In [2]:
```python
# Credits: Some code adapted from the Intel Time Series Analysis course

%matplotlib inline

from datetime import datetime
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pyflux as pf
import statsmodels as ss
import seaborn as sns
import sys
import warnings

warnings.filterwarnings("ignore")
matplotlib.rcParams['figure.figsize'] = (20, 5)
```

Note:

"Constant" refers to the requirement that the pattern of autocorrelation does not change over time. So the strength of the dependence of, say, 2020 and 2014 is the same as that between 1990 and 1984. When you for example have an AR(1) process $yt=\phi yt-1+et$, the autocorrelation coefficient of yt and $yt-j$ is well-known to be ϕj. This depends on the lag j, but not on when we look at that lag. If ϕ were to change over time, as in $\phi=\phi t$, the condition would be violated.

Note:

We should use multiplicative models when the percentage change of our data is more important than the absolute value change (e.g. stocks, commodities); as the trend rises and our values grow, we see amplitude growth in seasonal and random fluctuations. If our seasonality and fluctuations are stable, we likely have an additive model.

Algorithm	Trend	Seasonal	Correlations
ARIMA	X	X	X
SMA Smoothing			
Simple Exponential Smoothing			
Seasonal Adjustment(constant trend and seasonal)	X	X	
Holt-Winters's Exponential Smoothing	X		
Multplicative Winters	X	X	

Data Prep and EDA

We'll be looking at [monthly average temperatures between 1907-1972]

In [10]:
```python
# load data and convert to datetime
monthly_temp = pd.read_csv('../Data/mean-monthly-temperature-1907-19.csv',
                           skipfooter=2,
                           header=0,
                           index_col=0,
                           names=['month', 'temp'])

monthly_temp.head()
```

Out[10]:

month	temp
1907-01	33.3
1907-02	46.0
1907-03	43.0
1907-04	55.0
1907-05	51.8

In [11]:
```python
monthly_temp.index = pd.to_datetime(monthly_temp.index)
monthly_temp
```

Out[11]:

month	temp
1907-01-01	33.3
1907-02-01	46.0
1907-03-01	43.0
1907-04-01	55.0
1907-05-01	51.8
...	...
1972-08-01	75.6
1972-09-01	64.1
1972-10-01	51.7
1972-11-01	40.3
1972-12-01	30.3

792 rows × 1 columns

In [12]:
```python
monthly_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 792 entries, 1907-01-01 to 1972-12-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
                  non-null    float64
```

Loading [MathJax]/extensions/Safe.js

```
dtypes: float64(1)
memory usage: 12.4 KB
```

In [13]:

```
annual_temp = monthly_temp.resample('A').mean()
annual_temp
```

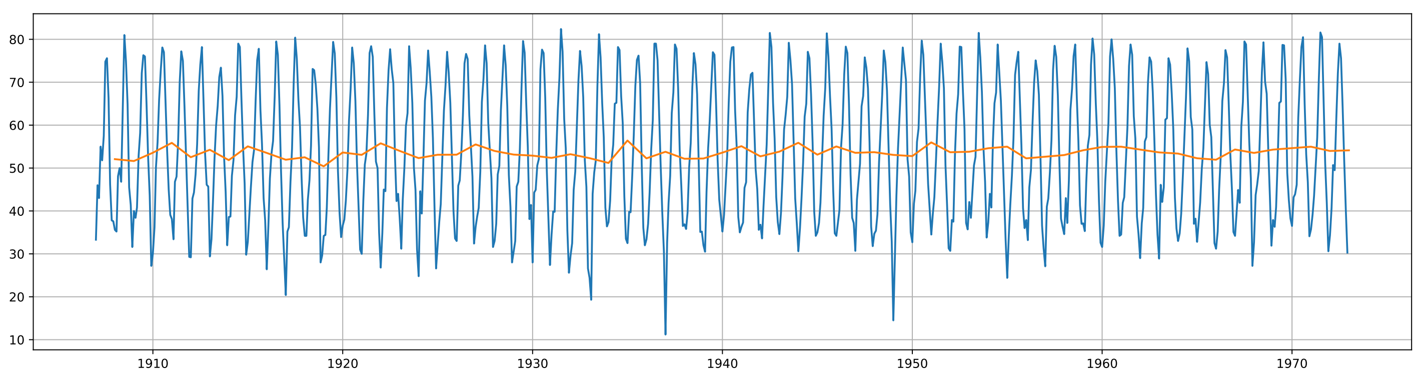
Out[13]:

	temp
month	
1907-12-31	52.075000
1908-12-31	51.633333
1909-12-31	53.566667
1910-12-31	55.875000
1911-12-31	52.525000
...	...
1968-12-31	54.300000
1969-12-31	54.641667
1970-12-31	54.975000
1971-12-31	53.983333
1972-12-31	54.125000

