DATS 6313 – Time Series Analysis & Modeling

Instructor: Reza Jafari

Lab #4

Bradley Reardon

2/16/2022

1 – Abstract:

This lab pertains to implementing and comparing the performance of 4 simple forecasting methods:

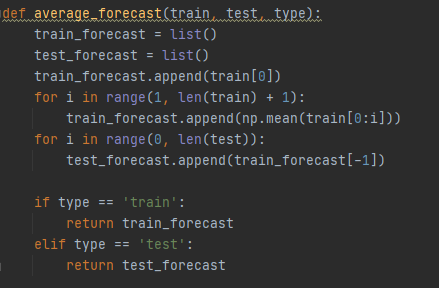
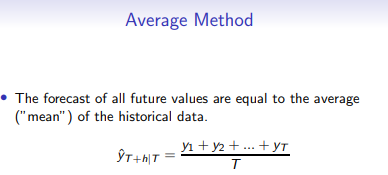
* Average method
* Naïve method
* Drift method
* Simple Exponential Smoothing (SES)

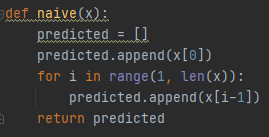
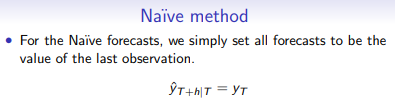
2 – Introduction:

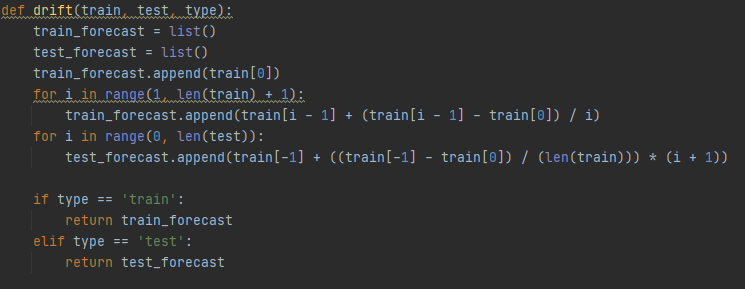
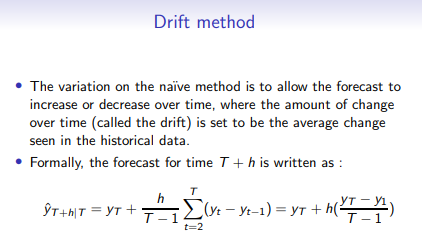
This experiment was performed to increase understanding of the four simple forecasting methods by learning how to calculate the methods given a train and test dataset, creating programs using python to calculate and plot the data, and comparing results.

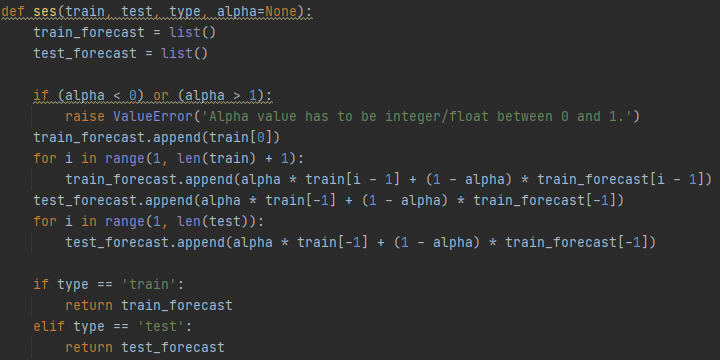
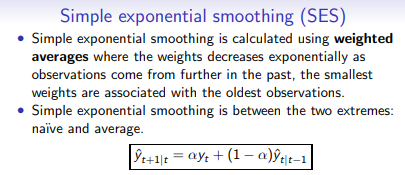
3 – Method, Theory, and Procedures:

Forecasting involves predicting values based on a given dataset. In time series forecasting, the four simple methods commonly used are average naïve, drift, and SES. The following formulas and python functions indicate how to calculate forecasted values per method type:





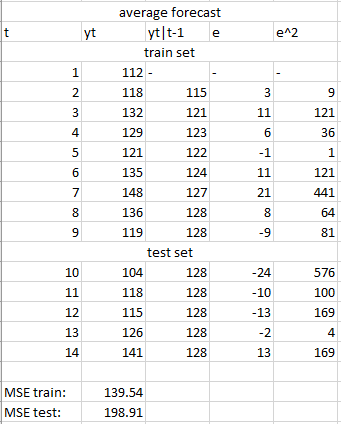
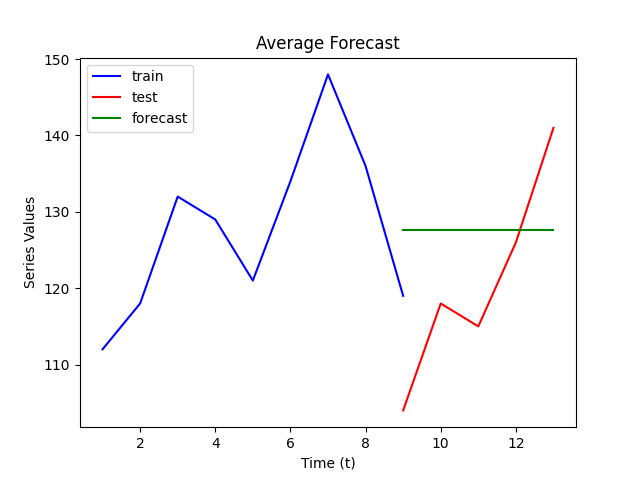
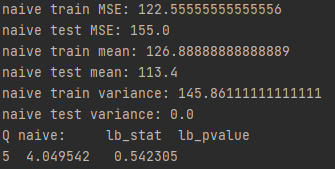
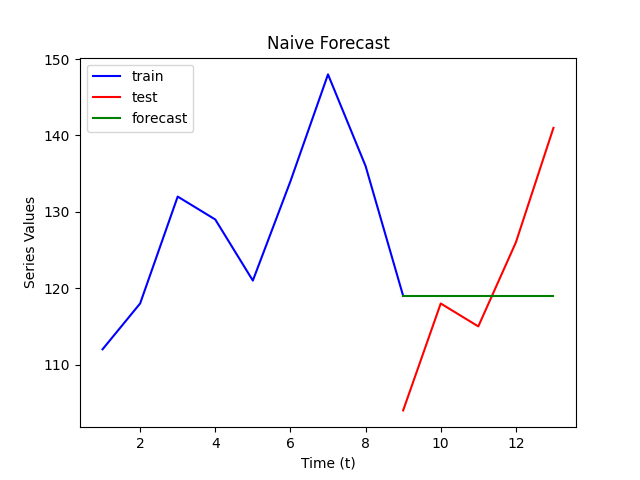
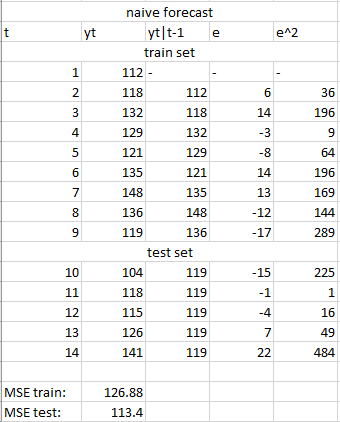
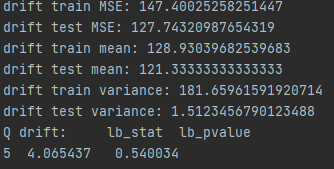
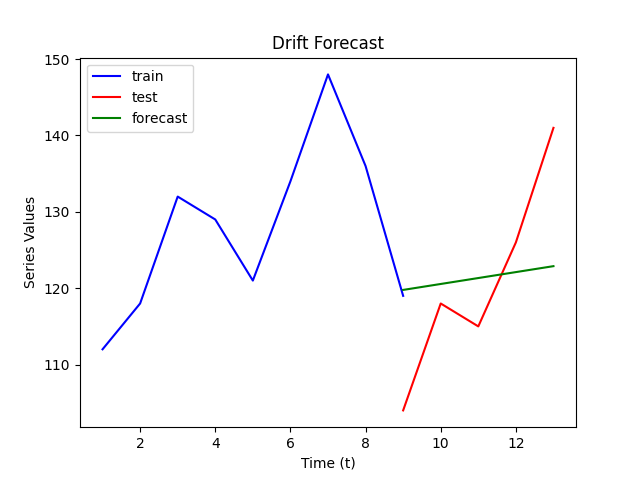
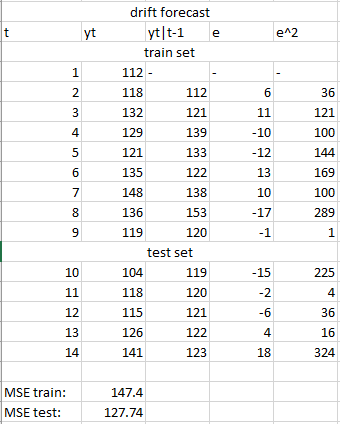
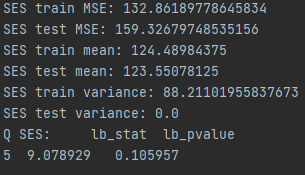
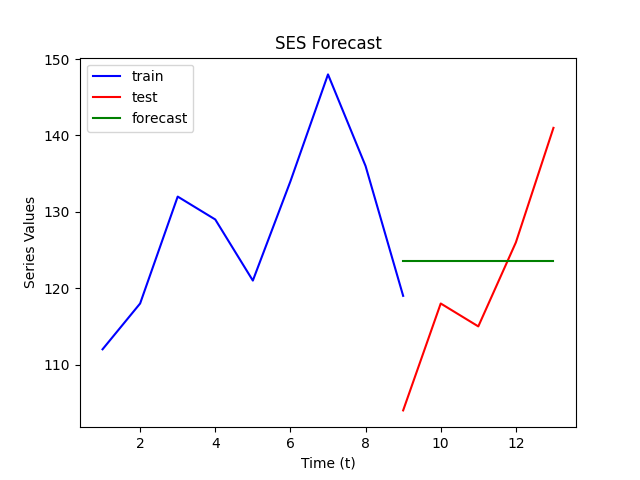
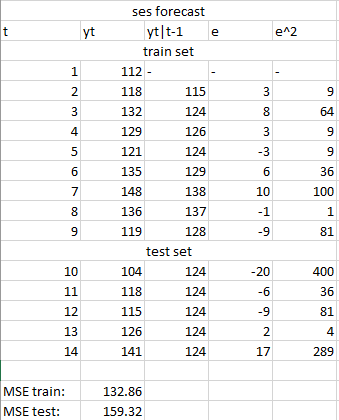
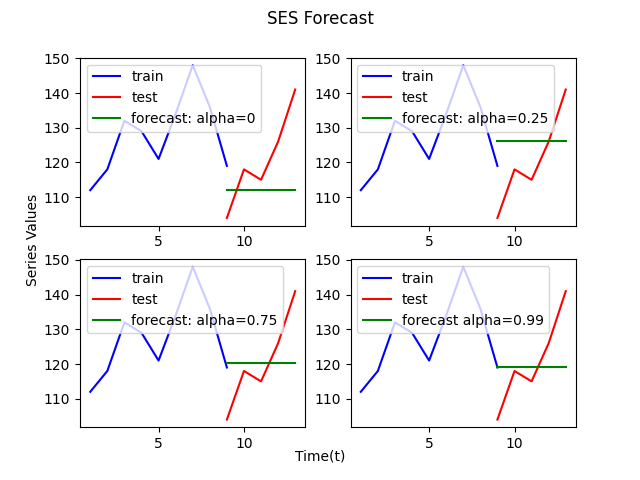
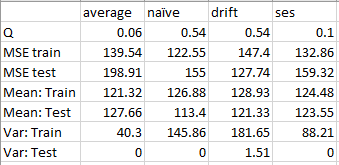
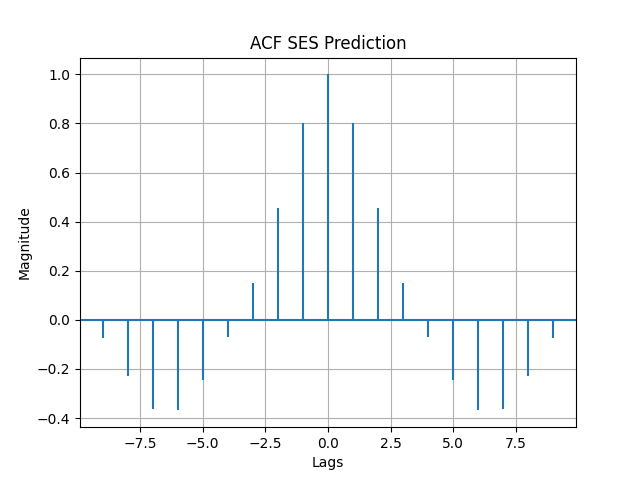
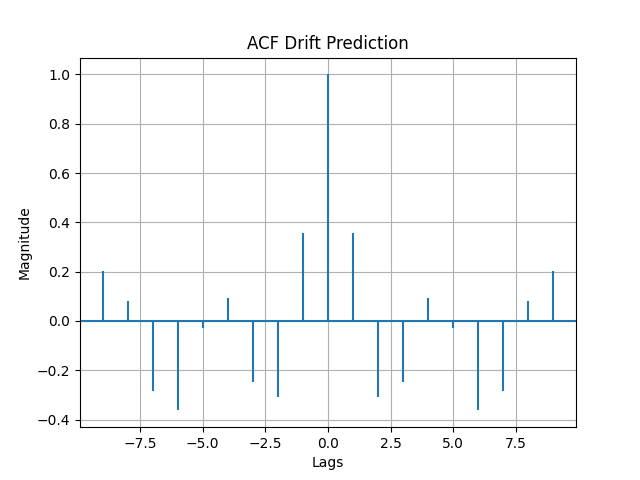
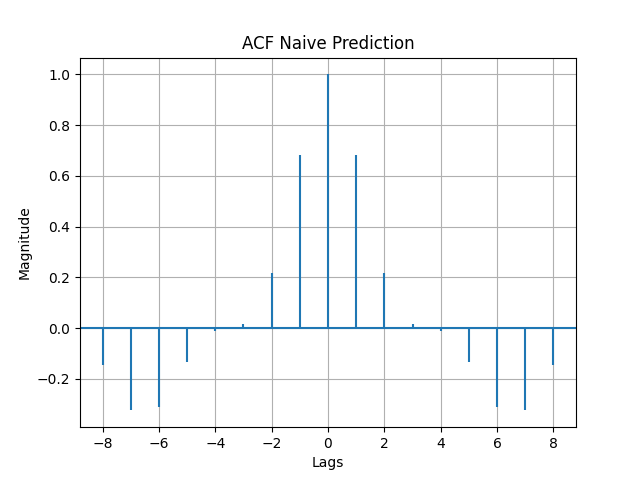
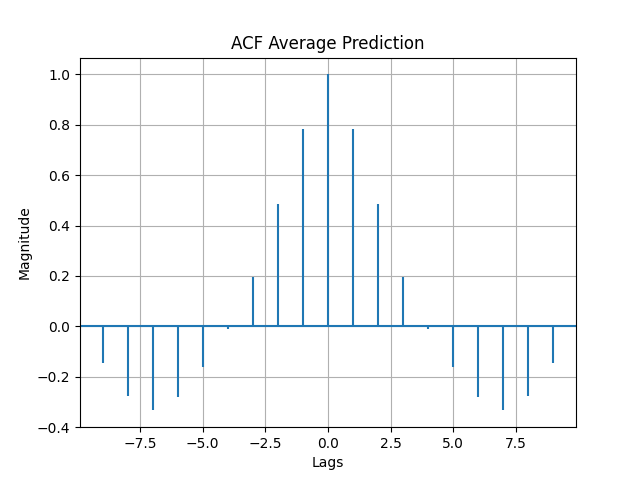




The following data was used throughout this experiment:

train = [112, 118, 132, 129, 121, 134, 148, 136, 119]  
test = [104, 118, 115, 126, 141]

4 – Answers to Lab Questions:

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. The best predictor would be

5 – Conclusion:

The four simple methods of forecasting provide some level of insight into possible future values, but none of the them provide in-depth complicated predictions. The results should not be taken as guaranteed values.

6 – Appendix

import matplotlib.pyplot as plt  
import numpy as np  
from statistics import variance  
import statsmodels.api as sm  
from toolbox import autocorrelation\_plot  
  
t\_train = [112, 118, 132, 129, 121, 134, 148, 136, 119]  
t\_test = [104, 118, 115, 126, 141]  
combined = t\_train + t\_test  
  
  
# question 2  
  
print(combined)  
def average\_forecast(train, test, type):  
 train\_forecast = list()  
 test\_forecast = list()  
 train\_forecast.append(train[0])  
 for i in range(1, len(train) + 1):  
 train\_forecast.append(np.mean(train[0:i]))  
 for i in range(0, len(test)):  
 test\_forecast.append(train\_forecast[-1])  
  
 if type == 'train':  
 return train\_forecast  
 elif type == 'test':  
 return test\_forecast  
  
print('----------------------average---------------------')  
print(average\_forecast(t\_train, t\_test, type='train'))  
print(average\_forecast(t\_train, t\_test, type='test'))  
  
plt.plot(np.arange(1,10), t\_train, c='b', label='train')  
plt.plot(np.arange(9,14), t\_test, c='r', label='test')  
plt.plot(np.arange(9,14), average\_forecast(t\_train, t\_test, type='test'), c='g', label='forecast')  
plt.legend(loc = 'upper left')  
plt.xlabel('Time (t)')  
plt.ylabel('Series Values')  
plt.title('Average Forecast')  
plt.show()  
  
# question 3  
  
def MSE(original, prediction):  
 error\_squared = 0  
 for i in range(len(original)):  
 error = original[i] - prediction[i]  
 error\_squared += error \*\* 2  
 mse = error\_squared/len(original)  
 return mse  
  
print('average train MSE:', MSE(t\_train, average\_forecast(t\_train, t\_test, type='train')))  
print('average test MSE:', MSE(t\_test, average\_forecast(t\_train, t\_test, type='test')))  
  
  
# question 4  
print('average train mean:', np.mean(average\_forecast(t\_train, t\_test, type = 'train')))  
print('average test mean:', np.mean(average\_forecast(t\_train, t\_test, type = 'test')))  
print('average train variance:', variance(average\_forecast(t\_train, t\_test, type='train')))  
print('average test variance:', variance(average\_forecast(t\_train, t\_test, type='test')))  
  
# question 5  
print('Q average:', sm.stats.acorr\_ljungbox(average\_forecast(t\_train, t\_test, type='train'), lags=[5], return\_df=True))  
  
# question 6  
  
def naive(x):  
 predicted = []  
 predicted.append(x[0])  
 for i in range(1, len(x)):  
 predicted.append(x[i-1])  
 return predicted  
  
plt.plot(np.arange(1,10), t\_train, c='b', label='train')  
plt.plot(np.arange(9,14), t\_test, c='r', label='test')  
plt.plot(np.arange(9,14),np.ones(5)\*t\_train[-1], c='g', label='forecast')  
plt.legend(loc = 'upper left')  
plt.xlabel('Time (t)')  
plt.ylabel('Series Values')  
plt.title('Naive Forecast')  
plt.show()  
  
  
print('----------------------naive---------------------')  
print(naive(t\_train))  
print(naive(t\_test))  
print('naive train MSE:', MSE(t\_train, naive(t\_train)))  
print('naive test MSE:', MSE(t\_test, np.ones(5)\*t\_train[-1]))  
  
print('naive train mean:', np.mean(naive(t\_train)))  
print('naive test mean:', np.mean(naive(t\_test)))  
print('naive train variance:', variance(naive(t\_train)))  
print('naive test variance:', variance(np.ones(5)\*t\_train[-1]))  
  
print('Q naive:', sm.stats.acorr\_ljungbox(naive(t\_train), lags=[5], return\_df=True))  
  
# question 7  
  
def drift(train, test, type):  
 train\_forecast = list()  
 test\_forecast = list()  
 train\_forecast.append(train[0])  
 for i in range(1, len(train) + 1):  
 train\_forecast.append(train[i - 1] + (train[i - 1] - train[0]) / i)  
 for i in range(0, len(test)):  
 test\_forecast.append(train[-1] + ((train[-1] - train[0]) / (len(train))) \* (i + 1))  
  
 if type == 'train':  
 return train\_forecast  
 elif type == 'test':  
 return test\_forecast  
  
  
  
plt.plot(np.arange(1,10), t\_train, c='b', label='train')  
plt.plot(np.arange(9,14), t\_test, c='r', label='test')  
plt.plot(np.arange(9,14), drift(t\_train, t\_test, type = 'test'), c='g', label='forecast')  
plt.legend(loc = 'upper left')  
plt.xlabel('Time (t)')  
plt.ylabel('Series Values')  
plt.title('Drift Forecast')  
plt.show()  
  
print('----------------------drift---------------------')  
print(drift(t\_train, t\_test, type = 'train'))  
print(drift(t\_train, t\_test, type = 'test'))  
print('drift train MSE:', MSE(t\_train, drift(t\_train, t\_test, type = 'train')))  
print('drift test MSE:', MSE(t\_test, drift(t\_train, t\_test, type = 'test')))  
  
print('drift train mean:', np.mean(drift(t\_train, t\_test, type = 'train')))  
print('drift test mean:', np.mean(drift(t\_train, t\_test, type = 'test')))  
print('drift train variance:', variance(drift(t\_train, t\_test, type = 'train')))  
print('drift test variance:', variance(drift(t\_train, t\_test, type = 'test')))  
  
print('Q drift:', sm.stats.acorr\_ljungbox(drift(t\_train, t\_test, type = 'train'), lags=[5], return\_df=True))  
  
# question 8  
  
def ses(train, test, type, alpha=None):  
 train\_forecast = list()  
 test\_forecast = list()  
  
 if (alpha < 0) or (alpha > 1):  
 raise ValueError('Alpha value has to be integer/float between 0 and 1.')  
 train\_forecast.append(train[0])  
 for i in range(1, len(train) + 1):  
 train\_forecast.append(alpha \* train[i - 1] + (1 - alpha) \* train\_forecast[i - 1])  
 test\_forecast.append(alpha \* train[-1] + (1 - alpha) \* train\_forecast[-1])  
 for i in range(1, len(test)):  
 test\_forecast.append(alpha \* train[-1] + (1 - alpha) \* train\_forecast[-1])  
  
 if type == 'train':  
 return train\_forecast  
 elif type == 'test':  
 return test\_forecast  
  
  
plt.plot(np.arange(1,10), t\_train, c='b', label='train')  
plt.plot(np.arange(9,14), t\_test, c='r', label='test')  
plt.plot(np.arange(9,14), ses(t\_train, t\_test, type='test', alpha=0.5), c='g', label='forecast')  
plt.legend(loc = 'upper left')  
plt.xlabel('Time (t)')  
plt.ylabel('Series Values')  
plt.title('SES Forecast')  
plt.show()  
  
print('----------------------ses---------------------')  
print(ses(t\_train, t\_test, type='train', alpha=0.5))  
print(ses(t\_train, t\_test, type='test', alpha=0.5))  
print('SES train MSE:', MSE(t\_train, ses(t\_train, t\_test, type='train', alpha=0.5)))  
print('SES test MSE:', MSE(t\_test, ses(t\_train, t\_test, type='test', alpha=0.5)))  
  
print('SES train mean:', np.mean(ses(t\_train, t\_test, type = 'train', alpha=0.5)))  
print('SES test mean:', np.mean(ses(t\_train, t\_test, type = 'test', alpha=0.5)))  
print('SES train variance:', variance(ses(t\_train, t\_test, type='train', alpha=0.5)))  
print('SES test variance:', variance(ses(t\_train, t\_test, type='test', alpha=0.5)))  
  
print('Q SES:', sm.stats.acorr\_ljungbox(ses(t\_train, t\_test, type='train', alpha=0.5), lags=[5], return\_df=True))  
  
# question 9  
fig, ax = plt.subplots(2, 2)  
ax1, ax2, ax3, ax4 = ax.flatten()  
fig.suptitle('SES Forecast')  
ax1.plot(np.arange(1,10), t\_train, c='b', label='train')  
ax1.plot(np.arange(9,14), t\_test, c='r', label='test')  
ax1.plot(np.arange(9,14), ses(t\_train, t\_test, type='test', alpha=0), c='g', label='forecast: alpha=0')  
ax1.legend(loc='upper left')  
ax2.plot(np.arange(1,10), t\_train, c='b', label='train')  
ax2.plot(np.arange(9,14), t\_test, c='r', label='test')  
ax2.plot(np.arange(9,14), ses(t\_train, t\_test, type='test', alpha=0.25), c='g', label='forecast: alpha=0.25')  
ax2.legend(loc='upper left')  
ax3.plot(np.arange(1,10), t\_train, c='b', label='train')  
ax3.plot(np.arange(9,14), t\_test, c='r', label='test')  
ax3.plot(np.arange(9,14), ses(t\_train, t\_test, type='test', alpha=0.75), c='g', label='forecast: alpha=0.75')  
ax3.legend(loc='upper left')  
ax4.plot(np.arange(1,10), t\_train, c='b', label='train')  
ax4.plot(np.arange(9,14), t\_test, c='r', label='test')  
ax4.plot(np.arange(9,14), ses(t\_train, t\_test, type='test', alpha=0.99), c='g', label='forecast alpha=0.99')  
ax4.legend(loc='upper left')  
fig.text(0.5, 0.04, 'Time(t)', ha='center')  
fig.text(0.04, 0.5, 'Series Values', va='center', rotation='vertical')  
  
plt.show()  
  
# question 11  
autocorrelation\_plot(np.array(average\_forecast(t\_train, t\_test, type='train')), title='ACF Average Prediction', lag=9)  
autocorrelation\_plot(np.array(naive(t\_train)), title='ACF Naive Prediction', lag=8)  
autocorrelation\_plot(np.array(drift(t\_train, t\_test, type='train')), title='ACF Drift Prediction', lag=9)  
autocorrelation\_plot(np.array(ses(t\_train, t\_test, type='train', alpha=0.5)), title='ACF SES Prediction', lag=9)