In [289…
```
# Importing libraries for data manipulation
import pandas as pd   # Used for data manipulation and analysis
import numpy as np     # Used for numerical computations

# Importing libraries for data visualization
import seaborn as sns  # Used for creating statistical graphics
from matplotlib import pyplot as plt  # Used for plotting

# Importing time series models from statsmodels
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt

# ExponentialSmoothing: Triple exponential smoothing (Holt-Winters method) for t
# SimpleExpSmoothing: Single exponential smoothing for trend only
# Holt: Double exponential smoothing for trend (Holt's linear method)

# Importing Linear Regression model from scikit-learn
from sklearn.linear_model import LinearRegression
# LinearRegression: Used to fit linear models to the data for predicting target

# Optional: Suppressing warnings for clean output
import warnings
warnings.filterwarnings('ignore')  # Used to ignore unnecessary warnings
```

In [19]:
```
#Loading Data Set from machine
df = pd.read_csv(r"C:\Users\10500\OneDrive\Desktop\gold_monthly_csv.csv")
```

In [21]:
```
# Check the first few rows of the dataset
df.head()
```

Out[21]:

	Date	Price
0	1950-01	34.73
1	1950-02	34.73
2	1950-03	34.73
3	1950-04	34.73
4	1950-05	34.73

In [25]:
```
df.shape
```

Out[25]: (847, 2)

In [31]:
```
print(f"Date range of gold prices available from - {df.loc[:, 'Date'][0]} to {df
```

```
Date range of gold prices available from - 1950-01 to 2020-07
```

In [33]:
```
date = pd.date_range (start = '1/1/1950', end = '8/1/2020', freq='M')
date
```

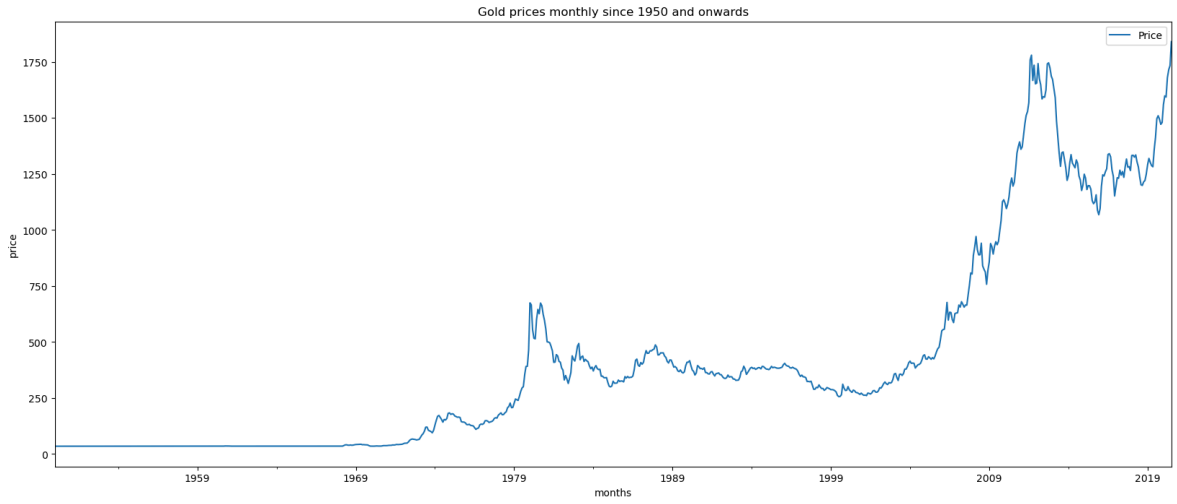
Out[33]:
```
DatetimeIndex(['1950-01-31', '1950-02-28', '1950-03-31', '1950-04-30',
               '1950-05-31', '1950-06-30', '1950-07-31', '1950-08-31',
               '1950-09-30', '1950-10-31',
               ...
               '2019-10-31', '2019-11-30', '2019-12-31', '2020-01-31',
               '2020-02-29', '2020-03-31', '2020-04-30', '2020-05-31',
               '2020-06-30', '2020-07-31'],
              dtype='datetime64[ns]', length=847, freq='ME')
```

In [37]:
```
df['month']= date
df.drop('Date', axis = 1, inplace = True )
df = df.set_index('month')
df.head
```

Out[37]:
```
<bound method NDFrame.head of                Price
month
1950-01-31    34.730
1950-02-28    34.730
1950-03-31    34.730
1950-04-30    34.730
1950-05-31    34.730
...              ...
2020-03-31  1593.764
2020-04-30  1680.030
2020-05-31  1715.697
2020-06-30  1734.032
2020-07-31  1840.807

[847 rows x 1 columns]>
```