66 rows × 1 columns

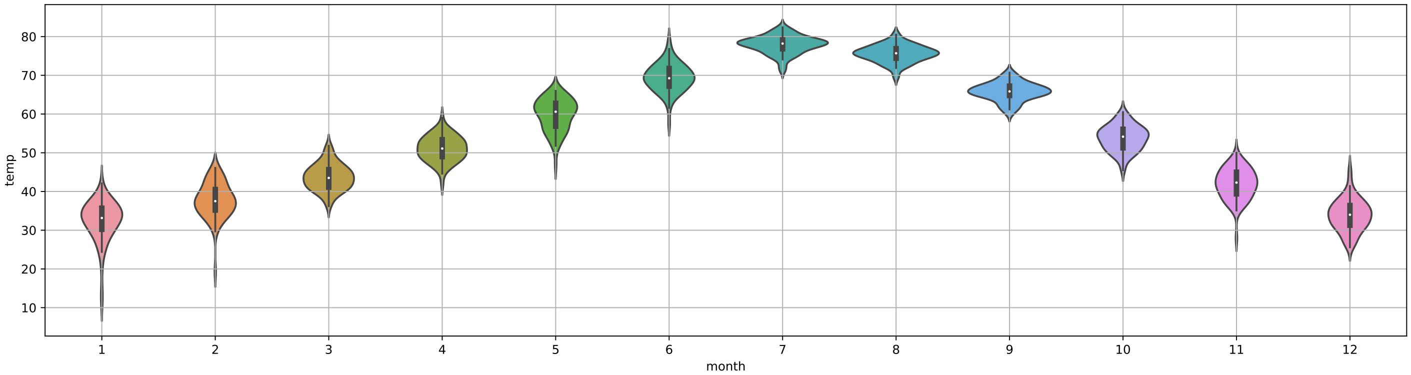
In [14]:

```
# plot both on same figure
plt.plot(monthly_temp)
plt.plot(annual_temp)
plt.grid();
```



In [15]:

```
# violinplot of months to determine variance and range
sns.violinplot(x=monthly_temp.index.month, y=monthly_temp.temp)
plt.grid();
```

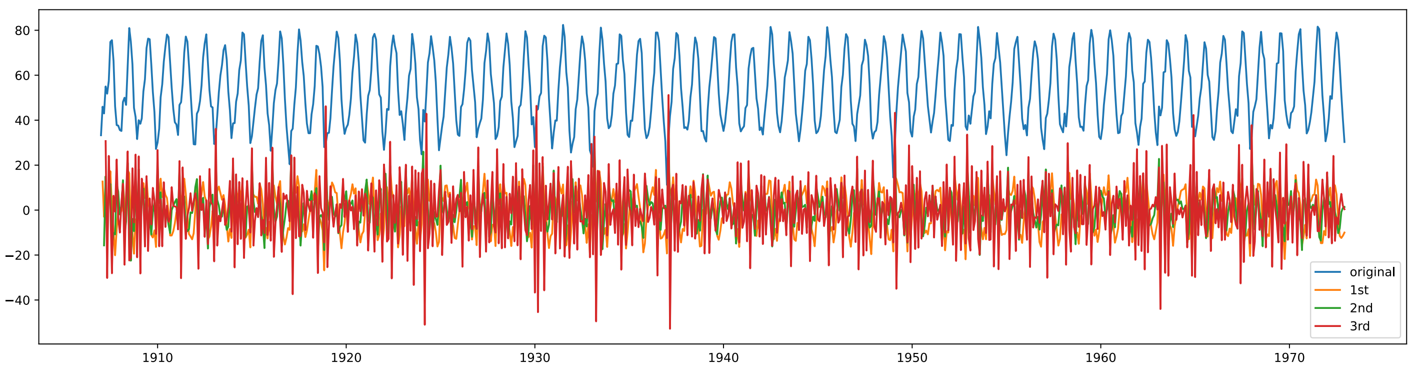


In [16]:

```
# check montly deviations for various diffs
print(monthly_temp.temp.std())
print(monthly_temp.temp.diff().std())
print(monthly_temp.temp.diff().diff().std()) # theoretically lowest, but > 1 is close enough
print(monthly_temp.temp.diff().diff().diff().std())

plt.figure(figsize=(20,5))
plt.plot(monthly_temp.temp)
plt.plot(monthly_temp.temp.diff())
plt.plot(monthly_temp.temp.diff().diff()) # theoretically lowest, but > 1 is close enough
plt.plot(monthly_temp.temp.diff().diff().diff())
plt.legend(['original','1st','2nd','3rd'])
plt.show()
```

```
15.815451540670232
9.45542698779467
9.25220062434484
14.726150786762325
```



In [17]:

```
# check annual deviations for various diffs
print(annual_temp.temp.std()) # looks stationary as is
print(annual_temp.temp.diff().std())
print(annual_temp.temp.diff().diff().std())
print(annual_temp.temp.diff().diff().diff().std())
```

```
1.2621242173990006
1.7725607336526374
3.117841613811376
5.803232109414729
```

In [18]:

```
# define Dickey-Fuller Test (DFT) function
import statsmodels.tsa.stattools as ts
def dftest(timeseries):
    dftest = ts.adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4],
                         index=['Test Statistic','p-value','Lags Used','Observations Used'])
    for key,value in dftest[4].items():
```

