

```
df = pd.read_csv('temperature.csv')
```

In [ ]:

In [7]:

```
pd.set_option('display.max_row',25)
pd.set_option('display.max_column',25)
```

In [ ]:

In [8]:

```
df.head()
```

Out[8]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tma
0	1.0	30-06-2013	28.7	21.4	58.255688	91.116364	28.
1	2.0	30-06-2013	31.9	21.6	52.263397	90.604721	29.
2	3.0	30-06-2013	31.6	23.3	48.690479	83.973587	30.
3	4.0	30-06-2013	32.0	23.4	58.239788	96.483688	29.
4	5.0	30-06-2013	31.4	21.9	56.174095	90.155128	29

In [ ]:

In [9]:

```
df.dtypes.value_counts()
```

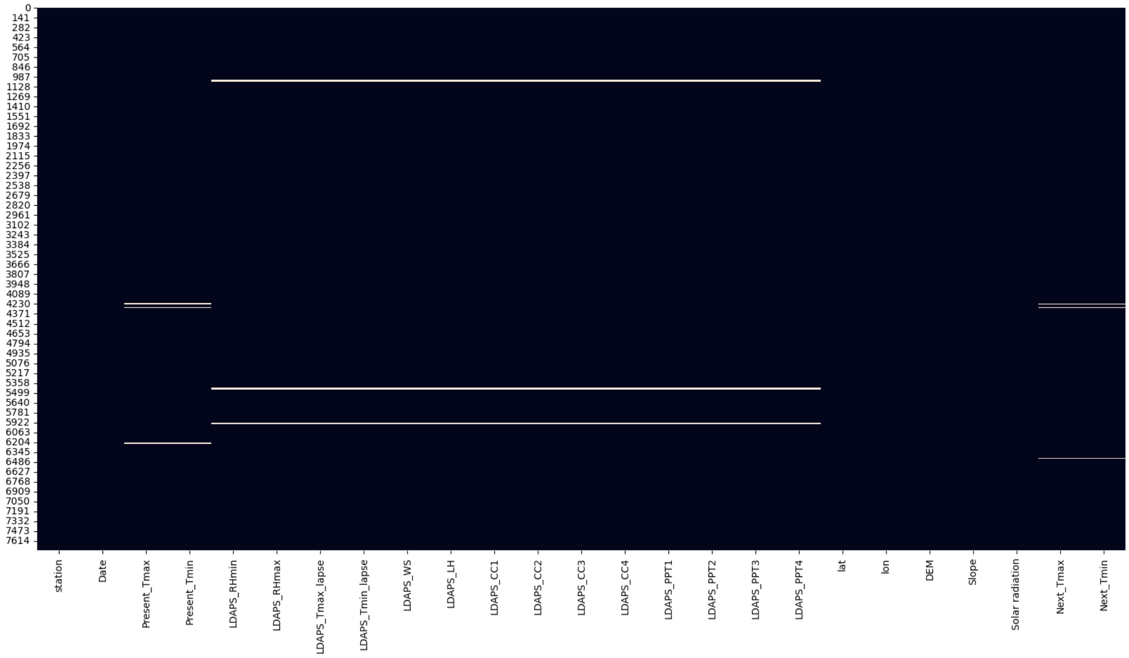
Out[9]:

```
float64    24
object      1
dtype: int64
```

In [ ]:

In [10]:

```
plt.figure(figsize=(20,10))
sns.heatmap(df.isna(),cbar=False)
plt.show()
print((df.isna().sum()/df.shape[0]*100).sort_values(ascending=False))
```

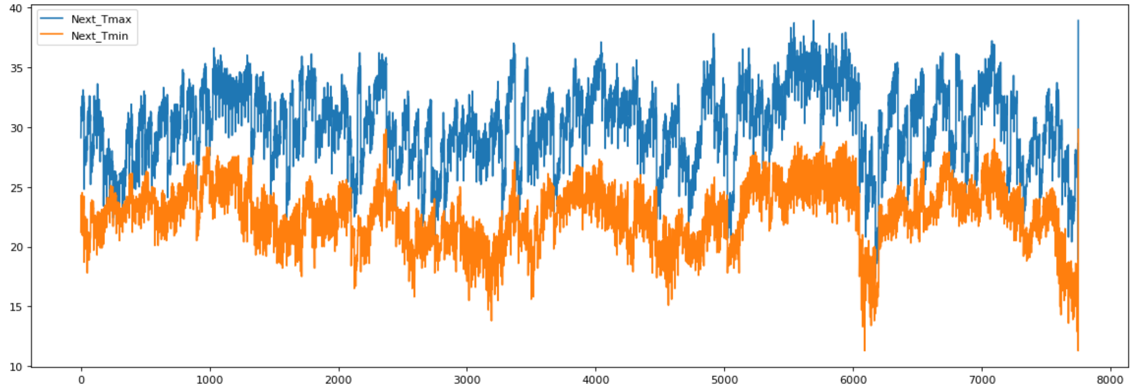


```
LDAPS_CC3           0.967492
LDAPS_PPT4          0.967492
LDAPS_PPT2          0.967492
LDAPS_PPT1          0.967492
LDAPS_CC4           0.967492
LDAPS_CC2           0.967492
LDAPS_CC1           0.967492
LDAPS_LH            0.967492
LDAPS_WS            0.967492
LDAPS_Tmin_lapse    0.967492
LDAPS_Tmax_lapse    0.967492
LDAPS_RHmax         0.967492
LDAPS_RHmin         0.967492
LDAPS_PPT3          0.967492
Present_Tmin        0.902993
Present_Tmax        0.902993
Next_Tmax           0.348297
Next_Tmin           0.348297
Date                0.025800
station             0.025800
lat                 0.000000
lon                 0.000000
DEM                 0.000000
Slope               0.000000
Solar radiation     0.000000
dtype: float64
```

In [ ]:

In [11]:

```
plt.figure(figsize=(18, 6), dpi=80)
plt.plot(df["Next_Tmax"],label="Next_Tmax")
plt.plot(df["Next_Tmin"],label="Next_Tmin")
plt.legend()
plt.show()
```



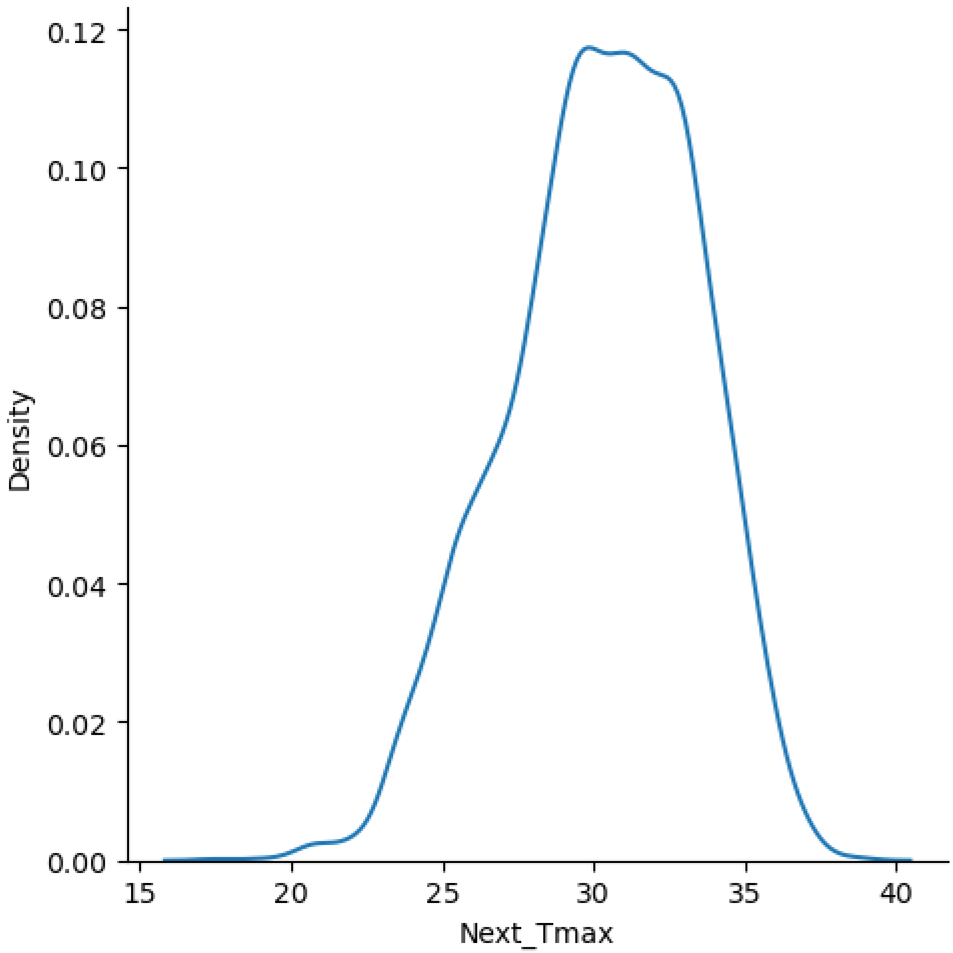
In [ ]:

```
```

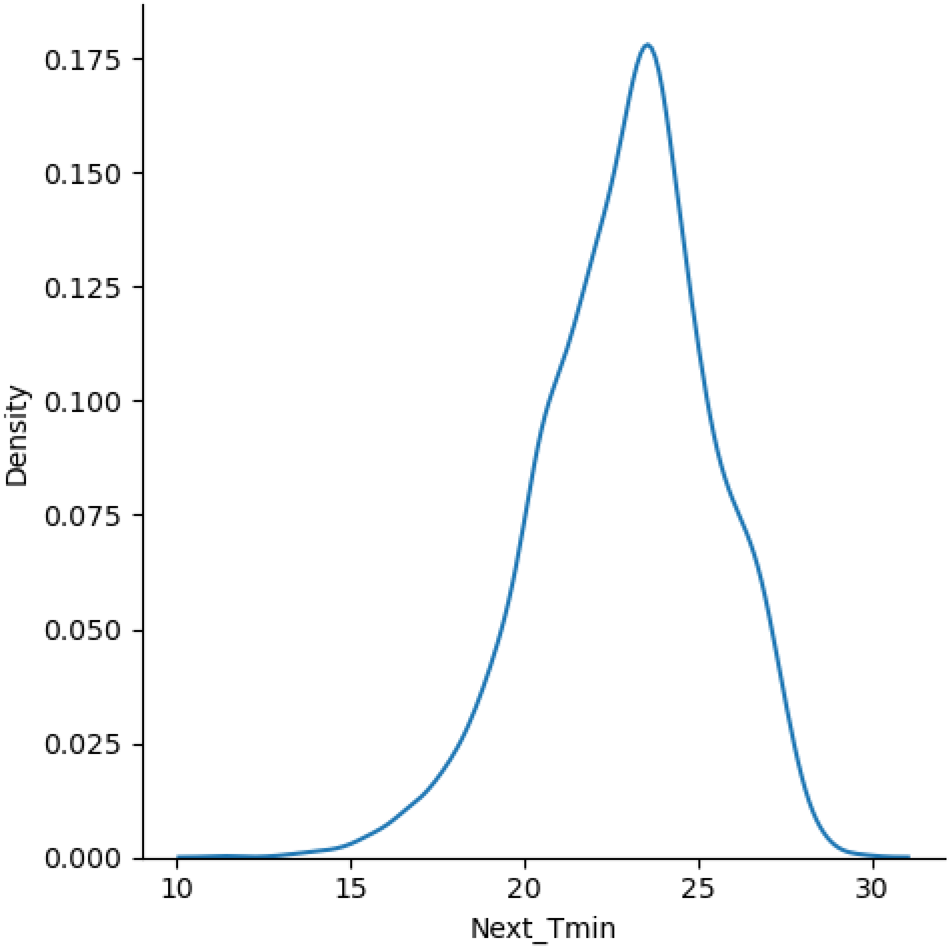
In [12]:

```
for col in ["Next_Tmax","Next_Tmin"]:
    plt.figure()
    sns.displot(df[col],kind='kde')
    plt.show()
print(df["Next_Tmax"].mean())
print(df["Next_Tmax"].std())
print(df["Next_Tmin"].mean())
print(df["Next_Tmin"].std())
```

```
<Figure size 640x480 with 0 Axes>
```



```
<Figure size 640x480 with 0 Axes>
```

