MS&E 226: Mini-project part 1

Recording from the Brains of Mice in Virtual Reality to Understand the Effects of Ketamine on Spatial Memory and Navigation

Erin Brown & Francis Kei Masuda

November 15, 2019

Part 1: Investigating & Exploring Your Data

Dataset Description & Motivation

Ketamine is a commonly used rapid-acting dissociative anesthetic drug that has been used in clinics since 1970. Recently however, ketamine has received significant clinical and scientific attention due to its ability to acutely treat depression a sub-anesthetic doses. This is notable because traditional first-line antidepressants take 7-8 weeks to start showing any effects, which is not helpful if a clinician is trying to treat someone acutely depressed or suicidal. The approval of ketamine to treat depression by the FDA on March 5, 2019 marks the first new treatment in almost two decades. Yet despite frequent use and much scientific and clinical attention, ketamine's mechanism of action on neurological circuitry remains poorly understood.

Ketamine, problematically, has a veritable zoo of undesirable side effects — dissociation, schizophrenic-like psychotomimetic effects, spatial navigation impairments, and memory impairments. As part of the Giocomo Laboratory, Kei Masuda (one of the project team members) was particularly interested in dissecting out how ketamine affected spatial memory and navigation. It is known that the region of the brain known as the medial entorhinal cortex formation is the part of the brain responsible for spatial memory and navigation. So, Kei decided to record the electrical activity from individual neurons in the medial entorhinal cortex of mice and compare the neural activity in the presence and absence of ketamine.

Additionally, Kei formed a hypothesis that ketamine is disrupting neural activity by acting on a specific pace-making ion-channel known as HCN1. In order to test this, he created genetically engineered mice that are missing HCN1 ion channels in the brain and then compared the neural activity of these transgenic mice in the presence and absence of ketamine.

How was the data collected?

The brain is essentially a black box for neuroscientists. Inputs are delivered to the brain via the body's sensory systems such as visual information from the eyes and motor information form the legs. Then, the brain processes the inputs and delivers an output in the form of behavior. In an ideal world for neuroscientists, you would be able to control all the inputs into the brain and record all of the outputs while recording the electrical activity from neurons.

Virtual reality (VR) is the closest we can come to that. This data set was collected by studying mice while they performed a spatial navigation task in a 1-dimension virtual reality environment. The mouse is head-fixed and on a wheel surrounded by the VR monitors that display the virtual environment. The mice are water-deprived and then trained to navigate through a 400cm unidirectional virtual linear track in search for water rewards. When mice reach the reward point on that track, they can lick to receive a water droplet. If a mouse licks, its tongue will break an infrared sensor beam which allows for the recording of when and where a mouse is licking. In each neural recording session, mice ran 300 trials through the 400cm VR hallways: the first 50 trials served as a baseline set with no modulation; trials 51-100 followed a control injection; trials 101-300 followed an intraperitoneal injection of 25mg/kg of ketamine. While the animal navigated the VR environment, Kei was able to use a high-density silicon probe to record the firing rate of neurons in the medial entorhinal cortex of mice.

By using virtual reality, we can control exactly what the mouse is experiencing in its visual environment and have the mouse perform the exact same task over and over again. We recorded 300 trials from 6788 neurons across 50 sessions in the medial entorhinal cortex in 14 mice (9 wild type mice, 5 HCN1 transgenic knockout mice).

Data Matrix Shape & Covariate Explanation

The shape of our data matrix shape is 5000 trials x 19 covariates. Each trial is one run down the 400cm long VR hallway. We took 100 trials from each of the 50 recording sessions to form 5000 rows of trials. The covariate columns a mixture of self-explanatory columns and calculated columns. The first eight covariates are relatively self-explanatory: *(1) Animal Name (2) Session Date (3) Trial Number (4) Total Number of Cells Recorded in the Session (5) Gender of Mouse (6) Genotype – Wild Type or HCN1 knockout animal (7) Weight of the animal in grams (8) Number of days the animal has been exposed to ketamine*. The rest of the covariates were calculated from various metrics: *(9) Correlation score* – how correlated is firing of the trial to the neuron's baseline firing pattern *(10) Lick Accuracy* – what percentage of licks were correctly at the reward site *(11) Lick Number* – how many times did the animal lick during the trial *(12) Average Firing Rate* of the all of the neurons recorded in a session *(13) Average Single Cell Variance* – find the variance of firing rate for each cell during a trial and average it across all cells *(14) Variance Firing Rate* – variance of average firing rates across all neurons *(15) Average Trial Speed* – the average speed of the mouse as it traversed the 400cm track (17) Variance Speed – how variable was the mouse's speed during the trial *(18) Median Cell Depth* – the median depth in the brain of all of the neurons recorded in a session *(19) Time Since Ketamine Injection* – time in seconds *(20) Ketamine Administered* – Boolean flag for whether or not the animal has been exposed to ketamine or not.

Description of Continuous Response Variable and Binary Response Variable & Data Cleaning

We choose to use 'Time Since Ketamine Injection' as our continuous response variable for the regression task. This allows us to ask the question — *given information about the neural activity and animal behavior held within our covariates, can we predict how long it has been in seconds since ketamine was administered to the animal?* We chose to use the bool 'Ketamine Administered' as our binary response variable. This allows us to ask a slightly different question — *given information about neural activity and animal behavior held within our covariates, can we classify the trial as a trial under the influence of ketamine or a normal trial*?

There were several critical preprocessing steps that are worth noting. First of all, we filtered the data differently for the regression task and the classification task. For the regression task regression task we picked 100 trials after ketamine was injected into the mouse so that we could predict. For the classification task we picked 100 trials before ketamine was administered and 100 trials after ketamine was administered. We factorized categorical variables and removed NaN rows. We removed metadata that had no bearing on the data analysis like the date the recording was conducted. We considered removing identifying columns like the mouse name, but we realized that every mouse responds to ketamine slightly differently. Thus, it was worth keeping it in the data set.

General Discussion & Data Exploration

We did general data exploration of the data before moving on the prediction. A few interesting summary statistics of covariates are as follows. The median trial average firing rate is 8.2 Hz with a range from 0 – 30.05 Hz. The median average single cell variance is 136.22 Hz with a range from 0 – 1554.4 Hz. The median average trial speed for all the trials is 20.62 cm/s with a range from 0.13 – 64.31 cm/s. If you break it up by genotype the median trial average firing rate for wild type animals is 7.94Hz and for knock out animals is 8.58 Hz. The median average single cell variance and for knock out animals is 132.54Hz. The median average trial speed for wild type animals is 20.66 cm/s and for knock out animals 20.544 cm/s.

We also generated a scatter matrix of meaningful covariates which revealed interesting relationships within the data. Notably there is an exponential correlation between average single cell variance and average firing rate across trials. This scatter matrix also reveals clusters within the time since ketamine and the correlation score plot. There seems to be a sub-cluster of trials that show a linear relationship. Similarly, there are two distinct sub-clusters of trials that have different sloped linear correlations between average single cell variance and average trial speed. See the appendix for the scatter matrix as well as other visualizations exploring the data.

Part 2: Prediction

	RMSE	Scaled RMSE	R2
Baseline, sample mean prediction	2207.2	0.1603	0.0000
OLS, full	1886.4	0.1370	0.2719
OLS, reduced	2019.6	0.1466	0.1727
OLS, augmented, variable transform, square	1863.8	0.1353	0.2796
OLS, augmented, variable transform, cube	1852.5	0.1345	0.2949
OLS augmented, variable transform, reciprocal	1913.6	0.1389	0.2775
OLS, augmented, interaction terms	**1505.6**	**0.1093**	**0.5386**
Ridge regression, full	1880.4	0.1365	0.2719
Ridge regression, augmented, interaction terms	1506.7	0.1094	0.5379
Lasso regression, full	1880.4	0.1365	0.2719
kNN, no augmentation, $k = 7$	***827.2***	***0.0591***	—
kNN, interaction terms augmentation, $k = 5$	905.7	0.0658	—

Table 1: Results for regression to time since ketamine administration

Regression For the task of regression, we considered both linear and k-nearest neighbor (KNN) models across a variety of parameters. We measured model performance via root mean squared error (RMSE) and scaled RMSE, which we define as the RMSE divided by the difference between maximum and minimum actual values of time since ketamine administration. For the linear models, we also computed R2 scores to better understand the effectiveness of each model at capturing variation in the observations. All RMSE and scaled RMSE scores are computed as the average over 10-fold cross validation on the training set, and as such represent our estimate of the test error, while R2 scores are computed on the entirety of the training set and serve only for further insight into performance on the training set. Full results are included in Table 1.

We consider as preliminary baseline a model that always predicts the sample mean, giving rise to a scaled RMSE of 0.16. We test also a simple ordinary least squares (OLS) model on all the covariates and use this as a further point of comparison. Ridge regression and Lasso regression on all the covariates both performed slightly better than the basic OLS model. Augmenting the data set with various transformations of key variables (notably, correlation score, average trial speed, and average firing rate) including squaring, cubing, and taking the reciprocal of the variable (where we added a small delta for computational stability) resulted in modest improvements in the first two cases and a worsening in the third.

We fit an OLS model on a reduced set of covariates, containing none of the categorical variables that comprise metadata of sorts that on their own could not be used for regression but that we suspect interact strongly with the other covariates. Notably, while this reduced model does outperform the baseline, it does significantly worse than any other model tested. The role of these metadata and the significance of this result will be explored more fully in the next portion of the project as we consider inference, but for now, we consider it as tentative support motivating the inclusion of multiplicative interaction terms between each metadata covariate and all continuous covariates.

Indeed, the best performing linear model was obtained by augmenting the observations by including multiplicative interaction terms comprised of metadata-type covariates animal name, gender, genotype, weight, total cell number, median cell depth, and ketamine day with all of the remaining covariates. This model achieved a scaled RMSE of 0.11 and an R2 score of 0.54. Ridge regression on this augmented dataset including interaction terms resulted in slightly worse performance.

Even this best linear model was outperformed by our KNN models, however. We tested KNN models with k ranging from 1 to 15 on the unaugmented data set as well as the dataset including multiplicative interaction terms. We found that the unaugmented model with $k = 7$ performed best, achieving a scaled RMSE of 0.06, a significant improvement upon the best linear model. However, as we move into the next part of the project, we will wish to consider the linear model over the KNN model even though it offers worse performance as the interpretability of our regression model is essential.

	0-1 Loss	**Accuracy**
Baseline, label occurrence frequency	0.5000	0.5000
Logistic regression, no augmentation, $C = 10$	0.1481	0.8519
Logistic regression, interaction terms augmentation, $C = 1000$	**0.0720**	**0.9280**
KNN, no augmentation, $k = 5$	0.0836	0.9164
KNN, interaction terms augmentation, $k = 3$	***0.0605***	***0.9395***

Table 2: Results for prediction of ketamine administration

Classification We considered logistic regression and KNN models for the task of classification, and measured performance by accuracy and zero-one loss scores. As these measures are equivalent, we will refer primarily to accuracy as it is easily intuitive. All scores are computed as the average over 10-fold cross-validation and represent our estimate of model test error. All covariates are standardized. Full results are included in Table 2.

As a baseline for our prediction performance, we consider a model that assigns the most frequent label in all cases. Since the labels have equal frequency in our dataset, this results in an accuracy of 0.5 For both KNN and logistic regression models, we considered the unaugmented dataset as well as the augmentation discussed in the regression section with multiplicative interaction terms. All models considerably outperformed the baseline, and in both cases, prediction performance was better on the augmented dataset.