In [47]:
```
df.plot(figsize=(20,8))
plt.title('Gold prices monthly since 1950 and onwards', )
plt.xlabel('months')
plt.ylabel('price')
```

Out[47]:
```
Text(0, 0.5, 'price')
```

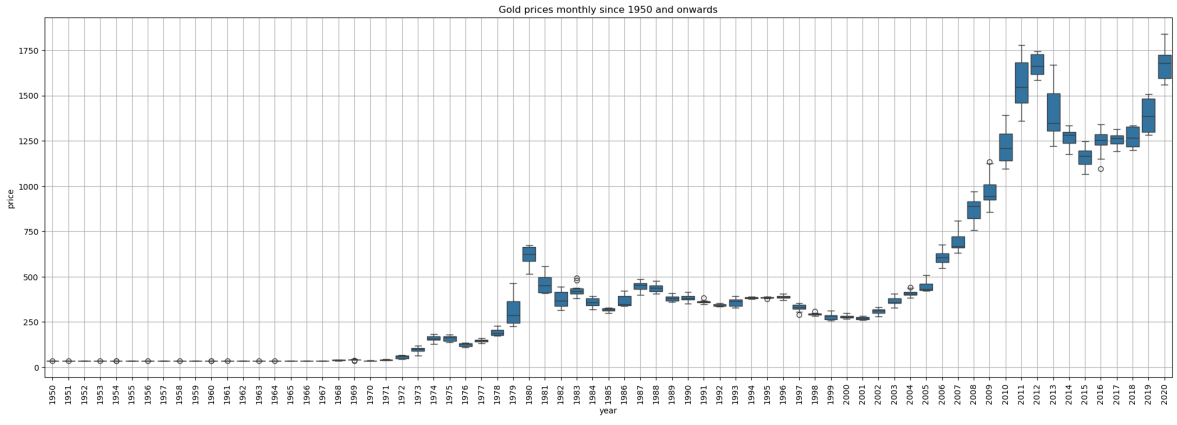


In [49]:
```
round(df.describe(),3)
```

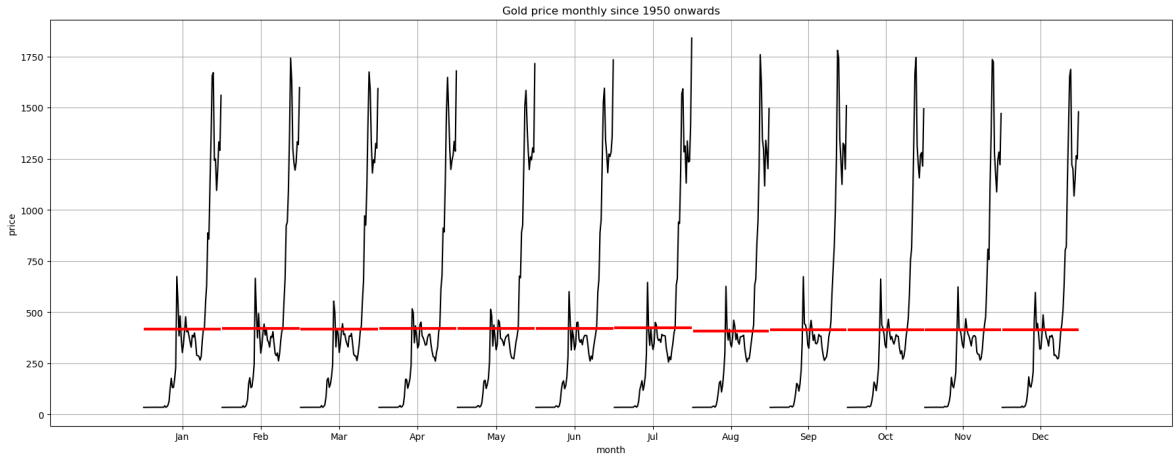
Out[49]:

	Price
count	847.000
mean	416.557
std	453.665
min	34.490
25%	35.190
50%	319.622
75%	447.029
max	1840.807

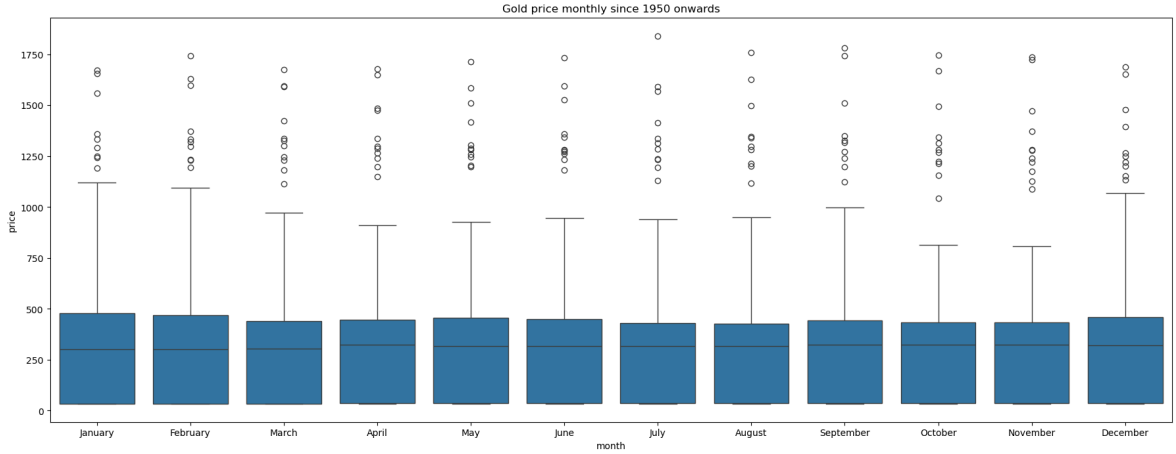
In [53]:
```
_, ax = plt.subplots(figsize=(25,8))
sns.boxplot(x = df.index.year, y = df.values[:,0], ax=ax)
plt.title('Gold prices monthly since 1950 and onwards')
plt.xlabel('year')
plt.ylabel('price')
plt.xticks(rotation = 90)
plt.grid();
```



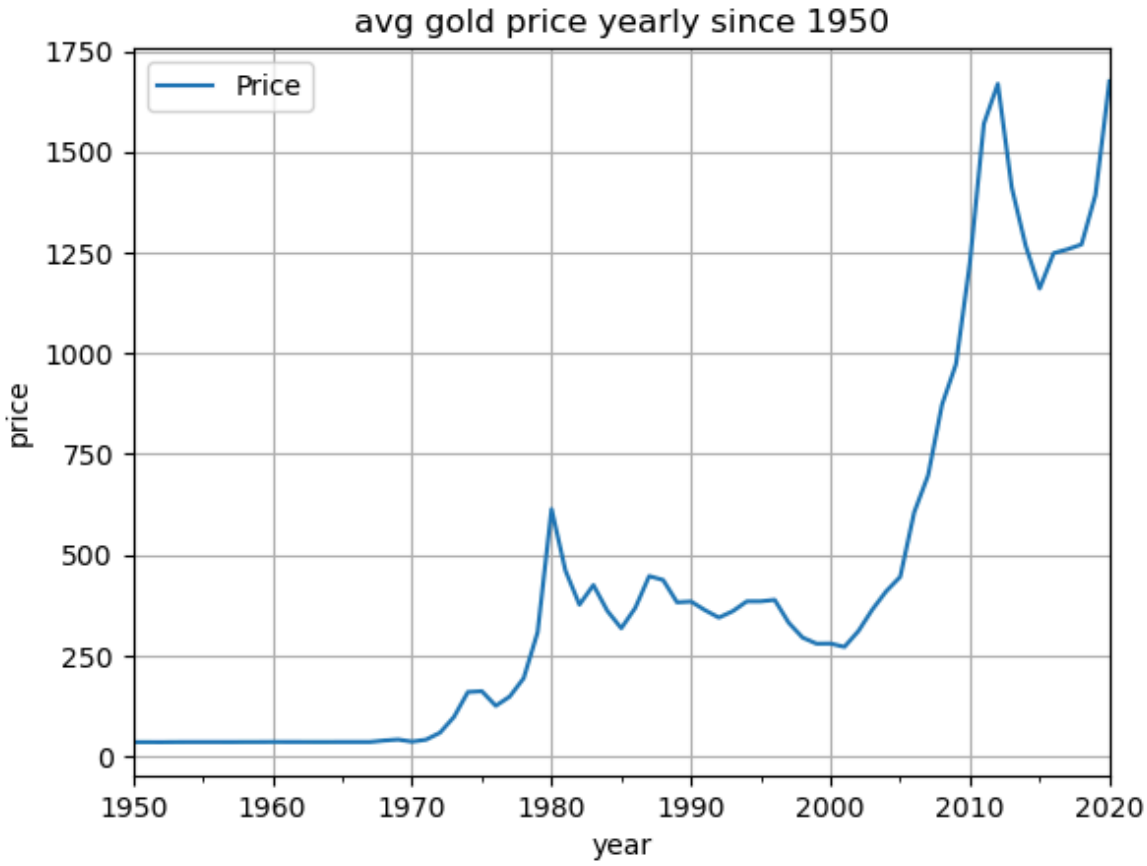
In [59]:
```
from statsmodels.graphics.tsaplots import month_plot

fig, ax = plt.subplots(figsize=(22,8))
month_plot(df, ylabel = 'Gold price', ax=ax)
plt.title('Gold price monthly since 1950 onwards')
plt.xlabel('month')
plt.ylabel('price')
plt.grid();
```



