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.

In [1]:
```
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

In [3]:
```
#Method to load all train files; slighlty modified to record flight insta
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)
```

In [4]:
```
path = r"data"
train = load_data(path)
```

In [5]:
```
#Lets check basic statistics on dtata
pd.set_option("display.max_columns", 250)
train.describe()
```

Out[5]:

	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

In [6]:
```
#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: ', l

    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)
```

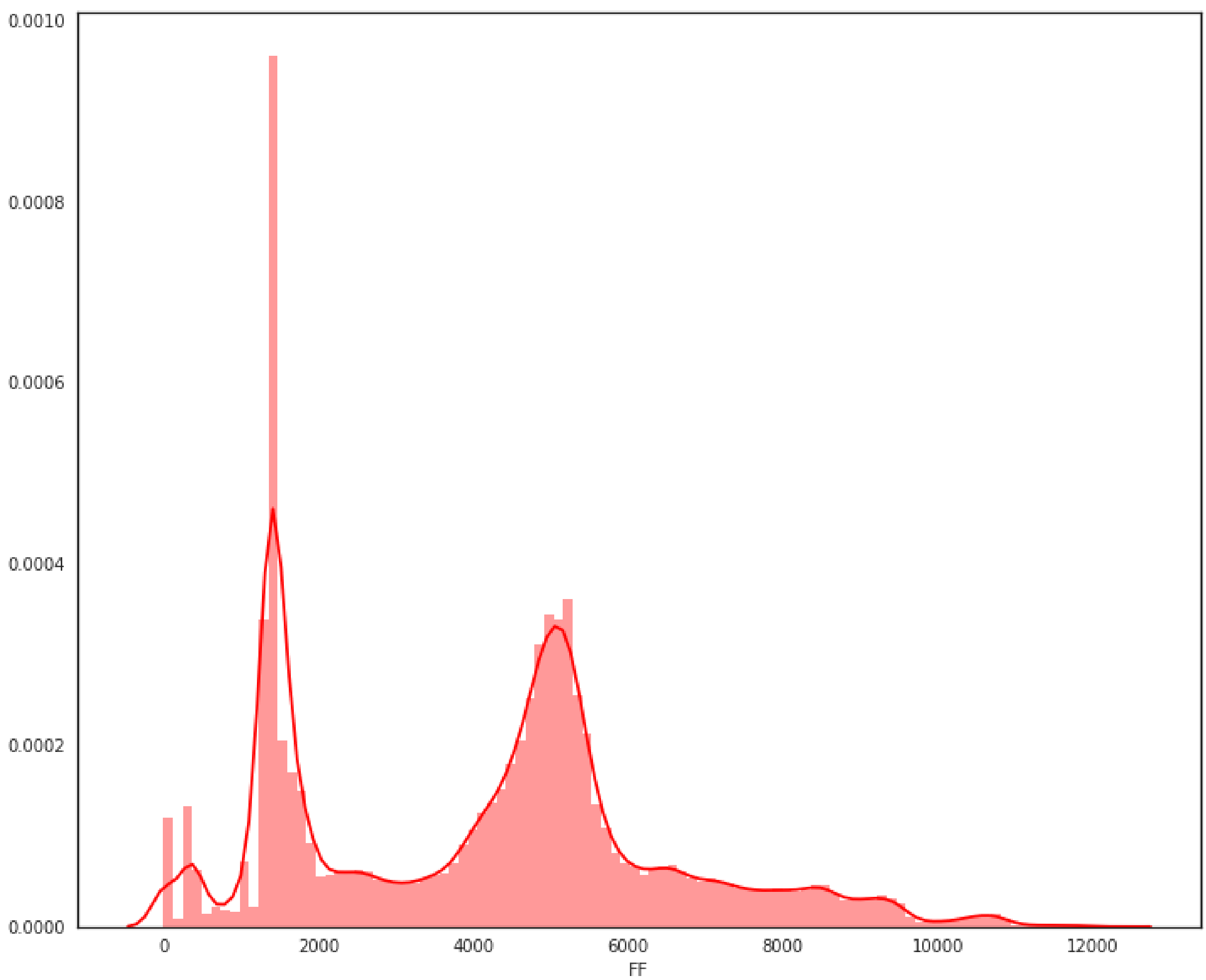
Visualizations

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

In [8]:
```
#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 "
```

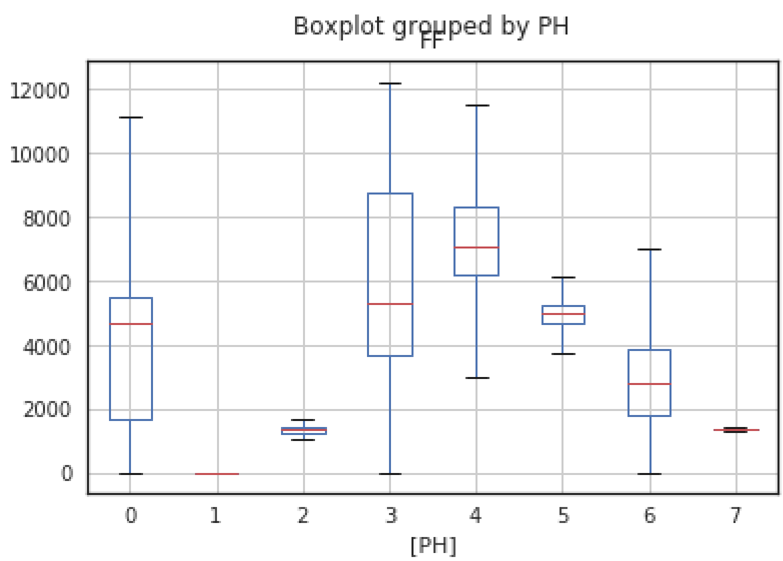


In [9]:
```
train.PH.value_counts()
```

Out[9]:
```
5    361086
2    242135
6    229122
4    216804
0    125960
3     27418
1     10447
7      4056
Name: PH, dtype: int64
```

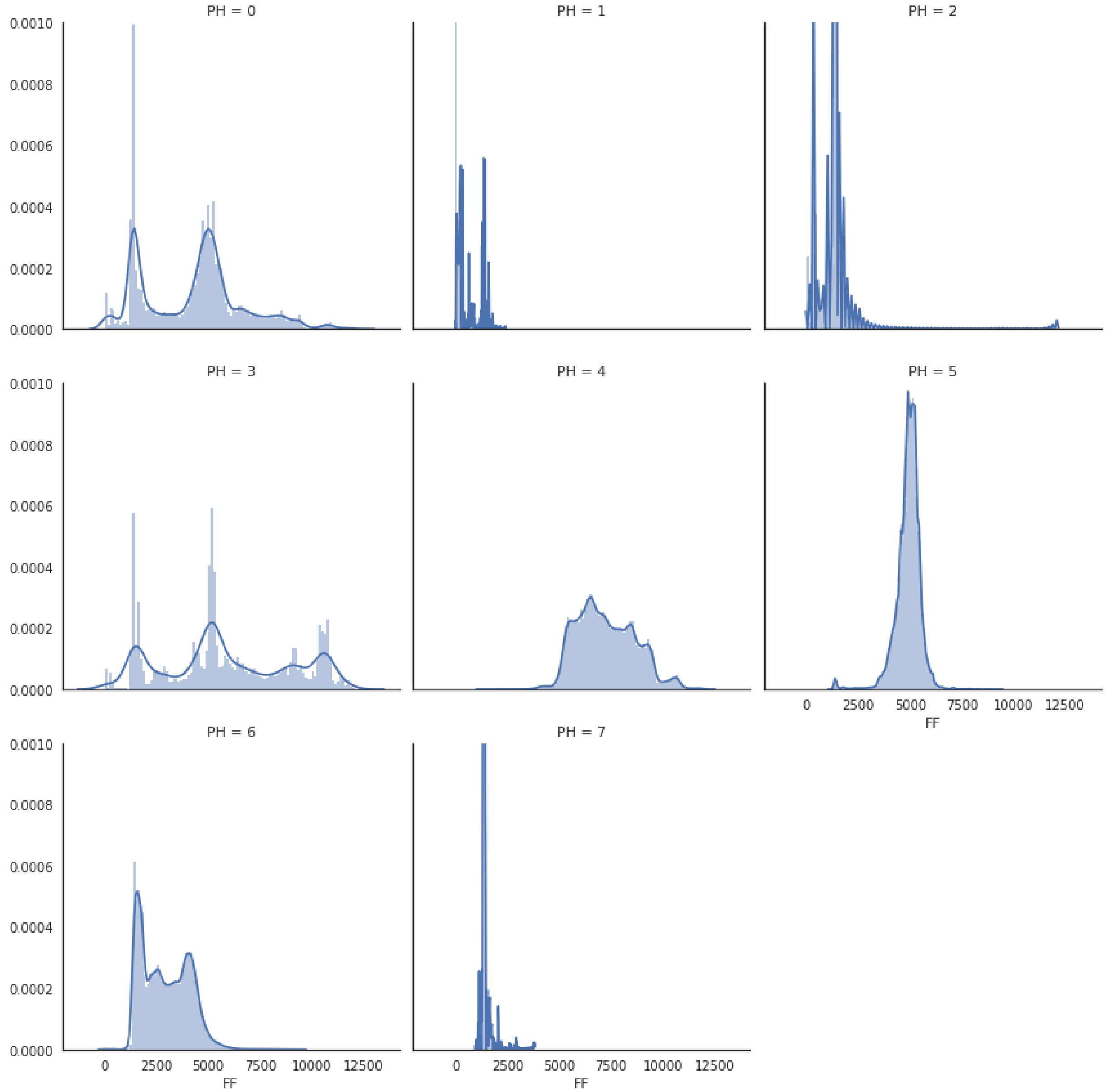
In [10]:
```
#Creating Box plot of Fuel Flow across different phases of flight
plt.figure()
train[["PH", "FF"]].boxplot( by="PH")
```

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.

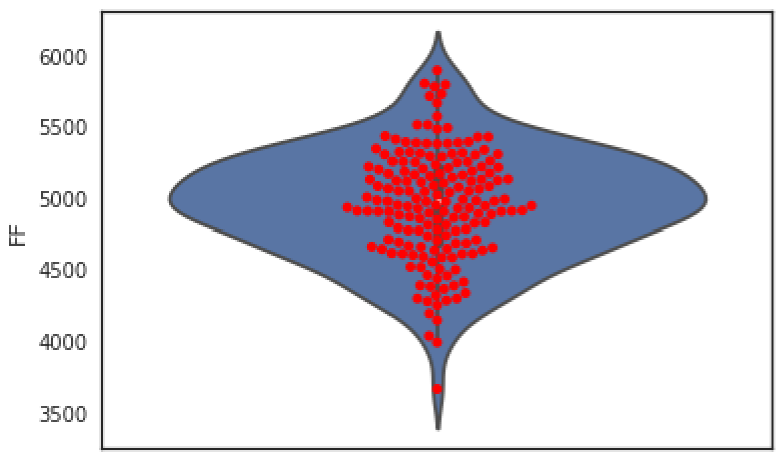
Phase 2 seems to have outliers

In [12]:
```python
#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()
```

Out[12]:

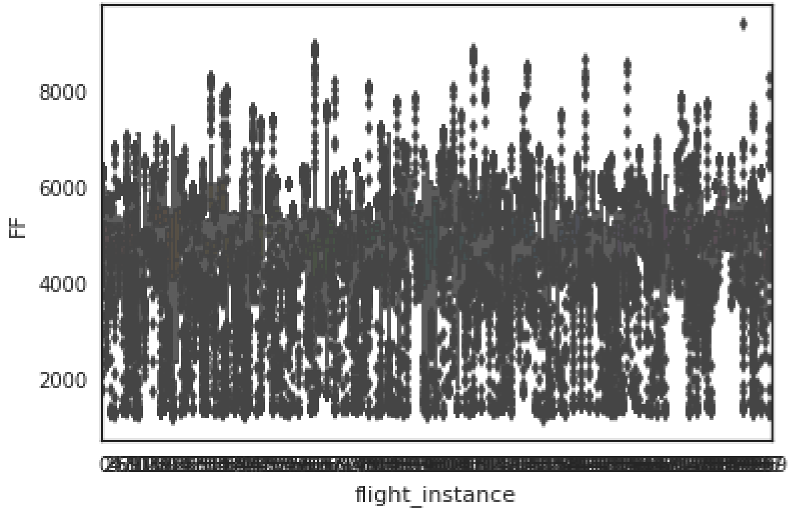
	Year	Month	Day	Hour	Minute	Second	ABRK	ELEV_1	ELE
flight_instance									
0	2004.0	1.0	18.0	10.158825	29.167289	29.451083	119.729707	-2.170143	79.573
2	2004.0	8.0	15.0	14.311631	37.683980	29.782736	119.939679	-7.867826	42.633
3	2004.0	2.0	28.0	11.000000	12.698593	29.582719	119.782676	-2.659749	71.942
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

In [13]:
```python
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

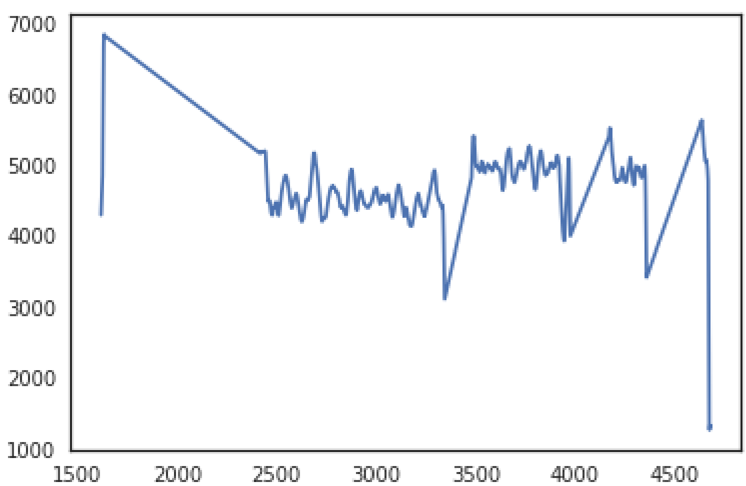
In [14]:
```
#How does distributions for different flight instances compare -
sns.boxplot(data=train_ph5, x='flight_instance', y='FF')
#sns.swarmplot(data=train_ph5, x='flight_instance', y='FF')
plt.show()
```



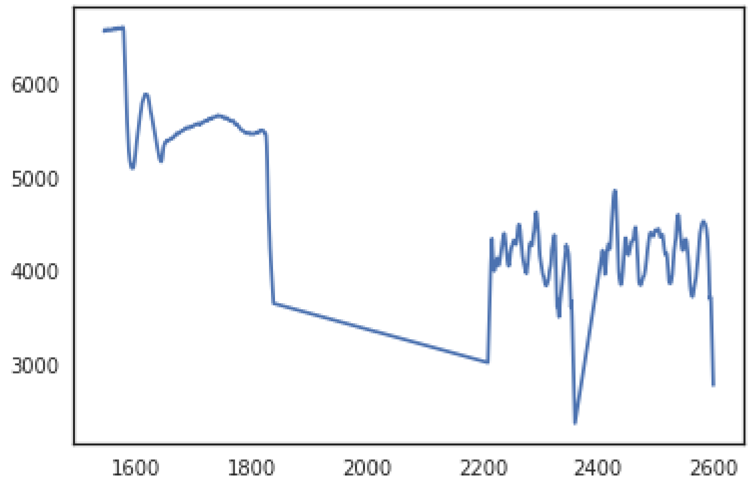
Looks interesting! Lot of values pretty far from median for all flight instances. Understanding them could be key here.

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)

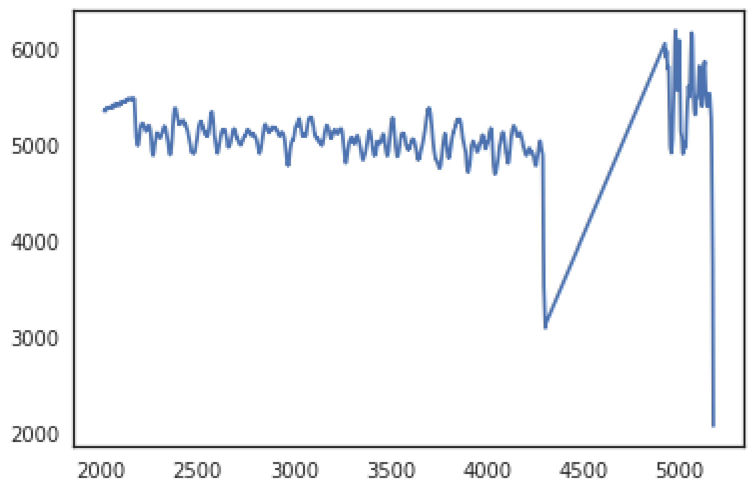
In [15]:
```
flight_instance = 4
plt.plot(train_ph5.loc[train_ph5.flight_instance == flight_instance, 'FF'
plt.show()
```



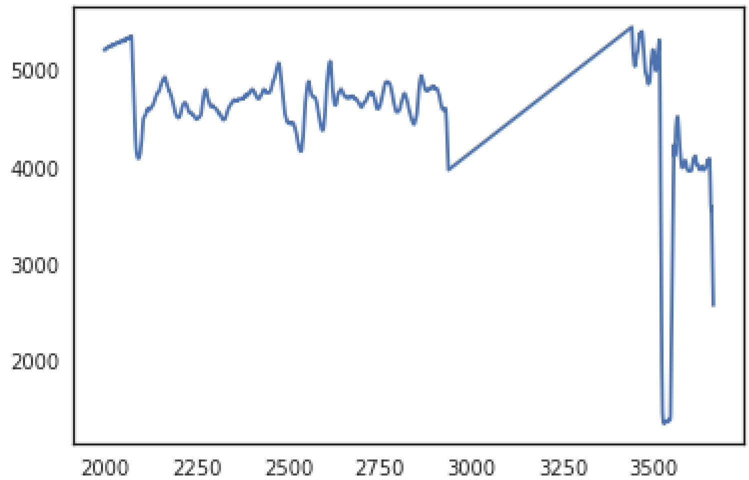
In [16]:
```
flight_instance = 23
plt.plot(train_ph5.loc[train_ph5.flight_instance == flight_instance, 'FF']
plt.show()
```



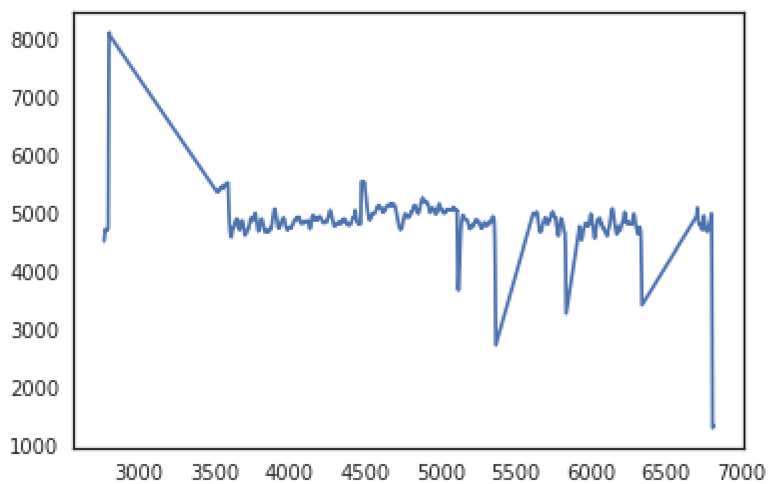
In [17]:
```
flight_instance = 49
plt.plot(train_ph5.loc[train_ph5.flight_instance == flight_instance, 'FF']
plt.show()
```



In [18]:
```
flight_instance = 61
plt.plot(train_ph5.loc[train_ph5.flight_instance == flight_instance, 'FF']
plt.show()
```



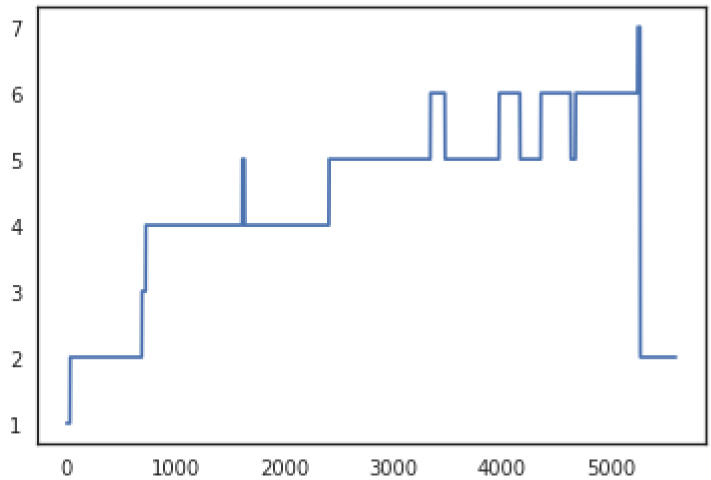
In [19]:
```
flight_instance = 79
plt.plot(train_ph5.loc[train_ph5.flight_instance == flight_instance, 'FF']
plt.show()
```



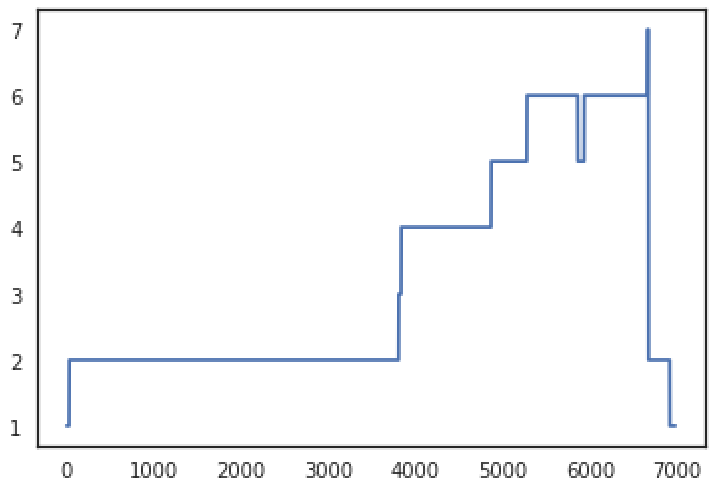
Observations:

- 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.)

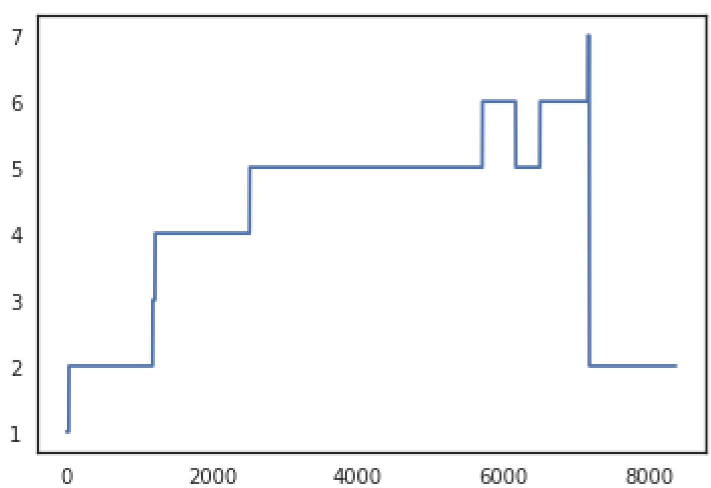
In [20]:
```
#To confirm faulty allocation of phases, lets plot instance 4.
plt.plot(train.loc[train.flight_instance == 4, 'PH'])
plt.show()
```



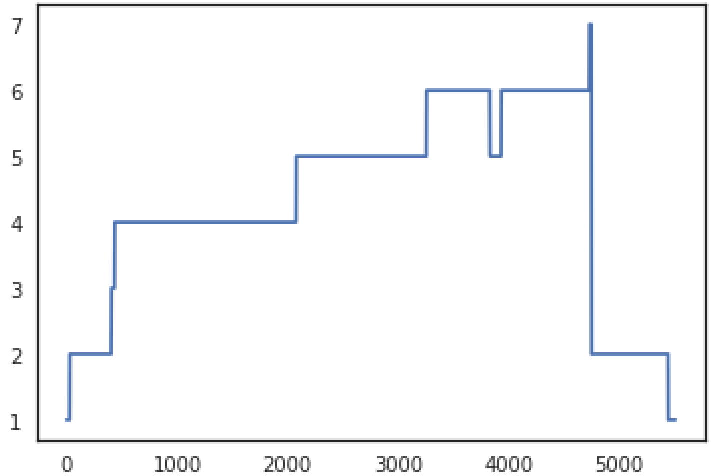
In [21]:
```
plt.plot(train.loc[train.flight_instance == 16, 'PH'])
plt.show()
```



In [22]:
```
plt.plot(train.loc[train.flight_instance == 56, 'PH'])
plt.show()
```



In [23]:
```
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.

In [24]:
```
#At this point lets split the dataset into train and validation sets base
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]]
```

In [25]:
```
#Check to ensure we are mixing flight instances between train and validat
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, verb

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)
```

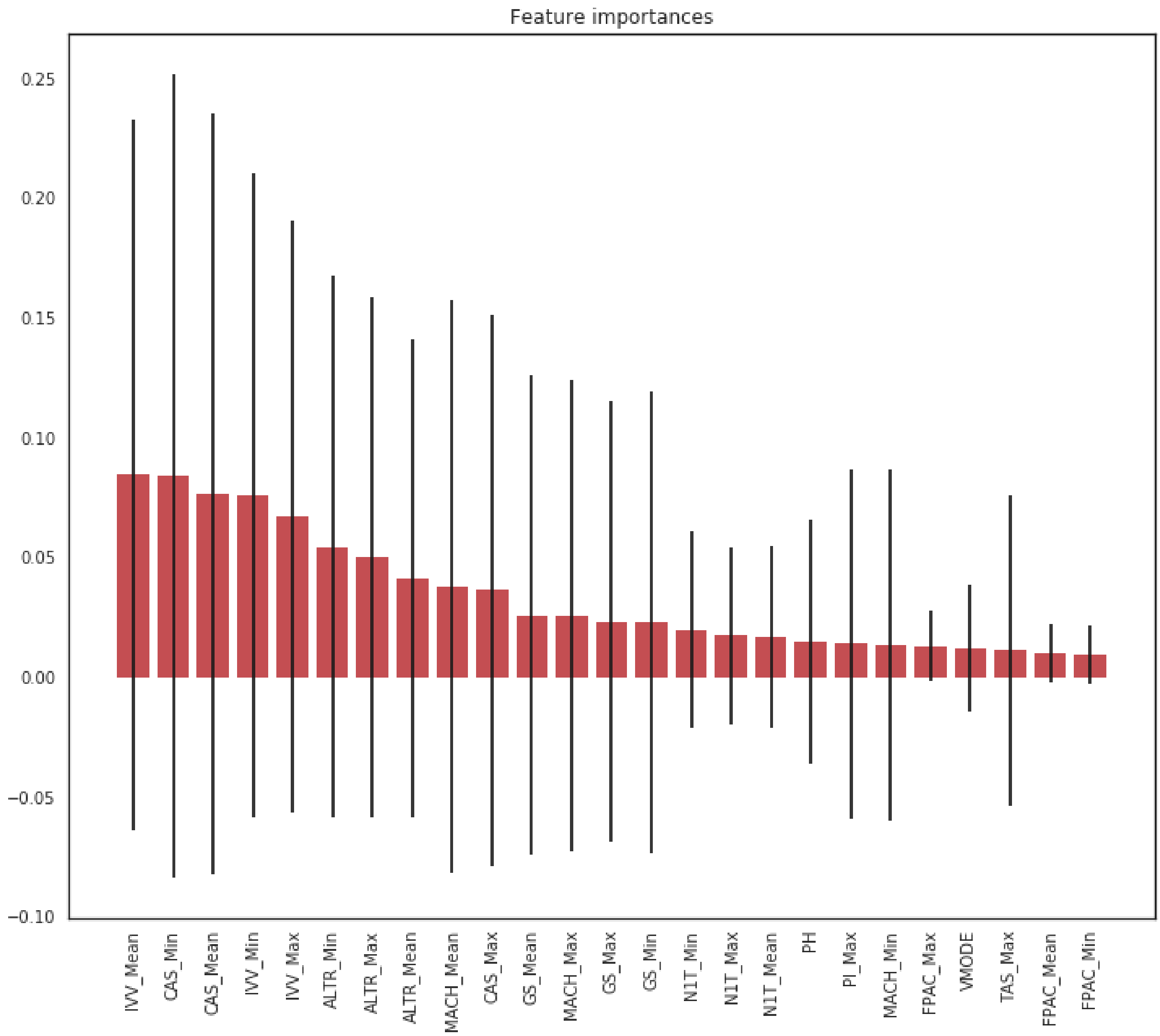
In [28]:
```
#Lets see rmse on hold out validation set
print("RMSE on train set :", rmse(tr['FF'], etr.predict(tr[feats])))
print("RMSE on hold out validation set:", rmse(val['FF'], etr.predict(val
```

```
[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[ind

    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)

- 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)

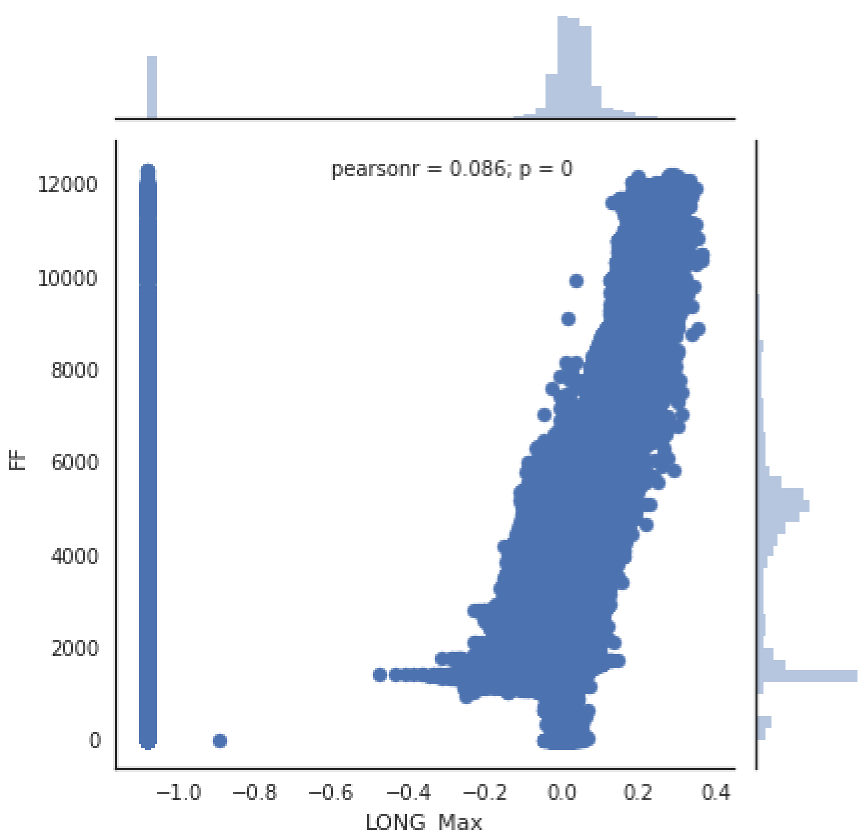
As expected we get Phase as one of the important variables.

Lets take a look at other variables

In [32]:
```
#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 "
```

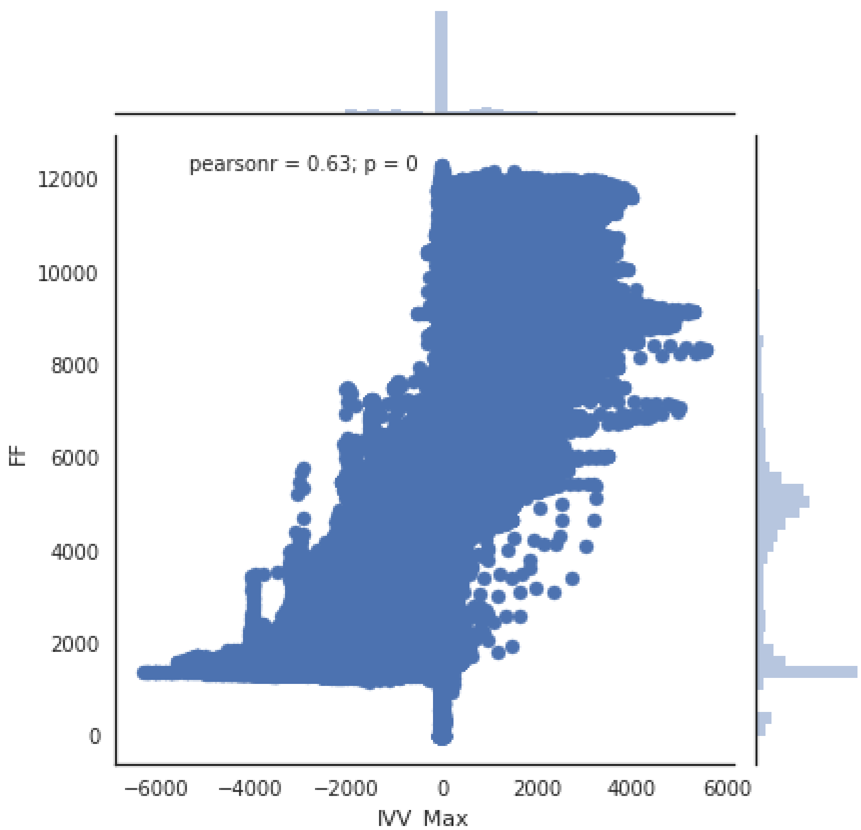
```
<Figure size 432x288 with 0 Axes>
```



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 "
```

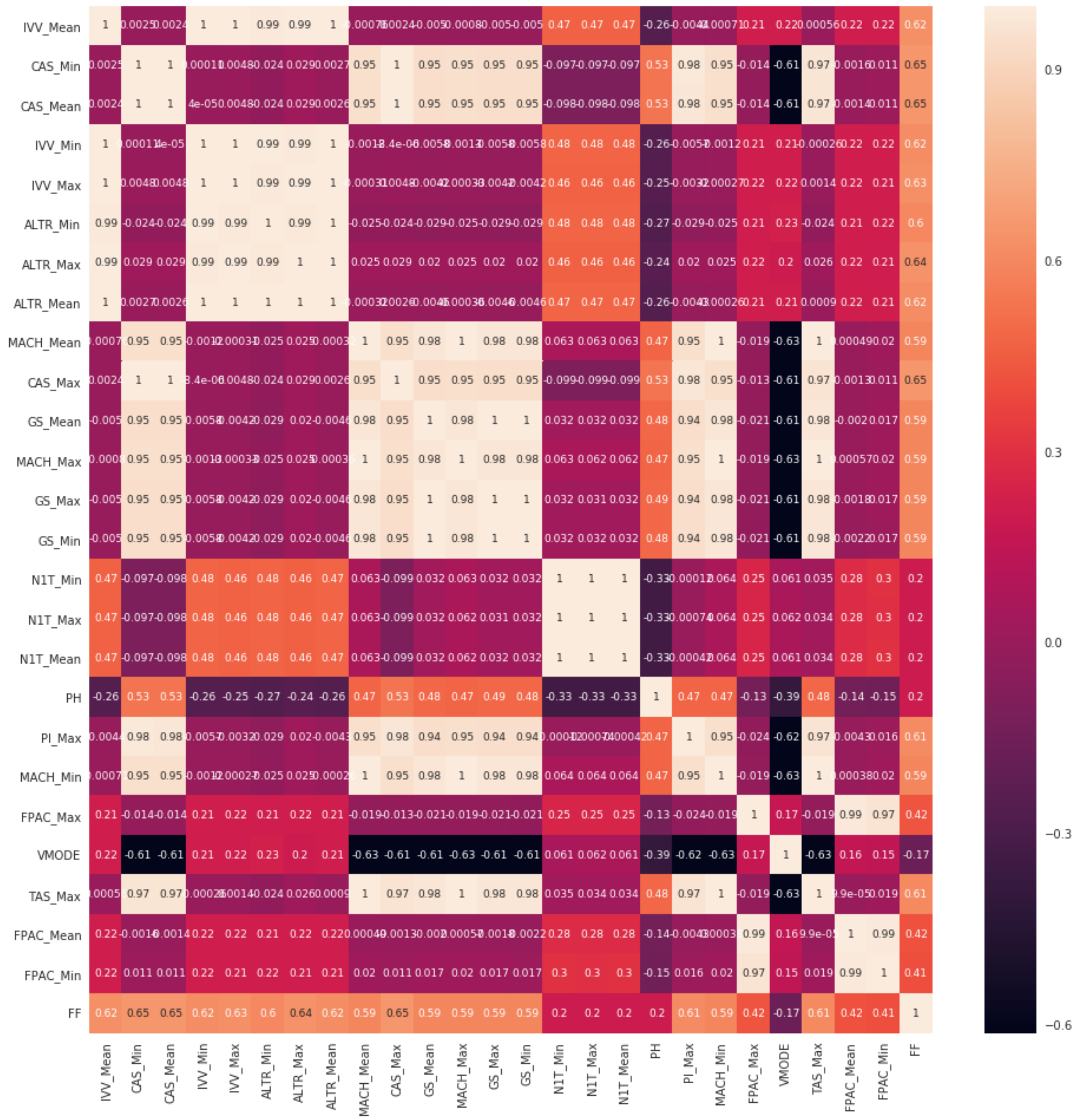
```
<Figure size 432x288 with 0 Axes>
```



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 [ ]:

```
#Lets dump everything into xgboost and see what we get.
#Warning: Not recommended to use this as final model. Remember - Garbage 
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=
xgb_dump.fit(X_tr, y_tr, eval_set=[(X_tr, y_tr), (X_val, y_val)], eval_me
```

In [ ]:

```
#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

In [ ]:

```
```