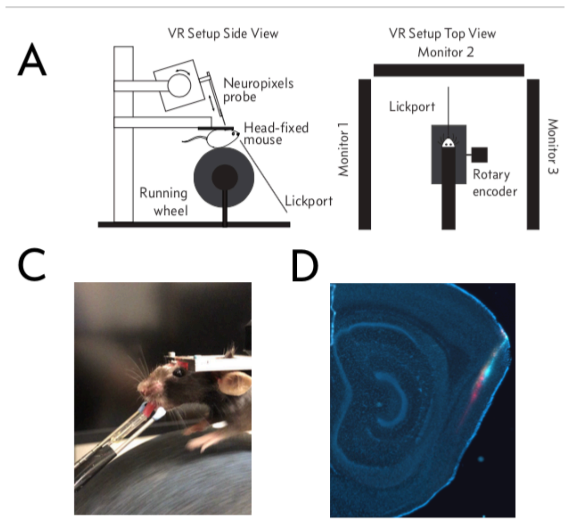
The logistic regression models used L-BFGS for the solver and incorporated an L2 penalty term C. The optimal penalty term was chosen from a set ranging from 0.01 to 10000 to minimize zero-one loss. The best logistic regression model on the augmented dataset was found to have $C = 1000$ and resulted in an accuracy of 0.93. The best KNN model ($k = 3$, chosen from a set ranging from 1 to 15) slightly outperformed the best logistic regression model on the augmented dataset and resulted in an accuracy of 0.94. Both the best KNN model and the best logistic regression model resulted in roughly equal numbers of false positives as false negatives, indicating low bias in the models.

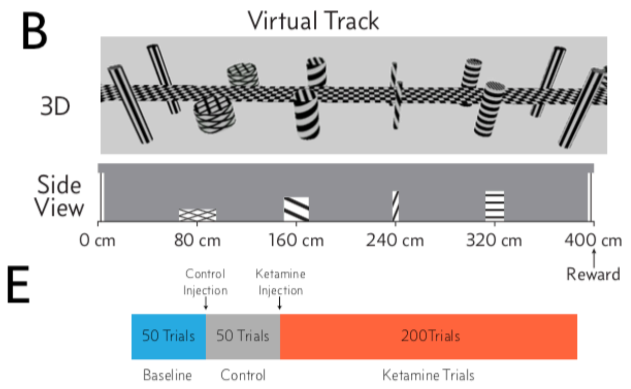
As in the regression case, while the best model performance was achieved by a KNN model, we will wish to consider logistic regression over the KNN model moving forward in order to prioritize interpretability.

Discussion Thus, far in this project, we have been pleasantly surprised by the expected performance of our best regression model to predict the time since the ketamine injection (scaled RMSE: 0.11, R2: 0.54) and the best classification model predicted whether or not the trial is under the influence of ketamine (0-1 Loss: 0.07 Accuracy 93%). There is a great deal of randomness involved in our dataset, inherent both in the noise of neural activity and in any behavioral measurements. As such, we cannot expect very low-error performance, particularly in the regression case, as to achieve as much would likely be to overfit beyond meaningfulness. With this perspective, it is neither surprising nor wholly undesirable that our R2 scores in the regression case remain low even with our best model. Considering there are 86 billion neurons in the brain the fact that we are able to predict anything from a dataset comprised of information from thousands of neurons plus animal behavior is promising.

Appendix

Figure 1: Virtual Reality & Neuropixel Probe Set up





A. Diagram of VR setup B. 400cm VR track C. Mouse licking for water reward D. Probe Track in the Medial Entorhinal Cortex in a Mouse Brain E. Trial Structure

Figure 2: Scatter Matrix of Selected Coefficients

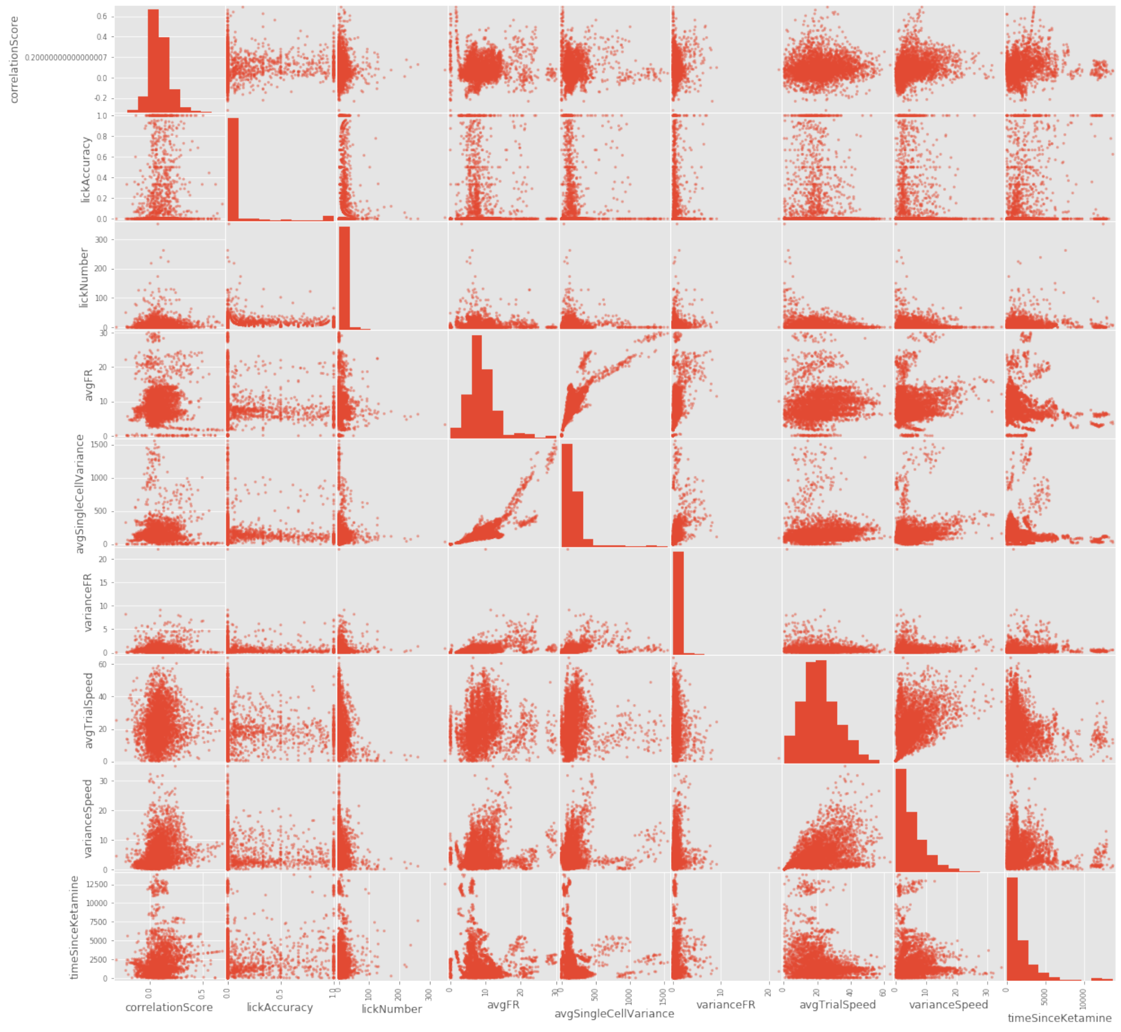
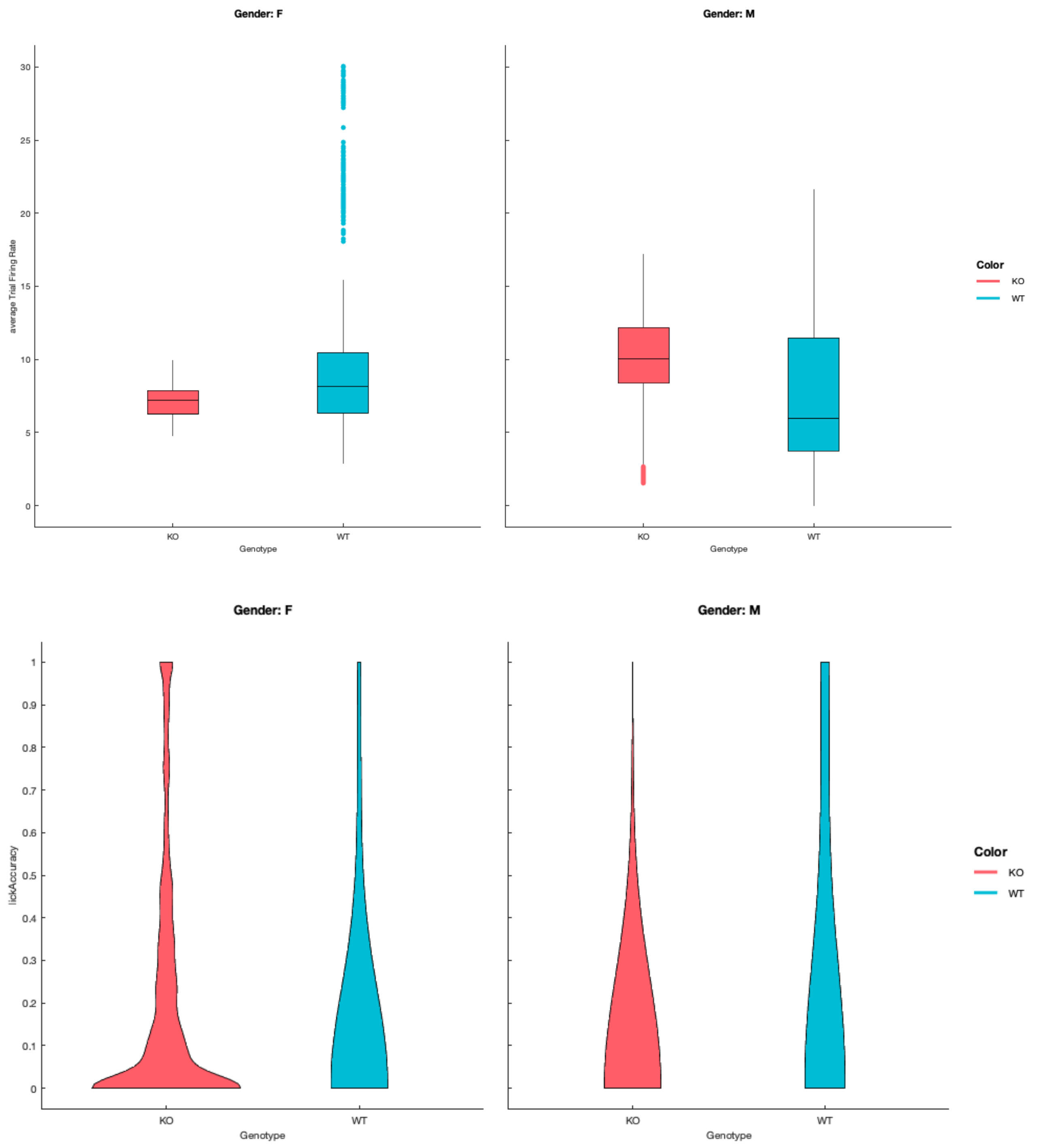


Figure 3: Data Exploration plots examinging genotype and gender on the average trial firing rates of neurons and the lick accuracy



Linear Regression

Predicting time since ketamine administration

In [1]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

Load in data and perform checks

In [2]:
```
allData = pd.read_csv('postKetamineTable.csv')
```

In [3]:
```
allData.keys()
```

Out[3]:
```
Index(['animalName', 'sessionDate', 'trialNum', 'totalCellNum', 'gender',
       'genotype', 'weight_g', 'ketamine_day', 'correlationScore',
       'lickAccuracy', 'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth',
       'timeSinceKetamine', 'ketamineAdministered'],
      dtype='object')
```

In [4]:
```
# Check size information
print("num_cols =",len(allData.keys()))
print("num_rows =",len(allData))

# Check for duplicate rows
print("num_dup =",np.sum(pd.DataFrame.duplicated(allData)))
```

```
num_cols = 19
num_rows = 5000
num_dup = 0
```

In [5]:
```
# Check for NaNs and see where they are coming from
np.sum(pd.isna(allData))
```

Out[5]:
```
animalName               0
sessionDate              0
trialNum                 0
totalCellNum             0
gender                   0
genotype                 0
weight_g                 0
ketamine_day             0
correlationScore         0
lickAccuracy             0
lickNumber               0
avgFR                    0
avgSingleCellVariance    0
varianceFR               5
avgTrialSpeed            0
varianceSpeed            0
medianCellDepth          0
timeSinceKetamine        0
ketamineAdministered     0
dtype: int64
```

In [6]:
```
# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

```
After Drop NaN
num_rows = 4995
```

In [7]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
neuralData = allDataNN[['animalName', 'totalCellNum',
       'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',
       'medianCellDepth']]
```

In [8]:
```
# Convert categorical columns
le = LabelEncoder()
neuralData_LE = neuralData.copy()
neuralData_LE['animalName'] = le.fit_transform(neuralData_LE['animalName'])
neuralData_LE['gender'] = le.fit_transform(neuralData_LE['gender'])
neuralData_LE['genotype'] = le.fit_transform(neuralData_LE['genotype'])
features = list(neuralData_LE.keys())
```

In [9]:
```
# Standardize data
stdNeuralData = StandardScaler().fit_transform(neuralData_LE)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [10]:
```
# Split off test set for later
X, X_test, y, y_test = train_test_split(stdNeuralData,timeSinceKetamine.values.ravel(), test_size=0.2)
```

In [11]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [12]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
```

In [13]:
```
print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)
```

```
Intercept:  1872.0361762406367
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 120.
11640579  -57.27030237 -382.64923316  260.46147862  547.72341522
 -368.09698592  233.04431468  320.71995695   70.65598812 -587.24827212
  250.46643903  229.81944949 -394.80643795  317.02730571 -420.76352303]
RMSE:  1882.8630710584791
R2: 0.27189568653650364
```

In [14]:
```
scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
0.1367074183913151
```

In [15]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X_train[i],y_
train[i])
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1886.442882586333
Average Scaled RMSE across Folds: 0.13696733468571948
```

Reduced model without metadata

In [17]:
```
neuralDataR = allDataNN[['correlationScore','lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',]]
featuresR = list(neuralDataR.keys())
stdNeuralDataR = StandardScaler().fit_transform(neuralDataR)

# Split off test set for later
XR, XR_test, yR, yR_test = train_test_split(stdNeuralDataR,timeSinceKetamine.values.ravel(), test_size=0.2)

# Split for cross validation, use 10 folds
num_folds = 10
XAR = np.array(XR)
yAR = np.array(yR)
XR_train = []
XR_test = []
yR_train = []
yR_test = []
kfR = KFold(n_splits=num_folds)
for train_index, test_index in kfR.split(XAR, yAR):
    XR_train.append(XAR[train_index])
    XR_test.append(XAR[test_index])
    yR_train.append(yAR[train_index])
    yR_test.append(yAR[test_index])

# Run basic linreg model on full train set, check performance against train
modelR = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(XR,yR)
print("Intercept (R): ",modelR.intercept_)
print(featuresR,modelR.coef_)
#print(model.coef_)
yR_pred = modelR.predict(XR)
rmseR = np.sqrt(mean_squared_error(yR,yR_pred))
print("RMSE (R): ",rmseR)
r2 = r2_score(yR,yR_pred)
print("R2:",r2)
```

```
Intercept (R):  1896.0893773881635
['correlationScore', 'lickAccuracy', 'lickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'va
rianceSpeed'] [ 278.37173952  365.00417952   21.00915038 -731.66554184  283.67379315
  356.13387551 -464.81832521  118.26010722]
RMSE (R):  2015.6220500164766
R2: 0.172765319389926

/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [ ]:
```

```

In [18]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(XR_train[i],y
R_train[i])
    yR_pred = model.predict(XR_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(yR_test[i],yR_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 2019.595159195902
Average Scaled RMSE across Folds: 0.14663500742731053
```

Now let's try ridge regression

In [19]:
```
# Run ridge reg model on full train set, check performance against train
model = linear_model.Ridge(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
```

In [20]:
```
print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)
```

```
Intercept:  1872.030078392652
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 119.
78256212  -57.41095397 -382.03016132  260.41130514  547.3549596
 -368.14067383  232.99851736  320.69274623   70.72204101 -586.44343355
  249.75241092  229.65397302 -394.67073218  316.78600852 -420.57499916]
RMSE:  1882.8631627270272
R2: 0.2718956156399336
```

In [21]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.Ridge(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1880.3619249713533
Average Scaled RMSE across Folds: 0.13652581983014184
```

Now let's try Lasso

In [22]:
```
# Run ridge reg model on full train set, check performance against train
model = linear_model.Lasso(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
```

In [23]:
```
print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)
```

```
Intercept:  1871.954110052363
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 110.
80067224  -55.91911744 -374.44283995  258.8543466   546.42248625
 -368.1360499   232.13346599  320.17122914   70.05053075 -580.04086679
  244.03741244  228.71535112 -393.71382111  314.94554771 -418.96136243]
RMSE:  1882.8746626469886
R2: 0.27188672156002414
```

In [24]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.Ridge(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1880.3619249713533
Average Scaled RMSE across Folds: 0.13652581983014184
```

Now let's try with some second order interaction terms

In [25]:
```
AugData = neuralData_LE.copy()
AugData.keys()
```

Out[25]:
```
Index(['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy', 'lickNumber',
       'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
       'varianceSpeed', 'medianCellDepth'],
      dtype='object')
```

In [26]:
```
primaryF = ['correlationScore', 'lickAccuracy', 'lickNumber',
       'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
       'varianceSpeed']
secondaryF = ['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
       'ketamine_day','medianCellDepth']
```

In [27]:
```
AugData['animalNamexCorrelationScore'] = AugData['animalName']*AugData['correlationScore']
AugData['animalNamexLickAccuracy'] = AugData['animalName']*AugData['lickAccuracy']
AugData['animalNamexLickNumber'] = AugData['animalName']*AugData['lickNumber']
AugData['animalNamexAvgFR'] = AugData['animalName']*AugData['avgFR']
AugData['animalNamexAvgSingleCellVariance'] = AugData['animalName']*AugData['avgSingleCellVariance']
AugData['animalNamexVarianceFR'] = AugData['animalName']*AugData['varianceFR']
AugData['animalNamexAvgTrialSpeed'] = AugData['animalName']*AugData['avgTrialSpeed']
AugData['animalNamexVarianceSpeed'] = AugData['animalName']*AugData['varianceSpeed']
```

In [28]:
```
AugData['totalCellNumxCorrelationScore'] = AugData['totalCellNum']*AugData['correlationScore']
AugData['totalCellNumxLickAccuracy'] = AugData['totalCellNum']*AugData['lickAccuracy']
AugData['totalCellNumxLickNumber'] = AugData['totalCellNum']*AugData['lickNumber']
AugData['totalCellNumxAvgFR'] = AugData['totalCellNum']*AugData['avgFR']
AugData['totalCellNumxAvgSingleCellVariance'] = AugData['totalCellNum']*AugData['avgSingleCellVariance']
AugData['totalCellNumxVarianceFR'] = AugData['totalCellNum']*AugData['varianceFR']
AugData['totalCellNumxAvgTrialSpeed'] = AugData['totalCellNum']*AugData['avgTrialSpeed']
AugData['totalCellNumxVarianceSpeed'] = AugData['totalCellNum']*AugData['varianceSpeed']
```

In [29]:
```
AugData['genderxCorrelationScore'] = AugData['gender']*AugData['correlationScore']
AugData['genderxLickAccuracy'] = AugData['gender']*AugData['lickAccuracy']
AugData['genderxLickNumber'] = AugData['gender']*AugData['lickNumber']
AugData['genderxAvgFR'] = AugData['gender']*AugData['avgFR']
AugData['genderxAvgSingleCellVariance'] = AugData['gender']*AugData['avgSingleCellVariance']
AugData['genderxVarianceFR'] = AugData['gender']*AugData['varianceFR']
AugData['genderxAvgTrialSpeed'] = AugData['gender']*AugData['avgTrialSpeed']
AugData['genderxVarianceSpeed'] = AugData['gender']*AugData['varianceSpeed']
```

In [30]:
```
AugData['genotypexCorrelationScore'] = AugData['genotype']*AugData['correlationScore']
AugData['genotypexLickAccuracy'] = AugData['genotype']*AugData['lickAccuracy']
AugData['genotypexLickNumber'] = AugData['genotype']*AugData['lickNumber']
AugData['genotypexAvgFR'] = AugData['genotype']*AugData['avgFR']
AugData['genotypexAvgSingleCellVariance'] = AugData['genotype']*AugData['avgSingleCellVariance']
AugData['genotypexVarianceFR'] = AugData['genotype']*AugData['varianceFR']
AugData['genotypexAvgTrialSpeed'] = AugData['genotype']*AugData['avgTrialSpeed']
AugData['genotypexVarianceSpeed'] = AugData['genotype']*AugData['varianceSpeed']
```

In [31]:
```
AugData['weight_gxCorrelationScore'] = AugData['weight_g']*AugData['correlationScore']
AugData['weight_gxLickAccuracy'] = AugData['weight_g']*AugData['lickAccuracy']
AugData['weight_gxLickNumber'] = AugData['weight_g']*AugData['lickNumber']
AugData['weight_gxAvgFR'] = AugData['weight_g']*AugData['avgFR']
AugData['weight_gxAvgSingleCellVariance'] = AugData['weight_g']*AugData['avgSingleCellVariance']
AugData['weight_gxVarianceFR'] = AugData['weight_g']*AugData['varianceFR']
AugData['weight_gxAvgTrialSpeed'] = AugData['weight_g']*AugData['avgTrialSpeed']
AugData['weight_gxVarianceSpeed'] = AugData['weight_g']*AugData['varianceSpeed']
```

In [32]:
```
AugData['ketamine_dayxCorrelationScore'] = AugData['ketamine_day']*AugData['correlationScore']
AugData['ketamine_dayxLickAccuracy'] = AugData['ketamine_day']*AugData['lickAccuracy']
AugData['ketamine_dayxLickNumber'] = AugData['ketamine_day']*AugData['lickNumber']
AugData['ketamine_dayxAvgFR'] = AugData['ketamine_day']*AugData['avgFR']
AugData['ketamine_dayxAvgSingleCellVariance'] = AugData['ketamine_day']*AugData['avgSingleCellVariance']
AugData['ketamine_dayxVarianceFR'] = AugData['ketamine_day']*AugData['varianceFR']
AugData['ketamine_dayxAvgTrialSpeed'] = AugData['ketamine_day']*AugData['avgTrialSpeed']
AugData['ketamine_dayxVarianceSpeed'] = AugData['ketamine_day']*AugData['varianceSpeed']
```

In [33]:
```
AugData['medianCellDepthxCorrelationScore'] = AugData['medianCellDepth']*AugData['correlationScore']
AugData['medianCellDepthxLickAccuracy'] = AugData['medianCellDepth']*AugData['lickAccuracy']
AugData['medianCellDepthxLickNumber'] = AugData['medianCellDepth']*AugData['lickNumber']
AugData['medianCellDepthxAvgFR'] = AugData['medianCellDepth']*AugData['avgFR']
AugData['medianCellDepthxAvgSingleCellVariance'] = AugData['medianCellDepth']*AugData['avgSingleCellVariance']
AugData['medianCellDepthxVarianceFR'] = AugData['medianCellDepth']*AugData['varianceFR']
AugData['medianCellDepthxAvgTrialSpeed'] = AugData['medianCellDepth']*AugData['avgTrialSpeed']
AugData['medianCellDepthxVarianceSpeed'] = AugData['medianCellDepth']*AugData['varianceSpeed']
```

In [34]:
```
# Standardize data
stdNeuralDataAug = StandardScaler().fit_transform(AugData)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [35]:
```
# Split off test set for later
X, X_test, y, y_test = train_test_split(stdNeuralDataAug,timeSinceKetamine.values.ravel(), test_size=0.2)
```

In [36]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [37]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)

print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  1886.4426242624386
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 1.37
114374e+03 -9.22880395e+02 -2.16761491e+03  6.81732307e+02
  6.94932421e+02 -2.00138064e+03 -7.41255090e+02  9.98187612e+02
  7.73511118e+01 -3.49017908e+03  1.08537237e+03  1.56594626e+03
 -2.02352013e+03  3.38116018e+01 -1.60264196e+03  2.82006203e+02
  2.77357710e+02 -7.95134032e+01 -1.16562069e+03  3.33957317e+02
  2.50406452e+02 -1.02226721e+03 -6.12920023e+02  4.49963248e+02
 -4.48566775e+02 -3.31209912e+02  3.46498599e+02  2.94644264e+02
  2.25828349e+00  4.36554638e+02  5.16016461e+01 -9.96179326e+02
  2.53999519e+02  4.79809858e+02 -1.70155455e+03  3.54354736e+03
 -6.24276994e+02  1.41410517e+03 -3.16568222e+02  3.16164003e+00
  2.24124030e+02  1.16736095e+02 -2.02186504e+03  2.66185980e+03
 -7.22050274e+02  1.58979282e+02 -3.64410817e+02  1.00338348e+03
 -5.64145751e+02 -2.67100012e+02  2.24078613e+03 -5.79885785e+03
  7.49504587e+02  8.13944545e+02  1.16465179e+03  2.72286725e+02
 -2.19418980e+02  5.14814870e-01  2.53760792e+03 -4.80368714e+02
 -6.48964355e+02  9.06101556e+02 -5.64913063e+02 -7.84416456e+01
 -2.60890014e+02  3.43760092e+02  1.31253637e+03  1.95389876e+03
 -6.71220807e+02  1.39763531e+02  3.94284963e+02]
RMSE:  1507.5199246241216
R2: 0.5385964411869812
0.10945520162173956
```

In [38]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1505.5534542864068
Average Scaled RMSE across Folds: 0.10931242380250018
```

Try Ridge reg on the augmented data set

In [39]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.Ridge(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  1885.9692921057788
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 1.04
366004e+03 -8.22468384e+02 -2.08658523e+03  6.12975416e+02
  9.46265334e+02 -1.92337202e+03 -7.10892211e+02  9.69052141e+02
  6.23910437e+01 -2.38972100e+03 -1.53221962e+01  1.44896693e+03
 -1.96195205e+03  1.21175989e+02 -1.59739770e+03  2.73953761e+02
  2.66364744e+02 -7.42714112e+01 -7.22889290e+02  3.21529029e+02
  2.24748624e+02 -9.46583248e+02 -5.88165344e+02  4.32428193e+02
 -4.45393861e+02 -3.37463559e+02  2.48281710e+02  2.93975600e+02
  7.13368365e+00  4.19712852e+02  5.27167173e+01 -9.95114365e+02
  2.58754056e+02  4.76395068e+02 -1.43316063e+03  3.12156043e+03
 -6.10730834e+02  1.35823080e+03 -2.86308944e+02 -7.94780067e-01
  2.21747066e+02  1.22102917e+02 -1.86443366e+03  2.59755069e+03
 -7.15259483e+02  1.51032959e+02 -3.45183074e+02  1.01640809e+03
 -5.27188392e+02 -2.59250573e+02  7.81767377e+02 -4.49419230e+03
  7.84014835e+02  7.34234760e+02  1.06284448e+03  2.50629679e+02
 -2.12112836e+02  1.30664538e+01  2.34969558e+03 -4.49724033e+02
 -5.81475559e+02  9.25947211e+02 -5.90494330e+02 -9.19729103e+01
 -2.67249426e+02  3.35043431e+02  1.42053060e+03  1.77402819e+03
 -6.41475428e+02  1.24623206e+02  3.94120028e+02]
RMSE:  1508.68442471356
R2: 0.5378833335191369
0.10953975147743039
```

In [40]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.Ridge(alpha=1.0,fit_intercept=True,normalize=False,copy_X=True).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1506.7402041759663
Average Scaled RMSE across Folds: 0.10939858913027765
```

Let's try a different augmentation

In [41]:
```
AugData2 = neuralData_LE.copy()
AugData2.keys()
```

Out[41]:
```
Index(['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy', 'lickNumber',
       'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
       'varianceSpeed', 'medianCellDepth'],
      dtype='object')
```

In [42]:
```
AugData2['corrScoreSq'] = AugData2['correlationScore']*AugData2['correlationScore']
AugData2['avgTrialSpeedSq'] = AugData2['avgTrialSpeed']*AugData2['avgTrialSpeed']
AugData2['avgFRSQ'] = AugData2['avgFR']*AugData2['avgFR']
```

In [43]:
```
# Standardize data
stdNeuralDataAug2 = StandardScaler().fit_transform(AugData2)
# Split off test set for later
X, X_test, y, y_test = train_test_split(stdNeuralDataAug2,timeSinceKetamine.values.ravel(), test_size=0.2)

# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [44]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)

print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  1888.1720751052355
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [   -
8.4902276    -19.45701449  -387.48146005   140.3208147
   600.39566312  -317.84486377   446.11577529   287.33205658
    36.55730237 -1318.87722509   -47.28339046   175.56657687
  -657.38999423   324.11341399  -434.12796164  -255.71402409
   299.65632484  1052.76693917]
RMSE:  1870.780049294536
R2: 0.279570492829024
0.135837020651291
```

In [45]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1863.797827017103
Average Scaled RMSE across Folds: 0.1353300426812997
```

Another augmentation yet

In [46]:
```
AugData3 = neuralData_LE.copy()

AugData3['corrScoreCu'] = AugData3['correlationScore']*AugData3['correlationScore']*AugData3['correlationScore']
AugData3['avgTrialSpeedCu'] = AugData3['avgTrialSpeed']*AugData3['avgTrialSpeed']*AugData3['avgTrialSpeed']
AugData3['avgFRCu'] = AugData3['avgFR']*AugData3['avgFR']*AugData3['avgFR']
```

In [47]:
```
# Standardize data
stdNeuralDataAug3 = StandardScaler().fit_transform(AugData3)
# Split off test set for later
X, X_test, y, y_test = train_test_split(stdNeuralDataAug3,timeSinceKetamine.values.ravel(), test_size=0.2)

# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [48]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)

print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  1885.1795660508608
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [-109.
64269421    5.56949394 -297.16054155  147.03517754  633.02484052
 -320.68463625  383.75356065  306.21282035   59.10936839 -885.17174324
 -124.44471122  148.60767264 -573.13065192  334.22873075 -435.51857752
 -212.57115878  234.36199972  776.5404991 ]
RMSE:  1856.8250464437979
R2: 0.29486632278414426
0.13481689794944274
```

In [49]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1852.5491690454987
Average Scaled RMSE across Folds: 0.1345064429994975
```

One more augmentation

In [50]:
```
AugData4 = neuralData_LE.copy()

AugData4['corrScoreInv'] = 1/(1+AugData4['correlationScore'])
AugData4['avgTrialSpeedInv'] = 1/(1+AugData4['avgTrialSpeed'])
AugData4['avgFRInv'] = 1/(1+AugData4['avgFR'])

# Standardize data
stdNeuralDataAug4 = StandardScaler().fit_transform(AugData4)
# Split off test set for later
X, X_test, y, y_test = train_test_split(AugData4,timeSinceKetamine.values.ravel(), test_size=0.2)

# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [51]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)

print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  3723.2940578811526
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 4.49
896054e+01 -2.34591455e+00 -7.26028374e+02  6.13177274e+02
  1.88335768e+02 -2.12580419e+02  1.53203865e+03  1.21257508e+03
  1.53685814e+00 -1.96863846e+02  2.33661963e+00  1.33560998e+02
 -3.69485406e+01  6.52247747e+01 -9.98465322e-01 -2.23107742e+03
  8.34356473e+02 -2.46653535e+03]
RMSE:  1917.019637370434
R2: 0.2774994015017411
0.13918739480244843
```

In [52]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1913.6267263538862
Average Scaled RMSE across Folds: 0.13894104863260032
```

In [ ]:

In [ ]:

In [ ]:

Miniproject Classification - Logistic Regression

In [29]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

Load in data

In [2]:
```
allData = pd.read_csv('sessionTrialTable.csv')
```

In [3]:
```
allData.keys()
```

Out[3]:
```
Index(['animalName', 'sessionDate', 'trialNum', 'totalCellNum', 'gender',
       'genotype', 'weight_g', 'ketamine_day', 'correlationScore',
       'lickAccuracy', 'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth',
       'timeSinceKetamine', 'ketamineAdministered'],
      dtype='object')
```

In [4]:
```
# Check size information
print("num_cols =",len(allData.keys()))
print("num_rows =",len(allData))

# Check for duplicate rows
print("num_dup =",np.sum(pd.DataFrame.duplicated(allData)))
```

```
num_cols = 19
num_rows = 5000
num_dup = 0
```

In [5]:
```
# Check for NaNs and see where they are coming from
np.sum(pd.isna(allData))
```

Out[5]:
```
animalName               0
sessionDate              0
trialNum                 0
totalCellNum             0
gender                   0
genotype                 0
weight_g                 0
ketamine_day             0
correlationScore         0
lickAccuracy             0
lickNumber               0
avgFR                    1
avgSingleCellVariance    1
varianceFR               3
avgTrialSpeed            0
varianceSpeed            0
medianCellDepth          0
timeSinceKetamine        0
ketamineAdministered     0
dtype: int64
```

In [6]:
```
# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

```
After Drop NaN
num_rows = 4997
```

Logistic Regression Classifier

In [7]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
neuralData = allDataNN[['animalName', 'totalCellNum',
       'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',
       'medianCellDepth']]
```

In [8]:
```
# Convert categorical columns
le = LabelEncoder()
neuralData_LE = neuralData.copy()
neuralData_LE['animalName'] = le.fit_transform(neuralData_LE['animalName'])
neuralData_LE['gender'] = le.fit_transform(neuralData_LE['gender'])
neuralData_LE['genotype'] = le.fit_transform(neuralData_LE['genotype'])
```

In [9]:
```
X, X_test, y, y_test = train_test_split(neuralData_LE,ketBool.values.ravel(), test_size=0.2)
# Now we save X_test and y_test for next part of the project!
```

In [10]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
skf = StratifiedKFold(n_splits=num_folds)
for train_index, test_index in skf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [11]:
```
# Run basic log reg model on train set, check performance against train
# Run model, tuning over C_param (l2 penalty by default)
for C_param in [0.01, 0.1, 1, 10, 100, 1000, 5000, 10000, 50000]:
    model = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000,C = C_param).fit(X, y)
    y_pred = model.predict(X)
    print(zero_one_loss(y, y_pred))
# Let's use C = 100
```

```
0.1848886664998749
0.16562421816362272
0.15186389792344257
0.14836127095321494
0.14535901926444839
0.14786089567175387
0.14961220915686768
0.14660995746810113
0.14660995746810113
```

Best loss was at C=10000, so let's use that! Note that the resulting loss score was 0.1411.

```
In [12]: C_param = 10000
         zo_loss = []
         accuracy = []
         num_folds=10
         for i in range(0,num_folds):
             LRmodel = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000, C = C_param).fit(X_train[i], y_train[i])
             y_pred = model.predict(X_test[i])
             zo_loss.append(zero_one_loss(y_test[i],y_pred))
             accuracy.append(accuracy_score(y_test[i],y_pred))

         avg_zo_loss = np.mean(zo_loss)
         avg_acc = np.mean(accuracy)
         print("Average zero-one loss across folds:",avg_zo_loss)
         print("Average accuracy across folds:",avg_acc)
```

```
Average zero-one loss across folds: 0.14660081313008205
Average accuracy across folds: 0.8533991868699179
```

Average zero-one loss across 10-fold CV was 0.1466, only very slightly worse than our overall training error.

Now try with standardization

```
In [13]: stdNeuralData = StandardScaler().fit_transform(neuralData_LE)
         X, X_test, y, y_test = train_test_split(stdNeuralData,ketBool.values.ravel(), test_size=0.2)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

```
In [14]: # Split for cross validation, use 10 folds
         num_folds = 10
         XA = np.array(X)
         yA = np.array(y)
         X_train = []
         X_test = []
         y_train = []
         y_test = []
         skf = StratifiedKFold(n_splits=num_folds)
         for train_index, test_index in skf.split(XA, yA):
             X_train.append(XA[train_index])
             X_test.append(XA[test_index])
             y_train.append(yA[train_index])
             y_test.append(yA[test_index])
```

```
In [15]: # Run basic log reg model on train set, check performance against train
         # Run model, tuning over C_param (L2 penalty by default)
         for C_param in [0.01, 0.1, 1, 10, 100, 1000]:
             model = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000,C = C_param).fit(X, y)
             y_pred = model.predict(X)
             print(zero_one_loss(y, y_pred))
```

```
0.1553665248936702
0.15061295971978983
0.14836127095321494
0.14786089567175387
0.14811108331248435
0.14811108331248435
```

Best loss was at C=10, so let's use that! Note that the resulting loss score was 0.1471.

In [16]:
```
C_param = 10
zo_loss = []
accuracy = []
num_folds=10
for i in range(0,num_folds):
    LRmodel = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000, C = C_param).fit(X_train[i], y_train[i])
    y_pred = model.predict(X_test[i])
    zo_loss.append(zero_one_loss(y_test[i],y_pred))
    accuracy.append(accuracy_score(y_test[i],y_pred))

avg_zo_loss = np.mean(zo_loss)
avg_acc = np.mean(accuracy)
print("Average zero-one loss across folds:",avg_zo_loss)
print("Average accuracy across folds:",avg_acc)
```

```
Average zero-one loss across folds: 0.1481146616541353
Average accuracy across folds: 0.8518853383458647
```

Average zero-one loss across 10-fold CV was 0.1471, the same within significant digits as our overall training error.

In [17]:
```
# TEST SET CHECKS - FOR LATER
# Make predictions# Let's use C = 100
#y_pred = model.predict(X_test)
# Keep prediction probability, not using for now, consider adding a threshold
#y_pred_proba = model.predict_proba(X_test)
#print(zero_one_loss(y_test, y_pred))
```

Now with augmentation!

In [18]:
```
AugData = neuralData_LE.copy()
```

In [19]:
```
AugData['animalNamexCorrelationScore'] = AugData['animalName']*AugData['correlationScore']
AugData['animalNamexLickAccuracy'] = AugData['animalName']*AugData['lickAccuracy']
AugData['animalNamexLickNumber'] = AugData['animalName']*AugData['lickNumber']
AugData['animalNamexAvgFR'] = AugData['animalName']*AugData['avgFR']
AugData['animalNamexAvgSingleCellVariance'] = AugData['animalName']*AugData['avgSingleCellVariance']
AugData['animalNamexVarianceFR'] = AugData['animalName']*AugData['varianceFR']
AugData['animalNamexAvgTrialSpeed'] = AugData['animalName']*AugData['avgTrialSpeed']
AugData['animalNamexVarianceSpeed'] = AugData['animalName']*AugData['varianceSpeed']

AugData['totalCellNumxCorrelationScore'] = AugData['totalCellNum']*AugData['correlationScore']
AugData['totalCellNumxLickAccuracy'] = AugData['totalCellNum']*AugData['lickAccuracy']
AugData['totalCellNumxLickNumber'] = AugData['totalCellNum']*AugData['lickNumber']
AugData['totalCellNumxAvgFR'] = AugData['totalCellNum']*AugData['avgFR']
AugData['totalCellNumxAvgSingleCellVariance'] = AugData['totalCellNum']*AugData['avgSingleCellVariance']
AugData['totalCellNumxVarianceFR'] = AugData['totalCellNum']*AugData['varianceFR']
AugData['totalCellNumxAvgTrialSpeed'] = AugData['totalCellNum']*AugData['avgTrialSpeed']
AugData['totalCellNumxVarianceSpeed'] = AugData['totalCellNum']*AugData['varianceSpeed']

AugData['genderxCorrelationScore'] = AugData['gender']*AugData['correlationScore']
AugData['genderxLickAccuracy'] = AugData['gender']*AugData['lickAccuracy']
AugData['genderxLickNumber'] = AugData['gender']*AugData['lickNumber']
AugData['genderxAvgFR'] = AugData['gender']*AugData['avgFR']
AugData['genderxAvgSingleCellVariance'] = AugData['gender']*AugData['avgSingleCellVariance']
AugData['genderxVarianceFR'] = AugData['gender']*AugData['varianceFR']
AugData['genderxAvgTrialSpeed'] = AugData['gender']*AugData['avgTrialSpeed']
AugData['genderxVarianceSpeed'] = AugData['gender']*AugData['varianceSpeed']

AugData['genotypexCorrelationScore'] = AugData['genotype']*AugData['correlationScore']
AugData['genotypexLickAccuracy'] = AugData['genotype']*AugData['lickAccuracy']
AugData['genotypexLickNumber'] = AugData['genotype']*AugData['lickNumber']
AugData['genotypexAvgFR'] = AugData['genotype']*AugData['avgFR']
AugData['genotypexAvgSingleCellVariance'] = AugData['genotype']*AugData['avgSingleCellVariance']
AugData['genotypexVarianceFR'] = AugData['genotype']*AugData['varianceFR']
AugData['genotypexAvgTrialSpeed'] = AugData['genotype']*AugData['avgTrialSpeed']
AugData['genotypexVarianceSpeed'] = AugData['genotype']*AugData['varianceSpeed']

AugData['weight_gxCorrelationScore'] = AugData['weight_g']*AugData['correlationScore']
AugData['weight_gxLickAccuracy'] = AugData['weight_g']*AugData['lickAccuracy']
AugData['weight_gxLickNumber'] = AugData['weight_g']*AugData['lickNumber']
AugData['weight_gxAvgFR'] = AugData['weight_g']*AugData['avgFR']
AugData['weight_gxAvgSingleCellVariance'] = AugData['weight_g']*AugData['avgSingleCellVariance']
AugData['weight_gxVarianceFR'] = AugData['weight_g']*AugData['varianceFR']
AugData['weight_gxAvgTrialSpeed'] = AugData['weight_g']*AugData['avgTrialSpeed']
AugData['weight_gxVarianceSpeed'] = AugData['weight_g']*AugData['varianceSpeed']

AugData['ketamine_dayxCorrelationScore'] = AugData['ketamine_day']*AugData['correlationScore']
AugData['ketamine_dayxLickAccuracy'] = AugData['ketamine_day']*AugData['lickAccuracy']
AugData['ketamine_dayxLickNumber'] = AugData['ketamine_day']*AugData['lickNumber']
AugData['ketamine_dayxAvgFR'] = AugData['ketamine_day']*AugData['avgFR']
AugData['ketamine_dayxAvgSingleCellVariance'] = AugData['ketamine_day']*AugData['avgSingleCellVariance']
AugData['ketamine_dayxVarianceFR'] = AugData['ketamine_day']*AugData['varianceFR']
AugData['ketamine_dayxAvgTrialSpeed'] = AugData['ketamine_day']*AugData['avgTrialSpeed']
AugData['ketamine_dayxVarianceSpeed'] = AugData['ketamine_day']*AugData['varianceSpeed']

AugData['medianCellDepthxCorrelationScore'] = AugData['medianCellDepth']*AugData['correlationScore']
AugData['medianCellDepthxLickAccuracy'] = AugData['medianCellDepth']*AugData['lickAccuracy']
AugData['medianCellDepthxLickNumber'] = AugData['medianCellDepth']*AugData['lickNumber']
AugData['medianCellDepthxAvgFR'] = AugData['medianCellDepth']*AugData['avgFR']
AugData['medianCellDepthxAvgSingleCellVariance'] = AugData['medianCellDepth']*AugData['avgSingleCellVariance']
AugData['medianCellDepthxVarianceFR'] = AugData['medianCellDepth']*AugData['varianceFR']
AugData['medianCellDepthxAvgTrialSpeed'] = AugData['medianCellDepth']*AugData['avgTrialSpeed']
AugData['medianCellDepthxVarianceSpeed'] = AugData['medianCellDepth']*AugData['varianceSpeed']
```

In [20]:
```
# Standardize data
stdNeuralDataAug = StandardScaler().fit_transform(AugData)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [21]:
```
X, X_test, y, y_test = train_test_split(stdNeuralDataAug,ketBool.values.ravel(), test_size=0.2)
```

In [22]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
skf = StratifiedKFold(n_splits=num_folds)
for train_index, test_index in skf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [24]:
```
# Run basic log reg model on train set, check performance against train
# Run model, tuning over C_param (l2 penalty by default)
for C_param in [0.01, 0.1, 1, 10, 100, 1000, 10000]:
    model = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000,C = C_param).fit(X, y)
    y_pred = model.predict(X)
    print(zero_one_loss(y, y_pred))
```

```
0.12109081811358524
0.09582186639979984
0.08006004503377528
0.07305479109332003
0.07305479109332003
0.07230422817112836
0.07205404053039777
```

In [25]:
```
C_param = 1000
zo_loss = []
accuracy = []
num_folds=10
for i in range(0,num_folds):
    LRmodel = linear_model.LogisticRegression(solver='lbfgs',max_iter=10000, C = C_param).fit(X_train[i], y_train[i])
    y_pred = model.predict(X_test[i])
    zo_loss.append(zero_one_loss(y_test[i],y_pred))
    accuracy.append(accuracy_score(y_test[i],y_pred))

avg_zo_loss = np.mean(zo_loss)
avg_acc = np.mean(accuracy)
print("Average zero-one loss across folds:",avg_zo_loss)
print("Average accuracy across folds:",avg_acc)
```

```
Average zero-one loss across folds: 0.07204581176559488
Average accuracy across folds: 0.927954188234405
```

In [26]:
```
model = linear_model.LogisticRegression(solver='lbfgs',max_iter=1000,C = 10000).fit(X, y)
y_pred = model.predict(X)
```

In [27]:
```
zero_one_loss(y,y_pred)
```

Out[27]:
```
0.07205404053039777
```

In [28]:
```
accuracy_score(y,y_pred)
```

Out[28]:
```
0.9279459594696022
```

In [37]:
```
cm = confusion_matrix(y,y_pred)
```

In [40]:
```
tn = cm[0,0]
fn = cm[1,0]
tp = cm[1,1]
fp = cm[0,1]
```

In [41]:
```
tn
```

Out[41]:
```
1832
```

In [42]:
```
fn
```

Out[42]:
```
152
```

In [43]:
```
tp
```

Out[43]:
```
1877
```

```
In [44]: fp

Out[44]: 136

In [47]: cm

Out[47]: array([[1832,  136],
                [ 152, 1877]])

In [50]: 1832+152+136+1877

Out[50]: 3997

In [ ]:
```

k-Nearest Neighbors for REGRESSION

In [35]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

In [36]:
```
allData = pd.read_csv('postKetamineTable.csv')
```

In [37]:
```
# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

```
After Drop NaN
num_rows = 4995
```

In [38]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
neuralData = allDataNN[['animalName', 'totalCellNum',
       'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',
       'medianCellDepth']]
```

In [39]:
```
# Convert categorical columns
le = LabelEncoder()
neuralData_LE = neuralData.copy()
neuralData_LE['animalName'] = le.fit_transform(neuralData_LE['animalName'])
neuralData_LE['gender'] = le.fit_transform(neuralData_LE['gender'])
neuralData_LE['genotype'] = le.fit_transform(neuralData_LE['genotype'])
features = list(neuralData_LE.keys())
```

In [40]:
```
# Standardize data
stdNeuralData = StandardScaler().fit_transform(neuralData_LE)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [41]:
```
# Split off test set for later
X, X_test, y, y_test = train_test_split(stdNeuralData,timeSinceKetamine.values.ravel(), test_size=0.2)
```

```
In [42]: # Split for cross validation, use 10 folds
         num_folds = 10
         XA = np.array(X)
         yA = np.array(y)
         X_train = []
         X_test = []
         y_train = []
         y_test = []
         kf = KFold(n_splits=num_folds)
         for train_index, test_index in kf.split(XA, yA):
             X_train.append(XA[train_index])
             X_test.append(XA[test_index])
             y_train.append(yA[train_index])
             y_test.append(yA[test_index])
```

```
In [43]: rmse_cv = []
         for i in range(0,num_folds):
             knn = KNeighborsRegressor(n_neighbors=5, metric='euclidean').fit(X_train[i],y_train[i])
             y_pred = knn.predict(X_test[i])
             rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))

         print("Average RMSE across Folds:",np.mean(rmse_cv))
         print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))

         Average RMSE across Folds: 853.4716346360168
         Average Scaled RMSE across Folds: 0.06197069996442225
```

```
In [44]: avg_rmse = []
         for k in [1,3,5,6,7,8,10,15,20]:
             rmse = []
             for i in range(0,num_folds):
                 knn = KNeighborsRegressor(n_neighbors=k, metric='euclidean').fit(X_train[i],y_train[i])
                 y_pred = knn.predict(X_test[i])
                 rmse.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
             avg_rmse.append(np.mean(rmse))
         avg_scaled_rmse = avg_rmse/(max(y)-min(y))
```

```
In [45]: avg_rmse

Out[45]: [924.0082872225673,
          857.9001586261759,
          853.4716346360168,
          841.9998230884452,
          827.2383878734227,
          841.4674660909595,
          851.3229975479755,
          925.9300390632383,
          989.6543402804988]
```

```
In [24]: avg_scaled_rmse

Out[24]: array([0.06777673, 0.06008589, 0.06001622, 0.0598738 , 0.0591028 ,
                0.05978212, 0.06073132, 0.0668751 , 0.07089633])
```

```
In [46]: min(avg_rmse)

Out[46]: 827.2383878734227
```

Best Avg Scaled RMSE is with k = 7

Average RMSE: 827.2383878734227

Average Scaled RMSE: 0.0591028

Now what if we augment? does that help?

```
In [48]: AugData = neuralData_LE.copy()
         AugData.keys()

Out[48]: Index(['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
                'ketamine_day', 'correlationScore', 'lickAccuracy', 'lickNumber',
                'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
                'varianceSpeed', 'medianCellDepth'],
               dtype='object')
```

In [49]:
```
AugData['animalNamexCorrelationScore'] = AugData['animalName']*AugData['correlationScore']
AugData['animalNamexLickAccuracy'] = AugData['animalName']*AugData['lickAccuracy']
AugData['animalNamexLickNumber'] = AugData['animalName']*AugData['lickNumber']
AugData['animalNamexAvgFR'] = AugData['animalName']*AugData['avgFR']
AugData['animalNamexAvgSingleCellVariance'] = AugData['animalName']*AugData['avgSingleCellVariance']
AugData['animalNamexVarianceFR'] = AugData['animalName']*AugData['varianceFR']
AugData['animalNamexAvgTrialSpeed'] = AugData['animalName']*AugData['avgTrialSpeed']
AugData['animalNamexVarianceSpeed'] = AugData['animalName']*AugData['varianceSpeed']

AugData['totalCellNumxCorrelationScore'] = AugData['totalCellNum']*AugData['correlationScore']
AugData['totalCellNumxLickAccuracy'] = AugData['totalCellNum']*AugData['lickAccuracy']
AugData['totalCellNumxLickNumber'] = AugData['totalCellNum']*AugData['lickNumber']
AugData['totalCellNumxAvgFR'] = AugData['totalCellNum']*AugData['avgFR']
AugData['totalCellNumxAvgSingleCellVariance'] = AugData['totalCellNum']*AugData['avgSingleCellVariance']
AugData['totalCellNumxVarianceFR'] = AugData['totalCellNum']*AugData['varianceFR']
AugData['totalCellNumxAvgTrialSpeed'] = AugData['totalCellNum']*AugData['avgTrialSpeed']
AugData['totalCellNumxVarianceSpeed'] = AugData['totalCellNum']*AugData['varianceSpeed']

AugData['genderxCorrelationScore'] = AugData['gender']*AugData['correlationScore']
AugData['genderxLickAccuracy'] = AugData['gender']*AugData['lickAccuracy']
AugData['genderxLickNumber'] = AugData['gender']*AugData['lickNumber']
AugData['genderxAvgFR'] = AugData['gender']*AugData['avgFR']
AugData['genderxAvgSingleCellVariance'] = AugData['gender']*AugData['avgSingleCellVariance']
AugData['genderxVarianceFR'] = AugData['gender']*AugData['varianceFR']
AugData['genderxAvgTrialSpeed'] = AugData['gender']*AugData['avgTrialSpeed']
AugData['genderxVarianceSpeed'] = AugData['gender']*AugData['varianceSpeed']

AugData['genotypexCorrelationScore'] = AugData['genotype']*AugData['correlationScore']
AugData['genotypexLickAccuracy'] = AugData['genotype']*AugData['lickAccuracy']
AugData['genotypexLickNumber'] = AugData['genotype']*AugData['lickNumber']
AugData['genotypexAvgFR'] = AugData['genotype']*AugData['avgFR']
AugData['genotypexAvgSingleCellVariance'] = AugData['genotype']*AugData['avgSingleCellVariance']
AugData['genotypexVarianceFR'] = AugData['genotype']*AugData['varianceFR']
AugData['genotypexAvgTrialSpeed'] = AugData['genotype']*AugData['avgTrialSpeed']
AugData['genotypexVarianceSpeed'] = AugData['genotype']*AugData['varianceSpeed']

AugData['weight_gxCorrelationScore'] = AugData['weight_g']*AugData['correlationScore']
AugData['weight_gxLickAccuracy'] = AugData['weight_g']*AugData['lickAccuracy']
AugData['weight_gxLickNumber'] = AugData['weight_g']*AugData['lickNumber']
AugData['weight_gxAvgFR'] = AugData['weight_g']*AugData['avgFR']
AugData['weight_gxAvgSingleCellVariance'] = AugData['weight_g']*AugData['avgSingleCellVariance']
AugData['weight_gxVarianceFR'] = AugData['weight_g']*AugData['varianceFR']
AugData['weight_gxAvgTrialSpeed'] = AugData['weight_g']*AugData['avgTrialSpeed']
AugData['weight_gxVarianceSpeed'] = AugData['weight_g']*AugData['varianceSpeed']

AugData['ketamine_dayxCorrelationScore'] = AugData['ketamine_day']*AugData['correlationScore']
AugData['ketamine_dayxLickAccuracy'] = AugData['ketamine_day']*AugData['lickAccuracy']
AugData['ketamine_dayxLickNumber'] = AugData['ketamine_day']*AugData['lickNumber']
AugData['ketamine_dayxAvgFR'] = AugData['ketamine_day']*AugData['avgFR']
AugData['ketamine_dayxAvgSingleCellVariance'] = AugData['ketamine_day']*AugData['avgSingleCellVariance']
AugData['ketamine_dayxVarianceFR'] = AugData['ketamine_day']*AugData['varianceFR']
AugData['ketamine_dayxAvgTrialSpeed'] = AugData['ketamine_day']*AugData['avgTrialSpeed']
AugData['ketamine_dayxVarianceSpeed'] = AugData['ketamine_day']*AugData['varianceSpeed']

AugData['medianCellDepthxCorrelationScore'] = AugData['medianCellDepth']*AugData['correlationScore']
AugData['medianCellDepthxLickAccuracy'] = AugData['medianCellDepth']*AugData['lickAccuracy']
AugData['medianCellDepthxLickNumber'] = AugData['medianCellDepth']*AugData['lickNumber']
AugData['medianCellDepthxAvgFR'] = AugData['medianCellDepth']*AugData['avgFR']
AugData['medianCellDepthxAvgSingleCellVariance'] = AugData['medianCellDepth']*AugData['avgSingleCellVariance']
AugData['medianCellDepthxVarianceFR'] = AugData['medianCellDepth']*AugData['varianceFR']
AugData['medianCellDepthxAvgTrialSpeed'] = AugData['medianCellDepth']*AugData['avgTrialSpeed']
AugData['medianCellDepthxVarianceSpeed'] = AugData['medianCellDepth']*AugData['varianceSpeed']
```

In [50]:
```
stdNeuralDataAug = StandardScaler().fit_transform(AugData)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [51]:
```
X, X_test, y, y_test = train_test_split(stdNeuralDataAug,timeSinceKetamine.values.ravel(), test_size=0.2)
```

```
In [52]: # Split for cross validation, use 10 folds
         num_folds = 10
         XA = np.array(X)
         yA = np.array(y)
         X_train = []
         X_test = []
         y_train = []
         y_test = []
         kf = KFold(n_splits=num_folds)
         for train_index, test_index in kf.split(XA, yA):
             X_train.append(XA[train_index])
             X_test.append(XA[test_index])
             y_train.append(yA[train_index])
             y_test.append(yA[test_index])
```

```
In [53]: rmse_cv = []
         for i in range(0,num_folds):
             knn = KNeighborsRegressor(n_neighbors=5, metric='euclidean').fit(X_train[i],y_train[i])
             y_pred = knn.predict(X_test[i])
             rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))

         print("Average RMSE across Folds:",np.mean(rmse_cv))
         print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 905.7573412545744
Average Scaled RMSE across Folds: 0.06576354367728128
```

```
In [54]: avg_rmse = []
         for k in [1,3,5,6,7,8,10,15,20]:
             rmse = []
             for i in range(0,num_folds):
                 knn = KNeighborsRegressor(n_neighbors=k, metric='euclidean').fit(X_train[i],y_train[i])
                 y_pred = knn.predict(X_test[i])
                 rmse.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
             avg_rmse.append(np.mean(rmse))
         avg_scaled_rmse = avg_rmse/(max(y)-min(y))
```

```
In [55]: avg_rmse
```

```
Out[55]: [990.5562528635273,
          912.405977965154,
          905.7573412545744,
          913.4123216584588,
          922.4442221364354,
          921.9807324167772,
          927.593206409659,
          979.3887627450247,
          1055.5126484171924]
```

```
In [57]: avg_scaled_rmse
```

```
Out[57]: array([0.07192047, 0.06624628, 0.06576354, 0.06631934, 0.06697511,
                0.06694146, 0.06734896, 0.07110964, 0.0766367 ])
```

Augmentation gets us to best scaled rmse of 0.064 with k = 5

Average RMSE: 905.7573412545744

Average Scaled RMSE: 0.06576354

```
In [56]: min(avg_rmse)
```

```
Out[56]: 905.7573412545744
```

```
In [ ]:
```

Miniproject: Classification with k-Nearest Neighbors

In [1]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

In [2]:
```
allData = pd.read_csv('sessionTrialTable.csv')

# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

```
After Drop NaN
num_rows = 4997
```

In [3]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
neuralData = allDataNN[['animalName', 'totalCellNum',
       'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',
       'medianCellDepth']]

# Convert categorical columns
le = LabelEncoder()
neuralData_LE = neuralData.copy()
neuralData_LE['animalName'] = le.fit_transform(neuralData_LE['animalName'])
neuralData_LE['gender'] = le.fit_transform(neuralData_LE['gender'])
neuralData_LE['genotype'] = le.fit_transform(neuralData_LE['genotype'])
```

In [4]:
```
X, X_test, y, y_test = train_test_split(neuralData_LE,ketBool.values.ravel(), test_size=0.2)
```

In [5]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
skf = StratifiedKFold(n_splits=num_folds)
for train_index, test_index in skf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [6]:
```
# Now try KNN on whole train set: NOTE data point is included in its own nearest neighbors, so this is a bit meaningles
s
# Let's start with k=5
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean').fit(X,y)
y_pred = knn.predict(X)
print(zero_one_loss(y, y_pred))
print(accuracy_score(y,y_pred))
```

```
0.058794095571678806
0.9412059044283212
```

In [7]:
```
zo_loss = []
accuracy = []
for i in range(0,num_folds):
    knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean').fit(X_train[i],y_train[i])
    y_pred = knn.predict(X_test[i])
    zo_loss.append(zero_one_loss(y_test[i],y_pred))
    accuracy.append(accuracy_score(y_test[i],y_pred))

avg_zo_loss = np.mean(zo_loss)
avg_acc = np.mean(accuracy)
print("Average zero-one loss across folds:",avg_zo_loss)
print("Average accuracy across folds:",avg_acc)
```

```
Average zero-one loss across folds: 0.08356214972103625
Average accuracy across folds: 0.9164378502789639
```

In [8]:
```
avg_zo = []
avg_ac = []
for k in [1,3,5,6,7,8,10,15,20]:
    zo_loss = []
    accuracy = []
    for i in range(0,num_folds):
        knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean').fit(X_train[i],y_train[i])
        y_pred = knn.predict(X_test[i])
        zo_loss.append(zero_one_loss(y_test[i],y_pred))
        accuracy.append(accuracy_score(y_test[i],y_pred))
    avg_zo.append(np.mean(zo_loss))
    avg_ac.append(np.mean(accuracy))
```

In [9]:
```
avg_zo
```

Out[9]:
```
[0.09157406392866588,
 0.08406527940454152,
 0.08356214972103625,
 0.0893115105603204,
 0.0873159091195325,
 0.09206967481517865,
 0.09431904510018767,
 0.09481339025956853,
 0.10557783277288699]
```

In [10]:
```
avg_ac
```

Out[10]:
```
[0.9084259360713342,
 0.9159347205954586,
 0.9164378502789639,
 0.9106884894396796,
 0.9126840908804675,
 0.9079303251848214,
 0.9056809548998125,
 0.9051866097404314,
 0.894422167227113]
```

Best model: k=5, accuracy = 0.916

Now what if we augment? does that help?

In [12]:
```
AugData = neuralData_LE.copy()
AugData.keys()

AugData['animalNamexCorrelationScore'] = AugData['animalName']*AugData['correlationScore']
AugData['animalNamexLickAccuracy'] = AugData['animalName']*AugData['lickAccuracy']
AugData['animalNamexLickNumber'] = AugData['animalName']*AugData['lickNumber']
AugData['animalNamexAvgFR'] = AugData['animalName']*AugData['avgFR']
AugData['animalNamexAvgSingleCellVariance'] = AugData['animalName']*AugData['avgSingleCellVariance']
AugData['animalNamexVarianceFR'] = AugData['animalName']*AugData['varianceFR']
AugData['animalNamexAvgTrialSpeed'] = AugData['animalName']*AugData['avgTrialSpeed']
AugData['animalNamexVarianceSpeed'] = AugData['animalName']*AugData['varianceSpeed']

AugData['totalCellNumxCorrelationScore'] = AugData['totalCellNum']*AugData['correlationScore']
AugData['totalCellNumxLickAccuracy'] = AugData['totalCellNum']*AugData['lickAccuracy']
AugData['totalCellNumxLickNumber'] = AugData['totalCellNum']*AugData['lickNumber']
AugData['totalCellNumxAvgFR'] = AugData['totalCellNum']*AugData['avgFR']
AugData['totalCellNumxAvgSingleCellVariance'] = AugData['totalCellNum']*AugData['avgSingleCellVariance']
AugData['totalCellNumxVarianceFR'] = AugData['totalCellNum']*AugData['varianceFR']
AugData['totalCellNumxAvgTrialSpeed'] = AugData['totalCellNum']*AugData['avgTrialSpeed']
AugData['totalCellNumxVarianceSpeed'] = AugData['totalCellNum']*AugData['varianceSpeed']

AugData['genderxCorrelationScore'] = AugData['gender']*AugData['correlationScore']
AugData['genderxLickAccuracy'] = AugData['gender']*AugData['lickAccuracy']
AugData['genderxLickNumber'] = AugData['gender']*AugData['lickNumber']
AugData['genderxAvgFR'] = AugData['gender']*AugData['avgFR']
AugData['genderxAvgSingleCellVariance'] = AugData['gender']*AugData['avgSingleCellVariance']
AugData['genderxVarianceFR'] = AugData['gender']*AugData['varianceFR']
AugData['genderxAvgTrialSpeed'] = AugData['gender']*AugData['avgTrialSpeed']
AugData['genderxVarianceSpeed'] = AugData['gender']*AugData['varianceSpeed']

AugData['genotypexCorrelationScore'] = AugData['genotype']*AugData['correlationScore']
AugData['genotypexLickAccuracy'] = AugData['genotype']*AugData['lickAccuracy']
AugData['genotypexLickNumber'] = AugData['genotype']*AugData['lickNumber']
AugData['genotypexAvgFR'] = AugData['genotype']*AugData['avgFR']
AugData['genotypexAvgSingleCellVariance'] = AugData['genotype']*AugData['avgSingleCellVariance']
AugData['genotypexVarianceFR'] = AugData['genotype']*AugData['varianceFR']
AugData['genotypexAvgTrialSpeed'] = AugData['genotype']*AugData['avgTrialSpeed']
AugData['genotypexVarianceSpeed'] = AugData['genotype']*AugData['varianceSpeed']

AugData['weight_gxCorrelationScore'] = AugData['weight_g']*AugData['correlationScore']
AugData['weight_gxLickAccuracy'] = AugData['weight_g']*AugData['lickAccuracy']
AugData['weight_gxLickNumber'] = AugData['weight_g']*AugData['lickNumber']
AugData['weight_gxAvgFR'] = AugData['weight_g']*AugData['avgFR']
AugData['weight_gxAvgSingleCellVariance'] = AugData['weight_g']*AugData['avgSingleCellVariance']
AugData['weight_gxVarianceFR'] = AugData['weight_g']*AugData['varianceFR']
AugData['weight_gxAvgTrialSpeed'] = AugData['weight_g']*AugData['avgTrialSpeed']
AugData['weight_gxVarianceSpeed'] = AugData['weight_g']*AugData['varianceSpeed']

AugData['ketamine_dayxCorrelationScore'] = AugData['ketamine_day']*AugData['correlationScore']
AugData['ketamine_dayxLickAccuracy'] = AugData['ketamine_day']*AugData['lickAccuracy']
AugData['ketamine_dayxLickNumber'] = AugData['ketamine_day']*AugData['lickNumber']
AugData['ketamine_dayxAvgFR'] = AugData['ketamine_day']*AugData['avgFR']
AugData['ketamine_dayxAvgSingleCellVariance'] = AugData['ketamine_day']*AugData['avgSingleCellVariance']
AugData['ketamine_dayxVarianceFR'] = AugData['ketamine_day']*AugData['varianceFR']
AugData['ketamine_dayxAvgTrialSpeed'] = AugData['ketamine_day']*AugData['avgTrialSpeed']
AugData['ketamine_dayxVarianceSpeed'] = AugData['ketamine_day']*AugData['varianceSpeed']

AugData['medianCellDepthxCorrelationScore'] = AugData['medianCellDepth']*AugData['correlationScore']
AugData['medianCellDepthxLickAccuracy'] = AugData['medianCellDepth']*AugData['lickAccuracy']
AugData['medianCellDepthxLickNumber'] = AugData['medianCellDepth']*AugData['lickNumber']
AugData['medianCellDepthxAvgFR'] = AugData['medianCellDepth']*AugData['avgFR']
AugData['medianCellDepthxAvgSingleCellVariance'] = AugData['medianCellDepth']*AugData['avgSingleCellVariance']
AugData['medianCellDepthxVarianceFR'] = AugData['medianCellDepth']*AugData['varianceFR']
AugData['medianCellDepthxAvgTrialSpeed'] = AugData['medianCellDepth']*AugData['avgTrialSpeed']
AugData['medianCellDepthxVarianceSpeed'] = AugData['medianCellDepth']*AugData['varianceSpeed']

stdNeuralDataAug = StandardScaler().fit_transform(AugData)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [14]:
```
X, X_test, y, y_test = train_test_split(stdNeuralDataAug,ketBool.values.ravel(), test_size=0.2)
```

In [15]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
skf = StratifiedKFold(n_splits=num_folds)
for train_index, test_index in skf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [16]:
```
avg_zo = []
avg_ac = []
for k in [1,3,5,6,7,8,10,15,20]:
    zo_loss = []
    accuracy = []
    for i in range(0,num_folds):
        knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean').fit(X_train[i],y_train[i])
        y_pred = knn.predict(X_test[i])
        zo_loss.append(zero_one_loss(y_test[i],y_pred))
        accuracy.append(accuracy_score(y_test[i],y_pred))
    avg_zo.append(np.mean(zo_loss))
    avg_ac.append(np.mean(accuracy))
```

In [17]:
```
avg_zo
```

Out[17]:
```
[0.068546465915412,
 0.060540804942530904,
 0.06354456434102715,
 0.06554582684891783,
 0.0675452034075213,
 0.06554082369264809,
 0.06729082994268715,
 0.07329647529047058,
 0.07705023468896682]
```

In [18]:
```
avg_ac
```

Out[18]:
```
[0.9314535340845881,
 0.9394591950574691,
 0.9364554356589728,
 0.9344541731510823,
 0.9324547965924787,
 0.934459176307352,
 0.9327091700573129,
 0.9267035247095293,
 0.9229497653110332]
```

Augmentation gets us to best accuracy of 0.939 with k = 3

In [ ]:
```

```

Some data visualization

In [1]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

In [2]:
```
allData = pd.read_csv('postKetamineTable.csv')
```

In [3]:
```
# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

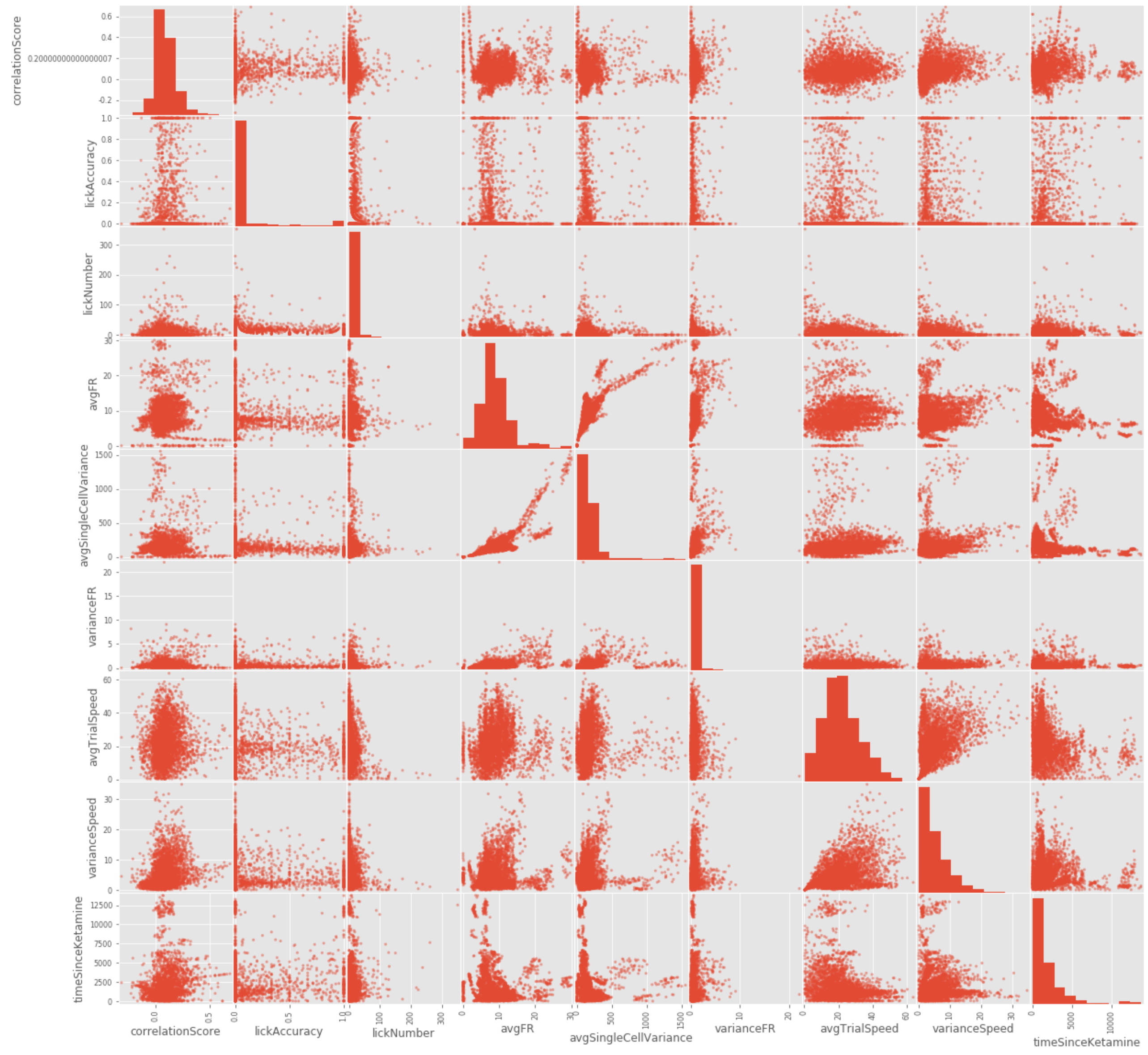
```
After Drop NaN
num_rows = 4995
```

In [4]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
VizData = allDataNN[['correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed']]
```

In [6]:
```
X, X_test, y, y_test = train_test_split(VizData,timeSinceKetamine.values.ravel(), test_size=0.2)
```

In [7]:
```
df_train = pd.DataFrame(X)
df_train['timeSinceKetamine'] = y
df_vis = df_train.copy()
```

```
In [9]: axes = pd.plotting.scatter_matrix(df_vis, figsize=(20,20))
        plt.savefig('scatter_matrix.png')
```



```
In [11]: max(y)
Out[11]: 13781.8

In [12]: min(y)
Out[12]: 8.86000000000001

In [14]: mean_y = np.mean(y)

In [22]: y_pred = mean_y*np.ones(np.shape(y))

In [24]: rmse = np.sqrt(mean_squared_error(y_pred,y))

In [25]: rmse
Out[25]: 2207.247738459438

In [26]: rmse_scaled = rmse/(max(y)-min(y))

In [27]: rmse_scaled
Out[27]: 0.16025973673445454
```

```
In [29]: r2_score(y,y_pred)

Out[29]: 0.0

In [ ]:
```

```matlab
% Matlab Code Used to PreProcess and Compile Data Matrix

% sessions = dir('/Volumes/groups/giocomo/export/data/Projects/JohnKei_NPH3/fkm_analysis/fr_corr_matrices/*.mat');
sessions = dir('/Users/KeiMasuda/Desktop/fkm_analysis/fr_corr_matrices_noSpeedFilter/*.mat');
filter = 'mec';
sessions = filterSessions(sessions, filter);
% load(sprintf('/Users/KeiMasuda/Desktop/fkm_analysis/allSpatialIndx%s_01.mat',filter));
load(sprintf('/Users/KeiMasuda/Desktop/fkm_analysis/allSpatialIndx%s.mat',filter));

sessionMetaData = readtable('/Users/KeiMasuda/Desktop/fkm_analysis/SessionList.xlsx');

fn = fieldnames(allSpatialIndx);
count = 0;
seshVector = [];
for k=1:numel(fn)
    if(isnumeric(allSpatialIndx.(fn{k})))
        count = count + size(allSpatialIndx.(fn{k}),2);
        seshVector = vertcat(seshVector, k);
    end
end

allCellsFR = nan(count,100,200);

metaData = struct;
for i = 1:numel(fn)
    session_name = fn{seshVector(i)};
    animalName = extractBefore(session_name,'_');
    sessionDate = extractAfter(session_name,'_');

    metaData(i).sessionName = session_name;
    metaData(i).animalName = animalName;
    metaData(i).sessionDate = sessionDate;
    metaData(i).gender = string(sessionMetaData(seshVector(i),:).Gender);
    metaData(i).genotype = string(sessionMetaData(seshVector(i),:).Genotype);
    metaData(i).weight_g = sessionMetaData(seshVector(i),:).Weight_g_;
    metaData(i).ketamine_day = sessionMetaData(seshVector(i),:).Ketamine_day;
end
fprintf('Done with the Pre-Allocation\n');
%%
varTypes = {'string','double','double','double','string','string','double','double',...
    'double','double','double','double','double','double','double','double','double','double','logical'};
varNames = {'animalName','sessionDate','trialNum','totalCellNum','gender','genotype','weight_g',...
    'ketamine_day','correlationScore','lickAccuracy','lickNumber','avgFR','avgSingleCellVariance',...
    'varianceFR','avgTrialSpeed','varianceSpeed','medianCellDepth','timeSinceKetamine','ketamineAdministered'};
sz = [numel(fn)*size(allCellsFR,2), numel(varTypes)];
sessionTrialTable = table('Size',sz,'VariableTypes',varTypes,'VariableNames',varNames);

%%
z = 0;
y = 0;
samplingRate = 50; %Hz
% trialRange = 51:150;
trialRange = 101:200;
numTrialsForTable = numel(trialRange);
for n = 1:numel(fn)
    matPath = fullfile(sessions(n).folder, sessions(n).name);
    dataPath = fullfile(sessions(n).folder(1:end-30), strcat(sessions(n).name(1:end-12),'.mat'));
    session_name = sessions(n).name(1:end-4);
    animalName = extractBefore(session_name,'_');
    sessionDate = extractBefore(extractAfter(session_name,'_'),'_');
    seshStr = sprintf('%s_%s',animalName, sessionDate);
    trackLength = 400;
    load(fullfile(matPath), 'all_fr', 'cells_to_plot','trial','all_cellCorrScore','spike_depth');
    load(dataPath, 'lickt','lickx', 'post','posx')

    spatialIndx = ismember(cells_to_plot,allSpatialIndx.(seshStr));

    all_fr = all_fr(spatialIndx,trialRange,:);

    nCells = size(all_fr,1);
    allCellsFR(z+1:z+nCells,1:numTrialsForTable,:) = all_fr(1:nCells,1:numTrialsForTable,1:200);

    ketamineInjxTimeSec = find(trial==100,1)/samplingRate;
    controlInjxTimeSec = find(trial==50,1)/samplingRate;


    animalName = string(metaData(n).animalName);
    sessionDate = metaData(n).sessionDate;
    gender = metaData(n).gender;
    genotype = metaData(n).genotype;
    weight_g = metaData(n).weight_g;
    ketamine_day = metaData(n).ketamine_day;

    rows = y+1:(y + numTrialsForTable);

    trialStartTime = arrayfun(@(x) find(trial==x,1)/samplingRate,trialRange','UniformOutput',false);
    timeSinceKetamine = cell2mat(trialStartTime)-ketamineInjxTimeSec;
    ketamineAdminBool = timeSinceKetamine>=0;


    [~,~,lick_idx] = histcounts(lickt,post);

    lickAccuracyByTrial = zeros(1,numTrialsForTable);
    lickNumByTrial = zeros(1,numTrialsForTable);

    for i = 1:numTrialsForTable
        j = trialRange(i);
        trialLicks = lickx(trial(lick_idx) == j);
        goodLicks = sum(trialLicks<5) + sum(trialLicks>max(posx)-15);
        if trialLicks ~= 0
            lickAccuracyByTrial(i) = goodLicks/numel(trialLicks);
            lickNumByTrial(i) = numel(trialLicks);
        else
            lickAccuracyByTrial(i) = 0.0;
            lickNumByTrial(i) = 0.0;
        end
    end

    corrScoreByTrial = nanmean(all_cellCorrScore(:,trialRange),1); %avg corr scorr for each trial avg across all cells

    avgFRbyTrial = nanmean(squeeze(nanmean(all_fr,1)),2);
    avgSingleCellVariance = var(squeeze(nanmean(all_fr,3)),0,1);
    varianceFR = var(squeeze(nanmean(all_fr,1)),0,2);

    trialSpeed = arrayfun(@(x) 400/(numel(find(trial==x))/samplingRate),trialRange','UniformOutput',false);
    trialSpeed = cell2mat(trialSpeed); %speed in cm/s

    varianceSpeed = arrayfun(@(x) var(diff(posx(find(trial==x))))*samplingRate,trialRange','UniformOutput',false);
    varianceSpeed = cell2mat(varianceSpeed);

    medianCellDepth = median(spike_depth);

    rowCount = numel(rows);
    rowData = table(...
        repmat(animalName,rowCount,1),...
        repmat(str2double(sessionDate),rowCount,1),...
        trialRange',...
        repmat(numel(cells_to_plot),rowCount,1),...
        repmat(gender,rowCount,1),...
        repmat(genotype,rowCount,1),...
        repmat(weight_g,rowCount,1),...
        repmat(ketamine_day,rowCount,1),...
        corrScoreByTrial',...
        lickAccuracyByTrial',...
        lickNumByTrial',...
        avgFRbyTrial,...
        avgSingleCellVariance',...
        varianceFR,...
        trialSpeed,...
        varianceSpeed,...
        repmat(medianCellDepth,rowCount,1),...
        timeSinceKetamine,...
        ketamineAdminBool...
    );


    sessionTrialTable(rows,:) = rowData;
    y = y + numTrialsForTable;


    z = z + nCells;
    fprintf('Session: %d; Adding %d for %d/%d cells\n', n,nCells,z,count)

end

fprintf('done\n')


%%
% writetable(sessionTrialTable,"/Users/KeiMasuda/Dropbox/1_SMS/NEURS/Electives/MS&E 226/Project/sessionTrialTable.csv");
writetable(sessionTrialTable,"/Users/KeiMasuda/Dropbox/1_SMS/NEURS/Electives/MS&E 226/Project/postKetamineTable.csv");
% [row, col] = find(ismissing(sessionTrialTable))

% %%
% postKetamineTrials = sessionTrialTable(sessionTrialTable.trialNum >100,:);
% writetable(postKetamineTrials,"/Users/KeiMasuda/Dropbox/1_SMS/NEURS/Electives/MS&E 226/Project/postKetamineTable.csv");

%%
WTtable = sessionTrialTable(sessionTrialTable.genotype == "WT",:);
HCNkoTable = sessionTrialTable(sessionTrialTable.genotype == "KO",:);
%%
addpath(genpath('/Users/KeiMasuda/Documents/MATLAB/Add-Ons/Functions/gramm (complete data visualization toolbox, ggplot2_R-like)/code'));
close all;
```

```
clear g;
g = gramm('x',categorical(sessionTrialTable.genotype),'y',sessionTrialTable.lickAccuracy,'color',categorical(sessionTrialTable.genotype));
g.facet_grid([],categorical(sessionTrialTable.gender))
g.set_names('x','Genotype','y','lickAccuracy', 'column','Gender')
g.stat_violin();
g.draw();
```