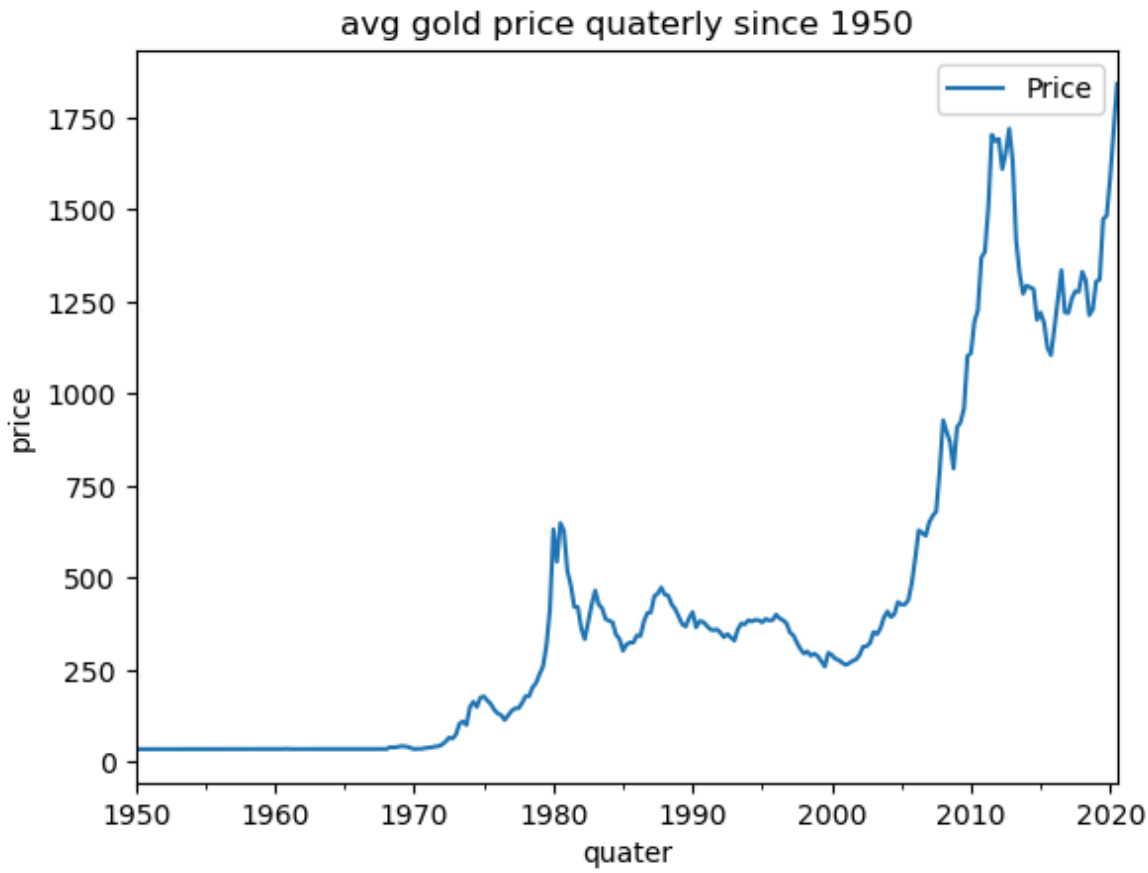
In [67]:
```
_, ax = plt.subplots(figsize=(22,8))
sns.boxplot(x=df.index.month_name(), y = df.values[:, 0], ax=ax)
plt.title('Gold price monthly since 1950 onwards')
plt.xlabel('month')
plt.ylabel('price')
plt.show();
```



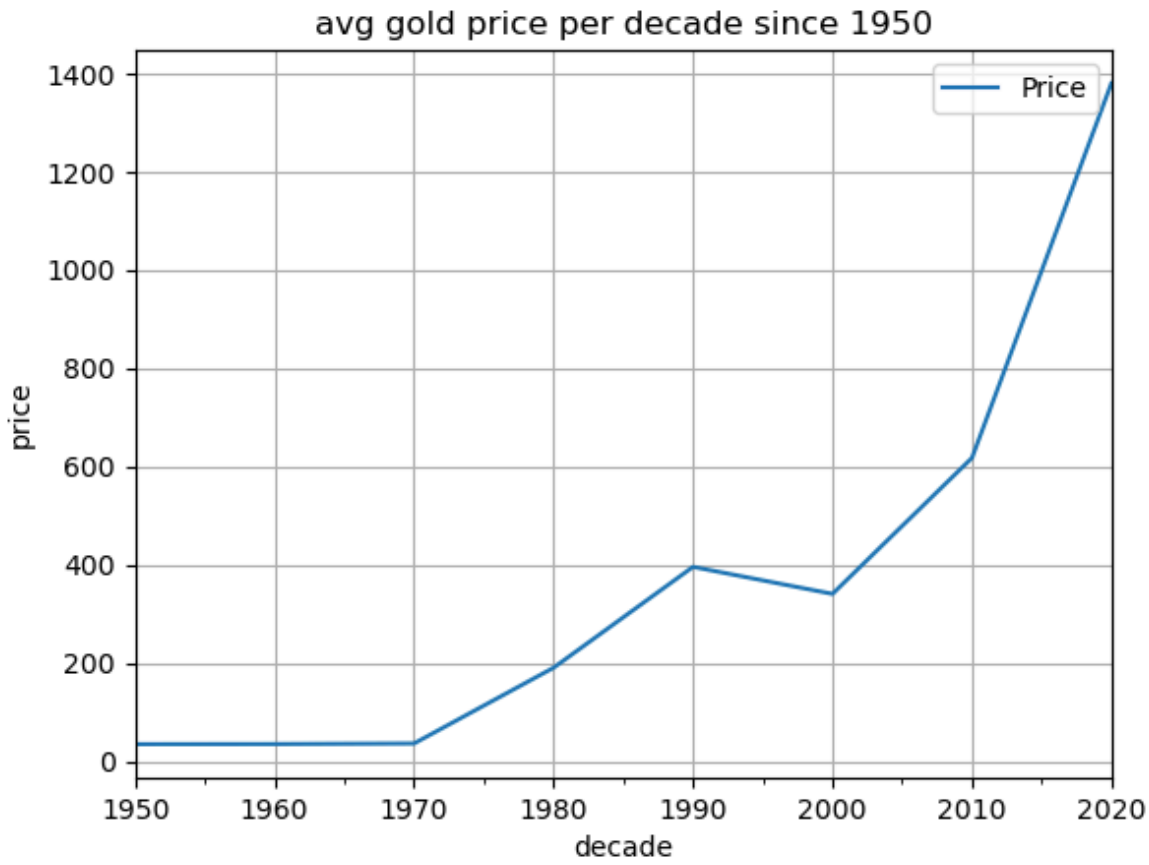
In [73]:
```
df_yearly_sum = df.resample('A').mean()
df_yearly_sum.plot();
plt.title('avg gold price yearly since 1950')
plt.xlabel('year')
plt.ylabel('price')
plt.grid();
```



In [75]:
```
df_quaterly_sum = df.resample('Q').mean()
df_quaterly_sum.plot();
plt.title('avg gold price quaterly since 1950')
plt.xlabel('quater')
plt.ylabel('price')
plt.show();
```



In [77]:
```
df_decade_sum = df.resample('10Y').mean()
df_decade_sum.plot();
plt.title('avg gold price per decade since 1950')
plt.xlabel('decade')
plt.ylabel('price')
plt.grid();
```



