Assignment 7
CSCI E 63 - Big Data Analytics

Problem 1:
Attached file `auto_mpg_original.csv` contains a set of data on automobile characteristics and fuel consumption. File `auto_mpg_description.csv` contains the description of the data. Import data into Spark. Randomly select 10-20% of you data for testing and use remaining data for training. Find all null values in all numerical columns. Replace nulls, if any, with average values for respective columns using Spark Data Frame API.

Answer:

→ Import data into Spark

```
>>> from pyspark.sql.types import StructType, TimestampType, StringType, FloatType
>>> autoSchema = StructType().add("mpg", FloatType()) \
      .add("cylinders", FloatType()) \
      .add("displacement", FloatType()) \
      .add("horsepower", FloatType()) \
      .add("weight",FloatType()) \
      .add("acceleration", FloatType()) \
      .add("modelYear", FloatType()) \
      .add("origin", FloatType()) \
      .add("carName", StringType())
>>> path = "file:///home/kbhandarkar/PythonProjects/Assignment7/auto_mpg_original.csv"
>>> autoDF = spark.read \
   .schema(autoSchema) \
   .option("header", "true") \
   .option("mode", "DROPMALFORMED") \
   .csv(path)
```

→ Replace nulls, if any, with average values for respective columns using Spark Data Frame API.

```
>>> from pyspark.ml.feature import Imputer
>>> cols = ['mpg','cylinders','displacement','horsepower','weight','acceleration','modelYear','origin']
>>> imputer = Imputer(
      inputCols = cols,
      outputCols=["{}_imputed".format(c) for c in cols]
)
>>> imputer.fit(autoDF).transform(autoDF)
>>> print "autoDF count: {}".format(autoDF.count())
>>> autoDF.show()
```

autoDF count: 405

mpg	cylinders	displacement	horsepower	weight	acceleration	modelYear	origin	carName
15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0	buick
18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0	plymouth
16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0	amc
17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0	ford
15.0	8.0	429.0	198.0	4341.0	10.0	70.0	1.0	ford
14.0	8.0	454.0	220.0	4354.0	9.0	70.0	1.0	chevrolet
14.0	8.0	440.0	215.0	4312.0	8.5	70.0	1.0	plymouth
14.0	8.0	455.0	225.0	4425.0	10.0	70.0	1.0	pontiac
15.0	8.0	390.0	190.0	3850.0	8.5	70.0	1.0	amc
26.0	4.0	133.0	115.0	3090.0	17.5	70.0	2.0	citroen
19.0	8.0	350.0	165.0	4142.0	11.5	70.0	1.0	chevrolet
27.0	8.0	351.0	153.0	4034.0	11.0	70.0	1.0	ford
28.0	8.0	383.0	175.0	4166.0	10.5	70.0	1.0	plymouth
25.0	8.0	360.0	175.0	3850.0	11.0	70.0	1.0	amc
15.0	8.0	383.0	170.0	3563.0	10.0	70.0	1.0	dodge
14.0	8.0	340.0	160.0	3609.0	8.0	70.0	1.0	plymouth
16.0	8.0	302.0	140.0	3353.0	8.0	70.0	1.0	ford
15.0	8.0	400.0	150.0	3761.0	9.5	70.0	1.0	chevrolet
14.0	8.0	455.0	225.0	3086.0	10.0	70.0	1.0	buick
24.0	4.0	113.0	95.0	2372.0	15.0	70.0	3.0	toyota

only showing top 20 rows

→ Randomly select 10-20% of you data for testing and use remaining data for training.

```
>>> testDF = autoDF.sample(False, 0.2, 42)
>>> print "testDF count: {}".format(testDF.count())
>>> testDF.show()

>>> trainDF = autoDF.subtract(testDF)
>>> print "trainDF count: {}".format(trainDF.count())
>>> trainDF.show()
```

testDF count: 83

mpg	cylinders	displacement	horsepower	weight	acceleration	modelYear	origin	carName
14.0	8.0	440.0	215.0	4312.0	8.5	70.0	1.0	plymouth
14.0	8.0	455.0	225.0	4425.0	10.0	70.0	1.0	pontiac
19.0	8.0	350.0	165.0	4142.0	11.5	70.0	1.0	chevrolet
27.0	8.0	351.0	153.0	4034.0	11.0	70.0	1.0	ford
14.0	8.0	455.0	225.0	3086.0	10.0	70.0	1.0	buick
24.0	4.0	113.0	95.0	2372.0	15.0	70.0	3.0	toyota
22.0	6.0	198.0	95.0	2833.0	15.5	70.0	1.0	plymouth
26.0	4.0	97.0	46.0	1835.0	20.5	70.0	2.0	volkswagen
25.0	4.0	113.0	95.0	2228.0	14.0	71.0	3.0	toyota
15.0	4.0	97.0	48.0	1978.0	20.0	71.0	2.0	volkswagen
18.0	6.0	232.0	100.0	3288.0	15.5	71.0	1.0	amc
14.0	8.0	400.0	175.0	4464.0	11.5	71.0	1.0	pontiac
14.0	8.0	351.0	153.0	4154.0	13.5	71.0	1.0	ford
14.0	8.0	318.0	150.0	4096.0	13.0	71.0	1.0	plymouth
13.0	8.0	400.0	170.0	4746.0	12.0	71.0	1.0	ford
30.0	4.0	79.0	70.0	2074.0	19.5	71.0	2.0	peugeot
30.0	4.0	88.0	76.0	2065.0	14.5	71.0	2.0	fiat
12.0	8.0	350.0	160.0	4456.0	13.5	72.0	1.0	oldsmobile
22.0	4.0	121.0	76.0	2511.0	18.0	72.0	2.0	volkswagen
13.0	8.0	350.0	175.0	4100.0	13.0	73.0	1.0	buick

only showing top 20 rows

trainDF count: 321

mpg	cylinders	displacement	horsepower	weight	acceleration	modelYear	origin	carName
22.0	4.0	108.0	94.0	2379.0	16.5	73.0	3.0	datsun
39.0	4.0	86.0	64.0	1875.0	16.4	81.0	1.0	plymouth
14.0	8.0	318.0	150.0	4237.0	14.5	73.0	1.0	plymouth
14.0	8.0	302.0	140.0	4638.0	16.0	74.0	1.0	ford
25.0	4.0	90.0	71.0	2223.0	16.5	75.0	2.0	volkswagen
20.5	6.0	231.0	105.0	3425.0	16.9	77.0	1.0	buick
26.4	4.0	140.0	88.0	2870.0	18.1	80.0	1.0	ford
16.0	6.0	258.0	110.0	3632.0	18.0	74.0	1.0	amc
18.0	6.0	199.0	97.0	2774.0	15.5	70.0	1.0	amc
27.4	4.0	121.0	80.0	2670.0	15.0	79.0	1.0	amc
26.0	4.0	97.0	78.0	2300.0	14.5	74.0	2.0	opel
13.0	8.0	318.0	150.0	3940.0	13.2	76.0	1.0	plymouth
30.0	4.0	111.0	80.0	2155.0	14.8	77.0	1.0	buick
29.8	4.0	89.0	62.0	1845.0	15.3	80.0	2.0	vokswagen
15.0	6.0	258.0	110.0	3730.0	19.0	75.0	1.0	amc
29.0	4.0	97.0	78.0	1940.0	14.5	77.0	2.0	volkswagen
26.0	4.0	133.0	115.0	3090.0	17.5	70.0	2.0	citroen
22.0	4.0	122.0	86.0	2395.0	16.0	72.0	1.0	ford
20.0	8.0	262.0	110.0	3221.0	13.5	75.0	1.0	chevrolet
20.0	4.0	130.0	102.0	3150.0	15.7	76.0	2.0	volvo

only showing top 20 rows

Problem 2:

Look initially at two variables in the data set from the previous problem: the `horsepower` and the `mpg` (miles per gallon). Treat `mpg` as a feature and `horsepower` as the target variable (label). Use MLlib linear regression to identify the model for the relationship. Use the test data to illustrate accuracy of the linear regression model and its ability to predict the relationship. Calculate two standard measures of model accuracy. Create a diagram using any technique of convenience to presents the model (straight ls line), and the original test data. Please label your axes and use different colors for original data and predicted data.

Answer:

→ For ease, create a new DF with these two variables

```
>>> p2DF = autoDF.select('mpg','horsepower')
>>> p2DF.show()
```

```
+----+----------+
| mpg|horsepower|
+----+----------+
|15.0|     165.0|
|18.0|     150.0|
|16.0|     150.0|
|17.0|     140.0|
|15.0|     198.0|
|14.0|     220.0|
|14.0|     215.0|
|14.0|     225.0|
|15.0|     190.0|
|26.0|     115.0|
|19.0|     165.0|
|27.0|     153.0|
|28.0|     175.0|
|25.0|     175.0|
|15.0|     170.0|
|14.0|     160.0|
|16.0|     140.0|
|15.0|     150.0|
|14.0|     225.0|
|24.0|      95.0|
+----+----------+
only showing top 20 rows
```

→ Create Feature Vectors for Linear Model

```
>>> records = p2DF.rdd.map(list)
>>> first = records.first()
>>> print first
[15.0, 165.0]
```

```
>>> from pyspark.mllib.regression import LabeledPoint
>>> import numpy as np

>>> def extract_features(record):
...     num_vec = np.array([float(field) for field in record[0:1]])
...     return num_vec
...
```

The extract_features function runs through each column in the row of data. The numeric vector num_vec is created directly first by converting the data to floating point numbers and wrapping these in a numpy array.

```
>>> data = records.map(lambda r: LabeledPoint(r[-1],extract_features(r)))
```

Class LabeledPoint contains a label, horsepower, and the feature value, mpg, corresponding to that label.

→ Use MLlib linear regression to identify the model for the relationship

```
>>> from pyspark.mllib.regression import LinearRegressionWithSGD
>>> model = LinearRegressionWithSGD.train(data, iterations=200,step=0.01, intercept=False)
```

Check the type and details of the model

```
>>> type(model)
<class 'pyspark.mllib.regression.LinearRegressionModel'>
>>> model
(weights=[3.65113153658], intercept=0.0)
```

→ Use the test data to illustrate accuracy of the linear regression model and its ability to predict the relationship

```
>>> true_vs_predicted = data.map(lambda p: (p.label, model. predict(p.features)))
>>> print "Linear Model predictions: " + str(true_vs_predicted.take(10))
```

```
Linear Model predictions: [(165.0, 54.766973048729021), (150.0, 65.720367658474828), (150.0, 58.41
8104585310957), (140.0, 62.069236121892892), (198.0, 54.766973048729021), (220.0, 51.1158415121470
85), (215.0, 51.115841512147085), (225.0, 51.115841512147085), (190.0, 54.766973048729021), (115.0
, 94.929419951130299)]
```

Assignment 7
CSCI E 63 - Big Data Analytics

→ Calculate two standard measures of model accuracy.

1. Mean Squared Error

Define the mean squared error function as:

```
>>> def squared_error(actual, pred):
...     return (pred - actual)**2
...
```

Use the function as:

```
>>> mse = true_vs_predicted.map(lambda (t, p): squared_error(t, p)).mean()
>>> print "Linear Model - Mean Squared Error: %2.4f" % mse
```

```
Linear Model - Mean Squared Error: 4319.5031
```

2. Mean Absolute Error

Define the mean absolute error function as

```
>>> def abs_error(actual, pred):
...     return np.abs(pred - actual)
...
```

Use the function as:

```
>>> mae = true_vs_predicted.map(lambda (t, p): abs_error(t, p)).mean()
>>> print "Linear Model - Mean Absolute Error: %2.4f" % mae
```

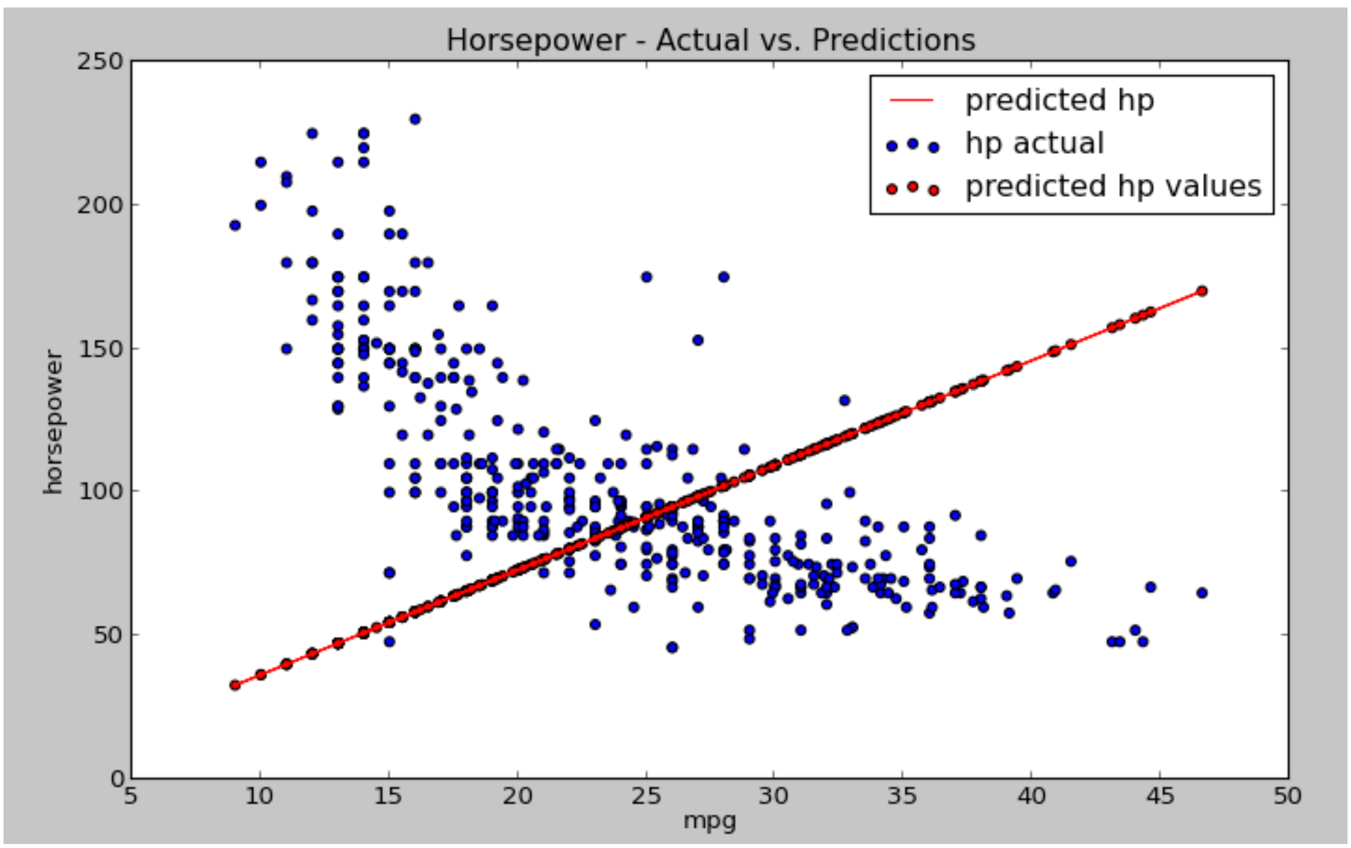
```
Linear Model - Mean Absolute Error: 51.4460
```

→ Create a diagram using any technique of convenience to present the model (straight ls line), and the original test data.

```
$ sudo pip install pandas
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> tvp = data.map(lambda p: (float(p.label), float(model.predict(p.features)),
float(p.features[0]))).toDF().toPandas()
```

Assignment 7
CSCI E 63 – Big Data Analytics

```
>>> tvp.columns = ['horsepower', 'predicted horsepower', 'mpg']
>>> plt.figure(1, figsize=(10,6))
<matplotlib.figure.Figure object at 0x30e7950>
>>> plt.scatter(tvp['mpg'], tvp['horsepower'], c='b', label='hp actual')
<matplotlib.collections.PathCollection object at 0x427b490>
>>> plt.plot(tvp['mpg'], tvp['predicted horsepower'], c='r',label='predicted hp')
[<matplotlib.lines.Line2D object at 0x427bd10>]
>>> plt.scatter(tvp['mpg'], tvp['predicted horsepower'], c='r', label='predicted hp values')
<matplotlib.collections.PathCollection object at 0x42d9910>
>>> plt.xlabel('mpg')
<matplotlib.text.Text object at 0x26f4250>
>>> plt.ylabel('horsepower')
<matplotlib.text.Text object at 0x3648490>
>>> plt.title('Horsepower - Actual vs. Predictions')
<matplotlib.text.Text object at 0x3655e50>
>>> plt.legend()
<matplotlib.legend.Legend object at 0x42d9f50>
>>> plt.show()
```



Assignment 7
CSCI E 63 - Big Data Analytics

Problem 3:

Consider attached file `Bike-Sharing-Dataset.zip`. This is the bike set discussed in class. Do not use all columns of the data set. Retain the following variables: `season, yr, mnth, hr, holiday, weekday, workingday, weathersit, temp, atemp, hum, windspeed, cnt`. Discard others. Regard `cnt` as the target variable and all other variables as features. Please note that some of those are categorical variables. Identify categorical variables and use 1-of-k binary encoding for those variables. If there are any null values in numerical columns, replace those with average values for those columns using Spark DataFrame API. Train your model using `LinearRegressionSGD` method. Use test data (15% of all) to assess the quality of prediction for `cnt` variable. Calculate at least two performance metrics of your model.

Answer:

→ Inspect and read data

Observation: There are no nulls in the data

```
>>> hoursdata = sc.textFile("file:///home/kbhandarkar/PythonProjects/Assignment7/BikeData/
hour_noHeader.csv")
>>> hoursdata.count()
17379
>>> hoursRecord = hoursdata.map(lambda x:x.split(","))
>>> hoursRecord.take(5)
[[u'1', u'2011-01-01', u'1', u'0', u'1', u'0', u'0', u'6', u'0', u'1', u'0.24', u'0.2879', u'0.81', u'0', u'3', u'13',
u'16'], [u'2', u'2011-01-01', u'1', u'0', u'1', u'1', u'0', u'6', u'0', u'1', u'0.22', u'0.2727', u'0.8', u'0', u'8', u'32',
u'40'], [u'3', u'2011-01-01', u'1', u'0', u'1', u'2', u'0', u'6', u'0', u'1', u'0.22', u'0.2727', u'0.8', u'0', u'5', u'27',
u'32'], [u'4', u'2011-01-01', u'1', u'0', u'1', u'3', u'0', u'6', u'0', u'1', u'0.24', u'0.2879', u'0.75', u'0', u'3', u'10',
u'13'], [u'5', u'2011-01-01', u'1', u'0', u'1', u'4', u'0', u'6', u'0', u'1', u'0.24', u'0.2879', u'0.75', u'0', u'0', u'1',
u'1']]
```

→ Some of the variables are categorical variables. Identify categorical variables and use 1-of-k binary encoding for those variables.

```
>>> hoursRecord.cache()
PythonRDD[257] at RDD at PythonRDD.scala:48
>>> def get_mapping(rdd,idx):
...     return rdd.map(lambda fields :fields[idx]).distinct().zipWithIndex().collectAsMap()
...

>>> print "Mapping of column#3:%s" % get_mapping(hoursRecord,2)
Mapping of column#3:{u'1': 0, u'3': 1, u'2': 2, u'4': 3}
```

Assignment 7
CSCI E 63 - Big Data Analytics

```
>> mappings = [get_mapping(hoursRecord,i) for i in range (2,10)]
>>> cat_len = sum(map(len,mappings))
>>> num_len =len(hoursRecord.first()[10:14])
>>> total_len = num_len +cat_len

>>> print "Categorical feature Vector length %d" %cat_len
Categorical feature Vector length 57
>>> print "Numerical  feature Vector length %d" %num_len
Numerical  feature Vector length 4
>>> print "Total  feature Vector length %d" %num_len
Total  feature Vector length 4
```

→ Create feature vector for Linear Model

```
>>> from pyspark.mllib.regression import LabeledPoint
>>> import numpy as np
>>> def extract_features(record):
...     cat_vec = np.zeros(cat_len)
...     i = 0
...     step = 0
...     for field in record[2:10]:
...             m = mappings[i]
...             idx = m[field]
...             cat_vec[idx + step] = 1
...             i = i + 1
...             step = step + len(m)
...             num_vec = np.array([float(field) for field in record[10:14]])
...     return np.concatenate((cat_vec, num_vec))
…
>>> def extract_label(record):
...     return float(record[-1])
...

>>> data = hoursRecord.map(lambda r:LabeledPoint(extract_label(r),extract_features(r)))
>>> first_point = data.first()
>>> print first_point
(16.0,
[1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,0.0,0.
0,0.0,0.24,0.2879,0.81,0.0]
```

Assignment 7
CSCI E 63 - Big Data Analytics

```
>>> print "Label: " + str(first_point.label)
Label: 16.0

>>> print "Linear Model feature vector:\n" + str(first_point.features)
Linear Model feature vector:
[1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,0.
0,0.0,0.24,0.2879,0.81,0.0]

>>> print "Linear Model feature vector length: " + str(len(first_point.features))
Linear Model feature vector length: 61
```

→ Train your model using `LinearRegressionSGD` method.

Create the training and test set

```
>>> data_with_idx = data.zipWithIndex().map(lambda (k, v): (v, k))
>>> test = data_with_idx.sample(False, 0.15, 42)
>>> train = data_with_idx.subtractByKey(test)
>>> train_data = train.map(lambda (idx, p): p)
>>> test_data = test.map(lambda (idx, p) : p)
>>> train_size = train_data.count()
>>> test_size = test_data.count()

>>> print "Training data size: %d" % train_size
Training data size: 14736
>>> print "Test data size: %d" % test_size
Test data size: 2643
```

Train the linear model with the training data

```
>>> from pyspark.mllib.regression import LinearRegressionWithSGD
>>> from pyspark.mllib.tree import DecisionTree
>>> from pyspark.mllib.linalg import SparseVector
>>> from pyspark.mllib.regression import LabeledPoint
>>> linear_model = LinearRegressionWithSGD.train(train_data, iterations=200,step=0.01,
intercept=False)
>>> linear_model
```

Assignment 7
CSCI E 63 - Big Data Analytics

(weights=[2.99091837716,10.7888070755,9.06558563146,8.3591908039,21.2691502866,9.93535160142,2.43153425318,3.3442283451,1.67082789441,0.65928774301,1.95142383049,0.995287781345,3.35367682907,2.51840970985,3.48846475552,3.60328886077,3.5964047067,3.59166717857,1.71623200899,1.08945308443,0.653961276169,0.170687302806,-0.442213214358,-0.202834137758,-0.660955925927,-0.559921196529,-0.578935314331,-0.700340684895,1.56396487565,0.0542120465756,1.64296023011,3.11903941345,1.52712360546,1.15446316802,2.01588382638,2.02977213495,2.0303170752,1.8761278404,4.3804936924,2.70391147221,2.65903917098,3.96206013763,0.821836442371,30.3826654456,4.2928681158,4.20524534667,4.41659848765,4.30447644886,4.67384307067,4.59818501415,4.71328540421,21.4641346948,9.74036719325,22.5389263541,1.08042972694,7.58425214943,0.000893657519407,19.0154640444,17.9378611452,16.551265826,6.49635721505], intercept=0.0)

→ Use test data (15% of all) to assess the quality of prediction for `cnt` variable.

```
>>> true_vs_predicted = test_data.map(lambda p: (p.label, linear_model.predict(p.features)))
>>> print "Linear Model predictions: " + str(true_vs_predicted.take(5))
Linear Model predictions: [(32.0, 102.71695040712191), (14.0, 107.51901747997432), (93.0, 99.816975588765175), (67.0, 102.04972685675229), (8.0, 90.263802576690622)]
```

→ Calculate at least two performance metrics of your model.

1. Mean Squared Error

Define the mean squared error function as:

```
>>> def squared_error(actual, pred):
...     return (pred - actual)**2
...
```

Use the function as:

```
>>> mse = true_vs_predicted.map(lambda (t, p): squared_error(t, p)).mean()
>>> print "Linear Model - Mean Squared Error: %2.4f" % mse
```

```
Linear Model - Mean Squared Error: 34083.2064
```

2. Mean Absolute Error

Define the mean absolute error function as

```
>>> def abs_error(actual, pred):
...    return np.abs(pred - actual)
...
```

Use the function as:
>>> mae = true_vs_predicted.map(lambda (t, p): abs_error(t, p)).mean()
>>> print "Linear Model - Mean Absolute Error: %2.4f" % mae

```
Linear Model - Mean Absolute Error: 132.1044
```

Assignment 7
CSCI E 63 - Big Data Analytics

Problem 4:
Use a Decision Tree model to predict mpg values in auto_mpg_original.txt data. Assess accuracy of your prediction using at least two performance metrics.

Answer:

→ Start with the same data frame from Problem1

```
>>> autoDF.show()
```

```
+----+---------+------------+----------+------+------------+---------+------+---------+
| mpg|cylinders|displacement|horsepower|weight|acceleration|modelYear|origin|  carName|
+----+---------+------------+----------+------+------------+---------+------+---------+
|15.0|      8.0|       350.0|     165.0|3693.0|        11.5|     70.0|   1.0|    buick|
|18.0|      8.0|       318.0|     150.0|3436.0|        11.0|     70.0|   1.0| plymouth|
|16.0|      8.0|       304.0|     150.0|3433.0|        12.0|     70.0|   1.0|      amc|
|17.0|      8.0|       302.0|     140.0|3449.0|        10.5|     70.0|   1.0|     ford|
|15.0|      8.0|       429.0|     198.0|4341.0|        10.0|     70.0|   1.0|     ford|
|14.0|      8.0|       454.0|     220.0|4354.0|         9.0|     70.0|   1.0|chevrolet|
|14.0|      8.0|       440.0|     215.0|4312.0|         8.5|     70.0|   1.0| plymouth|
|14.0|      8.0|       455.0|     225.0|4425.0|        10.0|     70.0|   1.0|  pontiac|
|15.0|      8.0|       390.0|     190.0|3850.0|         8.5|     70.0|   1.0|      amc|
|26.0|      4.0|       133.0|     115.0|3090.0|        17.5|     70.0|   2.0|  citroen|
|19.0|      8.0|       350.0|     165.0|4142.0|        11.5|     70.0|   1.0|chevrolet|
|27.0|      8.0|       351.0|     153.0|4034.0|        11.0|     70.0|   1.0|     ford|
|28.0|      8.0|       383.0|     175.0|4166.0|        10.5|     70.0|   1.0| plymouth|
|25.0|      8.0|       360.0|     175.0|3850.0|        11.0|     70.0|   1.0|      amc|
|15.0|      8.0|       383.0|     170.0|3563.0|        10.0|     70.0|   1.0|    dodge|
|14.0|      8.0|       340.0|     160.0|3609.0|         8.0|     70.0|   1.0| plymouth|
|16.0|      8.0|       302.0|     140.0|3353.0|         8.0|     70.0|   1.0|     ford|
|15.0|      8.0|       400.0|     150.0|3761.0|         9.5|     70.0|   1.0|chevrolet|
|14.0|      8.0|       455.0|     225.0|3086.0|        10.0|     70.0|   1.0|    buick|
|24.0|      4.0|       113.0|      95.0|2372.0|        15.0|     70.0|   3.0|   toyota|
+----+---------+------------+----------+------+------------+---------+------+---------+
only showing top 20 rows
```

```
>>> p4DF = autoDF.select('horsepower','mpg')
>>> p4DF.show()
```

```
+----------+----+
|horsepower| mpg|
+----------+----+
|     165.0|15.0|
|     150.0|18.0|
|     150.0|16.0|
|     140.0|17.0|
|     198.0|15.0|
|     220.0|14.0|
|     215.0|14.0|
|     225.0|14.0|
|     190.0|15.0|
|     115.0|26.0|
|     165.0|19.0|
|     153.0|27.0|
|     175.0|28.0|
|     175.0|25.0|
|     170.0|15.0|
|     160.0|14.0|
|     140.0|16.0|
|     150.0|15.0|
|     225.0|14.0|
|      95.0|24.0|
+----------+----+
only showing top 20 rows
```

Assignment 7
CSCI E 63 - Big Data Analytics

→ Create Feature Vectors for Regression Tree

```
>>> records = p4DF.rdd.map(list)
>>> first = records.first()
>>> print first
[165.0, 15.0]

>>> from pyspark.mllib.regression import LabeledPoint
>>> import numpy as np

>>> def extract_features(record):
...     num_vec = np.array([float(field) for field in record[0:1]])
...     return num_vec
...
```

The extract_features function runs through each column in the row of data. The numeric vector num_vec is created directly first by converting the data to floating point numbers and wrapping these in a numpy array.

```
>>> data = records.map(lambda r: LabeledPoint(r[-1],extract_features(r)))
```

Class LabeledPoint contains a label, mpg, and the feature value, horesepower, corresponding to that label.

→ Use MLlib Decision Tree model to identify the model for the relationship

```
>>> model = DecisionTree.trainRegressor(data,{})

>>> preds = model.predict(data.map(lambda p: p.features))
>>> actual = data.map(lambda p: p.label)
>>> true_vs_predicted_dt = actual.zip(preds)

>>> print "Decision Tree predictions: " + str(true_vs_predicted_dt.take(5))
Decision Tree predictions: [(15.0, 15.198039223166074), (18.0, 15.198039223166074), (16.0, 15.198039223166074), (17.0, 15.881250023841858), (15.0, 12.76923076923077)]
>>> print "Decision Tree depth: " + str(model.depth())
Decision Tree depth: 5
>>> print "Decision Tree number of nodes: " + str(model.numNodes())
Decision Tree number of nodes: 45
```

Assignment 7
CSCI E 63 - Big Data Analytics

→ Assess accuracy of your prediction using at least two performance metrics.

1. Mean Squared Error

Define the mean squared error function as:

```
>>> def squared_error(actual, pred):
...     return (pred - actual)**2
...
```

Use the function as:

```
>>> mse = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
>>> print "Decision Tree Model - Mean Squared Error: %2.4f" % mse
```

```
Decision Tree Model - Mean Squared Error: 18.1103
```

2. Mean Absolute Error

Define the mean absolute error function as

```
>>> def abs_error(actual, pred):
...    return np.abs(pred - actual)
...
```

Use the function as:

```
>>> mae = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
>>> print "Decision Tree Model - Mean Absolute Error: %2.4f" % mae
```

```
Decision Tree Model - Mean Absolute Error: 3.1453
```