Machine Learning - Assignment 1

Introduction to Machine Learning

Name : Kendrick Krause
Student ID [redacted]

Section 1 - Initial Data Analysis (EDA)

First we will load in the csv file and check that all the data has loaded in correctly

IMPORTANT - **Unless otherwise stated, all imports are taken from appropriate Machine Learning Labs (Weeks 01 - 05)**

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

deviceLife = pd.read_csv('./dataset/Data_Set.csv', delimiter=',')
deviceLife.shape
```

Out[1]:
```
(2071, 24)
```

In [2]:
```
deviceLife.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   ID                            2071 non-null   int64  
 1   TARGET_LifeExpectancy         2071 non-null   float64
 2   Country                       2071 non-null   int64  
 3   Year                          2071 non-null   int64  
 4   Company_Status                2071 non-null   int64  
 5   Company_Confidence            2071 non-null   int64  
 6   Company_device_confidence     2071 non-null   int64  
 7   Device_confidence             2071 non-null   int64  
 8   Device_returen                2071 non-null   int64  
 9   Test_Fail                     2071 non-null   float64
 10  PercentageExpenditure         2071 non-null   float64
 11  Engine_Cooling                2071 non-null   int64  
 12  Gas_Pressure                  2071 non-null   float64
 13  Obsolescence                  2071 non-null   int64  
 14  ISO_23                        2071 non-null   int64  
 15  TotalExpenditure              2071 non-null   float64
 16  STRD_DTP                      2071 non-null   float64
 17  Engine_failure                2071 non-null   float64
 18  GDP                           2071 non-null   float64
 19  Product_Quantity              2071 non-null   int64  
 20  Engine_failure_Prevalence     2071 non-null   float64
 21  Leakage_Prevalence            2071 non-null   float64
 22  IncomeCompositionOfResources  2071 non-null   float64
 23  RD                            2071 non-null   float64
dtypes: float64(12), int64(12)
memory usage: 388.4 KB
```

From the above code outputs, we can see that the data has loaded in with no null values in any cell.

Next we will glean a better understanding of the information by running some statistics on the data set.

In [3]:
```
deviceLife.describe()
```

Out[3]:

	ID	TARGET_LifeExpectancy	Country	Year	Company_Status	Company_
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	
mean	1036.000000	69.274505	95.360212	2009.518590	0.185418	
std	597.990524	9.482281	54.861641	4.614147	0.388730	
min	1.000000	37.300000	0.000000	2002.000000	0.000000	
25%	518.500000	63.000000	50.000000	2006.000000	0.000000	
50%	1036.000000	71.200000	94.000000	2010.000000	0.000000	
75%	1553.500000	76.000000	144.000000	2014.000000	0.000000	
max	2071.000000	92.700000	192.000000	2017.000000	1.000000	

8 rows × 24 columns

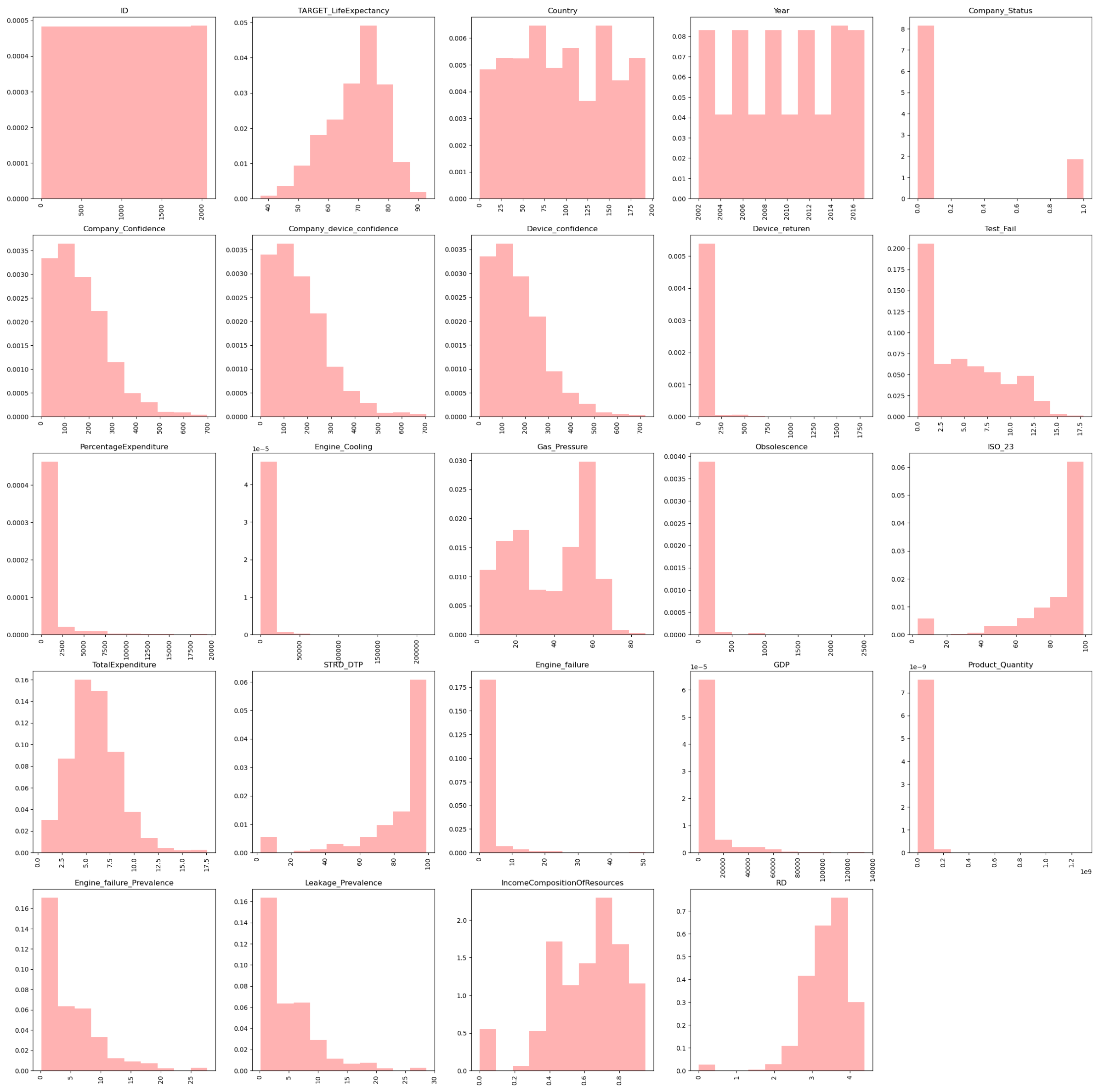
First lets split the data between numerical and categorical data

So that we can analyse each respective set appropriatly

In [4]:
```
# Categorical attributes and continuous variables
categorical_attributes = ['Country','Year','Company_Status']

