Create a dataset "C:\Users\DELL\Desktop\color.csv"

....

ID	color
1	red
2	blue
3	orange
4	white
5	red
6	orange
7	red
8	white
9	red

....

```
---> pyspark
>>> from pyspark.sql import SparkSession


>>> spark = SparkSession.builder.getOrCreate()


>>> data = spark.read.csv('C:\Users\DELL\Desktop\color.csv',header=True,inferSchema=True)


>>> data.show()
```

install numpy and setuptools using:

```
pip install numpy
pip install setuptools==58.0.4
```
(use this as newer version of pythons doesn't contains distutils)

-----------------------> encoding of catogorical data (data pre_processing) .

```
>>> from pyspark.ml.feature import StringIndexer
>>> indexer = StringIndexer(inputCol = "color",outputCol = "color_indexed")

>>> indexer_model = indexer.fit(data)
>>> indexer_data = indexer_model.transform(data)
>>> indexer_data.show()
+---+------+-------------+
| ID| color|color_indexed|
+---+------+-------------+
|  1|   red|          0.0|
|  2|  blue|          3.0|
|  3|orange|          1.0|
|  4| white|          2.0|
|  5|   red|          0.0|
|  6|orange|          1.0|
|  7|   red|          0.0|
|  8| white|          2.0|
|  9|   red|          0.0|
+---+------+-------------+
```

....

New dataset --> E:/downloads/wine.data

feature scaling --> Done for data pre-processing

to remove one coloume donination over another

methods--

1.>normalization

2.>standardization

....

```
>>> from pyspark.sql import SparkSession

>>> spark = SparkSession.builder.getOrCreate()

>>> data = spark.read.csv('C:/Users/Dell/Desktop/wine.data',header=False,inferSchema=True)

>>> data.show()
+---+-----+----+----+----+---+----+----+----+----+----+----+----+----+
|_c0|  _c1| _c2| _c3| _c4|_c5| _c6| _c7| _c8| _c9|_c10|_c11|_c12|_c13|
+---+-----+----+----+----+---+----+----+----+----+----+----+----+----+
|  1|14.23|1.71|2.43|15.6|127| 2.8|3.06|0.28|2.29|5.64|1.04|3.92|1065|
|  1| 13.2|1.78|2.14|11.2|100|2.65|2.76|0.26|1.28|4.38|1.05| 3.4|1050|
|  1|13.16|2.36|2.67|18.6|101| 2.8|3.24| 0.3|2.81|5.68|1.03|3.17|1185|
|  1|14.37|1.95| 2.5|16.8|113|3.85|3.49|0.24|2.18| 7.8|0.86|3.45|1480|
|  1|13.24|2.59|2.87|21.0|118| 2.8|2.69|0.39|1.82|4.32|1.04|2.93| 735|
|  1| 14.2|1.76|2.45|15.2|112|3.27|3.39|0.34|1.97|6.75|1.05|2.85|1450|
|  1|14.39|1.87|2.45|14.6| 96| 2.5|2.52| 0.3|1.98|5.25|1.02|3.58|1290|
|  1|14.06|2.15|2.61|17.6|121| 2.6|2.51|0.31|1.25|5.05|1.06|3.58|1295|
|  1|14.83|1.64|2.17|14.0| 97| 2.8|2.98|0.29|1.98| 5.2|1.08|2.85|1045|
```

....

....

ml.feature --> to put all col in one list

```
>>> from pyspark.ml.feature import VectorAssembler

>>> assembler = VectorAssembler(inputCols=data.columns[1:],outputCol="feature")

>>> data1 = assembler.transform(data)
>>> data1.show()

>>> from pyspark.ml.feature import StandardScaler

>>> scaler = StandardScaler(inputCol="feature",outputCol="scaled-features")

>>> scaler_model=scaler.fit(data1)



>>> scaled_data = scaler_model.transform(data1)

>>> scaled_data.show()

+---+-----+----+----+----+---+----+----+----+----+----+----+----+----+--------------------+--------------------+
|_c0|  _c1| _c2| _c3| _c4|_c5| _c6| _c7| _c8| _c9|_c10|_c11|_c12|_c13|             feature|
scaled-features|
+---+-----+----+----+----+---+----+----+----+----+----+----+----+----+--------------------+--------------------+
|  1|14.23|1.71|2.43|15.6|127|
2.8|3.06|0.28|2.29|5.64|1.04|3.92|1065|[14.23,1.71,2.43,...|[17.5283750084766...|
|  1| 13.2|1.78|2.14|11.2|100|2.65|2.76|0.26|1.28|4.38|1.05|
3.4|1050|[13.2,1.78,2.14,1...|[16.2596310690015...|
|  1|13.16|2.36|2.67|18.6|101| 2.8|3.24|
0.3|2.81|5.68|1.03|3.17|1185|[13.16,2.36,2.67,...|[16.2103594597015...|
|  1|14.37|1.95| 2.5|16.8|113|3.85|3.49|0.24|2.18|
7.8|0.86|3.45|1480|[14.37,1.95,2.5,1...|[17.7008256410266...|
|  1|13.24|2.59|2.87|21.0|118| 2.8|2.69|0.39|1.82|4.32|1.04|2.93|
735|[13.24,2.59,2.87,...|[16.3089026783015...|
|  1|
14.2|1.76|2.45|15.2|112|3.27|3.39|0.34|1.97|6.75|1.05|2.85|1450|[14.2,1.76,2.45,1...|[17.4914
213015016...|
|  1|14.39|1.87|2.45|14.6| 96| 2.5|2.52|
0.3|1.98|5.25|1.02|3.58|1290|[14.39,1.87,2.45,...|[17.7254614456766...|
|  1|14.06|2.15|2.61|17.6|121|
2.6|2.51|0.31|1.25|5.05|1.06|3.58|1295|[14.06,2.15,2.61,...|[17.3189706689516...|

|  1|14.83|1.64|2.17|14.0| 97| 2.8|2.98|0.29|1.98|
5.2|1.08|2.85|1045|[14.83,1.64,2.17,...|[18.2674491479767...|

|  1|13.86|1.35|2.27|16.0|
98|2.98|3.15|0.22|1.85|7.22|1.01|3.55|1045|[13.86,1.35,2.27,...|[17.0726126224516...|
```

………

```
>>> from pyspark.sql import SparkSession

>>> spark = SparkSession.builder.getOrCreate()


>>> data =
spark.read.csv('C:/Users/Dell/Desktop/boston_housing.csv',header=True,inferSchema=True)

>>> data.show()

+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+

|   crim|  zn|indus|chas|  nox|   rm|  age|   dis|rad|tax|ptratio|     b|lstat|medv|

+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+

|0.00632|18.0| 2.31|   0|0.538|6.575| 65.2|  4.09|  1|296|   15.3| 396.9| 4.98|24.0|

|0.02731| 0.0| 7.07|   0|0.469|6.421| 78.9|4.9671|  2|242|   17.8| 396.9| 9.14|21.6|

|0.02729| 0.0| 7.07|   0|0.469|7.185| 61.1|4.9671|  2|242|   17.8|392.83| 4.03|34.7|

|0.03237| 0.0| 2.18|   0|0.458|6.998| 45.8|6.0622|  3|222|   18.7|394.63| 2.94|33.4|

|0.06905| 0.0| 2.18|   0|0.458|7.147| 54.2|6.0622|  3|222|   18.7| 396.9| 5.33|36.2|

|0.02985| 0.0| 2.18|   0|0.458| 6.43| 58.7|6.0622|  3|222|   18.7|394.12| 5.21|28.7|

|0.08829|12.5| 7.87|   0|0.524|6.012| 66.6|5.5605|  5|311|   15.2| 395.6|12.43|22.9|
```

For all should be in same list ---->

```
>>> feature_colums = data.columns[:-1]

>>> from pyspark.ml.feature import VectorAssembler


>>> assembler = VectorAssembler(inputCols=feature_colums,outputCol="features")
>>> data1 = assembler.transform(data)

data1.show()

>>> train,test = data1.randomSplit([0.7,0.3])

>>> from pyspark.ml.regression import LinearRegression


>>> algo = LinearRegression(featuresCol="features",labelCol="medv")

>>> model = algo.fit(train)

23/04/19 12:30:33 WARN Instrumentation: [54ac5900] regParam is zero, which might cause
numerical instability and overfitting.

23/04/19 12:30:34 WARN InstanceBuilder$NativeBLAS: Failed to load implementation
from:dev.ludovic.netlib.blas.JNIBLAS

23/04/19 12:30:34 WARN InstanceBuilder$NativeBLAS: Failed to load implementation
from:dev.ludovic.netlib.blas.ForeignLinkerBLAS

23/04/19 12:30:35 WARN InstanceBuilder$NativeLAPACK: Failed to load implementation
from:dev.ludovic.netlib.lapack.JNILAPACK

>>> evaluation_summary = model.evaluate(test)

>>> evaluation_summary.meanAbsoluteError

3.36122126821546


>>> predictions = model.transform(test)

>>> predictions.select(predictions.columns[13:]).show()

23/04/19 12:33:03 WARN package: Truncated the string representation of a plan since it was too
large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.

+----+--------------------+------------------+

|medv|            features|        prediction|

+----+--------------------+------------------+

|24.0|[0.00632,18.0,2.3...|29.781225163659602|

|32.2|[0.00906,90.0,2.9...|32.566517184269316|

|24.5|[0.01501,80.0,2.0...|  28.0930632817314|

|30.1|[0.01709,90.0,2.0...| 25.57122158254233|

|23.1|[0.0187,85.0,4.15...| 26.05364569399883|

|20.1|[0.01965,80.0,1.7...|21.239133309306453|

|34.7|[0.02729,0.0,7.07...|30.501580642728754|
|30.8|[0.02763,75.0,2.9...| 32.28115224995954|
|26.6|[0.02899,40.0,1.2...|  22.9684207068466|
|34.9|[0.03359,75.0,2.9...| 35.26397183395046|
|19.4|[0.03466,35.0,6.0...|23.996182942724474|
---------------------------------------------------------------------------------------------------------
```

************************DECISION TREE****************************

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

df = spark.read.csv('C:/Users/Dell/Desktop/iris.csv',header=True,inferSchema=True)
df.show()

from pyspark.ml.feature import VectorAssembler
vector_assembler = VectorAssembler(inputCols=df.columns[:-1],outputCol="features")
df1 = vector_assembler.transform(df)
from pyspark.ml.feature import StringIndexer
l_indexer = StringIndexer(inputCol="species", outputCol="labelIndex")
df1 = l_indexer.fit(df1).transform(df1)
(trainingData, testData) = df1.randomSplit([0.7, 0.3])
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
dt = DecisionTreeClassifier(labelCol="labelIndex", featuresCol="features")
model = dt.fit(trainingData)
predictions = model.transform(testData)
predictions.select("prediction", "labelIndex").show(5)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="labelIndex",
predictionCol="prediction",metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print("Test Error = %g " % (1.0 - accuracy))

print(model)
```