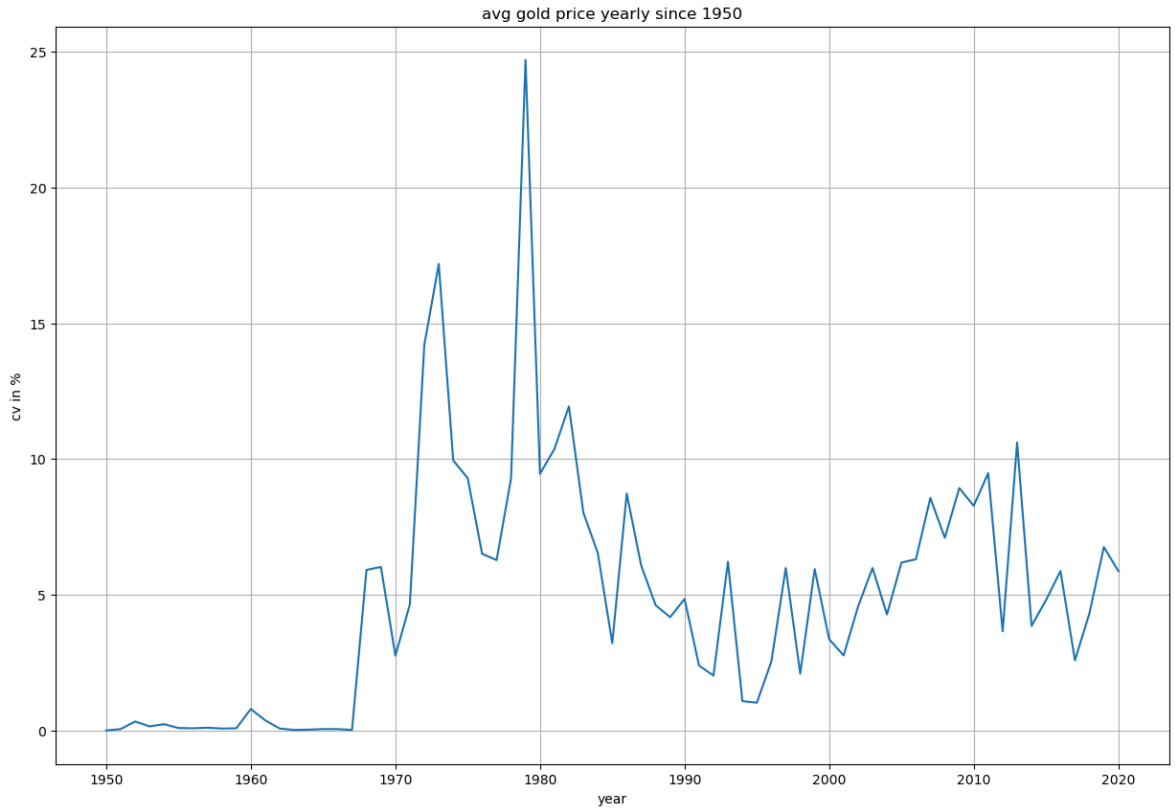
In [139…

```
# Group by year and calculate the mean
df_1 = df.groupby(df.index.year).mean().rename(columns={'Price': 'Mean'})

# Group by year and calculate the standard deviation
df_std = df.groupby(df.index.year).std().rename(columns={'Price': 'std_dev'})

# Merge the mean and standard deviation dataframes
df_1 = df_1.merge(df_std, left_index=True, right_index=True)

# Calculate the coefficient of variation
df_1['Cov_pct'] = ((df_1['std_dev'] / df_1['Mean']) * 100).round(2)

# Display the result
print(df_1.head())
```

```
             Mean   std_dev  Cov_pct
month
1950    34.729167  0.002887     0.01
1951    34.717500  0.020057     0.06
1952    34.628333  0.117538     0.34
1953    34.879167  0.056481     0.16
1954    35.020000  0.082792     0.24
```

In [145…

```
fig, ax= plt.subplots(figsize= (15,10))
df_1['Cov_pct'].plot();
plt.title('avg gold price yearly since 1950')
plt.xlabel('year')
plt.ylabel('cv in %')
plt.grid()
```



In [167…
```
train = df[df.index.year <= 2015]
test = df[df.index.year > 2015]
```

In [171…
```
print(train.shape)
```

```
(792, 1)
```

In [169…
```
print(test.shape)
```

```
(55, 1)
```

In [177…
```
train['Price'].plot(figsize=(13,3), fontsize = 15)
test['Price'].plot(figsize=(13,5), fontsize = 15)
plt.grid()
plt.legend(['Training Data', 'Test Data'])
plt.show()
```

In [181…
```
train_time = [i+1 for i in range(len(train))]
test_time = [i+len(train)+1 for i in range(len(test))]
```

```
len(train_time), len(test_time)
```

Out[181… (792, 55)

In [183…
```
LR_train = train.copy()
LR_test = test.copy()
```

In [185…
```
LR_train['time'] = train_time
LR_test['time'] = test_time
```

In [187…
```
lr = LinearRegression()
lr.fit(LR_train[['time']],LR_train['Price'].values)
```

Out[187…

LinearRegression

LinearRegression()

In [195…
```
test_predictions_model1 = lr.predict(LR_test[['time']])
LR_test['forecast'] = test_predictions_model1

plt.figure(figsize = (14,6))
plt.plot(train['Price'],label = 'train')
plt.plot(test['Price'],label = 'test')
plt.plot(LR_test['forecast'],label = 'reg on time_test data')
plt.legend(loc = 'best')
plt.grid();
```

In [201…
```
def mape(actual, pred) :
    return round((np.mean(abs(actual - pred)/actual))*100, 2)
```

In [203…
```
mape_model1_test = mape(test['Price'].values, test_predictions_model1)
print('MAPE is %3.3f'%(mape_model1_test),'%')
```

