Time Series Prediction on Gold Prices

🏫 *Author: Nhi Yen*

💡 *I write about Machine Learning on Medium || Github || Kaggle || Linkedin. If you found this article interesting, your support by giving me ☆ will help me spread the knowledge to others.*

Introduction

This Kaggle project provides code for forecasting the price of gold using various time series forecasting methods. The dataset used for the analysis is the daily price of gold in USD from 1950-01 to 2020-07, with a total of 847 data points. The Jupyter notebook included in the repository contains code for building and evaluating three different time series forecasting models, namely Linear Regression Model, Naive Model, and Exponential Smoothing Model. The Exponential Smoothing Model performed the best with a MAPE score of 17.235%. The predicted gold prices for the period 2020-08 to 2025-02 using the Exponential Smoothing Model are also provided in a CSV file named gold_price_predictions.csv. The dataset, code, and results can be accessed through the Kaggle project and Github repository provided in the references.

READ MORE

```
# !pip install pandas-profiling

import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from statsmodels.tsa.api import ExponentialSmoothing,
SimpleExpSmoothing, Holt
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings("ignore")
```

```
# Read the CSV file and display the first few rows
df =
pd.read_csv('/kaggle/input/monthly-gold-price/gold_monthly_csv.csv')
print(f"Gold prices data has {df.shape[0]} rows and {df.shape[1]}
columns.")
print(f"The date range of gold prices available is from
{df.loc[:,'Date'][0]} to {df.loc[:,'Date'][len(df) - 1]}")
df.head()
```

```
Gold prices data has 847 rows and 2 columns.
The date range of gold prices available is from 1950-01 to 2020-07
```

```
      Date  Price
0  1950-01  34.73
```

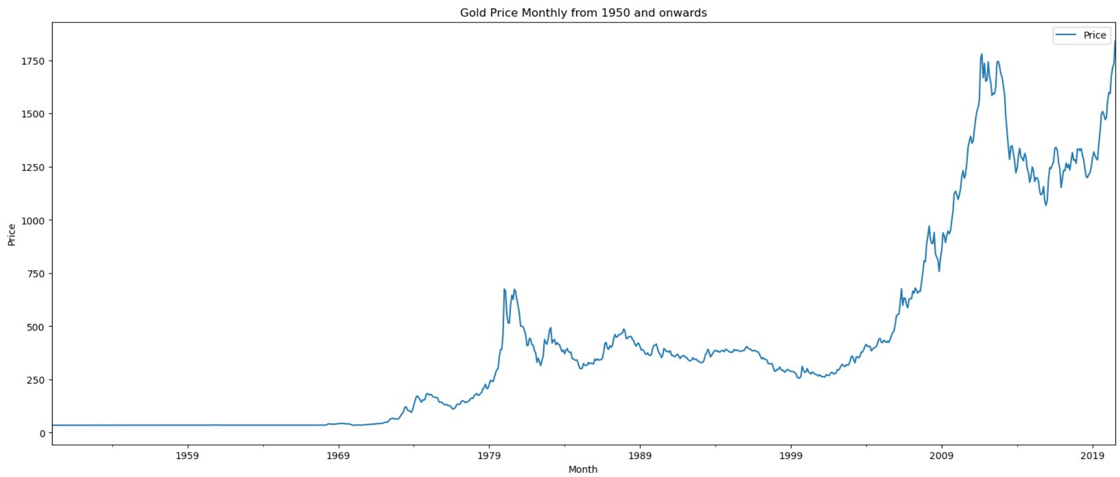
```
1  1950-02  34.73
2  1950-03  34.73
3  1950-04  34.73
4  1950-05  34.73
```

```
# Create a new dataframe with monthly dates as the index
date_range = pd.date_range(start='1/1/1950', end='8/1/2020', freq='M')
df['month'] = date_range
df.drop('Date', axis=1, inplace=True)
df = df.set_index('month')
df.head()
```

```
            Price
month
1950-01-31  34.73
1950-02-28  34.73
1950-03-31  34.73
1950-04-30  34.73
1950-05-31  34.73
```

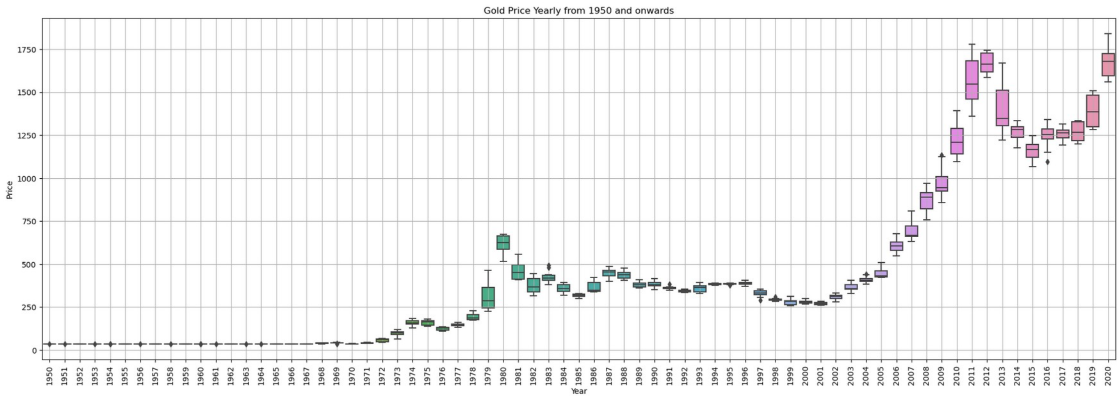
```
# Plot the gold prices over time
plt.figure(figsize=(20,8))
df.plot(figsize=(20,8))
plt.title('Gold Price Monthly from 1950 and onwards')
plt.xlabel('Month')
plt.ylabel('Price')
plt.show()
```

```
<Figure size 2000x800 with 0 Axes>
```




```
# Create a boxplot of the gold prices by year
plt.figure(figsize=(25,8))
sns.boxplot(x=df.index.year, y=df.values[:,0])
plt.title('Gold Price Yearly from 1950 and onwards')
plt.xlabel('Year')
plt.ylabel('Price')
```

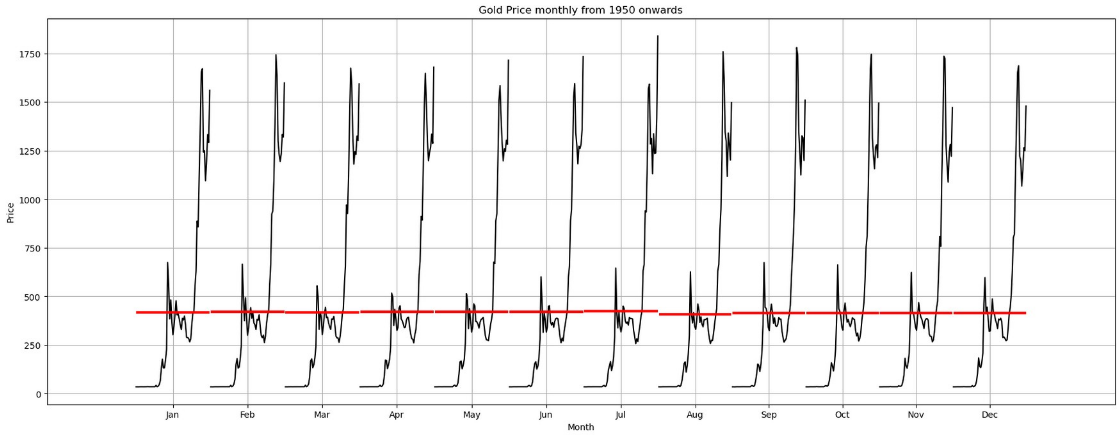
```
plt.xticks(rotation=90)
plt.grid()
plt.show()
```



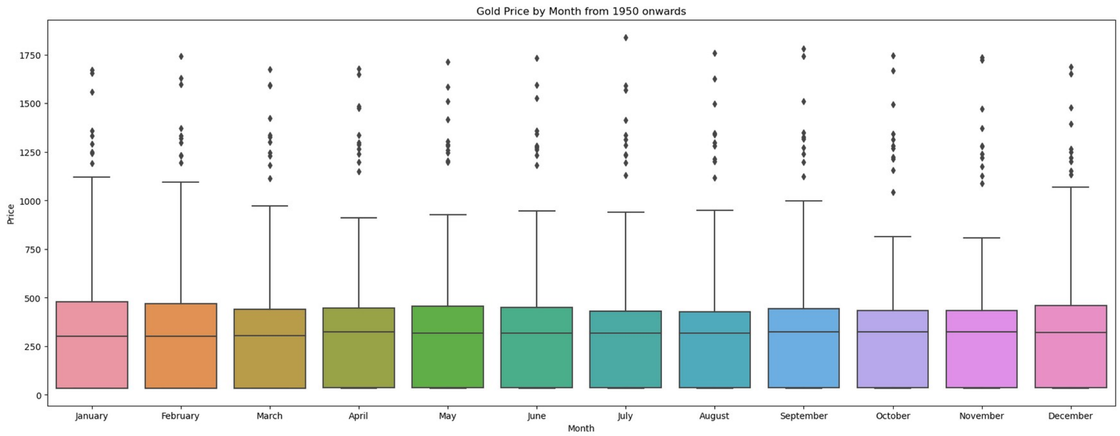

```
# Show summary statistics of the gold prices
print("Summary statistics of gold prices:\n", df.describe())
```

```
Summary statistics of gold prices:
              Price
count   847.000000
mean    416.556906
std     453.665313
min      34.490000
25%      35.190000
50%     319.622000
75%     447.029000
max    1840.807000
```

```
from statsmodels.graphics.tsaplots import month_plot

# Plot the gold prices by month
fig, ax = plt.subplots(figsize=(22,8))
month_plot(df, ylabel='Gold Price', ax=ax)
plt.title('Gold Price monthly from 1950 onwards')
plt.xlabel('Month')
plt.ylabel('Price')
plt.grid()
plt.show()
```

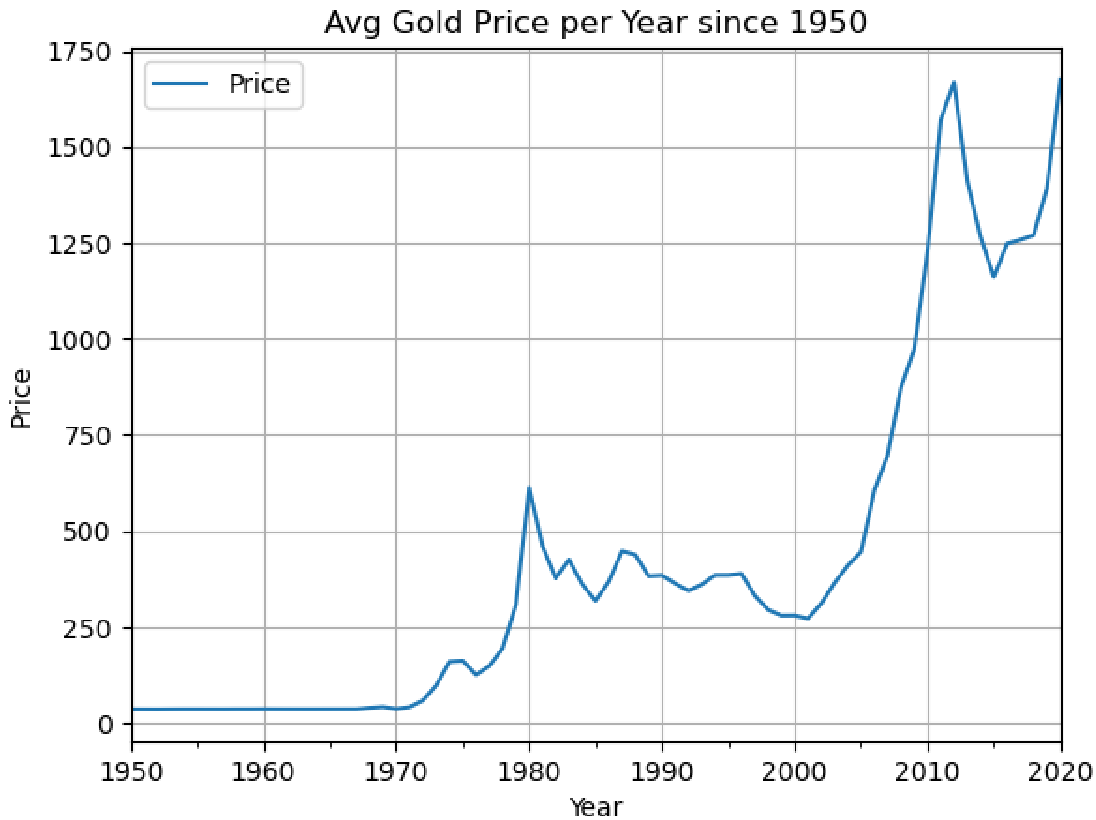



```python
# Create a boxplot of the gold prices by month
fig, ax = plt.subplots(figsize=(22,8))
sns.boxplot(x = df.index.month_name(), y=df.values[:,0], ax=ax)
plt.title('Gold Price by Month from 1950 onwards')
plt.xlabel('Month')
plt.ylabel('Price')
plt.show()
```

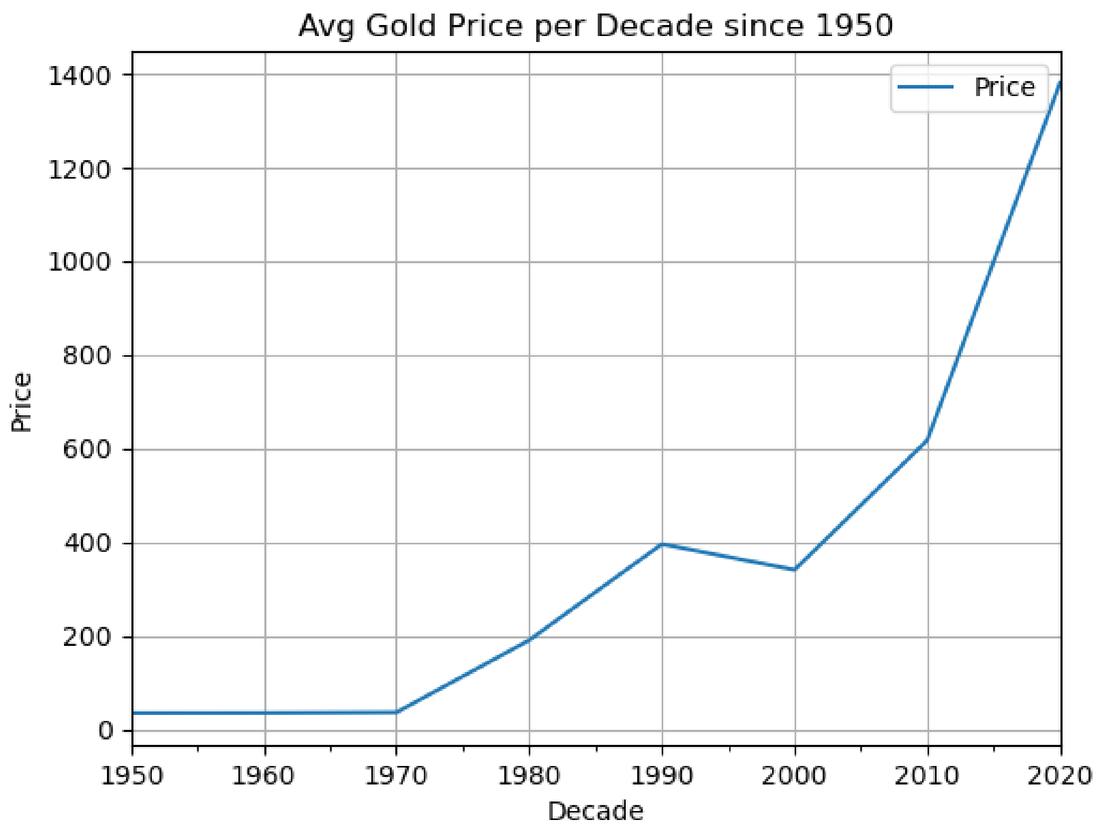



```python
# Create yearly, quarterly, and decade summaries of the data
df_yearly_sum = df.resample('A').mean()
df_yearly_sum.plot()
plt.title('Avg Gold Price per Year since 1950')
plt.xlabel('Year')
plt.ylabel('Price')
plt.grid()

df_quarterly_sum = df.resample('Q').mean()
df_quarterly_sum.plot()
plt.title('Avg Gold Price per Quarter since 1950')
plt.xlabel('Quarter')
plt.ylabel('Price')
plt.grid()
```

```
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()
```



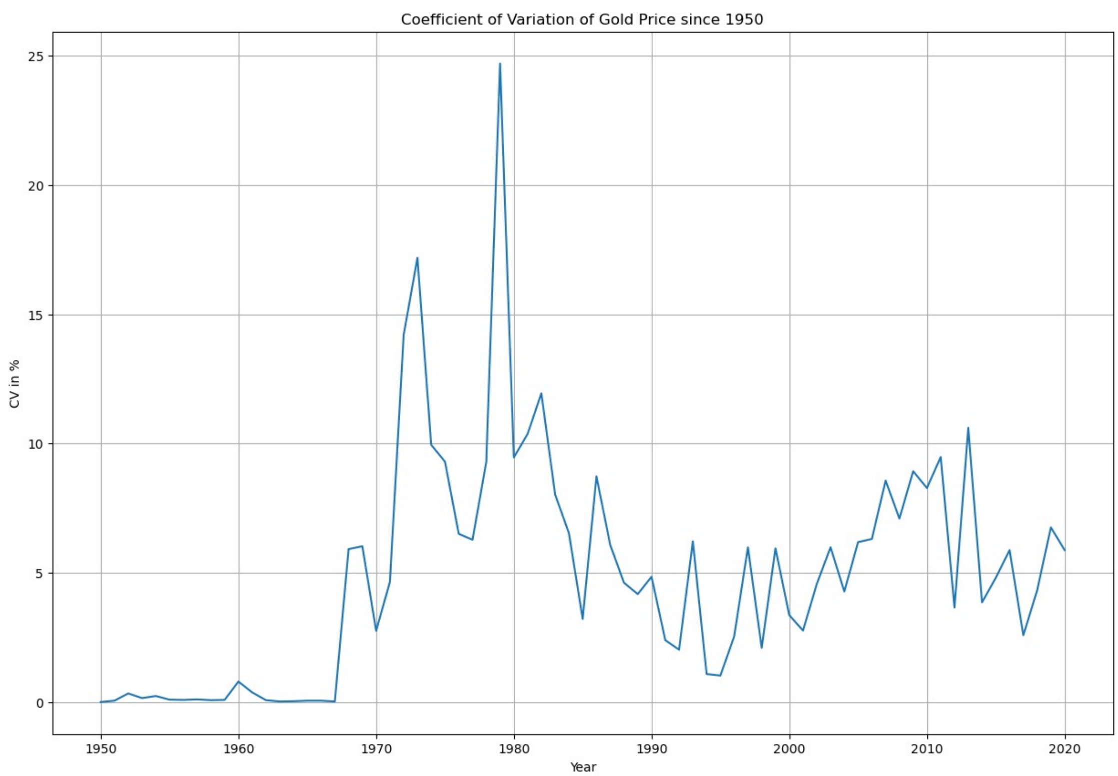
Avg Gold Price per Quarter since 1950
Price
1750
1500
1250
1000
750
500
250
0
Price
1959
1969
1979
1989
1999
2009
2019
Quarter




```
# Show summary statistics of the gold prices
print("Summary statistics of gold prices:\n", df.describe())

Summary statistics of gold prices:
              Price
count   847.000000
mean    416.556906
std     453.665313
min      34.490000
25%      35.190000
50%     319.622000
75%     447.029000
max    1840.807000

# Calculate the mean, standard deviation, and coefficient of variation
(CV) for the data by year
df_1 =
df.groupby(df.index.year).mean().rename(columns={'Price':'Mean'})
df_1 =
df_1.merge(df.groupby(df.index.year).std().rename(columns={'Price':'St
d'}), left_index=True, right_index=True)
df_1['Cov_pct'] = ((df_1['Std'] / df_1['Mean']) * 100).round(2)

# Plot the CV over time
fig, ax = plt.subplots(figsize=(15, 10))
```

```
df_1['Cov_pct'].plot()
plt.title('Coefficient of Variation of Gold Price since 1950')
plt.xlabel('Year')
plt.ylabel('CV in %')
plt.grid()

# Display the first few rows of the data
print('The first few rows of the coefficient of variation data:')
print(df_1.head())
```

```
The first few rows of the coefficient of variation data:
            Mean       Std  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
```

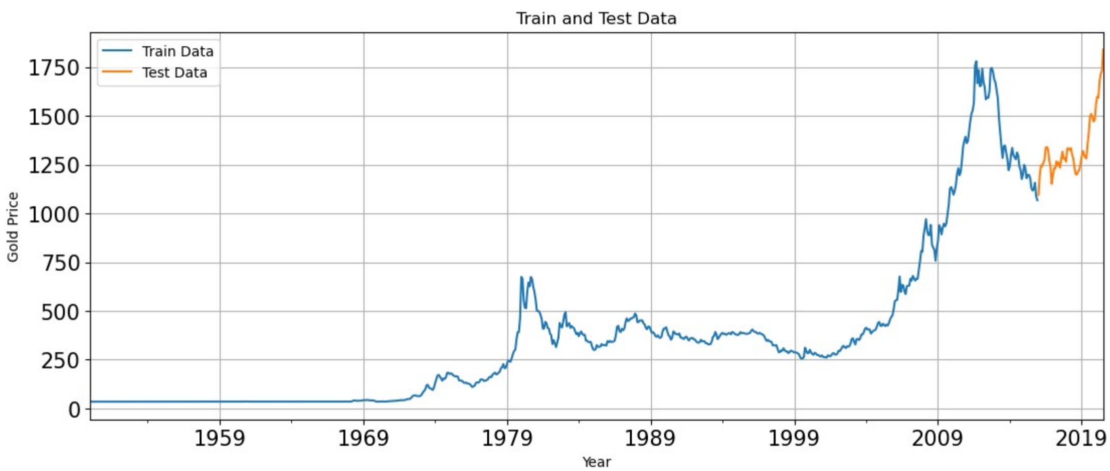



```
# Split the dataset into training and testing sets based on a cutoff
year
train = df[df.index.year <= 2015]
test = df[df.index.year > 2015]


# Check the shape of the training and testing sets
print("Training set shape:", train.shape)
```

```
print("Testing set shape:", test.shape)

# Checking for bias in the split
train_pct = len(train) / len(df) * 100
test_pct = len(test) / len(df) * 100
print("Percentage of data in training set:", round(train_pct, 2), '%')
print("Percentage of data in testing set:", round(test_pct, 2), '%')

# It is not necessarily biased to split the data this way.
# However, it may depend on the specific problem and the goals of the
analysis.
# It is always a good practice to carefully consider the split and to
test different split ratios to evaluate their impact on the model
performance.
```

```
Training set shape: (792, 1)
Testing set shape: (55, 1)
Percentage of data in training set: 93.51 %
Percentage of data in testing set: 6.49 %
```

```
# Plot train and test data
train['Price'].plot(figsize=(13,5), fontsize=15)
test['Price'].plot(figsize=(13,5), fontsize=15)
plt.grid()
plt.legend(['Train Data', 'Test Data'])
plt.title('Train and Test Data')
plt.xlabel('Year')
plt.ylabel('Gold Price')
plt.show()
```

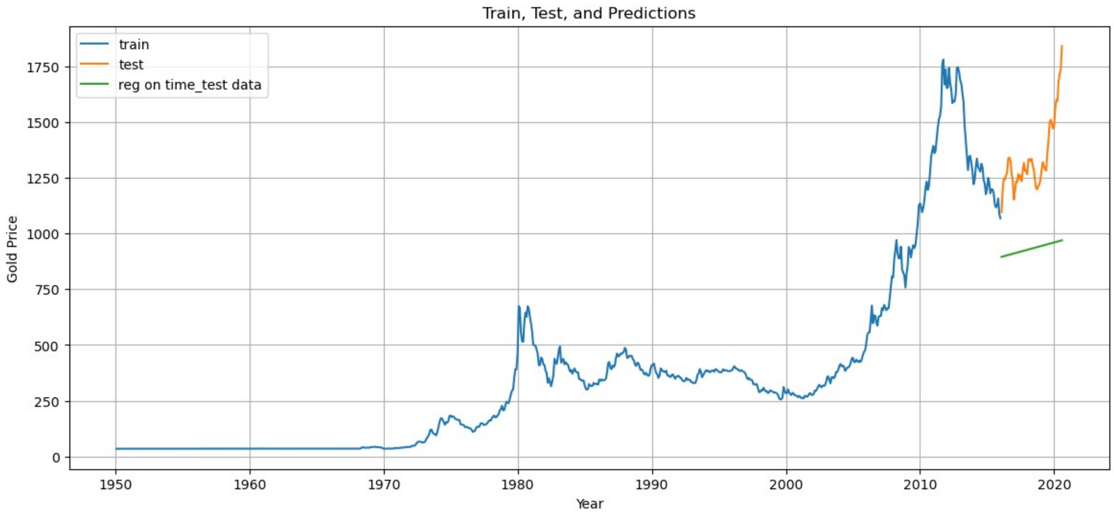



```
# Create time series for train and test data
train_time = [i+1 for i in range(len(train))]
test_time = [i+len(train)+1 for i in range(len(test))]
print(f"Train time series length: {len(train_time)}")
print(f"Test time series length: {len(test_time)}")
```

```
# Add time series as a feature to train and test data
LR_train = train.copy()
LR_test = test.copy()
LR_train['time'] = train_time
LR_test['time'] = test_time
```

```
Train time series length: 792
Test time series length: 55
```

```
# Train a linear regression model on the train data using time as the independent variable
lr = LinearRegression()
lr.fit(LR_train[['time']], LR_train['Price'].values)

# Use the trained model to make predictions on the test data
test_predictions_model1 = lr.predict(LR_test[['time']])
LR_test['forecast'] = test_predictions_model1

# Plot the train data, test data, and predictions made by the linear regression model
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()
plt.title('Train, Test, and Predictions')
plt.xlabel('Year')
plt.ylabel('Gold Price')
plt.show()
```




```
def mapе(y_true, y_pred):
    """Calculate mean absolute percentage error (MAPE)"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
# Calculate MAPE for the linear regression model
mape_model1_test = mape(test['Price'].values, test_predictions_model1)
print(f"MAPE for Regression on Time model on test data:
{mape_model1_test:.3f}%")
```

```
MAPE for Regression on Time model on test data: 29.760%
```

```
# Create results dataframe to store MAPE values for different models
results = pd.DataFrame({'Test MAPE (%)': [mape_model1_test]},
index=['Regression on Time'])
results
```

```
                    Test MAPE (%)
Regression on Time      29.759658
```

```
# Create naive model
naive_train = train.copy()
naive_test = test.copy()
naive_test['naive'] = np.asarray(train['Price'])
[len(np.asarray(train['Price'])) - 1]
naive_test['naive'].head()
```

```
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
```

```
# Plot the train, test, and naive forecast on test data
plt.figure(figsize=(12,8))
plt.plot(naive_train['Price'], label='Train Data')
plt.plot(test['Price'], label='Test Data')
plt.plot(naive_test['naive'], label='naive Forecast on Test Data')
plt.legend(loc='best')
plt.grid()
```


```
# Calculate MAPE for naive forecast model
mape_model2_test = mape(test['Price'].values,
naive_test['naive'].values)
print('MAPE for naive Forecast model is %3.3f%%' % mape_model2_test)

# Update the dataframe with the MAPE result for the naive forecast
model
resultsDf_2 = pd.DataFrame({'Test MAPE (%)': [mape_model2_test]},
index=['naiveForecast'])
results = pd.concat([results,resultsDf_2])

# Display the dataframe with the MAPE results for both models
results
```

```
MAPE for naive Forecast model is 19.385%
```

	Test MAPE (%)
Regression on Time	29.759658
naiveForecast	19.384586

```
# Perform exponential smoothing with additive trend and additive
seasonality
final_model = ExponentialSmoothing(df, trend='additive',
seasonal='additive').fit(
    smoothing_level=0.4, smoothing_trend=0.3, smoothing_seasonal=0.6)

# Calculate mean absolute percentage error (MAPE) of the fitted values
```

```
mape_final_model = mape(df['Price'].values, final_model.fittedvalues)
print('MAPE of final model: %.3f%%' % mape_final_model)
```

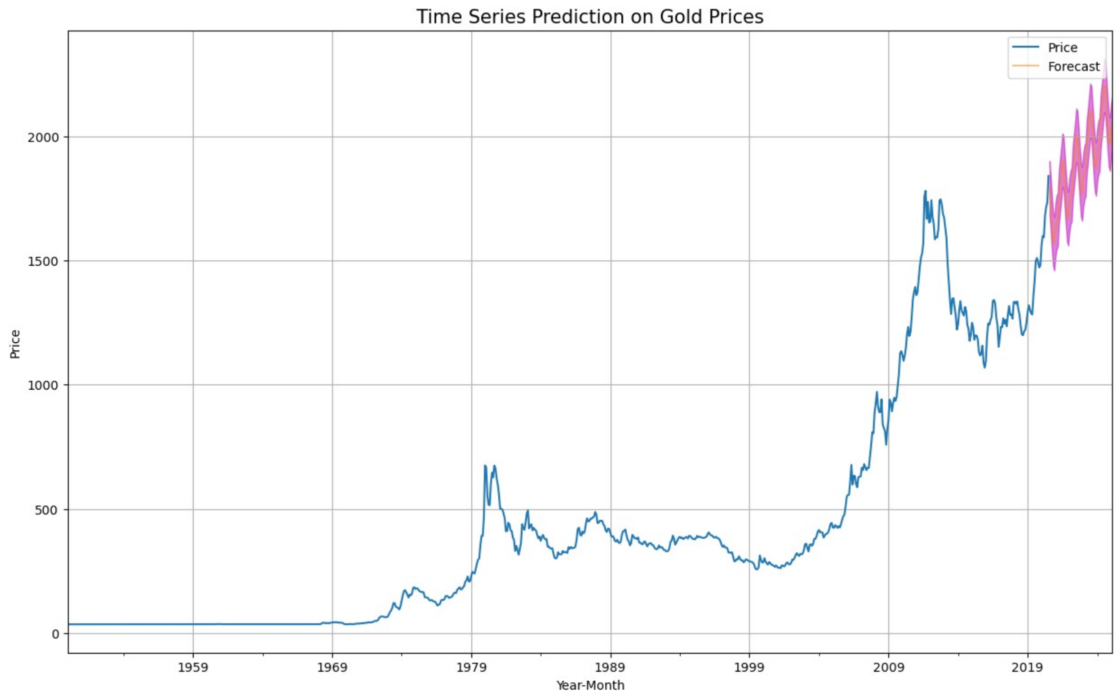
```
MAPE of final model: 17.235%
```

```
# Generate forecast using the final model
prediction = final_model.forecast(steps=len(test))

# Create a dataframe to store the prediction and confidence intervals
pred_df = pd.DataFrame({
    'lower_CI': prediction - 1.96 * np.std(final_model.resid, ddof=1),
    'prediction': prediction,
    'upper_CI': prediction + 1.96 * np.std(final_model.resid, ddof=1)
})
# Export the predictions into a csv file.
pred_df.to_csv('gold_price_predictions.csv', index=True,
index_label='Month')

print('Prediction and confidence intervals:')
pred_df
```

```
Prediction and confidence intervals:

               lower_CI   prediction     upper_CI
2020-08-31  1684.720065  1792.871037  1901.022009
2020-09-30  1615.306077  1723.457050  1831.608022
2020-10-31  1538.567922  1646.718895  1754.869867
2020-11-30  1476.758600  1584.909572  1693.060545
2020-12-31  1459.327290  1567.478262  1675.629235
2021-01-31  1514.417601  1622.568574  1730.719546
2021-02-28  1545.352396  1653.503369  1761.654341
2021-03-31  1556.764378  1664.915350  1773.066323
2021-04-30  1648.309829  1756.460802  1864.611774
2021-05-31  1694.225915  1802.376887  1910.527859
2021-06-30  1743.402088  1851.553061  1959.704033
2021-07-31  1796.108352  1904.259324  2012.410297
2021-08-31  1785.037437  1893.188409  2001.339381
2021-09-30  1715.623449  1823.774422  1931.925394
2021-10-31  1638.885294  1747.036267  1855.187239
2021-11-30  1577.075972  1685.226944  1793.377917
2021-12-31  1559.644662  1667.795634  1775.946607
2022-01-31  1614.734973  1722.885946  1831.036918
2022-02-28  1645.669768  1753.820741  1861.971713
2022-03-31  1657.081750  1765.232722  1873.383695
2022-04-30  1748.627201  1856.778174  1964.929146
2022-05-31  1794.543287  1902.694259  2010.845231
2022-06-30  1843.719460  1951.870433  2060.021405
2022-07-31  1896.425724  2004.576696  2112.727669
2022-08-31  1885.354809  1993.505781  2101.656753
2022-09-30  1815.940821  1924.091794  2032.242766
2022-10-31  1739.202666  1847.353639  1955.504611
```

```
2022-11-30  1677.393344  1785.544316  1893.695289
2022-12-31  1659.962034  1768.113006  1876.263979
2023-01-31  1715.052345  1823.203318  1931.354290
2023-02-28  1745.987140  1854.138113  1962.289085
2023-03-31  1757.399122  1865.550094  1973.701067
2023-04-30  1848.944573  1957.095545  2065.246518
2023-05-31  1894.860659  2003.011631  2111.162603
2023-06-30  1944.036832  2052.187805  2160.338777
2023-07-31  1996.743096  2104.894068  2213.045041
2023-08-31  1985.672180  2093.823153  2201.974125
2023-09-30  1916.258193  2024.409165  2132.560138
2023-10-31  1839.520038  1947.671011  2055.821983
2023-11-30  1777.710716  1885.861688  1994.012661
2023-12-31  1760.279406  1868.430378  1976.581350
2024-01-31  1815.369717  1923.520690  2031.671662
2024-02-29  1846.304512  1954.455485  2062.606457
2024-03-31  1857.716494  1965.867466  2074.018439
2024-04-30  1949.261945  2057.412917  2165.563890
2024-05-31  1995.178030  2103.329003  2211.479975
2024-06-30  2044.354204  2152.505177  2260.656149
2024-07-31  2097.060468  2205.211440  2313.362413
2024-08-31  2085.989552  2194.140525  2302.291497
2024-09-30  2016.575565  2124.726537  2232.877510
2024-10-31  1939.837410  2047.988383  2156.139355
2024-11-30  1878.028088  1986.179060  2094.330033
2024-12-31  1860.596778  1968.747750  2076.898722
2025-01-31  1915.687089  2023.838062  2131.989034
2025-02-28  1946.621884  2054.772857  2162.923829
```

```python
# Plot actual and predicted values with confidence intervals
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='m', alpha=0.5)
axis.set_xlabel('Year-Month')
axis.set_ylabel('Price')
plt.title('Time Series Prediction on Gold Prices',fontsize=15)
plt.legend(loc='best')
plt.grid()
plt.show()

# Save plot as png
plt.savefig('gold_price_predictions.png')
```



<Figure size 640x480 with 0 Axes>

References

- Kaggle Dataset: Monthly Gold Price
- Github Repo - HERE
- Kaggle Project - HERE
- Detail Explanation about the code on MEDIUM