Audit Data Analytic Project

```
ds = pd.read_csv('audit_data.csv')
```

```
ds.head()
```

	RECORD_ID	USER_NAME	SECURED_TARGET_ID	SECURED_TARGET_NAME	EVENT_TIME	SECURED_TARGET_TYPE	OSU
0	1	AUDITCOLLECTION_UT	26	targetAgentTargetCol9977	24-FEB-17 09.57.53.000000000 AM	Oracle Database	chlia
1	2	AUDITCOLLECTION_UT	26	targetAgentTargetCol9977	24-FEB-17 09.57.53.000000000 AM	Oracle Database	chlia
2	3	AUDITCOLLECTION_UT	26	targetAgentTargetCol9977	24-FEB-17 09.57.53.000000000 AM	Oracle Database	chlia
3	4	AUDITCOLLECTION_UT	26	targetAgentTargetCol9977	24-FEB-17 09.57.53.000000000 AM	Oracle Database	chlia

Features Information

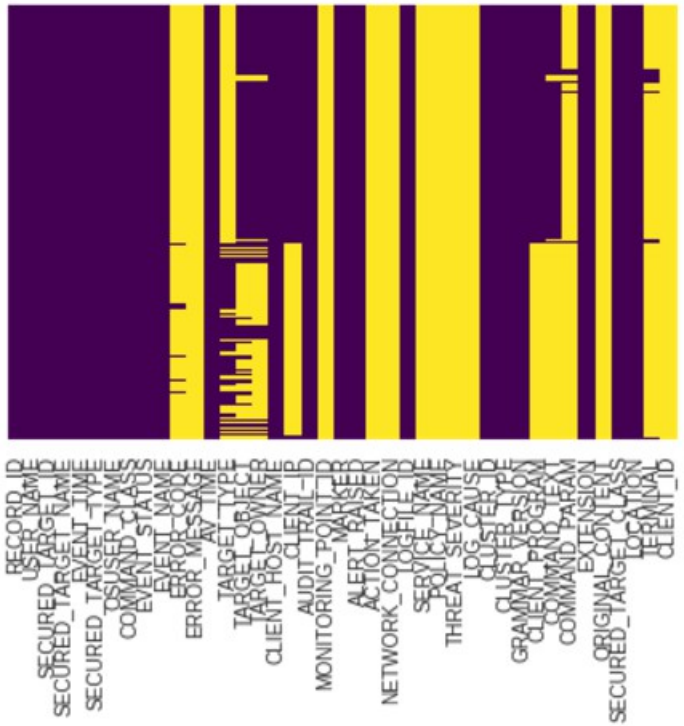
```
In [5]: ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606 entries, 0 to 605
Data columns (total 41 columns):
RECORD_ID                606 non-null int64
USER_NAME                606 non-null object
SECURED_TARGET_ID        606 non-null int64
SECURED_TARGET_NAME      606 non-null object
EVENT_TIME               606 non-null object
SECURED_TARGET_TYPE      606 non-null object
OSUSER_NAME              606 non-null object
COMMAND_CLASS            606 non-null object
EVENT_STATUS             606 non-null object
EVENT_NAME               606 non-null object
ERROR_CODE               13 non-null float64
ERROR_MESSAGE            0 non-null float64
AV_TIME                  606 non-null object
TARGET_TYPE              206 non-null object
TARGET_OBJECT            406 non-null object
TARGET_OWNER             375 non-null object
CLIENT_HOST_NAME         606 non-null object
CLIENT_IP                334 non-null object
AUDIT_TRAIL_ID           606 non-null int64
MONITORING_POINT_ID      0 non-null float64
MARKER                   606 non-null object
ALERT_RAISED             606 non-null int64
ACTION_TAKEN             0 non-null float64
NETWORK_CONNECTION       0 non-null float64
LOGFILE_ID               606 non-null int64
SERVICE_NAME             0 non-null float64
POLICY_NAME              0 non-null float64
THREAT_SEVERITY          0 non-null float64
LOG_CAUSE                0 non-null float64
CLUSTER_ID               606 non-null int64
CLUSTER_TYPE             606 non-null int64
GRAMMAR_VERSION          606 non-null int64
```

Visualize Null Values

```
sns.heatmap(ds.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94e6141358>
```



Data process and Dummy Variables

```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variables into
        binary (0/1) variables. Converts categorical variables into dummy variables. '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            # Example: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

ds1 = preprocess_features(ds1)
```

Unsupervised K-Mean Classification

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=10, random_state=1)
km.fit(ds1)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=10, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=1, tol=0.0001, verbose=0)
```

```
ds1['cluster'] = km.labels_
ds1.sort('cluster').head()
```

```
/home/zahra/miniconda3/lib/python3.6/site-packages/ipykernel/__main__.py:2: FutureWarning: sort(columns=....) is deprecated, use
sort_values(by=.....)
  from ipykernel import kernelapp as app
```

)IT_TRAIL_ID	ALERT_RAISED	LOGFILE_ID	CLUSTER_ID	CLUSTER_TYPE	SECURED_TARGET_CLASS_Database	GRAMMAR_VERSION	cluster
	0	0	0	0	1	0	0
	0	0	0	0	1	0	0
	0	0	0	0	1	0	0
	0	0	0	0	1	0	0
	0	0	0	0	1	0	0

```
km.cluster_centers_
```

Number of Transactions per Cluster

```
ds1.groupby('cluster')['SECURED_TARGET_ID'].count()
```

```
cluster
0    100
1    100
2    100
3      8
4      5
5      2
6      2
7      2
8      1
9      1
Name: SECURED_TARGET_ID, dtype: int64
```

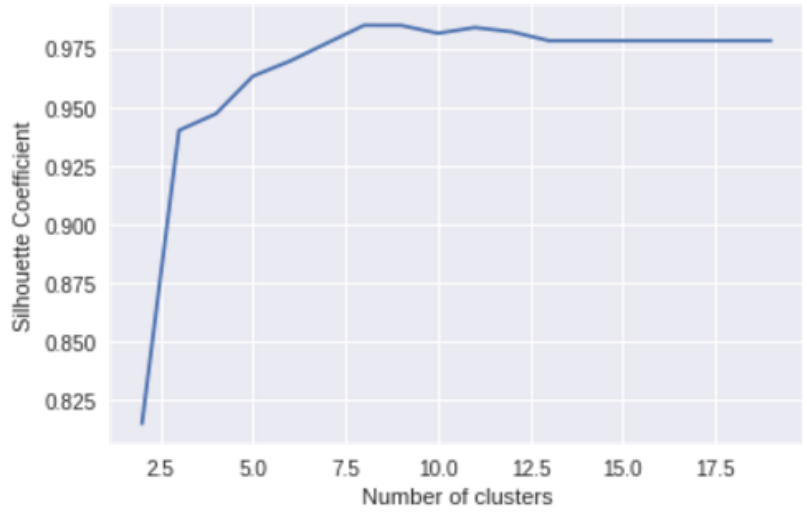
Silhouette Score

```
from sklearn import metrics
metrics.silhouette_score(ds1, km.labels_)
```

```
0.98134725082577667
```

```
k_range = range(2, 20)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(ds1)
    scores.append(metrics.silhouette_score(ds1, km.labels_))
```

```
# plot the results
plt.plot(k_range, scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(True)
```



Add a Target Label for Abnormality

```
def modify(row):
    if row['cluster']>2:
        return 1
    else:
        return 0
```

```
ds1['abnormal'] = ds1.apply(modify,axis=1)
ds1.sort('cluster').head()
ds1[ds1['abnormal']==1].head()
```

```
/home/zahra/miniconda3/lib/python3.6/site-packages/ipykernel/__main__.py:2: FutureWarning: sort(columns=....) is deprecated, use
sort_values(by=.....)
  from ipykernel import kernelapp as app
```

_ID	ALERT_RAISED	LOGFILE_ID	CLUSTER_ID	CLUSTER_TYPE	SECURED_TARGET_CLASS_Database	GRAMMAR_VERSION	cluster	abnormal
	0	0	0	0	1	0	3	1
	0	0	0	0	1	0	3	1
	0	0	0	0	1	0	9	1
	0	0	0	0	1	0	3	1
	0	0	0	0	1	0	3	1

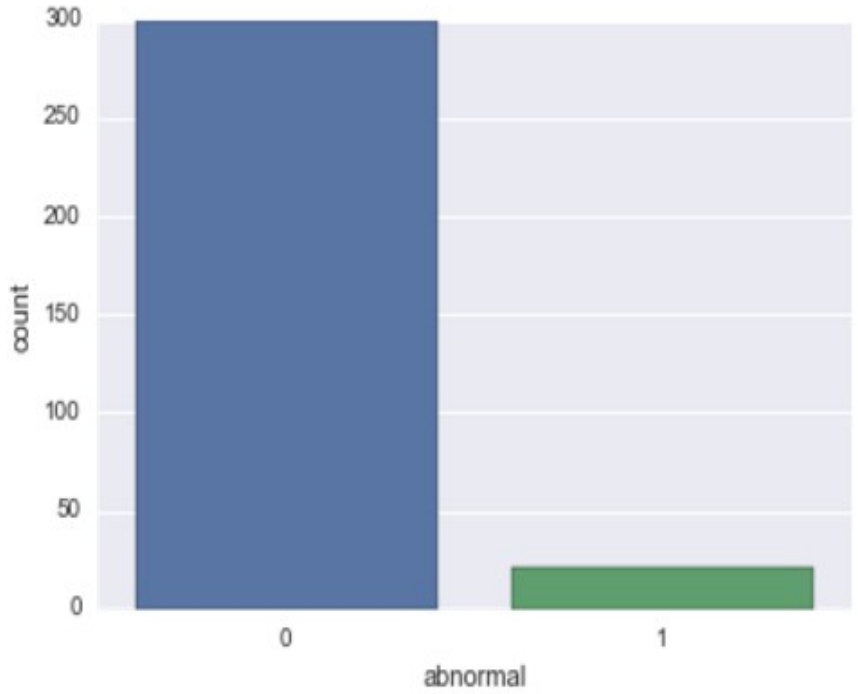
```
X = ds1.drop("abnormal", axis=1)
X = ds1.drop("cluster", axis=1)
y = ds1.abnormal
X.shape
```

```
(321, 30)
```

Visualize Abnormality

```
sns.countplot(x='abnormal',data=ds1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xe7cef28>
```



Grid Search + Gradient Boosting

```
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

```
parameters = [{'max_depth': [2,3,4],'n_estimators':[20,50,100,150]},{'learning_rate': [.50, .1,.15]}]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
model = GradientBoostingClassifier(random_state=42)
#f1_scorer = make_scorer(f1_score)
```

```
grid_obj = GridSearchCV(model, parameters, cv=5, scoring= 'roc_auc')
grid_obj.fit(X_train, y_train)
best_model = grid_obj.best_estimator_
best_model
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features=None, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=20, presort='auto', random_state=42,
              subsample=1.0, verbose=0, warm_start=False)
```

```
y_prob_pred = best_model.predict_proba(X_test)
threshold_adjusted_preds = np.where(y_prob_pred>=0.7, 1, 0)
predictions = threshold_adjusted_preds[:,1]
predictions
```

```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0])
```

Classification Report + Confusion Matrix

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        92
          1       1.00      1.00      1.00         5

avg / total       1.00      1.00      1.00        97
```

```
print(confusion_matrix(y_test,predictions))
```

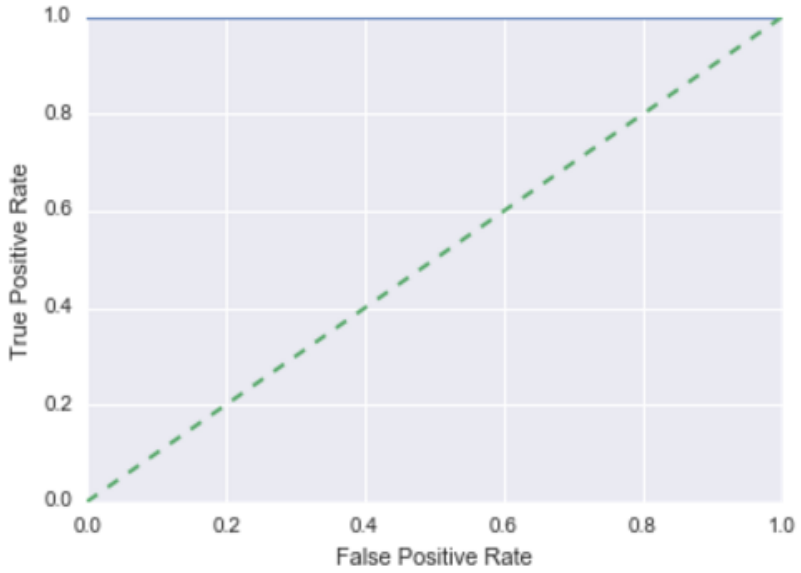
```
[[92  0]
 [ 0  5]]
```

Roc curve

```
from sklearn.metrics import roc_auc_score, roc_curve
y_prob_pred = best_model.predict_proba(X_test)[:,1]
fpr, tpr, thres = roc_curve(y_test, y_prob_pred)
roc_auc_score(y_test, y_prob_pred)
```

```
1.0
```

```
plt.plot(fpr, tpr)
plt.plot([0,1], [0,1], "--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



Tensor Flow Deep Classification

```
from sklearn import datasets, metrics
import tensorflow as tf
```

```
feature_columns = [tf.contrib.layers.real_valued_column("", dimension=4)]

optimizer = tf.train.GradientDescentOptimizer(learning_rate=.1)


# Build 4 layer DNN with 10, 20, 10 units respectively.
classifier = tf.contrib.learn.DNNClassifier(feature_columns=feature_columns,
                                            hidden_units=[10, 20, 20, 10],
                                            optimizer=optimizer,
                                            n_classes=2)
# Fit model.
classifier.fit(x=X_train,
               y=y_train,
               steps=2000)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(x=X_test,
                                     y=y_test)["accuracy"]
print('Accuracy: {0:f}'.format(accuracy_score))

predictions = classifier.predict(X_test)
print('Predictions: {}'.format(predictions))
```

Steps + Loss

```
# Evaluate accuracy.
accuracy_score = classifier.evaluate(x=X_test,
                                     y=y_test)["accuracy"]
print('Accuracy: {0:f}'.format(accuracy_score))

predictions = classifier.predict(X_test)
print('Predictions: {}'.format(predictions))
```

```
INFO:tensorflow:global_step/sec: 169.209
INFO:tensorflow:loss = 0.00622868, step = 1101
INFO:tensorflow:global_step/sec: 167.021
INFO:tensorflow:loss = 0.00504822, step = 1201
INFO:tensorflow:global_step/sec: 168.734
INFO:tensorflow:loss = 0.00415791, step = 1301
INFO:tensorflow:global_step/sec: 167.148
INFO:tensorflow:loss = 0.0034768, step = 1401
INFO:tensorflow:global_step/sec: 162.174
INFO:tensorflow:loss = 0.00294726, step = 1501
INFO:tensorflow:global_step/sec: 128.888
INFO:tensorflow:loss = 0.00252956, step = 1601
INFO:tensorflow:global_step/sec: 162.348
INFO:tensorflow:loss = 0.00219488, step = 1701
INFO:tensorflow:global_step/sec: 140.084
INFO:tensorflow:loss = 0.00192352, step = 1801
INFO:tensorflow:global_step/sec: 147.62
INFO:tensorflow:loss = 0.00170079, step = 1901
INFO:tensorflow:Saving checkpoints for 2000 into /tmp/tmpez0pdd70/model.ckpt.
INFO:tensorflow:Loss for final step: 0.00151734.
```

Accuracy

```
# Fit model.
classifier.fit(x=X_train,
               y=y_train,
               steps=2000)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(x=X_test,
                                     y=y_test)["accuracy"]
print('Accuracy: {0:f}'.format(accuracy_score))

predictions = classifier.predict(X_test)
print('Predictions: {}'.format(predictions))
```

```
Instructions for updating:
Please switch to tf.summary.scalar. Note that tf.summary.scalar uses the node name instead of the tag. This means that T
ensorFlow will automatically de-duplicate summary names based on the scope they are created in. Also, passing a tensor o
r list of tags to a scalar summary op is no longer supported.
INFO:tensorflow:Starting evaluation at 2017-03-25-02:47:32
INFO:tensorflow:Finished evaluation at 2017-03-25-02:47:32
INFO:tensorflow:Saving dict for global step 2000: accuracy = 1.0, accuracy/baseline_label_mean = 0.0515464, accuracy/thr
eshold_0.500000_mean = 1.0, auc = 1.0, global_step = 2000, labels/actual_label_mean = 0.0515464, labels/prediction_mean
= 0.0504955, loss = 0.00110008, precision/positive_threshold_0.500000_mean = 1.0, recall/positive_threshold_0.500000_mea
n = 1.0
WARNING:tensorflow:Skipping summary for global_step, must be a float or np.float32.
Accuracy: 1.000000
WARNING:tensorflow:From /home/zahra/miniconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/estimat
ors/dnn.py:374: calling BaseEstimator.predict (from tensorflow.contrib.learn.python.learn.estimators.estimator) with x i
s deprecated and will be removed after 2016-12-01.
Instructions for updating:
Estimator is decoupled from Scikit Learn interface by moving into
separate class SKCompat. Arguments x, y and batch_size are only
available in the SKCompat class, Estimator will only accept input_fn.
Example conversion:
```

Neural Network with Tensor flow Session

```
# Neural Network with session
X_train.shape
X1 = X_train.as_matrix()
y1 = y_train.as_matrix()
X1.shape

X_train.shape
X2= np.array([X1[1]])
X2.shape
X1[0].shape
```

```
(30L,)
```

```
# Parameters
learning_rate = 0.001
training_epochs = 15
#batch_size = 100
```

```
# Network Parameters
n_hidden_1 = 50 # 1st layer
n_hidden_2 = 50 # 2nd layer
n_input = 30 #  number of features
n_classes = 2 # total classes (0-9 digits)
n_samples = 224 # X_train size
```

```
x_ = tf.placeholder("float", [1,n_input])
y_ = tf.placeholder("float", [1,n_classes])
x_.shape
```

```
TensorShape([Dimension(1), Dimension(30)])
```

Multi-Layer Perceptron

```
def multilayer_perceptron(x, weights, biases):
    '''
    x : Place Holder for Data Input
    weights: Dictionary of weights
    biases: Dicitionary of biases
    '''
    # First Hidden layer with RELU activation
    layer_1 = tf.add(tf.matmul(x_, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)

    # Second Hidden layer with RELU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)

    # Last Output layer with linear activation
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer
```

```
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
}
```

```
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

Model + Cost + Optimizer

```
# Construct model
print(x_)
pred = multilayer_perceptron(x_, weights, biases)
```

```
Tensor("Placeholder_48:0", shape=(1, 30), dtype=float32)
Tensor("Variable_180/read:0", shape=(30, 50), dtype=float32)
Tensor("Relu_53:0", shape=(1, 50), dtype=float32)
Tensor("Variable_181/read:0", shape=(50, 50), dtype=float32)
Tensor("Relu_54:0", shape=(1, 50), dtype=float32)
Tensor("Variable_182/read:0", shape=(50, 2), dtype=float32)
Tensor("add_26:0", shape=(1, 2), dtype=float32)
```

```
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
# Initializing the variables
init = tf.initialize_all_variables()
```

```
WARNING:tensorflow:From <ipython-input-416-c70fa5604581>:2: initialize_all_variables (from tensorflow.python.ops.variables) is d
eprecated and will be removed after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
```

Launch Interactive Session

```
# Launch the session
sess = tf.InteractiveSession()

# Intialize all the variables
sess.run(init)

# Training Epochs
# Essentially the max amount of loops possible before we stop
# May stop earlier if cost/loss limit was set
for epoch in range(training_epochs):

    # Start with cost = 0.0
    avg_cost = 0.0

    # Loop over all samples
    for i in range(n_samples):

        # Feed dictionary for optimization and loss value
        # Returns a tuple, but we only need 'c' the cost
        # So we set an underscore as a "throwaway"
        X2= np.array([X1[i]])
        y2= np.array([y1[i]])
        if(y1[i]==0):
            y2= np.array([[1,0]])
        else:
            y2= np.array([[0,1]])

        _, c = sess.run([optimizer, cost], feed_dict={x_: X2, y_: y2})

        # Compute average loss
        avg_cost += c

    print("Epoch: {} cost={:.4f}".format(epoch+1,avg_cost))

print("Model has completed {} Epochs of Training".format(training_epochs))
```

Average Loss per Training Epochs

```
    # Start with cost = 0.0
    avg_cost = 0.0

    # Loop over all samples
    for i in range(n_samples):

        # Feed dictionary for optimization and loss value
        # Returns a tuple, but we only need 'c' the cost
        # So we set an underscore as a "throwaway"
        X2= np.array([X1[i]])
        y2= np.array([y1[i]])
        if(y1[i]==0):
            y2= np.array([[1,0]])
        else:
            y2= np.array([[0,1]])

        _, c = sess.run([optimizer, cost], feed_dict={x_: X2, y_: y2})

        # Compute average loss
        avg_cost += c

    print("Epoch: {} cost={:.4f}".format(epoch+1,avg_cost))

print("Model has completed {} Epochs of Training".format(training_epochs))
```

```
Epoch: 1 cost=93985.0337
Epoch: 2 cost=2543.2728
Epoch: 3 cost=724.6411
Epoch: 4 cost=281.5975
Epoch: 5 cost=205.3634
Epoch: 6 cost=157.5747
Epoch: 7 cost=171.5258
Epoch: 8 cost=100.8344
Epoch: 9 cost=137.3319
Epoch: 10 cost=63.8347
Epoch: 11 cost=69.8000
Epoch: 12 cost=66.3712
Epoch: 13 cost=87.4280
Epoch: 14 cost=0.0000
Epoch: 15 cost=0.0000
Model has completed 15 Epochs of Training
```

Q & A