# Numerical attributes (Discrete)
# Diff Code taken from week 04 ML Lab
numerical_attributes = list(set(deviceLife.columns).difference(set(categorical_att
```

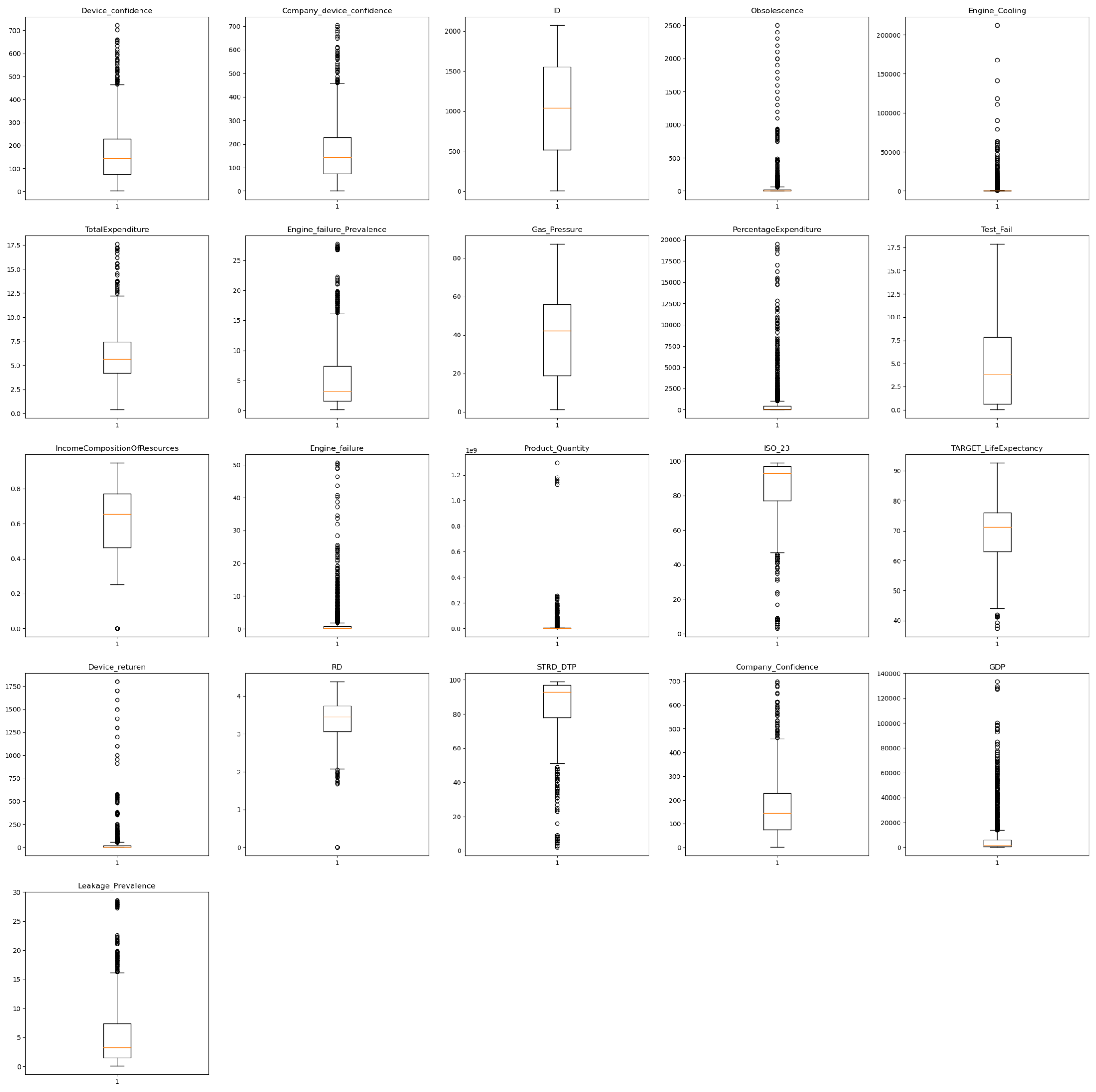
Lets first look at the distribution of all data values

In [5]:
```
# The following is taken from the week02 ML Lab
plt.figure(figsize=(30,30))
for index, column in enumerate(deviceLife.columns):
    plt.subplot(5,5,index+1)
    plt.hist(deviceLife[column], alpha=0.3, color='r', density=True)
    plt.title(column)
    plt.xticks(rotation='vertical')
```



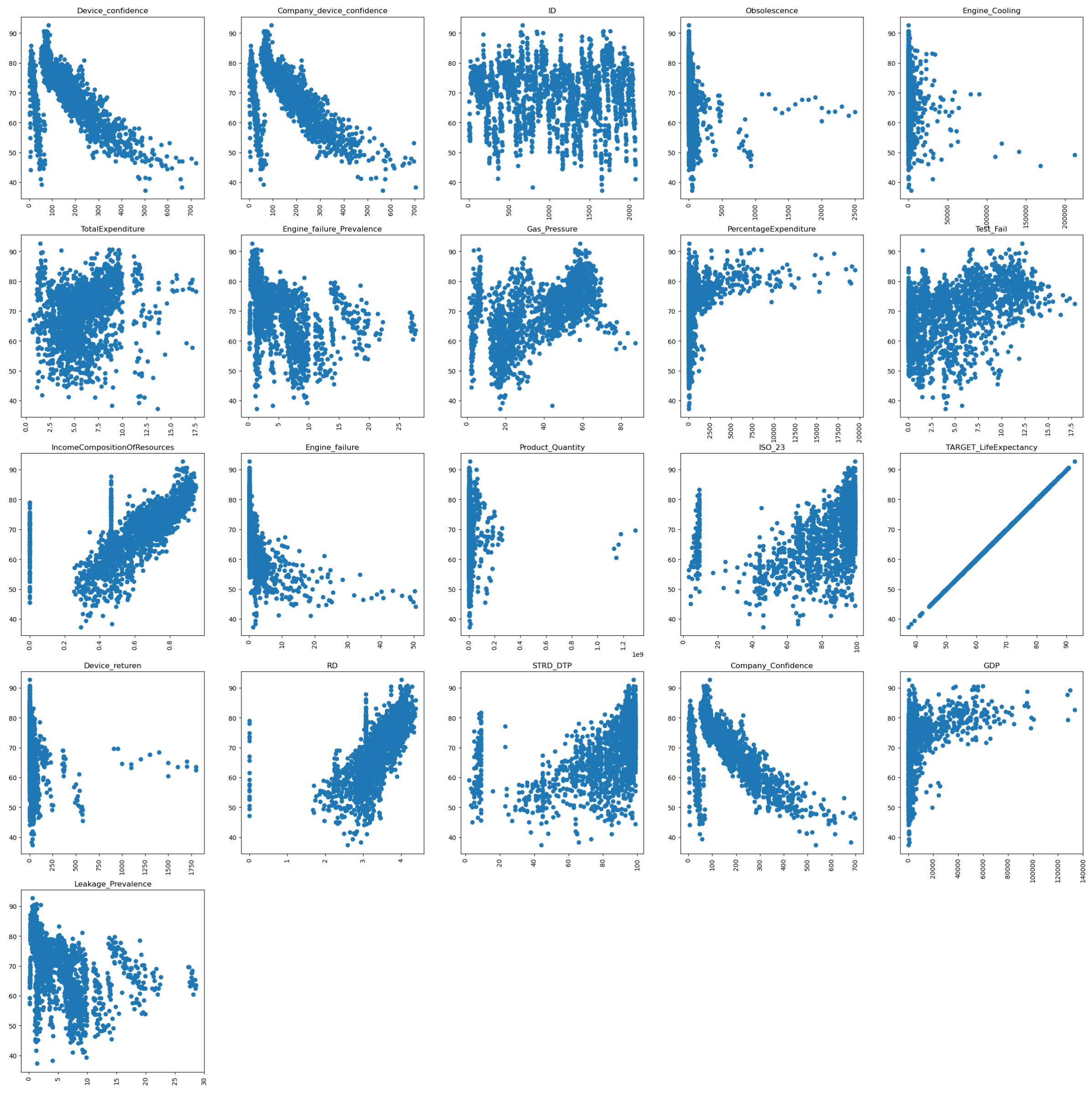
Now we'll look at the distribution of just the numerical data

In [6]:
```
# The following is refactored from the week02 ML Lab (used as above, but with boxpl
plt.figure(figsize=(30,30))
for index, column in enumerate(numerical_attributes):
