In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statistics
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv('Boston.csv')
df_copy = df.copy()
df_copy = df_copy.drop('Unnamed: 0', 1)
# crim = per capita crime rate by town
# zn = proportion of residential land zoned for lots over 25,000 sq.ft.
# INDUS - proportion of non-retail business acres per town.
# CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
# NOX - nitric oxides concentration (parts per 10 million)
# RM - average number of rooms per dwelling
# AGE - proportion of owner-occupied units built prior to 1940
# DIS - weighted distances to five Boston employment centres
# RAD - index of accessibility to radial highways
# TAX - full-value property-tax rate per $10,000
# PTRATIO - pupil-teacher ratio by town
# B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
# LSTAT - % lower status of the population
# MEDV - Median value of owner-occupied homes in $1000's



# Create 'crim01' column, 1 => crim > crim_median, 0 => crim < crim_median
crim_median = df_copy['crim'].median()
df_copy.loc[df_copy['crim'] >= crim_median, 'crim01'] = 1
df_copy.loc[df_copy['crim'] < crim_median, 'crim01'] = 0
df_copy.head()
```

Out[2]:

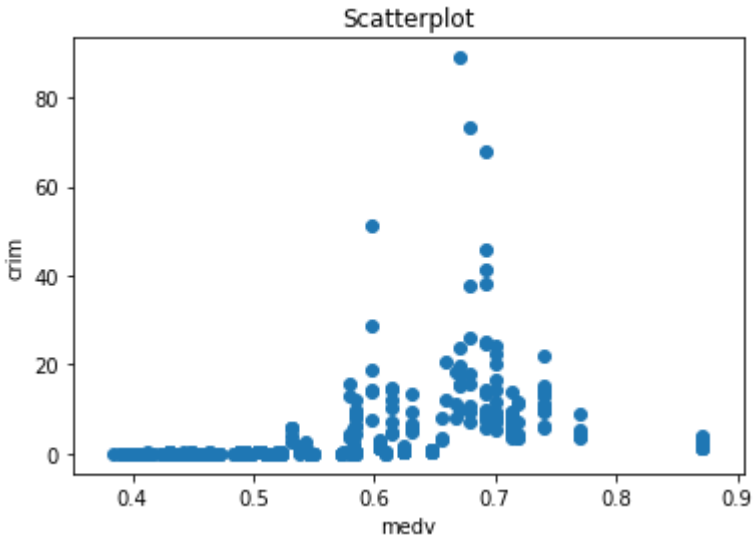
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [3]:
```
# pd.plotting.scatter_matrix(df_copy, figsize=(12,12)); # Semicolon used to su
ppress array output!
```

In [4]:
```
plt.scatter(df_copy['nox'], df_copy['crim']) # Slightly exponential relationsh
ip

plt.title("Scatterplot")
plt.xlabel("medv")
plt.ylabel("crim")
```

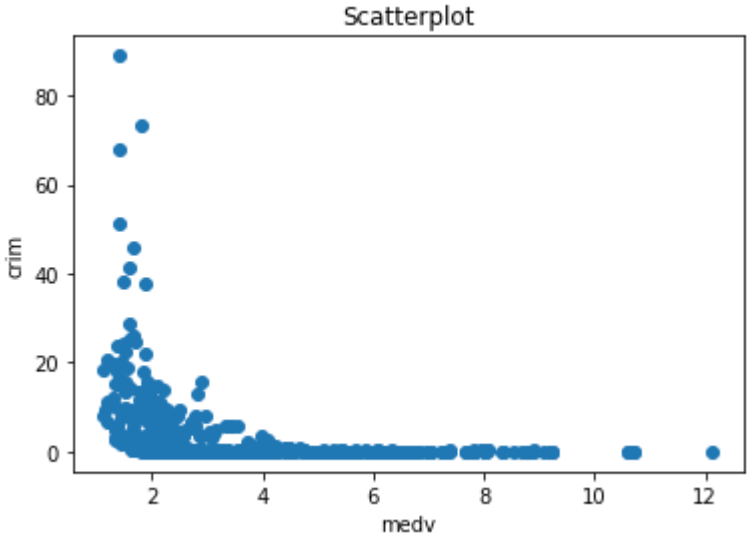
Out[4]: Text(0,0.5,'crim')



In [5]:
```
plt.scatter(df_copy['dis'], df_copy['crim']) # Slight inverse relationship

plt.title("Scatterplot")
plt.xlabel("medv")
plt.ylabel("crim")
```

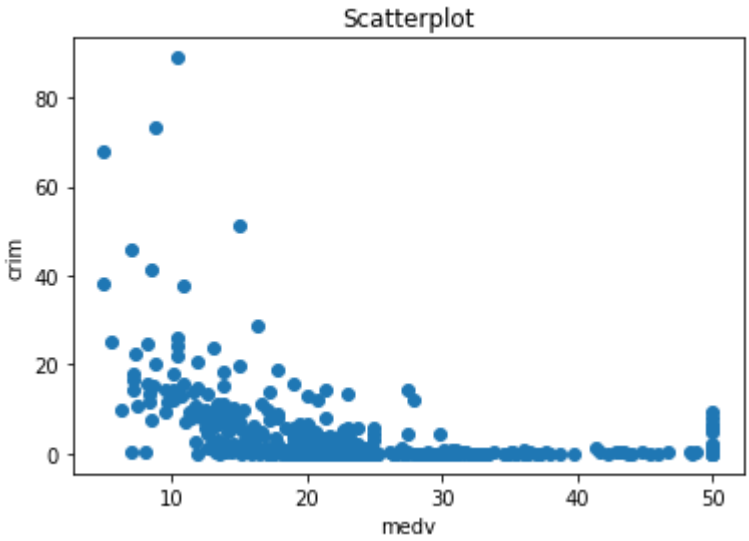
Out[5]: Text(0,0.5,'crim')



In [6]:
```
plt.scatter(df_copy['medv'], df_copy['crim']) # slight exponential decay/inver
se

plt.title("Scatterplot")
plt.xlabel("medv")
plt.ylabel("crim")
```

Out[6]: Text(0,0.5,'crim')



In [7]:
```
# plt.scatter(df_copy['zn'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['indus'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['chas'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['rm'], df_copy['crim']) # Random scatter on left, decrea
se to zero on right
# plt.scatter(df_copy['age'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['rad'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['tax'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['ptratio'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['black'], df_copy['crim']) # No visible correlation
# plt.scatter(df_copy['lstat'], df_copy['crim']) # No visible correlation
```

KNN Model

In [8]:
```
y_true = df_copy['crim01']
columns = ['nox', 'rm', 'dis', 'medv']
X_var = df_copy[columns]

# Make variable transformations
data = {
        'exp-nox': np.exp(df_copy['nox']),
#         'rm': df_copy['rm'],
        'inverse-dis': np.reciprocal(df_copy['dis']),
        'exp_decay-medv': np.exp(-1*df_copy['medv']),
       }
transformed_data = pd.DataFrame(data)




numExperiments = 20
accuracies_list = []

for i in range(numExperiments):
    X_train, X_test, y_train, y_test = train_test_split(transformed_data, y_tr
ue, test_size=0.20)

    # ---------------------- FITTING THE MODEL ------------------------------
---
    # Create the KNN Classifier Model and fit it
    KNN_classifier = KNeighborsClassifier(n_neighbors=9)
    KNN_classifier.fit(X_train, y_train)


    # ---------------------- MAKING PREDICTIONS ----------------------------
----
    y_pred_test = KNN_classifier.predict(X_test)
    num_TN, num_FP, num_FN, num_TP = confusion_matrix(y_test, y_pred_test).rav
el()
#     print("Number of True Positives: " + repr(num_TP))
#     print("Number of False Negatives: " + repr(num_FN))
#     print("Number of False Positives: " + repr(num_FP))
#     print("Number of True Negatives: " + repr(num_TN))

    currentAccuracy = (num_TP + num_TN)/(num_TP + num_TN + num_FP + num_FN)
    accuracies_list.append(currentAccuracy)

median_acc = statistics.median(accuracies_list)
print(median_acc)
```

```
0.8823529411764706
```

Logistic Regression Model

In [9]:
```
y_true = df_copy['crim01']


# Make variable transformations
data = {
        'exp-nox': np.exp(df_copy['nox']),
#         'rm': df_copy['rm'],
        'inverse-dis': np.reciprocal(df_copy['dis']),
        'exp_decay-medv': np.exp(-1*df_copy['medv']),
       }
transformed_data = pd.DataFrame(data)

numExperiments = 20
accuracies_list = []




for i in range(numExperiments):
    X_train, X_test, y_train, y_test = train_test_split(transformed_data, y_tr
ue, test_size=0.20)

    # ---------------------- FITTING THE MODEL ----------------------------
---
    # Create the Logistic Regression Model and fit it
    Logit_classifier = LogisticRegression()
    Logit_classifier.fit(X_train, y_train)


    # ---------------------- MAKING PREDICTIONS ---------------------------
----
    y_pred_test = Logit_classifier.predict(X_test)
    num_TN, num_FP, num_FN, num_TP = confusion_matrix(y_test, y_pred_test).rav
el()
#     print("Number of True Positives: " + repr(num_TP))
#     print("Number of False Negatives: " + repr(num_FN))
#     print("Number of False Positives: " + repr(num_FP))
#     print("Number of True Negatives: " + repr(num_TN))
    currentAccuracy = (num_TP + num_TN)/(num_TP + num_TN + num_FP + num_FN)
#     print(currentAccuracy)
    accuracies_list.append(currentAccuracy)

median_acc = statistics.median(accuracies_list)
print(median_acc)
```

```
0.7892156862745098
```

LDA Model

In [10]:
```
y_true = df_copy['crim01']


# Make variable transformations
data = {
        'exp-nox': np.exp(df_copy['nox']),
#         'rm': df_copy['rm'],
        'inverse-dis': np.reciprocal(df_copy['dis']),
        'exp_decay-medv': np.exp(-1*df_copy['medv']),
       }
transformed_data = pd.DataFrame(data)

numExperiments = 20
accuracies_list = []




for i in range(numExperiments):
    X_train, X_test, y_train, y_test = train_test_split(transformed_data, y_tr
ue, test_size=0.20)

    # ---------------------- FITTING THE MODEL ----------------------------
---
    # Create the Linear Discriminant Classifier Model and fit it
    LDA_classifier = LinearDiscriminantAnalysis()
    LDA_classifier.fit(X_train, y_train)


    # ---------------------- MAKING PREDICTIONS ---------------------------
----
    y_pred_test = LDA_classifier.predict(X_test)
    num_TN, num_FP, num_FN, num_TP = confusion_matrix(y_test, y_pred_test).rav
el()
#     print("Number of True Positives: " + repr(num_TP))
#     print("Number of False Negatives: " + repr(num_FN))
#     print("Number of False Positives: " + repr(num_FP))
#     print("Number of True Negatives: " + repr(num_TN))
    currentAccuracy = (num_TP + num_TN)/(num_TP + num_TN + num_FP + num_FN)
    accuracies_list.append(currentAccuracy)

median_acc = statistics.median(accuracies_list)
print(median_acc)
```

```
0.8186274509803921
```

In [ ]:
```

```