In [1]:
```
import pandas as pd
import re
import matplotlib.pyplot as plt
import os
import plotly.express as px
```

In [2]:
```
df=pd.read_csv('train.csv')
```

In [3]:
```
df.set_index('date',inplace=True)
```

In [4]:
```
df.head()
```

Out[4]:

date	store	item	sales
2013-01-01	1	1	13
2013-01-02	1	1	11
2013-01-03	1	1	14
2013-01-04	1	1	13
2013-01-05	1	1	10

Sales Total Storewise

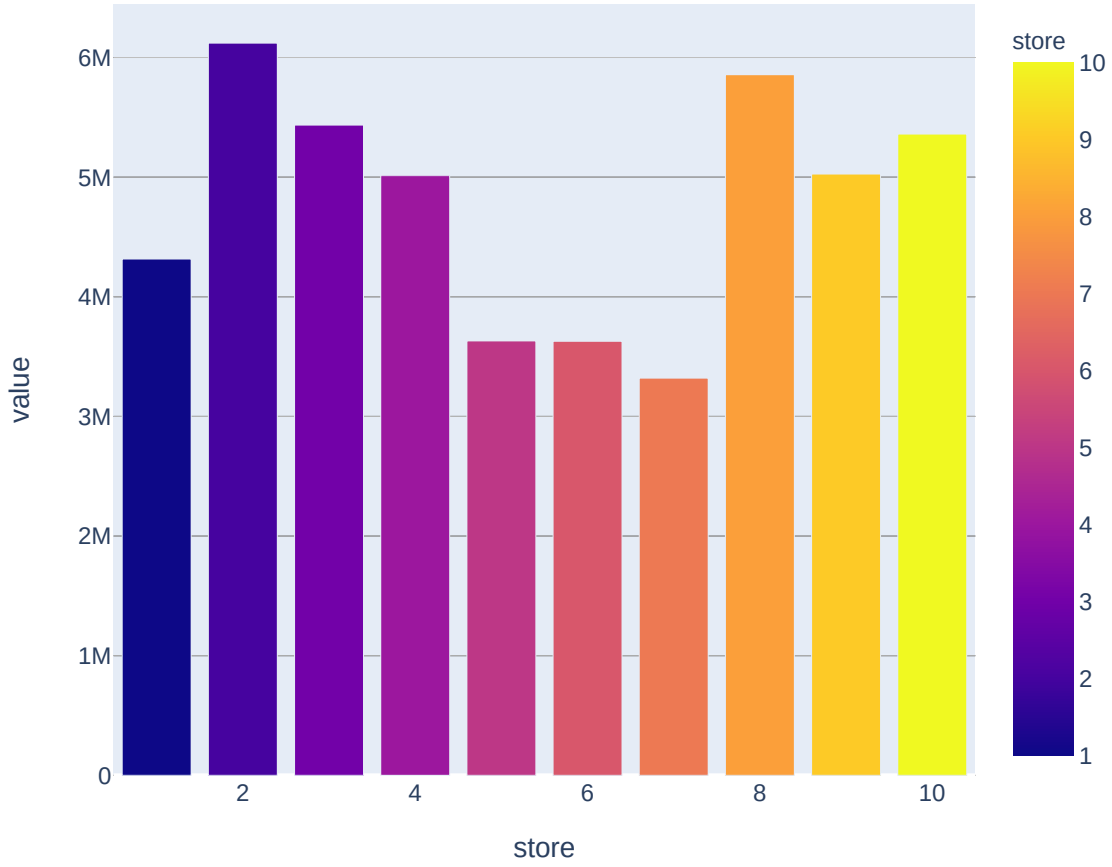
In [6]:
```
store_sales=df.groupby(by='store')[['sales']].sum()
store_sales
```

Out[6]:

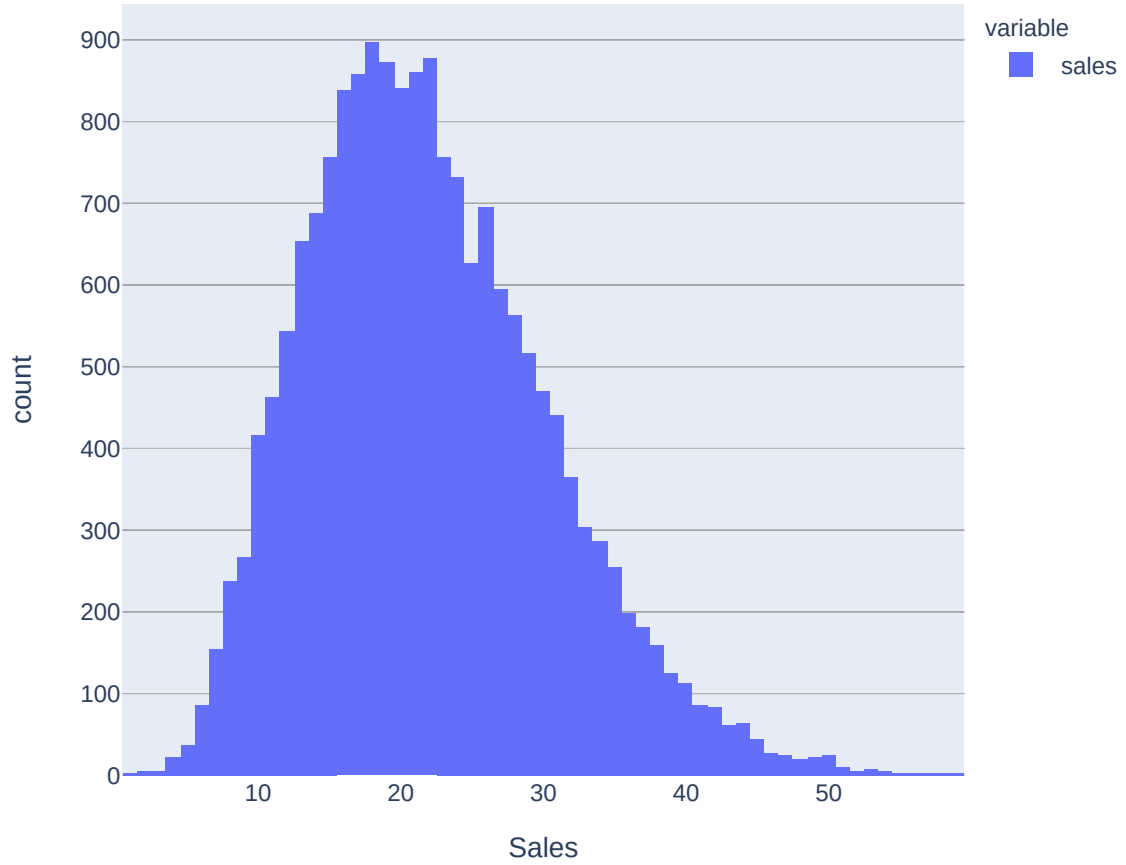
store	sales
1	4315603
2	6120128
3	5435144
4	5012639
5	3631016
6	3627670
7	3320009
8	5856169
9	5025976
10	5360158