Loading [MathJax]/extensions/Safe.js

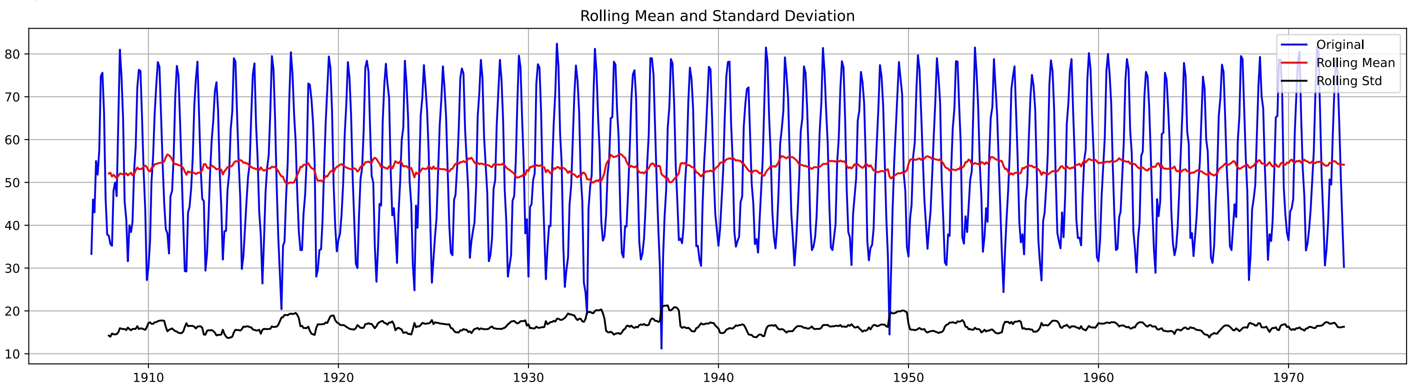
```
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
    #Determing rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.grid()
    plt.show(block=False)
```

In [19]:

```
# run DFT on monthly
dftest(monthly_temp.temp)
# p-value allows us to reject a unit root: data is stationary
```

```
Test Statistic          -6.481466e+00
p-value                  1.291867e-08
Lags Used                2.100000e+01
Observations Used        7.700000e+02
Critical Value (1%)     -3.438871e+00
Critical Value (5%)     -2.865301e+00
Critical Value (10%)    -2.568773e+00
dtype: float64
```

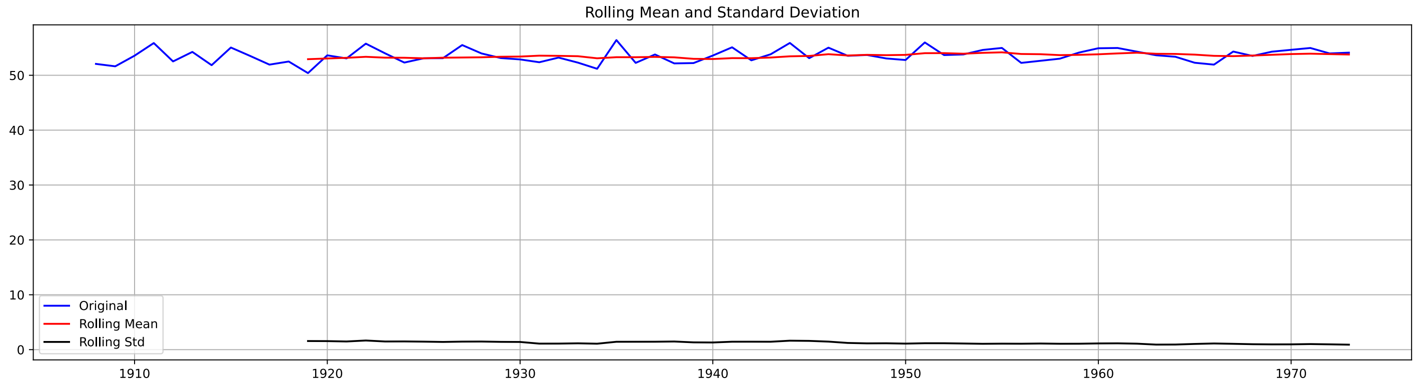


In my opinion, the monthly data is non-stationary as there is a seasonal component.

In [20]:

```
# run DFT on annual
dftest(annual_temp.temp)
```

```
Test Statistic          -7.878242e+00
p-value                  4.779473e-12
Lags Used                0.000000e+00
Observations Used        6.500000e+01
Critical Value (1%)     -3.535217e+00
Critical Value (5%)     -2.907154e+00
Critical Value (10%)    -2.591103e+00
dtype: float64
```

