Regression Models

https://colab.research.google.com/drive/1-6vKW30ZDQ_o-ANfb_M8Hr1_pGC0oI-x?usp=sharing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_boston
from sklearn.pipeline import make_pipeline
```

```
#import weather
dfWeather = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/wea
dfWeather
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN
0	1	1989	9	0.000000	32.000000	32.000000	32.000000
1	1	1989	10	0.200000	32.000000	32.000000	32.000000
2	1	1989	11	0.150000	32.000000	32.000000	32.000000
3	1	1989	12	0.003226	32.000000	32.000000	32.000000
4	1	1990	1	0.258065	32.000000	32.000000	32.000000
...	...	...	...	...	...	...	...
792	3	2019	9	0.001083	74.125000	83.300000	66.341667
793	3	2019	10	0.000000	68.153226	80.419355	56.870968
794	3	2019	11	0.067333	61.308333	72.033333	51.791667
795	3	2019	12	0.117742	56.491935	64.790323	48.233871
796	3	2020	1	0.000000	55.750000	64.250000	45.000000

797 rows × 7 columns

```
#average yearly weather
dfWeather_year = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.cc
dfWeather_year
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN
0	1	1989	10.500000	0.088306	32.000000	32.000000	32.000000
1	1	1990	6.500000	0.111255	35.571913	39.544139	32.622426
2	1	1991	6.230769	0.156505	40.832258	50.658125	33.320761
3	1	1992	6.500000	0.144008	41.039145	50.311868	33.822216
4	1	1993	6.500000	0.167905	38.265216	47.550114	31.100369
...	...	...	...	...	...	...	...
66	3	2016	6.500000	0.024942	65.762837	75.240536	57.711488
67	3	2017	6.500000	0.028285	65.816868	75.397960	57.828031
68	3	2018	6.500000	0.018802	65.940230	75.249064	58.155413
69	3	2019	6.500000	0.044640	64.580618	73.677263	56.700291
70	3	2020	1.000000	0.000000	55.750000	64.250000	45.000000

71 rows × 7 columns

```
#average 3 month weather
dfWeather_3month = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.
```

```
#import Acres Burned
dfAcres = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/acres
dfAcres
```

region YEAR_ Month GIS_ACRES

```
#Classify each fire
# 1 is small 4 is large
def getFireClass(FireSize):
  if FireSize < .25:
    return 1
  elif (FireSize >= .25 and FireSize <100):
    return 1
  elif FireSize < 300:
    return 2
  elif FireSize < 1000:
    return 3
  elif FireSize < 5000:
    return 3
  else:
    return 4
dfAcres['Acres_Class'] = dfAcres['GIS_ACRES'].apply(getFireClass)
dfAcres
```

	region	YEAR_	Month	GIS_ACRES	Acres_Class
0	1	1909	5	59.738968	1
1	1	1909	7	4.978550	1
2	1	1910	7	147285.675293	4
3	1	1910	8	1819.406002	3
4	1	1910	9	1633.051666	3
...	...	...	...	...	...
2128	3	2019	7	2604.718104	3
2129	3	2019	8	625.997220	3
2130	3	2019	9	3958.133733	3
2131	3	2019	10	28948.924878	4
2132	3	2019	11	3301.654997	3

2133 rows × 5 columns

```
# import fires burned per month
dfFireCount = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/c
dfFireCount
```

	region	YEAR_	Month	OBJECTID
0	1	1909	5	1
1	1	1909	7	1
2	1	1910	7	3
3	1	1910	8	6
4	1	1910	9	5
...	...	...	...	...
2128	3	2019	7	10
2129	3	2019	8	10
2130	3	2019	9	12

```
# merge weather df and acres
WDF_Acres = pd.merge(dfWeather, dfAcres, how ='left', left_on=('YEAR','MONTH','regic
WDF_Acres = WDF_Acres.fillna(0)
WDF_Acres = WDF_Acres.drop(['YEAR_', 'Month'], axis=1)
WDF_Acres
```

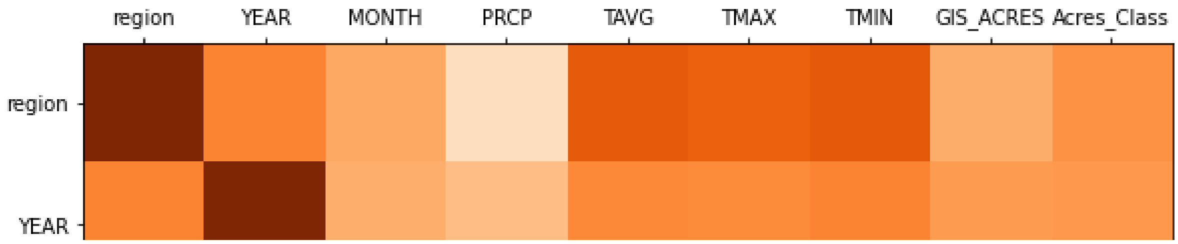
	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN	GIS_ACRES	Acres
0	1	1989	9	0.000000	32.000000	32.000000	32.000000	5306.863037	
1	1	1989	10	0.200000	32.000000	32.000000	32.000000	41.270145	
2	1	1989	11	0.150000	32.000000	32.000000	32.000000	121.872509	
3	1	1989	12	0.003226	32.000000	32.000000	32.000000	20.844431	
4	1	1990	1	0.258065	32.000000	32.000000	32.000000	0.000000	
...	...	...	...	...	...	...	...	...	
792	3	2019	9	0.001083	74.125000	83.300000	66.341667	3958.133733	
793	3	2019	10	0.000000	68.153226	80.419355	56.870968	28948.924878	
794	3	2019	11	0.067333	61.308333	72.033333	51.791667	3301.654997	
795	3	2019	12	0.117742	56.491935	64.790323	48.233871	0.000000	
796	3	2020	1	0.000000	55.750000	64.250000	45.000000	0.000000	

797 rows × 9 columns

```
# find correlation between weather parameters and acres burned
WDF_Acres.corr()['Acres_Class'].to_frame().sort_values('Acres_Class', ascending = Fa
```

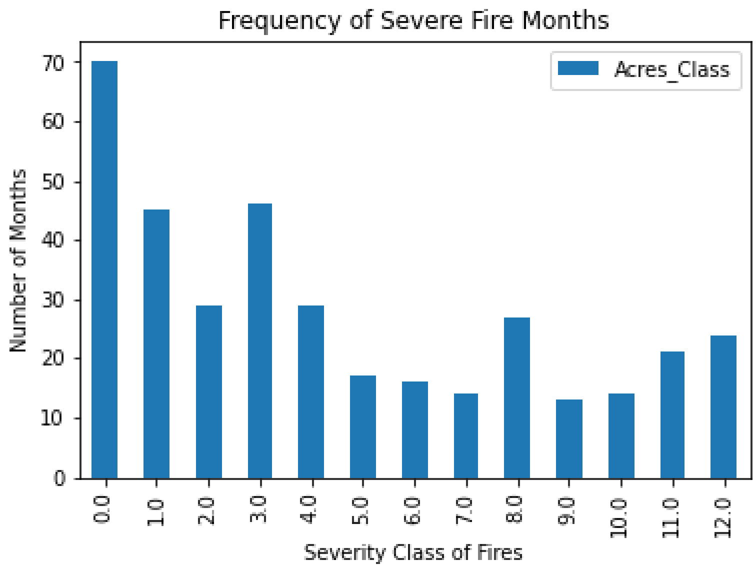
	Acres_Class
Acres_Class	1.000000
TMAX	0.637973
TAVG	0.632625
TMIN	0.626857
GIS_ACRES	0.322216
MONTH	0.258447
region	0.143570
YEAR	0.102958
PRCP	-0.530492

```
def plot_corr(df,size=10):
    corr = WDF_Acres.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    ax.matshow(corr,cmap=plt.cm.Oranges)
    plt.xticks(range(len(corr.columns)), corr.columns)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.show()
plot_corr(WDF_Acres)
```




```
#Plot count of classes including 0
#Combine months and years to get rid of regions
x = WDF_Acres.groupby(["YEAR", "MONTH"])["Acres_Class"].sum().to_frame()
x = x['Acres_Class'].value_counts().to_frame()
x.reset_index(inplace=True)
x= x.sort_values(by = 'index')
x.plot.bar(x = 'index', y = 'Acres_Class')
plt.ylabel("Number of Months")
plt.xlabel("Severity Class of Fires")
plt.title("Frequency of Severe Fire Months")
```

```
Text(0.5, 1.0, 'Frequency of Severe Fire Months')
```




```
#only region 1
WDF_Acres1 = WDF_Acres.loc[WDF_Acres['region'] == 1]
WDF_Acres1
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN	GIS_ACRES	Acres
0	1	1989	9	0.000000	32.000000	32.000000	32.000000	5306.863037	
1	1	1989	10	0.200000	32.000000	32.000000	32.000000	41.270145	
2	1	1989	11	0.150000	32.000000	32.000000	32.000000	121.872509	
3	1	1989	12	0.003226	32.000000	32.000000	32.000000	20.844431	
4	1	1990	1	0.258065	32.000000	32.000000	32.000000	0.000000	
...	...	...	...	...	...	...	...	...	
360	1	2019	9	0.034417	61.391667	72.591667	50.666667	79993.038281	

regression model

1. Predict the number of acres burned each month

```
# if negative correlation multiple column by -1
WDF_Acres['PRCP'] = WDF_Acres['PRCP']*(-1)
#normalize data
columns = ['TMAX', 'TMIN', 'TAVG','MONTH','PRCP', 'YEAR']
def normalize(df, columns):
  result = df[columns]
  result = (result - result.mean())/result.std()
  return result
weather_normlized = normalize(WDF_Acres, columns)
weather_normlized
```

	TMAX	TMIN	TAVG	MONTH	PRCP	YEAR

```
#split into test and training after weather normalized
# to get 80% train 20% test - split along 638 row, 292 if only using region 1
X_train = weather_normlized.iloc[:638]
Y_train = WDF_Acres['Acres_Class'].iloc[:638]
X_test = weather_normlized.iloc[638:]
Y_test = WDF_Acres['Acres_Class'].iloc[638:]
```

4	-1.951860	-0.895084	1.458965	-1.601860	1.473345	2.060698

```
#build the regression model
poly_model= make_pipeline(PolynomialFeatures(3), LinearRegression())
poly_model.fit(X_train,Y_train)
```

```
Pipeline(memory=None,
         steps=[('polynomialfeatures',
                 PolynomialFeatures(degree=3, include_bias=True,
                                    interaction_only=False, order='C')),
                ('linearregression',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

```
# average difference between the predicted num acres burned and the actual num acres
y_pred = poly_model.predict(X_test)
error = (abs(y_pred - Y_test)).sum()/len(y_pred)
error
```

```
1.0008113684156854
```

Use a KNeighborsRegressor Acres Burned Data

```
knn =KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train, Y_train)
predictions = knn.predict(X_test)
#computing the error
#get the actual value for the test set
#compute the average error of the prediction
error = (abs(predictions - Y_test)).sum() / len(predictions)
error
```

```
0.9773584905660376
```

Fire count data

```
DF_Count = pd.merge(dfWeather, dfFireCount, how ='left', left_on=('YEAR','MONTH','re
DF_Count = WDF_Count.fillna(0)
```

```
DF_Count = WDF_Count.fillna(0)
DF_Count = WDF_Count.drop(['YEAR_', 'Month'], axis=1)
DF_Count
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN	OBJECTID
0	1	1989	9	0.000000	32.000000	32.000000	32.000000	4.0
1	1	1989	10	0.200000	32.000000	32.000000	32.000000	2.0
2	1	1989	11	0.150000	32.000000	32.000000	32.000000	2.0
3	1	1989	12	0.003226	32.000000	32.000000	32.000000	2.0
4	1	1990	1	0.258065	32.000000	32.000000	32.000000	0.0
...	...	...	...	...	...	...	...	...
792	3	2019	9	0.001083	74.125000	83.300000	66.341667	12.0
793	3	2019	10	0.000000	68.153226	80.419355	56.870968	22.0
794	3	2019	11	0.067333	61.308333	72.033333	51.791667	9.0
795	3	2019	12	0.117742	56.491935	64.790323	48.233871	0.0
796	3	2020	1	0.000000	55.750000	64.250000	45.000000	0.0

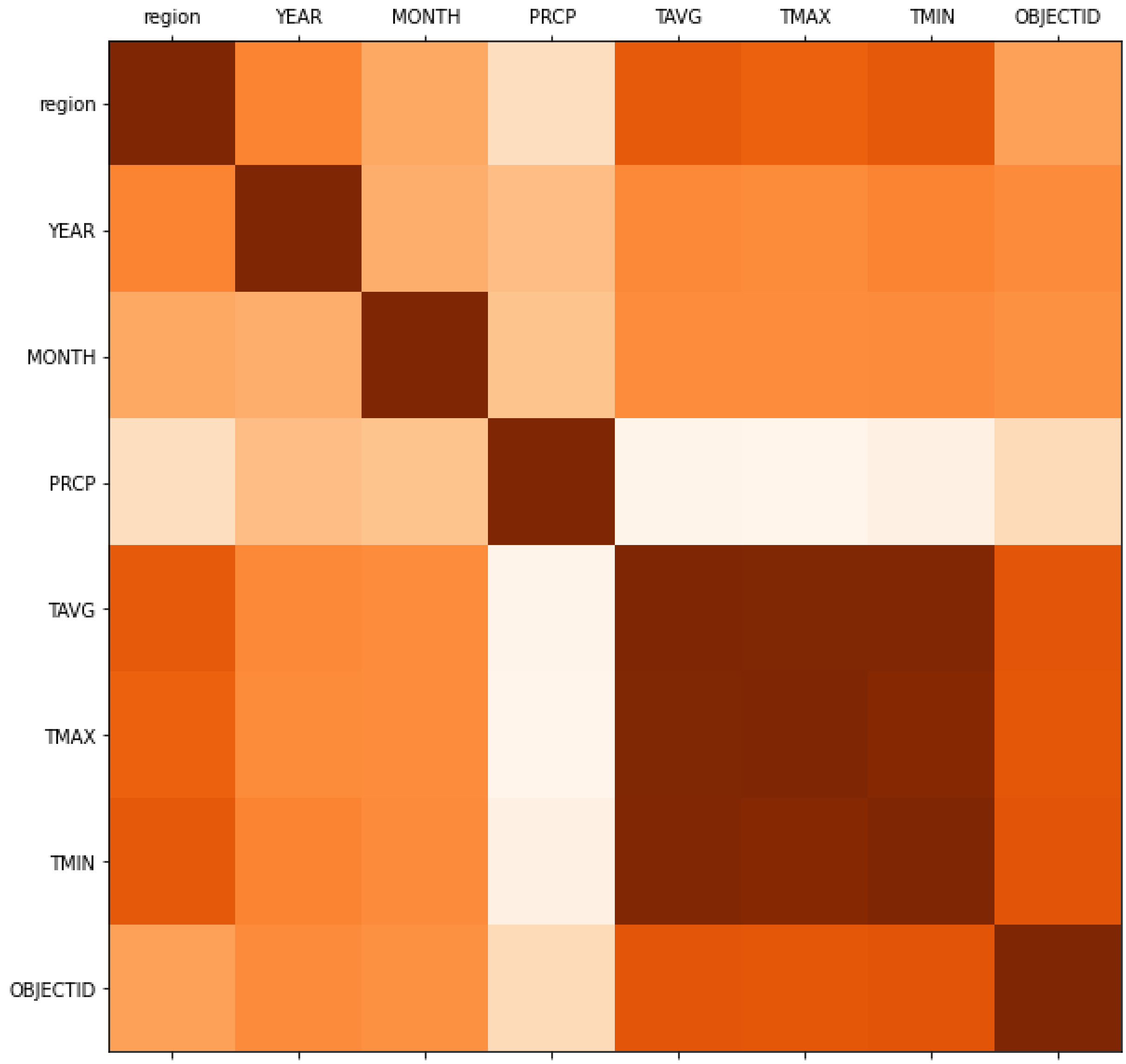
797 rows × 8 columns

```
WDF_Count.corr()['OBJECTID'].to_frame().sort_values('OBJECTID', ascending = False)
```

	OBJECTID
OBJECTID	1.000000
TMIN	0.502871
TAVG	0.493211
TMAX	0.489503
YEAR	0.179985
MONTH	0.148247
region	0.056098
PRCP	-0.339697

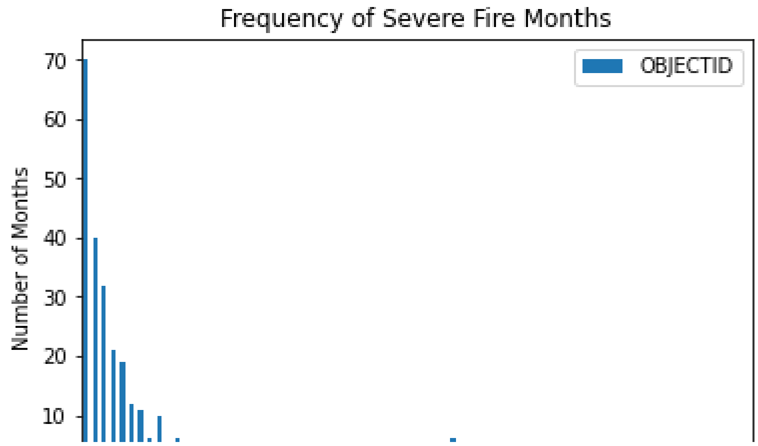
```
def plot_corr(df,size=10):
    corr = WDF_Count.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    ax.matshow(corr,cmap=plt.cm.Oranges)
    plt.xticks(range(len(corr.columns)), corr.columns)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.show()
```

```
plot_corr(WDF_Count)
```




```
#Plot count of classes including 0
y = WDF_Count.groupby(["YEAR", "MONTH"])["OBJECTID"].sum().to_frame()

y = y['OBJECTID'].value_counts().to_frame()
y.reset_index(inplace=True)
y= y.sort_values(by = 'index')
y
y.plot.bar(x = 'index', y = 'OBJECTID')
plt.ylabel("Number of Months")
plt.xlabel("Severity Class of Fires")
plt.title("Frequency of Severe Fire Months")
plt.style.use("dark_background")
```




```
# if negative correlation multiple column by -1
WDF_Count['PRCP'] = WDF_Acres['PRCP']*(-1)
#normalize data
columns = ['TMAX', 'TMIN', 'TAVG','MONTH','PRCP']
def normalize(df, columns):
  result = df[columns]
  result = (result - result.mean())/result.std()
  return result
weather_normlized_count = normalize(WDF_Count, columns)
weather_normlized_count
```

	TMAX	TMIN	TAVG	MONTH	PRCP
0	-1.951860	-0.895084	-1.458965	0.716220	-0.677493
1	-1.951860	-0.895084	-1.458965	1.005979	0.989406
2	-1.951860	-0.895084	-1.458965	1.295739	0.572682
3	-1.951860	-0.895084	-1.458965	1.585499	-0.650607
4	-1.951860	-0.895084	-1.458965	-1.601860	1.473345
...	...	...	...	...	...
792	1.284450	1.834153	1.526032	0.716220	-0.668463
793	1.102721	1.081488	1.102869	1.005979	-0.677493
794	0.573681	0.677820	0.617837	1.295739	-0.116303
795	0.116749	0.395072	0.276545	1.585499	0.303827
796	0.082662	0.138066	0.223971	-1.601860	-0.677493

797 rows × 5 columns

```
#split into test and training after weather normalized
# to get 80% train 20% test - split along 638 row, 292 if only using region 1
X_train_count = weather_normlized_count.iloc[:638]
Y_train_count= WDF_Count['OBJECTID'].iloc[:638]
X_test_count = weather_normlized_count.iloc[638:]
```

```
Y_test_count = WDF_Count['OBJECTID'].iloc[638:]
```

```
poly_model= make_pipeline(PolynomialFeatures(3, include_bias=False), LinearRegressic
poly_model.fit(X_train_count,Y_train_count)
```

```
Pipeline(memory=None,
         steps=[('polynomialfeatures',
                 PolynomialFeatures(degree=3, include_bias=False,
                                    interaction_only=False, order='C')),
                ('linearregression',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

```
y_pred = poly_model.predict(X_test_count)
error = (abs(y_pred - Y_test_count)).sum()/len(y_pred)
error
```

```
15.562014560331216
```

```
#KNNeighbors
knn =KNeighborsRegressor(n_neighbors=7)
knn.fit(X_train_count, Y_train_count)
predictions = knn.predict(X_test_count)
#computing the error
#get the actual value for the test set
#compute the average error of the prediction
error = (abs(predictions - Y_test_count)).sum() / len(predictions)
error
```

```
7.343216531895779
```

Try additional Regression models

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR',LinearReg
pipelines.append(('ScaledLASSO', Pipeline([('Scaler', StandardScaler()),('LASSO', La
pipelines.append(('ScaledEN', Pipeline([('Scaler', StandardScaler()),('EN', ElasticN
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN', KNeigh
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART', Deci
# pipelines.append(('ScaledGBM', Pipeline([('Scaler', StandardScaler()),('GBM', Grad

results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10, random_state=21)
    cv_results = cross_val_score(model, X_train_count, Y_train_count, cv=kfold, scor
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: Fu
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: Fu
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: Fu
  FutureWarning
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: Fu
  FutureWarning
ScaledLR: -110.875893 (102.421806)
ScaledLASSO: -110.098493 (106.814288)
ScaledEN: -112.496648 (110.345405)
ScaledKNN: -108.657478 (92.488297)
ScaledCART: -204.308929 (186.085874)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:296: Fu
  FutureWarning
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.