```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import warnings; warnings.simplefilter('ignore')
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
```

```
sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

```
Patients_df·=·pd.read_csv('Patients.csv')
```

```
from sklearn import preprocessing
label·=·preprocessing.LabelEncoder()·
·
Patients_df['Gender']=·label.fit_transform(Patients_df['Gender'])·
print(Patients_df['Gender'].unique())
```

```
    [1 0]
```

```
label·=·preprocessing.LabelEncoder()·
·
Patients_df['Location']=·label.fit_transform(Patients_df['Location'])·
print(Patients_df['Location'].unique())
```

```
    [0 2 1]
```

```
label = preprocessing.LabelEncoder()

Patients_df['SelfAssessedHealthStatus']= label.fit_transform(Patients_df['SelfAsse
print(Patients_df['SelfAssessedHealthStatus'].unique())
```

```
    [0 1 2 3]
```

```
from sklearn.preprocessing import OneHotEncoder
Gender = ['Female', 'Male']
Location·=·['County·General·Hospital',·'VA·Hospital',·"St.·Mary's·Medical·Center"]
SelfAssessedHealthStatus·=·['Fair',·'Good',·'Excellent','Poor']
enc = preprocessing.OneHotEncoder(categories=[Gender,Location,SelfAssessedHealthSt
X = [['Male', 'County General Hospital', 'Fair'], ['Female', 'VA Hospital', 'Good'
enc.fit(X)
OneHotEncoder(categories=[['Female', 'Male'],
                          ['County General Hospital', 'VA Hospital', "St. Mary's M
                          ['Fair', 'Good', 'Excellent']])
enc.transform([['Female', 'VA Hospital', 'Good']]).toarray()
```

```
    array([[1., 0., 0., 1., 0., 0., 1., 0., 0.]])
```

```
del·Patients_df['Diastolic']
```

```
del Patients_df['LastName']
```

```
enc = preprocessing.OneHotEncoder()
enc.fit(Patients_df[['Location']])
enc.categories_
one_hot = enc.transform(Patients_df[['Location']]).toarray()
one_hot
Patients_df[['County General Hospital', 'VA Hospital', "St. Mary's Medical Center"
```

```
enc = preprocessing.OneHotEncoder()
enc.fit(Patients_df[['SelfAssessedHealthStatus']])
enc.categories_
one_hot = enc.transform(Patients_df[['SelfAssessedHealthStatus']]).toarray()
one_hot
Patients_df[['Excellent', 'Fair', 'Good', 'Poor']] = one_hot
```

```
Patients_df
```

	Age	Gender	Height	Location	SelfAssessedHealthStatus	Smoker	Systolic
0	38	1	71	0	0	1	124
1	43	1	69	2	1	0	109
2	38	0	64	1	2	0	125
3	40	0	67	2	1	0	117
4	49	0	64	0	2	0	122
...	...	...	...	...	...	...	...
95	25	1	69	0	2	1	128
96	44	1	69	2	2	1	124
97	49	1	70	0	1	0	119
98	45	1	68	0	2	1	136
99	48	1	66	0	1	0	114

100 rows × 8 columns

```
# Create inputs and targets

inputs, targets = Patients_df[['Age', 'Weight', 'Smoker','Gender', 'County General

# Create and train the model
model = LinearRegression().fit(inputs, targets)


# Generate predictions
predictions = model.predict(inputs)

# Compute loss to evalute the model
loss = rmse(targets, predictions)
print('Loss:', loss)
```

```
from sklearn.preprocessing import StandardScaler
```

```
numeric_cols = ['Age', 'Weight', 'Height']
```

```
scaler = StandardScaler()
scaler.fit(Patients_df[numeric_cols])
scaled_inputs = scaler.transform(Patients_df[numeric_cols])
scaled_inputs
```

```
array([[-0.0390013 ,  0.83212833,  1.39250555],
       [ 0.65745049,  0.34041613,  0.68385132],
       [-0.0390013 , -0.86995234, -1.08778423],
       [ 0.23957941, -0.79430431, -0.0248029 ],
       [ 1.49319263, -1.32384052, -1.08778423],
       [ 1.07532156, -0.45388818,  0.32952421],
       [-0.73545309, -0.45388818, -1.08778423],
       [ 0.23957941,  0.98342439,  0.32952421],
       [-1.43190487,  1.09689643,  0.32952421],
       [-1.0140338 , -0.83212833, -0.37913001],
       [ 0.9360312 , -0.98342439,  0.32952421],
       [ 0.51816013, -0.64300825, -0.37913001],
       [-1.84977594,  0.7564803 ,  1.39250555],
       [ 0.10028906,  1.81555271,  1.74683266],
       [-0.31758201, -0.94560037, -0.73345712],
       [ 1.35390227,  1.0212484 ,  1.39250555],
       [-0.87474344,  1.39948855,  0.68385132],
       [-1.57119523, -0.86995234,  0.68385132],
       [-0.17829166,  0.94560037,  1.03817844],
       [ 1.63248299,  0.68083227,  0.32952421],
       [ 1.35390227, -0.79430431, -0.73345712],
       [ 0.10028906, -1.39948855, -1.08778423],
       [ 0.37886977, -0.64300825, -1.79643845],
       [ 0.79674084, -0.30259212, -0.37913001],
       [-1.43190487, -1.17254446, -0.73345712],
       [-1.84977594,  1.32384052,  1.03817844],
       [ 0.10028906, -0.41606416, -1.44211134],
       [-1.84977594, -1.51296059, -1.44211134],
       [-0.31758201,  0.45388818,  0.32952421],
       [-1.15332416,  1.21036847, -0.0248029 ],
       [ 0.9360312 , -1.05907242,  1.03817844],
       [ 0.23957941, -0.64300825, -0.37913001],
       [-1.84977594, -0.60518424, -1.08778423],
       [ 1.21461191,  1.24819249,  1.03817844],
       [ 0.79674084,  1.47513658,  1.39250555],
       [ 1.35390227, -0.64300825, -0.37913001],
       [ 0.79674084,  1.43731256,  1.39250555],
       [-0.45687237, -1.36166453, -0.37913001],
       [-0.73545309,  0.98342439, -0.37913001],
       [-0.0390013 , -0.98342439, -1.44211134],
       [ 0.10028906,  0.37824015,  1.39250555],
       [ 0.79674084,  1.09689643,  0.68385132],
       [ 0.79674084,  0.56736022,  1.03817844],
       [-0.17829166,  1.51296059,  1.03817844],
       [ 0.9360312 ,  0.68083227, -0.0248029 ],
       [-0.17829166, -0.71865628, -0.73345712],
       [-0.17829166, -0.71865628, -0.73345712],
       [-1.15332416,  1.05907242,  0.32952421],
       [ 0.10028906, -1.24819249, -1.79643845],
       [ 0.51816013,  0.15129606,  1.03817844],
       [ 0.51816013,  0.94560037, -0.0248029 ],
       [ 1.49319263,  0.60518424,  0.32952421],
       [ 0.79674084, -0.68083227, -1.79643845],
       [ 0.65745049, -0.71865628, -1.08778423],
       [ 1.21461191, -0.2647681 , -0.37913001],
       [ 1.63248299,  1.21036847,  1.74683266],
       [-0.0390013 , -1.13472045, -1.44211134],
       [ 0.37886977, -0.7564803 , -0.37913001],
       [ 0.9360312 ,  0.60518424,  1.03817844],
       [-0.31758201,  0.98342439,  1.39250555],
       [ 0.0390013 ,  0.90777636,  0.32952421]
```

```
cat_cols = ['Gender','Smoker','County General Hospital','VA Hospital',"St. Mary's 
categorical_data = Patients_df[cat_cols].values
```

```
from sklearn.model_selection import train_test_split
```

```
inputs_train, inputs_test, targets_train, targets_test = train_test_split(inputs,
```

```
# Create and train the model
model = LinearRegression().fit(inputs_train, targets_train)

# Generate predictions
predictions_test = model.predict(inputs_test)

# Compute loss to evalute the model
loss = rmse(targets_test, predictions_test)
print('Test Loss:', loss)
```

```
# Generate predictions
predictions_train = model.predict(inputs_train)

# Compute loss to evalute the model
loss = rmse(targets_train, predictions_train)
print('Training Loss:', loss)
```

```
Patients_df
```

	Age	Gender	Height	Location	SelfAssessedHealthStatus	Smoker	Systolic
0	38	1	71	0	0	1	124
1	43	1	69	2	1	0	109
2	38	0	64	1	2	0	125
3	40	0	67	2	1	0	117
4	49	0	64	0	2	0	122
...	...	...	...	...	...	...	...
95	25	1	69	0	2	1	128
96	44	1	69	2	2	1	124
97	49	1	70	0	1	0	119
98	45	1	68	0	2	1	136
99	48	1	66	0	1	0	114

100 rows × 8 columns

```
Patients_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Age                          100 non-null    int64  
 1   Gender                       100 non-null    int64  
 2   Height                       100 non-null    int64  
 3   Location                     100 non-null    int64  
 4   SelfAssessedHealthStatus     100 non-null    int64  
 5   Smoker                       100 non-null    int64  
 6   Systolic                     100 non-null    int64  
 7   Weight                       100 non-null    int64  
 8   County General Hospital      100 non-null    float64
 9   VA Hospital                  100 non-null    float64
 10  St. Mary's Medical Center    100 non-null    float64
 11  Excellent                    100 non-null    float64
 12  Fair                         100 non-null    float64
 13  Good                         100 non-null    float64
 14  Poor                         100 non-null    float64
dtypes: float64(7), int64(8)
memory usage: 11.8 KB
```

```
data = {'Location' : ['County General Hospital', 'VA Hospital', "St. Mary's Medica
        }
df = pd.DataFrame(data,columns=['Location'])

print(df)
print(df.dtypes)
```

```
                    Location
0    County General Hospital
1                VA Hospital
2  St. Mary's Medical Center
Location    object
dtype: object
```

```
'SelfAssessedHealthStatus' : ['Fair', 'Good', 'Excellent','Poor']
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
Patients_df[['Age', 'Height',
     'Weight']] = scaler.fit_transform(Patients_df[['Age','Height',
      'Weight']])
```

```
Patients_df.head(5)
```

	Age	Gender	Height	Location	SelfAssessedHealthStatus	Smoker	Systc
0	-0.039001	1	1.392506	0	0	1	
1	0.657450	1	0.683851	2	1	0	
2	-0.039001	0	-1.087784	1	2	0	
3	0.239579	0	-0.024803	2	1	0	
4	1.493193	0	-1.087784	0	2	0	

```
X = Patients_df.drop(['Systolic'], axis=1)
y = Patients_df['Systolic']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_s
```

```
from sklearn.linear_model import Lasso

reg = Lasso(alpha=0.5)
reg.fit(X_train, y_train)
```

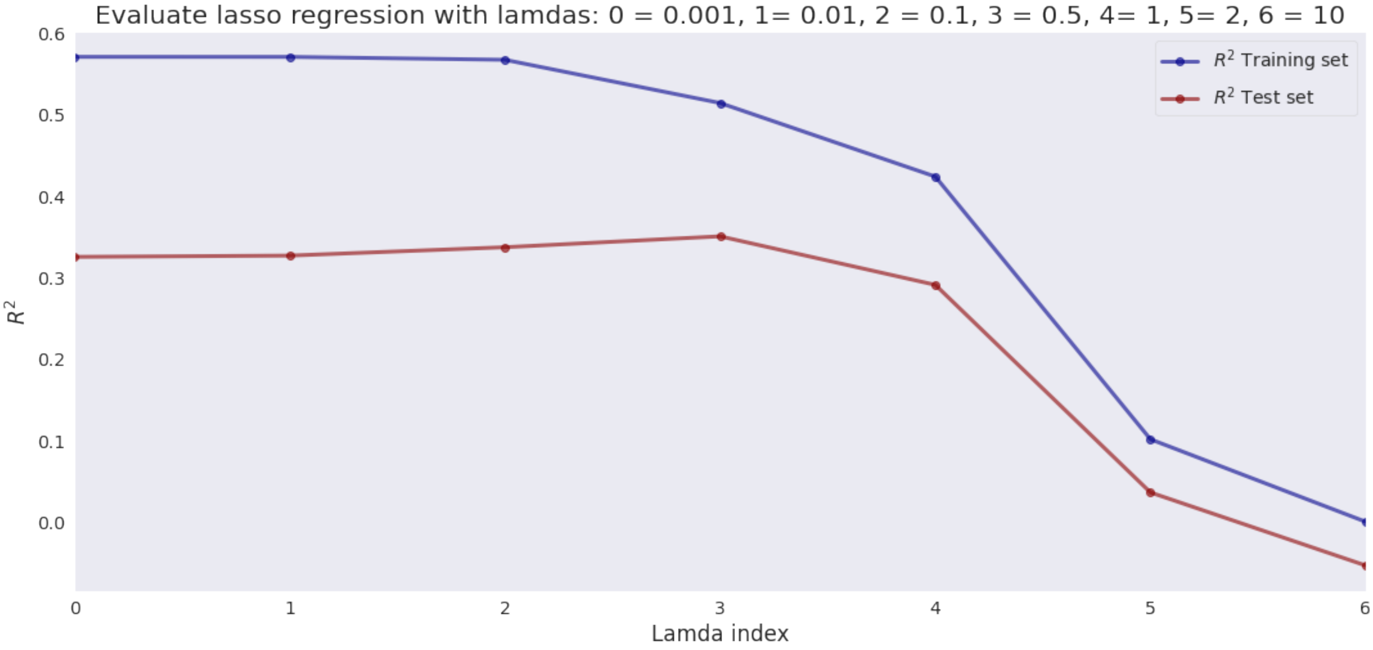
```
    Lasso(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=1000,
          normalize=False, positive=False, precompute=False, random_state=None,
          selection='cyclic', tol=0.0001, warm_start=False)
```

```
print('Lasso Regression: R^2 score on training set', reg.score(X_train, y_train)*1
print('Lasso Regression: R^2 score on test set', reg.score(X_test, y_test)*100)
```

```
Lasso Regression: R^2 score on training set 51.40426808603669
Lasso Regression: R^2 score on test set 35.038955672806814
```

```
lambdas = (0.001, 0.01, 0.1, 0.5, 1, 2, 10)
l_num = 7
pred_num = X.shape[1]

# prepare data for enumerate
coeff_a = np.zeros((l_num, pred_num))
train_r_squared = np.zeros(l_num)
test_r_squared = np.zeros(l_num)


for ind, i in enumerate(lambdas):
    reg = Lasso(alpha = i)
    reg.fit(X_train, y_train)

    coeff_a[ind,:] = reg.coef_
    train_r_squared[ind] = reg.score(X_train, y_train)
    test_r_squared[ind] = reg.score(X_test, y_test)
```

```
plt.figure(figsize=(18, 8))
plt.plot(train_r_squared, 'bo-', label=r'$R^2$ Training set', color="darkblue", al
plt.plot(test_r_squared, 'bo-', label=r'$R^2$ Test set', color="darkred", alpha=0.
plt.xlabel('Lamda index'); plt.ylabel(r'$R^2$')
plt.xlim(0, 6)
plt.title(r'Evaluate lasso regression with lamdas: 0 = 0.001, 1= 0.01, 2 = 0.1, 3
plt.legend(loc='best')
plt.grid()
```



```
df_lam = pd.DataFrame(test_r_squared*100, columns=['R_squared'])
df_lam['lambda'] = (lambdas)
# returns the index of the row where column has maximum value.
df_lam.loc[df_lam['R_squared'].idxmax()]
```

```
    R_squared    35.038956
    lambda        0.500000
    Name: 3, dtype: float64
```

```
reg_best = Lasso(alpha = 0.1)
reg_best.fit(X_train, y_train)
reg_best.coef_
```

```
    array([ 1.14804079, -0.        ,  1.04777052, -0.13619896,  0.45499808,
            8.75319922, -0.9764929 ])
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, reg_best.predict(X_test))
```

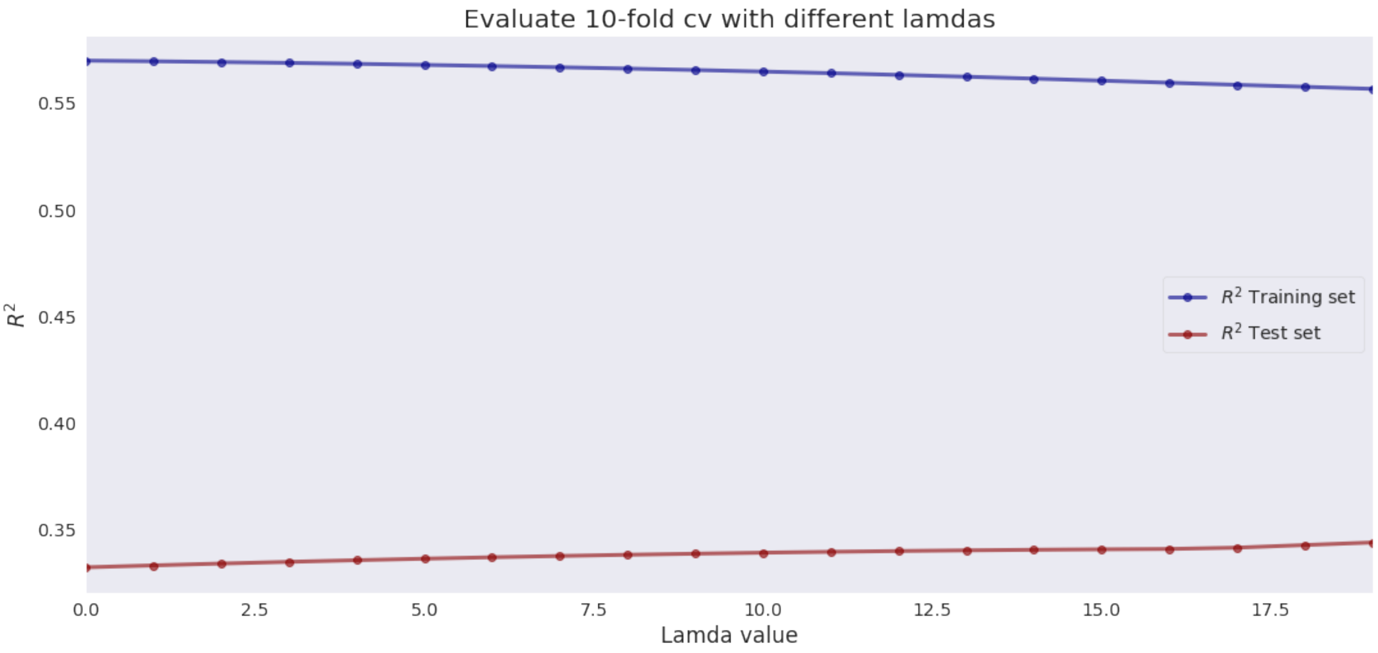
```
    32.67545716126301
```

```
l_min = 0.05
l_max = 0.2
l_num = 20
lambdas = np.linspace(l_min,l_max, l_num)

train_r_squared = np.zeros(l_num)
test_r_squared = np.zeros(l_num)

pred_num = X.shape[1]
coeff_a = np.zeros((l_num, pred_num))
```

```
from sklearn.model_selection import cross_val_score
```

```
for ind, i in enumerate(lambdas):
    reg = Lasso(alpha = i)
    reg.fit(X_train, y_train)
    results = cross_val_score(reg, X, y, cv=10, scoring="r2")

    train_r_squared[ind] = reg.score(X_train, y_train)
    test_r_squared[ind] = reg.score(X_test, y_test)
```

```
# Plotting
plt.figure(figsize=(18, 8))
plt.plot(train_r_squared, 'bo-', label=r'$R^2$ Training set', color="darkblue", al
plt.plot(test_r_squared, 'bo-', label=r'$R^2$ Test set', color="darkred", alpha=0.
plt.xlabel('Lamda value'); plt.ylabel(r'$R^2$')
plt.xlim(0, 19)
plt.title(r'Evaluate 10-fold cv with different lamdas')
plt.legend(loc='best')
plt.grid()
```



```
df_lam = pd.DataFrame(test_r_squared*100, columns=['R_squared'])
df_lam['lambda'] = (lambdas)
# returns the index of the row where column has maximum value.
df_lam.loc[df_lam['R_squared'].idxmax()]
```

```
R_squared    34.376749
lambda        0.200000
Name: 19, dtype: float64
```

```
# Best Model
reg_best = Lasso(alpha = 0.144737)
reg_best.fit(X_train, y_train)
```

```
Lasso(alpha=0.144737, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, reg_best.predict(X_test))
```

```
32.54105584096965
```

```
reg_best.coef_
```

```
array([ 1.11565225, -0.        ,  0.87507912, -0.06084871,  0.41829045,
        8.571702  , -0.78587678])
```

✓ 0s completed at 1:21 PM ● ×