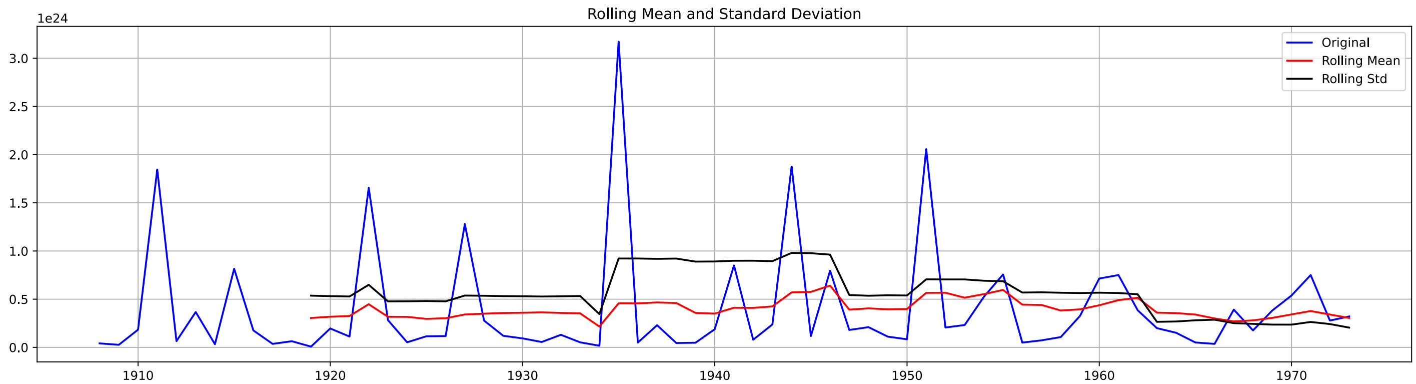


The p-value allows us to *reject* a unit root (i.e. the data is stationary).

In [21]:

```python
# here's an example of non-stationary with DFT results
dftest(np.exp(annual_temp.temp))
```

```
Test Statistic          -0.449458
p-value                  0.901508
Lags Used               10.000000
Observations Used       55.000000
Critical Value (1%)     -3.555273
Critical Value (5%)     -2.915731
Critical Value (10%)    -2.595670
dtype: float64
```



Create Helper Functions

In [22]:

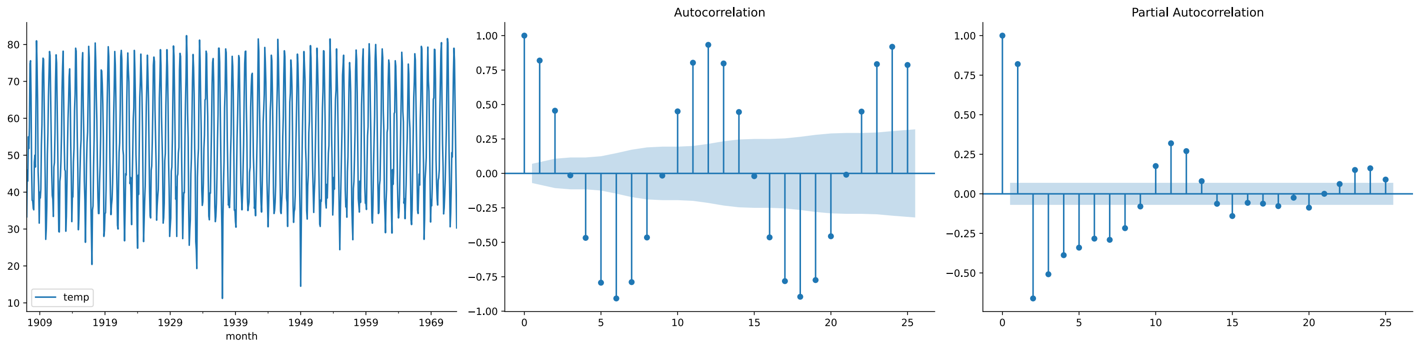
```python
# define helper plot function for visualization
import statsmodels.tsa.api as smt

def plots(data, lags=None):
    layout = (1, 3)
    raw  = plt.subplot2grid(layout, (0, 0))
    acf  = plt.subplot2grid(layout, (0, 1))
    pacf = plt.subplot2grid(layout, (0, 2))

    data.plot(ax=raw)
    smt.graphics.plot_acf(data, lags=lags, ax=acf)
    smt.graphics.plot_pacf(data, lags=lags, ax=pacf)
    sns.despine()
    plt.tight_layout()
```

In [23]:

```python
# helper plot for monthly temps
plots(monthly_temp, lags=25);
                 de for visual
# we note a 12-period cycle (yearly) with suspension bridge design, so must use SARIMA
```



