CS 109A/STAT 121A/AC 209A/CSCI E-109A:

Midterm - 2017

Harvard University
Fall 2017
Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave, Margo Levine

---

INSTRUCTIONS

- You must submit the Midterm on your own. **No group submissions are allowed**. You may use any print or online resources but **you may not work or consult with others**.
- Restart the kernel and run the whole notebook again before you submit.
- Please submit both a notebook and a pdf.

Flight Delays

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights are published in DOT's monthly Air Travel Consumer Report and in this dataset of 2015 flight delays and cancellations.

Data

Each entry of the flights.csv file corresponds to a flight. More than 5,800,000 flights were recorded in 2015. These flights are described according to 31 variables. Further details of these variables can be found here (https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time), if you are interested (not needed to answer these questions).

Name	Type	DESCRIPTION
DATE	object	The date in python datetime format
MONTH	int64	The month of the year(1-12)
DAY	int64	The day of the month
DAY_OF_WEEK	int64	The day of the week(1-7, MON-SUN)
AIRLINE	object	An identifier for the airline
FLIGHT_NUMBER	int64	The flight number
TAIL_NUMBER	object	The tail number (aircraft) corresponding to this flight
ORIGIN_AIRPORT	object	The code for origin airport
DESTINATION_AIRPORT	object	The code for destination airport
SCHED_DEP	object	The departure time in python datetime.time format
SCHED_ARR	object	The arrival time in python datetime.time format
DEPARTURE_DELAY	float64	The delay incurred at the origin (mins)
ARRIVAL_DELAY	float64	The delay when the flight reached the (mins) destination
DISTANCE	int64	Distance in miles between origin and destination
SCHEDULED_TIME	float64	Scheduled time of flight (minutes)
ELAPSED_TIME	float64	Actual time of flight (minutes)
AIR_SYSTEM_DELAY	float64	What part of the delay was NASD?(mins)
SECURITY_DELAY	float64	What part of the delay was due to security problems? (mins)
AIRLINE_DELAY	float64	What part of the delay is due to the airline? (mins)
LATE_AIRCRAFT_DELAY	float64	What part of the delay is due to previous flight(s) being late(mins)
WEATHER_DELAY	float64	Delay due to extreme weather events(min)

You can read more about the various weather delays here (https://www.rita.dot.gov/bts/help/aviation/html/understanding.html) if you are so inclined.

Data/Caveats

The data file, flights.csv, is found here (https://drive.google.com/file/d/0B9dVesTppCgHY0IwZHk3SGhjd00/view?usp=sharing) (note, it is about 70MB).

This data is already preprocessed, reduced, partially cleaned and therefore not identical to the original dataset.

Problem Description

We will build two separate models: one model that classifies whether a flight will be delayed and a second model that predicts the length of delay given that a flight is truly delayed. Only consider models taught in class so far.

Consider the following: This is a large dataset; think of strategies on how to solve this problem. Create a manageable subsample of the data that you can use to train and test/validate, but eventually you should predict on all the data (excluding the training set).

Questions

1. (5pts) Create a new variable, `DELAY_OR_NOT`: a boolean/indicator variable which indicates any arrival delay under 15 mins as a 0, and any delay at or above 15 mins as a 1 (`ARRIVAL_DELAY >= 15`).
2. (5pts) Make sure you understand the data variable descriptions before you start the analysis. Consider all the columns and determine and list which of these predictors should not be used.
3. (15pts) Perform EDA to gain intuition of the factors that affect delay and provide visuals: do delays vary across airlines, or time of departure, or airport (do, at the very least, Chicago (ORD), Boston (BOS), and your favorite another airport), or airport traffic?
4. (20pts) Build a classification model that classifies delays according to `DELAY_OR_NOT`. This is an unbalanced dataset, thus consider the appropriate performance metric when reporting your results.
5. (5pts) Given your model, comment on the importance of factors as related to whether a flight is delayed.
6. (5pts) Evaluate your model(s) on your test set, and finally provide a visual to show which airlines are predicted to have the most delays using all the data excluding the training and test set.
7. (15pts) Build a regression model that predicts the length of delay (on the log scale) given that a flight is truly delayed.
8. (20pts) Write a report (in the last markdown cell in your notebook with your findings (without code)). Describe the main design decisions you have made with justifications. Clearly explain your methodology and results. This should not be more than 300 words. You may use up to 5 diagrams.

Read Data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

import seaborn as sns
%matplotlib inline
```

In [2]:
```
flight_df = pd.read_csv('cs109a_midterm.csv')
print(flight_df.shape)
flight_df.head()
```

```
(804941, 21)
```

Out[2]:

	DATE	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHED_DEP	...
0	2015-09-19	9	19	6	AA	394	N3FMAA	ORD	LGA	07:15:00	...
1	2015-10-28	10	28	3	AA	375	N4YDAA	11298	13342	20:15:00	...
2	2015-08-19	8	19	3	MQ	3648	N512MQ	XNA	ORD	12:22:00	...
3	2015-12-01	12	1	2	WN	4096	N912WN	PHX	BWI	11:20:00	...
4	2015-09-15	9	15	2	WN	285	N7718B	MCI	DEN	14:10:00	...

5 rows × 21 columns

Q1

Create a new variable, DELAY_OR_NOT: a boolean/indicator variable which indicates any arrival delay under 15 mins as a 0, and any delay at or above 15 mins as a 1 (ARRIVAL_DELAY >= 15).

In [3]:
```
flight_df['DELAY_OR_NOT'] = 0
flight_df.loc[flight_df['ARRIVAL_DELAY'] >= 15, 'DELAY_OR_NOT'] = 1
(flight_df['DELAY_OR_NOT'] == 1).sum()
```

Out[3]: 82107

In [4]:
```
# Take 20% of the samples to do training and testing
np.random.seed(6666)
msk = np.random.rand(len(flight_df)) < 0.2
flight_df_sub = flight_df[msk]
flight_df_unuse = flight_df[~msk]

flight_df_sub = flight_df_sub.reset_index(drop=True)
flight_df_unuse = flight_df_unuse.reset_index(drop=True)

len(flight_df_sub)
```

Out[4]: 160889

In [5]:
```
# Split the data for training and testing
np.random.seed(6666)
msk = np.random.rand(len(flight_df_sub)) < 0.5
flight_train = flight_df_sub[msk]
flight_test = flight_df_sub[~msk]

flight_train = flight_train.reset_index(drop=True)
flight_test = flight_test.reset_index(drop=True)

print(len(flight_train))
flight_train.head()
```

80260

Out[5]:

	DATE	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHED_DEP	...
0	2015-09-15	9	15	2	WN	285	N7718B	MCI	DEN	14:10:00	...
1	2015-10-08	10	8	4	UA	1766	N17133	14771	11618	11:45:00	...
2	2015-06-10	6	10	3	US	599	N753US	PHX	SJC	16:40:00	...
3	2015-05-06	5	6	3	MQ	3225	N600MQ	DFW	GRK	15:16:00	...
4	2015-01-31	1	31	6	AA	2227	N020AA	MIA	IAH	16:40:00	...

5 rows × 22 columns

Q2

Make sure you understand the data variable descriptions before you start the analysis. Consider all the columns and determine and list which of these predictors should not be used.

DATE: Doesn't matter - should not be used

MONTH: Could matter as it relates to weather - should use

DAY: Doesn't matter - should not be used

DAY_OF_WEEK: Could matter as more people are flying during weekend

AIRLINE: Could matter as some airlines tend to have higher chance of delay

FLIGHT_NUMBER: Doesn't really matter - should not be used

TAIL_NUMBER: Doesn't matter - should not be used

ORIGIN_AIRPORT: Could matter as some cities have higher chance of delay

DESTINATION_AIRPORT: Could matter as some cities have higher chance of delay

SCHED_DEP: Could matter as in rush time flights are easily delayed

SCHED_ARR: Could matter as in rush time flights are easily delayed

DEPARTURE_DELAY: Certainly matter

ARRIVAL_DELAY: Responses

DISTANCE: Could matter as longer distance flight could compensate the departure delay

SCHEDULED_TIME: Could matter as longer duration flight could compensate the departure delay

ELAPSED_TIME: Redundant information - should not be used

AIR_SYSTEM_DELAY: Part of the responses - should not be used

SECURITY_DELAY: Part of the responses - should not be used

AIRLINE_DELAY: Part of the responses - should not be used

LATE_AIRCRAFT_DELAY: Part of the responses - should not be used

WEATHER_DELAY: Part of the responses - should not be used

In [6]:
```
drop_list = ['DATE', 'DAY', 'FLIGHT_NUMBER', 'TAIL_NUMBER', 'ELAPSED_TIME', 'AIR_SYSTEM_DELAY',
     'SECURITY_DELAY', 'AIRLINE_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY']
```

In [7]:
```
# Drop the list of un-suitable predictors
flight_train_drop = flight_train.drop(drop_list, axis=1)
flight_test_drop = flight_test.drop(drop_list, axis=1)
flight_train_drop.shape
```

Out[7]: (80260, 12)

Q3

Perform EDA to gain intuition of the factors that affect delay and provide visuals: do delays vary across airlines, or time of departure, or airport (do, at the very least, Chicago (ORD), Boston (BOS), and your favorite another airport), or airport traffic?

Effects on Airlines

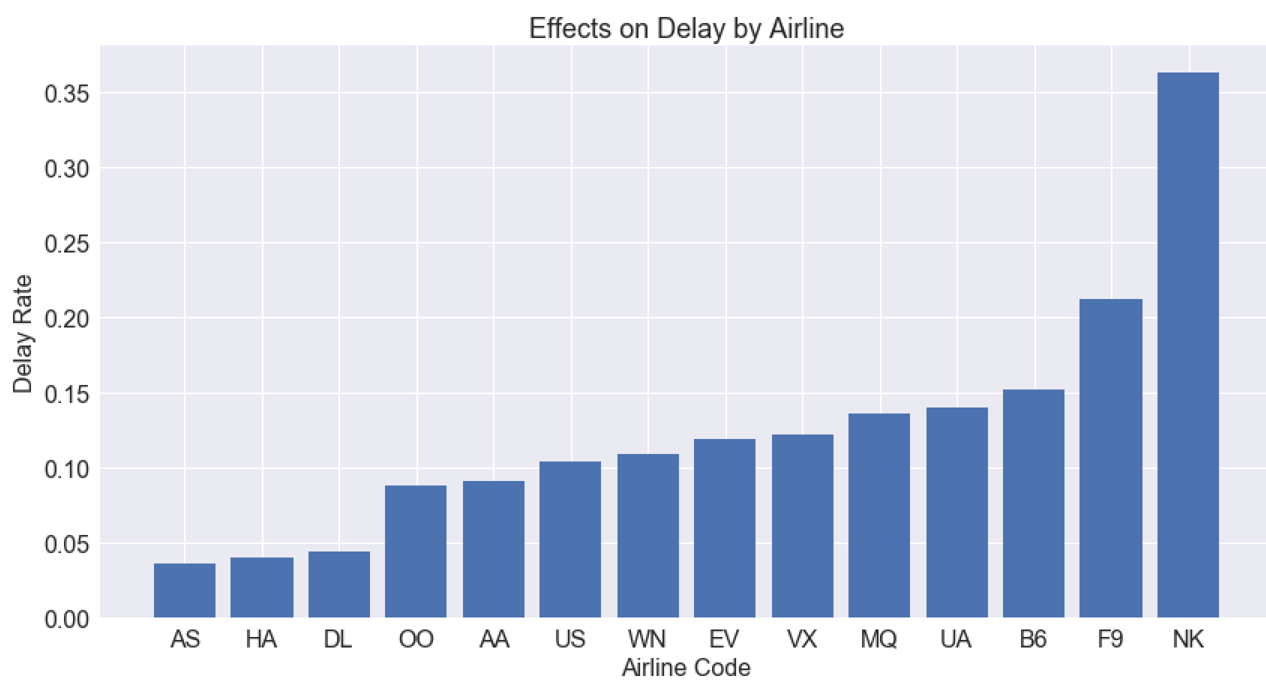
In [8]:
```
group_airline = flight_train[['DELAY_OR_NOT', 'ARRIVAL_DELAY', 'AIRLINE']].groupby(by='AIRLINE')
airline_delay_dict = {}
for name, group in group_airline:
    airline_delay_dict[name] = [group['DELAY_OR_NOT'].mean(), group[group['DELAY_OR_NOT']==1]['ARRIVAL_DELAY'].mean()]
```

In [9]:
```
airline_delay_df = pd.DataFrame.from_dict(airline_delay_dict, orient='index')
airline_delay_df.columns = ['DELAY_RATE', 'AVERAGE_DELAY']
airline_delay_df = airline_delay_df.sort_values(by='DELAY_RATE')
airline_delay_df
```

Out[9]:

	DELAY_RATE	AVERAGE_DELAY
AS	0.036131	46.927083
HA	0.039898	35.361702
DL	0.044088	56.235897
OO	0.087422	57.956399
AA	0.090385	59.325405
US	0.103279	51.029197
WN	0.108718	51.462394
EV	0.118604	60.109253
VX	0.121834	59.772277
MQ	0.135688	57.917808
UA	0.139705	63.400628
B6	0.151825	63.028846
F9	0.211397	65.234783
NK	0.362261	62.635165

In [10]:
```
plt.figure(figsize=(14, 7))
plt.bar(np.arange(len(airline_delay_df)), airline_delay_df['DELAY_RATE'])
plt.xticks(np.arange(len(airline_delay_df)), airline_delay_df.index, fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Airline Code', fontsize=16)
plt.ylabel('Delay Rate', fontsize=16)
plt.title('Effects on Delay by Airline', fontsize=18)
plt.show()
```



Some airlines do tend to have higher chance of delay.

Time of Departure

In [11]:
```
flight_t_dept_df = flight_train[['DELAY_OR_NOT', 'ARRIVAL_DELAY', 'SCHED_DEP']].copy()
flight_t_dept_df['HOUR_DEP'] = flight_t_dept_df['SCHED_DEP'].apply(lambda t: int(t.split(':')[0]))
flight_t_dept_df.head()
```

Out[11]:

	DELAY_OR_NOT	ARRIVAL_DELAY	SCHED_DEP	HOUR_DEP
0	0	-12.0	14:10:00	14
1	0	-17.0	11:45:00	11
2	0	0.0	16:40:00	16
3	0	-12.0	15:16:00	15
4	0	-11.0	16:40:00	16

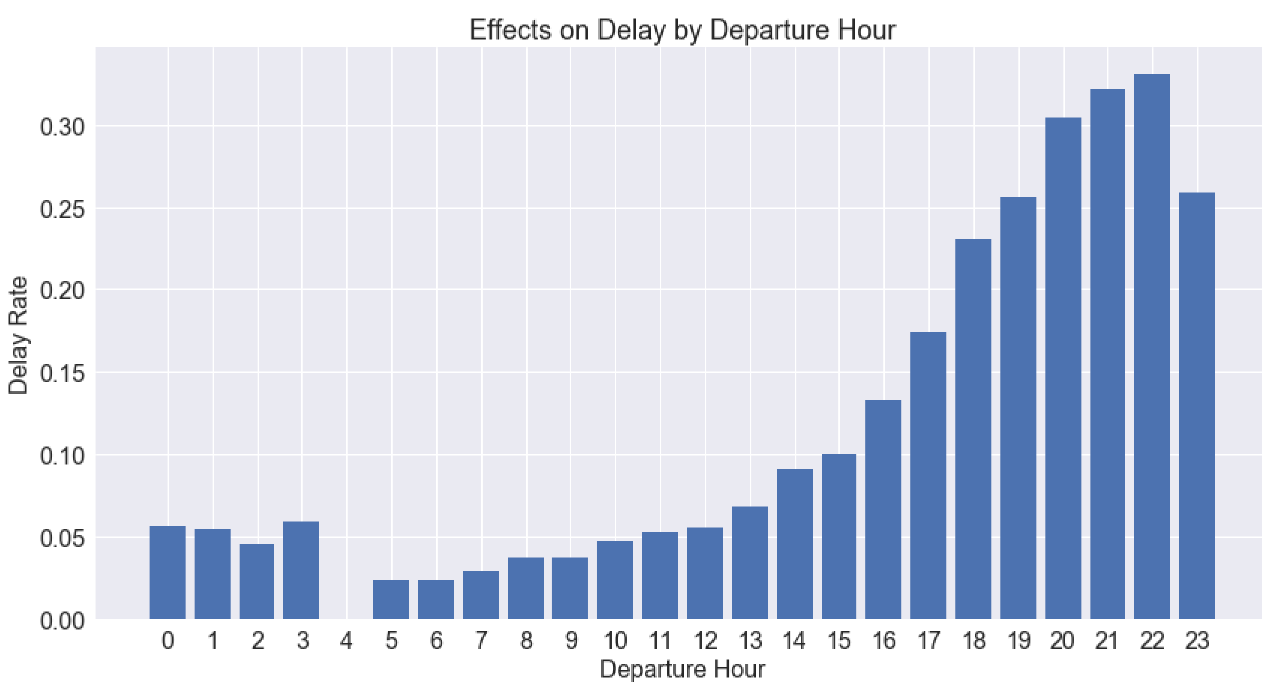
In [12]:
```
group_hour_dep = flight_t_dept_df.groupby(by='HOUR_DEP')
hour_dep_delay_dict = {}
for hour, group in group_hour_dep:
    hour_dep_delay_dict[hour] = [group['DELAY_OR_NOT'].mean(), group[group['DELAY_OR_NOT']==1]['ARRIVAL_DELAY'].mean()
```

In [13]:
```
hour_dep_delay_df = pd.DataFrame.from_dict(hour_dep_delay_dict, orient='index')
hour_dep_delay_df.columns = ['DELAY_RATE', 'AVERAGE_DELAY']
hour_dep_delay_df.head()
```

Out[13]:

	DELAY_RATE	AVERAGE_DELAY
0	0.056277	29.692308
1	0.054795	60.750000
2	0.045455	20.000000
3	0.058824	15.000000
4	0.000000	NaN

In [14]:
```
plt.figure(figsize=(14, 7))
plt.bar(np.arange(len(hour_dep_delay_df)), hour_dep_delay_df['DELAY_RATE'])
plt.xticks(np.arange(len(hour_dep_delay_df)), hour_dep_delay_df.index, fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Departure Hour', fontsize=16)
plt.ylabel('Delay Rate', fontsize=16)
plt.title('Effects on Delay by Departure Hour', fontsize=18)
plt.show()
```



Time of Departure (Hour in the day here) does affect the chance of delay.

Airports

In [15]:
```
flight_airport_df = flight_train[['DELAY_OR_NOT', 'ARRIVAL_DELAY', 'ORIGIN_AIRPORT']]
flight_airport_df.head()
```

Out[15]:

	DELAY_OR_NOT	ARRIVAL_DELAY	ORIGIN_AIRPORT
0	0	-12.0	MCI
1	0	-17.0	14771
2	0	0.0	PHX
3	0	-12.0	DFW
4	0	-11.0	MIA

In [16]:
```
group_airport = flight_airport_df.groupby(by='ORIGIN_AIRPORT')
airport_delay_dict = {}
for airport, group in group_airport:
    airport_delay_dict[airport] = [group['DELAY_OR_NOT'].mean(), group[group['DELAY_OR_NOT']==1]['ARRIVAL_DELAY'].mean
```

In [17]:
```
airport_delay_df = pd.DataFrame.from_dict(airport_delay_dict, orient='index')
airport_delay_df.columns = ['DELAY_RATE', 'AVERAGE_DELAY']
airport_delay_df.head()
```

Out[17]:

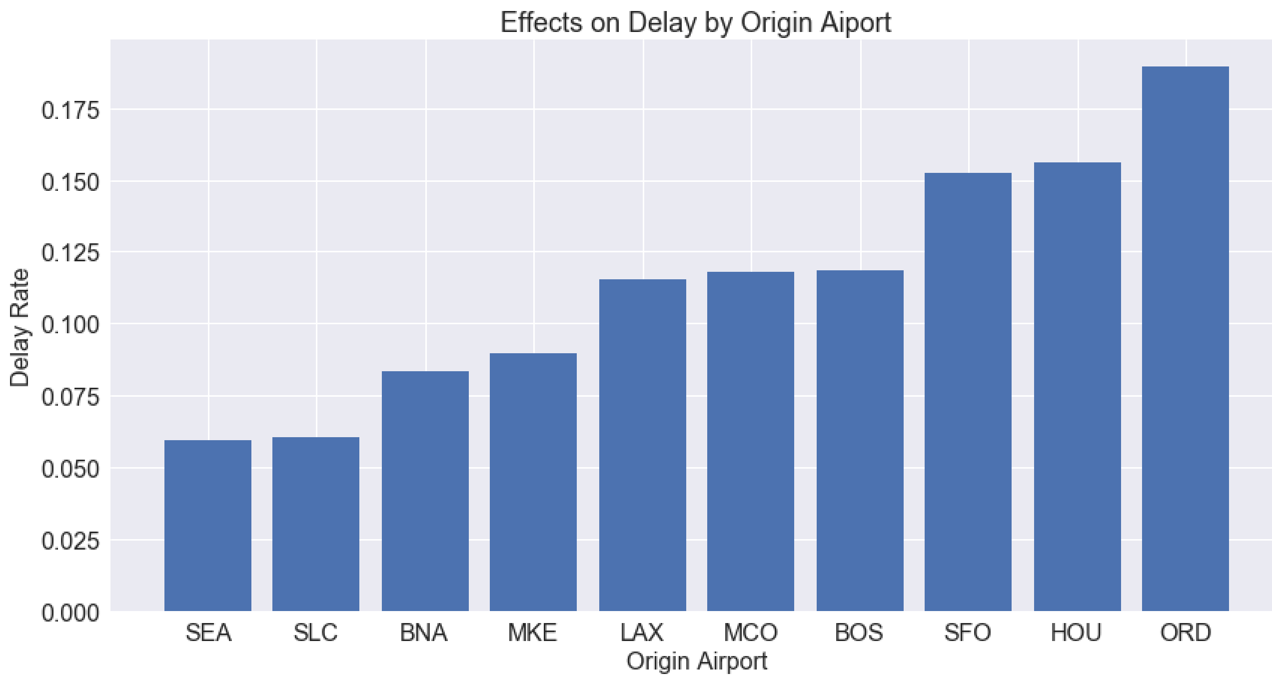
	DELAY_RATE	AVERAGE_DELAY
10135	0.0	NaN
10136	0.5	15.0
10140	0.0	NaN
10141	0.0	NaN
10146	0.0	NaN

In [18]:
```
airport_delay_df_sub = airport_delay_df.loc[['ORD', 'BOS', 'SEA', 'LAX', 'SFO', 'HOU', 'BNA', 'SLC', 'MKE', 'MCO']]
airport_delay_df_sub = airport_delay_df_sub.sort_values(by='DELAY_RATE')
airport_delay_df_sub
```

Out[18]:

	DELAY_RATE	AVERAGE_DELAY
SEA	0.059473	57.371134
SLC	0.060417	43.390805
BNA	0.083207	41.327273
MKE	0.089806	55.270270
LAX	0.115519	51.723906
MCO	0.118115	55.443820
BOS	0.118421	60.304094
SFO	0.152348	53.345865
HOU	0.155899	54.351351
ORD	0.189267	64.492378

In [19]:
```
plt.figure(figsize=(14, 7))
plt.bar(np.arange(len(airport_delay_df_sub)), airport_delay_df_sub['DELAY_RATE'])
plt.xticks(np.arange(len(airport_delay_df_sub)), airport_delay_df_sub.index, fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Origin Airport', fontsize=16)
plt.ylabel('Delay Rate', fontsize=16)
plt.title('Effects on Delay by Origin Aiport', fontsize=18)
plt.show()
```



Different airports do seem to have different chance of delay.

Distance

In [20]:
```
flight_distance_df = flight_train[['DELAY_OR_NOT', 'ARRIVAL_DELAY', 'DISTANCE']].copy()
flight_distance_df['DISTANCE_IN_100'] = flight_distance_df['DISTANCE'].apply(lambda d: int(np.round(d/250)))
flight_distance_df.head()
```

Out[20]:

	DELAY_OR_NOT	ARRIVAL_DELAY	DISTANCE	DISTANCE_IN_100
0	0	-12.0	533	2
1	0	-17.0	2565	10
2	0	0.0	621	2
3	0	-12.0	134	1
4	0	-11.0	964	4

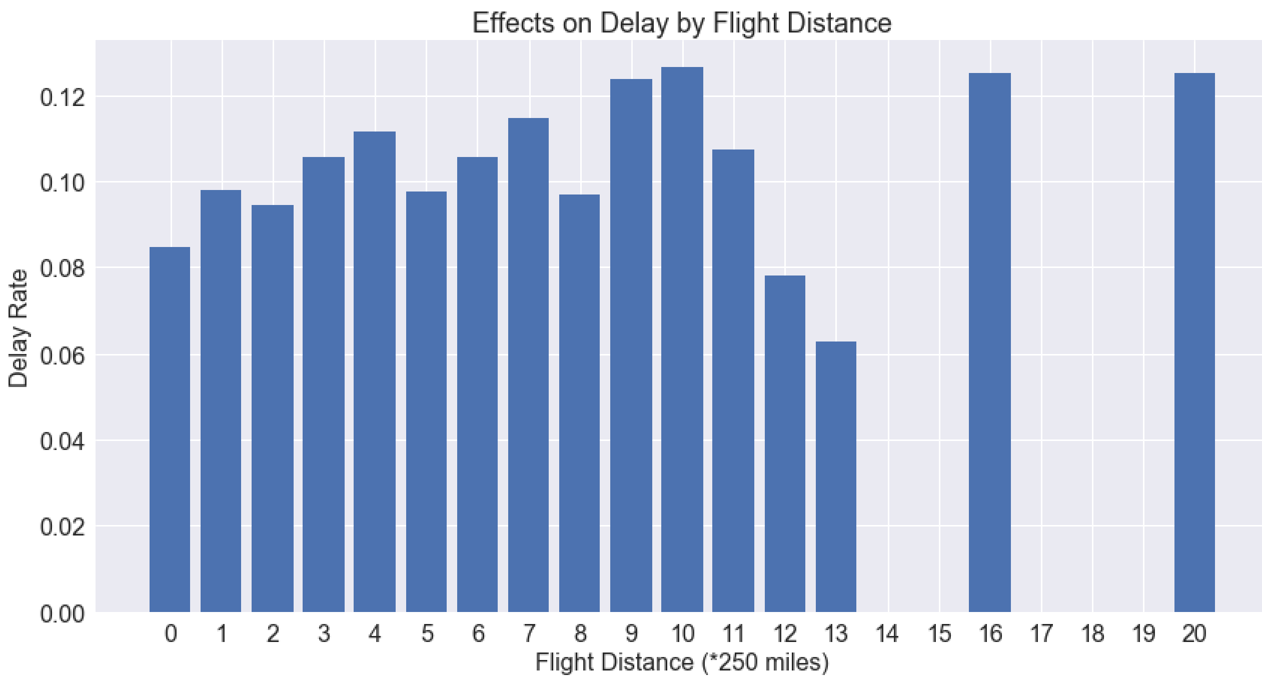
In [21]:
```
group_distance = flight_distance_df.groupby(by='DISTANCE_IN_100')
distance_delay_dict = {}
for distance, group in group_distance:
    distance_delay_dict[distance] = [group['DELAY_OR_NOT'].mean(), group[group['DELAY_OR_NOT']==1]['ARRIVAL_DELAY'].me
```

In [22]:
```
distance_delay_df = pd.DataFrame.from_dict(distance_delay_dict, orient='index')
distance_delay_df.columns = ['DELAY_RATE', 'AVERAGE_DELAY']
distance_delay_df.head()
```

Out[22]:

	DELAY_RATE	AVERAGE_DELAY
0	0.084702	56.800000
1	0.097902	55.492445
2	0.094360	58.487536
3	0.105697	58.912860
4	0.111501	57.716478

In [23]:
```
plt.figure(figsize=(14, 7))
plt.bar(np.arange(len(distance_delay_df)), distance_delay_df['DELAY_RATE'])
plt.xticks(np.arange(len(distance_delay_df)), distance_delay_df.index, fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Flight Distance (*250 miles)', fontsize=16)
plt.ylabel('Delay Rate', fontsize=16)
plt.title('Effects on Delay by Flight Distance', fontsize=18)
plt.show()
```



Distance doesn't seem to have effect on delay rate.

Q4 - Modeling

Build a classification model that classifies delays according to DELAY_OR_NOT. This is an unbalanced dataset, thus consider the appropriate performance metric when reporting your results.

In [24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
import sklearn.metrics as metrics
```

In [25]:
```
# Unbalanced dateset - test all 0 prediction accuracy
(flight_train['DELAY_OR_NOT'] == 0).mean()
```

Out[25]: 0.89788188387739842

In [26]:
```
# Obtain x_train and y_train
x_train_raw = flight_train_drop.drop(['DELAY_OR_NOT', 'ARRIVAL_DELAY'], axis=1).copy()
y_train_class = flight_train_drop['DELAY_OR_NOT']
y_train_delay = flight_train_drop['ARRIVAL_DELAY']

x_test_raw = flight_test_drop.drop(['DELAY_OR_NOT', 'ARRIVAL_DELAY'], axis=1).copy()
y_test_class = flight_test_drop['DELAY_OR_NOT']
y_test_delay = flight_test_drop['ARRIVAL_DELAY']
```

In [27]:
```
# Standardize all the data
x_train = pd.get_dummies(x_train_raw, columns=['MONTH', 'DAY_OF_WEEK', 'AIRLINE', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPO
x_train['SCHED_DEP'] = x_train_raw['SCHED_DEP'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array
x_train['SCHED_ARR'] = x_train_raw['SCHED_ARR'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array

x_test = pd.get_dummies(x_test_raw, columns=['MONTH', 'DAY_OF_WEEK', 'AIRLINE', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT
x_test['SCHED_DEP'] = x_test_raw['SCHED_DEP'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array([
x_test['SCHED_ARR'] = x_test_raw['SCHED_ARR'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array([

dep_delay_min = x_train_raw['DEPARTURE_DELAY'].min()
dep_delay_max = (x_train_raw['DEPARTURE_DELAY'] + dep_delay_min).max()
x_train['DEPARTURE_DELAY'] = (x_train['DEPARTURE_DELAY'] - dep_delay_min)/dep_delay_max
x_test['DEPARTURE_DELAY'] = (x_test['DEPARTURE_DELAY'] - dep_delay_min)/dep_delay_max

dist_max = x_train_raw['DISTANCE'].max()
x_train['DISTANCE'] = x_train['DISTANCE']/dist_max
x_test['DISTANCE'] = x_test['DISTANCE']/dist_max

duration_max = x_train_raw['SCHEDULED_TIME'].max()
x_train['SCHEDULED_TIME'] = x_train['SCHEDULED_TIME']/duration_max
x_test['SCHEDULED_TIME'] = x_test['SCHEDULED_TIME']/duration_max
```

In [28]:
```
# Fill the missing columns with 0s
train_col = x_train.columns.tolist()
test_col = x_test.columns.tolist()

for c in list(set(train_col) - set(test_col)):
    x_test[c] = 0

for c in list(set(test_col) - set(train_col)):
    x_train[c] = 0

x_train = x_train.sort_index(axis=1)
x_test = x_test.sort_index(axis=1)

print(x_test.columns.tolist() == x_train.columns.tolist())
```

```
True
```

In [29]:
```
# Build a logistic model for all the predictors
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train_class)
```

Out[29]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [30]:
```
print('Logistic Model Score (Training): %.3f' % logistic_model.score(x_train, y_train_class))
print('Logistic Model Score (Testing): %.3f' % logistic_model.score(x_test, y_test_class))
```

```
Logistic Model Score (Training): 0.954
Logistic Model Score (Testing): 0.951
```

In [31]:
```
# Contrat the confusion tables for the logistic model and the all 0 model
cm_log = metrics.confusion_matrix(y_train_class, logistic_model.predict(x_train))
cm_all_0 = metrics.confusion_matrix(y_train_class, np.zeros(len(y_train_class)))
print(cm_log)
print(cm_all_0)
```

```
[[71924   140]
 [ 3568  4628]]
[[72064     0]
 [ 8196     0]]
```

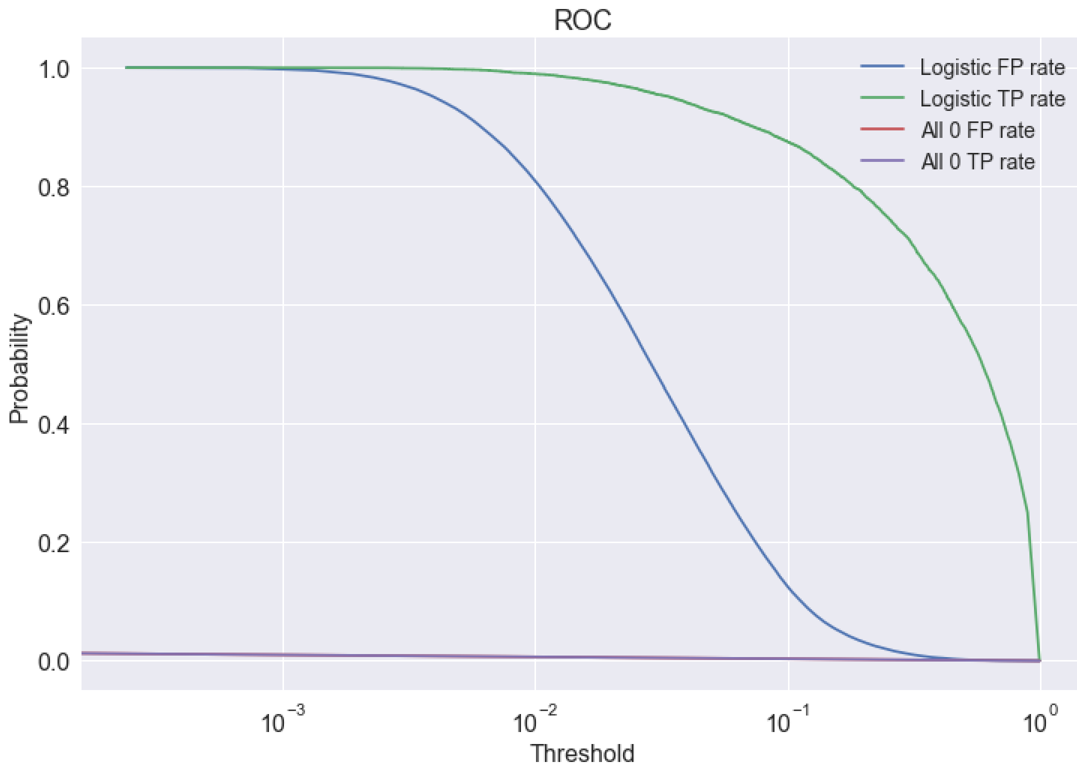
In [32]:
```
TN_logistic = cm_log[0][0]/((y_train_class==0).sum()) * 100
TP_logistic = cm_log[1][1]/((y_train_class==1).sum()) * 100
TN_all_0 = cm_all_0[0][0]/((y_train_class==0).sum()) * 100
TP_all_0 = cm_all_0[1][1]/((y_train_class==1).sum()) * 100

print('Logistic Model True Negative Rate: %.3f%%' % TN_logistic)
print('Logistic Model True Positive Rate: %.3f%%' % TP_logistic)
print('All 0 Model True Negative Rate: %.3f%%' % TN_all_0)
print('All 0 Model True Positive Rate: %.3f%%' % TP_all_0)
```

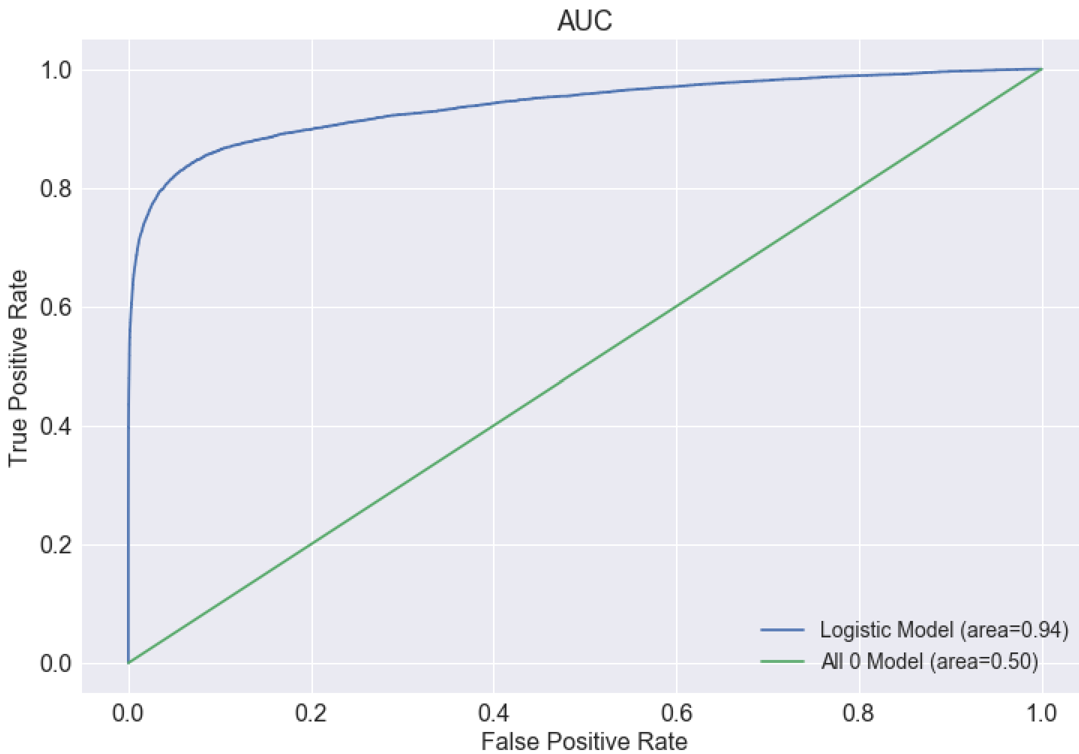
```
Logistic Model True Negative Rate: 99.806%
Logistic Model True Positive Rate: 56.467%
All 0 Model True Negative Rate: 100.000%
All 0 Model True Positive Rate: 0.000%
```

In [33]:
```
# Generate the ROC curve
log_fpr, log_tpr, log_th = metrics.roc_curve(y_train_class, logistic_model.predict_proba(x_train)[:, 1])
all_0_fpr, all_0_tpr, all_0_th = metrics.roc_curve(y_train_class, np.zeros(len(x_train)))
```

In [34]:
```
plt.figure(figsize=(12,8))
plt.plot(log_th, log_fpr, label='Logistic FP rate')
plt.plot(log_th, log_tpr, label='Logistic TP rate')
plt.plot(all_0_th, all_0_fpr, label='All 0 FP rate')
plt.plot(all_0_th, all_0_tpr, label='All 0 TP rate')
plt.xscale('log')
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Threshold', fontsize=16)
plt.ylabel('Probability', fontsize=16)
plt.title('ROC', fontsize=18)
plt.legend(loc='best', fontsize=14)
plt.show()
```



In [35]:
```
plt.figure(figsize=(12,8))
plt.plot(log_fpr, log_tpr, label='Logistic Model (area=%.2f)' % metrics.auc(log_fpr, log_tpr))
plt.plot(all_0_fpr, all_0_tpr, label='All 0 Model (area=%.2f)' % metrics.auc(all_0_fpr, all_0_tpr))
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('AUC', fontsize=18)
plt.legend(loc='best', fontsize=14)
plt.show()
```



In [36]:
```
# Since this dataset is unbalanced - tuning threshold value may contribute to high accuracy
y_train_prob = logistic_model.predict_proba(x_train)[:, 1]

scores_train = []
thresholds_train = np.arange(0, 1, 0.01)
for th in thresholds_train:
    y_train_class_pred = np.zeros(len(y_train_class))
    y_train_class_pred[y_train_prob >= th] = 1
    scores_train.append((y_train_class_pred == y_train_class).mean())

best_score_train = np.max(scores_train)
best_th = thresholds_train[np.argmax(scores_train)]
y_train_class_pred = np.zeros(len(y_train_class))
y_train_class_pred[y_train_prob >= best_th] = 1
```

In [37]:
```
print('Logistic Model Score (Training): %.3f (threshold=%.2f) vs %.3f (threshold=0.5)'
      % (best_score_train, best_th, logistic_model.score(x_train, y_train_class)))
```

```
Logistic Model Score (Training): 0.960 (threshold=0.32) vs 0.954 (threshold=0.5)
```

In [38]:
```
y_test_prob = logistic_model.predict_proba(x_test)[:, 1]
y_test_class_pred = np.zeros(len(y_test_class))
y_test_class_pred[y_test_prob >= best_th] = 1
best_score_test = (y_test_class_pred == y_test_class).mean()
print('Logistic Model Score (Testing): %.3f (threshold=%.2f) vs %.3f (threshold=0.5)'
      % (best_score_test, best_th, logistic_model.score(x_test, y_test_class)))
```

```
Logistic Model Score (Testing): 0.958 (threshold=0.32) vs 0.951 (threshold=0.5)
```

In [39]:
```
cm_log_th = metrics.confusion_matrix(y_train_class, y_train_class_pred)
print(cm_log_th)
```

```
[[71352   712]
 [ 2489  5707]]
```

In [40]:
```
cm_log_th_test = metrics.confusion_matrix(y_test_class, y_test_class_pred)
print(cm_log_th_test)
```

```
[[71649   726]
 [ 2683  5571]]
```

In [41]:
```
TN_logistic_th = cm_log_th[0][0]/((y_train_class==0).sum()) * 100
TP_logistic_th = cm_log_th[1][1]/((y_train_class==1).sum()) * 100

print('Logistic Model (Training) True Negative Rate: %.3f%% (threshold=%.2f) vs %.3f%% (threshold=0.5)'
      % (TN_logistic_th, best_th, TN_logistic))
print('Logistic Model (Training) True Positive Rate: %.3f%% (threshold=%.2f) vs %.3f%% (threshold=0.5)'
      % (TP_logistic_th, best_th, TP_logistic))

TN_logistic_th_test = cm_log_th_test[0][0]/((y_test_class==0).sum()) * 100
TP_logistic_th_test = cm_log_th_test[1][1]/((y_test_class==1).sum()) * 100

print('Logistic Model (Testing) True Negative Rate: %.3f%% (threshold=%.2f)'
      % (TN_logistic_th_test, best_th))
print('Logistic Model (Testing) True Positive Rate: %.3f%% (threshold=%.2f)'
      % (TP_logistic_th_test, best_th))
```

```
Logistic Model (Training) True Negative Rate: 99.012% (threshold=0.32) vs 99.806% (threshold=0.5)
Logistic Model (Training) True Positive Rate: 69.632% (threshold=0.32) vs 56.467% (threshold=0.5)
Logistic Model (Testing) True Negative Rate: 98.997% (threshold=0.32)
Logistic Model (Testing) True Positive Rate: 67.495% (threshold=0.32)
```

Tuning the threshold contribute to slight improvement on the accuracy in both training and testing set. It improve the True Positive Rate by a large margin while lower True Negative rate by a small margin.

Next step is to find the most significant predictors. However, p-values are not given with the logistic model, thus we can find the predictors with highest coefficients first, and then use forward selection to identidfy the significant predictors.

In [42]:
```
# Get the most significant coefficients
coef = logistic_model.coef_[0]
max_coef_ind = np.abs(coef).argsort()[-50:]
x_train_sub = x_train[x_train.columns[max_coef_ind]]
x_test_sub = x_test[x_test.columns[max_coef_ind]]
x_train_sub.head()
```

Out[42]:

	ORIGIN_AIRPORT_10136	DESTINATION_AIRPORT_ILM	ORIGIN_AIRPORT_14698	DESTINATION_AIRPORT_FAT	ORIGIN_AIRPORT_11146	ORIGIN_
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	

5 rows × 50 columns

In [43]:
```
# THIS MAY TAKE A COUPLE MINUTES TO RUN
# Use forward selection to find the best subset of predictor
all_predictors = x_train_sub.columns.tolist()

predictors = [([], 0.5, 0)] # (predictors, threshold, score)

for k in range(1, len(all_predictors)+1):
    best_k_minus_1 = predictors[-1][0]
    # Get the list of remaining predictors
    new_predictors = list(set(all_predictors) - set(best_k_minus_1))
    scores = []
    thresholds = []

    for predictor in new_predictors:
        k_predictors = best_k_minus_1 + [predictor]
        k_x_train = x_train_sub[k_predictors]

        logistic_model_sub = LogisticRegression()
        logistic_model_sub.fit(k_x_train, y_train_class)

        # Find the best threshold
        y_train_prob =logistic_model_sub.predict_proba(k_x_train)[:, 1]

        th_val = np.arange(0, 1, 0.01)
        score_val = []
        for th in th_val:
            y_train_class_pred = np.zeros(len(y_train_class))
            y_train_class_pred[y_train_prob >= th] = 1
            score_val.append((y_train_class_pred == y_train_class).mean())

        scores.append(np.max(score_val))
        thresholds.append(th_val[np.argmax(score_val)])

    besk_k = best_k_minus_1 + [new_predictors[np.argmax(scores)]]
    predictors.append((besk_k, thresholds[np.argmax(scores)], np.max(scores)))
```

In [44]:
```
sort_predictors = sorted(predictors, key=lambda p: (p[2], len(p[0])))
best_predictors = sort_predictors[-1]
best_predictors
```

Out[44]:
```
(['DEPARTURE_DELAY',
  'MONTH_10',
  'DAY_OF_WEEK_7',
  'ORIGIN_AIRPORT_11618',
  'ORIGIN_AIRPORT_ABE',
  'AIRLINE_VX',
  'AIRLINE_AS',
  'ORIGIN_AIRPORT_IAG',
  'DESTINATION_AIRPORT_BQN',
  'DESTINATION_AIRPORT_ILM',
  'ORIGIN_AIRPORT_12278',
  'DESTINATION_AIRPORT_SGU',
  'ORIGIN_AIRPORT_GGG',
  'ORIGIN_AIRPORT_LIH',
  'DESTINATION_AIRPORT_BPT',
  'ORIGIN_AIRPORT_HRL',
  'ORIGIN_AIRPORT_JAC',
  'DESTINATION_AIRPORT_12523',
  'ORIGIN_AIRPORT_10136',
  'ORIGIN_AIRPORT_11146',
  'ORIGIN_AIRPORT_14698',
  'DESTINATION_AIRPORT_11775',
  'ORIGIN_AIRPORT_10408',
  'ORIGIN_AIRPORT_14685',
  'ORIGIN_AIRPORT_AGS',
  'ORIGIN_AIRPORT_BJI',
  'DESTINATION_AIRPORT_FAT',
  'ORIGIN_AIRPORT_14006',
  'DESTINATION_AIRPORT_WYS',
  'ORIGIN_AIRPORT_14893',
  'DESTINATION_AIRPORT_SCE',
  'ORIGIN_AIRPORT_ABR',
  'ORIGIN_AIRPORT_SBA',
  'DESTINATION_AIRPORT_14683'],
 0.23000000000000001,
 0.95867181659606282)
```

In [45]:
```
len(best_predictors[0])
```

Out[45]:
```
34
```

In [46]:
```
best_1_predictor = predictors[1]
best_1_predictor
```

Out[46]:
```
(['DEPARTURE_DELAY'], 0.23000000000000001, 0.95822327435833543)
```

Departure delay is the most significant predictors - itself alone already contribute to ~96% of the accuracy. The rest of the significant predictors are some airlines, airports, months and days, contributing to less than 1% of the accuracy.

In [47]:

```
# Analyze the performance of logistic model using only significant coefficients
predictor_forward = best_predictors[0]
th_forward = best_predictors[1]
score_train_forward = best_predictors[2]

x_train_forward = x_train[predictor_forward]
x_test_forward = x_test[predictor_forward]

logistic_model_forward = LogisticRegression()
logistic_model_forward.fit(x_train_forward, y_train_class)

y_train_prob_forward = logistic_model_forward.predict_proba(x_train_forward)[:, 1]
y_train_class_forward = np.zeros(len(y_train_class))
y_train_class_forward[y_train_prob_forward >= th_forward] = 1
y_test_prob_forward = logistic_model_forward.predict_proba(x_test_forward)[:, 1]
y_test_class_forward = np.zeros(len(y_test_class))
y_test_class_forward[y_test_prob_forward >= th_forward] = 1

score_test_forward = (y_test_class_forward == y_test_class).mean()

print('Logistic Model (Training) Score): %.3f (Forward) vs %.3f (All predictors)'
      % (score_train_forward, best_score_train))
print('Logistic Model (Testing) Score: %.3f (Forward) vs %.3f (All predictors)'
      % (score_test_forward, best_score_test))

cm_log_forward = metrics.confusion_matrix(y_train_class, y_train_class_forward)
print(cm_log_forward)

TN_logistic_forward = cm_log_forward[0][0]/((y_train_class==0).sum()) * 100
TP_logistic_forward = cm_log_forward[1][1]/((y_train_class==1).sum()) * 100

print('Logistic Model (Training) True Negative Rate: %.3f%% (Forward) vs %.3f%% (All predictors)'
      % (TN_logistic_forward, TN_logistic_th))
print('Logistic Model (Training) True Positive Rate: %.3f%% (Forward) vs %.3f%% (All predictors)'
      % (TP_logistic_forward, TP_logistic_th))

cm_log_forward_test = metrics.confusion_matrix(y_test_class, y_test_class_forward)
print(cm_log_forward_test)

TN_logistic_forward_test = cm_log_forward_test[0][0]/((y_test_class==0).sum()) * 100
TP_logistic_forward_test = cm_log_forward_test[1][1]/((y_test_class==1).sum()) * 100

print('Logistic Model (Testing) True Negative Rate: %.3f%% (Forward) vs %.3f%% (All predictors)'
      % (TN_logistic_forward_test, TN_logistic_th_test))
print('Logistic Model (Testing) True Positive Rate: %.3f%% (Forward) vs %.3f%% (All predictors)'
      % (TP_logistic_forward_test, TP_logistic_th_test))
```

```
Logistic Model (Training) Score): 0.959 (Forward) vs 0.960 (All predictors)
Logistic Model (Testing) Score: 0.957 (Forward) vs 0.958 (All predictors)
[[71494   570]
 [ 2747  5449]]
Logistic Model (Training) True Negative Rate: 99.209% (Forward) vs 99.012% (All predictors)
Logistic Model (Training) True Positive Rate: 66.484% (Forward) vs 69.632% (All predictors)
[[71742   633]
 [ 2800  5454]]
Logistic Model (Testing) True Negative Rate: 99.125% (Forward) vs 98.997% (All predictors)
Logistic Model (Testing) True Positive Rate: 66.077% (Forward) vs 67.495% (All predictors)
```

In [48]:
```
# Analyze the performance of Logistic model using only the most significant coefficient
predictor_best_1 = best_1_predictor[0]
th_best_1 = best_1_predictor[1]
score_train_best_1 = best_1_predictor[2]

x_train_best_1 = x_train[predictor_best_1]
x_test_best_1 = x_test[predictor_best_1]

logistic_model_best_1 = LogisticRegression()
logistic_model_best_1.fit(x_train_best_1, y_train_class)

y_train_prob_best_1 = logistic_model_best_1.predict_proba(x_train_best_1)[:, 1]
y_train_class_best_1 = np.zeros(len(y_train_class))
y_train_class_best_1[y_train_prob_best_1 >= th_best_1] = 1
y_test_prob_best_1 = logistic_model_best_1.predict_proba(x_test_best_1)[:, 1]
y_test_class_best_1 = np.zeros(len(y_test_class))
y_test_class_best_1[y_test_prob_best_1 >= th_best_1] = 1

score_test_best_1 = (y_test_class_best_1 == y_test_class).mean()

print('Logistic Model (Training) Score: %.3f (Best 1) vs %.3f (All predictors)'
      % (score_train_best_1, best_score_train))
print('Logistic Model (Training) Score: %.3f (Best 1) vs %.3f (All predictors)'
      % (score_test_best_1, best_score_test))

cm_log_best_1 = metrics.confusion_matrix(y_train_class, y_train_class_best_1)
print(cm_log_best_1)

TN_logistic_best_1 = cm_log_best_1[0][0]/((y_train_class==0).sum()) * 100
TP_logistic_best_1 = cm_log_best_1[1][1]/((y_train_class==1).sum()) * 100

print('Logistic Model (Testing) True Negative Rate: %.3f%% (Best 1) vs %.3f%% (All predictors)'
      % (TN_logistic_best_1, TN_logistic_th_test))
print('Logistic Model (Testing) True Positive Rate: %.3f%% (Best 1) vs %.3f%% (All predictors)'
      % (TP_logistic_best_1, TP_logistic_th_test))
```

```
Logistic Model (Training) Score: 0.958 (Best 1) vs 0.960 (All predictors)
Logistic Model (Training) Score: 0.957 (Best 1) vs 0.958 (All predictors)
[[71511   553]
 [ 2800  5396]]
Logistic Model (Testing) True Negative Rate: 99.233% (Best 1) vs 98.997% (All predictors)
Logistic Model (Testing) True Positive Rate: 65.837% (Best 1) vs 67.495% (All predictors)
```

With the forward selection of predictors, the performance of the model only decreases slightly where the true positive rate drop from ~69% to ~67%. Further more, if only the most significant predictor (departure delay) are selected, the model performance is almost as competitive.

Q5

Given your model, comment on the importance of factors as related to whether a flight is delayed.

As shown in Q4 above, the most significant factor to dertermine whether a flight is delayed is the departure delay. Some airports (e.g. LGA), airlines (e.g. DL), months (e.g. Oct) and days (e.g. Sat and Sun), as well as scheduled departure time also contribute to the flight delay (but with minimal contribution)

Q6

Evaluate your model(s) on your test set, and finally provide a visual to show which airlines are predicted to have the most delays using all the data excluding the training and test set.

As shown in Q4 above, the model performance on the training and testing set are very similar.

The same method for visualizing delay due to airlines used in Q3 can be used here

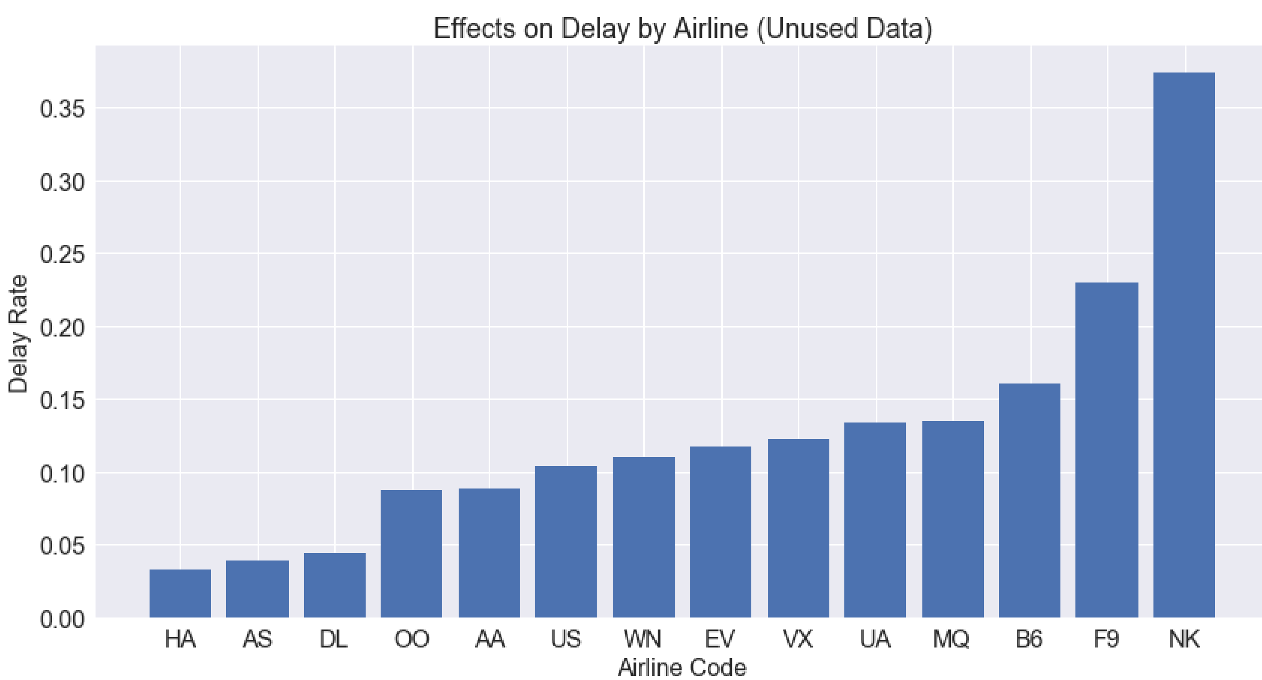
In [49]:
```
group_airline = flight_df_unuse[['DELAY_OR_NOT', 'ARRIVAL_DELAY', 'AIRLINE']].groupby(by='AIRLINE')
airline_delay_dict = {}
for name, group in group_airline:
    airline_delay_dict[name] = [group['DELAY_OR_NOT'].mean(), group[group['DELAY_OR_NOT']==1]['ARRIVAL_DELAY'].mean()]
```

In [50]:
```
airline_delay_df = pd.DataFrame.from_dict(airline_delay_dict, orient='index')
airline_delay_df.columns = ['DELAY_RATE', 'AVERAGE_DELAY']
airline_delay_df = airline_delay_df.sort_values(by='DELAY_RATE')
airline_delay_df
```

Out[50]:

	DELAY_RATE	AVERAGE_DELAY
HA	0.032563	42.059375
AS	0.038945	43.531325
DL	0.043917	56.687934
OO	0.087886	57.296232
AA	0.088889	57.756033
US	0.103841	49.978242
WN	0.109600	53.735837
EV	0.116950	63.311608
VX	0.122053	58.184343
UA	0.134209	62.824785
MQ	0.135283	61.414512
B6	0.160919	62.680774
F9	0.229299	66.736607
NK	0.373443	64.510198

In [51]:
```
plt.figure(figsize=(14, 7))
plt.bar(np.arange(len(airline_delay_df)), airline_delay_df['DELAY_RATE'])
plt.xticks(np.arange(len(airline_delay_df)), airline_delay_df.index, fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Airline Code', fontsize=16)
plt.ylabel('Delay Rate', fontsize=16)
plt.title('Effects on Delay by Airline (Unused Data)', fontsize=18)
plt.show()
```



Same as the figure in Q3, NK, F9 and B6 have the highest chance to delay

In [52]:
```
#Process Unuse dataset
flight_unuse_drop = flight_df_unuse.drop(drop_list, axis=1)
x_unuse_raw = flight_unuse_drop.drop(['DELAY_OR_NOT', 'ARRIVAL_DELAY'], axis=1).copy()
y_unuse_class = flight_unuse_drop['DELAY_OR_NOT']
y_unuse_delay = flight_unuse_drop['ARRIVAL_DELAY']

x_unuse = pd.get_dummies(x_unuse_raw, columns=['MONTH', 'DAY_OF_WEEK', 'AIRLINE', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPO
x_unuse['SCHED_DEP'] = x_unuse['SCHED_DEP'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array([60
x_unuse['SCHED_ARR'] = x_unuse['SCHED_ARR'].apply(lambda t: (np.array([int(i) for i in t.split(':')[:2]])*np.array([60

x_unuse['DEPARTURE_DELAY'] = (x_unuse['DEPARTURE_DELAY'] - dep_delay_min)/dep_delay_max
x_unuse['DISTANCE'] = x_unuse['DISTANCE']/dist_max
x_unuse['SCHEDULED_TIME'] = x_unuse['SCHEDULED_TIME']/duration_max
```

In [53]:
```
# Fill/Drop the missing columns in unused data (not fitted in the model)
train_col = x_train.columns.tolist()
unuse_col = x_unuse.columns.tolist()

for c in list(set(train_col) - set(unuse_col)):
    x_unuse[c] = 0

for c in list(set(unuse_col) - set(train_col)):
    x_unuse = x_unuse.drop(c, axis=1)

x_unuse = x_unuse.sort_index(axis=1)

unuse_col = x_unuse.columns.tolist()
train_col == unuse_col
```

Out[53]: True

In [54]:
```
# Analyze the performance of logistic model on the unused dataset
y_unuse_prob = logistic_model_forward.predict_proba(x_unuse[predictor_forward])[:, 1]
y_unuse_class_pred = np.zeros(len(y_unuse_class))
y_unuse_class_pred[y_unuse_prob >= th_forward] = 1

score_unuse = (y_unuse_class_pred == y_unuse_class).mean()

print('Logistic Model Score: %.3f (Forward, unused data)' % score_unuse)

cm_log_unuse = metrics.confusion_matrix(y_unuse_class, y_unuse_class_pred)
print(cm_log_unuse)

TN_logistic_unuse = cm_log_unuse[0][0]/((y_unuse_class==0).sum()) * 100
TP_logistic_unuse = cm_log_unuse[1][1]/((y_unuse_class==1).sum()) * 100

print('Logistic Model True Negative Rate: %.3f%% (Forward, unused data)' % TN_logistic_unuse)
print('Logistic Model True Positive Rate: %.3f%% (Forward, unused data)' % TP_logistic_unuse)
```

```
Logistic Model Score: 0.958 (Forward, unused data)
[[573713   4682]
 [ 22201  43456]]
Logistic Model True Negative Rate: 99.191% (Forward, unused data)
Logistic Model True Positive Rate: 66.186% (Forward, unused data)
```

In [55]:
```
# Analyze the performance of logistic model (one predictor) on the unused dataset
y_unuse_prob = logistic_model_best_1.predict_proba(x_unuse[predictor_best_1])[:, 1]
y_unuse_class_pred = np.zeros(len(y_unuse_class))
y_unuse_class_pred[y_unuse_prob >= th_best_1] = 1

score_unuse = (y_unuse_class_pred == y_unuse_class).mean()

print('Logistic Model Score: %.3f (Best 1, unused data)' % score_unuse)

cm_log_unuse = metrics.confusion_matrix(y_unuse_class, y_unuse_class_pred)
print(cm_log_unuse)

TN_logistic_unuse = cm_log_unuse[0][0]/((y_unuse_class==0).sum()) * 100
TP_logistic_unuse = cm_log_unuse[1][1]/((y_unuse_class==1).sum()) * 100

print('Logistic Model True Negative Rate: %.3f%% (Best 1, unused data)' % TN_logistic_unuse)
print('Logistic Model True Positive Rate: %.3f%% (Best 1, unused data)' % TP_logistic_unuse)
```

```
Logistic Model Score: 0.958 (Best 1, unused data)
[[573951   4444]
 [ 22410  43247]]
Logistic Model True Negative Rate: 99.232% (Best 1, unused data)
Logistic Model True Positive Rate: 65.868% (Best 1, unused data)
```

Q7

Build a regression model that predicts the length of delay (on the log scale) given that a flight is truly delayed.

In [56]:
```
from sklearn.metrics import r2_score
import statsmodels.api as sm
from statsmodels.api import OLS
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
```

```
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.cor
e.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module inste
ad.
  from pandas.core import datetools
```

In [57]:
```
# Obtain x_train and y_train
x_train_delay = x_train[y_train_class == 1]
y_train_delay = y_train_delay[y_train_class == 1]
x_test_delay = x_test[y_test_class == 1]
y_test_delay = y_test_delay[y_test_class == 1]

x_train_delay = x_train_delay.reset_index(drop=True)
y_train_delay = y_train_delay.reset_index(drop=True)
x_test_delay = x_test_delay.reset_index(drop=True)
y_test_delay = y_test_delay.reset_index(drop=True)

print(x_train_delay.shape)
x_train_delay.head()
```

```
(8196, 1251)
```

Out[57]:

	AIRLINE_AA	AIRLINE_AS	AIRLINE_B6	AIRLINE_DL	AIRLINE_EV	AIRLINE_F9	AIRLINE_HA	AIRLINE_MQ	AIRLINE_NK	AIRLINE_OO	...	OI
0	0	0	0	1	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	1	0	0	0	0	0	0	0	0	0	...	
3	0	0	0	1	0	0	0	0	0	0	...	
4	1	0	0	0	0	0	0	0	0	0	...	

5 rows × 1251 columns

Method 1: Simple OLS model

In [58]:
```
ols_model_all = OLS(y_train_delay, sm.add_constant(x_train_delay)).fit()
ols_model_all.summary()
```

```
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\statsmodels\base\model.py:1036: RuntimeWarning: invalid value
encountered in true_divide
  return self.params / self.bse
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: inv
alid value encountered in greater
  return (self.a < x) & (x < self.b)
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: inv
alid value encountered in less
  return (self.a < x) & (x < self.b)
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\scipy\stats\_distn_infrastructure.py:1818: RuntimeWarning: in
valid value encountered in less_equal
  cond2 = cond0 & (x <= self.a)
```

In [59]:
```
r2_ols_all_train = r2_score(y_train_delay, ols_model_all.predict(sm.add_constant(x_train_delay)))
r2_ols_all_test = r2_score(y_test_delay, ols_model_all.predict(sm.add_constant(x_test_delay)))
print("OLS (all predictors) Training R^2: %.5f and Testing R^2: %.5f" % (r2_ols_all_train, r2_ols_all_test))
```

```
OLS (all predictors) Training R^2: 0.91964 and Testing R^2: 0.90860
```

In [60]:
```
ols_all_sig_predictors = ols_model_all.pvalues[ols_model_all.pvalues < 1e-6]
ols_all_sig_predictors
```

Out[60]:
```
const                          3.800653e-13
AIRLINE_UA                     3.946394e-11
AIRLINE_WN                     6.017465e-12
DEPARTURE_DELAY                0.000000e+00
DESTINATION_AIRPORT_10136      2.559534e-09
DESTINATION_AIRPORT_10146      3.868753e-14
DESTINATION_AIRPORT_10155      5.285887e-11
DESTINATION_AIRPORT_10257      3.726495e-22
DESTINATION_AIRPORT_10268      1.439327e-10
DESTINATION_AIRPORT_10299      2.965871e-07
DESTINATION_AIRPORT_10333      7.581565e-09
DESTINATION_AIRPORT_10372      6.741617e-12
DESTINATION_AIRPORT_10631      5.472703e-12
DESTINATION_AIRPORT_LAX        1.216204e-12
DESTINATION_AIRPORT_ORD        3.792730e-09
DISTANCE                       1.057340e-11
ORIGIN_AIRPORT_BOS             5.533694e-08
ORIGIN_AIRPORT_DCA             2.018327e-07
ORIGIN_AIRPORT_IAD             8.577910e-07
ORIGIN_AIRPORT_JFK             5.989101e-11
ORIGIN_AIRPORT_LGA             3.645406e-11
ORIGIN_AIRPORT_PHL             1.666437e-09
SCHEDULED_TIME                 7.808707e-13
SCHED_DEP                      1.334575e-18
dtype: float64
```

A long list of predictors have significant coefficients that are not 0 using p = 0.05. Change p = 1e-6 to restrict more coefficients.

In [61]:
```
ols_model_sig_predictors = OLS(y_train_delay, sm.add_constant(x_train_delay[ols_all_sig_predictors.index[1:]])).fit()
ols_model_sig_predictors.summary()
```

Out[61]:

OLS Regression Results

Dep. Variable:	ARRIVAL_DELAY	**R-squared:**	0.912
Model:	OLS	**Adj. R-squared:**	0.912
Method:	Least Squares	**F-statistic:**	6054.
Date:	Fri, 03 Nov 2017	**Prob (F-statistic):**	0.00
Time:	22:17:03	**Log-Likelihood:**	-35085.
No. Observations:	8196	**AIC:**	7.020e+04
Df Residuals:	8181	**BIC:**	7.030e+04
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-16.4214	1.119	-14.673	0.000	-18.615	-14.228
AIRLINE_UA	-5.1359	0.651	-7.895	0.000	-6.411	-3.861
AIRLINE_WN	-4.9654	0.487	-10.190	0.000	-5.921	-4.010
DEPARTURE_DELAY	1066.4771	3.682	289.646	0.000	1059.259	1073.695
DESTINATION_AIRPORT_10136	2.615e-15	8.4e-16	3.114	0.002	9.69e-16	4.26e-15
DESTINATION_AIRPORT_10146	-3.435e-13	3.56e-15	-96.407	0.000	-3.51e-13	-3.37e-13
DESTINATION_AIRPORT_10155	-7.919e-14	3.35e-15	-23.621	0.000	-8.58e-14	-7.26e-14
DESTINATION_AIRPORT_10257	-2.375e-13	1.22e-15	-194.408	0.000	-2.4e-13	-2.35e-13
DESTINATION_AIRPORT_10268	-3.2e-14	9.18e-15	-3.484	0.000	-5e-14	-1.4e-14
DESTINATION_AIRPORT_10299	-6.576e-14	2.52e-15	-26.079	0.000	-7.07e-14	-6.08e-14
DESTINATION_AIRPORT_10333	1.108e-13	1.71e-14	6.479	0.000	7.73e-14	1.44e-13
DESTINATION_AIRPORT_10372	2.586e-13	2.61e-14	9.919	0.000	2.07e-13	3.1e-13
DESTINATION_AIRPORT_10631	6.475e-15	1.56e-16	41.635	0.000	6.17e-15	6.78e-15
DESTINATION_AIRPORT_LAX	4.1030	0.867	4.734	0.000	2.404	5.802
DESTINATION_AIRPORT_ORD	4.7977	0.830	5.780	0.000	3.171	6.425
DISTANCE	19.3060	10.057	1.920	0.055	-0.409	39.021
ORIGIN_AIRPORT_BOS	3.1475	1.399	2.249	0.025	0.404	5.890
ORIGIN_AIRPORT_DCA	7.8336	2.098	3.734	0.000	3.721	11.946
ORIGIN_AIRPORT_IAD	8.5355	2.122	4.022	0.000	4.375	12.696
ORIGIN_AIRPORT_JFK	3.7436	1.334	2.806	0.005	1.128	6.359
ORIGIN_AIRPORT_LGA	4.2441	1.218	3.484	0.000	1.857	6.632
ORIGIN_AIRPORT_PHL	7.8951	1.588	4.973	0.000	4.783	11.007
SCHEDULED_TIME	-28.5177	11.178	-2.551	0.011	-50.429	-6.606
SCHED_DEP	-10.4544	1.098	-9.522	0.000	-12.607	-8.302

Omnibus:	2799.232	**Durbin-Watson:**	2.007
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	14563.248
Skew:	1.559	**Prob(JB):**	0.00
Kurtosis:	8.738	**Cond. No.**	1.38e+19

In [62]:
```
r2_ols_sig_predictors_train = r2_score(
    y_train_delay, ols_model_sig_predictors.predict(sm.add_constant(x_train_delay[ols_all_sig_predictors.index[1:]])))
r2_ols_sig_predictors_test = r2_score(
    y_test_delay, ols_model_sig_predictors.predict(sm.add_constant(x_test_delay[ols_all_sig_predictors.index[1:]])))
print("OLS (significant predictors) Training R^2: %.5f and Testing R^2: %.5f"
      % (r2_ols_sig_predictors_train, r2_ols_sig_predictors_test))
```

```
OLS (significant predictors) Training R^2: 0.91197 and Testing R^2: 0.91254
```

The reduction of predictors gives a slightly worse training score but improve on the testing score slightly, which means the overfitting is reduced.

In [63]:
```
# Performance on unused data
x_unuse_delay = x_unuse[y_unuse_class == 1]
y_unuse_delay = y_unuse_delay[y_unuse_class == 1]

r2_ols_sig_predictors_unuse = r2_score(
    y_unuse_delay, ols_model_sig_predictors.predict(sm.add_constant(x_unuse_delay[ols_all_sig_predictors.index[1:]])))
print("OLS (significant predictors) Unused Data R^2: %.5f" % r2_ols_sig_predictors_unuse)
```

```
OLS (significant predictors) Unused Data R^2: 0.91501
```

Method 2: Ridge and Lasso model

In [64]:
```
shrikage = 10.**np.arange(-2, 3)
coef_ind = ols_model_all.params.index.tolist()
coef_dict = dict()
ridge_scores = []
lasso_scores = []
for s in shrikage:

    ridge_cv_model = RidgeCV(alphas=[s])
    ridge_cv_result = ridge_cv_model.fit(x_train_delay, y_train_delay)
    ridge_params = np.hstack((ridge_cv_result.intercept_, ridge_cv_result.coef_))
    ridge_series = pd.Series(data=ridge_params, index=coef_ind)
    ridge_scores.append(r2_score(y_train_delay, ridge_cv_result.predict(x_train_delay)))

    lasso_cv_model = LassoCV(alphas=[s], max_iter=1e5)
    lasso_cv_result = lasso_cv_model.fit(x_train_delay, y_train_delay)
    lasso_params = np.hstack((lasso_cv_result.intercept_, lasso_cv_result.coef_))
    lasso_series = pd.Series(data=lasso_params, index=coef_ind)
    lasso_scores.append(r2_score(y_train_delay, lasso_cv_result.predict(x_train_delay)))

    coef_dict[str(s)] = pd.concat({'Ridge':ridge_series, 'Lasso':lasso_series}, axis=1)
```

In [65]:
```
ridge_best_score = np.max(ridge_scores)
ridge_best_alpha = shrikage[np.argmax(ridge_scores)]

lasso_best_score = np.max(lasso_scores)
lasso_best_alpha = shrikage[np.argmax(lasso_scores)]

print('Ridge Model best score: %.5f (alpha=%f)' % (ridge_best_score, ridge_best_alpha))
print('Lasso Model best score: %.5f (alpha=%f)' % (lasso_best_score, lasso_best_alpha))
```

```
Ridge Model best score: 0.91964 (alpha=0.010000)
Lasso Model best score: 0.91452 (alpha=0.010000)
```

Shirkage for both Ridge and Lasso are small, which means that penality term does not really help with modeling

Method 3: OLS with Forward Selection

In [66]:
```
# Find the significant coefficients with p values < 0.05
sig_predictors = ols_model_all.pvalues[ols_model_all.pvalues < 0.05]
x_train_delay_sub = x_train_delay[sig_predictors.index[1:]]
x_train_delay_sub.shape
```

Out[66]:
```
(8196, 122)
```

In [67]:
```
# THIS MAY TAKE A COUPLE MINUTES TO RUN
# Use forward selection to find the best subset of predictor
all_predictors = x_train_delay_sub.columns.tolist()

predictors = [([], 0)] # (predictors, bic)

for k in range(1, len(all_predictors)+1):
    best_k_minus_1 = predictors[-1][0]
    # Get the list of remaining predictors
    new_predictors = list(set(all_predictors) - set(best_k_minus_1))
    bics = []

    for predictor in new_predictors:
        k_predictors = best_k_minus_1 + [predictor]
        k_x_train_delay = x_train_delay_sub[k_predictors]

        ols_model_k = OLS(y_train_delay, sm.add_constant(k_x_train_delay)).fit()

        bics.append(ols_model_k.bic)

    besk_k = best_k_minus_1 + [new_predictors[np.argmin(bics)]]
    predictors.append((besk_k, np.min(bics)))
```

In [68]:
```
best_predictors_forward = sorted(predictors, key=lambda p: (p[1], len(p[0])))[1]
print("The best predictors (forward selection) set is (%d predictors):\n%s\nwith BIC=%.6f"
      % (len(best_predictors_forward[0]), best_predictors_forward[0], best_predictors_forward[1]))
```

```
The best predictors (forward selection) set is (15 predictors):
['DEPARTURE_DELAY', 'AIRLINE_WN', 'SCHED_DEP', 'AIRLINE_UA', 'DESTINATION_AIRPORT_ORD', 'DESTINATION_AIRPORT_CWA',
 'ORIGIN_AIRPORT_PHL', 'DESTINATION_AIRPORT_LAX', 'AIRLINE_F9', 'ORIGIN_AIRPORT_13422', 'MONTH_9', 'MONTH_12', 'ORIG
IN_AIRPORT_DCA', 'DESTINATION_AIRPORT_IAH', 'ORIGIN_AIRPORT_IAD']
with BIC=70253.404087
```

In [69]:
```
x_train_delay_forward = x_train_delay[best_predictors_forward[0]]
x_test_delay_forward = x_test_delay[best_predictors_forward[0]]
ols_model_forward = OLS(y_train_delay, sm.add_constant(x_train_delay_forward)).fit()
ols_model_forward.summary()
```

Out[69]:

OLS Regression Results

Dep. Variable:	ARRIVAL_DELAY	**R-squared:**	0.913
Model:	OLS	**Adj. R-squared:**	0.912
Method:	Least Squares	**F-statistic:**	5695.
Date:	Fri, 03 Nov 2017	**Prob (F-statistic):**	0.00
Time:	22:20:35	**Log-Likelihood:**	-35055.
No. Observations:	8196	**AIC:**	7.014e+04
Df Residuals:	8180	**BIC:**	7.025e+04
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-18.8218	0.828	-22.733	0.000	-20.445	-17.199
DEPARTURE_DELAY	1064.8738	3.670	290.142	0.000	1057.679	1072.068
AIRLINE_WN	-4.5573	0.478	-9.530	0.000	-5.495	-3.620
SCHED_DEP	-10.2328	1.094	-9.357	0.000	-12.376	-8.089
AIRLINE_UA	-6.0056	0.634	-9.475	0.000	-7.248	-4.763
DESTINATION_AIRPORT_ORD	4.9307	0.821	6.004	0.000	3.321	6.541
DESTINATION_AIRPORT_CWA	63.1058	12.350	5.110	0.000	38.897	87.315
ORIGIN_AIRPORT_PHL	6.4448	1.567	4.112	0.000	3.372	9.517
DESTINATION_AIRPORT_LAX	3.6592	0.851	4.301	0.000	1.991	5.327
AIRLINE_F9	4.4186	1.181	3.740	0.000	2.103	6.734
ORIGIN_AIRPORT_13422	64.9510	17.460	3.720	0.000	30.726	99.176
MONTH_9	-3.8925	1.000	-3.891	0.000	-5.854	-1.931
MONTH_12	-2.4055	0.675	-3.566	0.000	-3.728	-1.083
ORIGIN_AIRPORT_DCA	7.1998	2.083	3.456	0.001	3.116	11.283
DESTINATION_AIRPORT_IAH	4.1313	1.213	3.407	0.001	1.754	6.509
ORIGIN_AIRPORT_IAD	6.9809	2.103	3.320	0.001	2.859	11.103

Omnibus:	2791.585	**Durbin-Watson:**	2.016
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	14960.084
Skew:	1.544	**Prob(JB):**	0.00
Kurtosis:	8.855	**Cond. No.**	114.

In [70]:
```
r2_ols_forward_train = r2_score(
    y_train_delay, ols_model_forward.predict(sm.add_constant(x_train_delay_forward)))
r2_ols_forward_test = r2_score(
    y_test_delay, ols_model_forward.predict(sm.add_constant(x_test_delay_forward)))

print("OLS Training R^2: %.5f (Forward Selection) vs %.5f (Method 1: Significant Predictors)"
      % (r2_ols_forward_train, r2_ols_sig_predictors_train))
print("OLS Testing R^2: %.5f (Forward Selection) vs %.5f (Method 1: Significant Predictors)"
      % (r2_ols_forward_test, r2_ols_sig_predictors_test))
```

```
OLS Training R^2: 0.91261 (Forward Selection) vs 0.91197 (Method 1: Significant Predictors)
OLS Testing R^2: 0.91205 (Forward Selection) vs 0.91254 (Method 1: Significant Predictors)
```

The performance of method 3 (Forward selection) is slightly better than method 1 (significant predictors using p-values)

In [71]:
```
x_unuse_delay_forward = x_unuse_delay[best_predictors_forward[0]]
r2_ols_forward_unuse = r2_score(
    y_unuse_delay, ols_model_forward.predict(sm.add_constant(x_unuse_delay_forward)))
print("OLS Unused Data R^2: %.5f (Forward Selection) vs %.5f (Method 1: Significant Predictors)"
      % (r2_ols_forward_unuse, r2_ols_sig_predictors_unuse))
```

```
OLS Unused Data R^2: 0.91442 (Forward Selection) vs 0.91501 (Method 1: Significant Predictors)
```

209 Additional questions

1. (10pts) Engineer two additional features that will help improve the classification model's performance.
2. (5pts) Add one additional feature from a data source not given to you. Do this only after you complete the rest of the exam.

Deliverable:

A well presented notebook with well structured and documented code to answer questions 1-7 (plus additional questions for 209 students) with brief explanations and/or clarifications (10pts for overall presenation). The last cell should contain the report for question 8.

Hints

1. For the classification model, an AUC of approximately 0.6 should be your base model.
2. $R^2 > 0.03$ for the regression is good, $R^2 > 0.05$ very good, and $R^2 > 0.1$ is impressive (measured on the log scale).

AC209 Part 1

1st additional feature: Holiday

> Should be a categorical feature indicating whether it is a holiday or not (0 for no, 1 for yes)
>
> More passengers will fly during holidays, which could have an impact on delay
>
> This can be directly obtained using the date information

2nd additional feature: Number of same flights in one day

> Should be an integer indicating the number of same flights (same origin, same destination, same date) in one day
>
> More identical flights in one day tend to have higher chance to delay
>
> This information can be obtained by analyzing the date, origin airport and destination airport information

Other possible additional features: Polynomial terms and Interaction terms

In [72]:
```
# THE FOLLOWING CODE ADD QUADRATIC TERMS OF DISTANCE AND SCHEDULED TIME AS THE TWO ADDITIONAL FEATURES
```

In [73]:
```
# Expand the selected column with their polynomials
# X: Dataframe
# col: List of column names
# order: the orders of polynomials

def expand_poly(X, col, order):
    X_temp = X.copy()
    for c in col:
        for o in range(2, order+1):
            X_temp[c+'_'+str(o)] = X_temp[c]**o

    return X_temp
```

In [74]:
```
# Add polynomial terms into training and testing dataset
x_train_poly = expand_poly(x_train, ['SCHED_DEP', 'DEPARTURE_DELAY'], 2)
x_test_poly = expand_poly(x_test, ['SCHED_DEP', 'DEPARTURE_DELAY'], 2)

x_train_poly.head()
```

Out[74]:

	AIRLINE_AA	AIRLINE_AS	AIRLINE_B6	AIRLINE_DL	AIRLINE_EV	AIRLINE_F9	AIRLINE_HA	AIRLINE_MQ	AIRLINE_NK	AIRLINE_OO	...	OF
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	0	0	0	0	0	0	0	0	0	0	...	
3	0	0	0	0	0	0	0	1	0	0	...	
4	1	0	0	0	0	0	0	0	0	0	...	

5 rows × 1253 columns

In [75]:
```
logistic_model_poly = LogisticRegression()
logistic_model_poly.fit(x_train_poly, y_train_class)
```

Out[75]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [76]:
```
y_train_poly = logistic_model_poly.predict(x_train_poly)
y_test_poly = logistic_model_poly.predict(x_test_poly)

print('Logistic Model Score (Training): %.5f (w/ poly) vs %.5f (w/o poly)'
      % (logistic_model_poly.score(x_train_poly, y_train_class), logistic_model.score(x_train, y_train_class)))
print('Logistic Model Score (Testing): %.5f (w/ poly) vs %.5f (w/o poly)'
      % (logistic_model_poly.score(x_test_poly, y_test_class), logistic_model.score(x_test, y_test_class)))
```

```
Logistic Model Score (Training): 0.95379 (w/ poly) vs 0.95380 (w/o poly)
Logistic Model Score (Testing): 0.95138 (w/ poly) vs 0.95131 (w/o poly)
```

With all the predictors included, adding the polynomial terms does not really change the model performance.

In [77]:
```
# Try forward selection to see if the added two terms contribute to the model fitness
coef_poly = logistic_model_poly.coef_[0]
max_coef_poly_ind = np.abs(coef_poly).argsort()[-50:]
max_coef_poly = x_train_poly.columns[max_coef_poly_ind]

print('SCHED_DEP_2' in max_coef_poly)
print('DEPARTURE_DELAY_2' in max_coef_poly)
```

```
True
True
```

In [78]:
```
x_train_poly_sub = x_train_poly[max_coef_poly]
x_test_poly_sub = x_test_poly[max_coef_poly]
x_train_poly_sub.head()
```

Out[78]:

	DESTINATION_AIRPORT_ILM	DESTINATION_AIRPORT_SFO	ORIGIN_AIRPORT_CDV	ORIGIN_AIRPORT_BTR	DESTINATION_AIRPORT_FAT	ORIGI
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	

5 rows × 50 columns

In [79]:
```
# THIS MAY TAKE A COUPLE MINUTES TO RUN
# Use forward selection to find the best subset of predictor
all_predictors = x_train_poly_sub.columns.tolist()

predictors = [([], 0.5, 0)] # (predictors, threshold, score)

for k in range(1, len(all_predictors)+1):
    best_k_minus_1 = predictors[-1][0]
    # Get the list of remaining predictors
    new_predictors = list(set(all_predictors) - set(best_k_minus_1))
    scores = []
    thresholds = []

    for predictor in new_predictors:
        k_predictors = best_k_minus_1 + [predictor]
        k_x_train_poly = x_train_poly_sub[k_predictors]

        logistic_model_poly_sub = LogisticRegression()
        logistic_model_poly_sub.fit(k_x_train_poly, y_train_class)

        # Find the best threshold
        y_train_prob =logistic_model_poly_sub.predict_proba(k_x_train_poly)[:, 1]

        th_val = np.arange(0, 1, 0.01)
        score_val = []
        for th in th_val:
            y_train_class_pred = np.zeros(len(y_train_class))
            y_train_class_pred[y_train_prob >= th] = 1
            score_val.append((y_train_class_pred == y_train_class).mean())

        scores.append(np.max(score_val))
        thresholds.append(th_val[np.argmax(score_val)])

    besk_k = best_k_minus_1 + [new_predictors[np.argmax(scores)]]
    predictors.append((besk_k, thresholds[np.argmax(scores)], np.max(scores)))
```

In [80]:
```
sort_predictors = sorted(predictors, key=lambda p: (p[2], len(p[0])))
best_predictors = sort_predictors[-1]
best_predictors
```

Out[80]:
```
(['DEPARTURE_DELAY',
  'MONTH_10',
  'DAY_OF_WEEK_7',
  'ORIGIN_AIRPORT_11618',
  'ORIGIN_AIRPORT_ABE',
  'AIRLINE_AS',
  'ORIGIN_AIRPORT_IAG',
  'DESTINATION_AIRPORT_BQN',
  'DESTINATION_AIRPORT_ILM',
  'ORIGIN_AIRPORT_12278',
  'DESTINATION_AIRPORT_SGU',
  'ORIGIN_AIRPORT_GGG',
  'ORIGIN_AIRPORT_LIH',
  'DESTINATION_AIRPORT_BPT',
  'ORIGIN_AIRPORT_HRL',
  'ORIGIN_AIRPORT_JAC',
  'DESTINATION_AIRPORT_12523',
  'ORIGIN_AIRPORT_11146',
  'DESTINATION_AIRPORT_11775',
  'DESTINATION_AIRPORT_14683',
  'ORIGIN_AIRPORT_10408',
  'ORIGIN_AIRPORT_14685',
  'ORIGIN_AIRPORT_AGS',
  'ORIGIN_AIRPORT_BJI',
  'DESTINATION_AIRPORT_FAT',
  'ORIGIN_AIRPORT_14006',
  'DESTINATION_AIRPORT_WYS',
  'ORIGIN_AIRPORT_14893',
  'DESTINATION_AIRPORT_SCE',
  'ORIGIN_AIRPORT_ABR',
  'ORIGIN_AIRPORT_CDV',
  'DEPARTURE_DELAY_2',
  'ORIGIN_AIRPORT_SBA'],
 0.23000000000000001,
 0.95864689758285571)
```

In [81]:
```
len(best_predictors[0])
```

Out[81]:
```
33
```

In [82]:
```
# Contrast the performance of the logistic models with and without polynomial terms
predictor_poly = best_predictors[0]
th_poly = best_predictors[1]
score_train_poly = best_predictors[2]

x_train_poly_forward = x_train_poly[predictor_poly]
x_test_poly_forward = x_test_poly[predictor_poly]

logistic_model_poly_forward = LogisticRegression()
logistic_model_poly_forward.fit(x_train_poly_forward, y_train_class)

y_train_prob_poly_forward = logistic_model_poly_forward.predict_proba(x_train_poly_forward)[:, 1]
y_train_class_poly_forward = np.zeros(len(y_train_class))
y_train_class_poly_forward[y_train_prob_poly_forward >= th_poly] = 1
y_test_prob_poly_forward = logistic_model_poly_forward.predict_proba(x_test_poly_forward)[:, 1]
y_test_class_poly_forward = np.zeros(len(y_test_class))
y_test_class_poly_forward[y_test_prob_poly_forward >= th_poly] = 1

score_test_poly = (y_test_class_poly_forward == y_test_class).mean()

print('Logistic Model (Training) Score: %.3f (w/ Poly) vs %.3f (w/o Poly)'
      % (score_train_poly, score_train_forward))
print('Logistic Model (Testing) Score: %.3f (w/ Poly) vs %.3f (w/o Poly)'
      % (score_test_poly, score_test_forward))

cm_log_poly_forward = metrics.confusion_matrix(y_train_class, y_train_class_poly_forward)
print(cm_log_poly_forward)

TN_logistic_poly_forward = cm_log_poly_forward[0][0]/((y_train_class==0).sum()) * 100
TP_logistic_poly_forward = cm_log_poly_forward[1][1]/((y_train_class==1).sum()) * 100

print('Logistic Model (Training) True Negative Rate: %.3f%% (w/ Poly) vs %.3f%% (w/o Poly)'
      % (TN_logistic_poly_forward, TN_logistic_forward))
print('Logistic Model (Training) True Positive Rate: %.3f%% (w/ Poly) vs %.3f%% (w/o Poly)'
      % (TP_logistic_poly_forward, TP_logistic_forward))

cm_log_poly_forward_test = metrics.confusion_matrix(y_test_class, y_test_class_poly_forward)
print(cm_log_poly_forward_test)

TN_logistic_poly_forward_test = cm_log_poly_forward_test[0][0]/((y_test_class==0).sum()) * 100
TP_logistic_poly_forward_test = cm_log_poly_forward_test[1][1]/((y_test_class==1).sum()) * 100

print('Logistic Model (Testing) True Negative Rate: %.3f%% (w/ Poly) vs %.3f%% (w/o Poly)'
      % (TN_logistic_poly_forward_test, TN_logistic_forward_test))
print('Logistic Model (Testing) True Positive Rate: %.3f%% (w/ Poly) vs %.3f%% (w/o Poly)'
      % (TP_logistic_poly_forward_test, TP_logistic_forward_test))
```

```
Logistic Model (Training) Score: 0.959 (w/ Poly) vs 0.959 (w/o Poly)
Logistic Model (Testing) Score: 0.957 (w/ Poly) vs 0.957 (w/o Poly)
[[71496   568]
 [ 2751  5445]]
Logistic Model (Training) True Negative Rate: 99.212% (w/ Poly) vs 99.209% (w/o Poly)
Logistic Model (Training) True Positive Rate: 66.435% (w/ Poly) vs 66.484% (w/o Poly)
[[71744   631]
 [ 2803  5451]]
Logistic Model (Testing) True Negative Rate: 99.128% (w/ Poly) vs 99.125% (w/o Poly)
Logistic Model (Testing) True Positive Rate: 66.041% (w/ Poly) vs 66.077% (w/o Poly)
```

Conclusion: Including polynomial terms of two of the significant predictors (time of departure and deprture delay) does not impact on the model performance

AC209 Part 2

A potential significant additional feature not given here is taxi time on runway (either at origin or at destination). Obviously long taxi time would most likely result in delay

Q8 - Report

Write a report (in the last markdown cell in your notebook with your findings (without code)). Describe the main design decisions you have made with justifications. Clearly explain your methodology and results. This should not be more than 300 words. You may use up to 5 diagrams.

Classification of whether a flight will delay or not:

> Since this is a unbalanced dataset, where ~90% of the flights are "not delayed" (DELAY_OR_NOT=0). The base line for modeling is the all 0 model. Where it gives ~90% accuracy. Thus a new model should generate better accuracy than 90%.
>
> Then we can create a most overfitting logistic model by including all the predictors (>1000) which gives both training and testing accuracy ~95%. In the ROC curve we can argue that, if a different threshold is chosen in the logistic model, we better accuracy (>96%) can be obtained due to the improvement of True Positive Rate.
>
> To deal with the overfitting (lower computational effort), the most significant predictors (with highest coefficients) are selected to further run forward selection, which identify 26 predictors. Modeling with these 26 predictors resulted in a model with only slightly worse performance but with great improvement on computational speed.
>
> During the forward selection, we can also see that one predictor (departure delay) contribute significantly (~96%) to the total accuracy. Modeling with only this predictor gives a model just 1% off than the most overfitting one.

Prediction of delay time given a flight is delayed:

> The thought process is actually similar with the classification. We can start with a linear regression model that is the most overfitting (include all predictors). Such model gives a R^2 score of ~0.922
>
> Then we can select the coefficients that are significant (p-values < 0.05) to run forward selection. Forward selection gives a set of 15 predictors. Linear regression model with these predictors gives a R^2 score of ~0.917, less than 0.5% changes but save a lot of computational effort.
>
> Ride and Lasso modeling did not show improvement on the modeling.

In [ ]: