-E1 solution analyze

대부분 학생들의 코드는 동일했으나 세부적으로 pandas와 numpy사용에 차이가 있는 부분이 있어서 해당부분을 보내드립니다.

```
def transition(given_pi,states,P_normal,P_speed):
    P_out = np.zeros(shape=(8,8))

    for i in range(len(states)):
        action_dist=given_pi.iloc[i,:]

        P = action_dist['normal']*P_normal + action_dist['speed']*P_speed
```

#alternative way

```
def transition(given_pi, states, P_normal, P_speed):
    P_out=pd.DataFrame(np.zeros((len(states),len(states))),index=states, columns=states)

    for s in states:
        action_dist=given_pi.loc[s]
        P=action_dist['normal']*P_normal+action_dist['speed']*P_speed
        P_out.loc[s]=P.loc[s]

    return P_out
```

★추가적으로 pi_50을 선언할 때도 차이가 있었습니다.

```
pi_50 = np.hstack((np.repeat(0.5,len(states)).reshape(8,1),np.repeat(0.5,len(states)).reshape(8,1)))
pi_50 = pd.DataFrame(pi_50,states,["normal","speed"])
print(pi_50)
```

#alternative way

```
pi_50=pd.DataFrame(np.c_[np.repeat(0.5,len(states)),np.repeat(0.5,len(states))], index=states, columns=['norm

pi_50
```

dataframe만으로 pi_50을 선언한 인원도 있었습니다.

1. Iterative estimation of state-value function for given policy (pi_speed)

```
R=np.hstack((np.repeat(-1.5,4),-0.5,np.repeat(-1.5,2),0)).reshape(-1,1)
states=np.arange(0,70+10,step=10)

P=np.matrix([[.1,0,.9,0,0,0,0,0],
             [.1,0,0,.9,0,0,0,0],
             [0,.1,0,0,.9,0,0,0],
             [0,0,.1,0,0,.9,0,0],
             [0,0,0,.1,0,0,.9,0],
             [0,0,0,0,.1,0,0,.9],
             [0,0,0,0,0,.1,0,.9],
             [0,0,0,0,0,0,0,1]])

P=pd.DataFrame(P,columns=states)

R

## array([[-1.5],
##        [-1.5],
##        [-1.5],
##        [-1.5],
##        [-0.5],
##        [-1.5],
##        [-1.5],
##        [ 0. ]])
```

```
gamma=1.0
epsilon=10**(-8)


v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R+np.dot(gamma*P,v_old)

while np.max(np.abs(v_new-v_old))>epsilon:
    v_old=v_new
    v_new=R+np.dot(gamma*P, v_old)

print(v_new.T)

## [[-5.80592905 -5.2087811  -4.13926239 -3.47576467 -2.35376031 -1.73537603
##   -1.6735376   0.        ]]
```

2. Rewritten with intermediate saving

```
R=np.hstack((np.repeat(-1.5,4),-0.5,np.repeat(-1.5,2),0)).reshape(-1,1)
states=np.arange(0,70+10,step=10)

P=np.matrix([[.1,0,.9,0,0,0,0,0],
             [.1,0,0,.9,0,0,0,0],
             [0,.1,0,0,.9,0,0,0],
             [0,0,.1,0,0,.9,0,0],
             [0,0,0,.1,0,0,.9,0],
             [0,0,0,0,.1,0,0,.9],
             [0,0,0,0,0,.1,0,.9],
             [0,0,0,0,0,0,0,1]])

P=pd.DataFrame(P,columns=states)

gamma=1.0
epsilon=10**(-8)

v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R+np.dot(gamma*P,v_old)

results=v_old.T
results=np.vstack((results,v_new.T))

while np.max(np.abs(v_new-v_old)) > epsilon:
    v_old=v_new
    v_new=R+np.dot(gamma*P, v_old)
    results=np.vstack((results,v_new.T))

results=pd.DataFrame(results, columns=states)
```

```
results.head()
```

```
##        0       10       20      30      40       50      60   70
## 0  0.000  0.0000  0.0000  0.000  0.000  0.0000  0.000  0.0
## 1 -1.500 -1.5000 -1.5000 -1.500 -0.500 -1.5000 -1.500  0.0
## 2 -3.000 -3.0000 -2.1000 -3.000 -2.000 -1.5500 -1.650  0.0
## 3 -3.690 -4.5000 -3.6000 -3.105 -2.285 -1.7000 -1.655  0.0
## 4 -5.109 -4.6635 -4.0065 -3.390 -2.300 -1.7285 -1.670  0.0
```

```
results.tail()
```

```
##            0         10         20         30        40         50         60   70
## 18 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 19 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 20 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 21 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 22 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
```

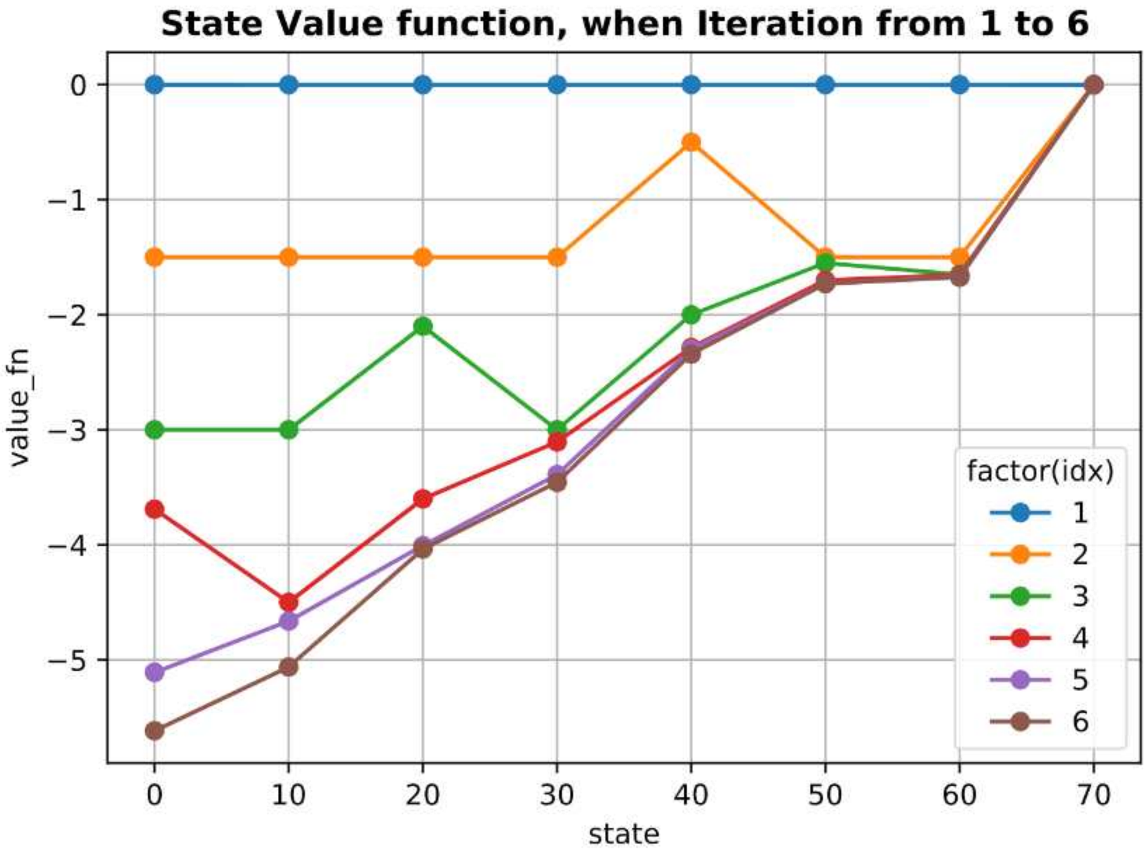
3. Plot(Iteration from 1 to 6 / Iteration from 7 to 12 / Iteration from 13 to 18)

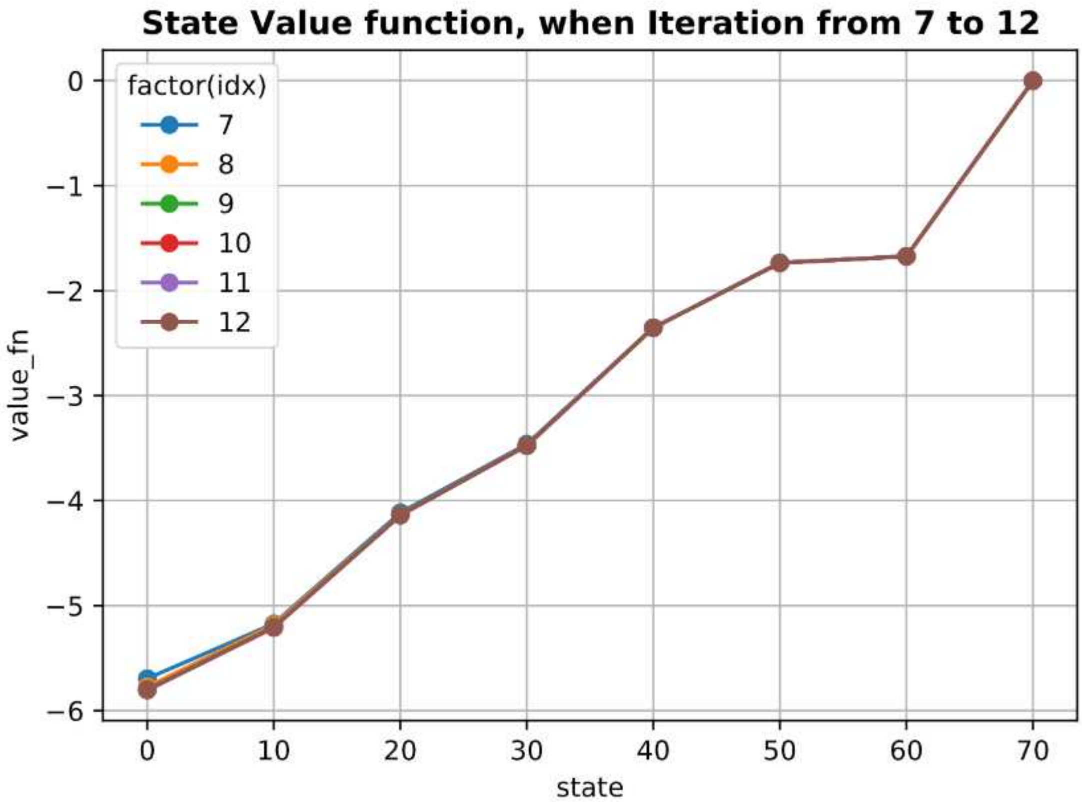
```
for i in range(6,12):
    plt.plot(states, result.iloc[i], marker='o', label=str(i+1))
    plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 7 to 12',fontweight='bold')
plt.show()
```

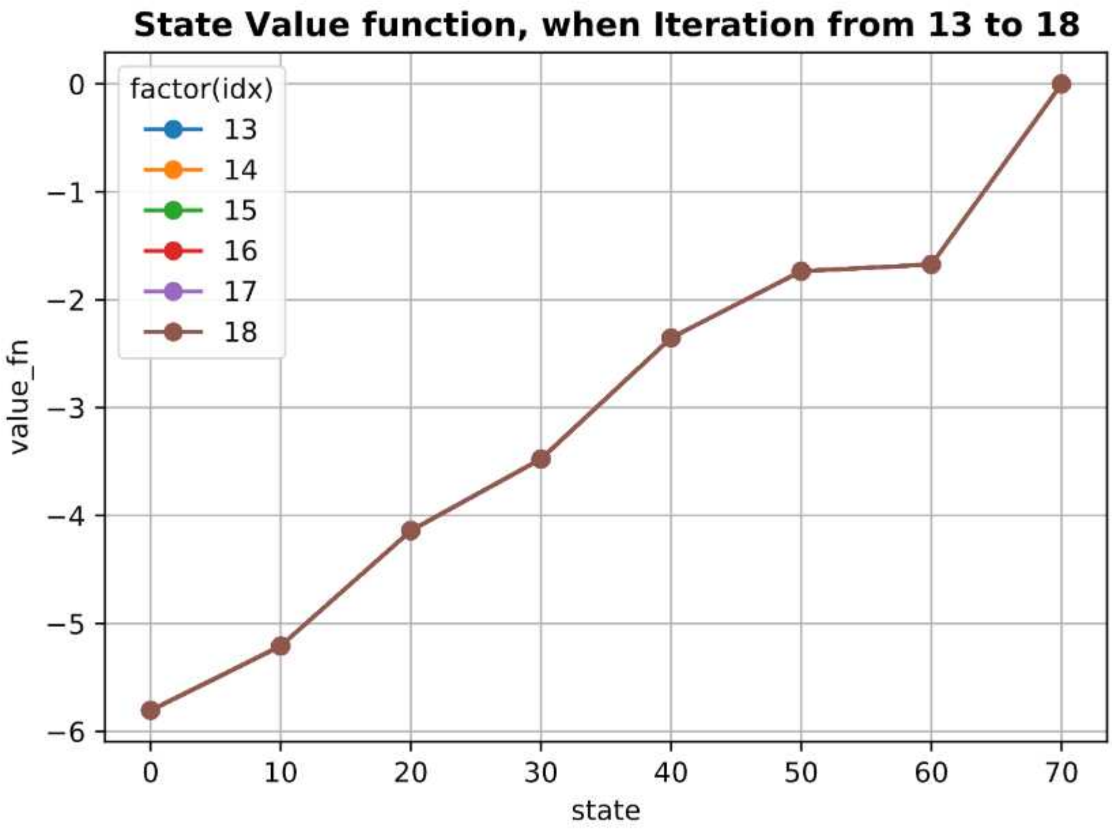
#for문을 활용한 방식

```
plt.plot(states,results.iloc[0],marker='o',label='1')
plt.plot(states,results.iloc[1],marker='o',label='2')
plt.plot(states,results.iloc[2],marker='o',label='3')
plt.plot(states,results.iloc[3],marker='o',label='4')
plt.plot(states,results.iloc[4],marker='o',label='5')
plt.plot(states,results.iloc[5],marker='o',label='6')
plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 1 to 6',fontweight='bold')
plt.show()
```

#모든 선언을 직접 작성한 형식


State Value function, when Iteration from 1 to 6
value_fn
state
factor(idx)
1
2
3
4
5
6
0
−1
−2
−3
−4
−5
0
10
20
30
40
50
60
70


State Value function, when Iteration from 7 to 12
value_fn
state
factor(idx)
7
8
9
10
11
12
0
−1
−2
−3
−4
−5
−6
0
10
20
30
40
50
60
70


State Value function, when Iteration from 13 to 18
factor(idx)
13
14
15
16
17
18
value_fn
state
0
−1
−2
−3
−4
−5
−6
0
10
20
30
40
50
60
70

4. Pi (S->A)

```
states = np.array(range(0,80,10)).astype(str)
pi_speed=np.c_[np.repeat(0,len(states)),np.repeat(1,len(states))]
pi_speed=pd.DataFrame(pi_speed, columns=['normal','speed'],index=states)

pi_speed
```

```
##     normal  speed
## 0        0      1
## 10       0      1
## 20       0      1
## 30       0      1
## 40       0      1
## 50       0      1
## 60       0      1
## 70       0      1
```

5. S->R

```python
def reward_fn(given_pi):

    R_s_a=pd.DataFrame(
        np.array([[-1, -1, -1, -1,0, -1, -1, 0],
          [-1.5, -1.5, -1.5,-1.5, -0.5, -1.5, -1.5, 0]]).T,columns=['normal','speed'],index=states)

    R_pi=np.sum(R_s_a*given_pi,axis=1)

    return R_pi

reward_fn(pi_speed).values
```

```
## array([[-1.5],
##        [-1.5],
##        [-1.5],
##        [-1.5],
##        [-0.5],
##        [-1.5],
##        [-1.5],
##        [ 0. ]])
```

6. S*A->S (해당부분은 가장먼저 언급한 선언방식에 차이가 있는 코드입니다.)

7. Test (대부분의 학생이 동일하였고, 단순 결과값 도출 선언문이었습니다.)

```
pi_speed
```

```
##      normal  speed
## 0         0      1
## 10        0      1
## 20        0      1
## 30        0      1
## 40        0      1
## 50        0      1
## 60        0      1
## 70        0      1
```

```
transition(pi_speed, states=states, P_normal=P_normal, P_speed=P_speed)
```

```
##        0   10   20   30   40   50   60   70
## 0    0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10   0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20   0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30   0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40   0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50   0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60   0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70   0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

8. Final implementation

```python
def policy_eval(given_pi):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_normal=P_normal, P_speed=P_speed)

    gamma=1.0
    epsilon=10**(-8)

    v_old=np.repeat(0,8).reshape(8,1)
    v_new=R+np.dot(gamma*P, v_old)

    while np.max(np.abs(v_new-v_old))>epsilon:
        v_old=v_new
        v_new=R+np.dot(gamma*P,v_old)

    return v_new.T
```

```python
policy_eval(pi_speed)
```

```
## array([[-5.80592905, -5.2087811 , -4.13926239, -3.47576467, -2.35376031,
##          -1.73537603, -1.6735376 ,  0.        ]])
```

```python
policy_eval(pi_50)
```

```
## array([[-5.96923786, -5.13359222, -4.11995525, -3.38922824, -2.04147003,
##          -2.02776769, -1.35138838,  0.        ]])
```