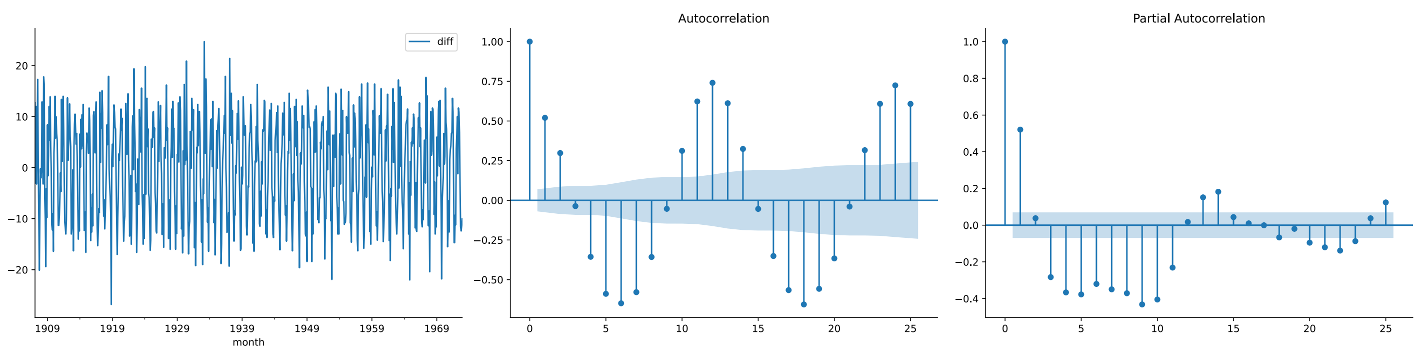
I honestly thought maybe avg temp of a year might correlate with previor year. But for annual temp, I don't see any AR or even any MR term. Maybe very high MR terms but not sure they have practical signficance (even though they have some statistical significance)

In [24]:

```
monthly_diff = monthly_temp.copy()
monthly_diff['diff'] = monthly_diff['temp'].diff()
monthly_diff_1 = monthly_diff.drop('temp', 1).iloc[1:]
```

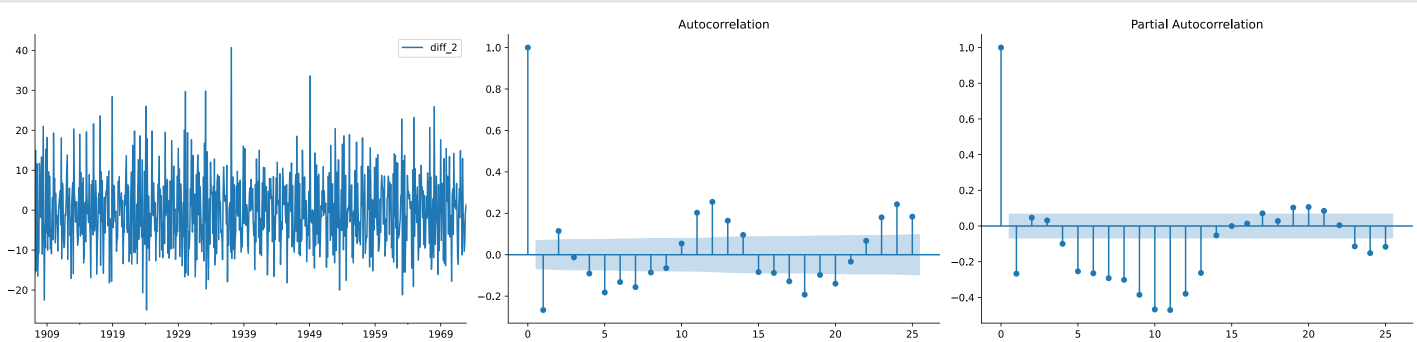
In [25]:

```
# helper plot for monthly temps
plots(monthly_diff_1, lags=25);
# open Duke guide for visual
# we note a 12-period cycle (yearly) with suspension bridge design, so must use SARIMA
```



In [26]:

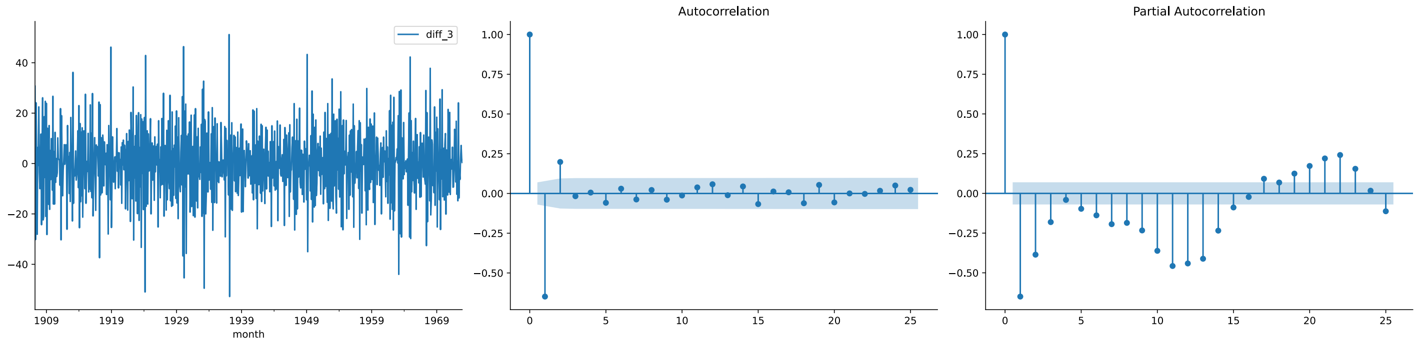
```
monthly_diff_1['diff_2'] = monthly_diff_1['diff'].diff()
monthly_diff_2 = monthly_diff_1.drop('diff', 1).iloc[1:]
plots(monthly_diff_2, lags=25);
```



In [27]:

```
monthly_diff_2['diff_3'] = monthly_diff_2['diff_2'].diff()
monthly_diff_3 = monthly_diff_2.drop('diff_2', 1).iloc[1:]
plots(monthly_diff_3, lags=25);
```

Loading [MathJax]/extensions/Safe.js



The diff_3 ACF plot shows over-differencing since lag-1 is greater than 0.5 in absolute terms. Also, we concluded earlier that the lower std is at diff_2. I am not sure why the author of this notebook did not difference before plotting ACF/PACF plots. Maybe, that is because we are building ARIMA model (not ARMA) and we will eventually tell the model to Integrate (I) parameter. Also, looking at this example, I see that even if the data is non-stationary, the plots can show the seasonality terms to be added. I still feel that the books methodology is more straightforward. i.e take the RegSeasDiff and then see what AR/MA terms to add.

Proper Textbook Methodology

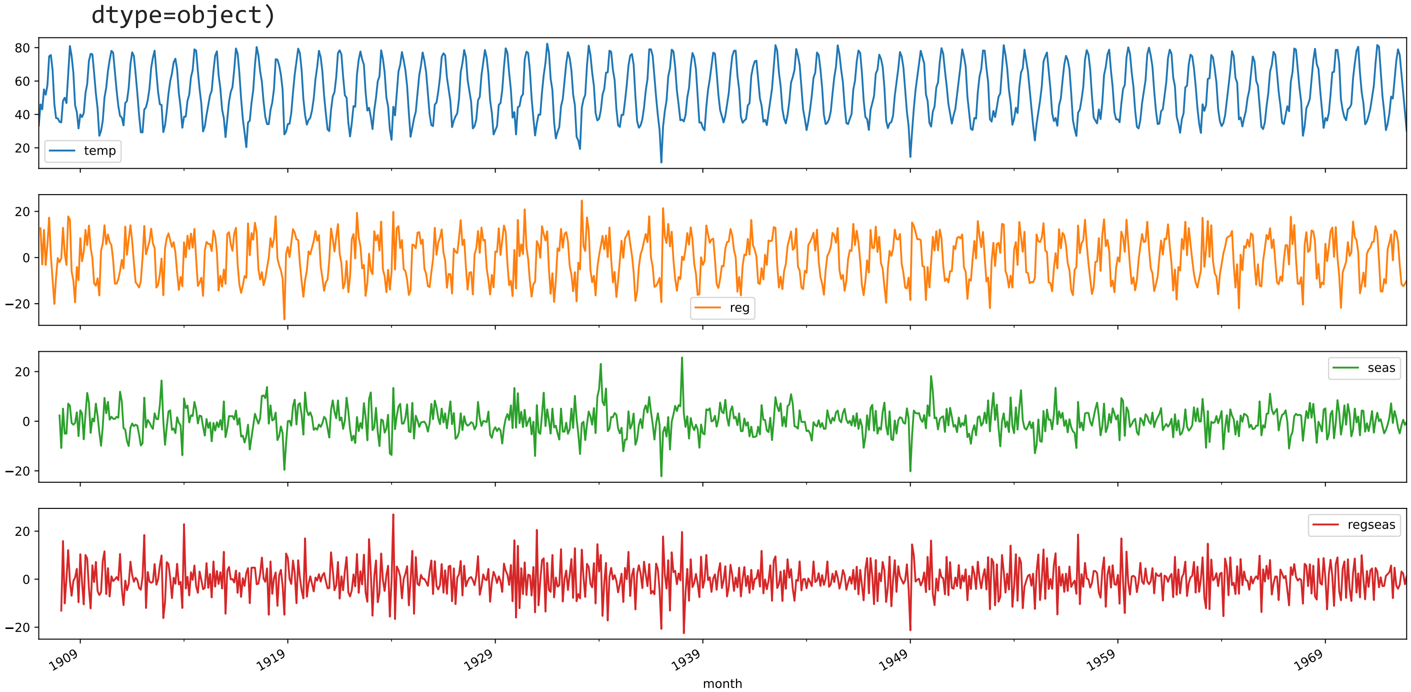
In [28]:

```
monthly_temp['reg'] = monthly_temp['temp'].diff(1)
monthly_temp['seas'] = monthly_temp['temp'].diff(12)
monthly_temp['regseas'] = monthly_temp['seas'].diff(1)
monthly_temp.head(14)

monthly_temp.plot(subplots=True, figsize=(20,10))
```

Out[28]:

```
array([<AxesSubplot:xlabel='month'>, <AxesSubplot:xlabel='month'>,
       <AxesSubplot:xlabel='month'>, <AxesSubplot:xlabel='month'>],
      dtype=object)
```



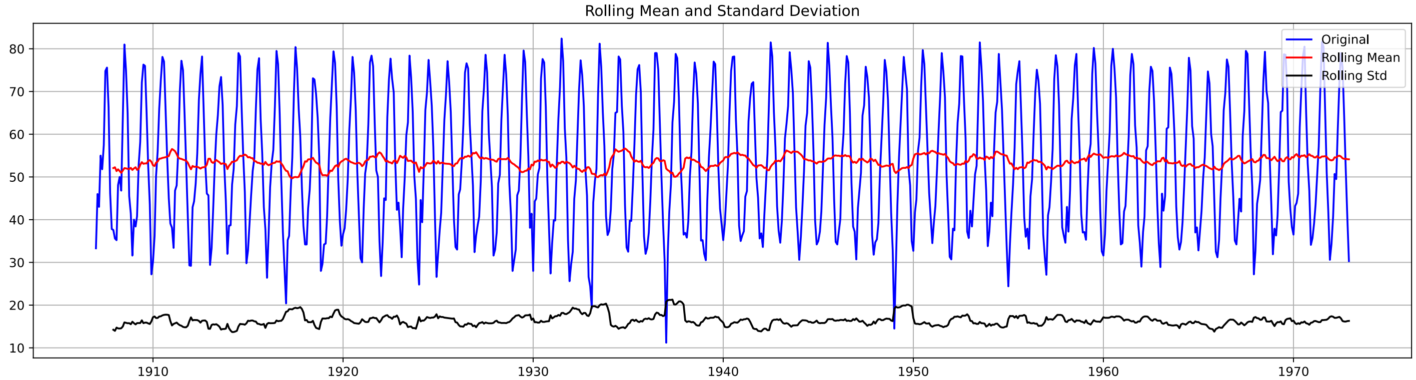
In [29]:

```
dftest(monthly_temp['temp'])
```

```
Test Statistic          -6.481466e+00
[illegible]              1.291867e-08
Lags Used                2.100000e+01
```

Loading [MathJax]/extensions/Safe.js

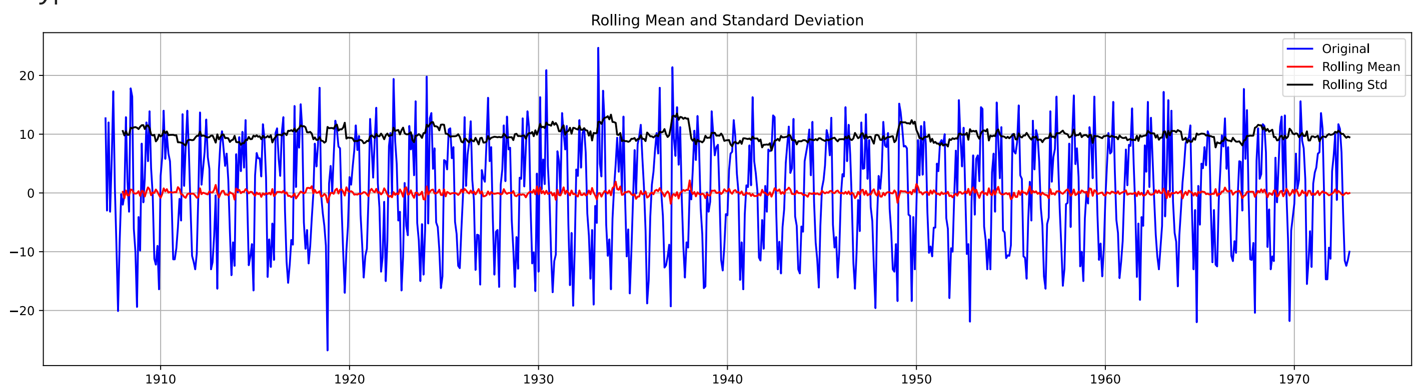
```
Observations Used       7.700000e+02
Critical Value (1%)    -3.438871e+00
Critical Value (5%)    -2.865301e+00
Critical Value (10%)   -2.568773e+00
dtype: float64
```



In [30]:

```
dftest(monthly_temp['reg'].dropna())
```

```
Test Statistic         -1.230261e+01
p-value                 7.391771e-23
Lags Used               2.100000e+01
Observations Used       7.690000e+02
Critical Value (1%)    -3.438882e+00
Critical Value (5%)    -2.865306e+00
Critical Value (10%)   -2.568775e+00
dtype: float64
```

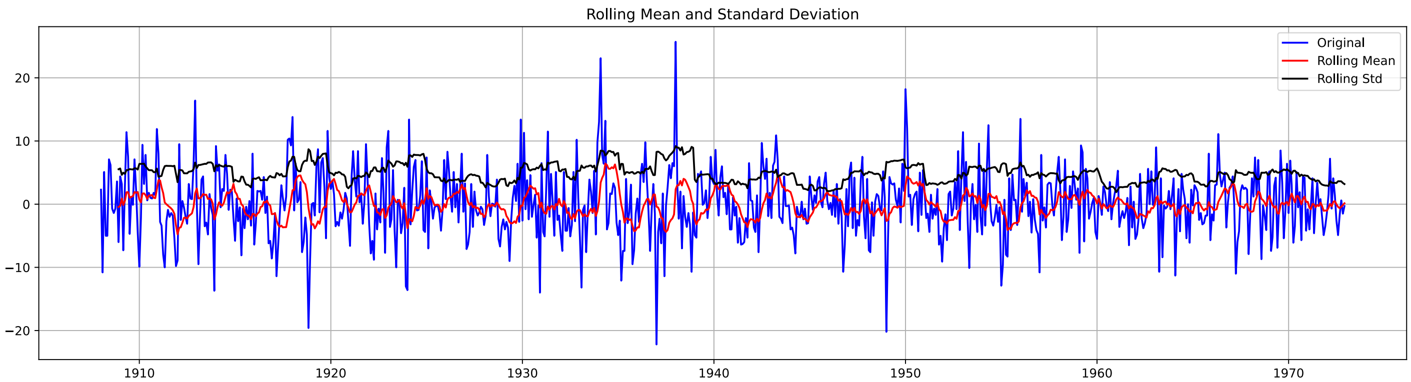


In [31]:

```
dftest(monthly_temp['seas'].dropna())
```

```
Test Statistic         -1.265808e+01
p-value                 1.323220e-23
Lags Used               1.200000e+01
Observations Used       7.670000e+02
Critical Value (1%)    -3.438905e+00
Critical Value (5%)    -2.865316e+00
Critical Value (10%)   -2.568781e+00
dtype: float64
```

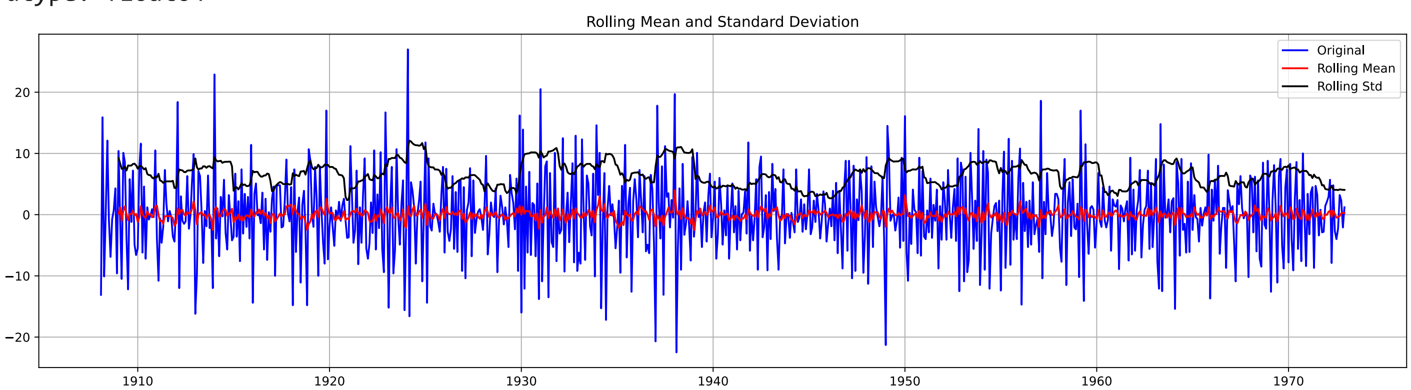
Loading [MathJax]/extensions/Safe.js



In [32]:

```
dftest(monthly_temp['regseas'].dropna())
```

```
Test Statistic          -1.184617e+01
p-value                  7.390517e-22
Lags Used                2.000000e+01
Observations Used        7.580000e+02
Critical Value (1%)     -3.439006e+00
Critical Value (5%)     -2.865361e+00
Critical Value (10%)    -2.568804e+00
dtype: float64
```



In [33]:

```
print(monthly_temp['reg'].std())
print(monthly_temp['seas'].std())
print(monthly_temp['regseas'].std())
```

```
9.45542698779467
5.228035433877334
6.6519331827
```

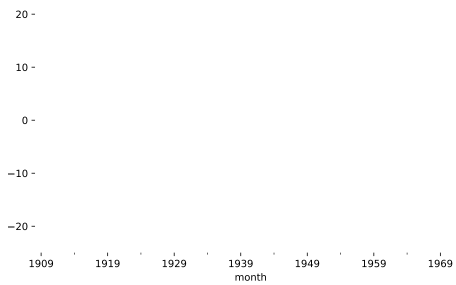
Okay so from the above graphs, we can see that the ADF test is unreliable because it doesn't seem to account for the monthly variations in the data. But it is reliable in the sense that it shows stronger stationary affect in the differences series. By the std rule, maybe the seas difference is good enough (or even better) but we may see short-term trends (stochastic trend effects) and therefore, may want to do a combined regular and seasonal difference (aka RegSeasDiff).

There is no clear cut guideline as to whether we should difference the data before test_train_split or after. Actually, it doesn't even matter. I am going to assume it doesn't and gonna plot ACF/PACF and choose paramaters based on complete dataset but while fitting the model, only use the training part.

In [34]:

```
plots(monthly_temp[['regseas']].dropna(), lags=25);
```

Loading [MathJax]/extensions/Safe.js


20
10
0
−10
−20
1909
1919
1929
1939
1949
1959
1969
month