In [1]:

In [3]: In [4]:

**Problem Statement**

**Reducing fuel consumption is extremely important for aviation industry as fuel constitutes ~ 30% of the operating cost of airlines. Reducing fuel intake can also have a significant positive impact on the environment. Hence, developing cost saving strategies especially on fuel is of prime importance to airlines. Driving fuel efficiency involves developing strategies that touch upon various aspects of airplanes - broadly some of which are highlighted below:**

Aspects related to Aircraft’s actions on the ground - e.g. include reducing taxiing times to reduce engine running times which translate in to reduced fuel intake.

Aspects related to route planning – e.g. taking shorter routes when inflight to destination taking in to consideration any altitude restrictions that exist.

Aspects related to aircraft design – e.g. improving aerodynamics, redesigning aircraft components to conserve fuel or reducing the weight on board like installation of lighter seats.

import pandas as pd

import numpy as np

import glob

import os

import matplotlib.pyplot as plt

%matplotlib inline

from pylab import rcParams

rcParams['figure.figsize'] = 12, 10

import seaborn as sns

sns.set(style="white", color\_codes=True)

from sklearn.feature\_selection import VarianceThreshold

from sklearn.ensemble import ExtraTreesRegressor

from sklearn import metrics

**Some preprocessing steps**

#Method to load all train files; slighlty modified to record flight instan def load\_data(path):

all\_files = glob.glob(path + "/\*.csv")

list = []

for i, file in enumerate(all\_files[:200]):

df = pd.read\_csv(file, index\_col = None, header = 0) df['flight\_instance'] = i

list.append(df)

return pd.concat(list)

path = r"data"

train = load\_data(path)

In [5]: Out[5]:

In [6]:

#Lets check basic statistics on dtata

pd.set\_option("display.max\_columns", 250)

train.describe()

**ACID Year Month Day Hour Minute Se count** 1217028.0 1.217028e+06 1.217028e+06 1.217028e+06 1.217028e+06 1.217028e+06 1.217028 **mean** 676.0 2.003531e+03 6.326908e+00 1.458690e+01 1.236361e+01 2.955334e+01 2.950014

**std** 0.0 5.637601e-01 3.734886e+00 8.426349e+00 4.694716e+00 1.741971e+01 1.731905 **min** 676.0 2.002000e+03 1.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000 **25%** 676.0 2.003000e+03 3.000000e+00 7.000000e+00 8.000000e+00 1.400000e+01 1.400000 **50%** 676.0 2.004000e+03 6.000000e+00 1.300000e+01 1.200000e+01 3.000000e+01 3.000000 **75%** 676.0 2.004000e+03 1.000000e+01 2.200000e+01 1.600000e+01 4.500000e+01 4.500000 **max** 676.0 2.004000e+03 1.200000e+01 3.100000e+01 2.300000e+01 5.900000e+01 5.900000

**Observations**

Few columns have same values throughout (no variance). Would be good idea to throw them away.

None of the columns have NA's

#Re affirming that there are no NA's

train.isnull().sum().sum()

Out[6]: 0

In [7]:

#Lets throw away all the columns with less than 0 variance

def remove\_low\_varcols(df, threshold):

var = VarianceThreshold(threshold=threshold)

var.fit(df)

all\_cols = df.columns.values

low\_var\_cols = all\_cols[~var.get\_support()]

print('Columns with Varianceless than or equal to threshold are: ', lo

final\_cols = all\_cols[var.get\_support()]

df\_new = df.loc[:, final\_cols]

print("New shape ", df\_new.shape)

return df\_new

train = remove\_low\_varcols(train, 0)

Columns with Varianceless than or equal to threshold are: ['ACID' 'FIRE \_2' 'FIRE\_3' 'FIRE\_4' 'FQTY\_3' 'POVT' 'SMOK' 'WAI\_2'

'APUF\_Mean' 'APUF\_Min' 'APUF\_Max' 'TOCW\_Min' 'CALT']

New shape (1217028, 214)

In [8]:

**Visualizations**

Few points to note here are:

We are required to compare fuel flow across various flight phases (There are 7 phases ). We are already aware that different phases have very different fuel flows.

We also need to segregate why few flight instances are different from others in terms of Fuel flow

We have 600 flight instances

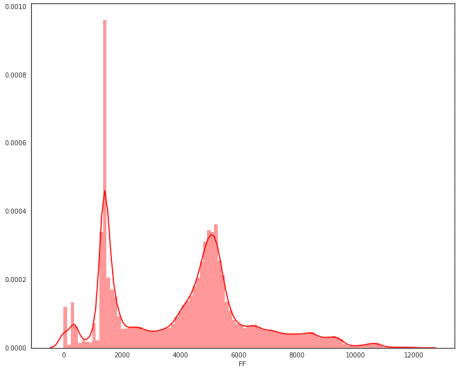
#Lets have a look at target distribution

plt.figure(figsize=(12,10))

sns.distplot(train['FF'], bins=100, color='red')

plt.show()

/home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " 

In [9]:

train.PH.value\_counts()

Out[9]: 5 361086

2 242135

6 229122

4 216804

0 125960

3 27418

1 10447

7 4056

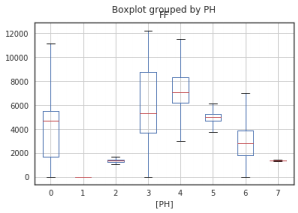
Name: PH, dtype: int64

In [10]:

#Creating Box plot of Fuel Flow across different phases of flight plt.figure()

train[["PH", "FF"]].boxplot( by="PH")

Out[10]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f3fddd6dbe0> <Figure size 432x288 with 0 Axes>



In [11]:

#Lets look at target vaiable for different flight phases g = sns.FacetGrid(train, col="PH", col\_wrap=3, size=4) g = g.map(sns.distplot , 'FF', bins=100)

plt.ylim([0, 0.001])

plt.show()

/home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

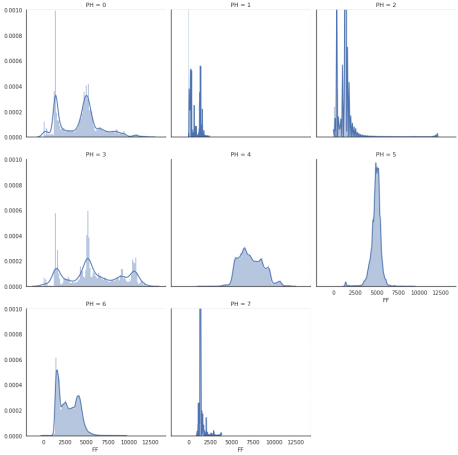
warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been "



Looks like few phases 3, 4,5, 6 have most spread.

Phase 2 seems to have outliers

In [12]: Out[12]:

In [13]:

#Lets look at cruise phase (5) where plane spends most time. train\_ph5 = train.loc[train.PH == 5]

#

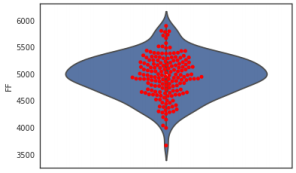
train\_ph5\_agg = train\_ph5.groupby('flight\_instance').agg('mean') train\_ph5\_agg.head()

**Year Month Day Hour Minute Second ABRK ELEV\_1 ELE flight\_instance**

**0** 2004.0 1.0 18.0 10.158825 29.167289 29.451083 119.729707 -2.170143 79.573 **2** 2004.0 8.0 15.0 14.311631 37.683980 29.782736 119.939679 -7.867826 42.633 **3** 2004.0 2.0 28.0 11.000000 12.698593 29.582719 119.782676 -2.659749 71.942 **4** 2003.0 12.0 22.0 13.650624 26.062983 29.805704 119.983559 -2.700639 80.911 **5** 2004.0 4.0 28.0 17.549166 28.220802 29.485978 119.877092 -3.981836 60.074

sns.violinplot(y='FF', data=train\_ph5\_agg)

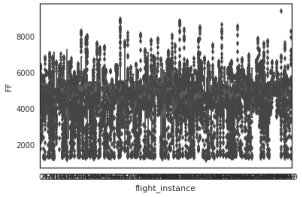
sns.swarmplot(y = 'FF', data= train\_ph5\_agg, color='red') plt.show()



Quite a spread in mean fuel flow @cruise for different instances of flight

In [14]: In [15]:

#How does distributions for different flight instances compare - sns.boxplot(data=train\_ph5, x='flight\_instance', y='FF') #sns.swarmplot(data=train\_ph5, x='flight\_instance', y='FF') plt.show()

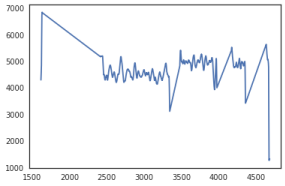


**Looks interesting! Lot of values pretty far from median for all flight instances. Understanding them could be key here.**

Lets pick few random flight instances and look if there is there are trends s w.r.t. duration of flight phase (We know that readings are sorted in time for a given flight instance)

flight\_instance = 4

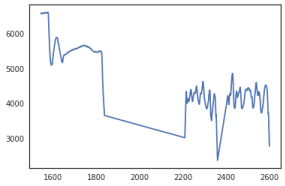
plt.plot(train\_ph5.loc[train\_ph5.flight\_instance == flight\_instance, 'FF'] plt.show()



In [16]: In [17]:

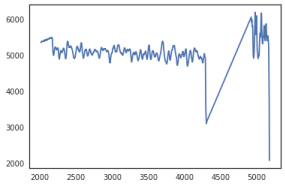
flight\_instance = 23

plt.plot(train\_ph5.loc[train\_ph5.flight\_instance == flight\_instance, 'FF'] plt.show()



flight\_instance = 49

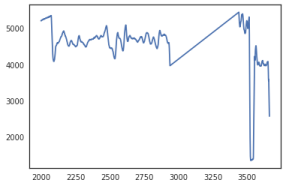
plt.plot(train\_ph5.loc[train\_ph5.flight\_instance == flight\_instance, 'FF'] plt.show()



In [18]: In [19]:

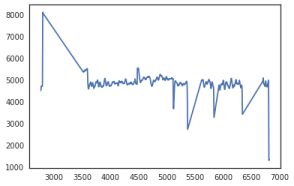
flight\_instance = 61

plt.plot(train\_ph5.loc[train\_ph5.flight\_instance == flight\_instance, 'FF'] plt.show()



flight\_instance = 79

plt.plot(train\_ph5.loc[train\_ph5.flight\_instance == flight\_instance, 'FF'] plt.show()



**Observations:**

There patches of values where there are no readings in phase 5; probably phase changed intermittently (to be checked below)

There are few values for each instance where sudden drop in fuel flow is observed (could be measurement/data processing error?)

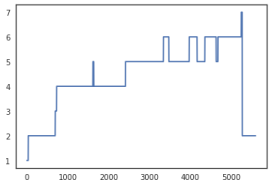
Last value and sometimes beginning values of a given phase are far off (could be due aggregation of values??)

There is mean shift of fuel flow in some instances (could be change of cruise altitude etc.)

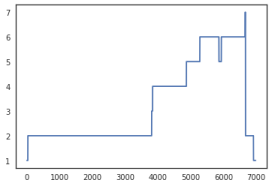
In [20]:

In [21]: In [22]:

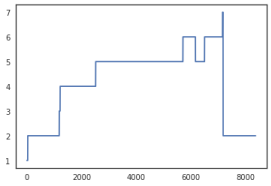
#To confirm faulty allocation of phases, lets plot instance 4. plt.plot(train.loc[train.flight\_instance == 4, 'PH']) plt.show()



plt.plot(train.loc[train.flight\_instance == 16, 'PH']) plt.show()



plt.plot(train.loc[train.flight\_instance == 56, 'PH']) plt.show()

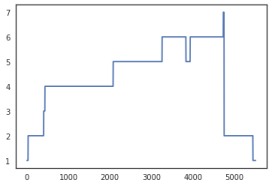


In [23]:

In [24]: In [25]:

plt.plot(train.loc[train.flight\_instance == 72, 'PH'])

plt.show()



**SO, indeed change in phases are not monotonous. Not sure, whether this actual or assignment error. Most likely, assignment error; Plane would not oscillate so mny times between climp and cruise or cruise and approach.**

#At this point lets split the dataset into train and validation sets based from sklearn.model\_selection import train\_test\_split, GroupKFold, cross\_va

#5 fold cv strategy

folder = GroupKFold(n\_splits=5)

cvlist = list(folder.split(train, y=None, groups=train.flight\_instance))

#Use first split as Hold out cv - for quick checking

tr = train.iloc[cvlist[0][0]]

val = train.iloc[cvlist[0][1]]

#Check to ensure we are mixing flight instances between train and validati set(tr.flight\_instance.unique()) & set(val.flight\_instance.unique())

Out[25]: set()

In [26]:

def rmse(y\_true, y\_pred):

return np.sqrt(metrics.mean\_squared\_error(y\_true, y\_pred))

In [27]:

#Lets dump everything in ETR and check which features come out on top etr = ExtraTreesRegressor(max\_depth=7, n\_estimators= 200, n\_jobs=-1, verbo

feats = [f for f in train.columns if f not in ['FF', 'flight\_instance']] etr.fit(tr[feats], tr['FF'])

[Parallel(n\_jobs=-1)]: Done 34 tasks | elapsed: 1.6min [Parallel(n\_jobs=-1)]: Done 184 tasks | elapsed: 7.6min [Parallel(n\_jobs=-1)]: Done 200 out of 200 | elapsed: 8.2min finished

Out[27]: ExtraTreesRegressor(bootstrap=False, criterion='mse', max\_depth=7, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=200, n\_jobs=-1, oob\_score=False, random\_state=None, verbose=1, warm\_start=Fals e)

In [28]:

#Lets see rmse on hold out validation set

print("RMSE on train set :", rmse(tr['FF'], etr.predict(tr[feats]))) print("RMSE on hold out validation set:", rmse(val['FF'], etr.predict(val

[Parallel(n\_jobs=8)]: Done 34 tasks | elapsed: 0.7s [Parallel(n\_jobs=8)]: Done 184 tasks | elapsed: 4.5s [Parallel(n\_jobs=8)]: Done 200 out of 200 | elapsed: 4.9s finished

RMSE on train set : 329.0492207519591

[Parallel(n\_jobs=8)]: Done 34 tasks | elapsed: 0.1s RMSE on hold out validation set: 341.46706972668557

[Parallel(n\_jobs=8)]: Done 184 tasks | elapsed: 0.4s [Parallel(n\_jobs=8)]: Done 200 out of 200 | elapsed: 0.4s finished

In [29]: def plot\_importance(model, feats, n\_feats):

importances = model.feature\_importances\_

std = np.std([tree.feature\_importances\_ for tree in model.estimators\_]

indices = np.argsort(model.feature\_importances\_)[::-1][:n\_feats]

feats = np.array(feats)

top\_feats = feats[indices]

#Print feature ranking

print("Feature Ranking: ")

for i, feat in enumerate(top\_feats):

print("{:d} {:s} ({:f}) ({:f})".format(i+1, feat, importances[indi

plt.figure(figsize=(12,10))

plt.title("Feature importances")

plt.bar(range(len(top\_feats)), importances[indices],

color="r", yerr=std[indices], align="center")

plt.xticks(range(len(top\_feats)), top\_feats, rotation=90)

plt.show()

return \_, top\_feats

In [30]:

\_, top\_feats = plot\_importance(etr, feats, 25)

Feature Ranking:

1 IVV\_Mean (0.084489) (0.148454)

2 CAS\_Min (0.084107) (0.167770)

3 CAS\_Mean (0.076386) (0.158847)

4 IVV\_Min (0.075828) (0.134424)

5 IVV\_Max (0.066964) (0.123356)

6 ALTR\_Min (0.054371) (0.112998)

7 ALTR\_Max (0.050322) (0.108571)

8 ALTR\_Mean (0.041237) (0.099513)

9 MACH\_Mean (0.037555) (0.119481)

10 CAS\_Max (0.036193) (0.115377)

11 GS\_Mean (0.025937) (0.100067)

12 MACH\_Max (0.025539) (0.098464)

13 GS\_Max (0.023136) (0.092056)

14 GS\_Min (0.022971) (0.096472)

15 N1T\_Min (0.019834) (0.041106)

16 N1T\_Max (0.017343) (0.036877)

17 N1T\_Mean (0.016924) (0.038210)

18 PH (0.014879) (0.050823)

19 PI\_Max (0.013783) (0.072954)

20 MACH\_Min (0.013265) (0.073216)

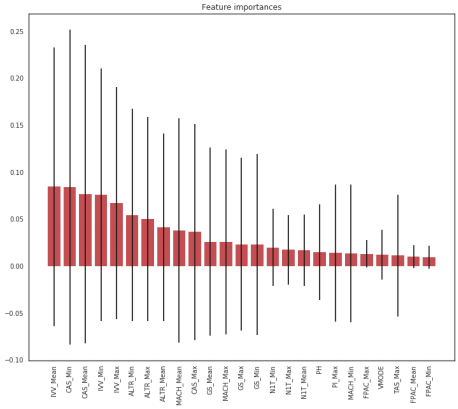
21 FPAC\_Max (0.012998) (0.014419)

22 VMODE (0.012004) (0.026403)

23 TAS\_Max (0.011043) (0.064733)

24 FPAC\_Mean (0.010024) (0.012104)

25 FPAC\_Min (0.009425) (0.012013)

Feature Ranking:

PH (0.107589) (0.167965)

LONG\_Max (0.069545) (0.145660)

IVV\_Mean (0.067866) (0.147514)

VIB\_1\_Mean (0.062372) (0.158833)

VIB\_1\_Max (0.052972) (0.149057)

CAS\_Min (0.051805) (0.132064)

LONG\_Mean (0.047327) (0.146224)

ALTR\_Min (0.038022) (0.105542)

IVV\_Max (0.037572) (0.106540)

VIB\_1\_Min (0.036064) (0.123443)

ALTR\_Mean (0.034586) (0.098909)

CAS\_Max (0.032939) (0.098710)

CAS\_Mean (0.029872) (0.098977)

IVV\_Min (0.025714) (0.068457)

ALTR\_Max (0.025460) (0.084879)

LONG\_Min (0.018638) (0.076449)

MACH\_Mean (0.017350) (0.074949)

MACH\_Min (0.016838) (0.076311)

In [32]:

TAS\_Mean (0.014920) (0.048868)

TAS\_Max (0.014237) (0.056166)

GS\_Mean (0.013355) (0.066076)

PI\_Mean (0.012561) (0.062683)

MACH\_Max (0.009783) (0.057446)

GS\_Max (0.009533) (0.058464)

FPAC\_Mean (0.009038) (0.012956)

As expected we get Phase as one of the important variables.

Lets take a look at other variables

#Scatter plot between LONG\_Max and FF (Fuel Flow)

plt.figure()

sns.jointplot("LONG\_Max" , "FF", data=train)

plt.show()

/home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " <Figure size 432x288 with 0 Axes>



In [33]:

plt.figure()

sns.jointplot("IVV\_Max" , "FF", data=train)

plt.show()

/home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " /home/user/anaconda3/lib/python3.6/site-packages/matplotlib/axes/\_axes.p y:6462: UserWarning: The 'normed' kwarg is deprecated, and has been repl aced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been " <Figure size 432x288 with 0 Axes>



Vibrations are related to acceleration, engine health and Phase. This might be agood one to dig deeper

In [34]:

corr\_feats = list(top\_feats) + ['FF']

corr\_df = train[corr\_feats].corr()

fig, ax = plt.subplots(figsize=(16,16))

sns.heatmap(corr\_df, robust =True, annot=True, ax=ax, annot\_kws={'size':9}

Out[34]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f3fdc715c18> 

Few other features that show up high are CAS(Corrected air speed), Mach, Ground speed, True air speed and Altitude related features. CAS and Mach are corelated. Also, min, max and mean are highly corelated for many features. It might be a good to remove some of the highly corelated ones.

**DUMP everything into XGboost to get us a baseline**

In [ ]: In [ ]:

In [ ]:

#Lets dump everything into xgboost and see what we get.

#Warning: Not recommended to use this as final model. Remember - Garbage I import xgboost

from xgboost.sklearn import XGBRegressor

X\_tr = tr[feats]

y\_tr = tr['FF']

X\_val = val[feats]

y\_val = val['FF']

xgb\_dump = XGBRegressor(max\_depth=6, n\_estimators=1000, colsample\_bytree=0 xgb\_dump.fit(X\_tr, y\_tr, eval\_set=[(X\_tr, y\_tr), (X\_val, y\_val)], eval\_met

#Feature importances from xgboost

from xgboost import plot\_importance

fig, ax = plt.subplots(figsize=(12,30))

plot\_importance(xgb\_dump, ax=ax)

**Validation RMSE - 200 (After dumping everything to xgboost)** We are overfitting by a lot here. Need to very careful about overfitting.

Some directions:

PCA would be a good idea given so many corelated features and few with very little varince. Features related to groupings by phases would be my first choice

Remove corelated features

Features charaterizing flight instance