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

import time
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.inspection import permutation_importance
import warnings
warnings.filterwarnings("ignore")

from sklearn.linear_model import LassoCV
from sklearn.datasets import make_regression
from sklearn.linear_model import ElasticNetCV
```

In [2]:
```
admits = pd.read_csv('admission.csv')
discharges = pd.read_csv('discharge.csv')
census = pd.read_csv('census.csv')
#predisc_orders = pd.read_csv('count_Prepare.for.Discharge.Order.Instant.csv')
#transport = pd.read_csv('count_Transport.Requested.csv')
#avs_firstprint = pd.read_csv('count_AVS.First.Print.csv')
#surgeries = pd.read_csv('scheduled_surgeries.csv')
#surgeries.rename(columns = {'Count': 'scheduled_surgery_count'}, inplace = True)
```

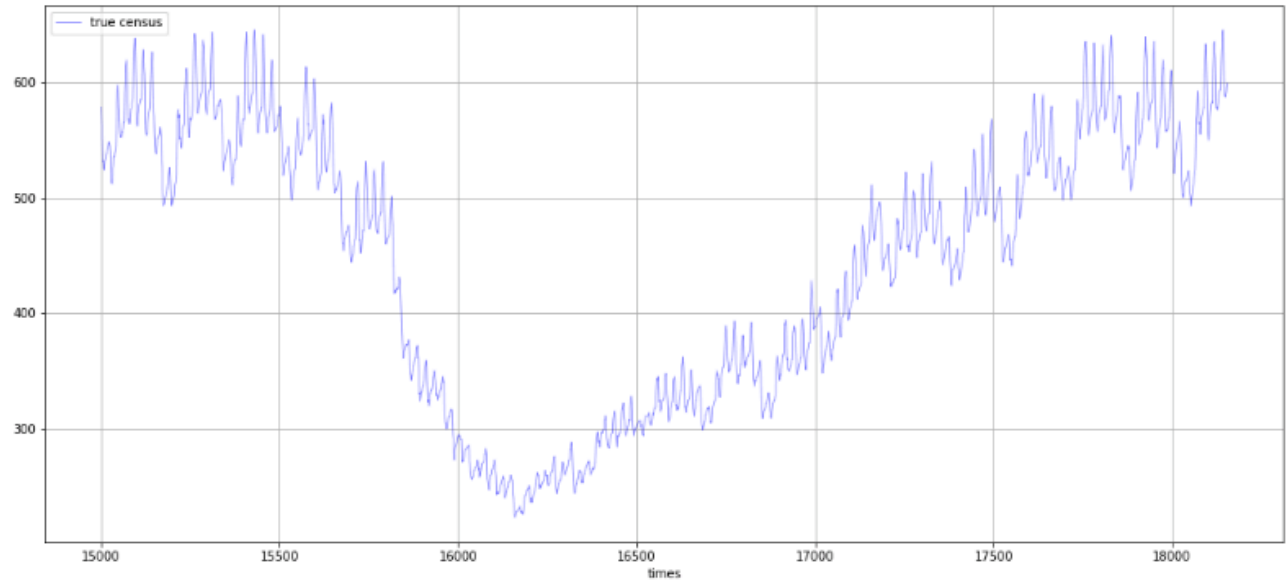
In [3]:
```
#this combines the different tables into a single dataset


admits.rename(columns = {"med": "med_admits", "surg":"surg_admits", "Total": "total_admits"}, inplace = True)
discharges.rename(columns = {"med": "med_disc", "surg":"surg_disc", "Total": "total_disc"}, inplace = True)
census.rename(columns = {"med": "med_census", "surg":"surg_census", "Total": "total_census"}, inplace = True)

#dataframes where the column with information has a different name
suffixes = ['a,b,c,d,e']
dataframes1 = (admits, discharges)
dataframe_names1 = ['admits', 'discharges', 'surgeries']

#dataframes where the column with information has the same name
#dataframes2 = (predisc_orders, transport, avs_firstprint)
#dataframe_names2 = ['predisc_orders', 'transport', 'avs_firstprint']

df = census
for (i, dataframe) in enumerate(dataframes1):
    cols_to_use = dataframe.columns.difference(df.columns)
    cols_to_use = cols_to_use.append(pd.Index(['times']))
    print(cols_to_use)
    df = df.merge(dataframe[cols_to_use],on = 'times', how = 'left', suffixes = ('',dataframe_names1[i]))

#cols_to_use = ['Total', 'med', 'surg', 'times']
#for (i, dataframe) in enumerate(dataframes2):
#    df = df.merge(dataframe[cols_to_use], how = 'left', on='times', suffixes = ('','_' + dataframe_names2[i]))

#df.rename(columns = {"Total":'total_predisc_orders', 'med':'med_predisc_orders', 'surg':'surg_predisc_orders'}, inpla
ce=True)
#df = census
#cols_to_use = discharges.columns.difference(df.columns)
#df = df.join(discharges[cols_to_use], how = 'left', rsuffix = "_d")
#cols_to_use = admits.columns.difference(df.columns)
#df = df.join(admits[cols_to_use], how = "left", rsuffix = "_a")
```

```
Index(['med_admits', 'surg_admits', 'total_admits', 'times'], dtype='object')
Index(['med_disc', 'surg_disc', 'total_disc', 'times'], dtype='object')
```

In [4]:
```
census.times[18000]
```

Out[4]:
```
'2020-06-20 00:00:00'
```

In [5]:
```
plt.figure(figsize=(18,8))
plt.xlabel('times')

ax1 = df.total_census[15000:].plot(color='blue', grid=True, label='true census', linewidth = 0.7, alpha = 0.5)

h1, l1 = ax1.get_legend_handles_labels()


plt.legend(h1, l1, loc=2)
plt.show()
```



In [6]:
```
df.columns
```

Out[6]:
```
Index(['Unnamed: 0', 'times', 'med_census', 'surg_census', 'Day_week',
       'total_census', 'Time', 'date', 'med_admits', 'surg_admits',
       'total_admits', 'med_disc', 'surg_disc', 'total_disc'],
      dtype='object')
```

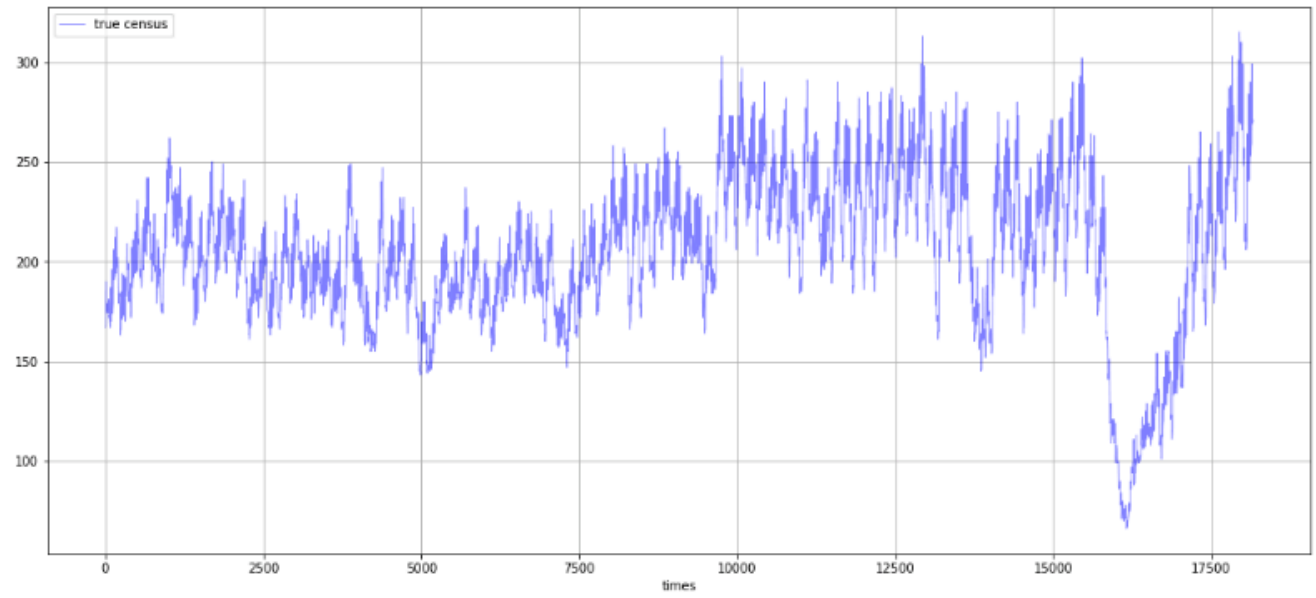
In [7]:
```
#create delta variables

variables = ['med_admits', 'surg_admits',
       'total_admits', 'med_disc', 'surg_disc', 'total_disc']

vars3h = ['med_admits', 'surg_admits',
       'total_admits', 'med_disc', 'surg_disc', 'total_disc']

vars24h = ['med_admits', 'surg_admits',
       'total_admits', 'med_disc', 'surg_disc', 'total_disc']

#varsforward = ['scheduled_surgery_count']
#for lookback in [3,8,24]:
#    for variable in variables:
#        name = 'prev_'+ str(lookback) +'h_'+ str(variable)
#        df[name] = df[variable].rolling(min_periods=1, window=lookback).sum()


for variable in vars3h:
    name = 'prev_'+ str(3) +'h_'+ str(variable)
    df[name] = df[variable].rolling(min_periods=1, window=3).sum()

for variable in vars24h:
    name = 'prev_'+ str(24) +'h_'+ str(variable)
    df[name] = df[variable].rolling(min_periods=1, window=24).sum()

#for variable in varsforward:
#    name = 'next_'+ str(8) +'h_'+ str(variable)
#    df[name] = df[variable].rolling(min_periods=1, window=24).sum()
#    df[name] = df[variable][::-1].rolling(8).sum()[::-1]

df['time_day'] = pd.to_datetime(df['Time']).apply(lambda x: x.strftime("%H"))

dfweekdays = pd.get_dummies(df.Day_week, prefix='week')
df = df.join(dfweekdays, how = 'left')
```

In [8]:
```
#we check that we have created the new variables
df.columns
```

Out[8]:
```
Index(['Unnamed: 0', 'times', 'med_census', 'surg_census', 'Day_week',
       'total_census', 'Time', 'date', 'med_admits', 'surg_admits',
       'total_admits', 'med_disc', 'surg_disc', 'total_disc',
       'prev_3h_med_admits', 'prev_3h_surg_admits', 'prev_3h_total_admits',
       'prev_3h_med_disc', 'prev_3h_surg_disc', 'prev_3h_total_disc',
       'prev_24h_med_admits', 'prev_24h_surg_admits', 'prev_24h_total_admits',
       'prev_24h_med_disc', 'prev_24h_surg_disc', 'prev_24h_total_disc',
       'time_day', 'week_Friday', 'week_Monday', 'week_Saturday',
       'week_Sunday', 'week_Thursday', 'week_Tuesday', 'week_Wednesday'],
      dtype='object')
```

In [9]:
```
#here we subset the dataframe to include only data elements between 500 and 1750
#those are the ones that don't include the low
dffirst = df[14000:15500]
#df = df[0:14000]
```

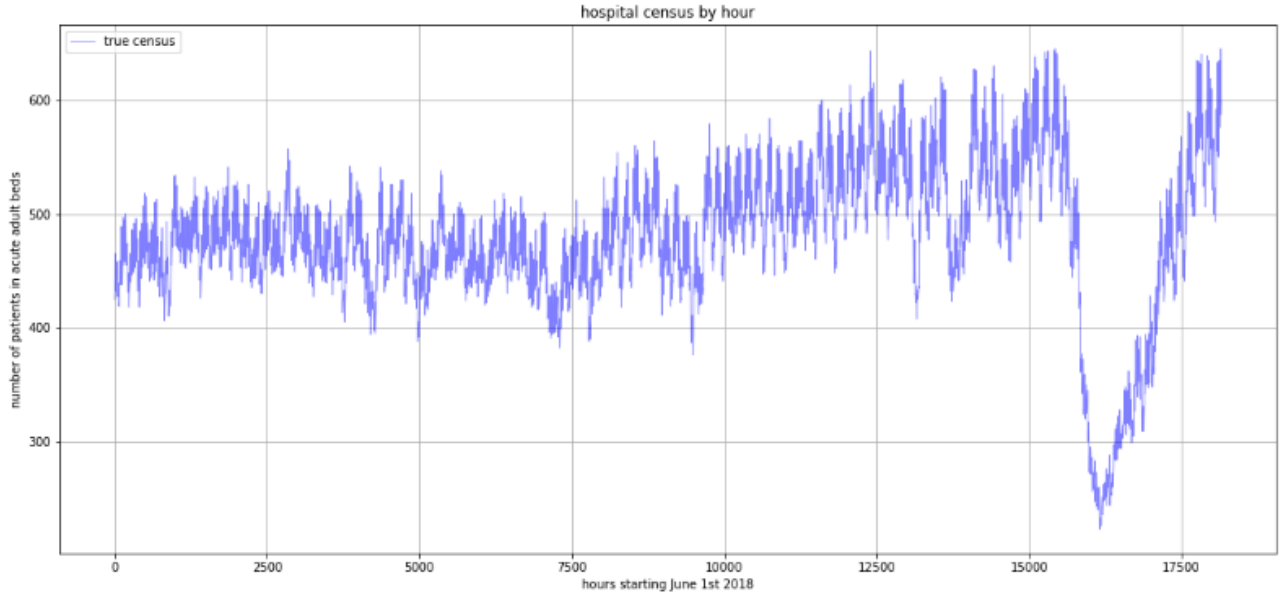
In [10]:
```
plt.figure(figsize=(18,8))
plt.xlabel('times')

ax1 = df.surg_census.plot(color='blue', grid=True, label='true census', linewidth = 0.7, alpha = 0.5)

h1, l1 = ax1.get_legend_handles_labels()


plt.legend(h1, l1, loc=2)
plt.show()
```



In [11]:
```
df.time_day.head()
```

Out[11]:
```
0    00
1    01
2    02
3    03
4    04
Name: time_day, dtype: object
```

In [12]:
```
#this is the data that I'm using for the model

plt.figure(figsize=(18,8))
plt.xlabel('times')

ax1 = df.total_census.plot(color='blue', grid=True, label='true census', linewidth = 0.7, alpha = 0.5)

h1, l1 = ax1.get_legend_handles_labels()


plt.legend(h1, l1, loc=2)
plt.title('hospital census by hour');
plt.xlabel('hours starting June 1st 2018');
plt.ylabel('number of patients in acute adult beds');
plt.show()
```



In [13]:
```
df.total_census.mean()
```

Out[13]:
```
477.5844348975545
```

In [14]:
```
#to check if we have any NaN, and where
#df.isnull().sum()
```

In [15]:
```
#this is the prediction horizon (how many hours ahead are we predicting) and the outcome of interest
horizon = -12
outcome = 'total_census'

#we create our X matrix with only the predictors
exclude = ['Unnamed: 0', 'times', 'Time', 'date', 'Day_week']
predictors = [x for x in df.columns if x not in exclude]
n_valid_rows = len(df) + horizon
X = df[predictors].head(n_valid_rows)

### then our y vector with the outcomes of interest
#use this y for cumulatives
#y = df[outcome].rolling(-horizon).sum().shift(horizon).head(n_valid_rows)

#use this y for census
y = df[outcome].shift(horizon).head(n_valid_rows) - df[outcome].head(n_valid_rows)



##we separate into train and test
X_train = X[0:12500]
X_test = X[12500:15000]
y_train = y[0:12500]
y_test = y[12500:15000]
```

Prediction using a linear model with ElasticNet regularization

In [16]:
```
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

## and we run a model for each time of day
#you can comment out the print options to make it less verbose
maelist = []
mean_caseslist = []
residuals = np.array(0)
coefficientsdf = pd.DataFrame({'Column':X_train.columns})
intercept = pd.DataFrame({'Column': 'intercept'}, index = [0])
coefficientsdf = pd.concat([intercept,coefficientsdf], axis = 0, ignore_index = True)


for tod in np.sort(X_train['time_day'].unique()):
    X_traintime = X_train[X_train['time_day']==tod];
    y_traintime = y_train[X_train['time_day']==tod];
    X_testtime = X_test[X_test['time_day']==tod]
    y_testtime = y_test[X_test['time_day']==tod]

    en = ElasticNetCV(l1_ratio=[.05, .15, .5, .7, .9, .95, .99, 1],n_jobs=8)
    en.fit(X_traintime, y_traintime)
    print("MODEL FOR: " + str(tod) + "h")

    #print("Best for alphas:")
    #print(en.alpha_)
    #print("Best l1-ratio:")
    #print(en.l1_ratio_)

    #print("R-squared for Train: %.2f" % en.score(X_traintime, y_traintime))
    #print("R-squared for Test: %.2f" % en.score(X_testtime, y_testtime))
    y_predtime = en.predict(X_testtime)
    # Calculate the absolute errors
    mae = round(np.mean(abs(y_predtime - y_testtime)),2)
    residualsnow = (np.array(y_predtime) - np.array(y_testtime))
    residuals = np.append(residuals,residualsnow)
    maelist.append(mae)
    mean_cases = round(np.mean(y_testtime),2)
    mean_caseslist.append(mean_cases)
    # Print out the mean absolute error (mae)
    #print('Mean Absolute Error:', mae, 'beds.')
    #print('Admits in the next', str(-horizon)+'h:', mean_cases, 'beds.')
    #print('')


    coefficients = pd.DataFrame({(str(tod)+'h'):en.coef_})

    intercept = pd.DataFrame({(str(tod)+'h'): en.intercept_}, index = [0])

    coefficients = pd.concat([intercept, coefficients], ignore_index=True)

    coefficientsdf = pd.concat([coefficientsdf, coefficients], axis=1)
```

```
MODEL FOR: 00h
MODEL FOR: 01h
MODEL FOR: 02h
MODEL FOR: 03h
MODEL FOR: 04h
MODEL FOR: 05h
MODEL FOR: 06h
MODEL FOR: 07h
MODEL FOR: 08h
MODEL FOR: 09h
MODEL FOR: 10h
MODEL FOR: 11h
MODEL FOR: 12h
MODEL FOR: 13h
MODEL FOR: 14h
MODEL FOR: 15h
MODEL FOR: 16h
MODEL FOR: 17h
MODEL FOR: 18h
MODEL FOR: 19h
MODEL FOR: 20h
MODEL FOR: 21h
MODEL FOR: 22h
MODEL FOR: 23h
```

In [17]:
```
#X.head(10)
```

In [18]:
```
#y
```

In [19]:
```
#coefficientsdf
```

In [20]:
```
coefficientsdf = coefficientsdf[coefficientsdf.Column != "time_day"]
file_name = "coef"+str(outcome)+str(-horizon)+"h.csv"
coefficientsdf.to_csv(file_name, index=False)
```

In [21]:
```
file_name
```

Out[21]:
```
'coeftotal_census12h.csv'
```

In [22]:
```
absresiduals = abs(residuals)
np.percentile(residuals,95)
```

Out[22]:
```
17.816289162210367
```

In [23]:
```
np.percentile(residuals,95)/X_test.total_census.mean()
```

Out[23]:
```
0.03394432345315463
```

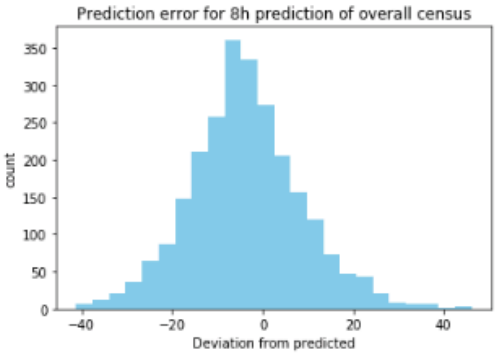
In [ ]:
```

```

In [24]:
```
absresiduals.mean()
```

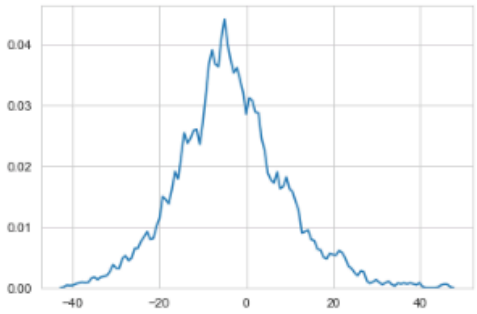
Out[24]:
```
10.12447563105875
```

In [25]:
```
plt.hist(residuals, bins = 24, color = 'skyblue');
plt.title('Prediction error for 8h prediction of overall census');
plt.xlabel('Deviation from predicted');
plt.ylabel('count')
```

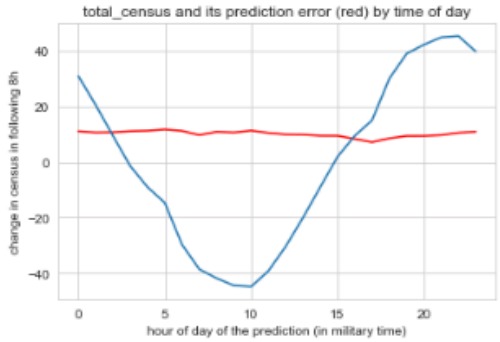
Out[25]:
```
Text(0, 0.5, 'count')
```



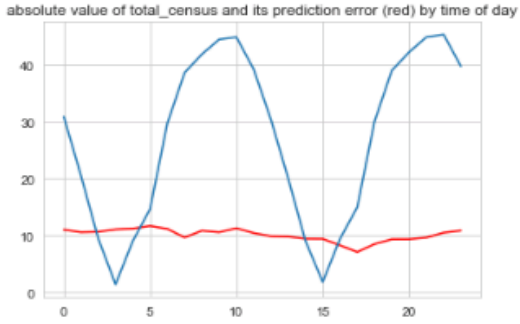
In [26]:
```
sns.set_style('whitegrid');
sns.kdeplot(residuals, bw=0.5);
```



In [27]:
```
plt.plot(maelist, c = "red");
plt.plot(mean_caseslist);
plt.title(str(outcome)+' and its prediction error (red) by time of day');
plt.xlabel('hour of day of the prediction (in military time)');
plt.ylabel('change in census in following 8h');
```



In [28]:
```
plt.plot(maelist, c = "red");
absolute_value =  [abs(ele) for ele in mean_caseslist]
plt.plot(absolute_value);
plt.title('absolute value of '+str(outcome)+' and its prediction error (red) by time of day');
```



In [29]:
```
np.mean(absolute_value)
```

Out[29]:
```
27.18958333333333
```

GBM model

In [30]:
```
######### THIS TAKES A LONG TIME TO RUN (~10-25 min) ###########


gb_hyperparameters = {
    "n_estimators": [50, 100, 200, 500],
    "min_samples_split" : [2, 5, 7],
    'learning_rate': [0.01, 0.05],
    "min_samples_leaf": [2, 3],
    "max_depth":[2,3,5,10]
}

#rf_random = RandomizedSearchCV(estimator = rf,
#                               param_distributions = random_grid,
#                               n_iter = 100, cv = 3, verbose=2,
#                               random_state=109, n_jobs = -1)

gbr = GradientBoostingRegressor(validation_fraction = 0.3)
#gs = GridSearchCV(estimator=gbr, param_grid=gb_hyperparameters, scoring='neg_mean_squared_error', cv=3, n_jobs=-1)
gs_random = RandomizedSearchCV(estimator = gbr,
                               param_distributions = gb_hyperparameters,
                               n_iter = 75, cv = 3, verbose=2,
                               random_state=109, n_jobs = -1)
gs_random = gs_random.fit(X_train, y_train)

gs_random.best_params_
```

```
Fitting 3 folds for each of 75 candidates, totalling 225 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks       | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 154 tasks       | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done 225 out of 225 | elapsed:  8.1min finished
```

Out[30]:
```
{'n_estimators': 500,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_depth': 5,
 'learning_rate': 0.01}
```

In [31]:
```
gbr = gs_random.best_estimator_
gbr.fit(X_train, y_train)
```

Out[31]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.01, loss='ls', max_depth=5,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=2, min_samples_split=5,
                          min_weight_fraction_leaf=0.0, n_estimators=500,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.3, verbose=0, warm_start=False)
```

In [32]:
```
print("R-squared for Train: %.2f" % gbr.score(X_train, y_train))
print("R-squared for Test: %.2f" % gbr.score(X_test, y_test))
y_pred = gbr.predict(X_test)
# Calculate the absolute errors
mae = round(np.mean(abs(y_pred - y_test)),2)
mean_cases = round(np.mean(y_test),2)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', mae, 'beds.')
print('Average census', mean_cases, 'beds.')
```

```
R-squared for Train: 0.91
R-squared for Test: 0.87
Mean Absolute Error: 9.96 beds.
Average census 0.14 beds.
```

In [33]:
```
#residuals = y_pred - y_test
np.percentile(residuals, 95)/X_test.total_census.mean()
```

Out[33]:
```
0.03394432345315463
```

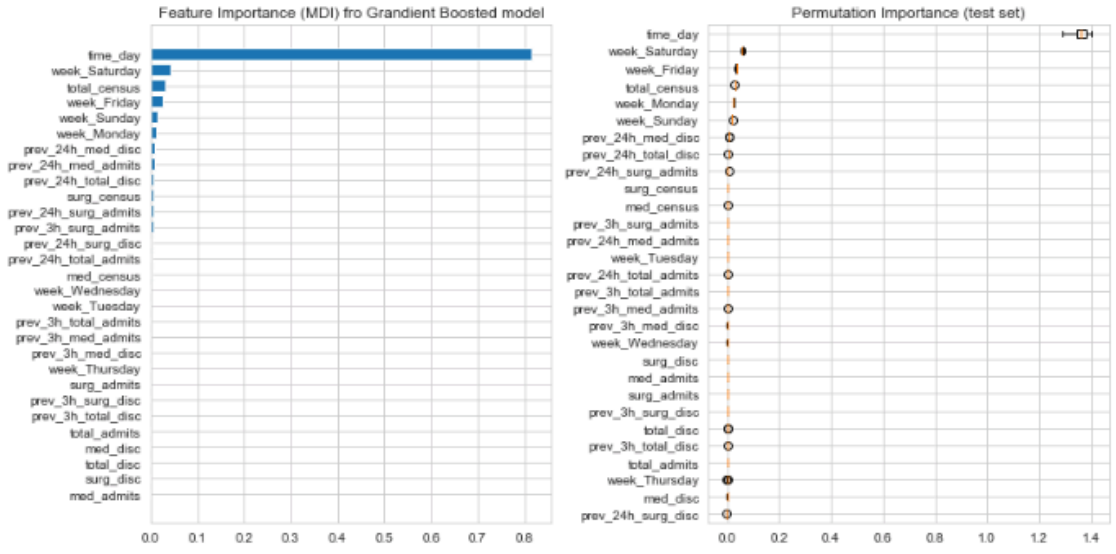
In [34]:
```
np.std(X_test.total_census)
```

Out[34]:
```
42.43054767499474
```

In [35]:
```
X_train.total_census.mean()
```

Out[35]:
```
476.37552
```

In [36]:
```
feature_importance = gbr.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')

#column_names = np.array(["Time of day", "Saturday", "Sunday", "Friday", "Total discharges"])
plt.yticks(pos, np.array(X_test.columns)[sorted_idx])
#plt.yticks(pos, column_names[sorted_idx])


plt.title('Feature Importance (MDI) fro Grandient Boosted model')

result = permutation_importance(gbr, X_test, y_test, n_repeats=10,
                                random_state=42, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(result.importances[sorted_idx].T,
            vert=False, labels=np.array(X_test.columns)[sorted_idx])
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
```



Naive Model

Predict total census by simply calculating the average rate of increase between t0 and t12 for each time of the day and each day of the week.

In [37]:
```
X_train.head()
```

Out[37]:

	med_census	surg_census	total_census	med_admits	surg_admits	total_admits	med_disc	surg_disc	total_disc	prev_3h_med_admits	...	prev_24h_surg_di
0	258	167	425	1	0	1	0	0	0	1.0	...	0
1	260	167	427	2	0	2	0	0	0	3.0	...	0
2	259	167	426	0	0	0	1	0	1	3.0	...	0
3	262	168	430	3	1	4	0	0	0	5.0	...	0
4	263	168	431	1	0	1	0	0	0	4.0	...	0

5 rows × 29 columns

In [38]:
```
# create the total census in 12 hours for each row
X_train['total_census_t12'] = X_train.total_census + y_train
X_test['total_census_t12'] = X_test.total_census + y_test

# create the ratio of census at t12 and at t0
X_train['total_census_ratio'] = X_train.total_census_t12 / X_train.total_census
```

In [39]:
```
X_train.columns
```

Out[39]:
```
Index(['med_census', 'surg_census', 'total_census', 'med_admits',
       'surg_admits', 'total_admits', 'med_disc', 'surg_disc', 'total_disc',
       'prev_3h_med_admits', 'prev_3h_surg_admits', 'prev_3h_total_admits',
       'prev_3h_med_disc', 'prev_3h_surg_disc', 'prev_3h_total_disc',
       'prev_24h_med_admits', 'prev_24h_surg_admits', 'prev_24h_total_admits',
       'prev_24h_med_disc', 'prev_24h_surg_disc', 'prev_24h_total_disc',
       'time_day', 'week_Friday', 'week_Monday', 'week_Saturday',
       'week_Sunday', 'week_Thursday', 'week_Tuesday', 'week_Wednesday',
       'total_census_t12', 'total_census_ratio'],
      dtype='object')
```

NAIVE MODEL USING TOD

In [40]:
```
#calculate average ratio by TOD (times of a day)
navie_ratio_tod = pd.DataFrame(X_train.groupby('time_day').mean()['total_census_ratio'])
navie_ratio_tod.columns = ['total_census_ratio_tod_ave']
navie_ratio_tod
```

Out[40]:

time_day	total_census_ratio_tod_ave
00	1.052423
01	1.035381
02	1.017356
03	0.999298
04	0.985306
05	0.975813
06	0.953234
07	0.938284
08	0.933232
09	0.929957
10	0.930200
11	0.938519
12	0.951236
13	0.966365
14	0.984032
15	1.001713
16	1.015961
17	1.025843
18	1.050109
19	1.066822
20	1.072646
21	1.076499
22	1.076277
23	1.066673

In [41]:
```
# calcuate predicted total census at t12 using navie ratio tod on the **test set**
X_test_tod = pd.merge(X_test,navie_ratio_tod, how = 'left', on = 'time_day')
y_pred_naive_tod = np.array(X_test.total_census) * np.array(X_test_tod.total_census_ratio_tod_ave)
X_test_tod.head()
```

Out[41]:

	med_census	surg_census	total_census	med_admits	surg_admits	total_admits	med_disc	surg_disc	total_disc	prev_3h_med_admits	...	time_day	week_
0	305	220	525	3	1	4	0	0	0	6.0	...	07	
1	308	220	528	3	0	3	0	0	0	6.0	...	08	
2	309	220	529	1	0	1	0	0	0	7.0	...	09	
3	307	221	528	1	2	3	3	1	4	5.0	...	10	
4	307	215	522	2	0	2	4	6	10	4.0	...	11	

5 rows × 31 columns

In [42]:
```
#MAE
mae_naive_tod = round(np.mean(abs(y_pred_naive_tod - X_test.total_census_t12)),2)

# 95 percentile residuals
residuals_naive_tod = (np.array(y_pred_naive_tod) - np.array(X_test.total_census_t12))
residuals_naive_tod_95 = np.percentile(residuals_naive_tod,95)

# average of absolute residuals
absresiduals_naive_tod = abs(residuals_naive_tod)
absresiduals_naive_tod_mean = absresiduals_naive_tod.mean()

print("mae for naive model using only times of the day: ",mae_naive_tod)
print("95 percentile residuals for naive model using only times of the day: ",residuals_naive_tod_95)
print("relative 95% error for naive model using only times of the day: ",residuals_naive_tod_95)
print("average of absolute residuals for naive model using only times of the day: ",absresiduals_naive_tod_mean)
```

```
mae for naive model using only times of the day:  14.42
95 percentile residuals for naive model using only times of the day:  29.67433648044756
relative 95% error for naive model using only times of the day:  29.67433648044756
average of absolute residuals for naive model using only times of the day:  14.41927948445061
```

In [43]:
```
residuals_naive_tod_95 - absresiduals_naive_tod_mean
```

Out[43]:
```
15.25505699599695
```

In [44]:
```
residuals_naive_tod_95/X_test.total_census.mean()
```

Out[44]:
```
0.05653676063400238
```

NAIVE MODEL USING TOD & DOW

In [45]:
```
# undo dummy DOW
dow_train = X_train[['week_Friday', 'week_Monday', 'week_Saturday','week_Sunday', 'week_Thursday', 'week_Tuesday', 'week_Wednesday']]
dow_train = pd.get_dummies(dow_train).idxmax(1)
X_train['dow'] = dow_train

dow_test = X_test[['week_Friday', 'week_Monday', 'week_Saturday','week_Sunday', 'week_Thursday', 'week_Tuesday', 'week_Wednesday']]
dow_test = pd.get_dummies(dow_test).idxmax(1)
X_test['dow'] = dow_test

X_train.head()
```

Out[45]:

	med_census	surg_census	total_census	med_admits	surg_admits	total_admits	med_disc	surg_disc	total_disc	prev_3h_med_admits	...	week_Friday	we
0	258	167	425	1	0	1	0	0	0	1.0	...	1	
1	260	167	427	2	0	2	0	0	0	3.0	...	1	
2	259	167	426	0	0	0	1	0	1	3.0	...	1	
3	262	168	430	3	1	4	0	0	0	5.0	...	1	
4	263	168	431	1	0	1	0	0	0	4.0	...	1	

5 rows × 32 columns

In [46]:
```
#calculate average ratio by TOD (times of a day) and DOW (days of a week)
navie_ratio_tod_dow = pd.DataFrame(X_train.groupby(['time_day','dow']).mean()['total_census_ratio'])
navie_ratio_tod_dow .columns = ['total_census_ratio_tod_dow_ave']
navie_ratio_tod_dow.head(10)
```

Out[46]:

		total_census_ratio_tod_dow_ave
time_day	dow	
00	week_Friday	1.060328
	week_Monday	1.084959
	week_Saturday	1.005496
	week_Sunday	1.016283
	week_Thursday	1.060053
	week_Tuesday	1.070732
	week_Wednesday	1.070126
01	week_Friday	1.040454
	week_Monday	1.071169
	week_Saturday	0.990743

In [47]:
```
# calcuate predicted total census at t12 using navie ratio tod_dow on the **test set**
X_test_tod_dow = pd.merge(X_test,navie_ratio_tod_dow, how = 'left', on = ['time_day','dow'])
y_pred_naive_tod_dow = np.array(X_test.total_census) * np.array(X_test_tod_dow.total_census_ratio_tod_dow_ave)
X_test_tod_dow.head()
```

Out[47]:

	med_census	surg_census	total_census	med_admits	surg_admits	total_admits	med_disc	surg_disc	total_disc	prev_3h_med_admits	...	week_Friday	we
0	305	220	525	3	1	4	0	0	0	6.0	...	0	
1	308	220	528	3	0	3	0	0	0	6.0	...	0	
2	309	220	529	1	0	1	0	0	0	7.0	...	0	
3	307	221	528	1	2	3	3	1	4	5.0	...	0	
4	307	215	522	2	0	2	4	6	10	4.0	...	0	

5 rows × 32 columns

In [48]:
```
#MAE
mae_naive_tod_dow = round(np.mean(abs(y_pred_naive_tod_dow - X_test.total_census_t12)),2)

# 95 percentile residuals
residuals_naive_tod_dow = (np.array(y_pred_naive_tod_dow) - np.array(X_test.total_census_t12))
residuals_naive_tod_dow_95 = np.percentile(residuals_naive_tod_dow,95)

# average of absolute residuals
absresiduals_naive_tod_dow = abs(residuals_naive_tod_dow)
absresiduals_naive_tod_dow_mean = absresiduals_naive_tod_dow.mean()

print("mae for naive model using time of the day and day of the week: ",mae_naive_tod_dow)
print("95 percentile residuals for naive model using time of the day and day of the week: ",residuals_naive_tod_dow_9
5)
print("average of absolute residuals for naive model using time of the day and day of the week: ",absresiduals_naive_t
od_dow_mean)
```

```
mae for naive model using time of the day and day of the week:  9.98
95 percentile residuals for naive model using time of the day and day of the week:  22.661824377954137
average of absolute residuals for naive model using time of the day and day of the week:  9.980184689441835
```

In [49]:
```
residuals_naive_tod_dow_95 - absresiduals_naive_tod_dow_mean
```

Out[49]:
```
12.681639688512302
```

In [50]:
```
residuals_naive_tod_dow_95/X_test.total_census.mean()
```

Out[50]:
```
0.043176235506744806
```

In [51]:
```
X_test.total_census.mean()
```

Out[51]:
```
524.868
```