    plt.subplot(5,5,index+1)
    plt.boxplot(deviceLife[column])
    plt.title(column)
```



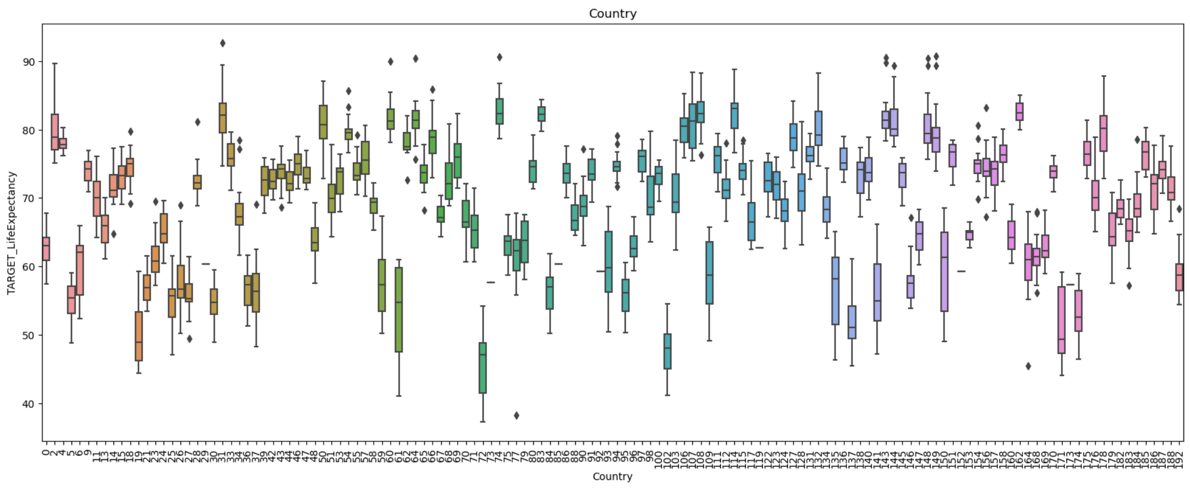
We'll also look at the relationship each numerical attribute has with our target value (TARGET_LifeExpectancy)

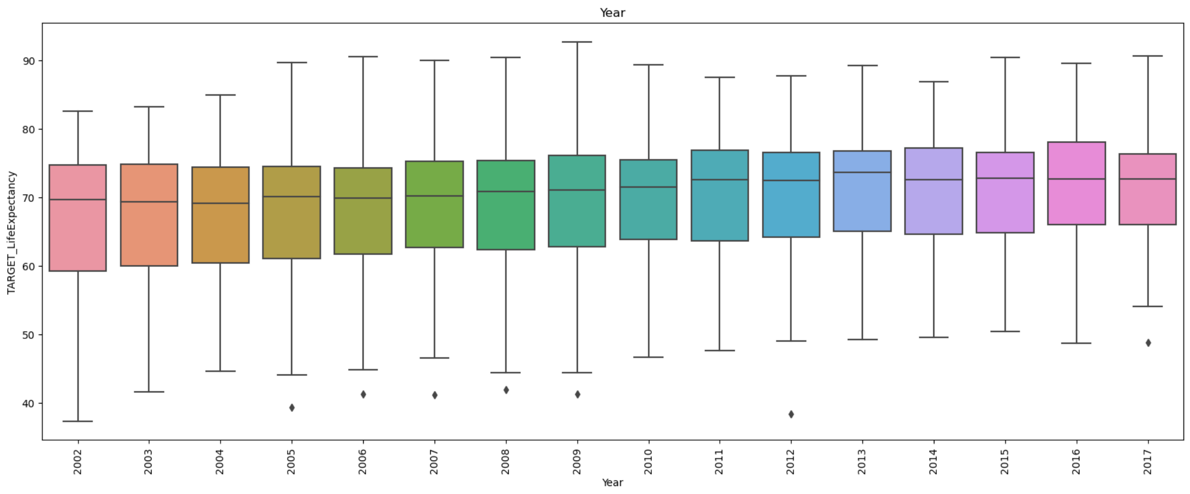
In [7]:
```
# The following is refactored from the week02 ML Lab (used as above, but with scat
plt.figure(figsize=(30,30))
for index, column in enumerate(numerical_attributes):
    plt.subplot(5,5,index+1)
    plt.scatter(deviceLife[column], deviceLife['TARGET_LifeExpectancy'])
    plt.title(column)
    plt.xticks(rotation='vertical')
```

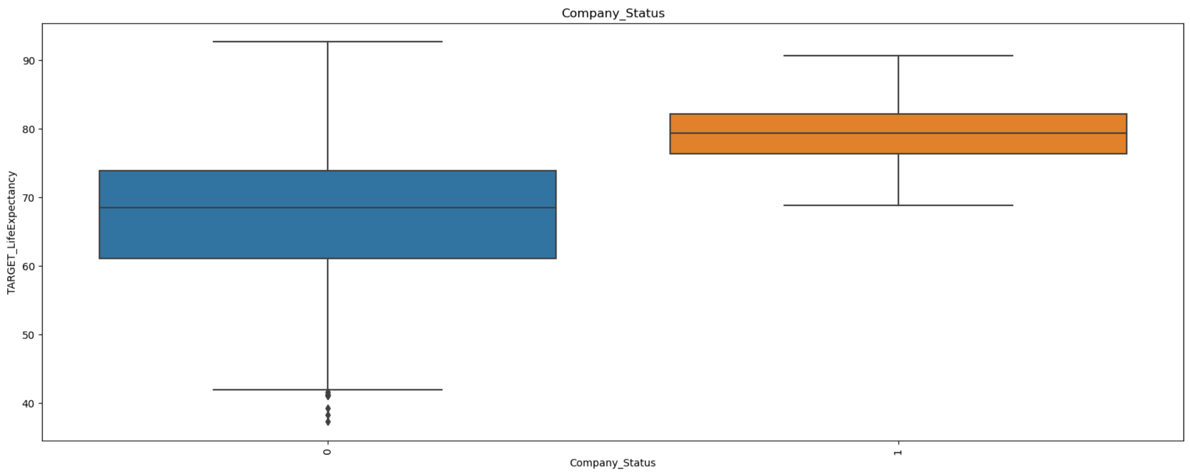


Now we'll look at the relationship our categorical data has with the target output (TARGET_LifeExpectancy)

In [8]:
```
# The following is refactored from the week02 ML Lab
import seaborn as sns

plt.figure(figsize=(20,25))
for index, column in enumerate(categorical_attributes):
    plt.subplot(3,1,index+1)
    ax = sns.boxplot(y='TARGET_LifeExpectancy',x=column,data=deviceLife)
    ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
    plt.title(column)
    plt.xticks(rotation='vertical')
```

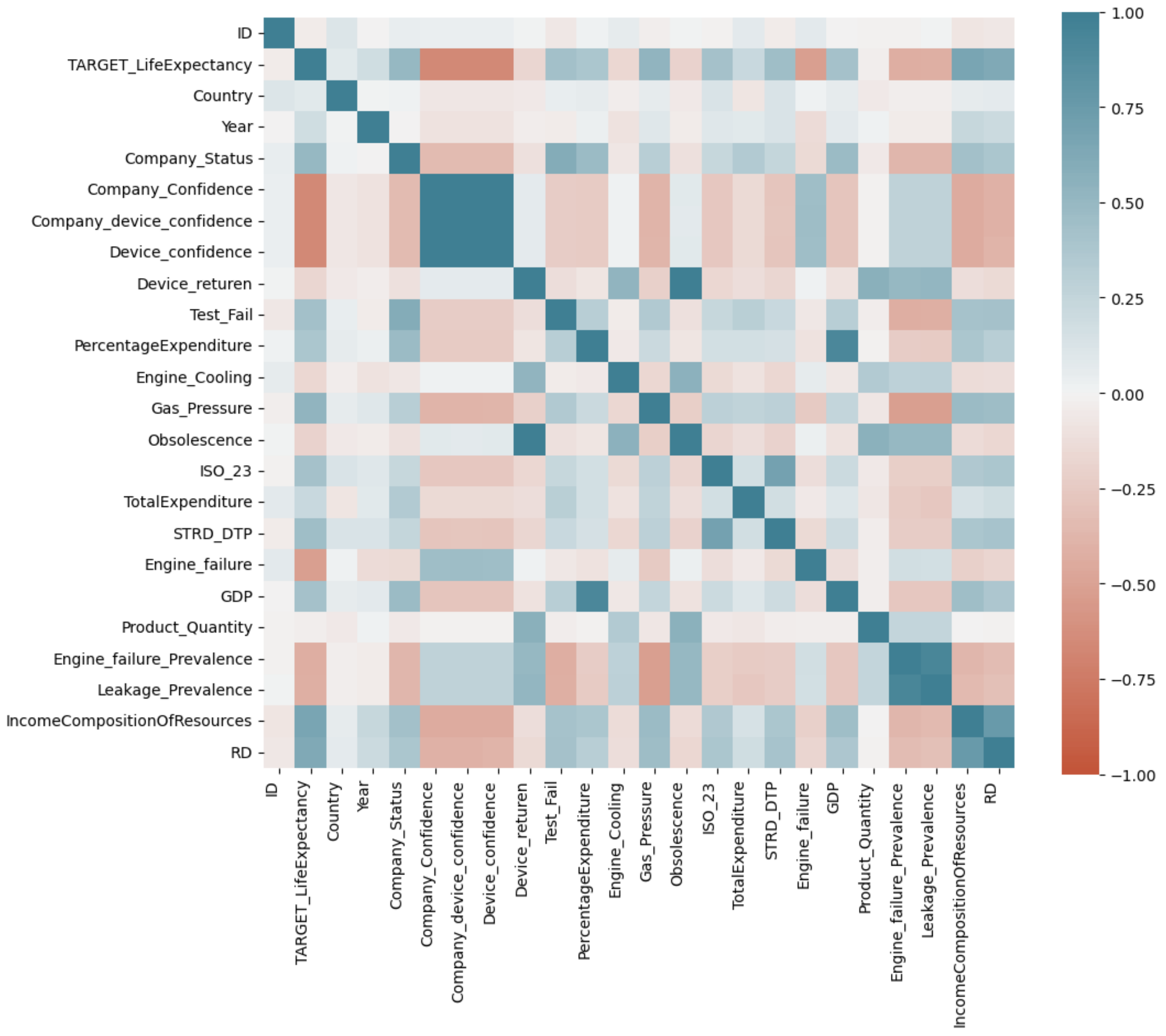






Finally, we will look at the correlation between each attribute.

In [9]:
```
#The following is taken directly from the week 02 ML Lab
f, ax = plt.subplots(figsize=(11, 9))
corr = deviceLife.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=90,
    horizontalalignment='right'
);
```



EDA Observations

Below are just some observations and deductions that can be made from the above statistics

- Most data is heavily skewed
- Skewed Data makes outliers have a stronger impact, as well as more frequent in comparison to even distribution.
- There are some obvious outliers (eg. 'Device Return' should be recorded per 1000 units, but the max shows it can reach 1800)
- The Output of the dataset (TARGET_LifeExpectancy) is distributed relatively high, which does explain why there are little to no larger outliers, but more smaller outliers (as shown in the boxplot above).
- The Histograms have trouble showing the true extent of the data, especially those with large ranges.
- Out of all the numerical data values, (IncomeCompositionOfResources),(ISO_23) and (RD) have a mostly linear relationship with the goal output
- This leaves the remaining numerical data values with more complex, polynomial relationships (eg. Company_device_confidence has a slightly curved trend)

- With categorical variables, there is a large distribution between company and life expectancy, though there is still a trend that could be grasped, with a noticable curve that fluctuates based on each respective company
- The year a device is made in, stays relatively stable, with little change in expected life expectancy.
- Logically, a company that is in development would expect a higher life expectancy of its devices
- There seems to be a strong positive correlation between the target value and several input attributes (eg. IncomeCompositionOfResources, RD, Company_Status have the strongest positive correlation)
- There are also some strong negative correlation between the life_expectancy and probabilitiy values (confidence levels)

Section 2 - Developing the Baseline Model

Now that we've done our preliminary data exploration, it is now time to start developing our baseline model, from which is started upon assumptions we can make from our EDA. As mentioned above, most (but not all) of the numerical data attributes take on a more complex trend than a linear function can accomodate for. Most of the data takes on a curved trend, therefore, it would be best to use a polynomial model to predict the expected output. A polynomial function would also help to determine how reliable our correlation data we obtained actually is, as those with strong correlation (namely our confidence variables), have a curved trend to them. We will begin with a simple quadratic polynomial regression.

We will also use multiple evaluation methods to see how each model fares.

First, lets split the data, we will use hold-out-validation as described in week05 ML Lab to split the data into training, testing and validation (a 60:20:20 split)

In [10]:
```
#We'll first drop the id attribute, since that is not a feature as described in the
deviceLifeNoID = deviceLife.drop(['ID'], axis=1)

#The following is taken directly from the week05 ML Lab
from sklearn.model_selection import train_test_split

with pd.option_context('mode.chained_assignment', None):
    train_data_, test_data = train_test_split(deviceLife, test_size=0.2,
                                              shuffle=True,random_state=0)

with pd.option_context('mode.chained_assignment', None):
    train_data, val_data = train_test_split(train_data_, test_size=0.25,
                                            shuffle=True,random_state=0)

train_X = train_data.drop(['ID','TARGET_LifeExpectancy',], axis=1).to_numpy()
train_y = train_data[['TARGET_LifeExpectancy']].to_numpy()

test_X = test_data.drop(['ID','TARGET_LifeExpectancy',], axis=1).to_numpy()
test_y = test_data[['TARGET_LifeExpectancy']].to_numpy()
#save the ids for test data (for when exporting final predictions)
test_id = test_data[['ID']].to_numpy()
```

```
val_X = val_data.drop(['ID','TARGET_LifeExpectancy',], axis=1).to_numpy()
val_y = val_data[['TARGET_LifeExpectancy']].to_numpy()

print(train_data.shape[0], val_data.shape[0], test_data.shape[0])
```

```
1242 414 415
```

Next we will set up a function to predict with our given models and evaluate their performance.

We will use a broad range of performance evaluation measures to evaluate our model:

- **r^2** will give a simple percentage based accuracy for our model
- **MAE** will give an average room for error of our predictions, which should be relatively small, given that all of our data was scaled to the same range, so it should be a good metric
- **MSE** will be a good indicator to see how great (or small) our outliers (which have yet to be addressed) affect our predictions.

We will use 2 methods to seperate our training and validation scores for a more coherent code. (training scores will be useful for when we want to use regularisation)

In [11]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

def get_val_performance_eval(model, val_X, val_Y):
    val_pred = model.predict(val_X)

    #The following is taken from the week 04 ML Lab
    r2 = r2_score(val_y, val_pred)

    #MAE and MSE was adapted from the week04 ML Lab and the relevant API
    MAE = mean_absolute_error(val_y, val_pred)
    MSE = mean_squared_error(val_y, val_pred)

    return r2, MAE, MSE
```

In [12]:
```
def get_train_performance_eval(model, train_X, train_Y):
    train_pred = model.predict(train_X)

    #The following is taken from the week 04 ML Lab
    r2 = r2_score(train_y, train_pred)

    #MAE and MSE was adapted from the week04 ML Lab and the relevant API
    MAE = mean_absolute_error(train_y, train_pred)
    MSE = mean_squared_error(train_y, train_pred)

    return r2, MAE, MSE
```

Next we will preprocess our data and scale it with appropriate measures.

We will first use the PolynomialFeatures class to apply polynomial conditions and scale our data. We will use MinMax Scaling for this, as seen in our EDA (our describe method), our data is outweighed with larger ranges than others, we will streamline the data and scale each data attribute to [0,1]

In [13]:
```
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler

#The following is taken directly from the week05 ML Lab
quadratic = PolynomialFeatures(2)
quadratic.fit(train_X)
train_X = quadratic.transform(train_X)
test_X = quadratic.transform(test_X)
val_X = quadratic.transform(val_X)

minMax = MinMaxScaler()
minMax.fit(train_X)

train_X = minMax.transform(train_X)
val_X = minMax.transform(val_X)
test_X = minMax.transform(test_X)
```

There is also the problem of the categorical data, particullarly country and year, which should be hot-encoded if they want to be thoroughly tested. But, since they have a neutral correlation, and that encoding would create many extra columns, it would be far to heavy then it is seemingly worth. So for now we will leave as is. Since company status is only a binary, it can stay as is also.

We'll start using the basic linear regression model

In [14]:
```
from sklearn.linear_model import LinearRegression

#The following is taken from the week04 ML Lab
linearReg_model = LinearRegression().fit(train_X, train_y)

r2, MAE, MSE = get_val_performance_eval(linearReg_model, val_X, val_y)

print('The R^2 score for the linear regression model is: {:.3f}'.format(r2))
print('The MAE score for the linear regression model is: {:.3f}'.format(MAE))
print('The MSE score for the linear regression model is: {:.3f}'.format(MSE))
```

```
The R^2 score for the linear regression model is: 0.421
The MAE score for the linear regression model is: 3.279
The MSE score for the linear regression model is: 52.825
```

Unregularised Polynomial Regression Analysis (Baseline)

With the bare minimum applied to our model, we can see that it does a very poor job at predicting unseen data values. An explanation for this poor performance could easily be explained by the lack of work done to minimise distribution skew (aka. our outliers). This is supported by our MSE score, which shows us explicitly that there is a drastic impact from our outliers, so we will definitely need to adjust our model to accomodate for this.

Section 3 - Refining our Model

Now it is time to implement some changes to our baseline model so that it can better perform the task of prediciting the life span of various devices.

We will now implement a regularised model, and determine the optimal hyperparameter value to increase the accuracy of our predictions. We will begin with implementing the base ridge model, with the default alpha value (alpha = 1).

Ridge Regression - "This model solves a regression model where the loss function is the linear least squares function" [https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html] Ridge regression is a regularised regression model which will aim to minimize the error of our squared difference, which will in theory, improve our MSE evaluation score drastically. It will also help fine tune our fit by adding a parameter value to data point (alpha).

In [15]:
```
from sklearn.linear_model import Ridge

ridge_model = Ridge().fit(train_X, train_y)
r2, MAE, MSE = get_val_performance_eval(ridge_model, val_X, val_y)

print('The R^2 score for the ridge regression model is: {:.3f}'.format(r2))
print('The MAE score for the ridge regression model is: {:.3f}'.format(MAE))
print('The MSE score for the ridge regression model is: {:.3f}'.format(MSE))
```

```
The R^2 score for the ridge regression model is: 0.858
The MAE score for the ridge regression model is: 2.722
The MSE score for the ridge regression model is: 12.997
```

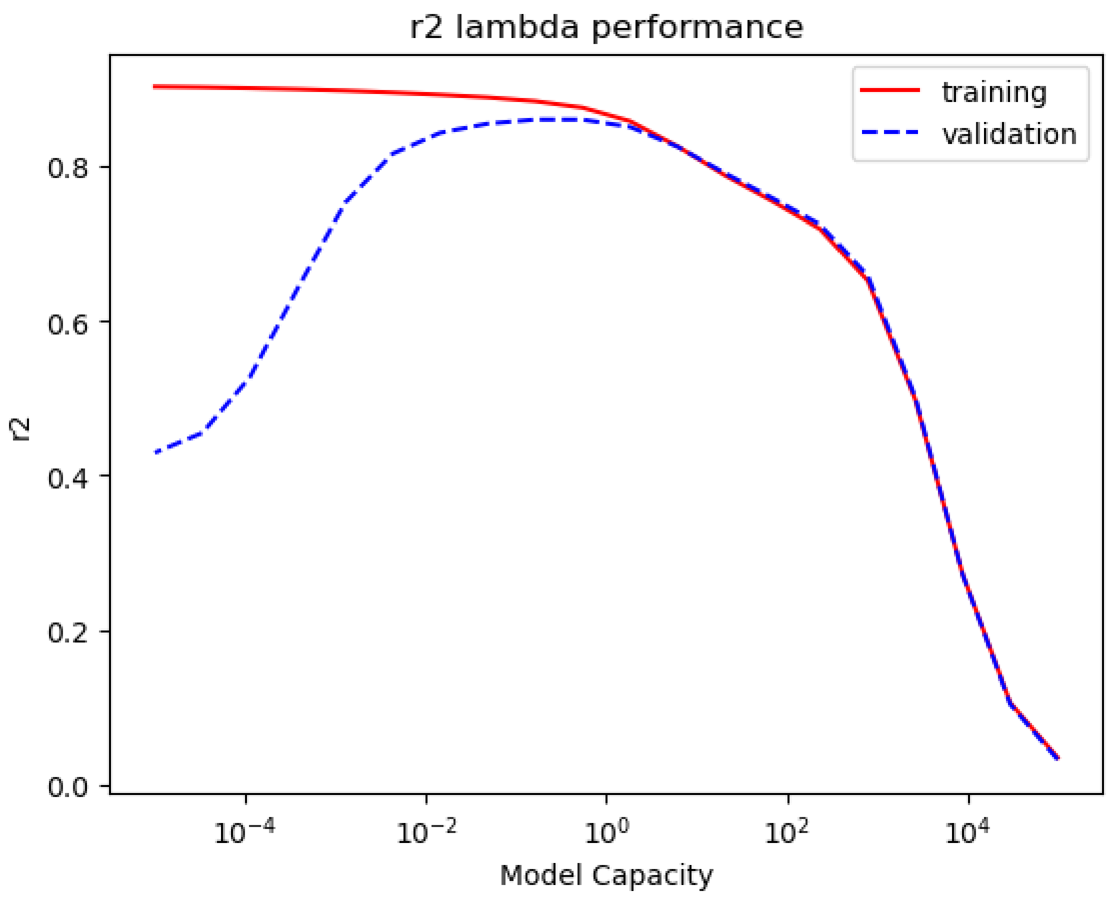
Even with the minimium regularisation, our model has improved drastically, but lets explore if we can improve our model any further by doing grid search to refine our hyperparameter.

Since the default alpha for ridge model (1.0) did fairly well in optimizing our model, we will collate a small sample size around that value to see if theres any other hyper parameter around that defualt that has a better performance.

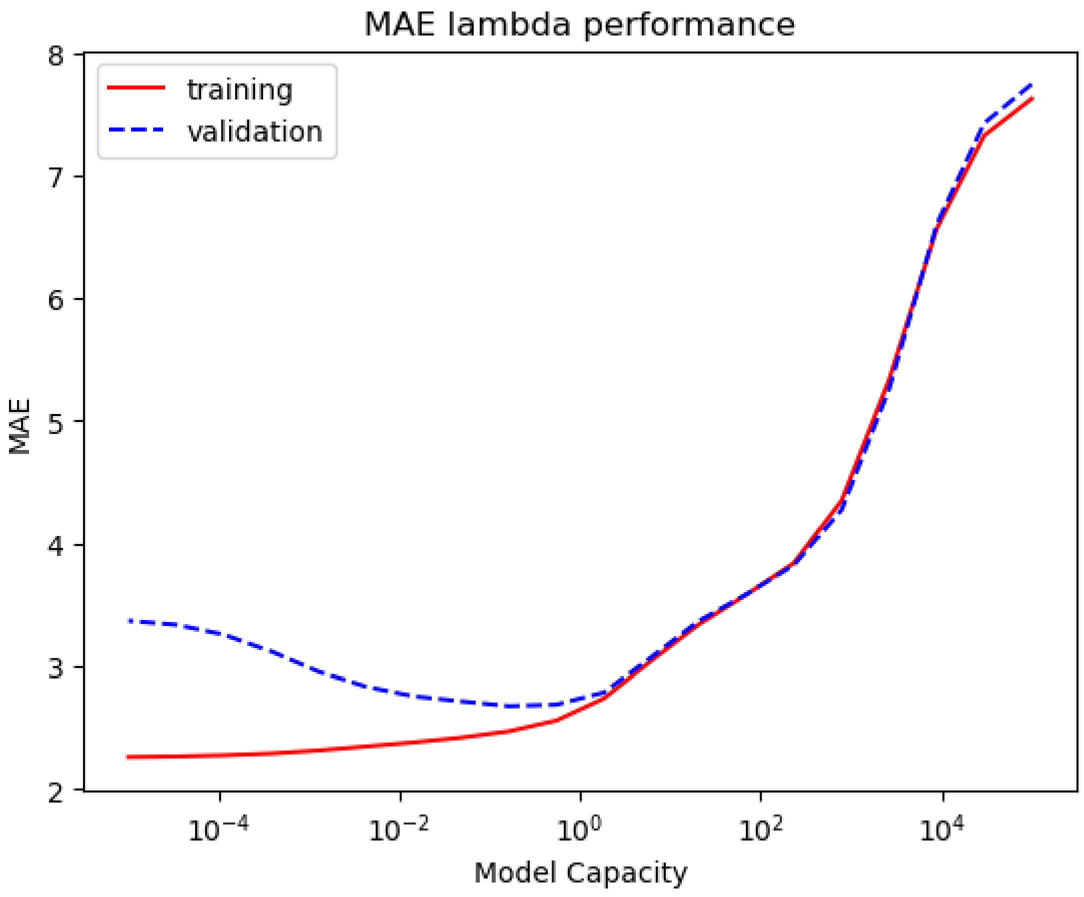
In [16]:
```
# The following is adapted from the week05 ML Lab
lambda_paras = np.logspace(-5, 5, num=20)

# Collate training and validation performance scores
r2_train_performance = list()
MAE_train_performance = list()
MSE_train_performance = list()

r2_val_performance = list()
MAE_val_performance = list()
MSE_val_performance = list()

#perform evaluations on training data and evaluation data
for lambda_para in lambda_paras:
    lam_model = Ridge(alpha = 1.0/lambda_para).fit(train_X, train_y.ravel())

    r2_train, MAE_train, MSE_train = get_train_performance_eval(lam_model, train_X
    r2_val, MAE_val, MSE_val = get_val_performance_eval(lam_model, val_X, val_y)

    r2_train_performance.append(r2_train)
    MAE_train_performance.append(MAE_train)
    MSE_train_performance.append(MSE_train)

    r2_val_performance.append(r2_val)
    MAE_val_performance.append(MAE_val)
    MSE_val_performance.append(MSE_val)
```

Now we will plot our scores against training and validation, to find the smallest gap and highest validation performance for some given parameter

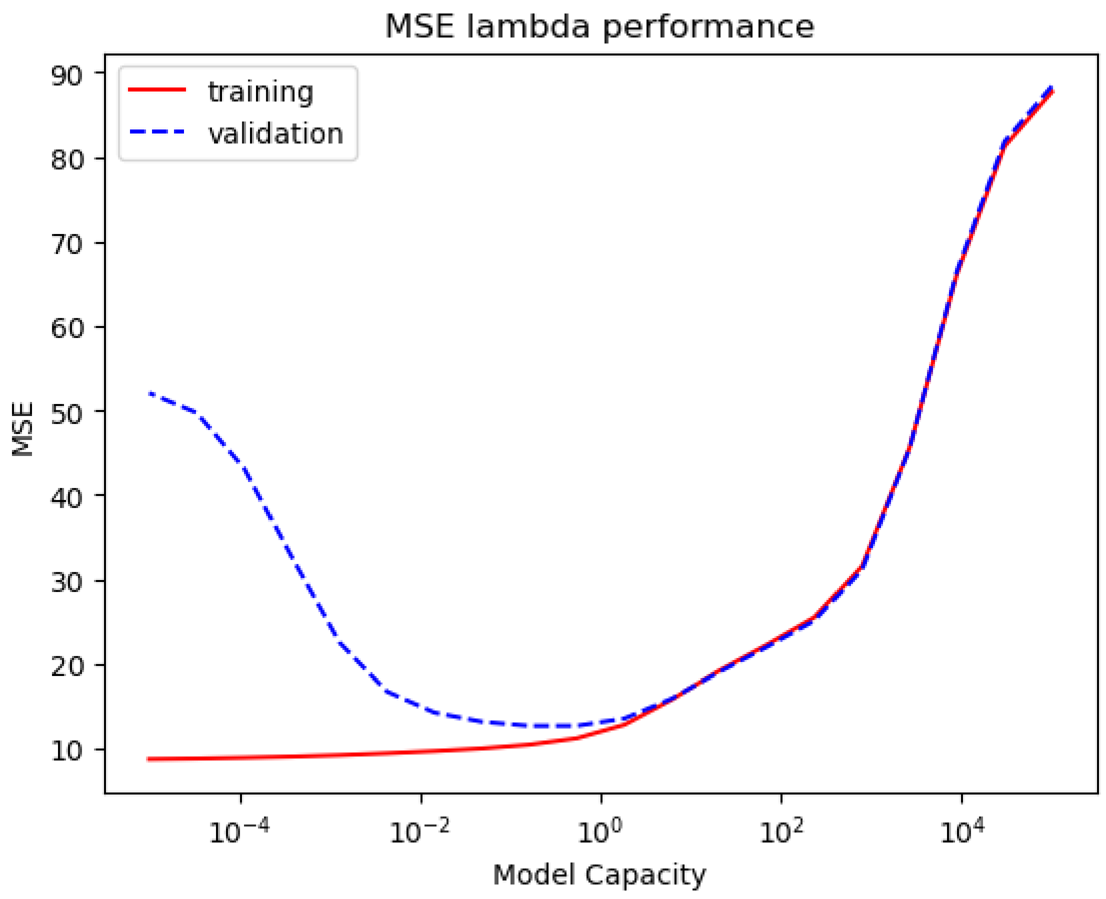
In [17]:
```python
# The following is adapted directly from the week05 ML Lab
plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [r2_train for r2_train in r2_train_performance], 'r-')

plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [r2_val for r2_val in r2_val_performance], 'b--')

plt.xscale("log")
plt.ylabel('r2')
plt.xlabel('Model Capacity')
plt.legend(['training','validation'])
plt.title('r2 lambda performance')
plt.show()
```



In [18]:
```python
# The following is adapted directly from the week05 ML Lab
plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [MAE_train for MAE_train in MAE_train_performance], 'r-')

plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [MAE_val for MAE_val in MAE_val_performance], 'b--')

plt.xscale("log")
plt.ylabel('MAE')
plt.xlabel('Model Capacity')
plt.legend(['training','validation'])
plt.title('MAE lambda performance')
plt.show()
```



In [19]:
```
# The following is adapted directly from the week05 ML Lab
plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [MSE_train for MSE_train in MSE_train_performance], 'r-')

plt.plot([1.0/lambda_para for lambda_para in lambda_paras],
         [MSE_val for MSE_val in MSE_val_performance], 'b--')

plt.xscale("log")
plt.ylabel('MSE')
plt.xlabel('Model Capacity')
plt.legend(['training','validation'])
plt.title('MSE lambda performance')
plt.show()
```



Regularised Polynomial Ridge Regression Analysis

Upon further investigations, it appears that the default alpha value (1.0) was more than enough to maximise the models performance. This can probably be explained by the all across scaling (MinMax) we did before developing our model, to create a universal domain.

Based on our research and development, we can come to the conclusion that the ridge regression model has the adequate capacity to perform our task (predicting the life expectancy of various devices) without manipulating the data any further (and thus risking overfitting). This will therefore be used to perform our final predictions.

Section 4 - Final Predictions and Conclusion

We will now implement our Ridge Model to predict the life_expectancy of our testing data

We will append the test predictions and map them back to the corresponding predictions (hstack, and dataFrame export method taken from https://stackoverflow.com/questions/45399950/how-to-create-a-pandas-dataframe-with-several-numpy-1d-arrays)

NOTE - The hstack method converts the test_ids to float values (will just add .0 to each id)

In [20]:
```
final_prediction = ridge_model.predict(test_X)
#append test ids to the final predictions
appended_prediction = np.hstack((test_id, final_prediction))
```

```
dataFrame = pd.DataFrame(appended_prediction, columns=['ID','TARGET_LifeExpectancy
dataFrame.to_csv("[redacted].csv", header=True, index=False)
```