In [7]:
```
store=store_sales.index
```

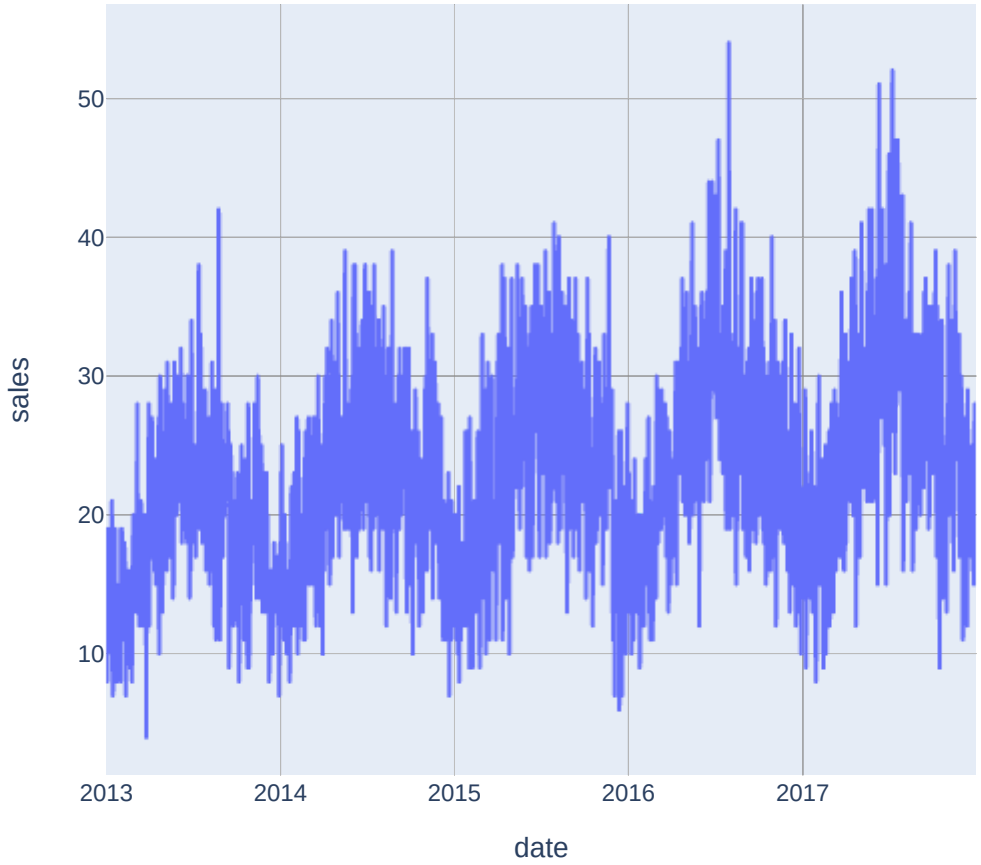
In [8]:
```
fig = px.bar(store_sales,color=store)
fig.show()
```



In [550]:
```
fig = px.histogram(df[df.item==1][['sales']],labels=dict(value="Sales"))
fig.show()
```



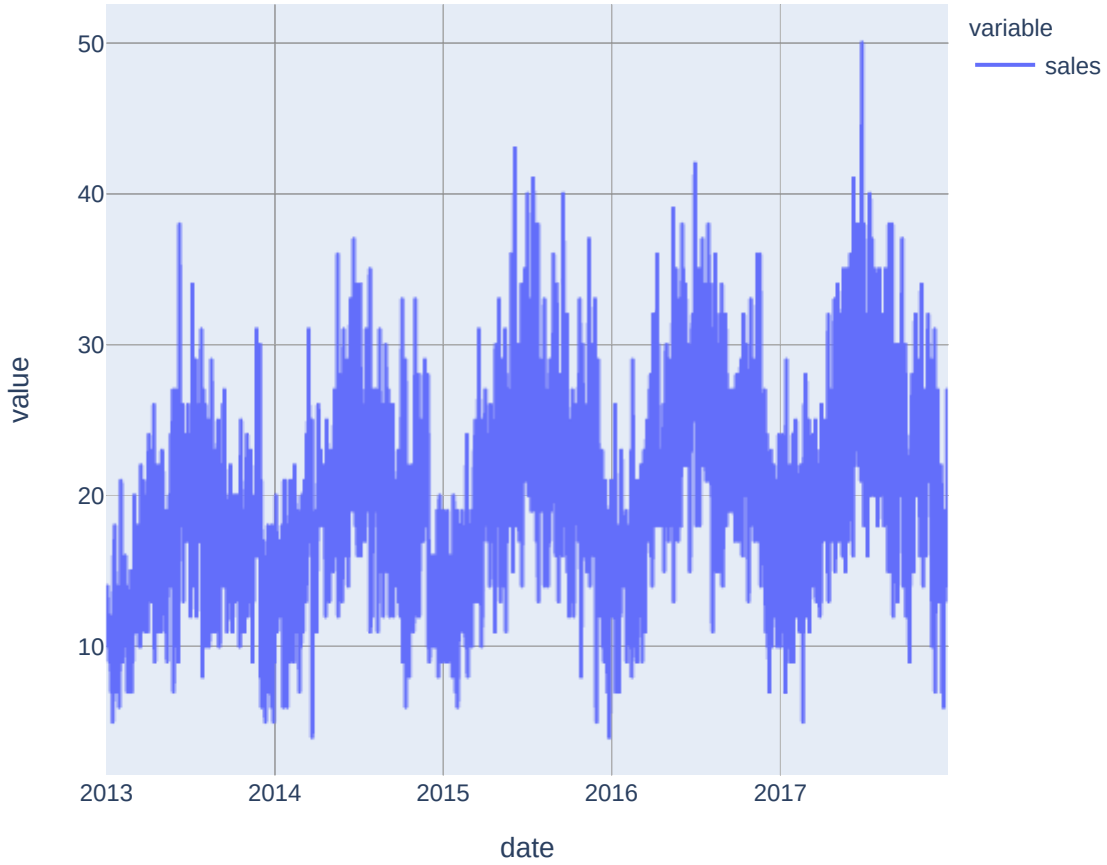
In [539]:
```
fig = px.line(df[(df.item==1) & (df.store==4)][['sales']],y='sales')
fig.show()
```



Predict 3 months of sales for 50 different items at 10 different stores.