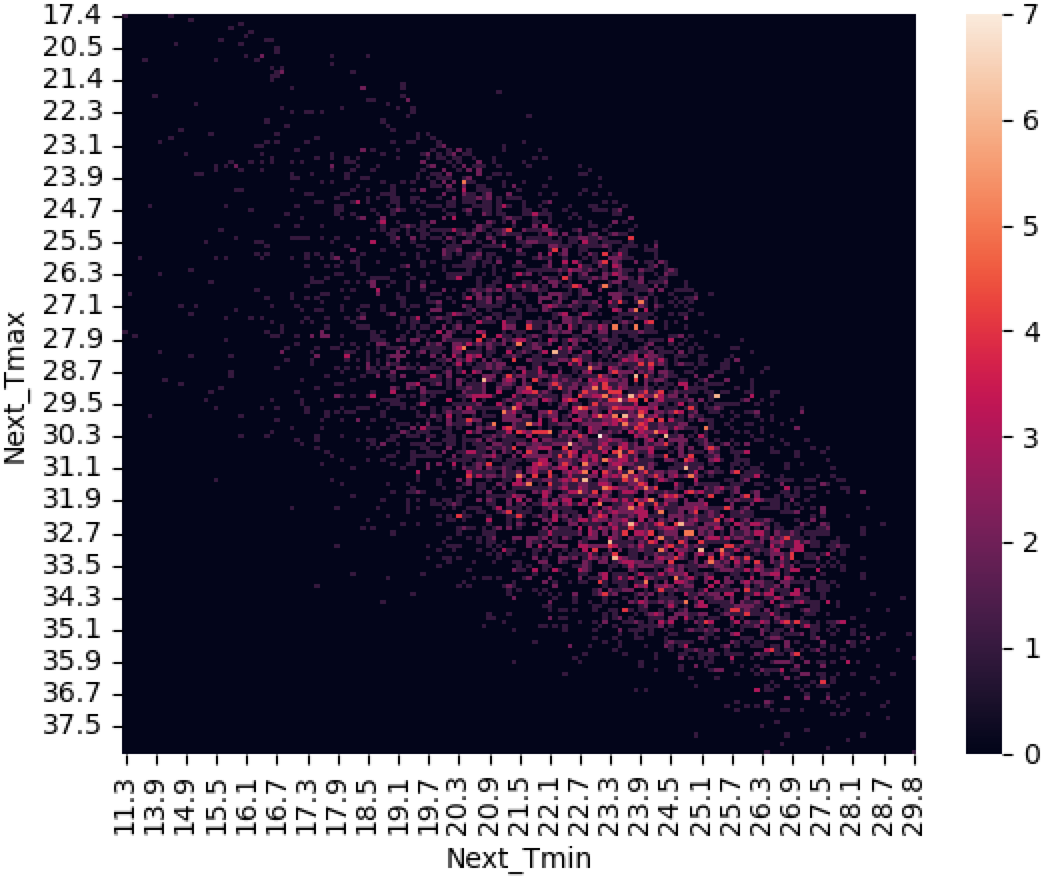


```
30.274886731391586
3.128010057855773
22.93222006472492
2.487612771331068
```

```
In [ ]:
```

In [13]:

```
plt.figure()
sns.heatmap(pd.crosstab(df['Next_Tmax'],df['Next_Tmin']))
plt.show()
```

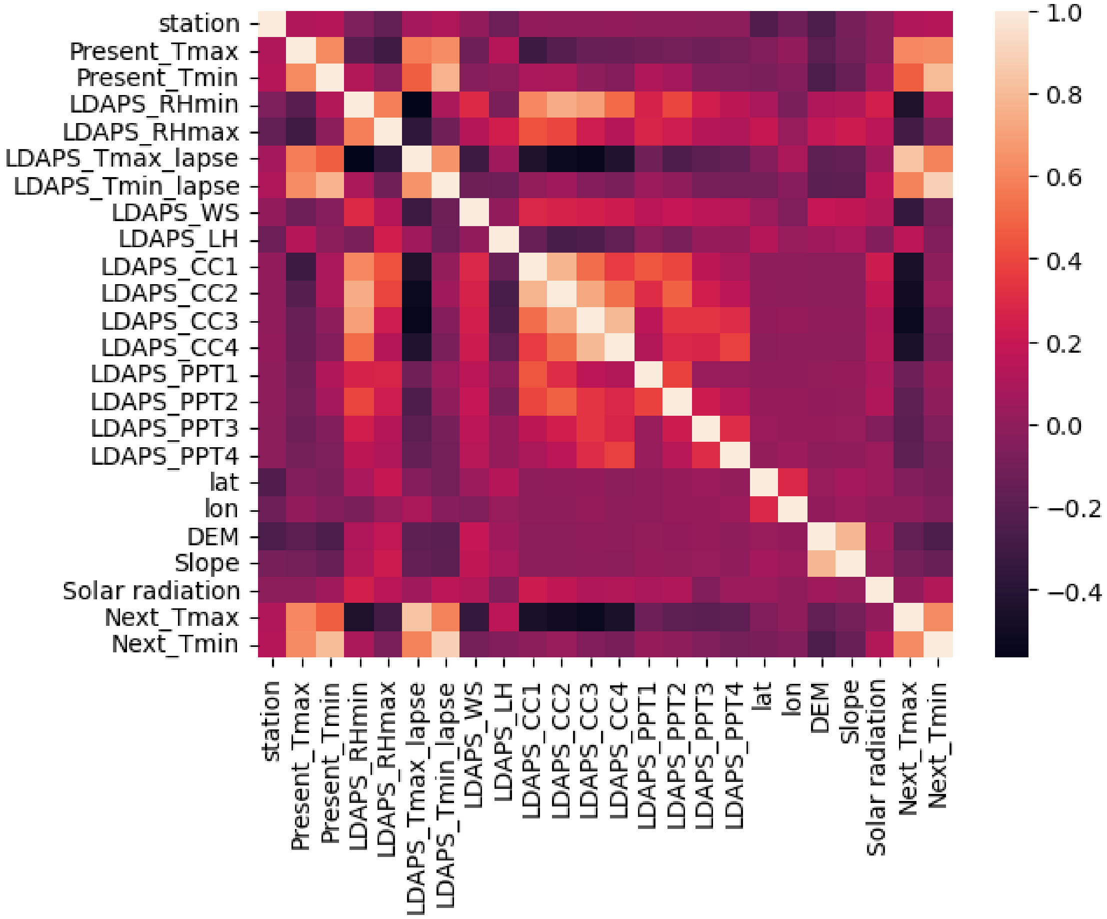


In [ ]:

In [14]:
```
sns.heatmap(df.corr())
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_2888\58359773.py:1: FutureWa
rning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns
or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr())
```

Out[14]:
```
<Axes: >
```



In [26]:

```
df.describe().T.style.bar(
    subset=['mean'],
    color='#606ff2').background_gradient(
    subset=['std'], cmap='PuBu').background_gradient(subset=['50%'], cmap=
```

Out[26]:

```
                        count         mean         std          min          25%
          station 7750.000000    13.000000    7.211568     1.000000     7.000000   13.0
     Present_Tmax 7682.000000    29.768211    2.969999    20.000000    27.800000   29.9
     Present_Tmin 7682.000000    23.225059    2.413961    11.300000    21.700000   23.4
      LDAPS_RHmin 7677.000000    56.759372   14.668111    19.794666    45.963543   55.0
      LDAPS_RHmax 7677.000000    88.374804    7.192004    58.936283    84.222862   89.7
 LDAPS_Tmax_lapse 7677.000000    29.613447    2.947191    17.624954    27.673499   29.7
 LDAPS_Tmin_lapse 7677.000000    23.512589    2.345347    14.272646    22.089739   23.7
         LDAPS_WS 7677.000000     7.097875    2.183836     2.882580     5.678705    6.5
         LDAPS_LH 7677.000000    62.505019   33.730589   -13.603212    37.266753   56.8
        LDAPS_CC1 7677.000000     0.368774    0.262458     0.000000     0.146654    0.3
        LDAPS_CC2 7677.000000     0.356080    0.258061     0.000000     0.140615    0.3
        LDAPS_CC3 7677.000000     0.318404    0.250362     0.000000     0.101388    0.2
        LDAPS_CC4 7677.000000     0.299191    0.254348     0.000000     0.081532    0.2
       LDAPS_PPT1 7677.000000     0.591995    1.945768     0.000000     0.000000    0.0
       LDAPS_PPT2 7677.000000     0.485003    1.762807     0.000000     0.000000    0.0
       LDAPS_PPT3 7677.000000     0.278200    1.161809     0.000000     0.000000    0.0
       LDAPS_PPT4 7677.000000     0.269407    1.206214     0.000000     0.000000    0.0
              lat 7752.000000    37.544722    0.050352    37.456200    37.510200   37.5
              lon 7752.000000   126.991397    0.079435   126.826000   126.937000  126.9
              DEM 7752.000000    61.867972   54.279780    12.370000    28.700000   45.7
            Slope 7752.000000     1.257048    1.370444     0.098475     0.271300    0.6
  Solar radiation 7752.000000  5341.502803  429.158867  4329.520508  4999.018555 5436.3
        Next_Tmax 7725.000000    30.274887    3.128010    17.400000    28.200000   30.5
        Next_Tmin 7725.000000    22.932220    2.487613    11.300000    21.300000   23.1
```

In [29]:

```
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
```

In [32]:

```
fig = px.line(df,x = 'Date', y = ['Present_Tmax', 'Present_Tmin',], templat
fig.show()
```

In [33]:

```
fig = px.line(df,x = 'Date', y = ['LDAPS_RHmax', 'LDAPS_RHmin'], template =
fig.show()
```

In [34]:

```
fig = px.line(df,x = 'Date', y = ['LDAPS_CC1', 'LDAPS_CC2', 'LDAPS_CC3', 'L
fig.show()
```

In [35]:

```
fig = px.line(df,x = 'Date', y = ['LDAPS_PPT1', 'LDAPS_PPT2', 'LDAPS_PPT3',
fig.show()
```

In [36]:

```
fig = px.density_mapbox(df, lat='lat', lon='lon', z='Present_Tmax', hover_

                        mapbox_style="stamen-terrain", template = 'plotly_
fig.show()
```

In [37]:

```
fig = px.density_mapbox(df, lat='lat', lon='lon', z='Present_Tmin',  hover_
                        mapbox_style="stamen-terrain", template = 'plotly_
fig.show()
```

In [38]:

```
fig = px.density_mapbox(df, lat='lat', lon='lon', z='LDAPS_WS', hover_data=

                        mapbox_style="stamen-terrain", template = 'plotly_c
fig.show()
```

In [39]:

```
fig = px.scatter_geo(df, lat='lat', lon='lon', color='Solar radiation', hov

                         template = 'plotly_dark', title = 'Solar Radiatior
fig.update_geos(
center=dict(lon=126.9780, lat=37.5665),
scope = 'asia'

)
fig.show()
```

In [40]:

```
fig = px.box(df, y=['Present_Tmax', 'Present_Tmin',
       'LDAPS_RHmin', 'LDAPS_RHmax',
       'LDAPS_WS', 'LDAPS_LH', 'LDAPS_CC1', 'LDAPS_CC2', 'LDAPS_CC3',
       'LDAPS_CC4', 'LDAPS_PPT1', 'LDAPS_PPT2', 'LDAPS_PPT3', 'LDAPS_PPT4',
fig.show()
```

In [58]:

```
features = df.drop( ['Date', 'Next_Tmax', 'Next_Tmin'], axis = 1)
labels_max = df['Next_Tmax']
labels_min = df['Next_Tmin']
```

In [59]:
```
features.head(), labels_max.head(), labels_min.head()
```

Out[59]:
```
(   station  Present_Tmax  Present_Tmin  LDAPS_RHmin  LDAPS_RHmax  \
 0      1.0          28.7          21.4    58.255688    91.116364   
 1      2.0          31.9          21.6    52.263397    90.604721   
 2      3.0          31.6          23.3    48.690479    83.973587   
 3      4.0          32.0          23.4    58.239788    96.483688   
 4      5.0          31.4          21.9    56.174095    90.155128   

    LDAPS_Tmax_lapse  LDAPS_Tmin_lapse  LDAPS_WS     LDAPS_LH  LDAPS_CC1  
\
 0         28.074101         23.006936  6.818887   69.451805   0.233947   
 1         29.850689         24.035009  5.691890   51.937448   0.225508   
 2         30.091292         24.565633  6.138224   20.573050   0.209344   
 3         29.704629         23.326177  5.650050   65.727144   0.216372   
 4         29.113934         23.486480  5.735004  107.965535   0.151407   

    LDAPS_CC2  LDAPS_CC3  LDAPS_CC4  LDAPS_PPT1  LDAPS_PPT2  LDAPS_PPT3  
\
 0   0.203896   0.161697   0.130928         0.0         0.0         0.0   
 1   0.251771   0.159444   0.127727         0.0         0.0         0.0   
 2   0.257469   0.204091   0.142125         0.0         0.0         0.0   
 3   0.226002   0.161157   0.134249         0.0         0.0         0.0   
 4   0.249995   0.178892   0.170021         0.0         0.0         0.0   

    LDAPS_PPT4      lat      lon       DEM   Slope  Solar radiation  
 0         0.0  37.6046  126.991  212.3350  2.7850      5992.895996  
 1         0.0  37.6046  127.032   44.7624  0.5141      5869.312500  
 2         0.0  37.5776  127.058   33.3068  0.2661      5863.555664  
 3         0.0  37.6450  127.022   45.7160  2.5348      5856.964844  
 4         0.0  37.5507  127.135   35.0380  0.5055      5859.552246  ,
 0    29.1
 1    30.5
 2    31.1
 3    31.7
 4    31.2
 Name: Next_Tmax, dtype: float64,
 0    21.2
 1    22.5
 2    23.9
 3    24.3
 4    22.5
 Name: Next_Tmin, dtype: float64)
```

In [60]:
```
features.shape, labels_max.shape, labels_min.shape
```

Out[60]:
```
((7752, 22), (7752,), (7752,))
```

In [ ]:
```

```

In [15]:
```
df = pd.read_csv('temperature.csv')
Save = df.copy()
```

In [16]:

```
def feature_engineering(df):
    df = df.drop(["Date"],axis=1)
    print(df.dtypes.value_counts())
    return(df)
```

In [17]:

```
def imputation(df):
    df = df.dropna(axis=0)
    return df
```

In [18]:

```
def encodage(df):
    return df
```

In [19]:

```
def preprocessing(df):
    df = imputation(df)
    df = encodage(df)
    df = feature_engineering(df)

    X = df.drop(['Next_Tmax','Next_Tmin'],axis=1)
    y_max = df["Next_Tmax"]
    y_min = df["Next_Tmin"]

    print(X.shape)
    print(y_max.shape)

    return X,y_max,y_min
```

In [63]:

```
from sklearn.model_selection import train_test_split
trainset, testset = train_test_split(df, test_size=0.2, random_state=0)
```

In [21]:

```
X_train, y_min_train, y_max_train = preprocessing(trainset)
X_test, y_min_test, y_max_test = preprocessing(testset)
```

```
float64    24
dtype: int64
(6068, 22)
(6068,)
float64    24
dtype: int64
(1520, 22)
(1520,)
```

