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:

Average Method

- The forecast of all future values are equal to the average ("mean") of the historical data.

$$\hat{y}_{T+h|T} = \frac{y_1 + y_2 + ... + y_T}{T}$$

```python
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
```

Naïve method

- For the Naïve forecasts, we simply set all forecasts to be the value of the last observation.

$$\hat{y}_{T+h|T} = y_T$$

```python
def naive(x):
    predicted = []
    predicted.append(x[0])
    for i in range(1, len(x)):
        predicted.append(x[i-1])
    return predicted
```

Drift method

- The variation on the naïve method is to allow the forecast to increase or decrease over time, where the amount of change over time (called the drift) is set to be the average change seen in the historical data.
- Formally, the forecast for time $T + h$ is written as :

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h\left(\frac{y_T - y_1}{T-1}\right)$$

```python
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
```

Simple exponential smoothing (SES)

- Simple exponential smoothing is calculated using **weighted averages** where the weights decreases exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations.
- Simple exponential smoothing is between the two extremes: naïve and average.

$$\boxed{\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}}$$

```python
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
```

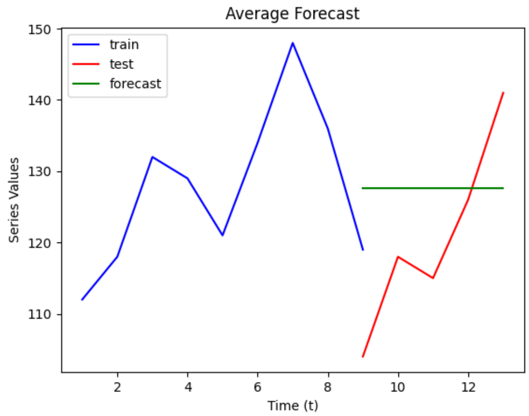
The following data was used throughout this experiment:

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

4 – Answers to Lab Questions:

1.

average forecast				
t	yt	yt|t-1	e	e^2
train set				
1	112	-	-	-
2	118	115	3	9
3	132	121	11	121
4	129	123	6	36
5	121	122	-1	1
6	135	124	11	121
7	148	127	21	441
8	136	128	8	64
9	119	128	-9	81
test set				
10	104	128	-24	576
11	118	128	-10	100
12	115	128	-13	169
13	126	128	-2	4
14	141	128	13	169
MSE train:	139.54			
MSE test:	198.91			



2.

```
average train MSE: 139.54373519778284
average test MSE: 198.91111111111118
```

3.

```
average train variance: 40.306314940791125
average test variance: 0.0
```

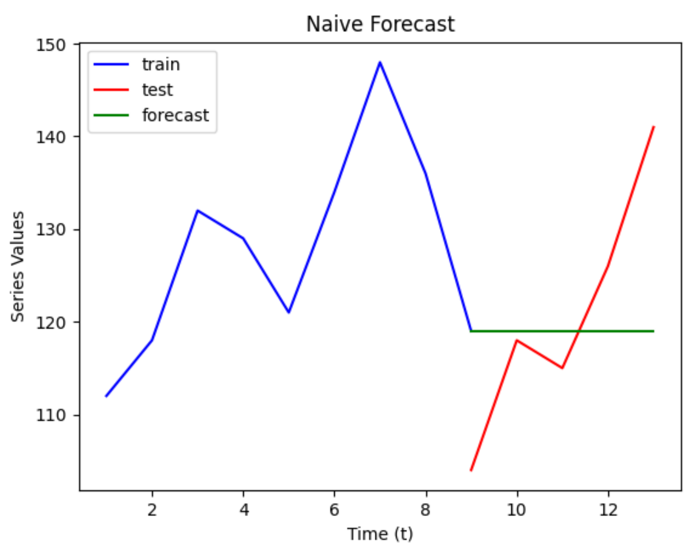
4.

```
Q average:      lb_stat   lb_pvalue
5  10.358237   0.065698
```

5.

6.

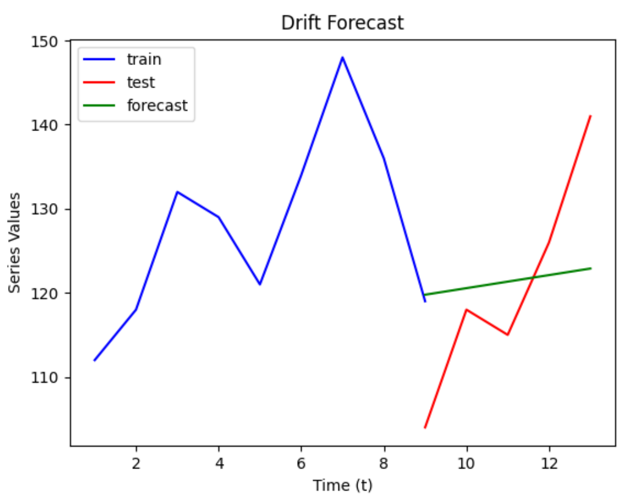
naive forecast				
t	yt	yt|t-1	e	e^2
train set				
1	112	-	-	-
2	118	112	6	36
3	132	118	14	196
4	129	132	-3	9
5	121	129	-8	64
6	135	121	14	196
7	148	135	13	169
8	136	148	-12	144
9	119	136	-17	289
test set				
10	104	119	-15	225
11	118	119	-1	1
12	115	119	-4	16
13	126	119	7	49
14	141	119	22	484
MSE train:	126.88			
MSE test:	113.4			




```
naive train MSE: 122.55555555555556
naive test MSE: 155.0
naive train mean: 126.88888888888889
naive test mean: 113.4
naive train variance: 145.86111111111111
naive test variance: 0.0
Q naive:     lb_stat  lb_pvalue
5  4.049542   0.542305
```

7.

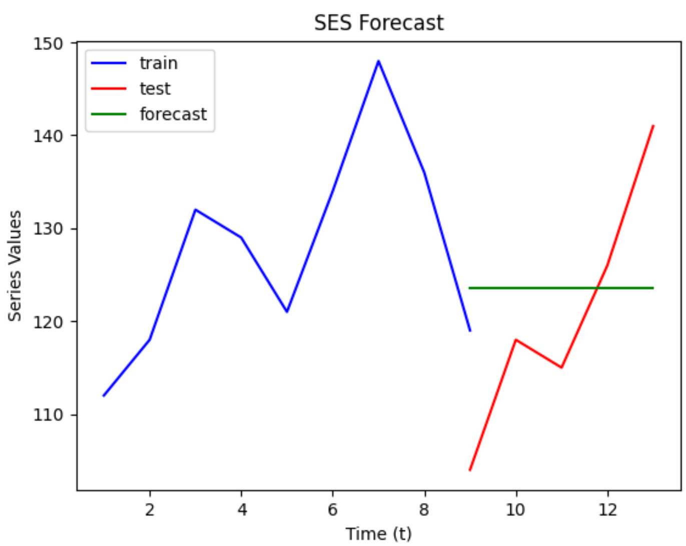
drift forecast				
t	yt	yt|t-1	e	e^2
train set				
1	112	-	-	-
2	118	112	6	36
3	132	121	11	121
4	129	139	-10	100
5	121	133	-12	144
6	135	122	13	169
7	148	138	10	100
8	136	153	-17	289
9	119	120	-1	1
test set				
10	104	119	-15	225
11	118	120	-2	4
12	115	121	-6	36
13	126	122	4	16
14	141	123	18	324
MSE train:	147.4			
MSE test:	127.74			



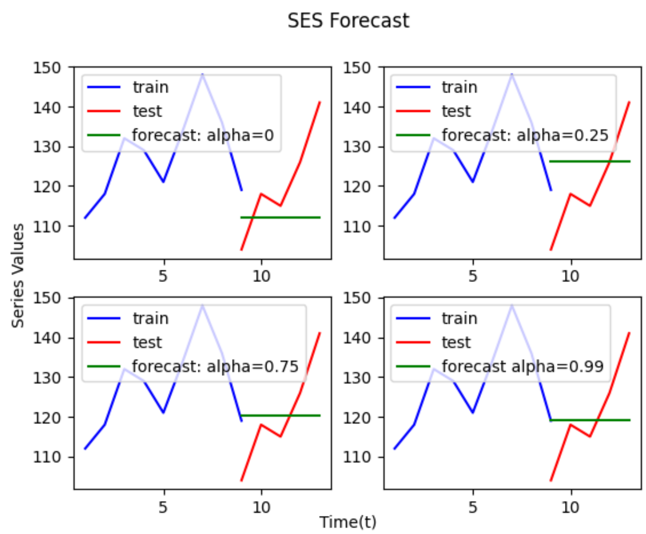

```
drift train MSE: 147.40025258251447
drift test MSE: 127.74320987654319
drift train mean: 128.93039682539683
drift test mean: 121.33333333333333
drift train variance: 181.65961591920714
drift test variance: 1.5123456790123488
Q drift:      lb_stat   lb_pvalue
5  4.065437   0.540034
```

8.

ses forecast				
t	yt	yt|t-1	e	e^2
train set				
1	112	-	-	-
2	118	115	3	9
3	132	124	8	64
4	129	126	3	9
5	121	124	-3	9
6	135	129	6	36
7	148	138	10	100
8	136	137	-1	1
9	119	128	-9	81
test set				
10	104	124	-20	400
11	118	124	-6	36
12	115	124	-9	81
13	126	124	2	4
14	141	124	17	289
MSE train:	132.86			
MSE test:	159.32			



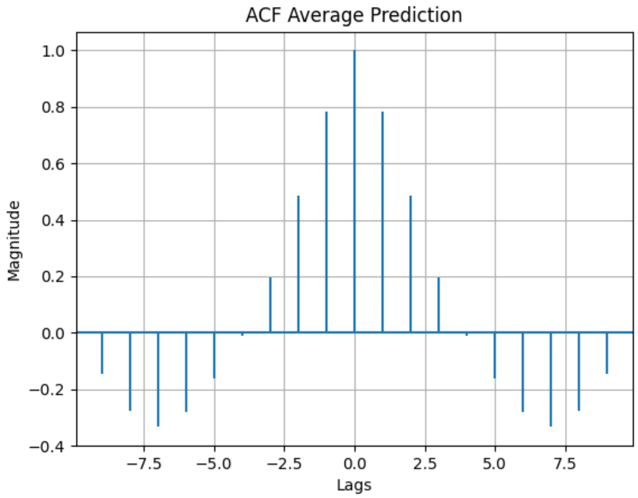

```
SES train MSE: 132.86189778645834
SES test MSE: 159.32679748535156
SES train mean: 124.48984375
SES test mean: 123.55078125
SES train variance: 88.21101955837673
SES test variance: 0.0
Q SES:      lb_stat   lb_pvalue
5  9.078929   0.105957
```



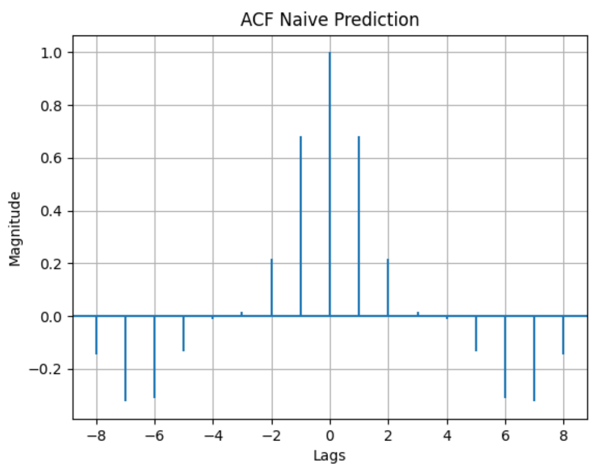
9.

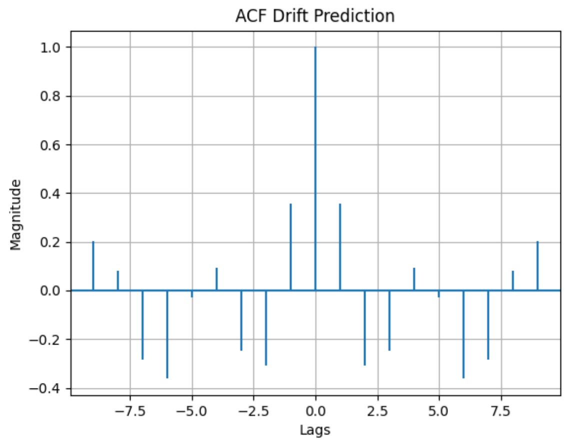
	average	naïve	drift	ses
Q	0.06	0.54	0.54	0.1
MSE train	139.54	122.55	147.4	132.86
MSE test	198.91	155	127.74	159.32
Mean: Train	121.32	126.88	128.93	124.48
Mean: Test	127.66	113.4	121.33	123.55
Var: Train	40.3	145.86	181.65	88.21
Var: Test	0	0	1.51	0

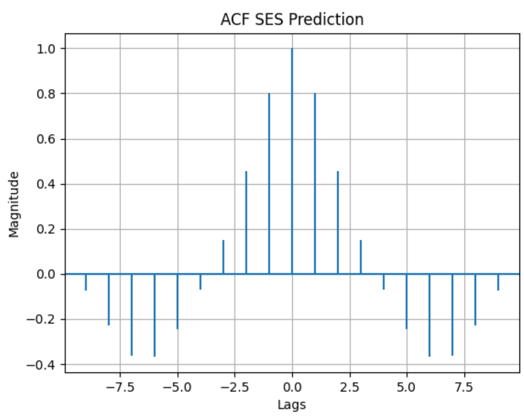
10.


ACF Average Prediction
Magnitude
Lags
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

11.


ACF Naive Prediction
Magnitude
Lags
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−8
−6
−4
−2
0
2
4
6
8





12. Based on the results I found when calculating the variance of predication and forecast errors, I would suggest using the variance of prediction as an estimator. The reason being that 3 of 4 methods resulted in 0 for the variance of forecast, meaning it would be tough to decide which method to use since they nearly all have the same values.

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, but can be used to help make actionable insights based on historical data and forecasted 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]
```

```python
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')))
```

```python
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
```

```python
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)
```