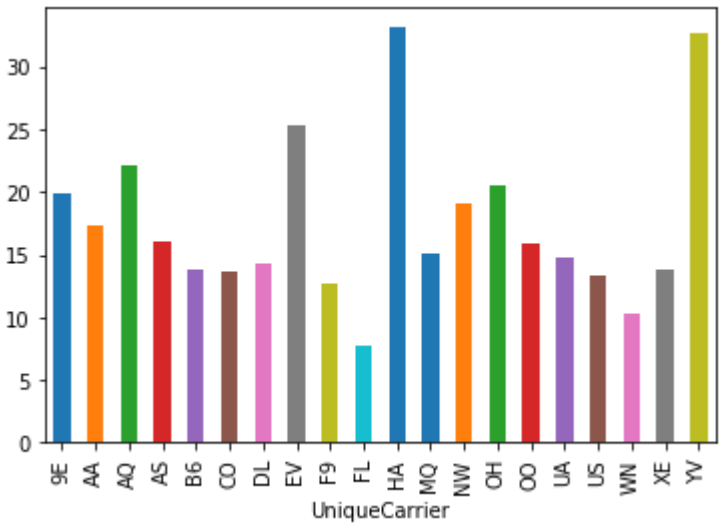
In [1]:
```
### Data Loading

import csv
import numpy as np
import pandas as pd
import time
from datetime import datetime
%matplotlib inline

df = pd.read_csv('/user/ss186102/2008.csv')
```

In [2]:
```
### Creation of Date field
df['date1'] = df.Year.astype(str).str.cat(df.Month.astype(str).str.zfill(2), s
ep='-').str.cat(df.DayOfWeek.astype(str).str.zfill(2), sep='-')

df['date1'] = df['date1'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d'))
```

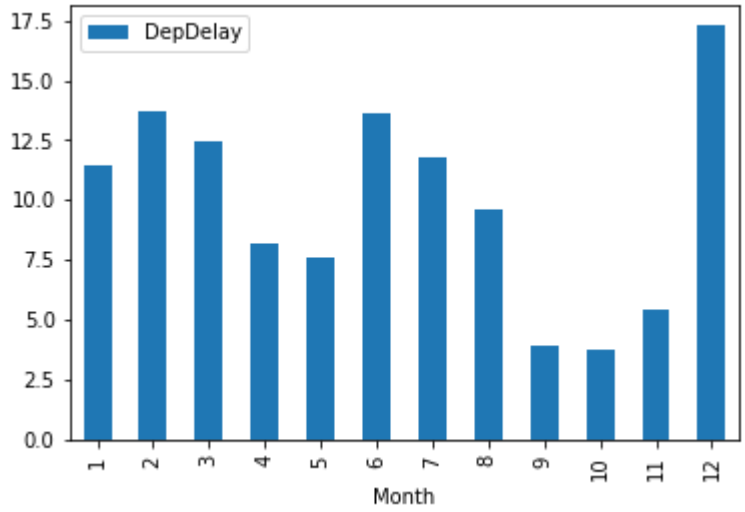
In [3]:
```
#Which carrier performs better
grouped= df['CarrierDelay'].groupby(df['UniqueCarrier']).mean()
#grouped.sort_values(['CarrierDelay'], ascending=True).plot(kind="bar")
grouped.plot(kind="bar")
print(grouped[grouped.min()==grouped])
```

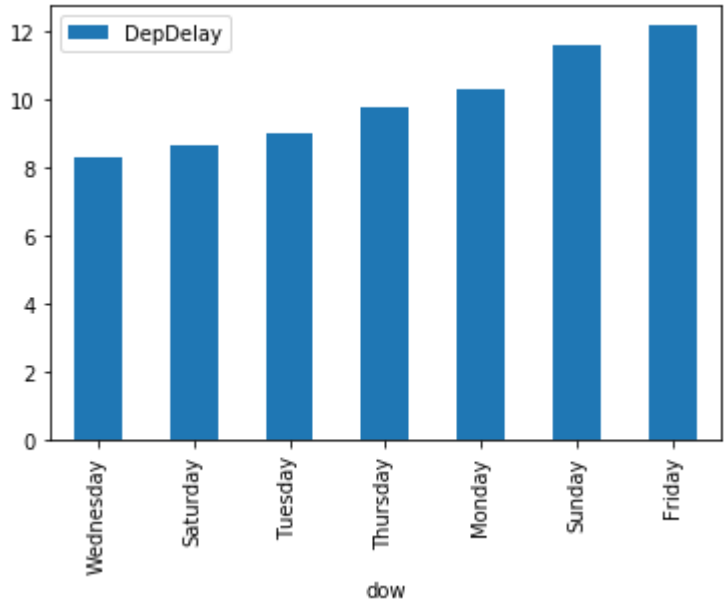
```
UniqueCarrier
FL    7.660308
Name: CarrierDelay, dtype: float64
```

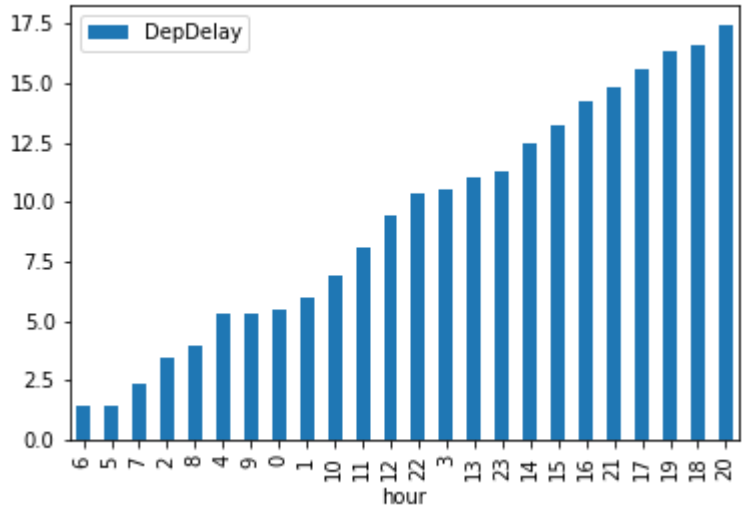


In [4]:
```
## When is the best day to minimise delay
Month_delays = df[['DepDelay', 'Month']].groupby('Month').mean()
Month_delays.plot(kind="bar")
di = {1:'Monday',2:'Tuesday',3:'Wednesday',4:'Thursday',5:'Friday',6:'Saturda
y',7:'Sunday'}
df['dow']= df['DayOfWeek'].map(di)
DayOfWeek_delays = df[['DepDelay', 'dow']].groupby('dow').mean()
#DayOfWeek_delays = df[['CarrierDelay', 'DayOfWeek']].groupby('DayOfWeek').mea
n()
DayOfWeek_delays.sort_values(by=['DepDelay'], ascending=True).plot(kind="bar")

df['hour'] = df['CRSDepTime'].map(lambda x: int(str(int(x)).zfill(4)[:2]))
hour_delays = df[['DepDelay', 'hour']].groupby('hour').mean()

# Plotting average delays by hour of day
hour_delays.sort_values(by=['DepDelay'], ascending=True).plot(kind='bar')
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f7fbb0940>
```

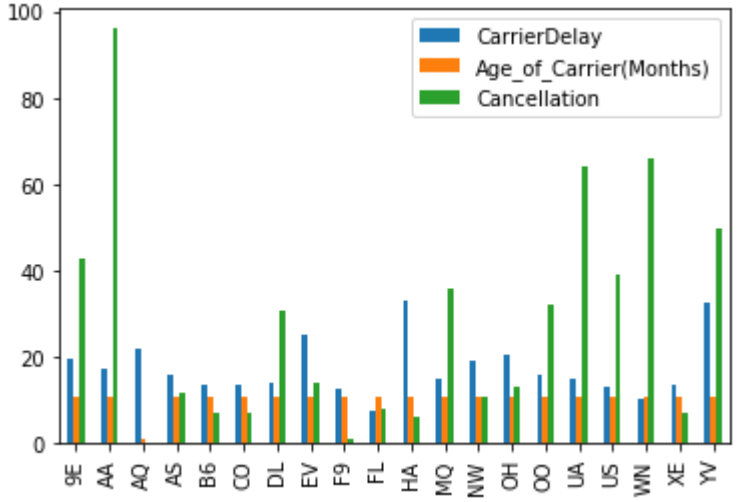






In [13]:
```
# Calculation of age of Carrier
group = df[['date1']].groupby(df['UniqueCarrier'])
carrier_duration = group.max()-group.min()


carrier_duration.rename(columns = {'date1':'Age_of_Carrier(Months)'}, inplace
= True)
carrier_duration['Age_of_Carrier(Months)'] = carrier_duration['Age_of_Carrier
(Months)'].map(lambda x: round(int(str(x).replace('days','').split()[0])/30))

# Carrier Delay
grouped= df['CarrierDelay'].groupby(df['UniqueCarrier'])
mean=grouped.mean()

## Cancellation of flight due to Carrier only
gr1= df[df['CancellationCode']=='A'].UniqueCarrier.value_counts()
gr1= round(gr1[:]/100)
gr1 = pd.DataFrame(gr1)
gr1.rename(columns = {'UniqueCarrier':'Cancellation'}, inplace = True)

## Merging of data sets
df1 = pd.concat([mean, carrier_duration,gr1], axis=1)
df1.plot(kind = 'bar')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f733c91d0>



In [37]:
```
## Delay Propogation
df1= df[['date1','DepTime','CRSDepTime','ArrTime', 'CRSArrTime', 'UniqueCarrie
r','FlightNum','DepDelay','ArrDelay','Origin', 'Dest',]]

df2 =pd.merge(df1, df1, left_on =['date1','UniqueCarrier','FlightNum','Dest'],
 right_on = ['date1','UniqueCarrier','FlightNum','Origin'],how='inner')

import operator
df2.loc[operator.and_(df2.DepDelay_x > 15, df2.DepDelay_y > 15 )].head()
```

Out[37]:

	date1	DepTime_x	CRSDepTime_x	ArrTime_x	CRSArrTime_x	UniqueCarrier	Flight
21	2008-01-04	1031.0	925	1125.0	1025	WN	535
22	2008-01-04	1031.0	925	1125.0	1025	WN	535
60	2008-01-04	1446.0	1430	1435.0	1425	WN	829
74	2008-01-04	1452.0	1425	1640.0	1625	WN	675
75	2008-01-04	1452.0	1425	1640.0	1625	WN	675

In [38]:
```
### Devloping a Model

df = df[['UniqueCarrier', 'Origin', 'Dest',
        'CRSDepTime', 'DepTime', 'DepDelay',
        'CRSArrTime', 'ArrTime', 'ArrDelay',
        'CRSElapsedTime', 'ActualElapsedTime','AirTime',]]

missing_df = df.isnull().sum(axis=0).reset_index()
missing_df.columns = ['variable', 'missing values']
missing_df['filling factor (%)']=(df.shape[0]-missing_df['missing values'])/df
.shape[0]*100
missing_df.sort_values('filling factor (%)').reset_index(drop = True)
```

Out[38]:

	variable	missing values	filling factor (%)
0	ArrDelay	154699	97.793081
1	ActualElapsedTime	154699	97.793081
2	AirTime	154699	97.793081
3	ArrTime	151649	97.836592
4	DepTime	136246	98.056330
5	DepDelay	136246	98.056330
6	CRSElapsedTime	844	99.987960
7	UniqueCarrier	0	100.000000
8	Origin	0	100.000000
9	Dest	0	100.000000
10	CRSDepTime	0	100.000000
11	CRSArrTime	0	100.000000

In [39]:
```
# 97% it was filled so dropped few
df.dropna(inplace = True)
```

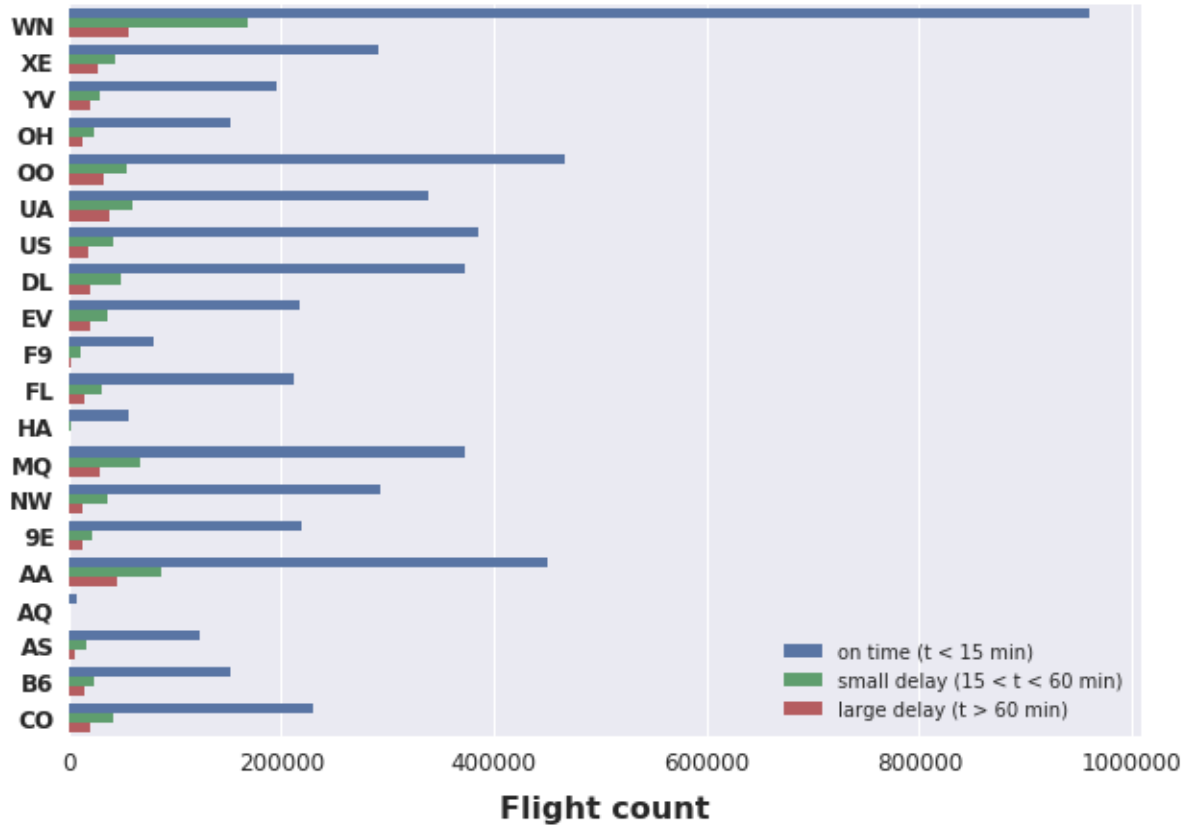
In [40]:
```
# function that extract statistical parameters from a grouby objet:
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
#_____________________________________________________________________
# Creation of a dataframe with statitical infos on each airline:
global_stats = df['DepDelay'].groupby(df['UniqueCarrier']).apply(get_stats).un
stack()
global_stats = global_stats.sort_values('count')
global_stats
```

Out[40]:

	count	max	mean	min
UniqueCarrier				
AQ	7752.0	336.0	-1.482456	-61.0
HA	61212.0	963.0	0.439211	-534.0
F9	95384.0	817.0	5.903107	-25.0
AS	148492.0	947.0	6.717439	-79.0
OH	190695.0	960.0	11.510150	-70.0
B6	192114.0	846.0	12.572827	-70.0
YV	245131.0	607.0	11.952744	-92.0
9E	254322.0	1127.0	6.733861	-42.0
FL	258713.0	1206.0	9.229818	-62.0
EV	274867.0	965.0	11.922875	-61.0
CO	293855.0	1011.0	13.135781	-28.0
NW	344110.0	2467.0	6.438267	-35.0
XE	363414.0	859.0	11.328870	-71.0
UA	437979.0	1303.0	14.064279	-34.0
DL	443934.0	1003.0	7.975474	-38.0
US	446086.0	886.0	5.687228	-40.0
MQ	471161.0	1710.0	10.628900	-40.0
OO	553412.0	996.0	7.406290	-67.0
AA	585485.0	1521.0	13.228520	-64.0
WN	1186911.0	668.0	10.323081	-38.0

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns
delay_type = lambda x:((0,1)[x > 15],2)[x > 60]


df['DELAY_LEVEL'] = df['DepDelay'].apply(delay_type)

fig = plt.figure(1, figsize=(10,7))
ax = sns.countplot(y="UniqueCarrier", hue='DELAY_LEVEL', data=df)
# We replace the abbreviations by the full names of the companies and set the
 labels
labels = ax.get_yticklabels()
ax.set_yticklabels(labels)
plt.setp(ax.get_xticklabels(), fontsize=12, weight = 'normal', rotation = 0);
plt.setp(ax.get_yticklabels(), fontsize=12, weight = 'bold', rotation = 0);
ax.yaxis.label.set_visible(False)
plt.xlabel('Flight count', fontsize=16, weight = 'bold', labelpad=10)
#_________________
# Set the legend
L = plt.legend()
L.get_texts()[0].set_text('on time (t < 15 min)')
L.get_texts()[1].set_text('small delay (15 < t < 60 min)')
L.get_texts()[2].set_text('large delay (t > 60 min)')
plt.show()
```

```
/opt/anaconda3/lib/python3.6/site-packages/seaborn/categorical.py:1468: Futur
eWarning: remove_na is deprecated and is a private function. Do not use.
  stat_data = remove_na(group_data[hue_mask])
/opt/anaconda3/lib/python3.6/site-packages/matplotlib/font_manager.py:1320: U
serWarning: findfont: Font family ['sans-serif'] not found. Falling back to D
ejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



In [43]:
```
# Import
from sklearn.preprocessing import LabelEncoder
import matplotlib
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn.preprocessing import Imputer
from sklearn.decomposition import PCA
from sklearn import linear_model, decomposition, datasets
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split,  GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import cross_validation
from sklearn.metrics import confusion_matrix, roc_curve
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
get_ipython().magic('matplotlib inline')
matplotlib.rcParams.update({'font.size': 12})
matplotlib.rc('xtick', labelsize=8)
matplotlib.rc('ytick', labelsize=8)
np.random.seed(10)
df = pd.read_csv('/user/ss186102/2008.csv')
```

In [51]:
```
df1= df[['DayofMonth', 'DayOfWeek', 'CRSDepTime',
       'CRSArrTime', 'CRSElapsedTime',
        'AirTime', 'Distance',
       'ArrDelay']]
missing_df = df1.isnull().sum(axis=0).reset_index()
missing_df.columns = ['variable', 'missing values']
missing_df['filling factor (%)']=(df1.shape[0]-missing_df['missing values'])/d
f1.shape[0]*100
missing_df.sort_values('filling factor (%)').reset_index(drop = True)
df1.dropna(inplace = True)
```

Out[51]:

	DayofMonth	DayOfWeek	CRSDepTime	CRSArrTime	CRSElapsedTime	AirTim
0	3	4	1955	2225	150.0	116.0
1	3	4	735	1000	145.0	113.0
2	3	4	620	750	90.0	76.0
3	3	4	930	1100	90.0	78.0
4	3	4	1755	1925	90.0	77.0
5	3	4	1915	2110	115.0	87.0
6	3	4	1830	1940	250.0	230.0
7	3	4	1040	1150	250.0	219.0
8	3	4	615	650	95.0	70.0
9	3	4	1620	1655	95.0	70.0
10	3	4	700	915	135.0	106.0
11	3	4	1510	1725	135.0	107.0
12	3	4	1430	1425	55.0	39.0
13	3	4	715	710	55.0	37.0
14	3	4	1700	1655	55.0	35.0
15	3	4	1020	1010	50.0	37.0
16	3	4	1425	1625	240.0	213.0
17	3	4	745	955	250.0	205.0
18	3	4	1255	1510	135.0	110.0
19	3	4	1325	1435	70.0	49.0
20	3	4	705	810	65.0	51.0
21	3	4	1625	1735	70.0	47.0
22	3	4	1840	1950	70.0	49.0
23	3	4	1030	1140	70.0	47.0
24	3	4	800	910	70.0	53.0
25	3	4	1455	1605	70.0	50.0
26	3	4	1255	1610	195.0	143.0
27	3	4	1925	2235	190.0	155.0
28	3	4	635	945	190.0	147.0
29	3	4	730	1020	350.0	314.0
...	...	...	...	...	...	...
7009698	13	6	1635	1758	83.0	46.0

	DayofMonth	DayOfWeek	CRSDepTime	CRSArrTime	CRSElapsedTime	AirTim
7009699	13	6	1221	1359	98.0	64.0
7009700	13	6	1845	2006	81.0	52.0
7009701	13	6	1500	1642	102.0	79.0
7009702	13	6	1522	1823	121.0	88.0
7009703	13	6	1910	2016	66.0	38.0
7009704	13	6	1445	1622	97.0	65.0
7009705	13	6	830	1008	98.0	82.0
7009706	13	6	1440	1704	84.0	56.0
7009707	13	6	1755	2015	140.0	113.0
7009708	13	6	710	837	87.0	49.0
7009709	13	6	1520	1718	58.0	27.0
7009710	13	6	1220	1552	152.0	120.0
7009711	13	6	1041	1303	82.0	58.0
7009712	13	6	843	1021	98.0	53.0
7009713	13	6	815	1526	251.0	210.0
7009714	13	6	545	650	65.0	38.0
7009715	13	6	850	1005	135.0	108.0
7009716	13	6	936	1119	103.0	70.0
7009717	13	6	600	749	109.0	78.0
7009718	13	6	847	1010	143.0	122.0
7009719	13	6	640	753	73.0	50.0
7009720	13	6	800	1026	86.0	56.0
7009721	13	6	615	907	112.0	103.0
7009722	13	6	750	859	69.0	41.0
7009723	13	6	959	1150	111.0	71.0
7009724	13	6	835	1023	168.0	139.0
7009725	13	6	700	856	116.0	85.0
7009726	13	6	1240	1437	117.0	89.0
7009727	13	6	1103	1418	135.0	104.0

7009728 rows × 8 columns

In [59]:
```
df1['Arrival_delay_classifier']= df1['ArrDelay'].map(lambda x: 0 if x>15 else
1)

df1.drop('ArrDelay',axis = 1,inplace=True)
```

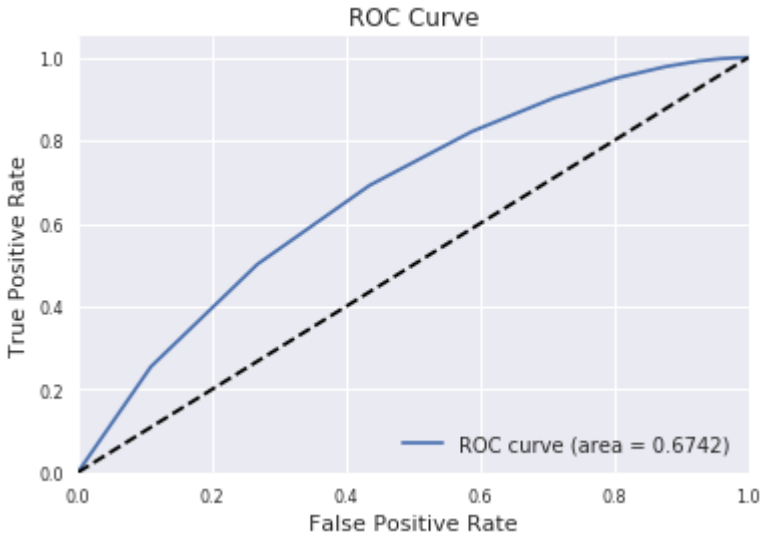
In [62]:
```
from sklearn.model_selection import train_test_split
train, valid = train_test_split(df1, test_size = 0.3,random_state=1991)
train=pd.DataFrame(train,columns=df1.columns)
valid=pd.DataFrame(valid,columns=df1.columns)
X_train=train.drop(['Arrival_delay_classifier'],axis=1)
Y_train=train['Arrival_delay_classifier']
X_valid=valid.drop(['Arrival_delay_classifier'],axis=1)
Y_valid=valid['Arrival_delay_classifier']
```

In [67]:
```
from sklearn.metrics import roc_curve, auc
def Performance(Model,Y,X):
    # Perforamnce of the model
    fpr, tpr, _ = roc_curve(Y, Model.predict_proba(X)[:,1])
    AUC  = auc(fpr, tpr)
    print ('the AUC is : %0.4f' %  AUC)
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.4f)' % AUC)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()

#Performance(Model=RF0,Y=Y_valid,X=X_valid)
```

In [72]:
```
RF0 = RandomForestClassifier(n_jobs = 1,random_state =1)
RF1= RF0.fit(X_train,Y_train)
```

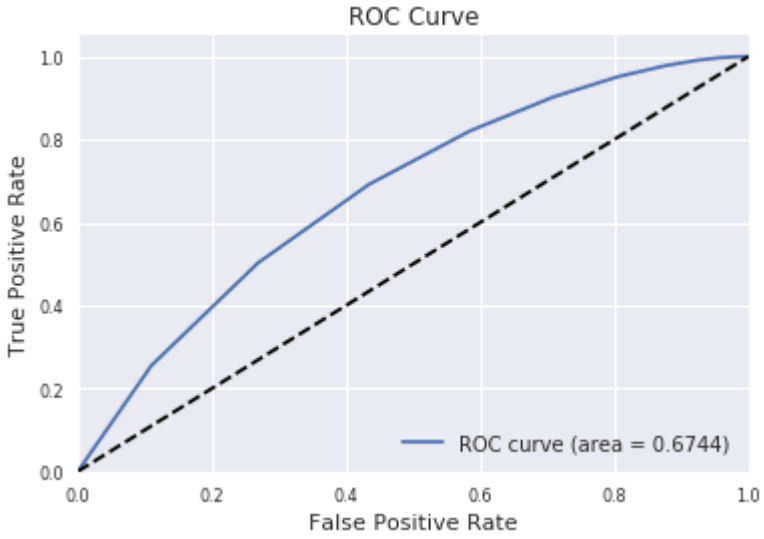
In [73]:
```
Performance(Model=RF1,Y=Y_valid,X=X_valid)
```

```
the AUC is : 0.6742
```

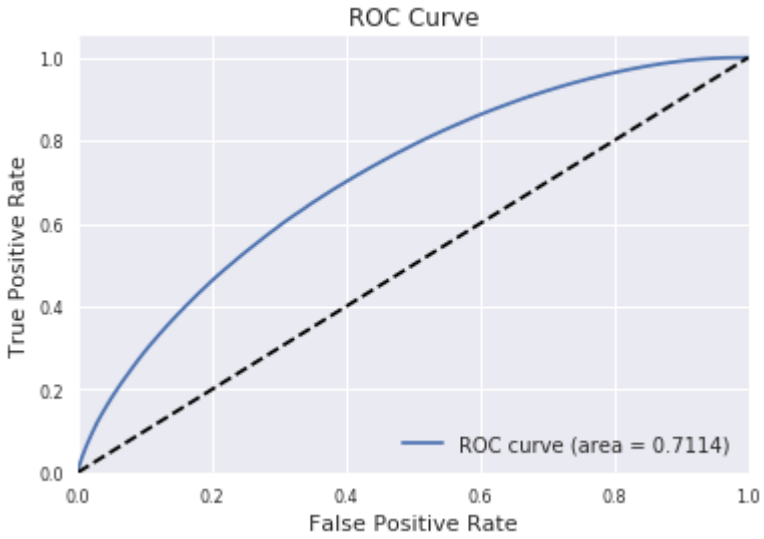


In [77]:
```
sample_leaf_options = [1,10,20,75,100,200,500]
for leaf_size in sample_leaf_options :
    RF0 = RandomForestClassifier( n_jobs = -1,random_state =50,
                                     max_features = "auto", min_samples_leaf = l
eaf_size)
    RF1= RF0.fit(X_train,Y_train)
    Performance(Model=RF1,Y=Y_valid,X=X_valid)
```

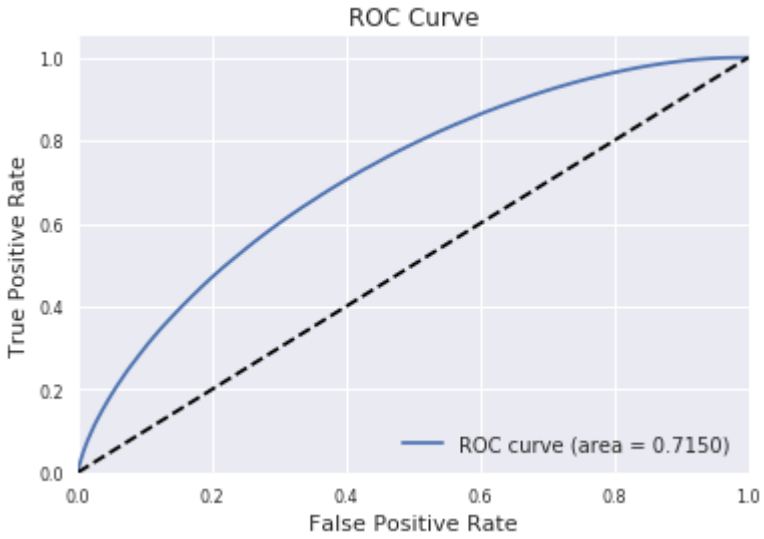
```
the AUC is : 0.6744
```



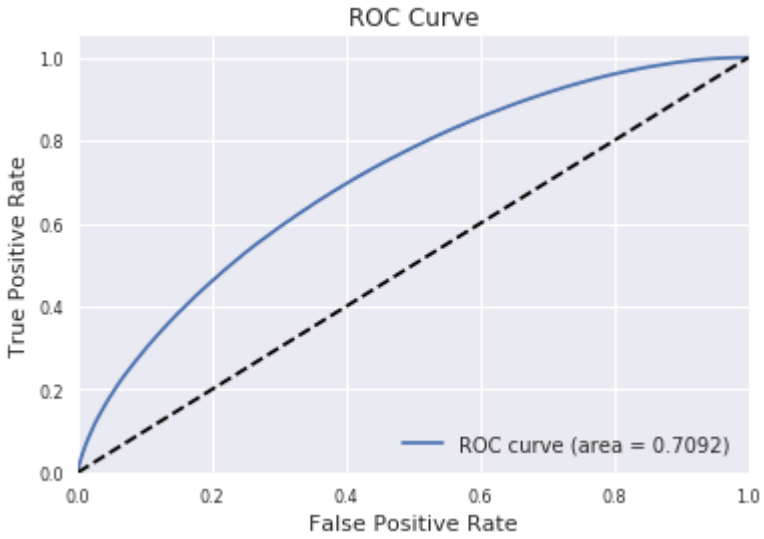

```
the AUC is : 0.7114
```



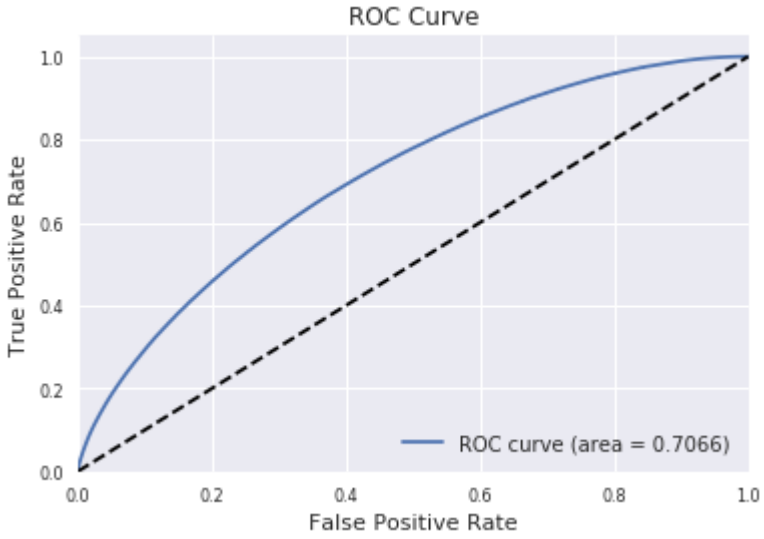

```
the AUC is : 0.7150
```



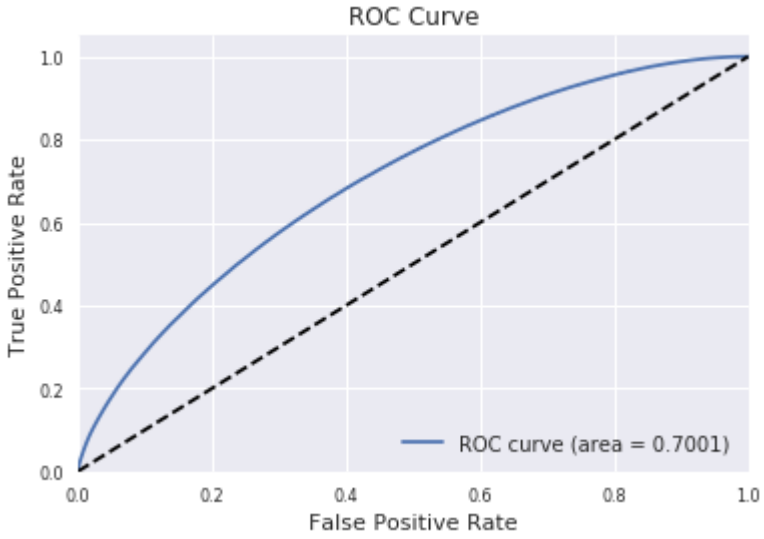

```
the AUC is : 0.7092
```



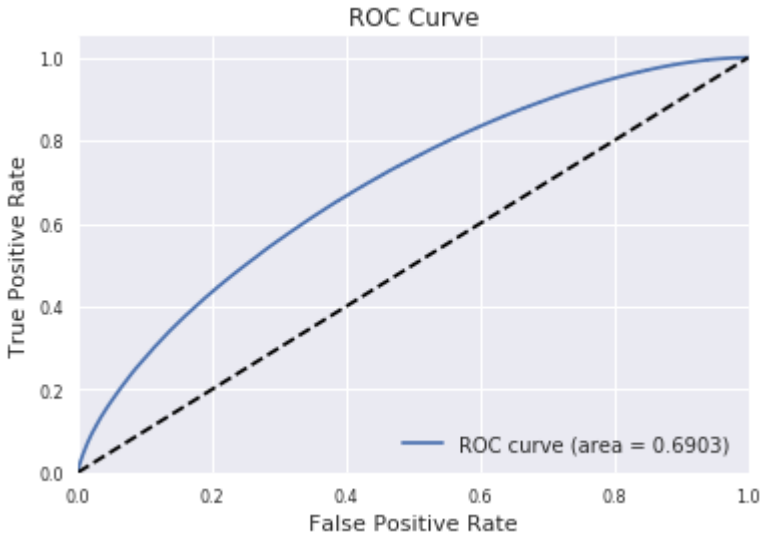

```
the AUC is : 0.7066
```




```
the AUC is : 0.7001
```




```
the AUC is : 0.6903
```


ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.6903)
0.0
0.2
0.4
0.6
0.8
1.0

In [81]:
```
RF0 = RandomForestClassifier( n_jobs = -1,random_state =50,
                                     max_features = "auto", min_samples_leaf = 2
0)
RF1= RF0.fit(X_train,Y_train)

from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, classification_report, \
    accuracy_score, f1_score

prediction = RF1.predict(X_valid)

print ('Accuracy:', accuracy_score(Y_valid, prediction))
print ('F1 score:', f1_score(Y_valid, prediction))
print ('Recall:', recall_score(Y_valid, prediction))
print ('Precision:', precision_score(Y_valid, prediction))
print ('\n clasification report:\n', classification_report(Y_valid,prediction
))
print ('\n confussion matrix:\n',confusion_matrix(Y_valid, prediction))

Performance(Model=RF1,Y=Y_valid,X=X_valid)
```

```
Accuracy: 0.801778645267
F1 score: 0.886738058073
Recall: 0.987081118329
Precision: 0.804913493064

 clasification report:
              precision    recall  f1-score   support

          0       0.72      0.12      0.21    439883
          1       0.80      0.99      0.89   1616626

avg / total       0.79      0.80      0.74   2056509


 confussion matrix:
 [[  53124  386759]
 [  20885 1595741]]
the AUC is : 0.7150
```