In [64]:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import cross_validate
```

In [68]:

```
reg_max = make_pipeline(StandardScaler(), SGDRegressor( penalty='l2', max_
reg_max.fit(X_train, y_max_train)
```

Out[68]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [69]:

```
reg_min = make_pipeline(StandardScaler(),SGDRegressor( penalty='l2', max_it
reg_min.fit(X_train, y_min_train)
```

Out[69]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [70]:

```
cv_results_min = cross_validate(reg_min, X_train, y_min_train, cv=5, scorin
cv_results_max = cross_validate(reg_max, X_train, y_max_train, cv=5, scorin
```

In [71]:

```
print('Pour le Next_Tmin :')
print('Test RMSE :' , -cv_results_min['test_neg_root_mean_squared_error'].
print('Test r2 :' , cv_results_min['test_r2'].mean())
print("Train RMSE :" , -cv_results_min['train_neg_root_mean_squared_error'
print("Train r2 :" , cv_results_min['train_r2'].mean())
print("*-----------------------------------------*")
print('Pour le Next_Tmax :')
print('Test RMSE :' , -cv_results_max['test_neg_root_mean_squared_error'].
print('Test r2 :' , cv_results_max['test_r2'].mean())
print("Train RMSE :" , -cv_results_max['train_neg_root_mean_squared_error'
print("Train r2 :" , cv_results_max['train_r2'].mean())
```

```
Pour le Next_Tmin :
Test RMSE : 1.4716543842508778
Test r2 : 0.7752968207264456
Train RMSE : 1.4668193967011594
Train r2 : 0.7770290336876492
*-----------------------------------------*
Pour le Next_Tmax :
Test RMSE : 1.0111581476617029
Test r2 : 0.83325726155167
Train RMSE : 1.0061652510980292
Train r2 : 0.8353727766481154
```

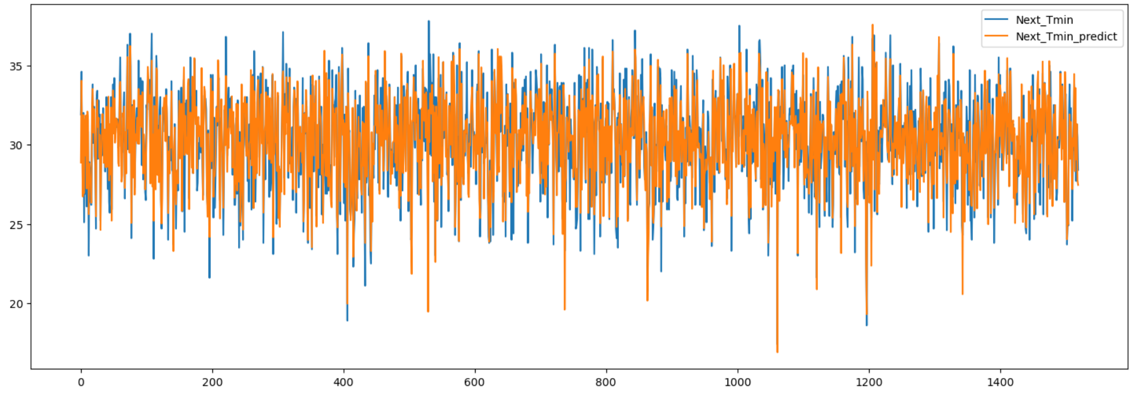
In [72]:

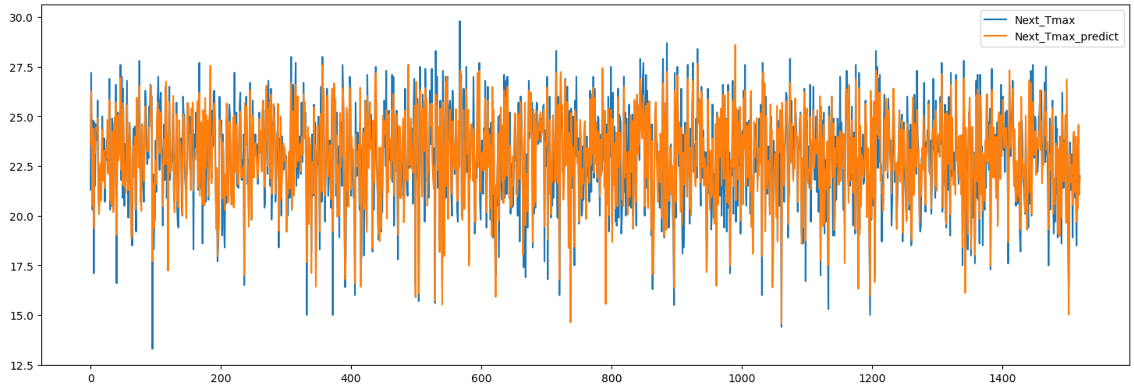
```
Next_Tmin_predict = reg_min.predict(X_test)
Next_Tmax_predict = reg_max.predict(X_test)
```

In [73]:

```
plt.figure(figsize=(18,6))
plt.plot(y_min_test.to_numpy(),label="Next_Tmin")
plt.plot(Next_Tmin_predict,label="Next_Tmin_predict")
plt.legend()
plt.show()

plt.figure(figsize=(18,6))
plt.plot(y_max_test.to_numpy(),label="Next_Tmax")
plt.plot(Next_Tmax_predict,label="Next_Tmax_predict")
plt.legend()
plt.show()
```





In [74]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

param_grid = {
    'bootstrap': [True],
    'max_depth': [70, 130],
    'max_features': [3, 6],
    'min_samples_leaf': [2, 3],
    'min_samples_split': [4, 8],
    'n_estimators': [1000, 500]
}
# Create a based model
rf = RandomForestRegressor()

grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

In [75]:

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

In [76]:

```
grid_search.fit(X_train, y_max_train)
print(grid_search.best_params_)
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, X_test, y_max_test)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
{'bootstrap': True, 'max_depth': 70, 'max_features': 6, 'min_samples_lea
f': 2, 'min_samples_split': 4, 'n_estimators': 500}
Model Performance
Average Error: 0.5544 degrees.
Accuracy = 97.51%.
```

In [77]:

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(X_train, y_max_train)
base_accuracy = evaluate(base_model, X_test, y_max_test)

print('Improvement of {:0.2f}%.'.format( 100 * (grid_accuracy - base_accura
```

```
Model Performance
Average Error: 0.6310 degrees.
Accuracy = 97.16%.
Improvement of 0.36%.
```

In [78]:

```

grid_search.fit(X_train, y_min_train)
print(grid_search.best_params_)
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, X_test, y_min_test)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
{'bootstrap': True, 'max_depth': 70, 'max_features': 6, 'min_samples_lea
f': 2, 'min_samples_split': 4, 'n_estimators': 500}
Model Performance
Average Error: 0.6881 degrees.
Accuracy = 97.69%.
```

In [79]:

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(X_train, y_min_train)
base_accuracy = evaluate(base_model, X_test, y_min_test)

print('Improvement of {:0.2f}%.'.format( 100 * (grid_accuracy - base_accura
```

```
Model Performance
Average Error: 0.7870 degrees.
Accuracy = 97.36%.
Improvement of 0.34%.
```

In [80]:

```python
Next_Tmax_TreeRegressor = RandomForestRegressor(random_state = 42,
                                                bootstrap=True, max_depth=1
                                                min_samples_leaf=2, min_sam
Next_Tmin_TreeRegressor = RandomForestRegressor(random_state = 42,
                                                bootstrap=True, max_depth=7
                                                min_samples_leaf=2, min_sam



print("---Next_Tmax---")
Next_Tmax_TreeRegressor.fit(X_train,y_max_train)
Next_Tmax_Accuracy = evaluate(Next_Tmax_TreeRegressor, X_test, y_max_test)

base_max_model = RandomForestRegressor(n_estimators = 10, random_state = 42
base_max_model.fit(X_train, y_max_train)
base_max_accuracy = evaluate(base_max_model, X_test, y_max_test)

print('Improvement of {:0.2f}%.'.format( 100 * (Next_Tmax_Accuracy - base_m
print("---------")


print("---Next_Tmin---")
Next_Tmin_TreeRegressor.fit(X_train,y_min_train)
Next_Tmin_Accuracy = evaluate(Next_Tmin_TreeRegressor, X_test, y_min_test)

base_min_model = RandomForestRegressor(n_estimators = 10, random_state = 42
base_min_model.fit(X_train, y_min_train)
base_min_accuracy = evaluate(base_min_model, X_test, y_min_test)

print('Improvement of {:0.2f}%.'.format( 100 * (Next_Tmin_Accuracy - base_m
print("---------")
```

```
---Next_Tmax---
Model Performance
Average Error: 0.5598 degrees.
Accuracy = 97.48%.
Model Performance
Average Error: 0.6310 degrees.
Accuracy = 97.16%.
Improvement of 0.33%.
---------
---Next_Tmin---
Model Performance
Average Error: 0.6888 degrees.
Accuracy = 97.69%.
Model Performance
Average Error: 0.7870 degrees.
Accuracy = 97.36%.
Improvement of 0.33%.
---------
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: