Machine Learning Final Project

By: Brett Beaulieu

Classification of Fradulent Charges.

Classification Dataset

Exploration and Preprocessing

In [25]:

```
df = pd.read_csv('creditcard.csv')
df
```

Out[25]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2395
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0788
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7914
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2376
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5929
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9182
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0243
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2968
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6861
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5770

284807 rows × 31 columns

Gather infomation about the variables in the dataset.

In [26]:

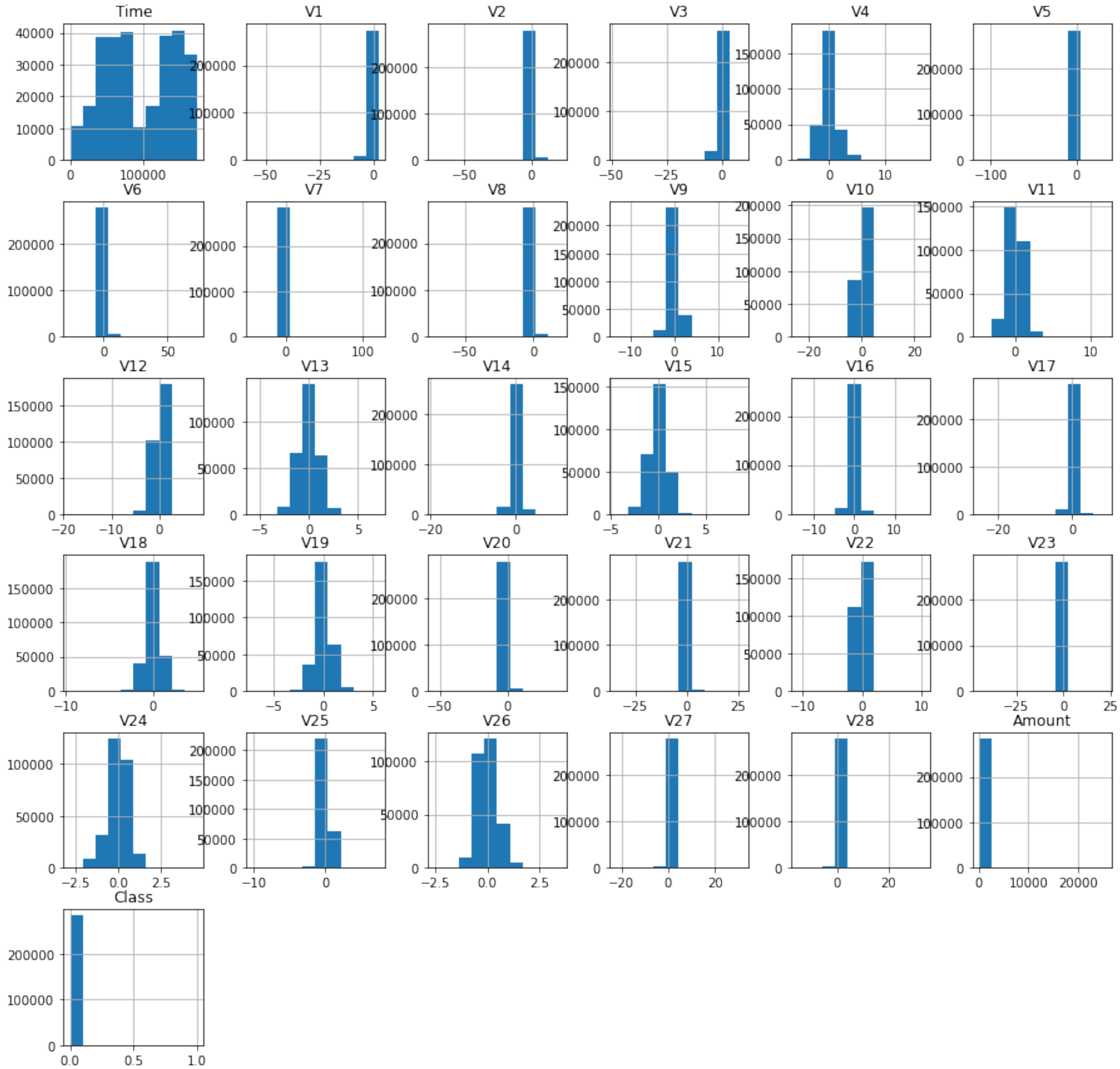
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Visualize the data through histograms looking for common frequencies amongst the data.

In [27]:

```
df.hist(figsize=(15,15))
plt.show()
```



Search for null values in the dataset.

In [28]:
```
df.isnull().sum()
```

Out[28]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

There are no null values requiring data manipulation.

View the dispersion of the binary Class column.

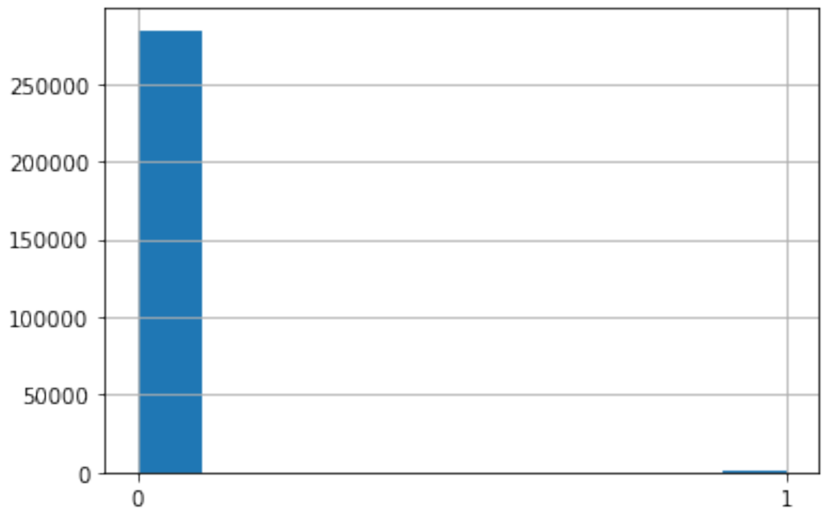
In [29]:
```
count=df.Class.value_counts()
print(count)
```

```
0    284315
1       492
Name: Class, dtype: int64
```

Histogram of the Class column.

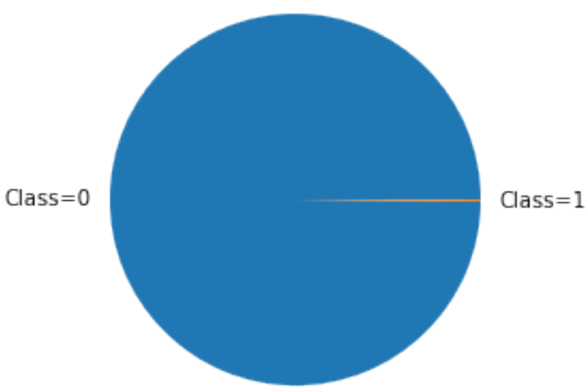
In [30]:
```
df.Class.hist()
plt.xticks([0,1])
```

Out[30]:
```
([<matplotlib.axis.XTick at 0x7f894a7228e0>,
  <matplotlib.axis.XTick at 0x7f894a7228b0>],
 [Text(0, 0, ''), Text(0, 0, '')])
```



View piechart of the data to properly show the significant skew imbalance.

In [31]:
```
l = ['Class=0','Class=1']
plt.pie(count,labels=l)
plt.show()
```



Separate the features and target variables.

In [32]:
```
X = df.drop(['Class'], axis=1)
y = df['Class']
X
```

Out[32]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2395
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0788
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7914
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2376
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5929
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9182
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0243
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2968
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6861
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5770

284807 rows × 30 columns

Split the training and testing data.

In [33]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, ra
```

Alter the imbalanced data with the SMOTE function from imblearn.

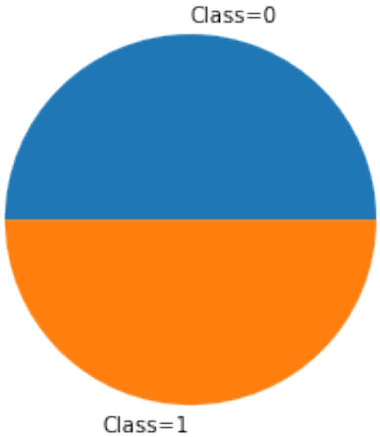
In [34]:

```
smo = SMOTE(random_state=42)
X_smo, y_smo = smo.fit_resample(X_train,y_train)
count[0]=len(X_smo)
count[1]=len(y_smo)
```

Show the updated pie chart after imbalance of the dataset was altered by the SMOTE function.

In [35]:

```
plt.pie(count,labels=l)
plt.show()
```



Split our balanced data in a training and testing set.

In [36]:
```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_smo)
X_test_scale = scaler.transform(X_test)
```

Logistic Regression

Use GridsearchCV to determine the best parameters for the Logistic Regression model.

In [37]:
```
parameters = {
    'C': [0.01, 0.1, 1, 10, 10],
    'solver' : ["lbfgs","liblinear"]
}
lr = LogisticRegression(max_iter=1000,random_state=42)
glr = GridSearchCV(lr, parameters, cv=3, verbose=5, n_jobs=3)
gd=glr.fit(X_train_scale, y_smo)
gd
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[37]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=1000, random_state=42
),
             n_jobs=3,
             param_grid={'C': [0.01, 0.1, 1, 10, 10],
                         'solver': ['lbfgs', 'liblinear']},
             verbose=5)
```

Print the best parameters for the GridSearchCV.

In [38]:
```
print("Best parameters : %s" % gd.best_params_)
```

```
Best parameters : {'C': 10, 'solver': 'lbfgs'}
```

Train the GridSearch best parameters on the Logistic Regression model.

In [39]:
```
log_reg = LogisticRegression(solver='lbfgs', random_state = 42, max_iter = 10
log_reg.fit(X_train_scale, y_smo)
```

Out[39]:
```
LogisticRegression(C=10, max_iter=1000, random_state=42)
```

Make predictions for the Logistic Regression model.

In [40]:
```
y_pred = log_reg.predict(X_test_scale)
y_pred
```

Out[40]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

Show the confusion matrix.

In [41]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[112744    988]
 [    23    168]]
```

Print the accuracy score.

In [42]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.9911255848248378
```

Print the classification report for the Logisitic Regression model.

In [43]:
```
target_names = ['class=0', 'class=1']
print(classification_report(y_test,y_pred,target_names=target_names))
```

```
              precision    recall  f1-score   support

     class=0       1.00      0.99      1.00    113732
     class=1       0.15      0.88      0.25       191

    accuracy                           0.99    113923
   macro avg       0.57      0.94      0.62    113923
weighted avg       1.00      0.99      0.99    113923
```

Decision Tree Classifier

Use GridsearchCV to determine the best parameters for the Decision Tree Classifer model.

In [44]:
```
parameters = {
    'criterion' : ["gini", "entropy"],
    'max_depth' : [10,12,15,20]
}
dtc = DecisionTreeClassifier(random_state=42)
glr = GridSearchCV(dtc, parameters, cv=3, verbose=5, n_jobs=3)
gd=glr.fit(X_train_scale, y_smo)
gd
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

Out[44]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=42), n_jobs=3
,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [10, 12, 15, 20]},
             verbose=5)
```

Print the best parameters for the GridSearchCV of the Decision Tree Classifier.

In [45]:
```
print("Best parameters : %s" % gd.best_params_)
```

```
Best parameters : {'criterion': 'entropy', 'max_depth': 20}
```

Train the GridsearchCV parameters on the Decision Tree Clasifier model.

In [46]:
```
dtc=DecisionTreeClassifier(criterion='entropy',max_depth=20, random_state=42)
dtc.fit(X_train_scale, y_smo)
```

Out[46]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=20, random_state=42)
```

Make predicitons with the Decision Tree Classifier model.

In [47]:
```
y_pred2 = dtc.predict(X_test_scale)
y_pred2
```

Out[47]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

Print out the Decision Tree Classifier confusion matrix.

In [48]:
```
cm = confusion_matrix(y_test, y_pred2)
cm
```

Out[48]:
```
array([[113521,    211],
       [    35,    156]])
```

Print out the Decision Tree Classifier accuracy score.

In [49]:
```
print(accuracy_score(y_test,y_pred2))
```

```
0.9978406467526312
```

Print out the Decision Tree Classifier classification report.

In [50]:
```
target_names = ['class=0', 'class=1']
print(classification_report(y_test,y_pred2,target_names=target_names))
```

```
              precision    recall  f1-score   support

     class=0       1.00      1.00      1.00    113732
     class=1       0.43      0.82      0.56       191

    accuracy                           1.00    113923
   macro avg       0.71      0.91      0.78    113923
weighted avg       1.00      1.00      1.00    113923
```

Random Forest Classifier

Use GridsearchCV to determine the best parameters for the Ranfom Forest Classifer model.

In [51]:

```
parameters = {
    'max_depth' : [9,10,11],
    'max_features': list(range(1,4))
}
rfc = RandomForestClassifier(random_state=42)
glr = GridSearchCV(rfc, parameters, cv=3, verbose=5, n_jobs=3)
gd=glr.fit(X_train, y_train)
gd
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

Out[51]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42), n_jobs=3
,
             param_grid={'max_depth': [9, 10, 11], 'max_features': [1, 2, 3]},
             verbose=5)
```

Find the best parameters from the GridSearchCV function

In [52]:

```
print("Best parameters : %s" % gd.best_params_)
```

```
Best parameters : {'max_depth': 11, 'max_features': 3}
```

Apply the best parameters to the Random Forest Classifer Model.

In [53]:

```
rfc=RandomForestClassifier(random_state=42,max_depth=11, max_features= 3)
rfc.fit(X_train, y_train)
```

Out[53]:

```
RandomForestClassifier(max_depth=11, max_features=3, random_state=42)
```

Predict data using the Random Forest Classifier Model.

In [54]:

```
y_pred3 = rfc.predict(X_test)
y_pred3
```

Out[54]:

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

Show the confusion matrix for the model.

In [55]:
```
cm = confusion_matrix(y_test, y_pred3)
cm
```

Out[55]:
```
array([[113724,      8],
       [    50,    141]])
```

Show the accuracy score for the model.

In [56]:
```
print(accuracy_score(y_test,y_pred3))
```

```
0.9994908841937098
```

Show the classification report for the Random Forest Classifier.

In [57]:
```
target_names = ['class=0', 'class=1']
print(classification_report(y_test,y_pred3,target_names=target_names))
```

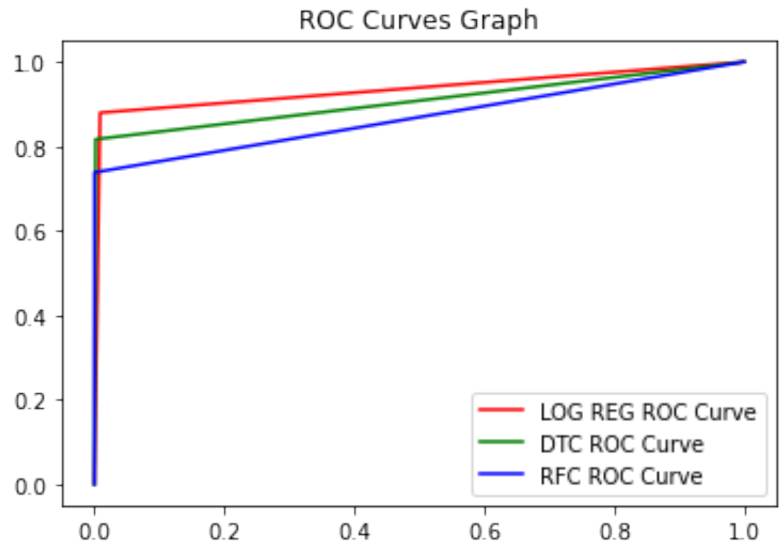
```
              precision    recall  f1-score   support

     class=0       1.00      1.00      1.00    113732
     class=1       0.95      0.74      0.83       191

    accuracy                           1.00    113923
   macro avg       0.97      0.87      0.91    113923
weighted avg       1.00      1.00      1.00    113923
```

Show the ROC cruve graph for all three classification models.

In [58]:
```
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_test, y_pred)
plt.plot(fpr_forest,tpr_forest, 'r-', label="LOG REG ROC Curve")
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_test, y_pred2)
plt.plot(fpr_forest,tpr_forest, 'g-', label="DTC ROC Curve")
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_test, y_pred3)
plt.plot(fpr_forest,tpr_forest, 'b-', label="RFC ROC Curve")
plt.title('ROC Curves Graph')
plt.legend()
plt.show()
```



ROC Curves Graph
LOG REG ROC Curve
DTC ROC Curve
RFC ROC Curve
0.0
0.2
0.4
0.6
0.8
1.0