Car CO2 Emission Prediction (Regression)

About this notebook

In this notebook, I will use data about cars and regression models to predict CO2 emission. This data provides model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. Note: This is a part of IBM Data Scientist course projects.

Table of contents

</div>

In [2]:
```
#import packages
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
import seaborn as sns
```

In [3]:
```
#get data
df=pd.read_csv(r'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv')
```

This dataset has 1,067 observations and 13 columns.

Variable Descriptions

We have downloaded a fuel consumption dataset, **`FuelConsumption.csv`** , which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. [Dataset source (http://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64)](http://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64)

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182

Understand Data

In [12]:
```
#shape of the data
df.shape
```

Out[12]: (1067, 13)

In [10]:
```
#the head of data
df.head()
```

Out[10]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMI
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	

In [103]:
```
df.dtypes
```

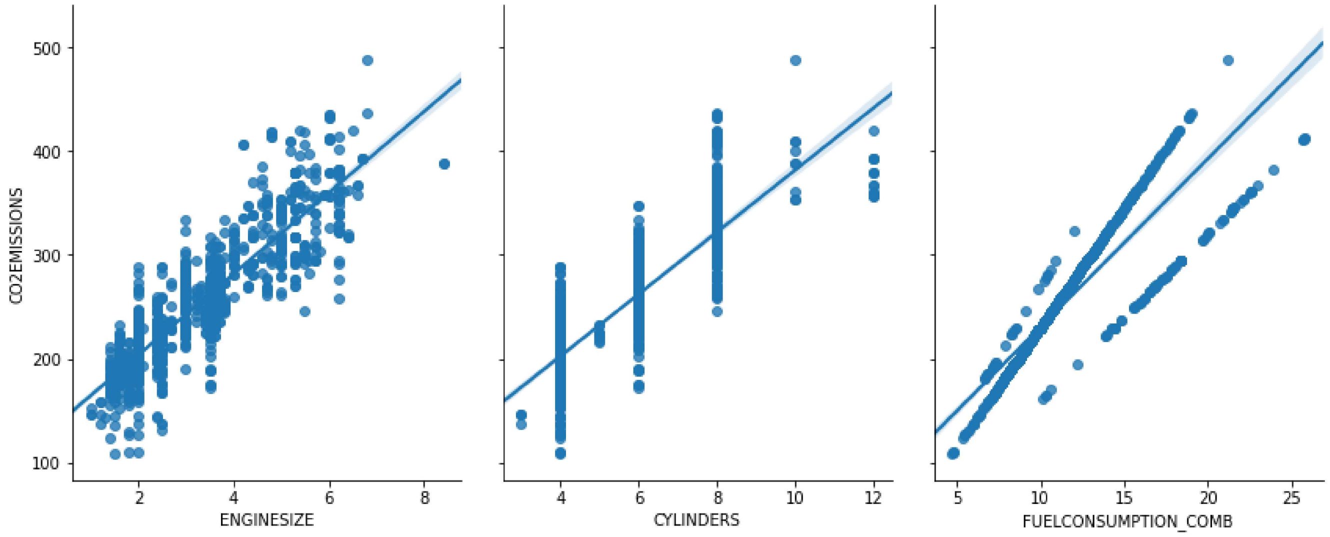
Out[103]:
```
MODELYEAR                      int64
MAKE                          object
MODEL                         object
VEHICLECLASS                  object
ENGINESIZE                   float64
CYLINDERS                      int64
TRANSMISSION                  object
FUELTYPE                      object
FUELCONSUMPTION_CITY         float64
FUELCONSUMPTION_HWY          float64
FUELCONSUMPTION_COMB         float64
FUELCONSUMPTION_COMB_MPG       int64
CO2EMISSIONS                   int64
dtype: object
```

In [13]:
```
#see the descriptive statistics of the numeric variables
df.describe()
```

Out[13]:

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067
mean	2014.0	3.346298	5.794752	13.296532	9.474602	11
std	0.0	1.415895	1.797447	4.101253	2.794510	3
min	2014.0	1.000000	3.000000	4.600000	4.900000	4
25%	2014.0	2.000000	4.000000	10.250000	7.500000	9
50%	2014.0	3.400000	6.000000	12.600000	8.800000	10
75%	2014.0	4.300000	8.000000	15.550000	10.850000	13
max	2014.0	8.400000	12.000000	30.200000	20.500000	25

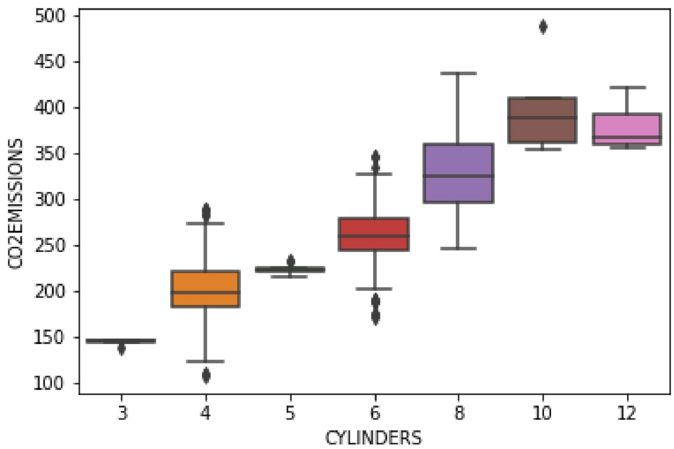
In [19]:
```
#Generate scatter plots for numeric variables VS CO2EMISSIONS to see if correlations
sns.pairplot(df, x_vars=["ENGINESIZE", "CYLINDERS", 'FUELCONSUMPTION_COMB'], y_vars=["CO2EMISSIONS"
],
             height=5, aspect=.8, kind="reg");
```



CYLINDERS are categorical. There are 5 distinct values in the dataset. We want to see if there are any difference in the mean of each type of cylinder numbers realting to CO2EMISSIONS.

In [25]:
```
#box plot for CYLINERS
sns.boxplot(x="CYLINDERS", y="CO2EMISSIONS", data=df)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x20c70bbf630>



So, the number of cylinders has some influence in CO2EMISSIONS.

There are four columns about fuel consumptions: FUELCONSUMPTION_CITY, FUELCONSUMPTION_HWY, FUELCONSUMPTION_COMB, and FUELCONSUMPTION_COMB_MPG. If they are highly correlated, we can eliminate some of them to make the model lighter. We will run a correlation.

In [29]:
```python
df.corr()
```

Out[29]:

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_
MODELYEAR	NaN	NaN	NaN	NaN	
ENGINESIZE	NaN	1.000000	0.934011	0.832225	0.77
CYLINDERS	NaN	0.934011	1.000000	0.796473	0.72
FUELCONSUMPTION_CITY	NaN	0.832225	0.796473	1.000000	0.96
FUELCONSUMPTION_HWY	NaN	0.778746	0.724594	0.965718	1.00
FUELCONSUMPTION_COMB	NaN	0.819482	0.776788	0.995542	0.98
FUELCONSUMPTION_COMB_MPG	NaN	-0.808554	-0.770430	-0.935613	-0.89
CO2EMISSIONS	NaN	0.874154	0.849685	0.898039	0.86

So, we can use one of the columns about fuel consumption without lossing too much information.Now we know ENGINESIZE, CYLINERS, and FUELCOMSUMPTION_COMB are useful for building the regression model.

Linear Regression

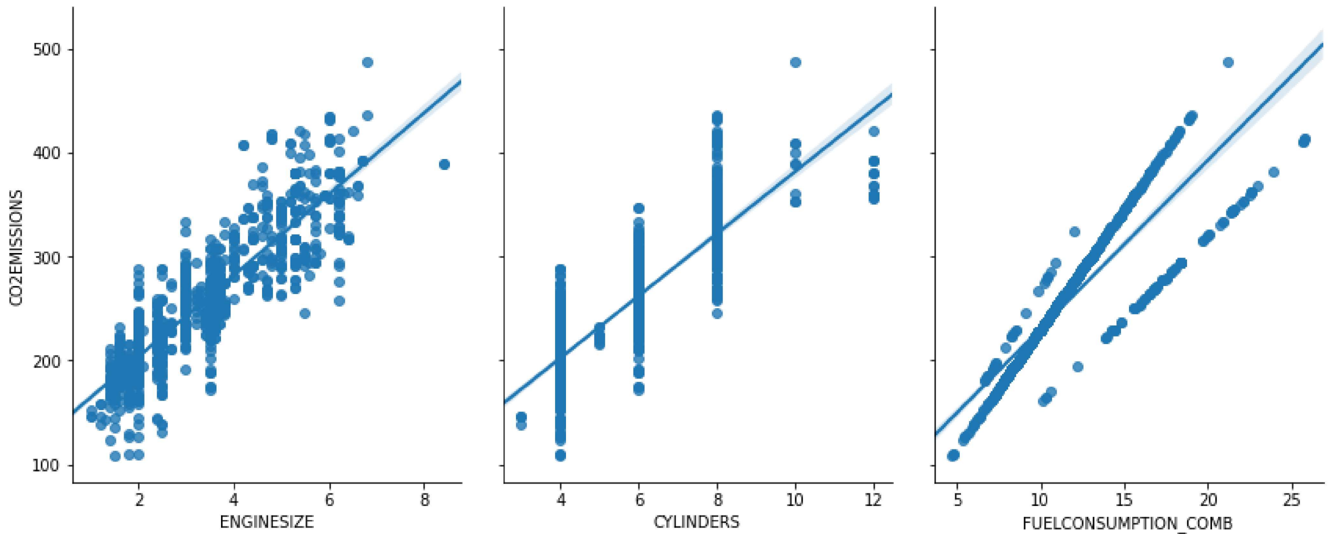
In [4]:
```python
#create a new dataframe for model building
lrdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
lrdf.shape
```

Out[4]: (1067, 4)

In [5]:
```python
#spilt train and test data
msk = np.random.rand(len(df)) < 0.8
train = lrdf[msk]
test = lrdf[~msk]
```

In [37]:
```python
#train data distribution
sns.pairplot(lrdf, x_vars=['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB'], y_vars=['CO2EMISSION
S'],
             height=5, aspect=.8, kind="reg")
```

Out[37]: <seaborn.axisgrid.PairGrid at 0x20c70c95f60>



In [6]:
```
#build the model
#step 1, import packages
from sklearn.linear_model import LinearRegression
lm=LinearRegression()

#step 2, create independent variables
X=train[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']]
Y=train[['CO2EMISSIONS']]
```

In [47]:
```
#step 3, fit the linear regression model
lm.fit(X,Y)
# The coefficients
print ('Coefficients: ', lm.coef_)
print ('Intercept: ',lm.intercept_)
```

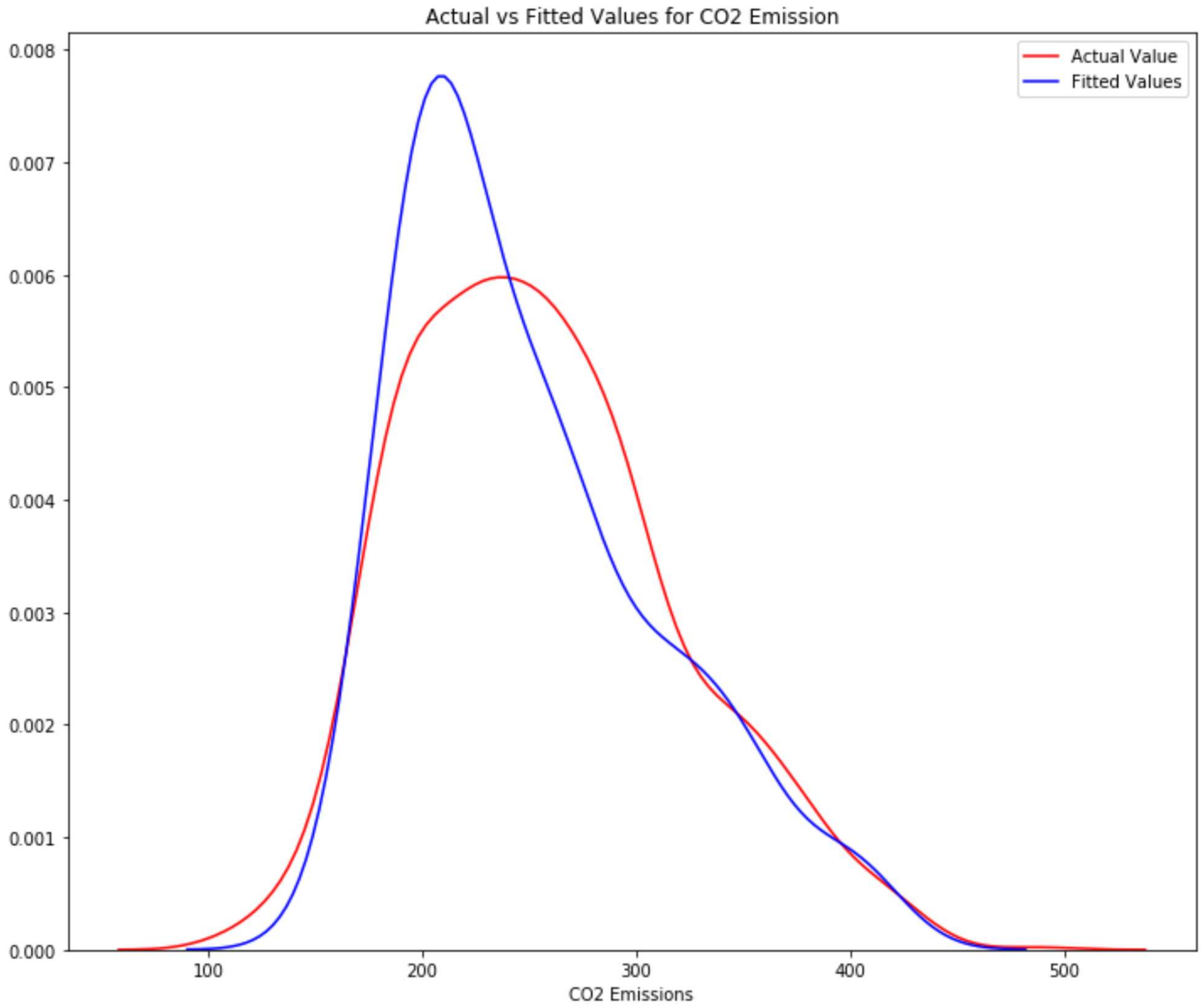
```
Coefficients:  [[ 9.92107897  7.75011521 10.0252928 ]]
Intercept:  [61.92097822]
```

Now the model equation is: Y = 61.92097822 + 9.92107897*ENGINESIZE + 7.75011521*CYLINDERS + 10.0252928*FUELCONSUMPTION_COMB + Error

In [48]:
```
#model evaluation
from sklearn.metrics import r2_score

test_x = test[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']]
test_y = test[['CO2EMISSIONS']]
test_y_hat = lm.predict(test_x)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_hat , test_y) )
```

```
Mean absolute error: 16.67
Residual sum of squares (MSE): 527.73
R2-score: 0.85
```

In [51]:
```
#Distribution Plot
width = 12
height = 10
plt.figure(figsize=(width, height))

ax1 = sns.distplot(lrdf['CO2EMISSIONS'], hist=False, color="r", label="Actual Value")
sns.distplot(test_y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)

plt.title('Actual vs Fitted Values for CO2 Emission')
plt.xlabel('CO2 Emissions')
plt.ylabel('')

plt.show()
plt.close()
```



Non Linear Regression

Although the above linear model fits well (R^2 is 0.85), as seen from the distribution plot, it fails to explain some data points. This time we try non-linear regression models.

Polynomial Model

In [7]:
```
#import packages
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
```

In [8]:
```
#first, try ENGINESIZE as the only independent variable
train_x=train[['ENGINESIZE']]
train_y=train[['CO2EMISSIONS']]
poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly
```

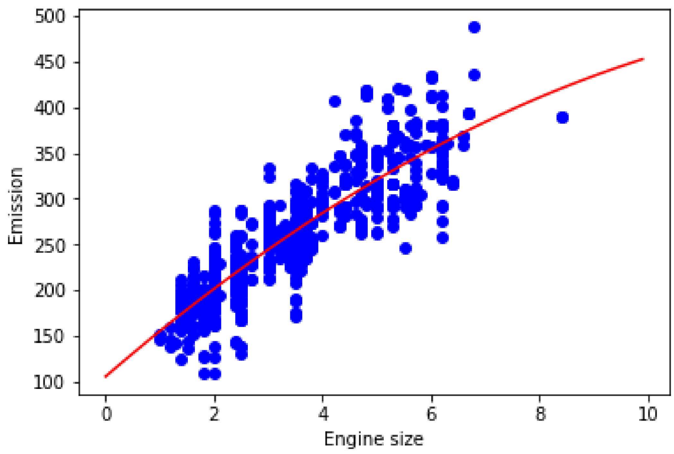
Out[8]:
```
array([[ 1.  ,  2.4 ,  5.76],
       [ 1.  ,  1.5 ,  2.25],
       [ 1.  ,  3.5 , 12.25],
       ...,
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24]])
```

In [59]:
```
train_y_ = lm.fit(train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', lm.coef_)
print ('Intercept: ',lm.intercept_)
```

```
Coefficients:  [[ 0.         51.17136746 -1.63000129]]
Intercept:  [105.69276866]
```

In [60]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = lm.intercept_[0]+ lm.coef_[0][1]*XX+ lm.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[60]:
```
Text(0, 0.5, 'Emission')
```

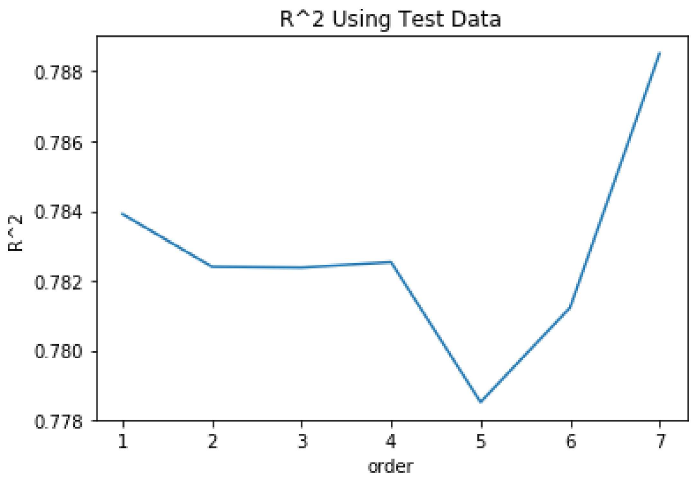


In [62]:
```
#Evaluation
test_x = test[['ENGINESIZE']]
test_y = test[['CO2EMISSIONS']]
test_x_poly = poly.fit_transform(test_x)
test_y_ = lm.predict(test_x_poly)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_ , test_y) )
```

```
Mean absolute error: 20.65
Residual sum of squares (MSE): 737.28
R2-score: 0.74
```

In [13]:
```
#Optimize the polynomial degrees
Rsqu_test = []

order = [1, 2, 3, 4, 5, 6, 7]
for n in order:
    poly = PolynomialFeatures(degree=n)
    train_x=train[['ENGINESIZE']]
    train_y=train[['CO2EMISSIONS']]
    train_x_poly = poly.fit_transform(train_x)
    test_x = test[['ENGINESIZE']]
    test_y = test[['CO2EMISSIONS']]
    test_x_poly = poly.fit_transform(test_x)
 #lr=LinearRegression()
    lm.fit(train_x_poly, train_y)
    Rsqu_test.append(lm.score(test_x_poly, test_y))

plt.plot(order, Rsqu_test)
#plt.figure(order,Rsqu_test, figsize=(3,4))
plt.xlabel('order')
plt.ylabel('R^2')
plt.title('R^2 Using Test Data')
```

Out[13]: Text(0.5, 1.0, 'R^2 Using Test Data')

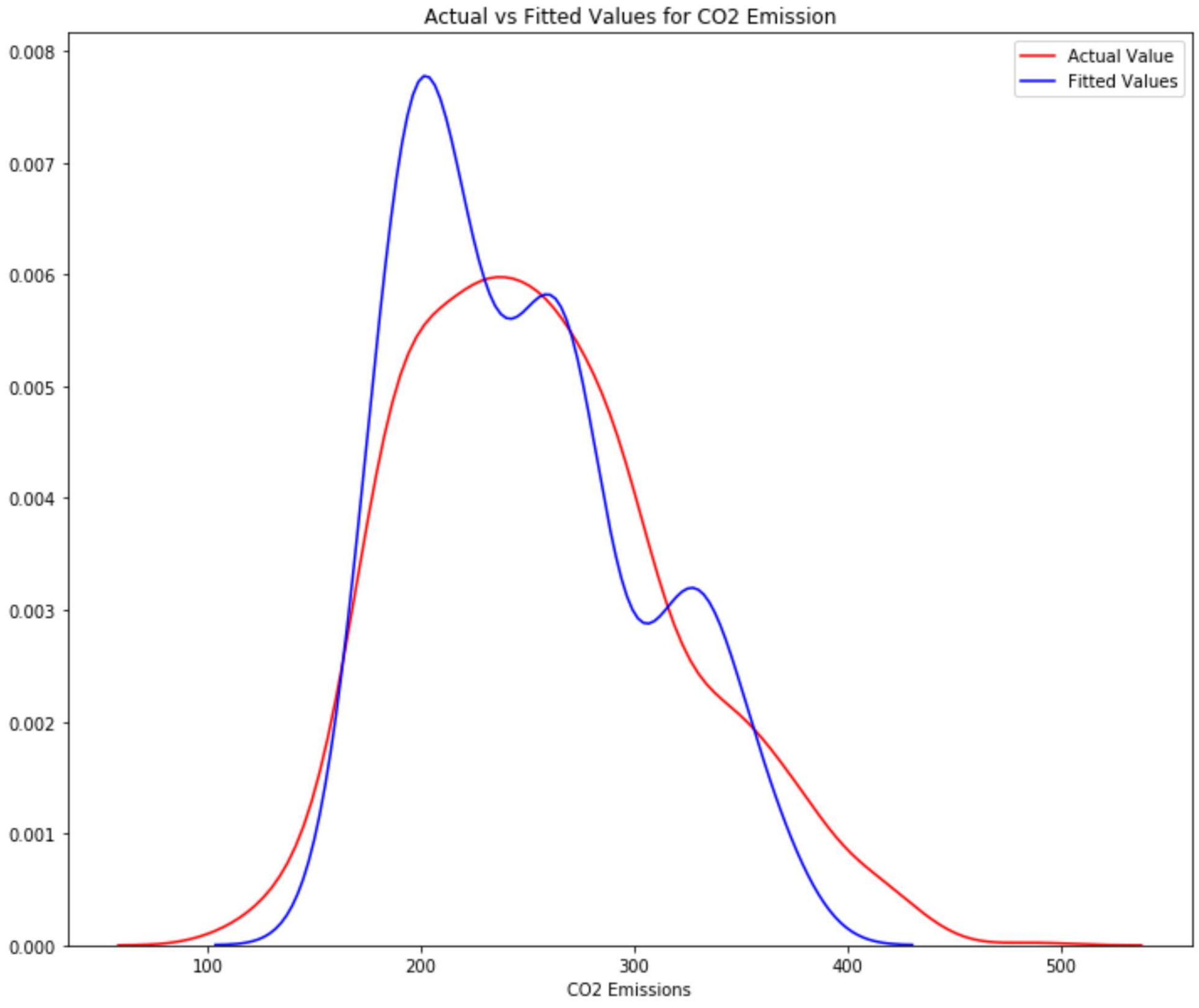


When the polynomial degree for ENGINESIZE is 4, we have a good R^2. Although it the R^2 increases when the degree increases, we use degreee=4 because it's not necessary to let the degree equal to 7 or even higer. The model may fit the noise too well. Let's have a look of the distribution plot.

In [81]:

```
#degree equals to 4
poly = PolynomialFeatures(degree=4)
train_x=train[['ENGINESIZE']]
train_y=train[['CO2EMISSIONS']]
train_x_poly = poly.fit_transform(train_x)
test_x = test[['ENGINESIZE']]
test_y = test[['CO2EMISSIONS']]
test_x_poly = poly.fit_transform(test_x)
lm.fit(train_x_poly, train_y)
test_y_hat=lm.predict(test_x_poly)

width = 12
height = 10

plt.figure(figsize=(width, height))

ax1 = sns.distplot(lrdf['CO2EMISSIONS'], hist=False, color="r", label="Actual Value")
sns.distplot(test_y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)

plt.title('Actual vs Fitted Values for CO2 Emission')
plt.xlabel('CO2 Emissions')
plt.ylabel('')

plt.show()
plt.close()
```

```
D:\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequenc
e for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Ridge Regression

We will use Ridge Regression to improve the model. Ridge Regression controls the degree of polynomial regression by the parameter Alpha. Ridge Regression has the following benefits: -reduce the multicollinearity of endogenous variables in models -reduce model complexity and prevent over-fitting which may result from simple linear regression.

In [123]:
```
#prepare data
x_data = lrdf[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']]
y_data= lrdf[['CO2EMISSIONS']]
train_x, test_x, train_y, test_y = train_test_split(x_data, y_data, test_size=0.2, random_state=0)
poly=PolynomialFeatures(degree=2)
lm=LinearRegression()
```

In [117]:
```
#import packages
from sklearn.linear_model import Ridge
```

Let's create a Ridge regression object, setting the regularization parameter to 0.1

In [118]:
```
RigeModel=Ridge(alpha=0.1)
```

In [124]:
```
train_x_poly= poly.fit_transform(train_x)
test_x_poly = poly.fit_transform(test_x)
```

In [125]:
```
RigeModel.fit(train_x_poly, train_y)
yhat = RigeModel.predict(test_x_poly)
```

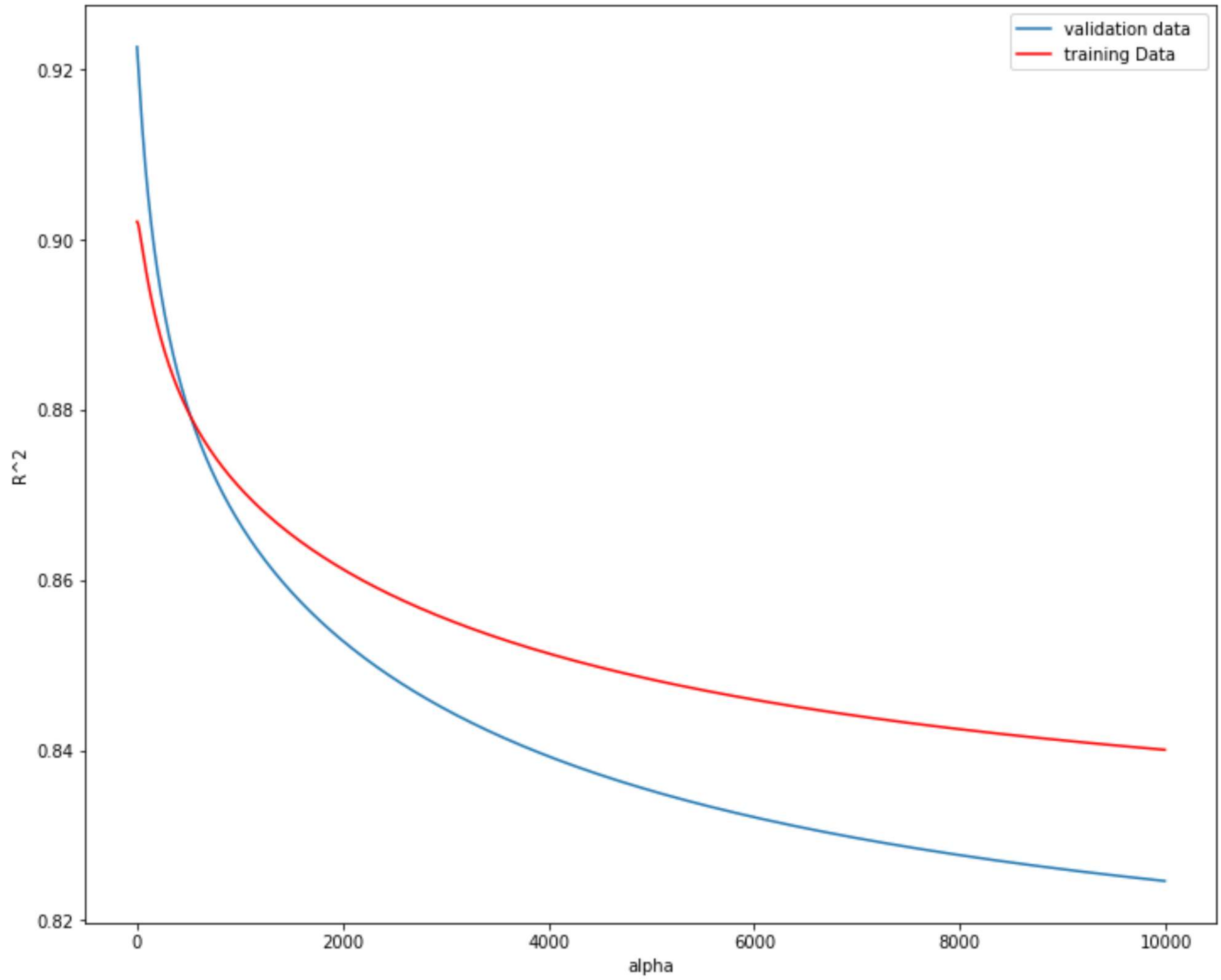
In [127]:
```
print('predicted:', yhat[0:4])
print('test set :', test_y[0:4].values)
```

```
predicted: [[346.34060871]
 [211.77465377]
 [224.66033349]
 [221.681592  ]]
test set : [[356]
 [209]
 [230]
 [212]]
```

In [128]:
```
Rsqu_test = []
Rsqu_train = []
dummy1 = []
ALFA = 10 * np.array(range(0,1000))
for alfa in ALFA:
    RigeModel = Ridge(alpha=alfa)
    RigeModel.fit(train_x_poly, train_y)
    Rsqu_test.append(RigeModel.score(test_x_poly, test_y))
    Rsqu_train.append(RigeModel.score(train_x_poly, train_y))
```

In [129]:
```
#plot for different alfa
width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(ALFA,Rsqu_test, label='validation data  ')
plt.plot(ALFA,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
```

Out[129]: <matplotlib.legend.Legend at 0x20c72115c18>



The blue line represents the R^2 of the test data, and the red line represents the R^2 of the training data. The x-axis represents the different values of Alpha. The red line in represents the R^2 of the test data, as Alpha increases the R^2 decreases; therefore as Alpha increases the model performs worse on the test data. The blue line represents the R^2 on the validation data, as the value for Alpha increases the R^2 decreases.

Lower alpha value (lower polynomial degree) will yield better R^2. Let's find the exact alpha value and then determine the model.

In [189]:
```
x_data = lrdf[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']]
y_data= lrdf[['CO2EMISSIONS']]
train_x, test_x, train_y, test_y = train_test_split(x_data, y_data, test_size=0.2, random_state=0)
```

In [ ]:
```
alphas = 10**np.linspace(10,-2,100)*0.5
alphas
```

In [190]:
```
ridge = Ridge(normalize = True)
coefs = []

for a in alphas:
    ridge.set_params(alpha = a)
    ridge.fit(train_x, train_y)
    coefs.append(ridge.coef_)

np.shape(coefs)
```

Out[190]: (100, 1, 3)

In [181]:
```
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
```

In [191]:
```
#obtian the best alpha by cross-vaildation
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(train_x, train_y)
ridgecv.alpha_
```

Out[191]: 0.011550648500041579

Therefore, we see that the value of alpha that results in the smallest cross-validation error is 0.012.

In [192]:
```
#obtain the RMSE
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(train_x, train_y)
mean_squared_error(test_y, ridge.predict(test_x))
```

Out[192]: 593.7935827354731

In [195]:
```
#Use the optimized alpha for the new ridge regression model using all data
ridge_final = Ridge(alpha=0.012)
ridge_final.fit(x_data, y_data)
ridge_final.coef_
```

Out[195]: array([[10.85494779, 7.51636562, 9.5956638]])

Is it necessary to use Ridge Regression than Linear Regression? We compare the MSE. The MSE obtained from the linear regression model is 737.28, however MSE from Ridge Regression is much better: 593.79. So in this case, we use the Ridge Regression model.