In [24]:
```
df_1_1=df[(df.item==1) & (df.store==1)][['sales']]
```

In [32]:
```
fig = px.line(df_1_1)
fig.show()
```

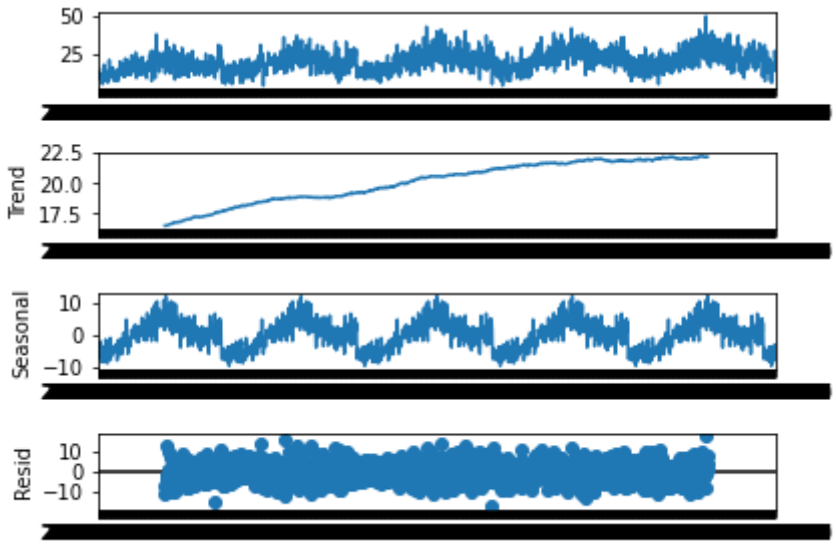


Stationary

mean,variance and co-variance is constant over periods

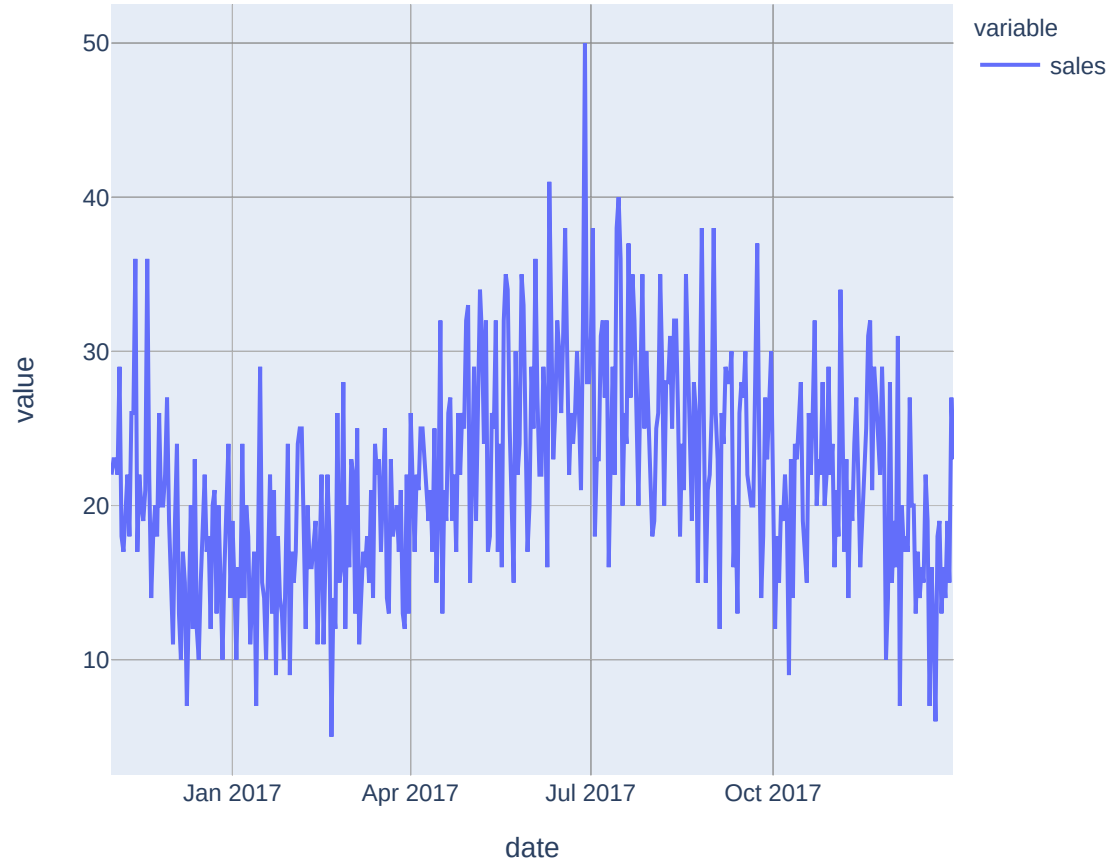
In [568]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df_1_1, model='additive', period=365)
plt.figure(figsize=(36, 24))
result.plot()
plt.show()
```

```
<Figure size 2592x1728 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

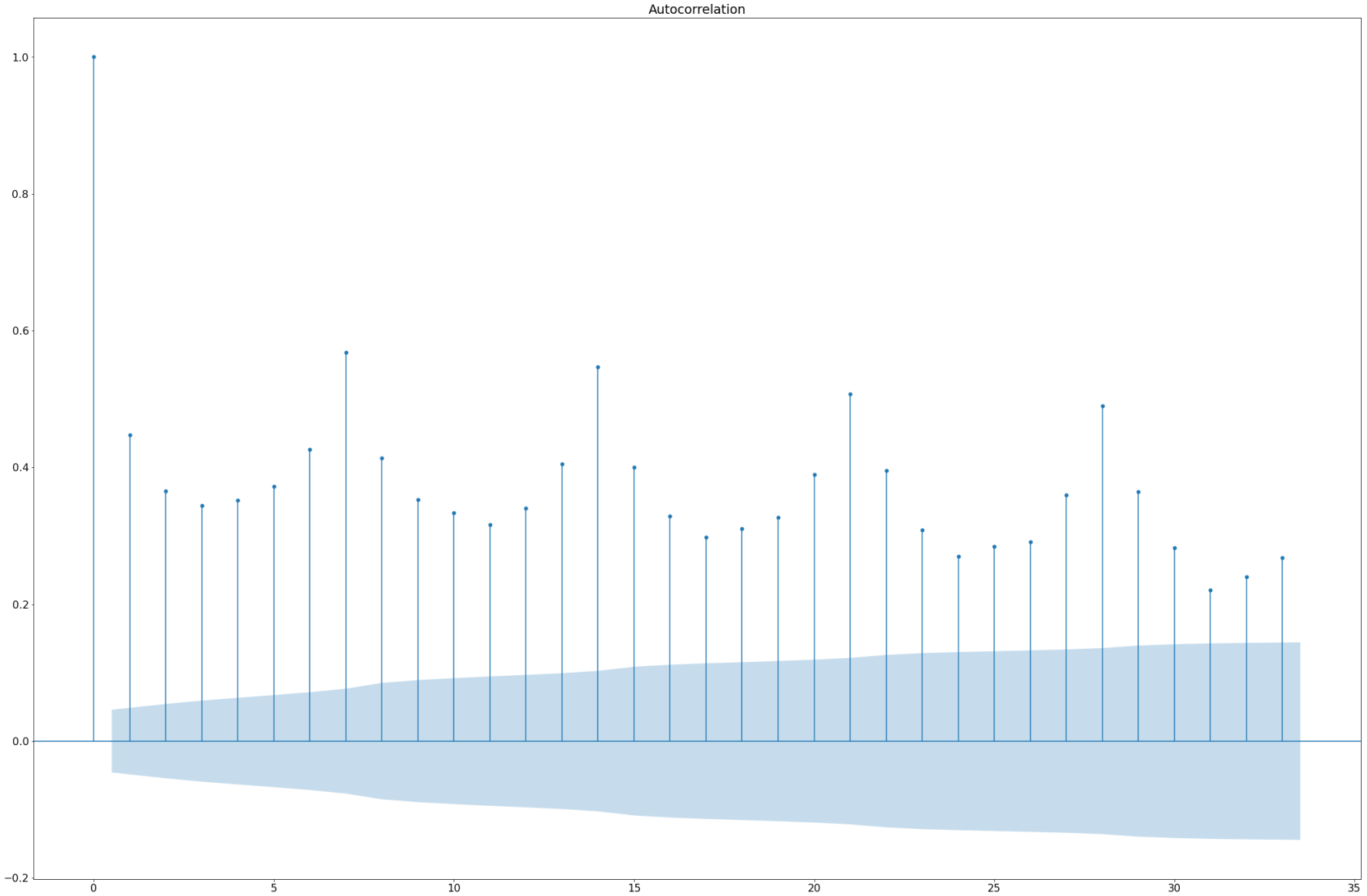
In [562]:
```
df_1_1.iloc[1400:,].plot()
```



