Machine Learning Model for Predicting a Ship's Crew Size

Laurent

September 30, 2021

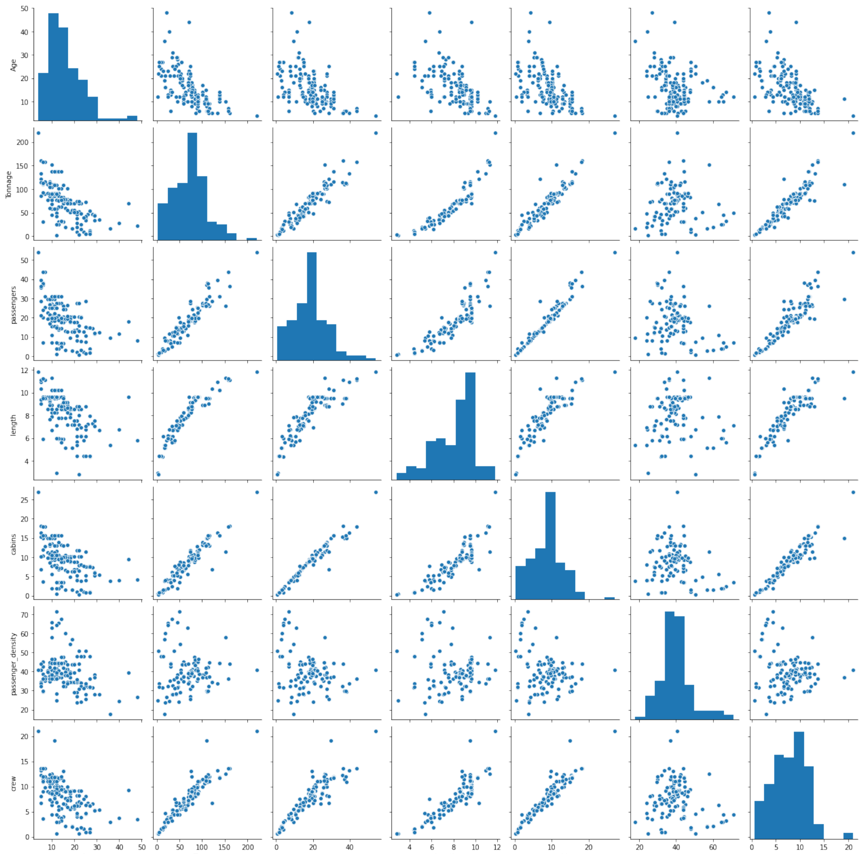
1 Basic statistics

```
df=pd.read_csv("./cruise_ship_info.csv")
df.describe()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipykernel_3012107/3000665307.py in <module>
----> 1 df=pd.read_csv("./cruise_ship_info.csv")
      2 df.describe()

~/.pyenv/versions/my/lib/python3.8/site-packages/pandas/io/parsers.py in read_csv(filep
    686     )
    687
--> 688     return _read(filepath_or_buffer, kwds)
    689
    690

~/.pyenv/versions/my/lib/python3.8/site-packages/pandas/io/parsers.py in _read(filepat
    452
    453     # Create the parser.
--> 454     parser = TextFileReader(fp_or_buf, **kwds)
    455
    456     if chunksize or iterator:

~/.pyenv/versions/my/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self
    946             self.options["has_index_names"] = kwds["has_index_names"]
```

```
    947
--> 948         self._make_engine(self.engine)
    949
    950     def close(self):

~/.pyenv/versions/my/lib/python3.8/site-packages/pandas/io/parsers.py in _make_engine(
   1178     def _make_engine(self, engine="c"):
   1179         if engine == "c":
-> 1180             self._engine = CParserWrapper(self.f, **self.options)
   1181         else:
   1182             if engine == "python":

~/.pyenv/versions/my/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self
   2008         kwds["usecols"] = self.usecols
   2009
-> 2010         self._reader = parsers.TextReader(src, **kwds)
   2011         self.unnamed_cols = self._reader.unnamed_cols
   2012

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] No such file or directory: './cruise_ship_info.csv'
```

```python
cols = ['Age', 'Tonnage', 'passengers', 'length',
   'cabins','passenger_density','crew']
_ = sns.pairplot(df[cols])
```



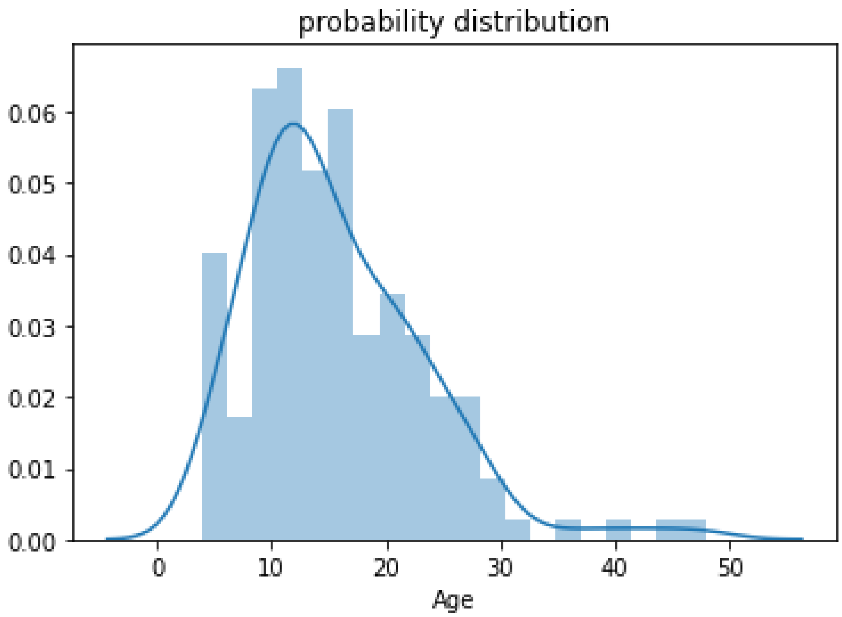
2 Observations

1. We observe that variables are on different scales, for sample the Age variable ranges from about 16 years to 48 years, while the Tonnage variable ranges from 2 to 220, see probability density plots below. It is therefore important that when a regression model is built using these variables, variables be brought to same scale either by standardizing or normalizing the data.

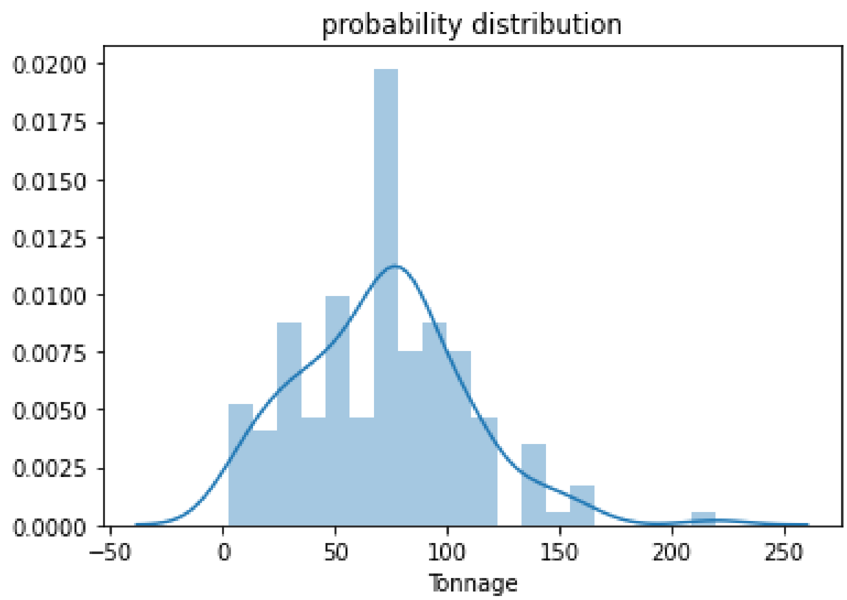
2. We also observe that the target variable 'crew' correlates well with 4 predictor variables, namely, 'Tonnage', 'passengers', 'length', and 'cabins'.

```
sns.distplot(df['Age'],bins=20)
```

```
plt.title('probability distribution')
plt.show()
```



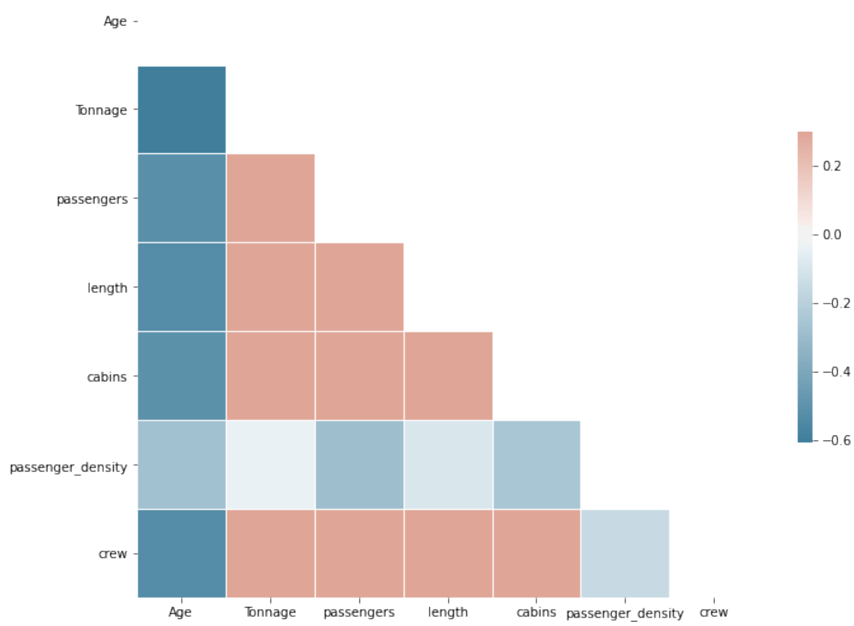

```
sns.distplot(df['Tonnage'],bins=20)
plt.title('probability distribution')
plt.show()
```



3 Variable selection for predicting "crew" size

3.1 Calculation of covariance matrix

```
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect
    ratio
_ = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
    center=0,
            square=True, linewidths=.5,
               cbar_kws={"shrink": .5})
```



3.2 Selecting important variables

From the covariance matrix plot above, we see that the "crew" variable correlates strongly with 4 predictor variables: "Tonnage", "passengers", "length, and "cabins".

```
cols_selected = ['Tonnage', 'passengers', 'length',
    'cabins','crew']
df[cols_selected].head()
```

	Tonnage	passengers	length	cabins	crew
0	30.277	6.94	5.94	3.55	3.55
1	30.277	6.94	5.94	3.55	3.55
2	47.262	14.86	7.22	7.43	6.7
3	110	29.74	9.53	14.88	19.1
4	101.353	26.42	8.92	13.21	10

```
X = df[cols_selected].iloc[:,0:4].values     #
    features matrix
y = df[cols_selected]['crew'].values         # target
    variable
```

4 Data partitioning into training and testing sets

In order to build a simplified regression model, we shall focus only on ordinal features. The categorical features “Ship_name” and “Cruise_line” will not be used. A simple model built using only the 4 ordinal features “Tonnage”, “passengers”, “length, and ”cabins“ will be simple to interpret.

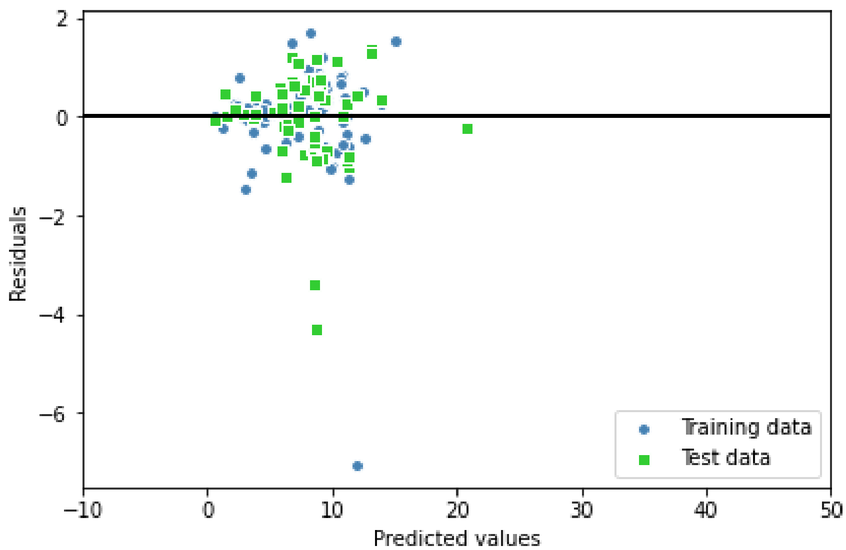
```
from sklearn.model_selection import train_test_split
X = df[cols_selected].iloc[:,0:4].values
y = df[cols_selected]['crew']
X_train, X_test, y_train, y_test = train_test_split(
   X,
                                                                    y,
                                                                    test_size=0.4,
                                                                       random_state=0)
```

5 Building a linear regression model

```
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
```

```
plt.scatter(y_train_pred,  y_train_pred - y_train,
            c='steelblue', marker='o',
               edgecolor='white',
            label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test,
            c='limegreen', marker='s',
               edgecolor='white',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, color='black',
   lw=2)
plt.xlim([-10, 50])
```

```
plt.tight_layout()
plt.legend(loc='lower right')
plt.show()
```



6 Evaluation of regression model

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 0.955, test: 0.889
R^2 train: 0.920, test: 0.928
```

7 Regression coefficients

```
slr.fit(X_train, y_train).intercept_
```

```
-0.7525074496158393
```

```
slr.fit(X_train, y_train).coef_
```

```
array([ 0.01902703, -0.15001099,  0.37876395,  0.77613801])
```

8 Feature Standardization, Cross Validation, and Hyper-parameter Tuning

```
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
X = df[cols_selected].iloc[:,0:4].values
y = df[cols_selected]['crew']
from sklearn.preprocessing import StandardScaler
sc_y = StandardScaler()
sc_x = StandardScaler()
y_std = sc_y.fit_transform(y_train[:,
   np.newaxis]).flatten()
```

```
/tmp/ipykernel_3012107/1485608700.py:8: FutureWarning: Support for multi-dimensional i
  y_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
```

```
train_score = []
test_score = []

for i in range(10):
    X_train, X_test, y_train, y_test =
       train_test_split( X, y, test_size=0.4,
       random_state=i)
    y_train_std = sc_y.fit_transform(y_train[:,
       np.newaxis]).flatten()
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.linear_model import LinearRegression
```

```
    from sklearn.pipeline import Pipeline
    pipe_lr = Pipeline([('scl',
       StandardScaler()),('pca',
       PCA(n_components=4)),('slr',
       LinearRegression())])
    pipe_lr.fit(X_train, y_train_std)
    y_train_pred_std=pipe_lr.predict(X_train)
    y_test_pred_std=pipe_lr.predict(X_test)
    y_train_pred=sc_y.inverse_transform(y_train_pred_std)
    y_test_pred=sc_y.inverse_transform(y_test_pred_std)
    train_score = np.append(train_score,
       r2_score(y_train, y_train_pred))
    test_score = np.append(test_score,
       r2_score(y_test, y_test_pred))
```

```
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/1780874285.py:6: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
```

```
train_score
```

```
array([0.92028261, 0.91733937, 0.94839385, 0.93899476, 0.90621451,
```

```
       0.91156903, 0.92726066, 0.94000795, 0.93922948, 0.93629554])
```

```
test_score
```

```
array([0.92827978, 0.93807946, 0.8741834 , 0.89901199, 0.94781315,
       0.91880183, 0.91437408, 0.89660876, 0.90427477, 0.90139208])
```

```
print('R2 train: %.3f +/- %.3f' %
   (np.mean(train_score),np.std(train_score)))
```

```
R2 train: 0.929 +/- 0.013
```

```
print('R2 test: %.3f +/- %.3f' %
   (np.mean(test_score),np.std(test_score)))
```

```
R2 test: 0.912 +/- 0.021
```

9 Techniques of Dimensionality Reduction

9.1 Principal Component Analysis (PCA)

```
train_score = []
test_score = []
cum_variance = []

for i in range(1,5):
    X_train, X_test, y_train, y_test =
       train_test_split( X, y, test_size=0.4,
       random_state=0)
    y_train_std = sc_y.fit_transform(y_train[:,
       np.newaxis]).flatten()
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.linear_model import LinearRegression
    from sklearn.pipeline import Pipeline
```

```
    pipe_lr = Pipeline([('scl',
       StandardScaler()),('pca',
       PCA(n_components=i)),('slr',
       LinearRegression())])
    pipe_lr.fit(X_train, y_train_std)
    y_train_pred_std=pipe_lr.predict(X_train)
    y_test_pred_std=pipe_lr.predict(X_test)
    y_train_pred=sc_y.inverse_transform(y_train_pred_std)
    y_test_pred=sc_y.inverse_transform(y_test_pred_std)
    train_score = np.append(train_score,
       r2_score(y_train, y_train_pred))
    test_score = np.append(test_score,
       r2_score(y_test, y_test_pred))
    cum_variance = np.append(cum_variance,
       np.sum(pipe_lr.fit(X_train,
       y_train).named_steps['pca'].explained_variance_ratio_))
```

```
/tmp/ipykernel_3012107/2598761563.py:7: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/2598761563.py:7: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/2598761563.py:7: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
/tmp/ipykernel_3012107/2598761563.py:7: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
```

```
train_score
```

```
array([0.90411898, 0.9041488 , 0.90416405, 0.92028261])
```

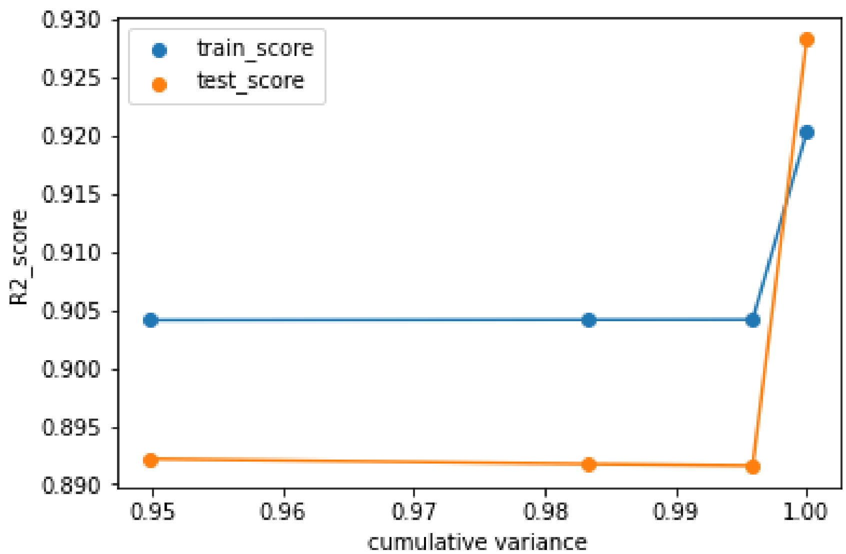
```
test_score
```

```
array([0.89217843, 0.89174896, 0.89159266, 0.92827978])
```

```
cum_score
```

[goto error]

```
plt.scatter(cum_variance,train_score, label =
    'train_score')
plt.plot(cum_variance, train_score)
plt.scatter(cum_variance,test_score, label =
    'test_score')
plt.plot(cum_variance, test_score)
plt.xlabel('cumulative variance')
plt.ylabel('R2_score')
plt.legend()
plt.show()
```



Observations (PCA)

We observe that by increasing the number of principal components from 1 to 4, the train and test scores improve. This is because with less components, there is high bias error in the model, since model is overly simplified. As we increase the number of principal components, the bias error will reduce, but complexity in the model increases.

9.2 Regularized Regression: Lasso

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

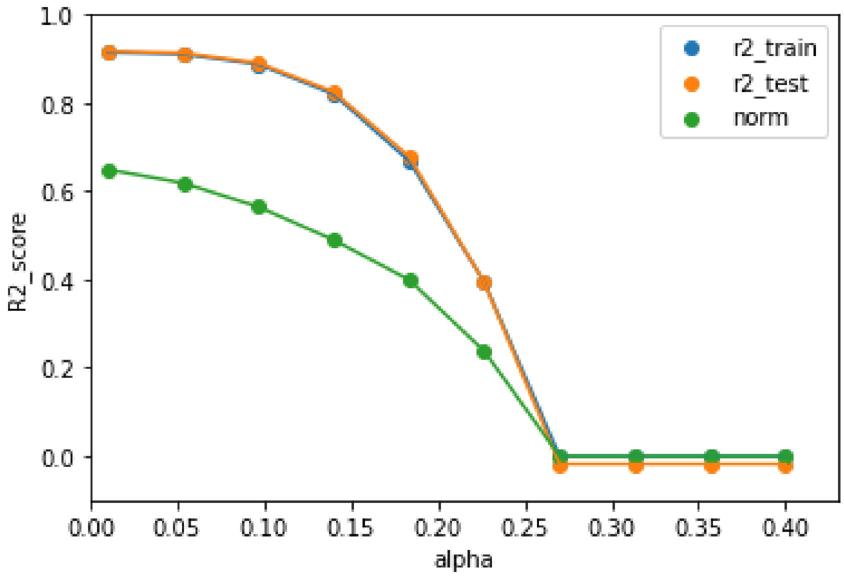
```
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.4, random_state=0)
y_train_std = sc_y.fit_transform(y_train[:,
   np.newaxis]).flatten()
X_train_std = sc_x.fit_transform(X_train)
X_test_std = sc_x.transform(X_test)
```

```
/tmp/ipykernel_3012107/3811144624.py:3: FutureWarning: Support for multi-dimensional i
  y_train_std = sc_y.fit_transform(y_train[:, np.newaxis]).flatten()
```

```
alpha = np.linspace(0.01,0.4,10)
```

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.7)

r2_train=[]
r2_test=[]
norm = []
for i in range(10):
    lasso = Lasso(alpha=alpha[i])
    lasso.fit(X_train_std,y_train_std)
    y_train_std=lasso.predict(X_train_std)
    y_test_std=lasso.predict(X_test_std)
    r2_train=np.append(r2_train,r2_score(y_train,sc_y.inverse_transform(
    r2_test=np.append(r2_test,r2_score(y_test,sc_y.inverse_transform(y_t
    norm= np.append(norm,np.linalg.norm(lasso.coef_))
```

```
plt.scatter(alpha,r2_train,label='r2_train')
plt.plot(alpha,r2_train)
plt.scatter(alpha,r2_test,label='r2_test')
plt.plot(alpha,r2_test)
plt.scatter(alpha,norm,label = 'norm')
plt.plot(alpha,norm)
plt.ylim(-0.1,1)
plt.xlim(0,.43)
plt.xlabel('alpha')
plt.ylabel('R2_score')
plt.legend()
plt.show()
```



Observations (Lasso)

We observe that as the regularization parameter α increases, the norm of the regression coefficients become smaller and smaller. This means more regression coefficients are forced to zero, which intend increases bias error (over simplification). The best value to balance bias-variance tradeoff is when α is kept low, say $\alpha = 0.1$ or less.