University of Oxford

Financial Data Analysis

MSc in Mathematical Finance
January 14, 2019

GNP Time Series: Forecasting using Exponential Smoothing, Autoregressive Model, Climatology and Persistence

1.1 GNP Returns

(a) Computed log returns of the GNP data and divided them into a training and test set. Please see file financial_data_analysis.py for the relevant code.

(b) (i) Plot of the log-returns of the training data:

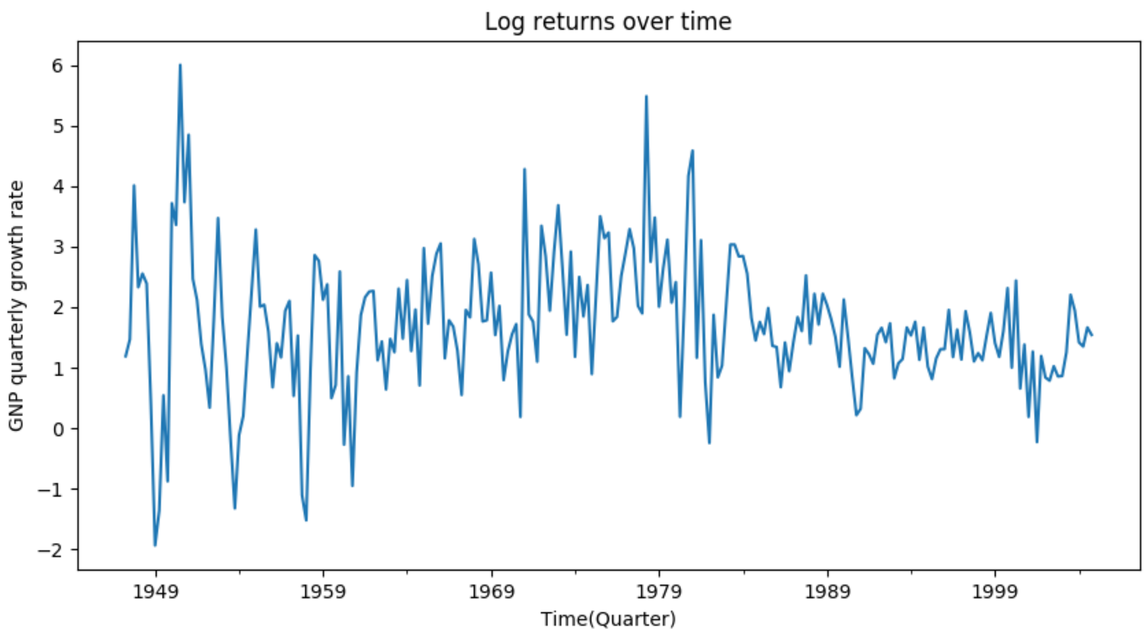

Figure 1: Log-returns for GNP training data

(ii) Plot of the Auto-correlation function of the training data

(iii) Plot of the Partial auto-correlation function of the training data

(c) A multi-step ahead forecast is where we use time series observations up to a given period t to generate a multi-step forecast $k(k>1)$, where k is the forecast horizon. Thus we use y_t to generate a forecast for y_{t+k}. A direct forecast models and estimates y_{t+k} directly using a horizon specific model while an iterative forecast models and estimates y_{t+1} first and iterates this forward until it reaches y_{t+k}. A direct forecast is more robust to model mis-specification while an iterative forecast is better if the one step model is correctly specified.

1.2 Exponential Smoothing

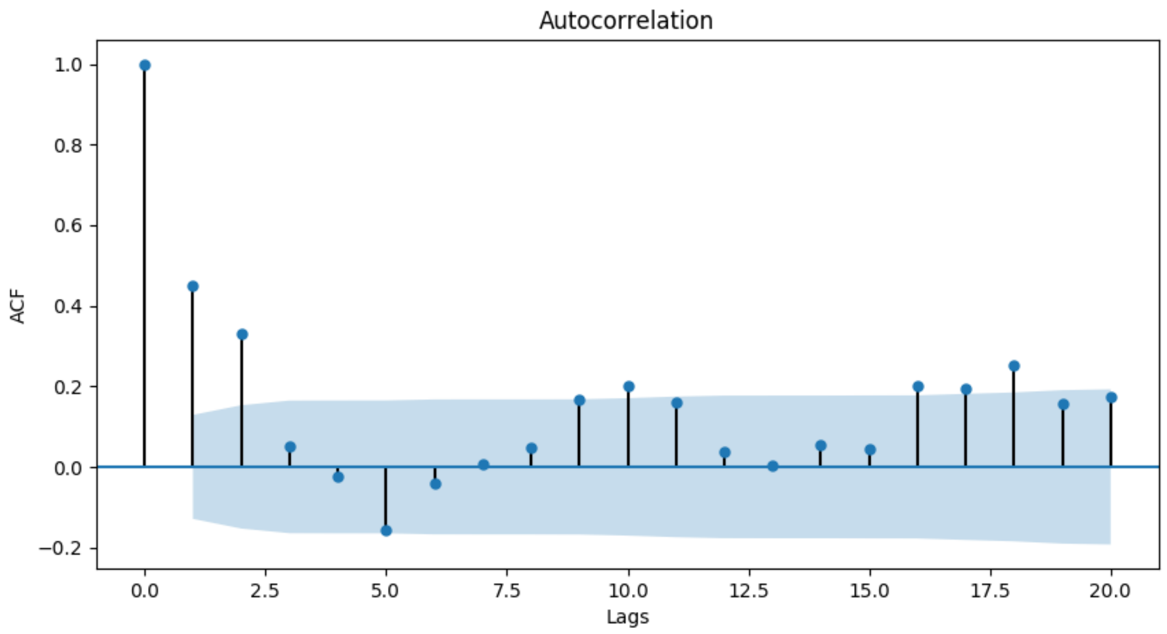

Figure 2: Autocorrelation plot for training data

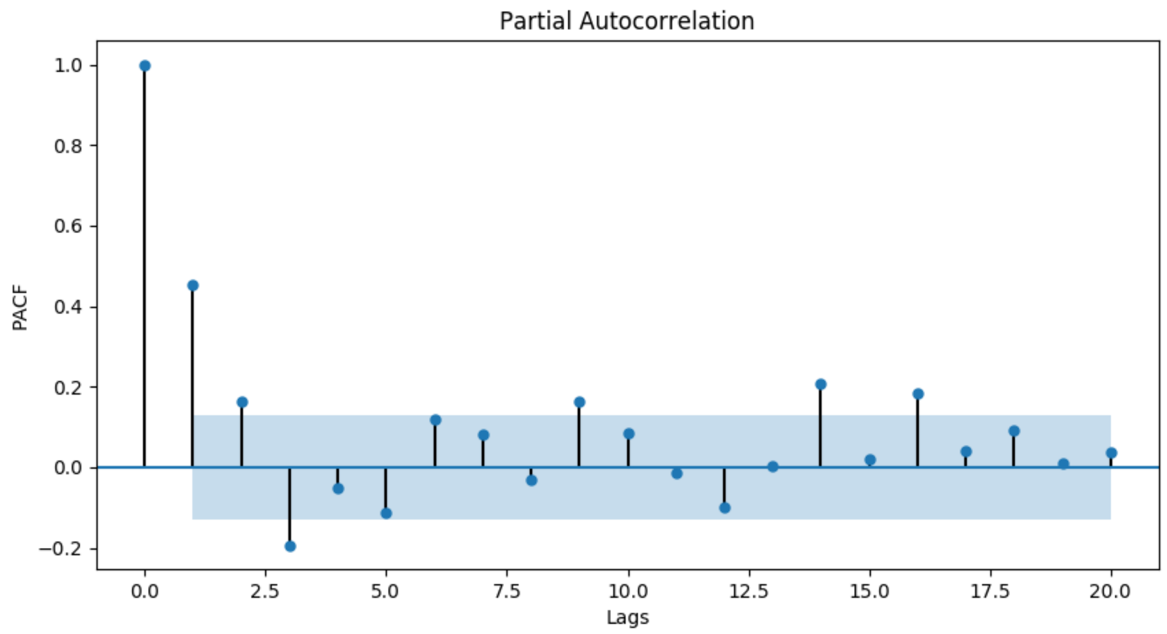

Figure 3: Partial Autocorrelation plot for training data

(a) Single exponential smoothing uses an exponential window function to smooth the time series data. While a simple moving average (MA) model assigns equal weights to past observations, SES uses exponential functions to assign decreasing weights to past observations with time. The smoothing factor α determines the rate of decay of old observations. Setting the k-step ahead forecast: $y_{t+k} = s_t$, we can define s_t recursively as:
$s_0 = y_0$
$s_t = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + ...$
$s_t = \alpha y_t + (1-\alpha)s_{t-1}$

(b) Plot of in-sample SSE vs alpha:

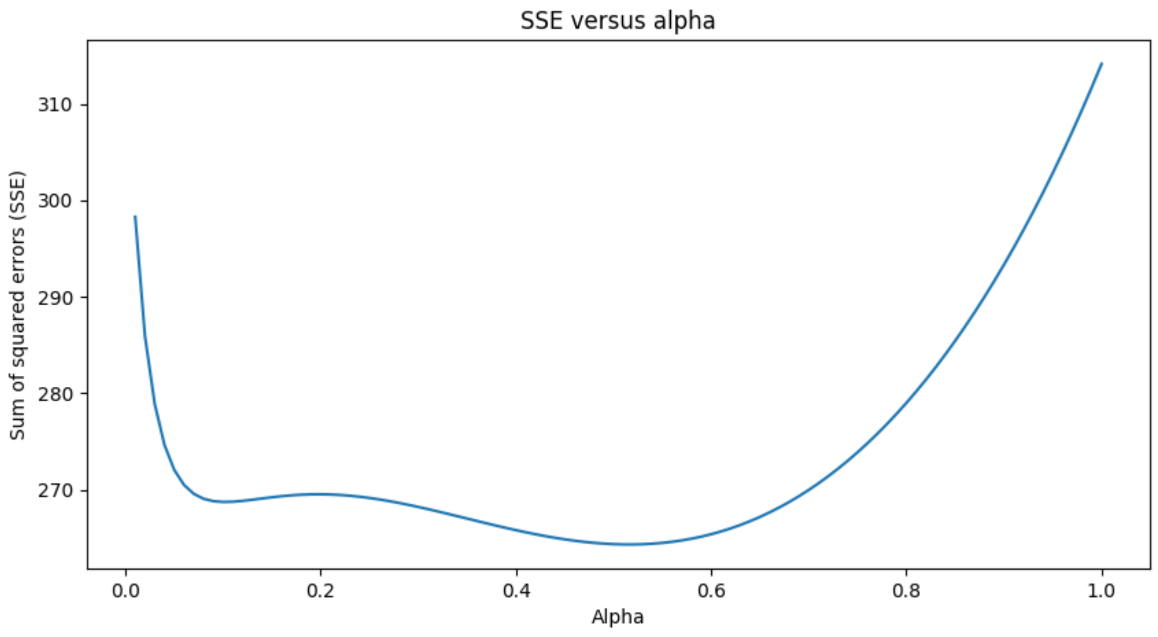

Figure 4: In-sample SSE versus different values of the smoothing parameter α

(c) The optimal value of alpha that minimises the in-sample SSE is 0.52. Thus we can set $\alpha_{opt} = 0.52$

(d) Plot of in-sample model response along with the corresponding actual predictions:

1.3 Auto-Regressive models

(a) To find the optimal order, I experimented with the statsmodels package in Python, specifically the AR and ARMA models. Each of them gave me similar optimal orders. I chose to go with the ARMA model with q set to 0 for the MA part. The optimal order of the Auto-Regressive model according to the ARMA model was 16.

(b) There are 17 coefficients for the AR(16) model.They are:

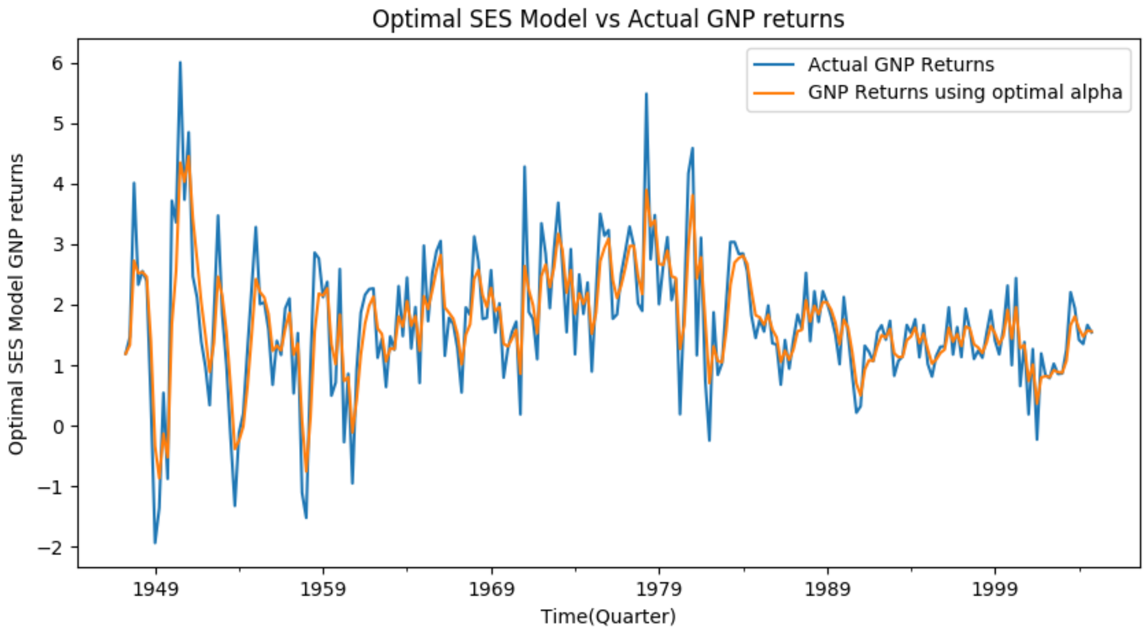

Figure 5: In-sample optimal SES predictions vs actual observations for GNP log returns

- intercept = 1.68
- α_1 = 0.38
- α_2 = 0.23
- α_3 = -0.12
- α_4 = -0.0077
- α_5 = -0.19
- α_6 = 0.11
- α_7 = 0.021
- α_8 = -0.11
- α_9 = 0.14
- α_{10} = 0.090
- α_{11} = 0.077
- α_{12} = -0.15
- α_{13} = -0.054
- α_{14} = 0.15
- α_{15} = -0.065
- α_{16} = 0.21

1.4 Out-of-sample forecasting

(a) Out-of-sample forecasting is used for model evaluation as a good fit on training data does not always translate into a good prediction on test data.

- Persistence - A persistence forecast corresponds to assuming that the underlying dynamics are generated by a random walk. Given a training data set $\{y_t\}$ and a horizon k, the prediction for $\hat{y}_{t+k}$ is:, the prediction for a data point $\hat{y}_{t+k}$ is y_t. The forecasting scheme is given by the equation:

$$y_t \rightarrow \hat{y}_{t+k}$$

- Climatology - The climatology forecast uses the distribution of the in-sample observations as a forecast for out-of-sample data. The unconditional average represents the expected value of the unconditional distribution of the training data. Given a training data set $\{y_t\}$ and a horizon k, the prediction for $\hat{y}_{t+k}$ is:

$$\frac{1}{t}\sum_{i=1}^{t} y_i \rightarrow \hat{y}_{t+k}$$

  Climatology forecasting does not give importance to the time ordering of the observations. Another idea is to take the mean of only the previous ρ observations, where ρ is estimated using cross validation.

- SES(α_{opt}) - Single exponential smoothing uses an exponential window function to smooth the time series data. While a simple moving average (MA) model assigns equal weights to past observations, SES uses exponential functions to assign decreasing weights to past observations with time. The smoothing factor α determines the rate of decay of old observations. Given a training data set $\{y_t\}$ and a horizon k, the prediction for $\hat{y}_{t+k}$ is:

$$\begin{aligned} s_t &\rightarrow \hat{y}_{t+k} \\ s_t &= \alpha_{opt} y_t + (1-\alpha_{opt}) s_{t-1} \\ s_0 &= y_0 \end{aligned}$$

- AR(p_{opt}) - An AR(p) model assumes a linear relationship between observations. The order p refers to the number of lagged variables in the model. Forecast for out-of-sample points is the most recent observation. Given a training data set $\{y_t\}$ and a horizon k, the prediction for $\hat{y}_{t+k}$ is:

$$y_t \rightarrow \hat{y}_{t+k}$$

$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \epsilon_t, \epsilon \sim N(0, \sigma^2)$$

  where α_i denotes AR model parameters and y_t depends linearly on the previous P values.

(b) Plot of out-of-sample optimal AR Model response along with the corresponding actual predictions and PDF of out-of-sample errors

(c) Plot of out-of-sample root mean square error versus the forecast horizon for the four models - Persistence, Climatology, AR(p_{opt}) and SES(α_{opt})

(d) Plot of out-of-sample mean absolute error versus the forecast horizon for the four models - Persistence, Climatology, AR(p_{opt}) and SES(α_{opt})

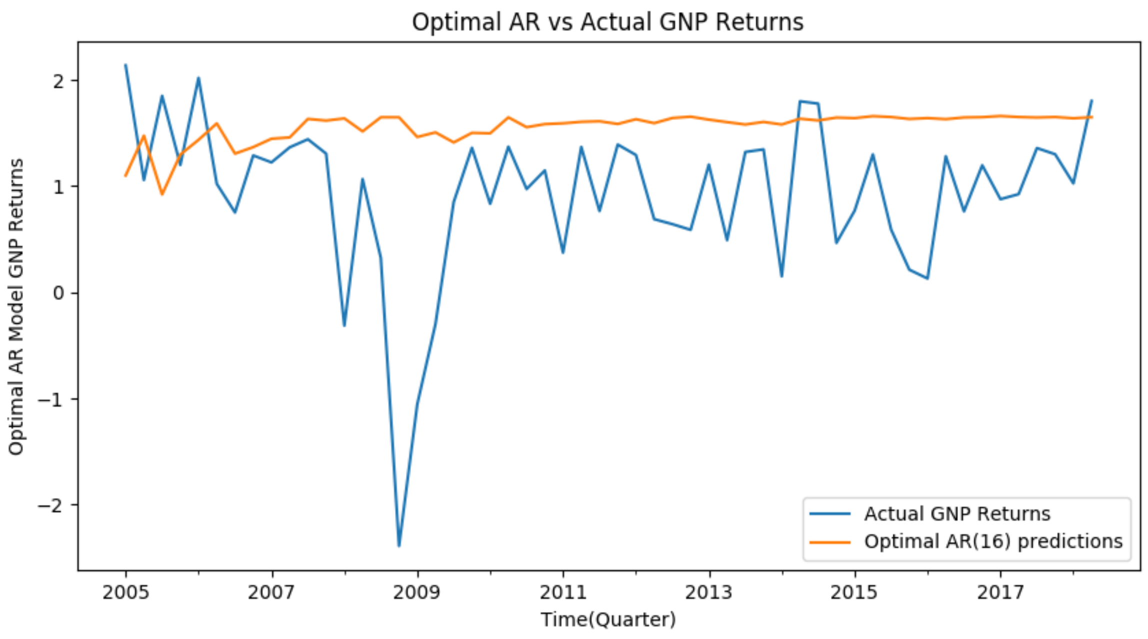

Figure 6: Out-of-sample optimal AR predictions vs testing data for GNP log returns

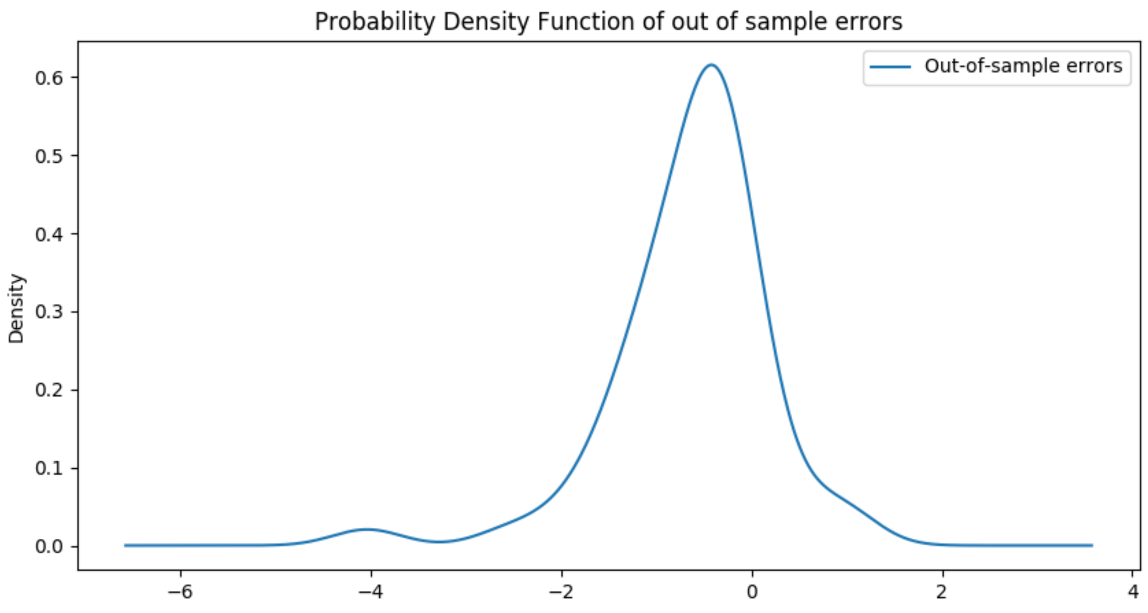

Figure 7: PDF of out of sample errors for the optimal AR model vs the testing data

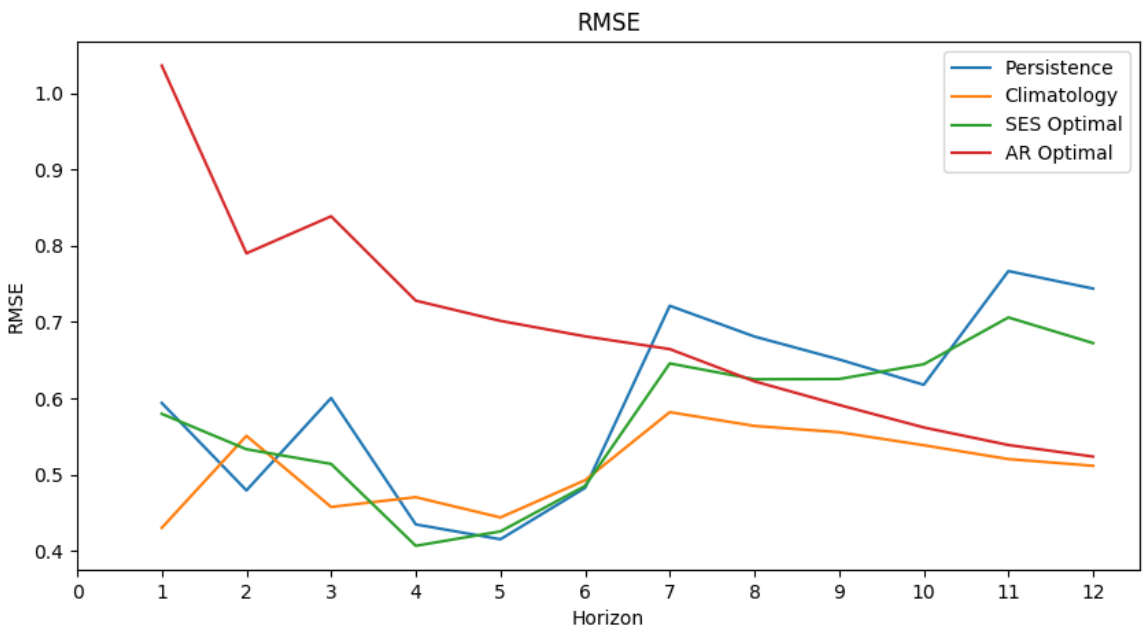

Figure 8: Out-of-sample RMSE vs forecast horizon for four models

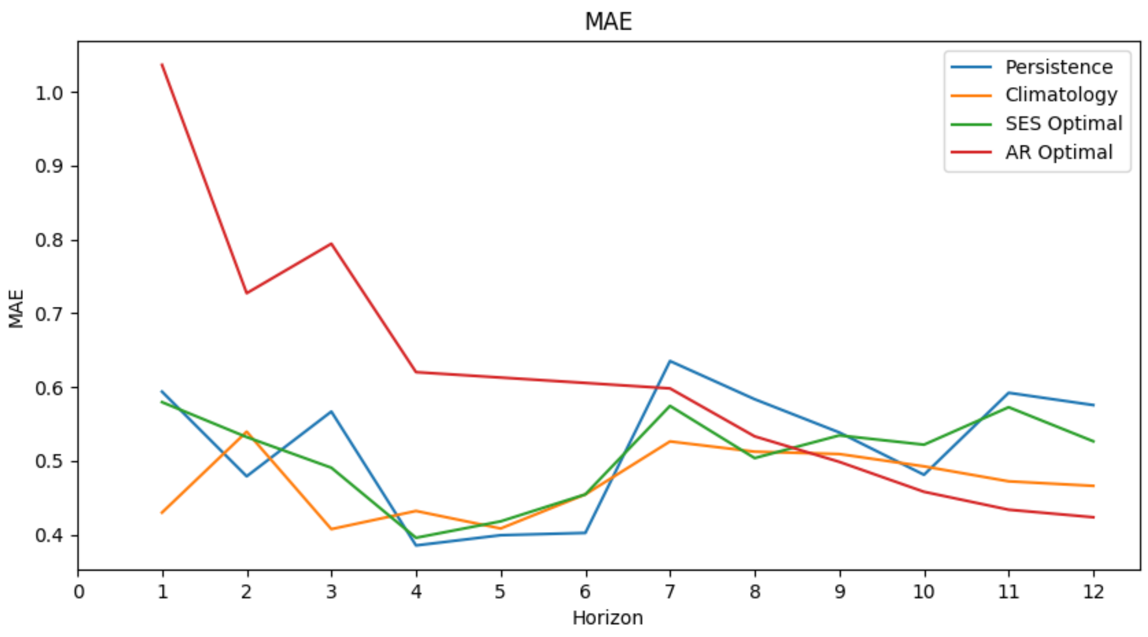

Figure 9: Out-of-sample MAE vs forecast horizon for four models

(e) Based on the mean absolute error and the root mean square error for the four models, the two best models are Climatology and the SES(α_{opt}). To generate an out-of-sample forecast using the Naive Composite, we take the average of the forecasts for each of these models. Figure 10 and 11 show the RMSE and MAE of all 5 models with the Naive Composite included.

(f) The model with the most accurate out-of-sample forecast was the Climatology model, as can be seen in Fig 10 and 11. It has the lowest mean absolute error and root mean square error. This could be because it is the unconditional mean benchmark for the time-series. It assumes that the ordering of the data points does not provide additional information on future predictions which is true of the GNP data as a good or bad performance in one year does not necessarily mean the trend will continue.

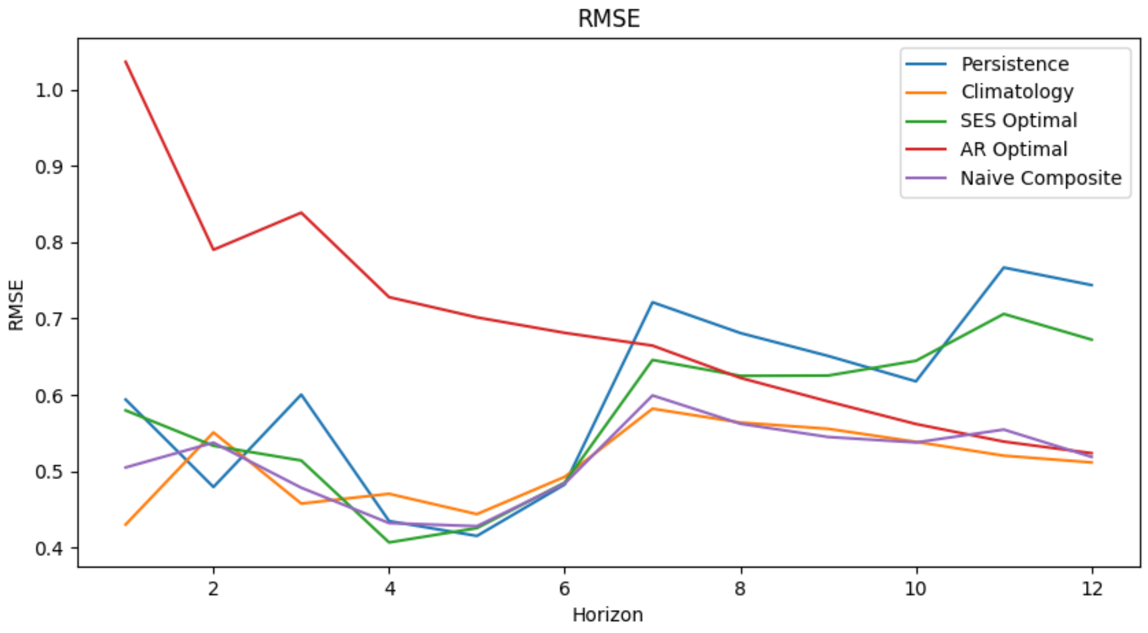

Figure 10: Out-of-sample RMSE vs forecast horizon for five models including Naive Composite

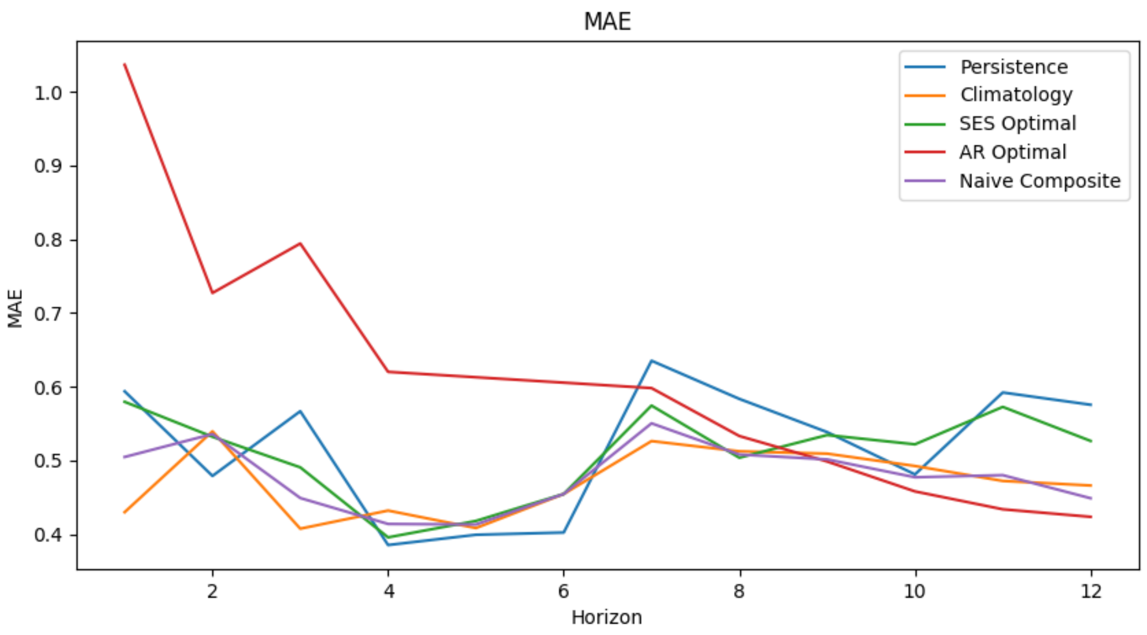

Figure 11: Out-of-sample MAE vs forecast horizon for five models including Naive Composite

1 Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA

#helper functions
 def f(x):
   if x.year>2018:
      x = x.replace(year=x.year-100)
   return x

# 1.1
# set PATH accordingly
gnp_data = pd.read_csv(PATH, index_col='DATE')
log_returns = np.log(gnp_data.GNP/gnp_data.shift(1).GNP)*100
log_returns.dropna(inplace=True)
dates_index = pd.to_datetime(log_returns.index)
dates = map(f, dates_index)
log_returns.index = dates

training_data = log_returns[:231]
testing_data = log_returns[231:]


# plotting training data log returns
training_returns_plot, ax = plt.subplots(figsize=(10,5))
training_returns_plot = training_data.plot()
training_returns_plot.set(xlabel="Time(Quarter)", ylabel="GNP quarterly growth rate",
    title="Log returns over time")

# plotting training data acf
acf_plot, ax = plt.subplots(figsize=(10,5))
ax.set_xlabel('Lags')
ax.set_ylabel('ACF')
acf_plot = plot_acf(training_data, ax=ax, lags=20)


# plotting training data pacf
pacf_plot, ax = plt.subplots(figsize=(10,5))
ax.set_xlabel('Lags')
ax.set_ylabel('PACF')
pacf_plot = plot_pacf(training_data, ax=ax, lags=20)


# 1.2 Exponential Smoothing
```

```
ses = training_data.to_frame('y')
alpha_values = np.arange(0.01, 1.01, 0.01)

for alpha in alpha_values:
   col_name = 's_{}'.format(alpha)
   #TODO: Check if this should be 1.0
   ses[col_name]=ses.y.tolist()[0]
   for i in range(1, len(ses)):
      ses[col_name][i] = ses[col_name][i-1]*(1-alpha) + alpha*ses.y[i]

errors = pd.DataFrame()
errors['alpha'] = alpha_values
error_vals = []
for alpha in alpha_values:
   error_diff = (ses['s_{}'.format(alpha)].shift(1)-ses['y']).dropna()
   error_val = sum(error_diff**2)
   error_vals.append(error_val)

 errors['sse'] = error_vals
 errors.set_index('alpha', inplace=True)

# plot of alpha values versus in sample SSE
sse_plot, ax = plt.subplots(figsize=(10,5))
sse_plot = errors.plot(figsize=(10,5))
sse_plot.set(xlabel="Alpha", ylabel="Sum of squared errors (SSE)", title="SSE versus
    alpha")

# optimal value of alpha that minimises in sample SSE
optimal_alpha = errors.idxmin()

# plot of in-sample model response vs optimal alpha
optimal_df = ses[['y', 's_0.52']]
optimal_df = optimal_df.rename(columns={'y':'Actual GNP Returns', 's_0.52':'GNP Returns
    using optimal alpha'})
optimal_plot = optimal_df.plot(figsize=(10,5))
optimal_plot.set(xlabel="Time(Quarter)", ylabel="Optimal SES Model GNP returns",
    title="Optimal SES Model vs Actual GNP returns")

# 1.3 Autoregressive models

min_aic = 20000000
min_order=0
for p in range(1,21):
   model = ARMA(training_data, order=(p,0))
   model_fit = model.fit()
   model_aic = model_fit.aic
   if model_aic < min_aic:
      min_aic = model_aic
      min_order = p
```

```
# Out of sample forecasting

persistence_forecast = []
climatology_forecast = []
ses_forecast = []
ar_forecast = []

avg_vals = [training_data[:-k].mean() if k>0 else training_data.mean() for k in
    range(12,-1,-1)]

ar_optimal = ARMA(training_data, order=(16,0))
ar_fit = ar_optimal.fit()
params = ar_fit.params
residuals = ar_fit.resid
p = ar_fit.k_ar
q = ar_fit.k_ma
k_exog = ar_fit.k_exog
k_trend = ar_fit.k_trend
for k in range(1,13):
   persistence_forecast.append(pd.Series(data=training_data[-k:].values,
       index=testing_data[:k].index))
   climatology_forecast.append(pd.Series(data = avg_vals[-k:],
       index=testing_data.index[:k]))
   ses_forecast.append(pd.Series(data=optimal_df['GNP Returns using optimal
       alpha'][-k:].values, index=testing_data[:k].index))
   ar_prediction = pd.Series(data=_arma_predict_out_of_sample(params, k, residuals, p, q,
       k_trend, k_exog, endog=training_data[1:], exog=None, start=len(training_data[1:])),
       index=testing_data[:k].index)
   ar_forecast.append(ar_prediction)

ar_prediction = pd.Series(data=_arma_predict_out_of_sample(params, len(testing_data),
    residuals, p, q, k_trend, k_exog, endog=training_data, exog=None,
    start=len(training_data)), index=testing_data.index)
preds=ar_prediction.to_frame('Optimal AR(16) predictions')
td = testing_data.to_frame('Actual GNP Returns')
ar_vs_td = td.join(preds)

ar_vs_td_plot = ar_vs_td.plot(figsize=(10,5))
ar_vs_td_plot.set(xlabel="Time(Quarter)", ylabel="Optimal AR Model GNP Returns",
    title="Optimal AR vs Actual GNP Returns")

ar_vs_td['Out-of-sample errors'] = ar_vs_td['Actual GNP Returns'] - ar_vs_td['Optimal
    AR(16) predictions']
pdf_plot = ar_vs_td[['Out-of-sample errors']].plot.kde(figsize=(10,5))
pdf_plot.set(title="Probability Density Function of out of sample errors")

rmse_persistence_data = []
mae_per_data = []
for p in persistence_forecast:
   k = len(p)
   e = p - testing_data.head(k)
   rmse_persistence_data.append((e**2).mean()**.5)
   mae_per_data.append(e.abs().mean())
rmse_persistence = pd.Series(data=rmse_persistence_data,
    index=range(1,13)).to_frame('Persistence')
mae_persistence = pd.Series(data=mae_per_data, index=range(1,13)).to_frame('Persistence')

rmse_climatology_data = []
mae_clim_data = []
for p in climatology_forecast:
   k = len(p)
   e = p - testing_data.head(k)
   rmse_climatology_data.append((e**2).mean()**.5)
   mae_clim_data.append(e.abs().mean())
rmse_climatology = pd.Series(data=rmse_climatology_data,
    index=range(1,13)).to_frame('Climatology')
mae_climatology = pd.Series(data=mae_clim_data, index=range(1,13)).to_frame('Climatology')

rmse_ses_data = []
ses_mae_data = []
for p in ses_forecast:
   k = len(p)
   e = p - testing_data.head(k)
   rmse_ses_data.append((e**2).mean()**.5)
   ses_mae_data.append(e.abs().mean())
rmse_ses = pd.Series(data=rmse_ses_data, index=range(1,13)).to_frame('SES Optimal')
mae_ses = pd.Series(data=ses_mae_data, index=range(1,13)).to_frame('SES Optimal')


rmse_ar_data = []
mae_ar_data = []
for p in ar_forecast:
   k = len(p)
   e = p - testing_data.head(k)
   rmse_ar_data.append((e**2).mean()**.5)
   mae_ar_data.append(e.abs().mean())
rmse_ar = pd.Series(data=rmse_ar_data, index=range(1,13)).to_frame('AR Optimal')
mae_ar = pd.Series(data=mae_ar_data, index=range(1,13)).to_frame('AR Optimal')


rmse_df = rmse_persistence.join(rmse_climatology).join(rmse_ses).join(rmse_ar)
rmse_plot = rmse_df.plot(figsize=(10,5))
rmse_plot.set(xlabel="Horizon", ylabel="RMSE", title="RMSE")


mae_df = mae_persistence.join(mae_climatology).join(mae_ses).join(mae_ar)
mae_plot = mae_df.plot(figsize=(10,5))
mae_plot.set(xlabel="Horizon", ylabel="MAE", title="MAE")

# naive composite forecast scheme
```

```
naive_composite = [(climatology_forecast[i]+ses_forecast[i])/2 for i,x in
    enumerate(climatology_forecast)]

rmse_naive_data = []
mae_naive_data = []
for p in naive_composite:
   k = len(p)
   e = p - testing_data.head(k)
   rmse_naive_data.append((e**2).mean()**.5)
   mae_naive_data.append(e.abs().mean())
rmse_naive = pd.Series(data=rmse_naive_data, index=range(1,13)).to_frame('Naive Composite')
mae_naive = pd.Series(data=mae_naive_data, index=range(1,13)).to_frame('Naive Composite')

rmse_df['Naive Composite'] = rmse_naive['Naive Composite']
mae_df['Naive Composite'] = mae_naive['Naive Composite']

rmse_naive_plot = rmse_df.plot(figsize=(10,5))
rmse_naive_plot.set(xlabel="Horizon", ylabel="RMSE", title="RMSE")
mae_naive_plot = mae_df.plot(figsize=(10,5))
mae_naive_plot.set(xlabel="Horizon", ylabel="MAE", title="MAE")
```