In [36]:
```
pd.options.plotting.backend = "plotly"
```

In [76]:
```
import statsmodels.api as sm

fig, ax = plt.subplots(figsize=(36,24))
sm.graphics.tsa.plot_acf(df_1_1, ax=ax)
plt.show()
```



Check for stationary using Ad Fuller Hypothesis test

The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

Null Hypothesis (H0): If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure. Alternate Hypothesis (H1): The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure. We interpret this result using the p-value from the test. A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

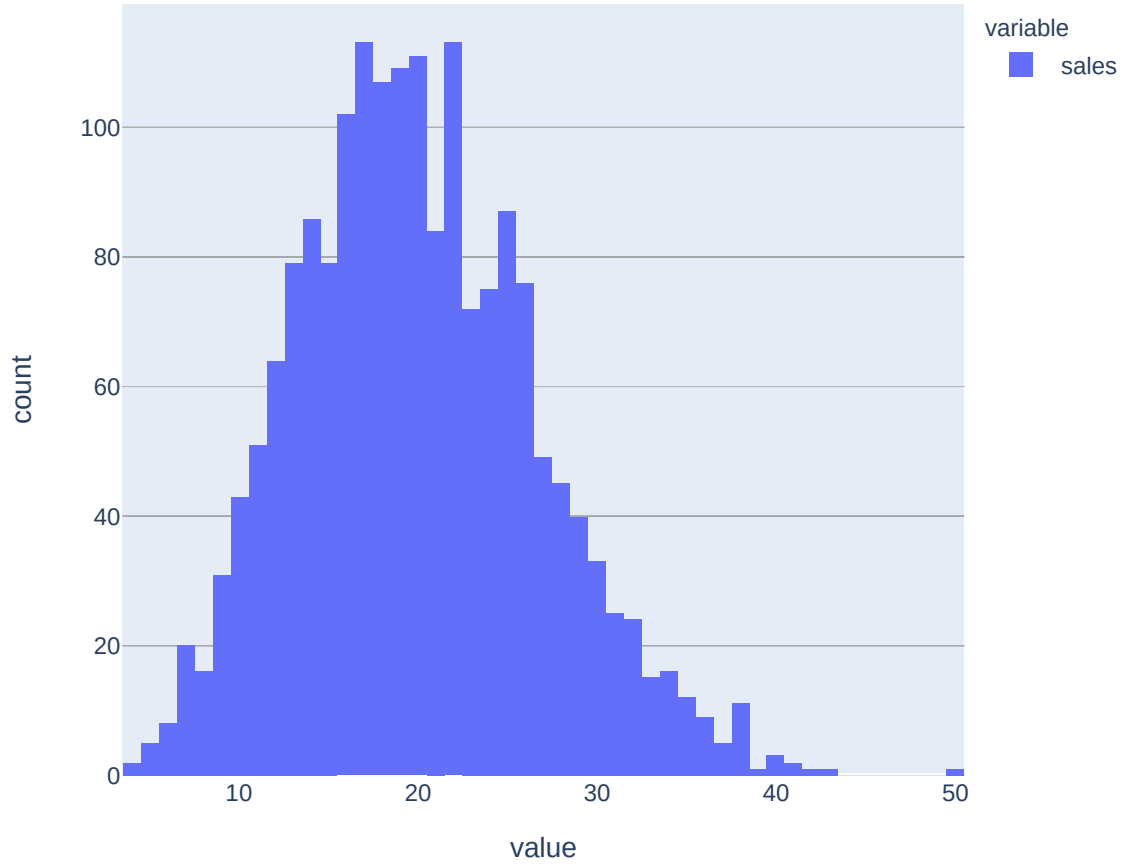
In [78]:
```
from statsmodels.tsa.stattools import adfuller
hypothesis_test=adfuller(df_1_1)
print('ADF Statistic: %f' % hypothesis_test[0])
print('p-value: %f' % hypothesis_test[1])
print('Critical Values:')
for key, value in hypothesis_test[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -3.157671
p-value: 0.022569
Critical Values:
        1%: -3.434
        5%: -2.863
        10%: -2.568
```

The more negative this statistic, the more likely we are to reject the null hypothesis

Histogram

In [77]:
```
fig = px.histogram(df_1_1)
fig.show()
```



p<0.05 so we reject the null hypothesis and the Time series is Stationary

In [80]:
```
df_1_1.diff(periods=1).fillna(0).head()
```

Out[80]:

date	sales
2013-01-01	0.0
2013-01-02	-2.0
2013-01-03	3.0
2013-01-04	-1.0
2013-01-05	-3.0

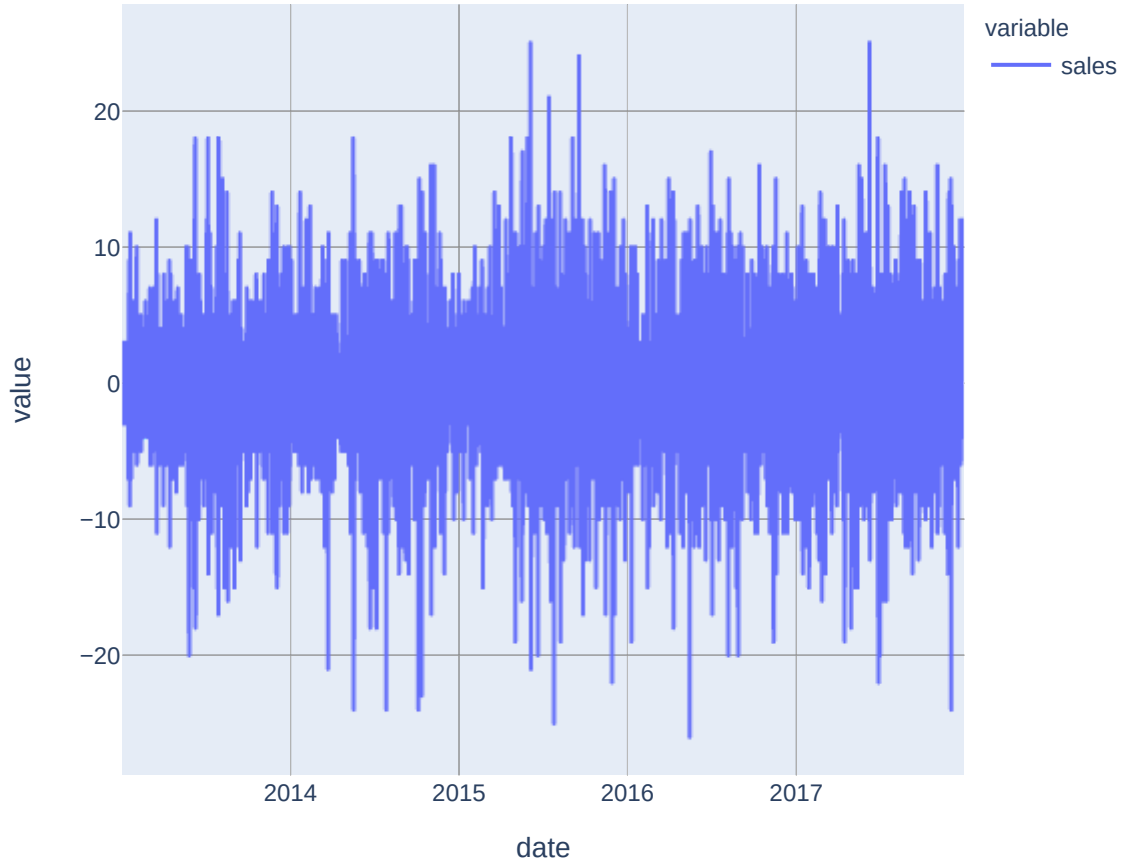
In [90]:
```
df_diff=df_1_1.diff(periods=1) #Integrated of order 1 denoted by d
```

In [91]:
```
df_diff=df_diff[1:]
df_diff.head()## 1 Lag
```

Out[91]:

date	sales
2013-01-02	-2.0
2013-01-03	3.0
2013-01-04	-1.0
2013-01-05	-3.0
2013-01-06	2.0

In [93]:
```
fig = px.line(df_diff)
fig.show()
```



ARIMA Model

In [130]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [132]:
```
import itertools
p=d=q=range(0,5)
pdq =list(itertools.product(p,d,q))
```

In [318]:
```
X = df_1_1.values
size = int(len(X) * 0.66)
predictions = []
```

In [319]:
```
X = df_1_1.values
size = int(len(X) * 0.88)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
```

In [476]:
```
train_date=df_1_1.index[0:size]
test_date=df_1_1.index[size:len(X)]
```

In [432]:
```
import warnings
warnings.filterwarnings("ignore")
AIC={}
for i in pdq:
    try:
        model_arima=ARIMA(train,order=(i))
        model_fit=model_arima.fit()
        print(model_fit.aic," ",i)
        AIC[model_fit.aic]=i
    except:
        continue
```

```
10589.258825757217   (0, 0, 0)
10367.211558053608   (0, 0, 1)
10292.248736752234   (0, 0, 2)
10235.806148907104   (0, 0, 3)
10225.53433820196   (0, 0, 4)
10791.119185694371   (0, 1, 0)
9910.661190802963   (0, 1, 1)
9906.84699497884   (0, 1, 2)
9903.313527540431   (0, 1, 3)
9887.288179269828   (0, 1, 4)
12476.917019606857   (0, 2, 0)
10794.80988182853   (0, 2, 1)
9921.970189746653   (0, 2, 2)
9916.711356680364   (0, 2, 3)
9914.84604318292   (0, 2, 4)
14367.532990346645   (0, 3, 0)
12479.525622583456   (0, 3, 1)
10834.470192288718   (0, 3, 2)
9952.485621226266   (0, 3, 3)
9965.081156514974   (0, 3, 4)
16347.362177940278   (0, 4, 0)
14369.095427634598   (0, 4, 1)
12488.980190056529   (0, 4, 2)
10837.610179147403   (0, 4, 3)
10002.567785570867   (0, 4, 4)
10262.95280427559   (1, 0, 0)
9916.577889625332   (1, 0, 1)
9912.961295776411   (1, 0, 2)
9909.156376233652   (1, 0, 3)
9892.166745015093   (1, 0, 4)
10454.955245295752   (1, 1, 0)
9907.543771792863   (1, 1, 1)
9903.903615886484   (1, 1, 2)
9885.426266661207   (1, 1, 3)
9881.724509932501   (1, 1, 4)
11650.873403997504   (1, 2, 0)
10459.542922283486   (1, 2, 1)
9917.480272945557   (1, 2, 2)
9918.933674940105   (1, 2, 3)
9917.292938197575   (1, 2, 4)
13142.511008412042   (1, 3, 0)
```

```
11654.974896461197   (1, 3, 1)
10526.70242984685   (1, 3, 2)
9940.775287465414   (1, 3, 3)
9944.664629833009   (1, 3, 4)
14803.393185873258   (1, 4, 0)
13145.781892805095   (1, 4, 1)
11666.101474907216   (1, 4, 2)
10504.482584679805   (1, 4, 3)
10850.809896424398   (1, 4, 4)
10192.108054857954   (2, 0, 0)
9913.646873567626   (2, 0, 1)
9909.915140200457   (2, 0, 2)
9911.675289994873   (2, 0, 3)
9911.511106776352   (2, 0, 4)
10312.527638268388   (2, 1, 0)
9902.914260425598   (2, 1, 1)
9905.65318892506   (2, 1, 2)
9693.782579994118   (2, 1, 3)
9887.639858494502   (2, 1, 4)
11252.82963837537   (2, 2, 0)
10317.800239781722   (2, 2, 1)
9914.08666767153   (2, 2, 2)
9916.468478605551   (2, 2, 3)
9861.828477360548   (2, 2, 4)
12470.32062758406   (2, 3, 0)
11257.954063357294   (2, 3, 1)
10346.104224553585   (2, 3, 2)
10480.154689374549   (2, 3, 3)
9938.404191069203   (2, 3, 4)
13892.129959259426   (2, 4, 0)
12474.928515483165   (2, 4, 1)
11270.542304692239   (2, 4, 2)
10383.760867684408   (2, 4, 3)
10497.099902944987   (2, 4, 4)
10151.265296728385   (3, 0, 0)
9908.848364918422   (3, 0, 1)
9911.706257867225   (3, 0, 2)
9877.72674207166   (3, 0, 3)
9856.244707394999   (3, 0, 4)
10230.23918213157   (3, 1, 0)
9891.149376016601   (3, 1, 1)
9873.280401597656   (3, 1, 2)
```

```
9709.843323356283   (3, 1, 3)
9700.335924303288   (3, 1, 4)
11042.885219536534   (3, 2, 0)
10235.885480505436   (3, 2, 1)
10321.702580390162   (3, 2, 2)
10321.6824455278   (3, 2, 3)
9917.591412435415   (3, 2, 4)
12072.667002649938   (3, 3, 0)
11048.740065088943   (3, 3, 1)
10252.641717549435   (3, 3, 2)
10338.126556536066   (3, 3, 3)
10486.871830352453   (3, 3, 4)
13337.292125351061   (3, 4, 0)
12078.294986494831   (3, 4, 1)
11062.285642526138   (3, 4, 2)
11274.011161711644   (3, 4, 3)
10366.579572575152   (3, 4, 4)
10121.972354969019   (4, 0, 0)
9896.778058759293   (4, 0, 1)
9911.27774058084   (4, 0, 2)
9858.376898248125   (4, 0, 3)
9899.013138011836   (4, 0, 4)
10144.47853994085   (4, 1, 0)
9869.833703642586   (4, 1, 1)
9876.680692081129   (4, 1, 2)
9864.348718295296   (4, 1, 3)
9698.929897220964   (4, 1, 4)
10895.908496291198   (4, 2, 0)
10150.58841990859   (4, 2, 1)
9878.297722728545   (4, 2, 2)
9901.193422043772   (4, 2, 3)
9828.191278816097   (4, 2, 4)
11742.780886436118   (4, 3, 0)
10902.377977363085   (4, 3, 1)
10167.480042083655   (4, 3, 2)
10251.642351779854   (4, 3, 3)
10334.31183005539   (4, 3, 4)
12722.90555233062   (4, 4, 0)
11749.33318349488   (4, 4, 1)
10916.339902511754   (4, 4, 2)
11060.75356304982   (4, 4, 3)
11187.403635341838   (4, 4, 4)
```

In [437]:
```
AIC[min(AIC.keys())]
```

Out[437]:
```
(2, 1, 3)
```

In [439]:
```
model_arima=ARIMA(train,order=(2,1,3))
model_fit=model_arima.fit()
```

Selecting the parameter (p,d,q) -> (4,3,2) as it has minimum AIC

In [442]:
```
model_fit.summary()
```

Out[442]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1606
Model:	ARIMA(2, 1, 3)	**Log Likelihood**	-4840.891
Date:	Sun, 30 May 2021	**AIC**	9693.783
Time:	18:40:11	**BIC**	9726.068
Sample:	0	**HQIC**	9705.769
	- 1606		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.2456	0.002	717.017	0.000	1.242	1.249
ar.L2	-0.9982	0.002	-565.724	0.000	-1.002	-0.995
ma.L1	-2.1451	0.012	-173.126	0.000	-2.169	-2.121
ma.L2	2.0999	0.018	115.065	0.000	2.064	2.136
ma.L3	-0.8876	0.012	-72.236	0.000	-0.912	-0.863
sigma2	24.3634	0.811	30.041	0.000	22.774	25.953

Ljung-Box (L1) (Q):	3.73	**Jarque-Bera (JB):**	9.08
Prob(Q):	0.05	**Prob(JB):**	0.01
Heteroskedasticity (H):	1.29	**Skew:**	0.11
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.30

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [443]:
```
residuals=pd.DataFrame(model_fit.resid)
residuals.plot()
print(residuals.describe())
```

```
                0
count  1606.000000
mean      0.099039
std       4.944046
min     -17.094609
25%      -3.202408
50%      -0.008339
75%       3.177371
max      19.359866
```

In [451]:
```
predictions=[]
# walk-forward validation
for t in range(len(test)):
        model = ARIMA(history, order=(2,1,3))
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        obs = test[t]
        history.append(obs)
        print('predicted=%f, expected=%f' % (yhat, obs))
# evaluate forecasts
```

```
predicted=17.605073, expected=24.000000
predicted=16.054515, expected=35.000000
predicted=16.916438, expected=33.000000
predicted=19.192123, expected=23.000000
predicted=0.000000, expected=17.000000
predicted=22.678341, expected=20.000000
predicted=23.291576, expected=29.000000
predicted=23.298720, expected=25.000000
predicted=21.414487, expected=36.000000
predicted=20.647347, expected=27.000000
predicted=22.178445, expected=22.000000
predicted=23.026424, expected=22.000000
predicted=24.817104, expected=29.000000
predicted=26.964763, expected=26.000000
predicted=26.472314, expected=16.000000
predicted=23.114099, expected=41.000000
predicted=23.961148, expected=28.000000
predicted=23.628028, expected=23.000000
predicted=24.165437, expected=26.000000
predicted=26.866655, expected=32.000000
predicted=28.302701, expected=30.000000
predicted=28.528563, expected=26.000000
predicted=27.058535, expected=31.000000
predicted=25.656097, expected=38.000000
predicted=26.355099, expected=30.000000
predicted=27.199176, expected=22.000000
predicted=27.711751, expected=26.000000
predicted=29.090700, expected=24.000000
predicted=28.928539, expected=26.000000
predicted=28.158254, expected=30.000000
predicted=26.997111, expected=26.000000
predicted=25.679679, expected=21.000000
predicted=24.812970, expected=32.000000
predicted=26.946321, expected=50.000000
predicted=31.469115, expected=28.000000
predicted=30.308564, expected=28.000000
predicted=29.150521, expected=31.000000
predicted=28.497006, expected=38.000000
predicted=29.440991, expected=18.000000
predicted=28.169173, expected=23.000000
predicted=28.734854, expected=23.000000
```

```
predicted=29.461272, expected=31.000000
predicted=29.977460, expected=32.000000
predicted=29.352791, expected=27.000000
predicted=27.421348, expected=32.000000
predicted=27.017696, expected=16.000000
predicted=25.797954, expected=23.000000
predicted=26.736239, expected=29.000000
predicted=29.118780, expected=22.000000
predicted=28.453124, expected=38.000000
predicted=28.988728, expected=40.000000
predicted=28.164154, expected=36.000000
predicted=26.226962, expected=20.000000
predicted=26.004534, expected=26.000000
predicted=27.992534, expected=24.000000
predicted=30.109365, expected=37.000000
predicted=32.473269, expected=27.000000
predicted=30.593423, expected=35.000000
predicted=28.592989, expected=32.000000
predicted=26.790042, expected=27.000000
predicted=26.367820, expected=20.000000
predicted=27.843136, expected=28.000000
predicted=31.020656, expected=35.000000
predicted=32.813498, expected=25.000000
predicted=30.698705, expected=30.000000
predicted=28.129878, expected=26.000000
predicted=25.563168, expected=22.000000
predicted=24.818411, expected=18.000000
predicted=25.971307, expected=19.000000
predicted=28.057340, expected=25.000000
predicted=29.497604, expected=26.000000
predicted=28.271638, expected=35.000000
predicted=26.409100, expected=29.000000
predicted=23.848041, expected=20.000000
predicted=22.402970, expected=28.000000
predicted=25.077388, expected=28.000000
predicted=28.531849, expected=31.000000
predicted=30.731857, expected=25.000000
predicted=29.334956, expected=32.000000
predicted=27.193613, expected=32.000000
predicted=25.160667, expected=26.000000
predicted=24.302521, expected=18.000000
predicted=25.002262, expected=24.000000
```

```
predicted=27.981973, expected=21.000000
predicted=29.356110, expected=35.000000
predicted=30.072398, expected=29.000000
predicted=27.270839, expected=27.000000
predicted=24.111291, expected=19.000000
predicted=22.175007, expected=28.000000
predicted=24.713495, expected=26.000000
predicted=27.846662, expected=15.000000
predicted=28.386774, expected=30.000000
predicted=28.838302, expected=38.000000
predicted=27.770376, expected=26.000000
predicted=24.150389, expected=15.000000
predicted=21.463191, expected=21.000000
predicted=22.780892, expected=22.000000
predicted=26.163275, expected=26.000000
predicted=28.907805, expected=38.000000
predicted=30.023854, expected=26.000000
predicted=26.200818, expected=23.000000
predicted=22.531809, expected=12.000000
predicted=20.047041, expected=26.000000
predicted=23.009340, expected=24.000000
predicted=26.618342, expected=29.000000
predicted=29.038467, expected=28.000000
predicted=28.300091, expected=28.000000
predicted=25.182902, expected=30.000000
predicted=22.732314, expected=16.000000
predicted=21.063487, expected=20.000000
predicted=22.876962, expected=13.000000
predicted=25.362079, expected=26.000000
predicted=28.041091, expected=28.000000
predicted=27.131444, expected=27.000000
predicted=24.171390, expected=30.000000
predicted=21.352929, expected=22.000000
predicted=20.183387, expected=21.000000
predicted=22.105447, expected=20.000000
predicted=25.669075, expected=20.000000
predicted=27.670266, expected=28.000000
predicted=27.871985, expected=37.000000
predicted=26.034643, expected=24.000000
predicted=21.647986, expected=14.000000
predicted=18.738282, expected=18.000000
predicted=20.938116, expected=27.000000
```

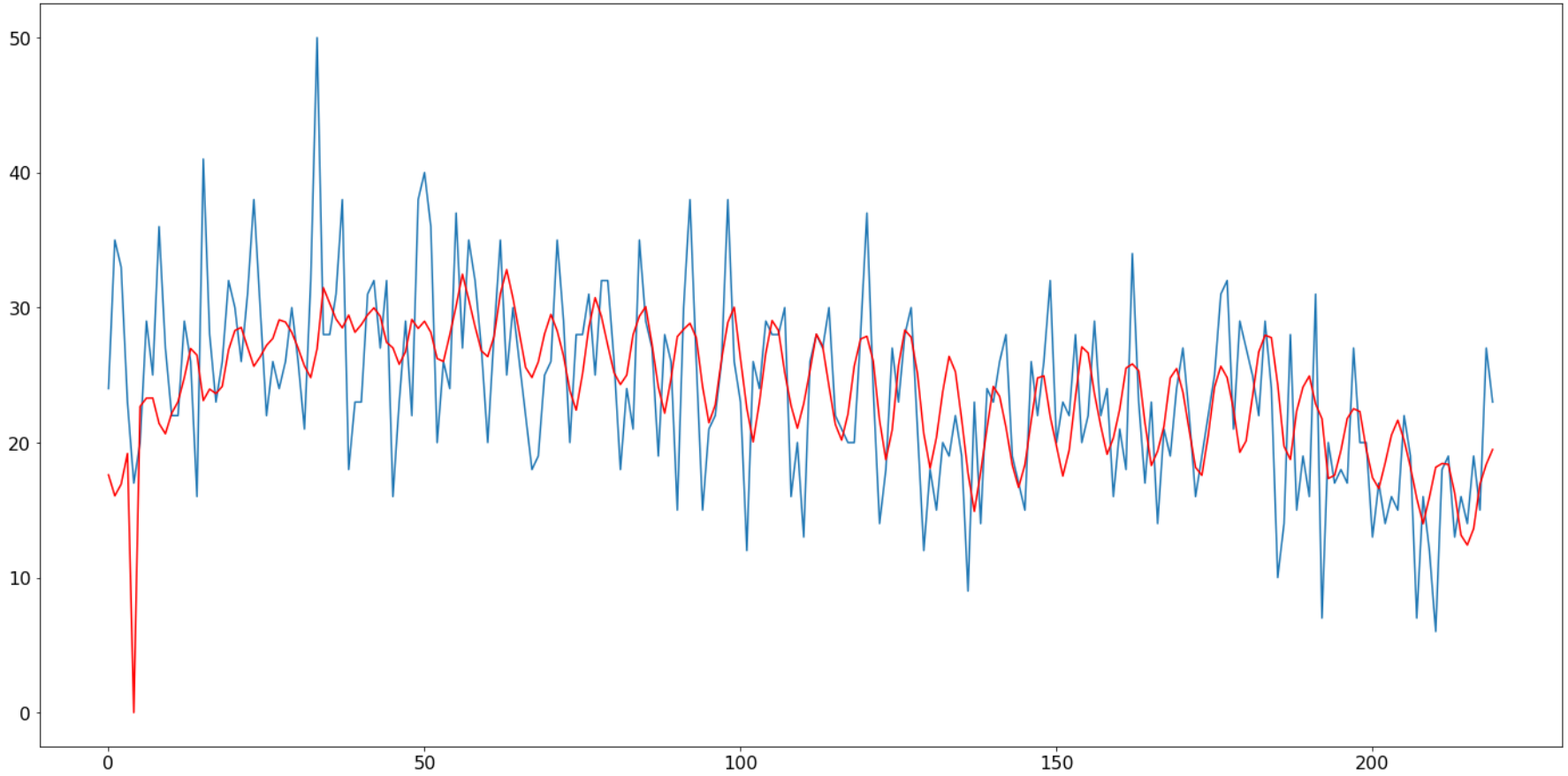
```
predicted=25.717300, expected=23.000000
predicted=28.334515, expected=28.000000
predicted=27.803064, expected=30.000000
predicted=25.115645, expected=21.000000
predicted=20.659527, expected=12.000000
predicted=18.103717, expected=18.000000
predicted=20.367861, expected=15.000000
predicted=23.774822, expected=20.000000
predicted=26.384047, expected=19.000000
predicted=25.253254, expected=22.000000
predicted=21.697896, expected=19.000000
predicted=17.735819, expected=9.000000
predicted=14.900274, expected=23.000000
predicted=17.831590, expected=14.000000
predicted=21.041739, expected=24.000000
predicted=24.152431, expected=23.000000
predicted=23.418739, expected=26.000000
predicted=21.144033, expected=28.000000
predicted=18.339699, expected=19.000000
predicted=16.678259, expected=17.000000
predicted=18.360895, expected=15.000000
predicted=21.620906, expected=26.000000
predicted=24.786536, expected=22.000000
predicted=24.941993, expected=26.000000
predicted=21.939170, expected=32.000000
predicted=19.725708, expected=20.000000
predicted=17.520501, expected=23.000000
predicted=19.422332, expected=22.000000
predicted=23.378093, expected=28.000000
predicted=27.090456, expected=20.000000
predicted=26.640272, expected=22.000000
predicted=23.629347, expected=29.000000
predicted=21.231318, expected=22.000000
predicted=19.132307, expected=24.000000
predicted=20.342916, expected=16.000000
predicted=22.470392, expected=21.000000
predicted=25.517013, expected=18.000000
predicted=25.822169, expected=34.000000
predicted=25.326071, expected=23.000000
predicted=21.438731, expected=17.000000
predicted=18.300932, expected=23.000000
predicted=19.370160, expected=14.000000
```

```
predicted=21.201959, expected=21.000000
predicted=24.792218, expected=19.000000
predicted=25.478497, expected=24.000000
predicted=23.734604, expected=27.000000
predicted=20.897372, expected=22.000000
predicted=18.160841, expected=16.000000
predicted=17.574929, expected=19.000000
predicted=20.510791, expected=22.000000
predicted=24.112190, expected=25.000000
predicted=25.655639, expected=31.000000
predicted=24.810785, expected=32.000000
predicted=22.453309, expected=21.000000
predicted=19.277023, expected=29.000000
predicted=20.123253, expected=27.000000
predicted=23.383394, expected=25.000000
predicted=26.717069, expected=22.000000
predicted=27.953145, expected=29.000000
predicted=27.772131, expected=24.000000
predicted=24.295961, expected=10.000000
predicted=19.724285, expected=14.000000
predicted=18.747219, expected=28.000000
predicted=22.343041, expected=15.000000
predicted=24.109100, expected=19.000000
predicted=24.933318, expected=16.000000
predicted=22.850260, expected=31.000000
predicted=21.770390, expected=7.000000
predicted=17.338230, expected=20.000000
predicted=17.581564, expected=17.000000
predicted=19.407204, expected=18.000000
predicted=21.776960, expected=17.000000
predicted=22.509550, expected=27.000000
predicted=22.296695, expected=20.000000
predicted=19.537201, expected=20.000000
predicted=17.403851, expected=13.000000
predicted=16.594954, expected=17.000000
predicted=18.484043, expected=14.000000
predicted=20.556427, expected=16.000000
predicted=21.656242, expected=15.000000
predicted=20.165449, expected=22.000000
predicted=18.185052, expected=19.000000
predicted=15.873367, expected=7.000000
predicted=13.986835, expected=16.000000
```

```
predicted=15.953312, expected=12.000000
predicted=18.168554, expected=6.000000
predicted=18.447297, expected=18.000000
predicted=18.387629, expected=19.000000
predicted=16.264984, expected=13.000000
predicted=13.144562, expected=16.000000
predicted=12.408569, expected=14.000000
predicted=13.623812, expected=19.000000
predicted=16.921150, expected=15.000000
predicted=18.387081, expected=27.000000
predicted=19.482307, expected=23.000000
```

In [452]:
```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
# plot forecasts against actual outcomes
plt.figure(figsize=(24,12))
plt.plot(test)
plt.plot(predictions, color='red')
plt.show()
```

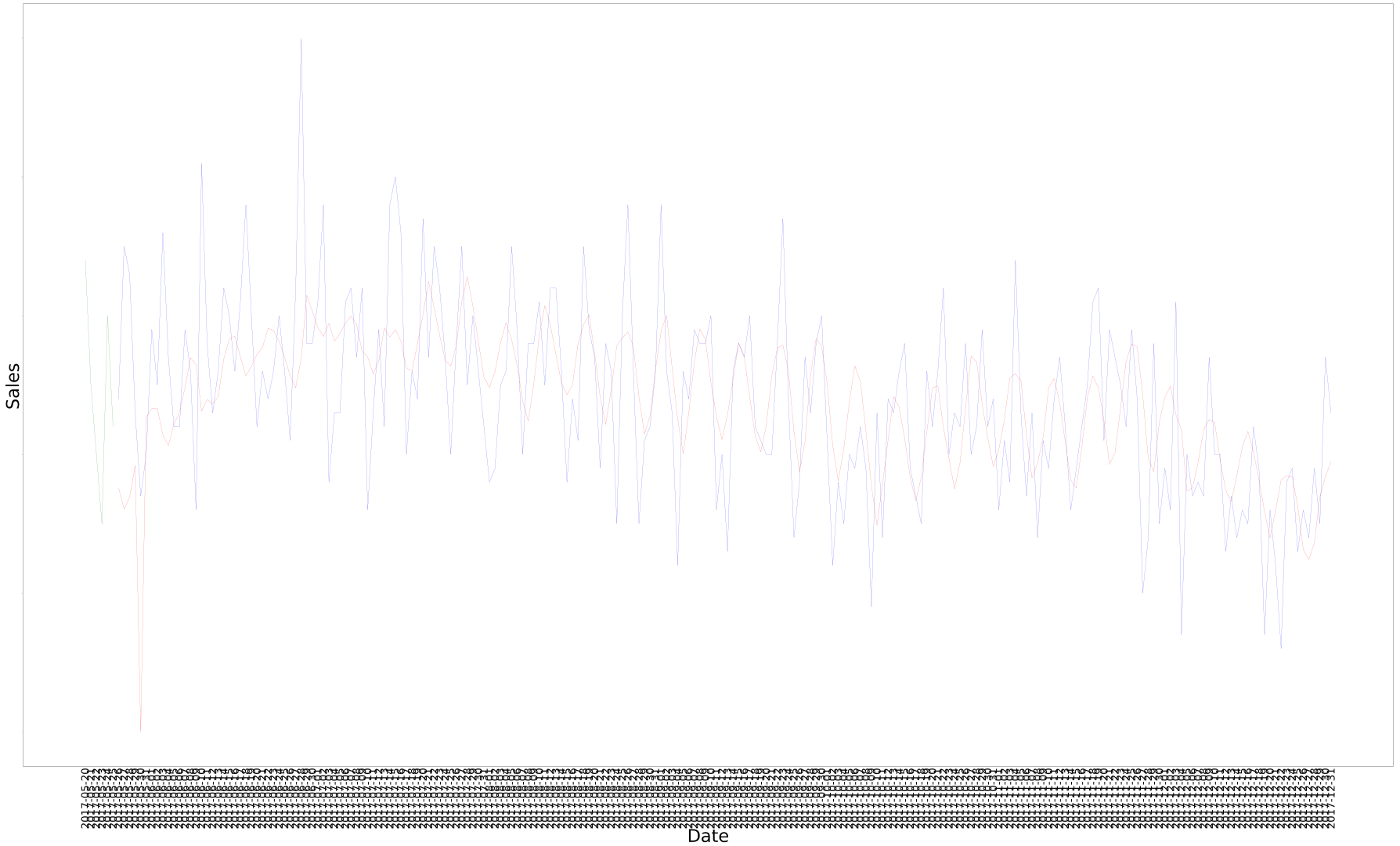
```
Test RMSE: 5.970
```



In [453]:
```
from sklearn.metrics import mean_absolute_error
print('Mean Absolute Error:',mean_absolute_error(test.reshape(-1),predictions))
```

```
Mean Absolute Error: 4.558100554832593
```

In [536]:
```
plt.figure(figsize=(350,200))
plt.plot(train_date[1600:],train[1600:],'green')
plt.plot(test_date,test,'blue')
plt.plot(test_date,predictions,'red')
plt.savefig('A.png')
plt.xticks(rotation = 90,fontsize=150)
plt.xlabel('Date', fontsize=250)
plt.ylabel('Sales', fontsize=250)
plt.savefig('Arima.png');
```



In [528]:
```
#fig = px.line(train_date,train)
#fig.show()
%matplotlib inline
```

In [503]:
```
df_pred=pd.DataFrame({'Predictions':predictions},index=test_date)
```

In [511]:
```
#df_1_1.merge(df_pred,'inner',on='index')
```

In [ ]: