In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
df = pd.read_csv('/content/drive/MyDrive/gold_monthly_csv.csv')
df.head(10)
```

Out[4]:

	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
5	1950-06	34.73
6	1950-07	34.73
7	1950-08	34.73
8	1950-09	34.73
9	1950-10	34.73

In [5]:

```
df.shape
```

Out[5]:

```
(847, 2)
```

In [7]:

```
print(f"Date range of gold prices available from: {df['Date'].min()} to {df['Date'].max()
}")
```

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

In [8]:

```
date = pd.date_range(start = '1/1/1950', end='8/1/2023',freq='M')
date
```

Out[8]:

```
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',
               ...
               '2022-10-31', '2022-11-30', '2022-12-31', '2023-01-31',
               '2023-02-28', '2023-03-31', '2023-04-30', '2023-05-31',
               '2023-06-30', '2023-07-31'],
```

```
               '2023-06-30', '2023-07-31'],
              dtype='datetime64[ns]', length=883, freq='ME')
```

In [9]:

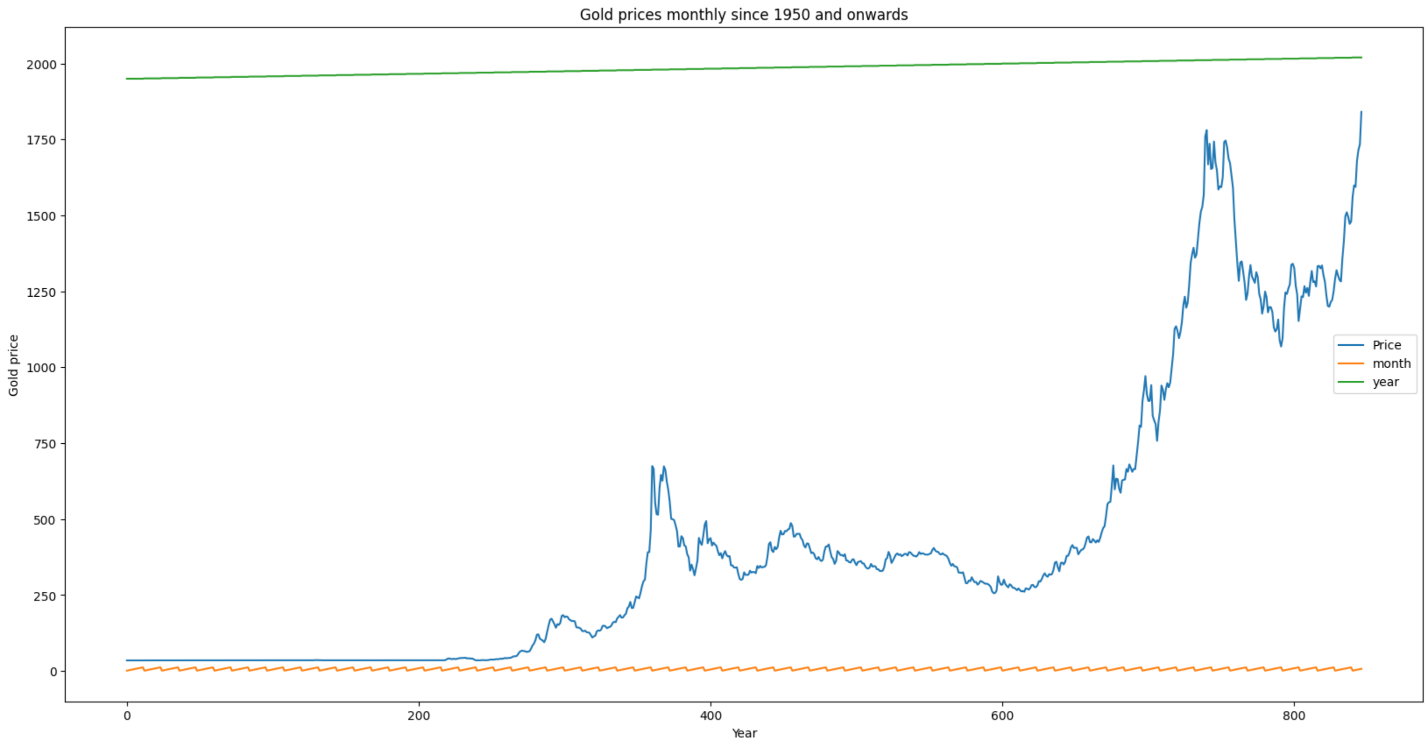
```
df['month']=pd.to_datetime(df['Date']).dt.month
df['year']=pd.to_datetime(df['Date']).dt.year
df.head()
```

Out[9]:

	Date	Price	month	year
0	1950-01	34.73	1	1950
1	1950-02	34.73	2	1950
2	1950-03	34.73	3	1950
3	1950-04	34.73	4	1950
4	1950-05	34.73	5	1950

In [13]:

```
df.plot(figsize=(20,10))
plt.title("Gold prices monthly since 1950 and onwards") # Call title on plt
plt.xlabel("Year") # Call xlabel on plt
plt.ylabel("Gold price") # Call ylabel on plt
plt.show()
```



In [14]:

```
round(df.describe(),2)
```

Out[14]:

	Price	month	year
count	847.00	847.00	847.00
mean	416.56	6.48	1984.79
std	453.67	3.45	20.39
min	34.49	1.00	1950.00
25%	35.19	3.00	1967.00
50%	319.62	6.00	1985.00

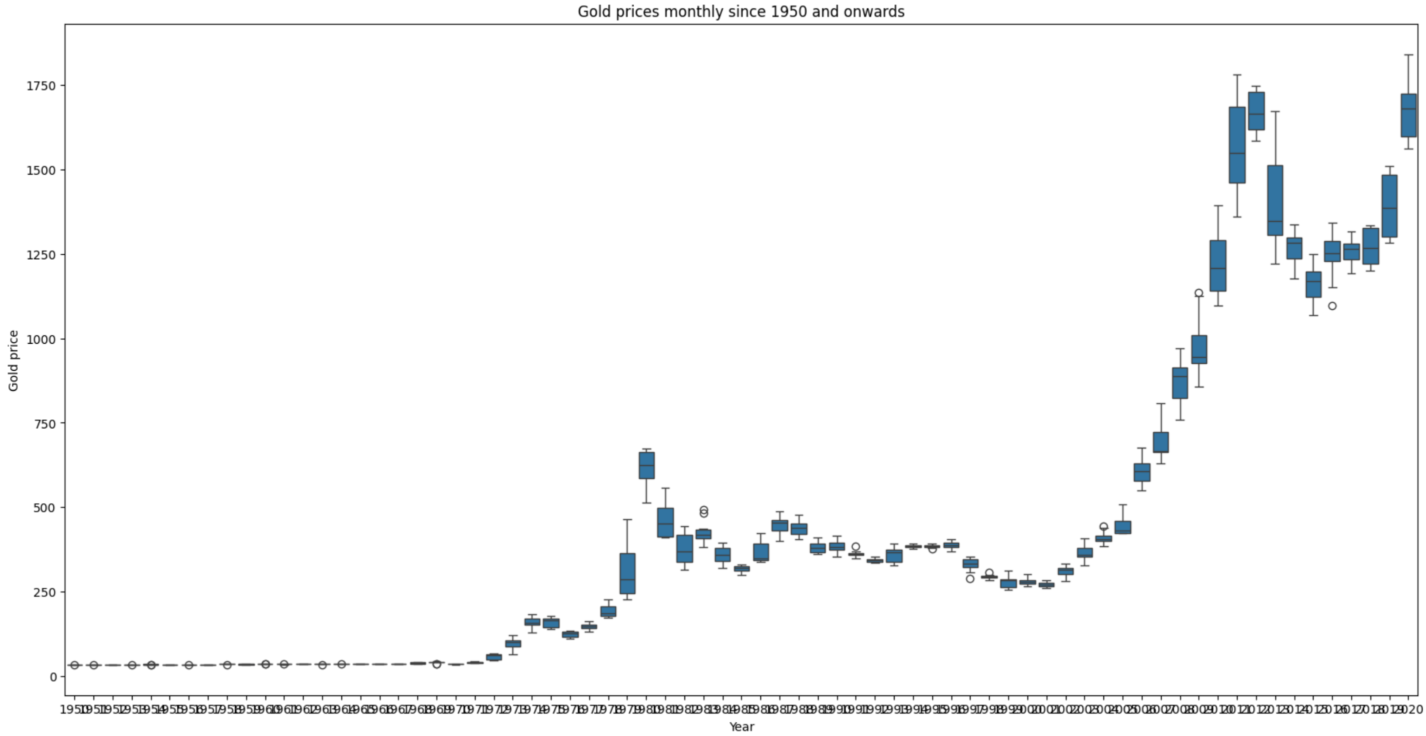
	Price	month	year
75%	447.93	9.00	2002.00
max	1840.81	12.00	2020.00

In [18]:

```
df['year']=pd.to_datetime(df['Date']).dt.year
```

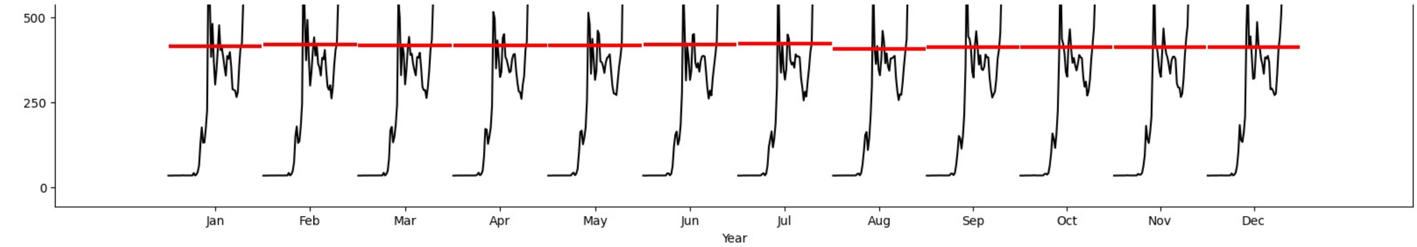
In [20]:

```
fig, ax = plt.subplots(figsize=(20,10)) # Changed ax to fig, ax
sns.boxplot(x=df['year'], y=df['Price'],ax=ax) # Changed df.index.year to df['year']
plt.title("Gold prices monthly since 1950 and onwards") # Call title on plt
plt.xlabel("Year") # Call xlabel on plt
plt.ylabel("Gold price") # Call ylabel on plt
plt.show()
```



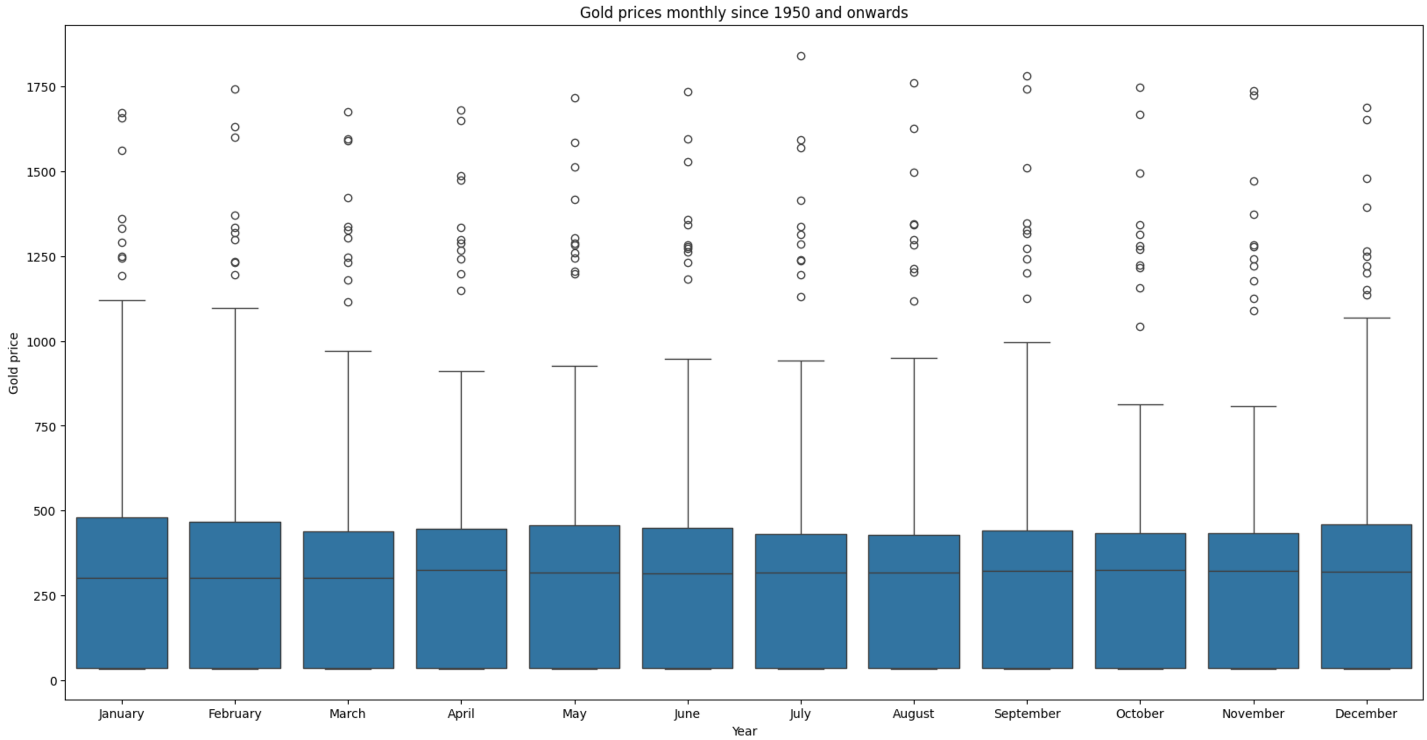
In [22]:

```
from statsmodels.graphics.tsaplots import month_plot
fig, ax = plt.subplots(figsize=(20,10))
# Set the 'Date' column as the index and convert it to DatetimeIndex
df.set_index(pd.to_datetime(df['Date']), inplace=True)
month_plot(df['Price'],ylabel=['Gold Price'], ax=ax) # Pass the 'Price' column for plott
ing
plt.title("Gold prices monthly since 1950 and onwards")
plt.xlabel("Year")
plt.ylabel("Gold price")
plt.show()
```



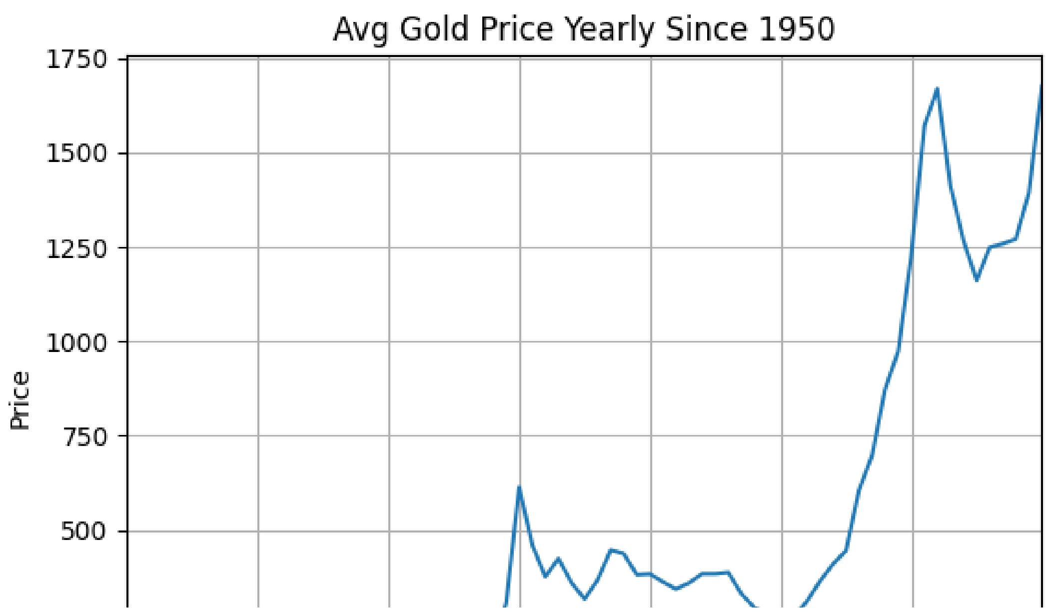
In [24]:

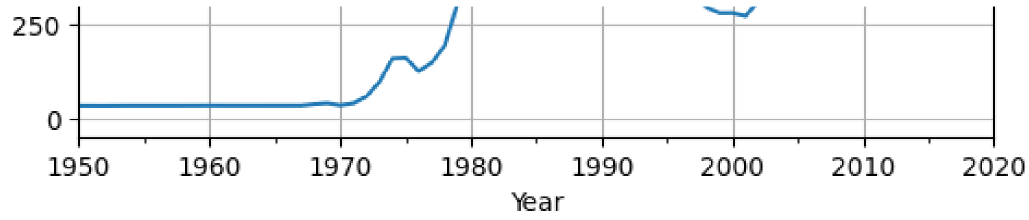
```
fig, ax = plt.subplots(figsize=(20,10))
sns.boxplot(x=df.index.month_name(), y=df['Price'],ax=ax)
plt.title("Gold prices monthly since 1950 and onwards")
plt.xlabel("Year")
plt.ylabel("Gold price")
plt.show()
```



In [26]:

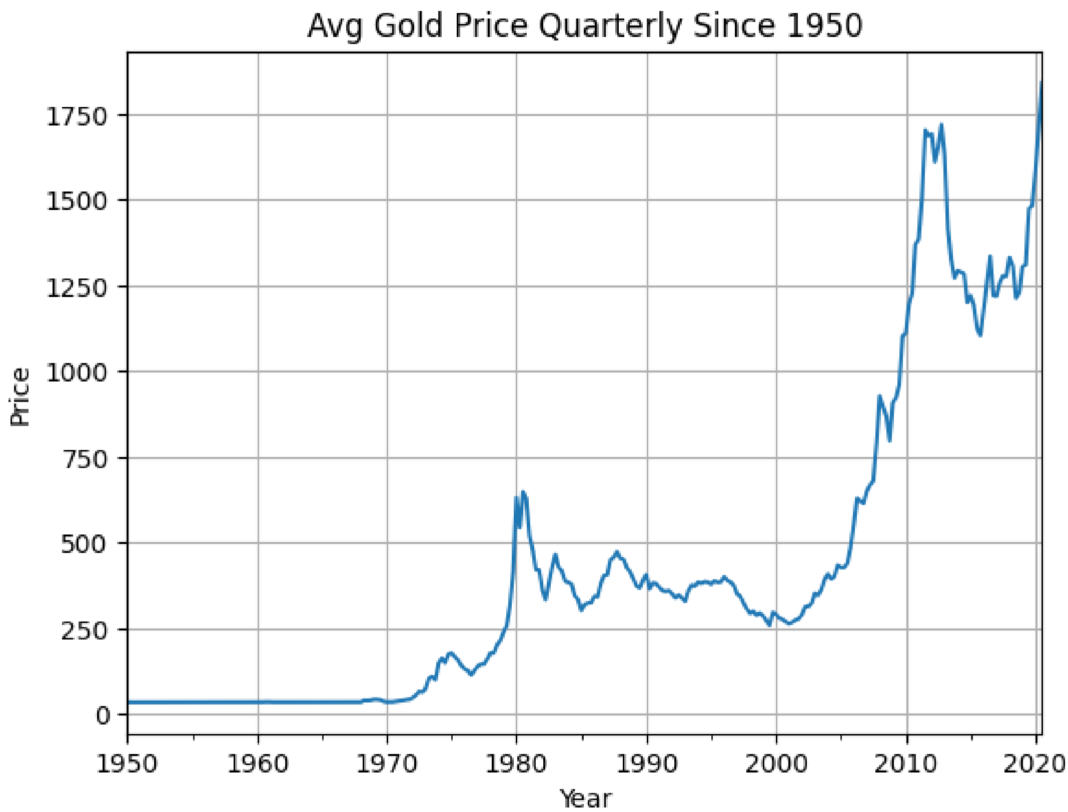
```
df_yearly_sum = df['Price'].resample('A').mean() # Select 'Price' column for resampling
df_yearly_sum.plot();
plt.title("Avg Gold Price Yearly Since 1950")
plt.xlabel('Year')
plt.ylabel('Price')
plt.grid();
```





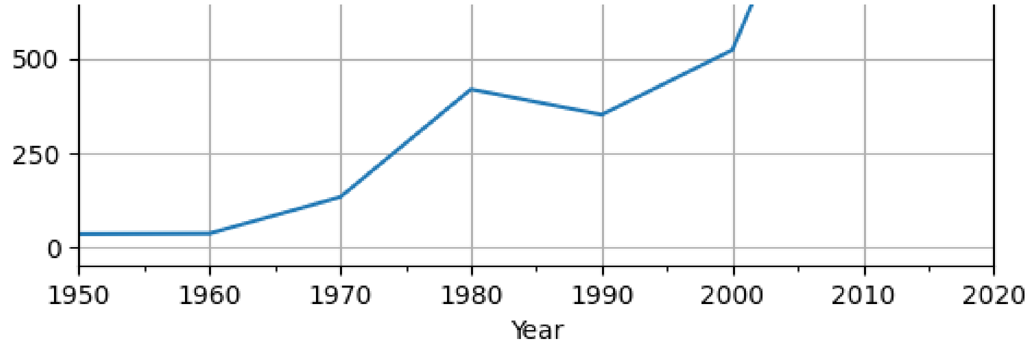
In [27]:

```
df_quarterly_sum = df['Price'].resample('Q').mean()  # Select 'Price' column for resampli
ng
df_quarterly_sum.plot();
plt.title("Avg Gold Price Quarterly Since 1950")
plt.xlabel('Year')
plt.ylabel('Price')
plt.grid();
```



In [28]:

```
df_decade_sum = df['Price'].resample('10AS').mean()  # Select 'Price' column for resampli
ng
df_decade_sum.plot();
plt.title("Avg Gold Price Decade Since 1950")
plt.xlabel('Year')
plt.ylabel('Price')
plt.grid();
```



In [32]:

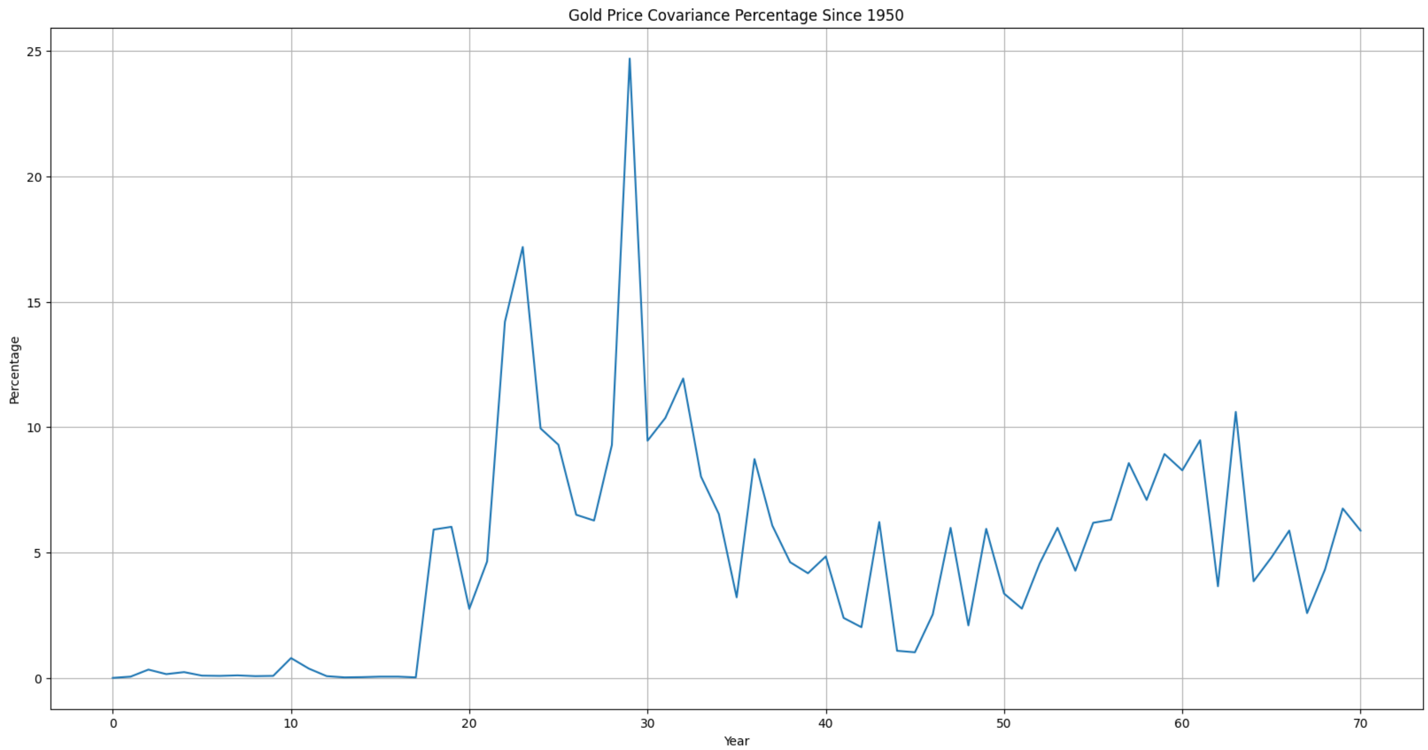
```
df_1 = df.groupby(df['year'])['Price'].mean().reset_index().rename(columns={'Price':'Mean'}) # Group by the 'year' column and select 'Price' for mean calculation
df_2 = df.groupby(df['year'])['Price'].std().reset_index().rename(columns={'Price':'Std'}) # Group by the 'year' column and select 'Price' for std calculation
df_1 = df_1.merge(df_2, on='year', how='left') # Merge based on 'year' column
df_1['Cov_pct']=((df_1['Std']/df_1['Mean'])*100).round(2) # Calculate Cov_pct
df_1.drop(columns=['Std'],inplace=True) # Drop the 'Std' column
df_1.head()
```

Out[32]:

	year	Mean	Cov_pct
0	1950	34.729167	0.01
1	1951	34.717500	0.06
2	1952	34.628333	0.34
3	1953	34.879167	0.16
4	1954	35.020000	0.24

In [33]:

```
fig, ax = plt.subplots(figsize=(20,10))
df_1['Cov_pct'].plot();
plt.title("Gold Price Covariance Percentage Since 1950")
plt.xlabel('Year')
plt.ylabel('Percentage')
plt.grid()
```



In [35]:

In [35]:

```
train = df[df.index.year <= 2015] # Compare year extracted from index
test = df[df.index.year > 2015]   # Compare year extracted from index
```

In [36]:

```
print(train.shape)
print(test.shape)
```

```
(792, 4)
(55, 4)
```

In [37]:

```
train["Price"].plot(figsize=(13,5),fontsize=15)
test["Price"].plot(figsize=(13,5),fontsize=15)
plt.grid()
plt.legend(["Train","Test"])
plt.show()
```

In [38]:

```
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[38]:

```
(792, 55)
```

In [39]:

```
LR_train = train.copy()
LR_test = test.copy()
```

In [40]:

```
LR_train["time"] = train_time
LR_test["time"] = test_time
```

In [42]:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()  # Use Linear Regression instead of Logistic Regression
lr.fit(np.array(LR_train["time"]).reshape(-1,1), LR_train["Price"])
```

Out[42]:

▾

LinearRegression

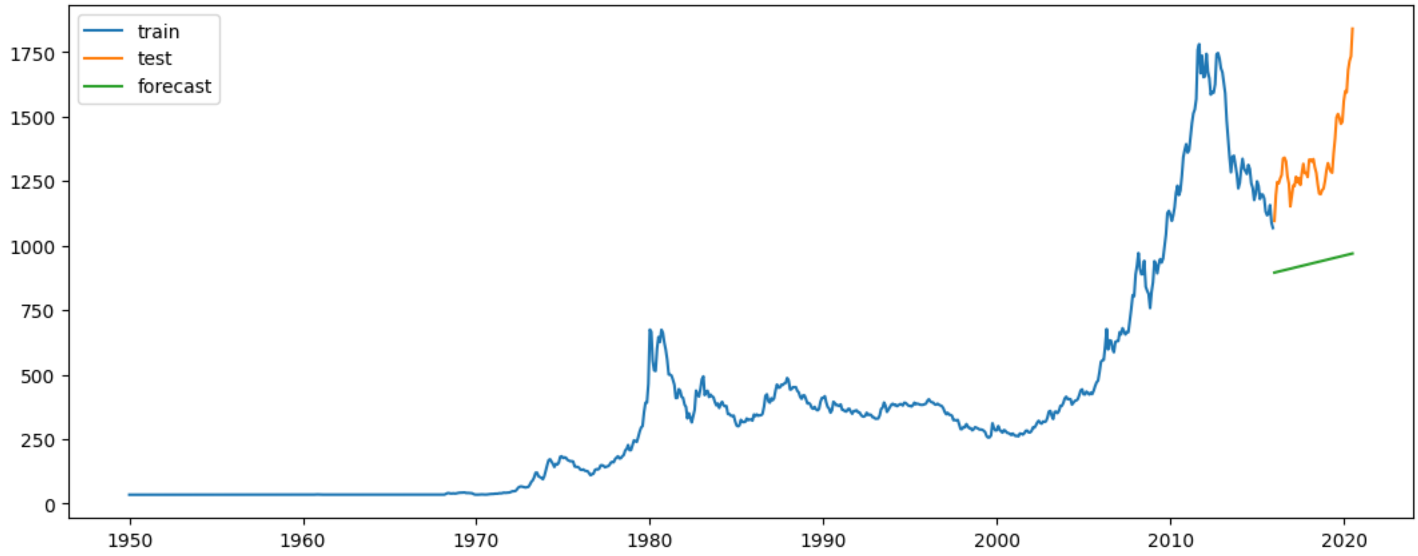
i ?

```
LinearRegression()
```

In [43]:

```
test_predictions_model1 = lr.predict(np.array(LR_test["time"]).reshape(-1,1))
LR_test['forecast'] = test_predictions_model1

plt.figure(figsize=(13,5))
plt.plot(train['Price'], label = 'train')
plt.plot(test['Price'], label = 'test')
plt.plot(LR_test['forecast'], label = 'forecast')
plt.legend()
plt.show()
```



In [44]:

```
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean((np.abs((actual - pred) / actual)) * 100,2)
```

In [46]:

```
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean((np.abs((actual - pred) / actual)) * 100) # Remove axis argument
```

In [48]:

```
mape_model1_test = mape(test['Price'], test_predictions_model1)

results = pd.DataFrame({'Test Mape (%)': [mape_model1_test]}, index=['Linear Regression'
])
results
```

Out[48]:

	Test Mape (%)
Linear Regression	29.759658

In [49]:

```
final_model = ExponentialSmoothing(train['Price'], trend='add', seasonal='add', seasonal
_periods=12).fit(smoothing_level = 0.4, smoothing_trend = 0.3, smoothing_seasonal = 0.6)
final_model.summary()
```

Out[49]:

ExponentialSmoothing Model Results

ExponentialSmoothing Model Results

Dep. Variable:	Price	**No. Observations:**	792
Model:	ExponentialSmoothing	**SSE**	1722120.573
Optimized:	True	**AIC**	6118.128
Trend:	Additive	**BIC**	6192.921
Seasonal:	Additive	**AICC**	6119.013
Seasonal Periods:	12	**Date:**	Thu, 20 Feb 2025
Box-Cox:	False	**Time:**	14:08:44
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.4000000	alpha	False
smoothing_trend	0.3000000	beta	False
smoothing_seasonal	0.6000000	gamma	False
initial_level	24.522114	l.0	True
initial_trend	3.4367234	b.0	True
initial_seasons.0	6.8136610	s.0	True
initial_seasons.1	0.6850910	s.1	True
initial_seasons.2	-5.4237864	s.2	True
initial_seasons.3	-10.209069	s.3	True
initial_seasons.4	-12.606444	s.4	True
initial_seasons.5	-12.198092	s.5	True
initial_seasons.6	-9.0349084	s.6	True
initial_seasons.7	-3.7471878	s.7	True
initial_seasons.8	2.3913919	s.8	True
initial_seasons.9	8.1437567	s.9	True
initial_seasons.10	12.245447	s.10	True
initial_seasons.11	13.800143	s.11	True

In [50]:

```
Mape_final_model = mape(test['Price'], final_model.forecast(len(test)))
results['Final Model Mape (%)'] = Mape_final_model
results
```

Out[50]:

	Test Mape (%)	Final Model Mape (%)
Linear Regression	29.759658	118.835014

In [51]:

```
print("MAPE:", Mape_final_model)
```

```
MAPE: 118.83501404778552
```

In [52]:

```
predictions = final_model.forecast(len(test))
```

In [54]:

```
pred_df = pd.DataFrame(predictions, columns=['Predicted'])
```

```
pred_df.head(10)
```

Out[54]:

	Predicted
2016-01-01	1052.357459
2016-02-01	958.368204
2016-03-01	850.811130
2016-04-01	778.097484
2016-05-01	688.579889
2016-06-01	602.767389
2016-07-01	531.554870
2016-08-01	522.347181
2016-09-01	539.918017
2016-10-01	577.434659

In [57]:

```
axis = df.plot(label='Actual',figsize=(13,5))
pred_df['Predicted'].plot(ax=axis, label='Predicted') # Changed 'predictions' to 'Predict
ed'
# Removed fill_between as 'lower_ci' and 'upper_ci' are not in pred_df
# axis.fill_between(pred_df.index, pred_df['lower_ci'], pred_df['upper_ci'], color='k', a
lpha=.15)
axis.set_xlabel('Date')
axis.set_ylabel('Gold Price')
plt.legend(loc = 'best')
plt.show()
```