MAPE is 29.760 %

In [209…
```
results = pd.DataFrame({'Test Mape (%)' : [mape_model1_test]}, index= ['Regressi
results
```

Out[209…

	Test Mape (%)
RegressionOnTime	29.76

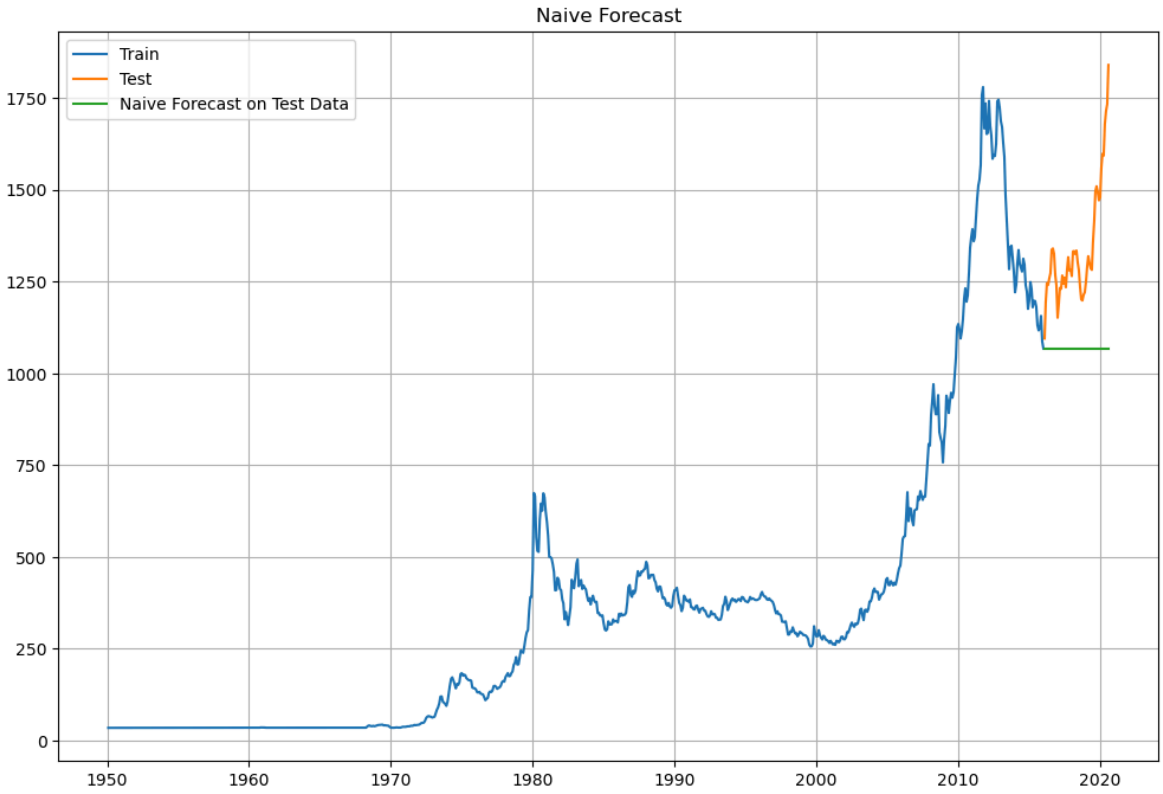
In [223…
```
Naive_train = train.copy()
```

In [225…
```
Naive_train = train.copy()
Naive_test = test.copy()
```

In [237…
```
Naive_test['Naive'] = np.asarray(train['Price'])[len(np.asarray(train['Price']))
Naive_test['Naive'].head()
```

Out[237…
```
month
2016-01-31    1068.317
2016-02-29    1068.317
2016-03-31    1068.317
2016-04-30    1068.317
2016-05-31    1068.317
Name: Naive, dtype: float64
```

In [241…
```
plt.figure(figsize=(12,8))
plt.plot(Naive_train['Price'], label = 'Train')
plt.plot(test['Price'], label = 'Test')
plt.plot(Naive_test['Naive'], label = 'Naive Forecast on Test Data')
plt.legend(loc='best')
plt.title('Naive Forecast')
plt.grid()
```



In [243…
```
mape_model2_test = mape(test['Price'].values, Naive_test['Naive'].values)
print('for Naive forecast on the Test Data, MAPE is %3.3f'%(mape_model2_test),'%
```

```
for Naive forecast on the Test Data, MAPE is 19.380 %
```

In [259…
```
resultsDf_2 = pd.DataFrame({'Test MAPE (%)': [mape_model2_test]}, index=['NaiveM
results = pd.concat([results, resultsDf_2])
results
```

Out[259…

	Test Mape (%)	Test MAPE (%)
RegressionOnTime	29.76	NaN
NaiveModel	NaN	19.38

In [261…
```
final_model = ExponentialSmoothing(df,
                                   trend = 'additive',
                                   seasonal = 'additive' ).fit(smoothing_level =
                                                               smoothing_trend = 0
                                                               smoothing_seasonal
```

In [267…
```
Mape_final_model = mape(df['Price'].values, final_model.fittedvalues)
print('MAPE:', Mape_final_model)
```

```
MAPE: 17.24
```

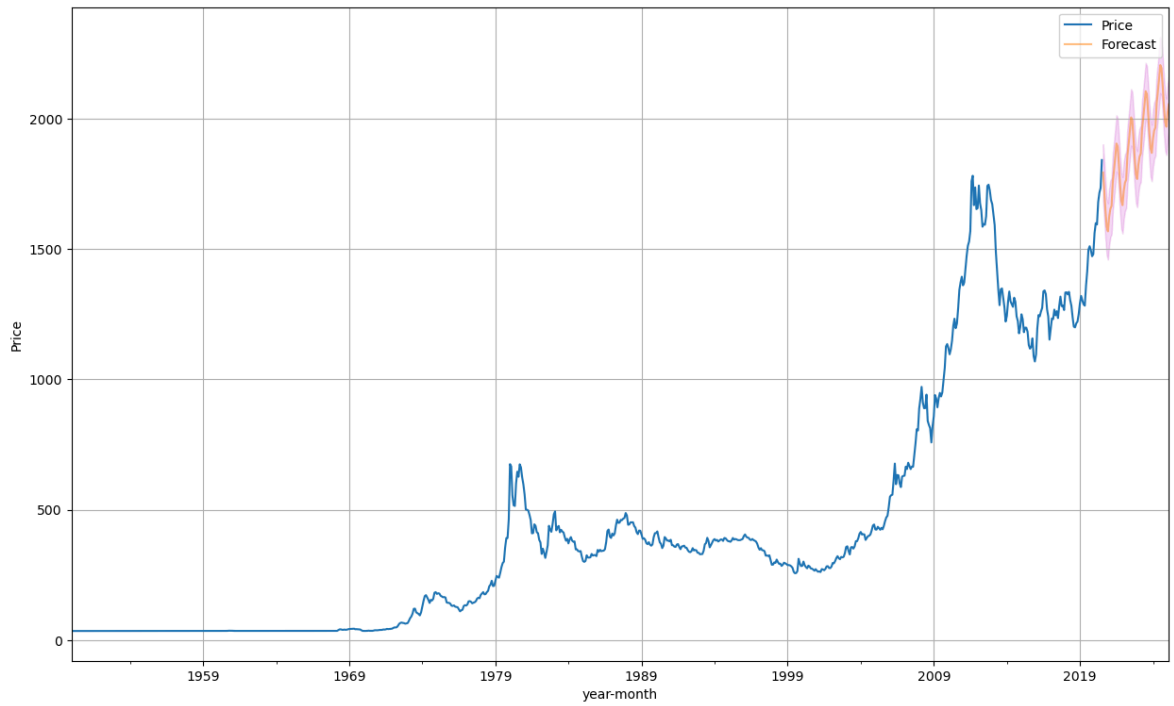
In [269…
```
predictions = final_model.forecast(steps=len(test))
```

In [279…
```
pred_df = pd.DataFrame({'lower_CI': predictions - 1.96*np.std(final_model.resid,
                        'prediction' : predictions,
                        'upper_CI': predictions+ 1.96*np.std(final_model.resid,dd
pred_df.head()
```

Out[279…

	lower_CI	prediction	upper_CI
2020-08-31	1684.718274	1792.869246	1901.020219
2020-09-30	1615.301815	1723.452788	1831.603760
2020-10-31	1538.560879	1646.711851	1754.862823
2020-11-30	1476.748832	1584.899804	1693.050776
2020-12-31	1459.315210	1567.466182	1675.617154

In [287…
```
axis = df.plot(label= 'Actual', figsize=(15,9))
pred_df['prediction'].plot(ax= axis, label = 'Forecast', alpha =0.5)
axis.fill_between(pred_df.index,pred_df['lower_CI'],pred_df['upper_CI'],color =
axis.set_xlabel('year-month')
axis.set_ylabel('Price')
plt.legend(loc = 'best')
plt.grid()
plt.show()
```


Price
Forecast
2000
1500
1000
500
0
Price
1959
1969
1979
1989
1999
2009
2019
year-month

In [ ]: