In [1]:

```
import pandas as pd
import numpy as np
import datetime
```

In [2]:

```
chunksize = 1000000
count = 0

for chunk in pd.read_csv("flights.csv", chunksize=chunksize):
    jfk = chunk.loc[chunk['ORIGIN_AIRPORT'] == "JFK"]

    jfk.to_csv("data/test" + str(count) +".csv", index = False)
    count += 1
    # late = jfk.loc[jfk['DEPARTURE_DELAY'] > 15]
    # print(len(jfk))
```

```
/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshel
l.py:3071: DtypeWarning: Columns (7,8) have mixed types.Specify dtype o
ption on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

In [2]:

```
weather = pd.read_csv("jfk_weather_cleaned.csv")
filtered2015 = weather[weather['DATE'].apply(lambda x: x.startswith('201
5'))]
nyd = weather[weather['DATE'].apply(lambda x: x.startswith('2016-01-01 0
0:00:00'))]
filtered2015 = filtered2015.append(nyd)
# filtered2015.to_csv("2015weather.csv")
```

In [3]:

```
filtered2015
```

Out[3]:

	DATE	HOURLYVISIBILITY	HOURLYDRYBULBTEMPF	HOURLYWETBULBTEMPF	HOURLY
43823	2015-01-01 00:00:00	10.0	30.0	24.0	
43824	2015-01-01 01:00:00	10.0	29.0	24.0	
43825	2015-01-01 02:00:00	10.0	29.0	24.0	
43826	2015-01-01 03:00:00	10.0	29.0	24.0	
43827	2015-01-01 04:00:00	10.0	29.0	23.0	
...	...	...	...	...	
52579	2015-12-31 20:00:00	10.0	47.0	40.0	
52580	2015-12-31 21:00:00	10.0	46.0	39.0	
52581	2015-12-31 22:00:00	10.0	45.0	38.0	
52582	2015-12-31 23:00:00	10.0	44.0	37.0	
52583	2016-01-01 00:00:00	10.0	44.0	38.0	

8761 rows × 16 columns

In [4]:

```
filtered2015.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8761 entries, 43823 to 52583
Data columns (total 16 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   DATE                          8761 non-null   object
 1   HOURLYVISIBILITY              8761 non-null   float64
 2   HOURLYDRYBULBTEMPF            8761 non-null   float64
 3   HOURLYWETBULBTEMPF            8761 non-null   float64
 4   HOURLYDewPointTempF           8761 non-null   float64
 5   HOURLYRelativeHumidity        8761 non-null   float64
 6   HOURLYWindSpeed               8761 non-null   float64
 7   HOURLYStationPressure         8761 non-null   float64
 8   HOURLYSeaLevelPressure        8761 non-null   float64
 9   HOURLYPrecip                  8761 non-null   float64
 10  HOURLYAltimeterSetting        8761 non-null   float64
 11  HOURLYWindDirectionSin        8761 non-null   float64
 12  HOURLYWindDirectionCos        8761 non-null   float64
 13  HOURLYPressureTendencyIncr    8761 non-null   int64
 14  HOURLYPressureTendencyDecr    8761 non-null   int64
 15  HOURLYPressureTendencyCons    8761 non-null   int64
dtypes: float64(12), int64(3), object(1)
memory usage: 1.1+ MB
```

In [5]:

```
test0 = pd.read_csv("data/test0.csv")
test0
```

Out[5]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
0	2015	1	1	4	B6	2023	N324JB	
1	2015	1	1	4	AA	2299	N3LLAA	
2	2015	1	1	4	B6	939	N794JB	
3	2015	1	1	4	B6	353	N570JB	
4	2015	1	1	4	B6	583	N531JB	
...	...	...	...	...	...	...	...	
18007	2015	3	7	6	B6	615	N942JB	
18008	2015	3	7	6	B6	208	N198JB	
18009	2015	3	7	6	B6	1634	N328JB	
18010	2015	3	7	6	DL	2474	N698DL	
18011	2015	3	7	6	DL	2600	N686DA	

18012 rows × 31 columns

In [6]:

```
test0.loc[test0['DEPARTURE_DELAY'] >= 15, 'DELAYED'] = 1
test0.loc[test0['DEPARTURE_DELAY'] < 15, 'DELAYED'] = 0
```

In [7]:

```
test0['CANCELLED'].unique()
test0 = test0.loc[test0['CANCELLED'] == 0].copy()
test0['CANCELLED'].unique()
test0 = test0.drop(['CANCELLED', 'CANCELLATION_REASON'], axis = 1)
```

In [8]:

```
test0 = test0.drop(['DEPARTURE_DELAY', 'ARRIVAL_DELAY','AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY'], axis = 1).copy()
```

In [9]:

```
test0 = test0.drop(["ORIGIN_AIRPORT"], axis = 1).copy()
```

In [10]:

```
print(test0.isnull().sum())
test0 = test0.dropna().copy()
print(test0.isnull().sum())
```

```
YEAR                    0
MONTH                   0
DAY                     0
DAY_OF_WEEK             0
AIRLINE                 0
FLIGHT_NUMBER           0
TAIL_NUMBER             0
DESTINATION_AIRPORT     0
SCHEDULED_DEPARTURE     0
DEPARTURE_TIME          0
TAXI_OUT                0
WHEELS_OFF              0
SCHEDULED_TIME          0
ELAPSED_TIME           59
AIR_TIME               59
DISTANCE                0
WHEELS_ON               1
TAXI_IN                 1
SCHEDULED_ARRIVAL       0
ARRIVAL_TIME            1
DIVERTED                0
DELAYED                 0
dtype: int64
YEAR                   0
MONTH                  0
DAY                    0
DAY_OF_WEEK            0
AIRLINE                0
FLIGHT_NUMBER          0
TAIL_NUMBER            0
DESTINATION_AIRPORT    0
SCHEDULED_DEPARTURE    0
DEPARTURE_TIME         0
TAXI_OUT               0
WHEELS_OFF             0
SCHEDULED_TIME         0
ELAPSED_TIME           0
AIR_TIME               0
DISTANCE               0
WHEELS_ON              0
TAXI_IN                0
SCHEDULED_ARRIVAL      0
ARRIVAL_TIME           0
DIVERTED               0
DELAYED                0
dtype: int64
```

In [11]:

```
test0["SCHEDULED_DEPARTURE_FORMAT"] = pd.to_datetime(test0["SCHEDULED_DEPARTURE"], format='%H%M')
test0["SCHEDULED_DEPARTURE_FORMAT"] = test0["SCHEDULED_DEPARTURE_FORMAT"].dt.time
```

In [12]:
```
test0
```

Out[12]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	DESTIN
0	2015	1	1	4	B6	2023	N324JB	
1	2015	1	1	4	AA	2299	N3LLAA	
2	2015	1	1	4	B6	939	N794JB	
3	2015	1	1	4	B6	353	N570JB	
4	2015	1	1	4	B6	583	N531JB	
...	...	...	...	...	...	...	...	
18007	2015	3	7	6	B6	615	N942JB	
18008	2015	3	7	6	B6	208	N198JB	
18009	2015	3	7	6	B6	1634	N328JB	
18010	2015	3	7	6	DL	2474	N698DL	
18011	2015	3	7	6	DL	2600	N686DA	

16780 rows × 23 columns

In [13]:

```
def createJoin(row):
    year = row["YEAR"]
    month = row["MONTH"]
    day = row["DAY"]
    hour = (row["SCHEDULED_DEPARTURE_FORMAT"].hour)
    minute = (row["SCHEDULED_DEPARTURE_FORMAT"].minute)
    month30 = [4, 6, 9 , 11]

    # for december 31
    if (month == 12 and day == 31 and hour == 23 and minute > 30):
        return datetime.datetime(year+1, 1, 1, 0, 0)

    # for febuary 28th
    elif (day == 28 and month == 2 and hour == 23 and minute > 30):
        return datetime.datetime(year, month+1, 1, 0,0)

    # for 30th day of 30day months
    elif (day == 30 and (month in month30) and hour == 23 and minute > 3
0 ):
        return datetime.datetime(year, month+1,1,0,0)

    # for every other end of month
    elif (day == 31 and hour == 23 and minute> 30):
        return datetime.datetime(year, month+1,1,0,0)

    # for every end of day at 11.30
    elif (hour == 23 and minute > 30):
        return datetime.datetime(year, month,day+1,0,0)

    # for every hour
    elif (minute > 30):
        return datetime.datetime(year, month,day,hour+1,0)

    elif (minute <= 30):
        return datetime.datetime(year, month,day,hour,0)
```

In [14]:

```
new_column = test0.apply(lambda row: createJoin(row), axis=1)
test0.insert(0, "DATETIME", new_column)
test0 = test0.drop(["SCHEDULED_DEPARTURE_FORMAT"], axis = 1).copy()
```

In [15]:
```
test0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16780 entries, 0 to 18011
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype         
---  ------               --------------  -----         
 0   DATETIME             16780 non-null  datetime64[ns]
 1   YEAR                 16780 non-null  int64         
 2   MONTH                16780 non-null  int64         
 3   DAY                  16780 non-null  int64         
 4   DAY_OF_WEEK          16780 non-null  int64         
 5   AIRLINE              16780 non-null  object        
 6   FLIGHT_NUMBER        16780 non-null  int64         
 7   TAIL_NUMBER          16780 non-null  object        
 8   DESTINATION_AIRPORT  16780 non-null  object        
 9   SCHEDULED_DEPARTURE  16780 non-null  int64         
 10  DEPARTURE_TIME       16780 non-null  float64       
 11  TAXI_OUT             16780 non-null  float64       
 12  WHEELS_OFF           16780 non-null  float64       
 13  SCHEDULED_TIME       16780 non-null  float64       
 14  ELAPSED_TIME         16780 non-null  float64       
 15  AIR_TIME             16780 non-null  float64       
 16  DISTANCE             16780 non-null  int64         
 17  WHEELS_ON            16780 non-null  float64       
 18  TAXI_IN              16780 non-null  float64       
 19  SCHEDULED_ARRIVAL    16780 non-null  int64         
 20  ARRIVAL_TIME         16780 non-null  float64       
 21  DIVERTED             16780 non-null  int64         
 22  DELAYED              16780 non-null  float64       
dtypes: datetime64[ns](1), float64(10), int64(9), object(3)
memory usage: 3.1+ MB
```

In [16]:
```
data_with_weather = test0.set_index('DATETIME').join(filtered2015.set_index('DATE')).copy()
```

In [17]:
```
data_with_weather.reset_index(inplace = True)
```

In [18]:
```
data_with_weather.rename({'index': 'DATETIME'}, axis=1, inplace=True)
```

In [19]:

```
data_with_weather.nunique()
data_with_weather = data_with_weather.drop(['YEAR', 'DIVERTED'], axis = 1)
data_with_weather.nunique()
```

Out[19]:

```
DATETIME                       1218
MONTH                             3
DAY                              31
DAY_OF_WEEK                       7
AIRLINE                           8
FLIGHT_NUMBER                   479
TAIL_NUMBER                    1369
DESTINATION_AIRPORT              59
SCHEDULED_DEPARTURE             416
DEPARTURE_TIME                 1207
TAXI_OUT                        138
WHEELS_OFF                     1219
SCHEDULED_TIME                  282
ELAPSED_TIME                    488
AIR_TIME                        427
DISTANCE                         58
WHEELS_ON                      1295
TAXI_IN                          58
SCHEDULED_ARRIVAL               738
ARRIVAL_TIME                   1300
DELAYED                           2
HOURLYVISIBILITY                 16
HOURLYDRYBULBTEMPF               51
HOURLYWETBULBTEMPF               52
HOURLYDewPointTempF              68
HOURLYRelativeHumidity           71
HOURLYWindSpeed                  31
HOURLYStationPressure           140
HOURLYSeaLevelPressure          138
HOURLYPrecip                     20
HOURLYAltimeterSetting          140
HOURLYWindDirectionSin           23
HOURLYWindDirectionCos           22
HOURLYPressureTendencyIncr        2
HOURLYPressureTendencyDecr        2
HOURLYPressureTendencyCons        2
dtype: int64
```

In [21]:

```
data_with_weather.to_csv("cleaned_test0.csv")
```

In [20]:
```
data_with_weather
```

Out[20]:

	DATETIME	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	DE
0	2015-01-01 06:00:00	1	1	4	B6	2023	N324JB	
1	2015-01-01 06:00:00	1	1	4	AA	2299	N3LLAA	
2	2015-01-01 06:00:00	1	1	4	B6	939	N794JB	
3	2015-01-01 06:00:00	1	1	4	B6	353	N570JB	
4	2015-01-01 06:00:00	1	1	4	B6	583	N531JB	
...	...	...	...	...	...	...	...	
16775	2015-03-07 09:00:00	3	7	6	B6	615	N942JB	
16776	2015-03-07 09:00:00	3	7	6	B6	208	N198JB	
16777	2015-03-07 09:00:00	3	7	6	B6	1634	N328JB	
16778	2015-03-07 09:00:00	3	7	6	DL	2474	N698DL	
16779	2015-03-07 09:00:00	3	7	6	DL	2600	N686DA	

16780 rows × 36 columns

In [21]:
```
features = data_with_weather.drop(['DELAYED', 'DATETIME','DESTINATION_AIRPORT','TAIL_NUMBER', 'AIRLINE', 'FLIGHT_NUMBER'], axis = 1)
target = data_with_weather['DELAYED']
```

In [22]:

```
features
```

Out[22]:

	MONTH	DAY	DAY_OF_WEEK	SCHEDULED_DEPARTURE	DEPARTURE_TIME	TAXI_OUT
0	1	1	4	535	618.0	13.0
1	1	1	4	545	640.0	17.0
2	1	1	4	545	545.0	17.0
3	1	1	4	600	554.0	16.0
4	1	1	4	600	557.0	16.0
...	...	...	...	...	...	...
16775	3	7	6	904	932.0	29.0
16776	3	7	6	907	903.0	11.0
16777	3	7	6	910	902.0	17.0
16778	3	7	6	910	1022.0	25.0
16779	3	7	6	915	1105.0	18.0

16780 rows × 30 columns

In [60]:

```
col = features.columns
```

In [23]:

```
target
```

Out[23]:

```
0        1.0
1        1.0
2        0.0
3        0.0
4        0.0
        ...
16775    1.0
16776    0.0
16777    0.0
16778    1.0
16779    1.0
Name: DELAYED, Length: 16780, dtype: float64
```

In [46]:

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    log_loss
)
```

In [34]:

```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.25)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)

model = LogisticRegression(max_iter = 1000000)
model.fit(X_train, y_train)
```

Out[34]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100000
0,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbo
se=0,
                   warm_start=False)
```

In [35]:

```
results = model.predict(X_test)
```

In [51]:

```
cm = confusion_matrix(y_test,results)
ps = precision_score(y_test, results)
rs = recall_score(y_test, results)
f1 = f1_score(y_test, results)
accuracy = accuracy_score(y_test, results)
error_score = 1 - accuracy

# accuracy score
print('Accuracy: ', accuracy)

# error score
print('Error: ', error_score)

# confusion matrix
print('Confusion Matrix: ')
print(cm)

# precision score
print('Precision: ', ps)

# recall score
print('recall: ',rs)

# f1 score
print('f1: ', f1)
```

```
Accuracy:  0.7573301549463647
Error:  0.2426698450536353
Confusion Matrix:
[[2869  140]
 [ 878  308]]
Precision:  0.6875
recall:  0.2596964586846543
f1:  0.3769889840881273
```

In [63]:

```
coefficients = pd.DataFrame(model.coef_)
coefficients.columns = col
coefficients = coefficients.transpose()
coefficients
```

Out[63]:

```
                                    0
MONTH                        0.163326
DAY                         -0.151517
DAY_OF_WEEK                  0.082438
SCHEDULED_DEPARTURE         -0.210547
DEPARTURE_TIME               0.704581
TAXI_OUT                     0.230144
WHEELS_OFF                   0.156255
SCHEDULED_TIME               1.141603
ELAPSED_TIME                -0.229657
AIR_TIME                    -0.263161
DISTANCE                    -0.742146
WHEELS_ON                   -0.282085
TAXI_IN                     -0.067659
SCHEDULED_ARRIVAL            0.196533
ARRIVAL_TIME                 0.005855
HOURLYVISIBILITY            -0.062388
HOURLYDRYBULBTEMPF           0.477551
HOURLYWETBULBTEMPF          -0.113846
HOURLYDewPointTempF         -1.107021
HOURLYRelativeHumidity       1.081997
HOURLYWindSpeed              0.070660
HOURLYStationPressure       -0.128967
HOURLYSeaLevelPressure       0.552012
HOURLYPrecip                 0.022300
HOURLYAltimeterSetting      -0.434483
HOURLYWindDirectionSin      -0.151030
HOURLYWindDirectionCos       0.143829
HOURLYPressureTendencyIncr  -0.036513
HOURLYPressureTendencyDecr   0.037445
HOURLYPressureTendencyCons  -0.005436
```

In [68]:

```python
# thresholds values
thresholds = [0.25, 0.5, 0.75, 0.9]

# function to test all thresholds
def thresholds_metrics(thresholds):
    model = LogisticRegression(max_iter=5000)
    model.fit(X_train, y_train)

    for i in thresholds:

        # if prediction proba is greater than threshold, set as 1, otherwise 0
        predictions = np.where(model.predict_proba(X_test)[:,1] > i, 1, 0)
        print('Thresholds: ', i)

        accuracy_rate = accuracy_score(y_test, predictions)
        print('Accuracy: ' + str(accuracy_rate))

        ps = precision_score(y_test, predictions)
        print('Precision: ', ps)

        rs = recall_score(y_test, predictions)
        print('recall: ',rs)
        print('--------------')
```

In [69]:

```python
thresholds_metrics(thresholds)
```

```
Thresholds:  0.25
Accuracy: 0.6395709177592371
Precision:  0.4184184184184184
recall:  0.7048903878583473
--------------
Thresholds:  0.5
Accuracy: 0.7573301549463647
Precision:  0.6875
recall:  0.2596964586846543
--------------
Thresholds:  0.75
Accuracy: 0.73492252681764
Precision:  0.8490566037735849
recall:  0.07588532883642496
--------------
Thresholds:  0.9
Accuracy: 0.7230035756853397
Precision:  0.9615384615384616
recall:  0.021079258010118045
--------------
```

In [70]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import plot_roc_curve, roc_auc_score
```

In [72]:

```
# sklearn Knn
def kNN(x_train, y_train, x_test, y_test):
    for i in range(1,10):
        print('k = ',i)
        knn = KNeighborsClassifier(n_neighbors = i)
        knn.fit(x_train,y_train)
        predictions = knn.predict(x_test)

        accuracy = accuracy_score(y_test, predictions)

        print("Accuracy: ", accuracy)
        print("error:", 1-accuracy)
```

In [74]:

```
kNN(X_train, y_train, X_test, y_test)
```

```
k =  1
Accuracy:  0.7253873659117998
error: 0.27461263408820025
k =  2
Accuracy:  0.7508939213349225
error: 0.2491060786650775
k =  3
Accuracy:  0.7492252681764004
error: 0.25077473182359955
k =  4
Accuracy:  0.7578069129916567
error: 0.2421930870083433
k =  5
Accuracy:  0.7520858164481525
error: 0.24791418355184747
k =  6
Accuracy:  0.7659117997616209
error: 0.23408820023837906
k =  7
Accuracy:  0.7539928486293206
error: 0.2460071513706794
k =  8
Accuracy:  0.767342073897497
error: 0.23265792610250302
k =  9
Accuracy:  0.7568533969010727
error: 0.24314660309892733
```

In [76]:

```
# run all four classifier
def four_classifiers(x_train, y_train, x_test, y_test):

    print('Logistic Regression')
    lgr = LogisticRegression(max_iter=1000)
    lgr.fit(x_train,y_train)
    log_predict = pd.DataFrame(lgr.predict(x_test))
    accuracy = accuracy_score(y_test, log_predict)

    print("Accuracy: ", accuracy)
    print("error:", 1-accuracy)

    print('LDA')
    lda = LinearDiscriminantAnalysis()
    lda.fit(x_train,y_train)
    lda_predict = lda.predict(x_test)
    accuracy = accuracy_score(y_test, lda_predict)

    print("Accuracy: ", accuracy)
    print("error:", 1-accuracy)


    print('KNearestNeighbor')
    knn = KNeighborsClassifier(n_neighbors = 8)
    knn.fit(x_train,y_train)
    knn_predict = knn.predict(x_test)
    accuracy = accuracy_score(y_test, knn_predict)

    print("Accuracy: ", accuracy)
    print("error:", 1-accuracy)

    print('Naive Bayes')
    nb = GaussianNB()
    nb.fit(x_train, y_train)
    naives_predict = nb.predict(x_test)
    accuracy = accuracy_score(y_test, naives_predict)

    print("Accuracy: ", accuracy)
    print("error:", 1-accuracy)

    return [lgr, lda, knn ,nb]
```

In [78]:

```
predictors = four_classifiers(X_train, y_train, X_test, y_test)
```

```
Logistic Regression
Accuracy:  0.7573301549463647
error: 0.2426698450536353
LDA
Accuracy:  0.7518474374255065
error: 0.24815256257449347
KNearestNeighbor
Accuracy:  0.767342073897497
error: 0.23265792610250302
Naive Bayes
Accuracy:  0.7158522050059595
error: 0.2841477949940405
```

In [ ]:

```
# produce roc graphs and prints auc
def roc_auc(predictors):
    predictor_names = ['LogisticRegression','LDA', 'KNN', 'NB']
    for i in range(4):
        pred = predictors[i]
        plot_roc_curve(pred,x_test,y_test)
        auc = roc_auc_score(y_test, pred.predict(x_test))
        print(predictor_names[i], 'AUC scores is ',auc)
```

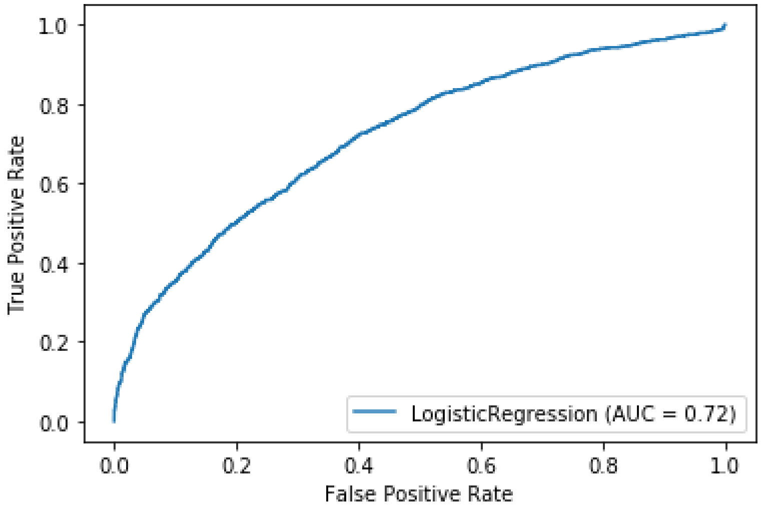
In [82]:

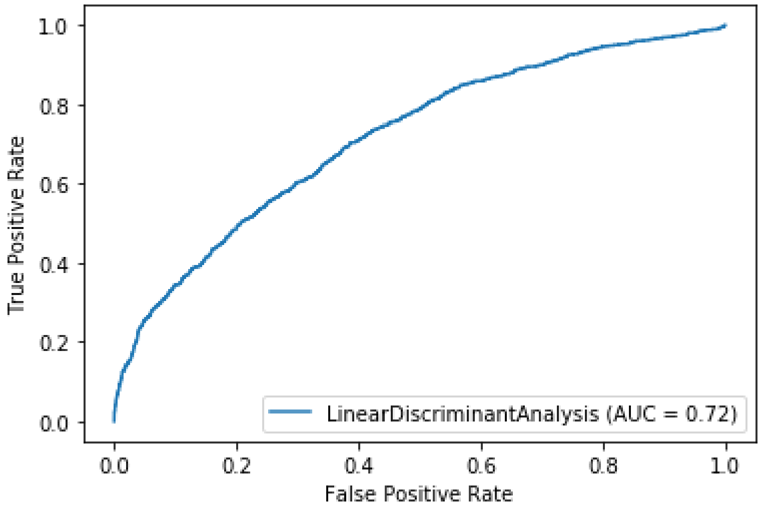
```
# produce roc graphs and prints auc
def roc_auc(predictors):
    predictor_names = ['LogisticRegression','LDA', 'KNN', 'NB']
    for i in range(4):
        pred = predictors[i]
        plot_roc_curve(pred,X_test,y_test)
        auc = roc_auc_score(y_test, pred.predict(X_test))
        print(predictor_names[i], 'AUC scores is ',auc)
```

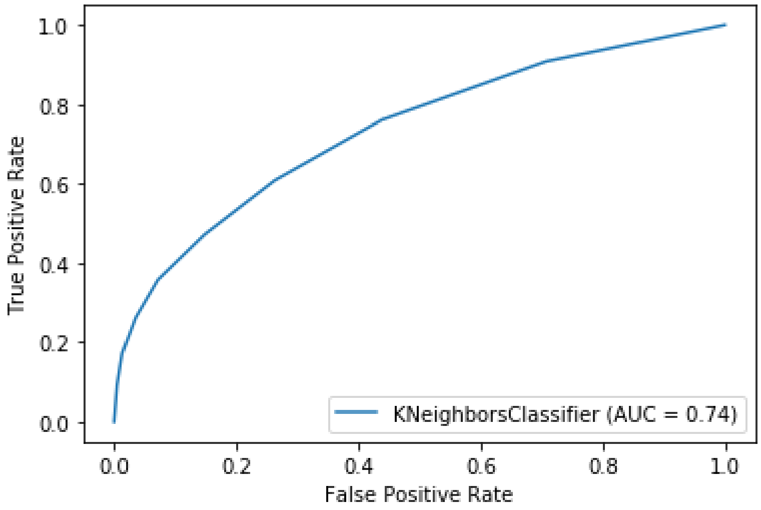
In [83]:

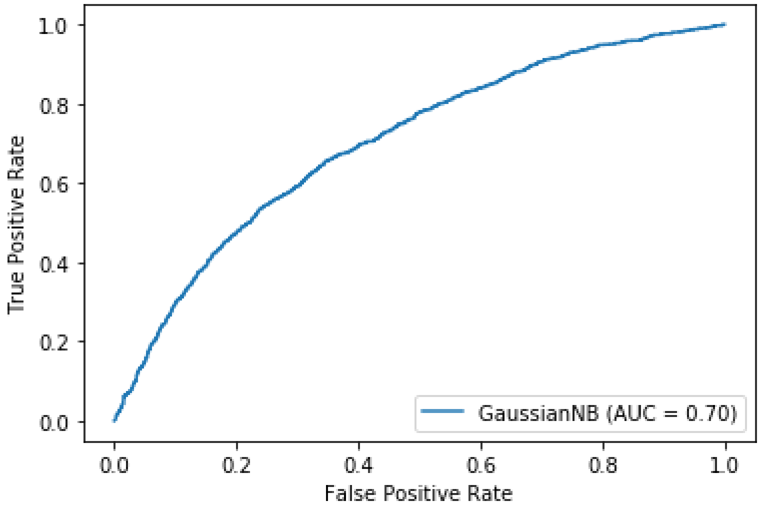
```
roc_auc(predictors)
```

```
LogisticRegression AUC scores is  0.6065846866371094
LDA AUC scores is  0.6035290699010333
KNN AUC scores is  0.643192121219254
NB AUC scores is  0.6310535790044145
```









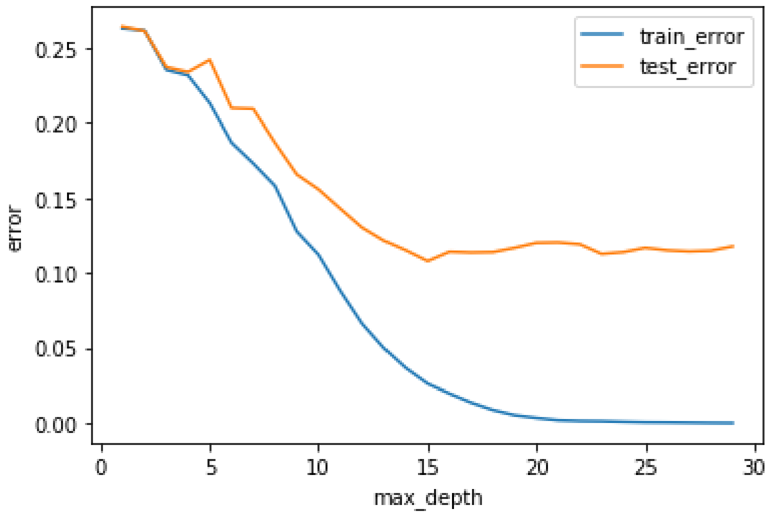
In [37]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier

import matplotlib.pyplot as plt
%matplotlib inline
```

In [42]:

```
print("-------------------------------------------")
print("----- DECISION TREE W/ VARIOUS DEPTHS -----")
print("-------------------------------------------")

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

train_error = []
test_error = []
max_depths = []

for depth in range(1, 30):

    max_depths.append(depth)

    dt = DecisionTreeClassifier(max_depth=depth)
    dt.fit(X_train, y_train)

    # ON TRAINING
    predict_label = dt.predict(X_train)

    c_matrix = confusion_matrix(y_train, predict_label)

    tp = c_matrix[1][1]
    tn = c_matrix[0][0]

    accuracy = (tp + tn) / len(predict_label)
    error = 1-accuracy

    train_error.append(error)

    # ON TESTING
    predict_label = dt.predict(X_test)

    c_matrix = confusion_matrix(y_test, predict_label)

    tp = c_matrix[1][1]
    tn = c_matrix[0][0]

    accuracy = (tp + tn) / len(predict_label)
    error = 1-accuracy

    test_error.append(error)


df = pd.DataFrame({'train_error':pd.Series(train_error),
                   'test_error':pd.Series(test_error),
                   'max_depth':pd.Series(max_depths)})

plt.plot('max_depth','train_error', data=df, label='train_error')
plt.plot('max_depth','test_error', data=df, label='test_error')
plt.xlabel('max_depth')
plt.ylabel('error')
plt.legend()
```

```
-------------------------------------------
----- DECISION TREE W/ VARIOUS DEPTHS -----
-------------------------------------------
```

```
Out[42]: <matplotlib.legend.Legend at 0x7f988f6eaf90>
```



In [85]:

```
print("-----------------------------------------")
print("----- DECISION TREE W/ MAX DEPTH 15 -----")
print("-----------------------------------------")

dt = DecisionTreeClassifier(max_depth = 15)
dt.fit(X_train, y_train)

# METRIC REPORTING

# ON TRAIN
print("\nFor Training Set:")
predict_label = dt.predict(X_train)

c_matrix = confusion_matrix(y_train, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

# metric calculation
accuracy = (tp + tn) / len(predict_label)
precision = (tp) / (tp + fp)
recall = (tp) / (tp + fn)
# avoid a division by 0 error
if precision+recall > 0:
    f1 = 2 * (precision*recall) / (precision+recall)
else:
    f1 = 0

print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))
print("The precision is: {}".format(precision))
print("The recall is: {}".format(recall))
print("The F1 score is: {}".format(f1))


# ON TEST
print("\nFor Testing Set:")
predict_label = dt.predict(X_test)

c_matrix = confusion_matrix(y_test, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

# metric calculation
accuracy = (tp + tn) / len(predict_label)
precision = (tp) / (tp + fp)
recall = (tp) / (tp + fn)
# avoid a division by 0 error
if precision+recall > 0:
    f1 = 2 * (precision*recall) / (precision+recall)
else:
    f1 = 0

print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))
print("The precision is: {}".format(precision))
print("The recall is: {}".format(recall))
print("The F1 score is: {}\n".format(f1))
```

```
-----------------------------------------
----- DECISION TREE W/ MAX DEPTH 15 -----
-----------------------------------------

For Training Set:

The accuracy is: 0.9751291219705999
The error is: 0.024870878029400134
The precision is: 0.9906962785114045
The recall is: 0.921295004186436
The F1 score is: 0.9547360809833695

For Testing Set:

The accuracy is: 0.8893921334922527
The error is: 0.11060786650774734
The precision is: 0.7794117647058824
The recall is: 0.8490725126475548
The F1 score is: 0.8127522195318806
```

In [92]:

```
print("-------------------------")
print("----- RANDOM FOREST -----")
print("-------------------------\n")

estimators = [10, 50, 100]

for estimator in estimators:

    print("Estimators: {}".format(estimator))
    rf = RandomForestClassifier(n_estimators=estimator)
    rf = rf.fit(X_train, y_train)

    # METRIC REPORTING

    # ON TRAIN
    print("\nFor Training Set:")
    predict_label = rf.predict(X_train)

    c_matrix = confusion_matrix(y_train, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    # metric calculation
    accuracy = (tp + tn) / len(predict_label)
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
    # avoid a division by 0 error
    if precision+recall > 0:
        f1 = 2 * (precision*recall) / (precision+recall)
    else:
        f1 = 0

    print("\nThe accuracy is: {}".format(accuracy))
    print("The error is: {}".format(1-accuracy))
    print("The precision is: {}".format(precision))
    print("The recall is: {}".format(recall))
    print("The F1 score is: {}".format(f1))


    # ON TEST
    print("\nFor Testing Set:")
    predict_label = rf.predict(X_test)

    c_matrix = confusion_matrix(y_test, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    # metric calculation
    accuracy = (tp + tn) / len(predict_label)
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
    # avoid a division by 0 error
    if precision+recall > 0:
        f1 = 2 * (precision*recall) / (precision+recall)
    else:
        f1 = 0

    print("\nThe accuracy is: {}".format(accuracy))
    print("The error is: {}".format(1-accuracy))
    print("The precision is: {}".format(precision))
    print("The recall is: {}".format(recall))
    print("The F1 score is: {}\n".format(f1))
```

```
-------------------------
----- RANDOM FOREST -----
-------------------------

Estimators: 10

For Training Set:

The accuracy is: 0.9933253873659118
The error is: 0.006674612634088195
The precision is: 0.9991440798858773
The recall is: 0.977393245883338
The F1 score is: 0.9881489841986456

For Testing Set:

The accuracy is: 0.865554231227652
The error is: 0.13444576877234804
The precision is: 0.8402625820568927
The recall is: 0.6475548060708263
The F1 score is: 0.7314285714285714

Estimators: 50

For Training Set:

The accuracy is: 0.999761620977354
The error is: 0.00023837902264600697
The precision is: 1.0
The recall is: 0.999162712810494
The F1 score is: 0.9995811810693843

For Testing Set:

The accuracy is: 0.8750893921334922
The error is: 0.12491060786650776
The precision is: 0.8566810344827587
The recall is: 0.6703204047217538
The F1 score is: 0.7521286660359509

Estimators: 100

For Training Set:

The accuracy is: 1.0
The error is: 0.0
The precision is: 1.0
The recall is: 1.0
The F1 score is: 1.0

For Testing Set:

The accuracy is: 0.8777115613825983
The error is: 0.12228843861740168
The precision is: 0.8560846560846561
The recall is: 0.6821247892074199
The F1 score is: 0.7592679493195683
```

In [45]:

```python
print("---------------------")
print("----- ADA BOOST -----")
print("---------------------\n")

estimators = [10, 50, 100, 500, 1000, 5000]

for estimator in estimators:

    print("Estimators: {}".format(estimator))
    ada = AdaBoostClassifier(n_estimators=estimator)
    ada.fit(X_train, y_train)

    # METRIC REPORTING

    # ON TRAIN
    print("\nFor Training Set:")
    predict_label = ada.predict(X_train)

    c_matrix = confusion_matrix(y_train, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    # metric calculation
    accuracy = (tp + tn) / len(predict_label)
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
    # avoid a division by 0 error
    if precision+recall > 0:
        f1 = 2 * (precision*recall) / (precision+recall)
    else:
        f1 = 0

    print("\nThe accuracy is: {}".format(accuracy))
    print("The error is: {}".format(1-accuracy))
    print("The precision is: {}".format(precision))
    print("The recall is: {}".format(recall))
    print("The F1 score is: {}".format(f1))


    # ON TEST
    print("\nFor Testing Set:")
    predict_label = ada.predict(X_test)

    c_matrix = confusion_matrix(y_test, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    # metric calculation
    accuracy = (tp + tn) / len(predict_label)
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
    # avoid a division by 0 error
    if precision+recall > 0:
        f1 = 2 * (precision*recall) / (precision+recall)
    else:
        f1 = 0

    print("\nThe accuracy is: {}".format(accuracy))
    print("The error is: {}".format(1-accuracy))
    print("The precision is: {}".format(precision))
    print("The recall is: {}".format(recall))
    print("The F1 score is: {}\n".format(f1))
```

```
---------------------
----- ADA BOOST -----
---------------------

Estimators: 10

For Training Set:

The accuracy is: 0.7508144616607072
The error is: 0.2491855383392928
The precision is: 0.636697247706422
The recall is: 0.2905386547585822
The F1 score is: 0.39900344959754697

For Testing Set:

The accuracy is: 0.7489868891537544
The error is: 0.25101311084624556
The precision is: 0.624765478424015
The recall is: 0.28077571669477236
The F1 score is: 0.387434554973822

Estimators: 50

For Training Set:

The accuracy is: 0.8045292014302742
The error is: 0.19547079856972582
The precision is: 0.7781079742446756
The recall is: 0.438459391571309
The F1 score is: 0.5608711174580507

For Testing Set:

The accuracy is: 0.8019070321811681
The error is: 0.1980929678188319
The precision is: 0.7598828696925329
The recall is: 0.4376053962900506
The F1 score is: 0.5553772070626004

Estimators: 100

For Training Set:

The accuracy is: 0.8235995232419547
The error is: 0.17640047675804527
The precision is: 0.8345606283750614
The recall is: 0.474462740720067
The F1 score is: 0.6049822064056939

For Testing Set:

The accuracy is: 0.8181168057210966
The error is: 0.18188319427890343
The precision is: 0.8008534850640113
The recall is: 0.47470489038785835
The F1 score is: 0.5960825833774483
```

```
Estimators: 500

For Training Set:

The accuracy is: 0.8675407230830353
The error is: 0.13245927691696469
The precision is: 0.9049027895181742
The recall is: 0.597543957577449
The F1 score is: 0.7197848377878636

For Testing Set:

The accuracy is: 0.8545887961859356
The error is: 0.14541120381406436
The precision is: 0.8555555555555555
The recall is: 0.5843170320404721
The F1 score is: 0.6943887775551102

Estimators: 1000

For Training Set:

The accuracy is: 0.8833531982518872
The error is: 0.11664680174811282
The precision is: 0.9184804115552038
The recall is: 0.6477811889478091
The F1 score is: 0.7597381342062193

For Testing Set:

The accuracy is: 0.865315852205006
The error is: 0.13468414779499405
The precision is: 0.8564867967853043
The recall is: 0.6290050590219224
The F1 score is: 0.7253281477880409

Estimators: 5000

For Training Set:

The accuracy is: 0.9225268176400476
The error is: 0.07747318235995238
The precision is: 0.9396493594066082
The recall is: 0.7778397990510745
The F1 score is: 0.8511223087494274

For Testing Set:

The accuracy is: 0.8841477949940405
The error is: 0.1158522050059595
The precision is: 0.8211009174311926
The recall is: 0.7546374367622259
The F1 score is: 0.7864674868189807
```

```
In [ ]:
```