---

*You are currently looking at **version 0.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

In [47]:
```
import numpy as np
import pandas as pd
```

Question 1

Import the data from `assets/fraud_data.csv`. What percentage of the observations in the dataset are instances of fraud?

This function should return a float between 0 and 1.

In [48]:
```
def answer_one():
    import pandas as pd
    df = pd.read_csv('assets/fraud_data.csv')
    return len(df[df['Class'] == 1]) / len(df)
answer_one()
```

Out[48]:
```
0.016410823768035772
```

In [ ]:
```

```

In [49]:
```
# Use X_train, X_test, y_train, y_test for all of the following questions
from sklearn.model_selection import train_test_split

df = pd.read_csv('assets/fraud_data.csv')

X = df.iloc[:,:-1]
y = df.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Question 2

Using `X_train`, `X_test`, `y_train`, and `y_test` (as defined above), train a dummy classifier that classifies everything as the majority class of the training data. What is the accuracy of this classifier? What is the recall?

This function should a return a tuple with two floats, i.e. `(accuracy score, recall score)`.

In [50]:
```
def answer_two():
    from sklearn.dummy import DummyClassifier
    from sklearn.metrics import recall_score, accuracy_score

    # Your code here

    dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train, y_train)
    # Therefore the dummy 'most_frequent' classifier always predicts class 0
    y_dummy_predictions = dummy_majority.predict(X_test)

    ans = (accuracy_score(y_test, y_dummy_predictions), recall_score(y_test, y_dummy_predictions))

    return ans
answer_two()
```

Out[50]:
```
(0.9852507374631269, 0.0)
```

In [ ]:
```

```

Question 3

Using X_train, X_test, y_train, y_test (as defined above), train a SVC classifer using the default parameters. What is the accuracy, recall, and precision of this classifier?

This function should a return a tuple with three floats, i.e. `(accuracy score, recall score, precision score)`.

In [51]:
```
def answer_three():
    from sklearn.metrics import recall_score, precision_score, accuracy_score
    from sklearn.svm import SVC

    # Your code here
    svm = SVC().fit(X_train, y_train)
    y_predictions = svm.predict(X_test)

    ans = (accuracy_score(y_test, y_predictions), recall_score(y_test, y_predictions), precision_score(y_test, y_predictions))

    return ans
answer_three()
```

Out[51]:
```
(0.9900442477876106, 0.35, 0.9333333333333333)
```

In [ ]:
```

```

Question 4

Using the SVC classifier with parameters `{'C': 1e9, 'gamma': 1e-07}`, what is the confusion matrix when using a threshold of -220 on the decision function. Use X_test and y_test.

This function should return a confusion matrix, a 2x2 numpy array with 4 integers.

In [52]:
```
def answer_four():
    from sklearn.metrics import confusion_matrix
    from sklearn.svm import SVC

    # Your code here
    svm = SVC(C = 1e9, gamma = 1e-07).fit(X_train, y_train)
    svm_predicted = svm.decision_function(X_test) > -220

#     print(svm_predicted)

    confusion = confusion_matrix(y_test, svm_predicted)

    ans = confusion

    return ans

answer_four()
```

Out[52]:
```
array([[5320,   24],
       [  14,   66]])
```

In [ ]:

Question 5

Train a logisitic regression classifier with default parameters using X_train and y_train. This classifier should use the parameter solver='liblinear'.

For the logisitic regression classifier, compute the scores using decision_function() or with predict_proba(), then create a precision recall curve and a roc curve using y_test and the probability estimates for X_test (probability it is fraud).

Looking at the precision recall curve, what is the recall when the precision is `0.75`?

Looking at the roc curve, what is the true positive rate when the false positive rate is `0.16`?

Note: When getting the ROC curve and finding the records where the FPR entry is closest to 0.16, take the corresponding TPRs. As there are two such records where the FPR is close to 0.16, take the higher TPR of these two records.

This function should return a tuple with two floats, i.e. `(recall, true positive rate)`.

In [53]:
```
def answer_five():
    import numpy as np
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import precision_recall_curve, roc_curve

    lr = LogisticRegression()
    lr.fit(X_train, y_train)

    y_scores_lr = lr.predict_proba(X_test)[:, 1]

    # Precision-Recall curve
    precision, recall, _ = precision_recall_curve(y_test, y_scores_lr)

    # Find recall where precision is closest to 0.75 from above
    # Method 1: Find where precision >= 0.75, then take the maximum recall
    recall_at_075 = recall[np.where(precision >= 0.75)[0]].max()

    # ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_scores_lr)

    # Find TPR where FPR is closest to 0.16 from below
    tpr_at_016 = tpr[np.where(fpr <= 0.16)[0]].max()

    return (recall_at_075, tpr_at_016)

answer_five()
```

Out[53]:
```
(0.825, 0.925)
```

In [ ]:

Question 6

Perform a grid search over the parameters listed below for a Logisitic Regression classifier, using recall for scoring and the default 3-fold cross validation. (Suggest to use `solver='liblinear'`, more explanation here)

`'penalty': ['l1', 'l2']`

`'C':[0.01, 0.1, 1, 10]`

From `.cv_results_`, create an array of the mean test scores of each parameter combination. i.e.

	`l1`	`l2`
`0.01`	?	?
`0.1`	?	?
`1`	?	?
`10`	?	?

This function should return a 4 by 2 numpy array with 8 floats.

Note: do not return a DataFrame, just the values denoted by `?` in a numpy array.

In [54]:
```
def answer_six():
    import numpy as np
    from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LogisticRegression

    lr = LogisticRegression(solver='liblinear')

    grid_values = {
        'penalty': ['l1', 'l2'],
        'C': [0.01, 0.1, 1, 10]
    }

    grid_lr = GridSearchCV(
        lr,
        param_grid=grid_values,
        scoring='recall'
    )

    grid_lr.fit(X_train, y_train)

    # Reshape the mean_test_score directly
    # GridSearchCV iterates in order: first all penalties for C=0.01, then C=0.1, etc.
    scores = grid_lr.cv_results_['mean_test_score'].reshape(4, 2)
```

```
    return scores

answer_six()
```

Out[54]:
```
array([[0.69558442, 0.77168831],
       [0.80792208, 0.81155844],
       [0.80428571, 0.81149351],
       [0.80064935, 0.80064935]])
```

In [ ]:
```
```

In [36]:
```
def GridSearch_Heatmap(scores):
    import seaborn as sns
    import matplotlib.pyplot as plt

    plt.figure()
    sns.heatmap(
        scores.reshape(4, 2),
        xticklabels=['l1', 'l2'],
        yticklabels=[0.01, 0.1, 1, 10]
    )
    plt.yticks(rotation=0)
#GridSearch_Heatmap(scores)
```

In [ ]:
```
```