Assignment 9

This dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring. The recordings were automatically captured in the patient's homes.

Columns in the table contain subject number, subject age, subject gender, time interval from baseline recruitment date, motor UPDRS, total UPDRS, and 16 biomedical voice measures.

Each row corresponds to one of 5,875 voice recording from these individuals.

The main aim of the data is to predict the motor and total UPDRS scores ('motor_UPDRS' and 'total_UPDRS') from the 16 voice measures.

In [28]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import cross_validate, cross_val_score, train_test_split
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn import tree
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
```

In [29]:
```
os.chdir(r"E:\GoogleDriveNew\PSU\DAAN862\Course contents\Lesson 9")
parkinson = pd.read_csv("parkinsons_updrs.data")
pd.set_option('max_rows',25)
pd.set_option('precision',3)
```

In [30]:
```
parkinson.columns
```

Out[30]:
```
Index(['subject#', 'age', 'sex', 'test_time', 'motor_UPDRS', 'total_UPDRS',
       'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP',
       'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE'],
      dtype='object')
```

Since column names contains puncations, we need to redefine the column names

In [31]:
```
parkinson.columns = ['subject_ID', 'age', 'sex', 'test_time', 'motor_UPDRS',
                     'total_UPDRS', 'Jitter_percentage', 'Jitter_Abs',
                     'Jitter_RAP', 'Jitter_PPQ5', 'Jitter_DDP','Shimmer',
                     'Shimmer(dB)', 'Shimmer_APQ3', 'Shimmer_APQ5','Shimmer_APQ11',
                     'Shimmer_DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE']
```

In [32]:
```
#Optional, re package will be introduced in Lesson 12.
#import re

#colnames = list(parkinson.columns)
#for i in range(len(colnames)):
#    colnames[i] = re.sub('[^0-9a-zA-Z]+', '_', colnames[i])

#colnames
```

In [33]:
```
parkinson.head()
```

Out[33]:

	subject_ID	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter_percentage	Jitter_Abs	Jitter_RAP	Jitter_PPQ5	...	Shimmer(dB)	Shimmer_APQ3	Shimmer_APQ5	Shimm
0	1	72	0	5.643	28.199	34.398	0.007	3.380e-05	4.010e-03	0.003	...	0.230	0.014	0.013	0.017
1	1	72	0	12.666	28.447	34.894	0.003	1.680e-05	1.320e-03	0.002	...	0.179	0.010	0.011	0.017
2	1	72	0	19.681	28.695	35.389	0.005	2.462e-05	2.050e-03	0.002	...	0.181	0.007	0.008	0.015
3	1	72	0	25.647	28.905	35.810	0.005	2.657e-05	1.910e-03	0.003	...	0.327	0.011	0.013	0.020
4	1	72	0	33.642	29.187	36.375	0.003	2.014e-05	9.300e-04	0.001	...	0.176	0.007	0.009	0.018

5 rows × 22 columns

In [34]:
```
parkinson.shape
```

Out[34]:
```
(5875, 22)
```

1. Perform an exploratory analysis on the data and Remove motor_UPDRS column (10 points)

In [35]:
```
parkinson.subject_ID = parkinson.subject_ID.astype('category')
```

In [36]:
```
parkinson.describe(include = "all")
```

Out[36]:

	subject_ID	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter_percentage	Jitter_Abs	Jitter_RAP	Jitter_PPQ5	...	Shimmer(dB)	Shimmer_APQ3	Shimme
count	5875.0	5875.000	5875.000	5875.000	5875.000	5875.000	5.875e+03	5.875e+03	5.875e+03	5.875e+03	...	5875.000	5875.000	5875.00
unique	42.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
top	29.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
freq	168.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
mean	NaN	64.805	0.318	92.864	21.296	29.019	6.154e-03	4.403e-05	2.987e-03	3.277e-03	...	0.311	0.017	0.020
std	NaN	8.822	0.466	53.446	8.129	10.700	5.624e-03	3.598e-05	3.124e-03	3.732e-03	...	0.230	0.013	0.017
min	NaN	36.000	0.000	-4.263	5.038	7.000	8.300e-04	2.250e-06	3.300e-04	4.300e-04	...	0.026	0.002	0.002
25%	NaN	58.000	0.000	46.847	15.000	21.371	3.580e-03	2.243e-05	1.580e-03	1.820e-03	...	0.175	0.009	0.011
50%	NaN	65.000	0.000	91.523	20.871	27.576	4.900e-03	3.453e-05	2.250e-03	2.490e-03	...	0.253	0.014	0.016
75%	NaN	72.000	1.000	138.445	27.596	36.399	6.800e-03	5.334e-05	3.290e-03	3.460e-03	...	0.365	0.021	0.024
max	NaN	85.000	1.000	215.490	39.511	54.992	9.999e-02	4.456e-04	5.754e-02	6.956e-02	...	2.107	0.163	0.167

11 rows × 22 columns

There are 42 unique subject in this data. The age is between 36-85. The main aim of the data is to predict the motor and total UPDRS scores ('motor_UPDRS' and 'total_UPDRS') from the 16 voice measures. I will remove irrelevant variables.

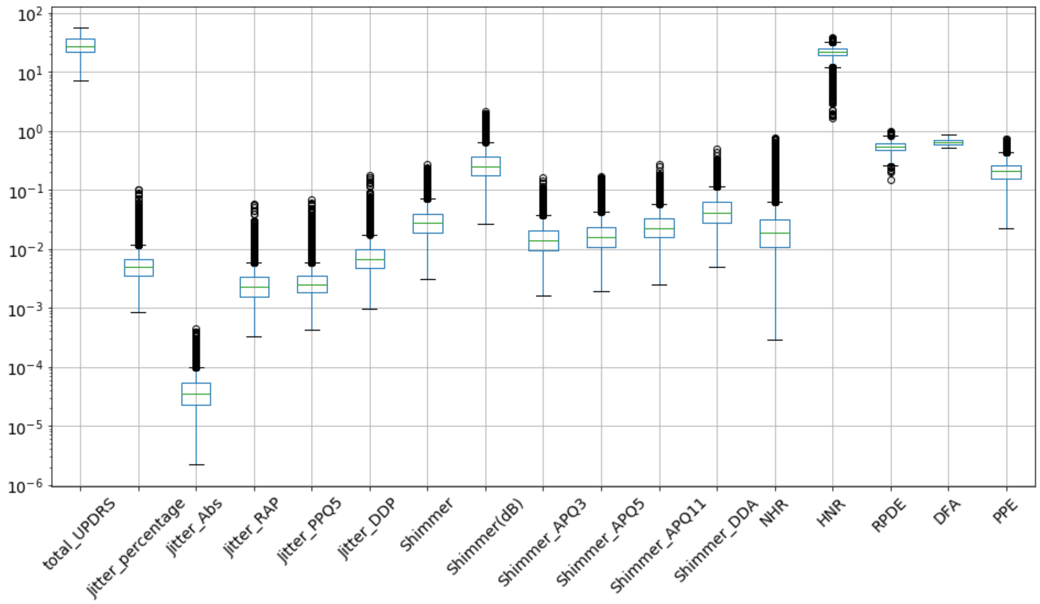
In [37]:
```
parkinson = parkinson.drop(columns = ['subject_ID', 'age', 'sex',
                                      'test_time', 'motor_UPDRS'])
```

In [38]:
```
parkinson.corr().round(2)
```

Out[38]:

	total_UPDRS	Jitter_percentage	Jitter_Abs	Jitter_RAP	Jitter_PPQ5	Jitter_DDP	Shimmer	Shimmer(dB)	Shimmer_APQ3	Shimmer_APQ5	Shimmer_APQ11	S
total_UPDRS	1.00	0.07	0.07	0.06	0.06	0.06	0.09	0.10	0.08	0.08	0.12	0.
Jitter_percentage	0.07	1.00	0.87	0.98	0.97	0.98	0.71	0.72	0.66	0.69	0.65	0.
Jitter_Abs	0.07	0.87	1.00	0.84	0.79	0.84	0.65	0.66	0.62	0.62	0.59	0.
Jitter_RAP	0.06	0.98	0.84	1.00	0.95	1.00	0.68	0.69	0.65	0.66	0.60	0.
Jitter_PPQ5	0.06	0.97	0.79	0.95	1.00	0.95	0.73	0.73	0.68	0.73	0.67	0.
Jitter_DDP	0.06	0.98	0.84	1.00	0.95	1.00	0.68	0.69	0.65	0.66	0.60	0.
Shimmer	0.09	0.71	0.65	0.68	0.73	0.68	1.00	0.99	0.98	0.98	0.94	0.
Shimmer(dB)	0.10	0.72	0.66	0.69	0.73	0.69	0.99	1.00	0.97	0.98	0.94	0.
Shimmer_APQ3	0.08	0.66	0.62	0.65	0.68	0.65	0.98	0.97	1.00	0.96	0.89	1.
Shimmer_APQ5	0.08	0.69	0.62	0.66	0.73	0.66	0.98	0.98	0.96	1.00	0.94	0.
Shimmer_APQ11	0.12	0.65	0.59	0.60	0.67	0.60	0.94	0.94	0.89	0.94	1.00	0.
Shimmer_DDA	0.08	0.66	0.62	0.65	0.68	0.65	0.98	0.97	1.00	0.96	0.89	1.
NHR	0.06	0.83	0.70	0.79	0.86	0.79	0.80	0.80	0.73	0.80	0.71	0.
HNR	-0.16	-0.68	-0.71	-0.64	-0.66	-0.64	-0.80	-0.80	-0.78	-0.79	-0.78	-0
RPDE	0.16	0.43	0.55	0.38	0.38	0.38	0.47	0.47	0.44	0.45	0.48	0.
DFA	-0.11	0.23	0.35	0.21	0.18	0.21	0.13	0.13	0.13	0.13	0.18	0.
PPE	0.16	0.72	0.79	0.67	0.66	0.67	0.62	0.64	0.58	0.59	0.62	0.

In [39]:
```
plt.figure(figsize = (16, 8))
parkinson.boxplot(rot = 45, fontsize=14)
plt.yscale('log')
```



2. Use cross-validation to build a linear regression model to predict total_UPDRS (25 points)

From the above correlation matrics, we can see that Jitter related columns have very strong correlation, Shimmer related columns have strong correlation. For linear model, we only pick up one column for each. The best way is to use PCA to remove colinear issues. Since tree models and neural networks doesn't require variables to be independent, therefore, all variable will be used.

In [40]:
```
X_selected = parkinson[['Jitter_percentage', 'Shimmer', 'NHR',
                        'HNR', 'RPDE', 'DFA', 'PPE']]
y = parkinson.total_UPDRS
X_selected.corr()
```

Out[40]:

	Jitter_percentage	Shimmer	NHR	HNR	RPDE	DFA	PPE
Jitter_percentage	1.000	0.710	0.825	-0.675	0.427	0.227	0.722
Shimmer	0.710	1.000	0.795	-0.801	0.468	0.133	0.616
NHR	0.825	0.795	1.000	-0.684	0.417	-0.022	0.565
HNR	-0.675	-0.801	-0.684	1.000	-0.659	-0.291	-0.759
RPDE	0.427	0.468	0.417	-0.659	1.000	0.192	0.566
DFA	0.227	0.133	-0.022	-0.291	0.192	1.000	0.395
PPE	0.722	0.616	0.565	-0.759	0.566	0.395	1.000

For regression, it is typical to use mean square error or mean absolute error as the metrics. R2 is not a good metrics to compare models. In the cross_validation the names for them are neg_mean_absolute_error and neg_mean_squared_error. The reseaon to use negative values for them is that you can always select model by the maximum of these scores.

In [41]:
```
linear_reg = linear_model.LinearRegression()
metricses = ['r2', 'neg_mean_absolute_error', 'neg_mean_squared_error']
linear_reg_results = cross_validate(linear_reg, X_selected, y, cv = 10,
                                    scoring = metricses, return_train_score = True)
linear_reg_results = pd.DataFrame(linear_reg_results)
linear_reg_results.mean()
```

Out[41]:
```
fit_time                          1.397e-03
score_time                        4.985e-04
test_r2                          -1.142e+00
train_r2                          9.655e-02
test_neg_mean_absolute_error     -9.249e+00
train_neg_mean_absolute_error    -8.321e+00
test_neg_mean_squared_error      -1.250e+02
train_neg_mean_squared_error     -1.031e+02
dtype: float64
```

The data is sorted by subject ID. Cross validataion divides the data into k part in order (without shuffling or resampling). The train data is lacking the information of particular subject that's why R2 is negative. We can shuffle data first, then assign it to X, y.

In [42]:
```
from sklearn.utils import shuffle
parkinson_shuffled = shuffle(parkinson)
X_selected_shuffled = parkinson_shuffled[['Jitter_percentage', 'Shimmer', 'NHR',
                                          'HNR', 'RPDE', 'DFA', 'PPE']]
y_shuffled = parkinson_shuffled.total_UPDRS
linear_reg = linear_model.LinearRegression()
metricses = ['r2', 'neg_mean_absolute_error', 'neg_mean_squared_error']
linear_reg_results_shuffled = cross_validate(linear_reg, X_selected_shuffled, y_shuffled,
                                             cv = 5,
                                             scoring = metricses,
                                             return_train_score = True)
linear_reg_results_shuffled = pd.DataFrame(linear_reg_results_shuffled)
linear_reg_results_shuffled.mean()
```

Out[42]:
```
fit_time                        1.595e-03
score_time                      1.996e-04
test_r2                         8.818e-02
train_r2                        9.047e-02
test_neg_mean_absolute_error   -8.389e+00
train_neg_mean_absolute_error  -8.382e+00
test_neg_mean_squared_error    -1.043e+02
train_neg_mean_squared_error   -1.041e+02
dtype: float64
```

3. Use cross-validation to build a regression tree model to predict total_UPDRS (25 points)

In [43]:
```
X_all = parkinson_shuffled.loc[:, parkinson.columns != "total_UPDRS"]
y = parkinson_shuffled.total_UPDRS
```

In [44]:
```
tree_model = tree.DecisionTreeRegressor(max_depth = 5)
tree_results = cross_validate(tree_model, X_all, y, cv = 10,
                              scoring = metricses,
                              return_train_score = True)
tree_results = pd.DataFrame(tree_results)
tree_results
```

Out[44]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_absolute_error	train_neg_mean_absolute_error	test_neg_mean_squared_error	train_neg_mean_squared_error
0	0.032	9.975e-04	0.127	0.235	-7.941	-7.482	-100.255	-87.549
1	0.029	9.975e-04	0.181	0.228	-8.018	-7.478	-97.849	-87.935
2	0.031	9.973e-04	0.132	0.248	-7.621	-7.409	-93.925	-86.554
3	0.017	0.000e+00	0.202	0.223	-7.509	-7.532	-90.355	-89.039
4	0.039	0.000e+00	0.112	0.219	-8.120	-7.540	-104.815	-89.049
5	0.025	0.000e+00	0.131	0.247	-7.724	-7.438	-92.967	-86.774
6	0.030	0.000e+00	0.245	0.240	-7.799	-7.387	-92.771	-86.289
7	0.018	0.000e+00	0.147	0.217	-7.857	-7.564	-100.190	-89.363
8	0.031	0.000e+00	0.153	0.247	-7.814	-7.418	-97.216	-86.158
9	0.035	9.975e-04	0.079	0.226	-7.963	-7.561	-99.437	-89.177

In [45]:
```
tree_results.mean()
```

Out[45]:
```
fit_time                        2.870e-02
score_time                      3.990e-04
test_r2                         1.508e-01
train_r2                        2.331e-01
test_neg_mean_absolute_error   -7.837e+00
train_neg_mean_absolute_error  -7.481e+00
test_neg_mean_squared_error    -9.698e+01
train_neg_mean_squared_error   -8.779e+01
dtype: float64
```

4. Use cross-validation to build a neural network model to predict total_UPDRS (25 points)

In [46]:
```
scaler = MinMaxScaler()
parkinson_scaled = scaler.fit_transform(parkinson_shuffled)
parkinson_scaled = pd.DataFrame(parkinson_scaled, columns = parkinson.columns)
X_scaled = parkinson_scaled.loc[:, parkinson.columns != "total_UPDRS"]
y_scaled = parkinson_scaled.total_UPDRS
```

In [47]:
```
neural_net_model = MLPRegressor((20, 10),activation = 'relu', max_iter = 10000)
neural_net_results = cross_validate(neural_net_model, X_scaled, y_scaled,
                                    cv = 10,
                                    scoring = metricses,
                                    return_train_score = True)
neural_net_results = pd.DataFrame(neural_net_results)
neural_net_results
```

Out[47]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_absolute_error	train_neg_mean_absolute_error	test_neg_mean_squared_error	train_neg_mean_squared_error
0	0.131	0.000e+00	0.109	0.097	-0.170	-0.172	-0.044	-0.045
1	0.217	0.000e+00	0.139	0.160	-0.174	-0.164	-0.045	-0.042
2	0.149	0.000e+00	0.083	0.123	-0.170	-0.169	-0.043	-0.044
3	0.297	0.000e+00	0.143	0.126	-0.165	-0.168	-0.042	-0.044
4	0.141	0.000e+00	0.087	0.099	-0.175	-0.169	-0.047	-0.045
5	0.220	0.000e+00	0.079	0.104	-0.169	-0.173	-0.043	-0.045
6	0.203	0.000e+00	0.115	0.106	-0.177	-0.170	-0.047	-0.044
7	0.148	9.973e-04	0.038	0.050	-0.174	-0.176	-0.049	-0.047
8	0.148	9.973e-04	0.085	0.073	-0.172	-0.174	-0.046	-0.046
9	0.178	9.975e-04	0.136	0.167	-0.163	-0.165	-0.040	-0.042

In [48]:
```
neural_net_results.mean()
```

Out[48]:
```
fit_time                         1.831e-01
score_time                       2.992e-04
test_r2                          1.014e-01
train_r2                         1.106e-01
test_neg_mean_absolute_error    -1.710e-01
train_neg_mean_absolute_error   -1.701e-01
test_neg_mean_squared_error     -4.462e-02
train_neg_mean_squared_error    -4.420e-02
dtype: float64
```

It is hard to compare the models with original data and scalled data. It is better to scale it first, then use it to build different models.

linear regression for scaled data

In [49]:
```
np.mean(cross_val_score(linear_reg, X_scaled, y_scaled,
                        cv = 10,
                        scoring = 'neg_mean_squared_error'))
```

Out[49]: -0.044942864852472376

In [50]:
```
np.mean(cross_val_score(linear_reg, X_scaled, y_scaled,
                        cv = 10,
                        scoring = 'neg_mean_absolute_error'))
```

Out[50]: -0.17361206624797412

Decision tree model for scaled data

In [51]:
```
np.mean(cross_val_score(tree_model, X_scaled, y_scaled,
                        cv = 10, scoring = 'neg_mean_squared_error'))
```

Out[51]: -0.04183141535641728

In [52]:
```
np.mean(cross_val_score(tree_model, X_scaled, y_scaled,
                        cv = 10, scoring = 'neg_mean_absolute_error'))
```

Out[52]: -0.16287545236913492

5. Which model has better performance? Is there any way to improve the model? (5 points)

Based on the results, the neural network model gives lower MAE and MSE. For linear regression, feature selection or PCA can be use to select feature first. For tree and neural networks models, since currently we randomly select parameters, the model should be improved if we optimize the model.

6. Try to optimize the tree model or neural network model (Choose one). (10 points)

In [26]:
```
parameters = {'max_depth': range(2, 20),
              'min_samples_leaf': np.linspace(0.05, 0.4, 36)}
tree_model = tree.DecisionTreeRegressor()
optmizer = GridSearchCV(tree_model, parameters,
                            scoring = "neg_mean_squared_error",
                            cv = 10,
                            return_train_score=False,
                            verbose = 0)
optmizer.fit(X_scaled, y_scaled)
```

Out[26]:
```
GridSearchCV(cv=10, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           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,
           presort=False, random_state=None, splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': range(2, 20), 'min_samples_leaf': array([0.05, 0.06, 0.07, 0.08, 0.09, 0.1 , 0.11, 0.12, 0.13, 0.14, 0.15,
       0.16, 0.17, 0.18, 0.19, 0.2 , 0.21, 0.22, 0.23, 0.24, 0.25, 0.26,
       0.27, 0.28, 0.29, 0.3 , 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37,
       0.38, 0.39, 0.4 ])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
       scoring='neg_mean_squared_error', verbose=0)
```

In [27]:
```
results = pd.DataFrame(optmizer.cv_results_)[['param_max_depth',
                       'param_min_samples_leaf', 'mean_test_score',
                       'std_test_score', 'rank_test_score']].round(4)
results.sort_values('rank_test_score').head()
```

Out[27]:

	param_max_depth	param_min_samples_leaf	mean_test_score	std_test_score	rank_test_score
505	16	0.06	-0.043	0.002	1
433	14	0.06	-0.043	0.002	2
397	13	0.06	-0.043	0.002	2
289	10	0.06	-0.043	0.002	2
541	17	0.06	-0.043	0.002	5