In [82]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style='white', context='notebook', palette='deep')
import matplotlib.style as style
style.use('fivethirtyeight')
```

EDA

In [152]:
```python
train = pd.read_csv("BUDStrain.csv", index_col = 0)
pd.set_option('display.max_columns', 999)
print(train.shape)
#train.describe(include = 'all')
train.info()
```

```
(486, 31)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 486 entries, 177 to 133
Data columns (total 31 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      486 non-null    object
 1   sex         486 non-null    object
 2   age         486 non-null    int64 
 3   address     486 non-null    object
 4   famsize     486 non-null    object
 5   Pstatus     486 non-null    object
 6   Medu        486 non-null    int64 
 7   Fedu        486 non-null    int64 
 8   Mjob        486 non-null    object
 9   Fjob        486 non-null    object
 10  reason      486 non-null    object
 11  guardian    486 non-null    object
 12  traveltime  486 non-null    int64 
 13  studytime   486 non-null    int64 
 14  failures    486 non-null    int64 
 15  schoolsup   486 non-null    object
 16  famsup      486 non-null    object
 17  paid        486 non-null    object
 18  activities  486 non-null    object
 19  nursery     486 non-null    object
 20  higher      486 non-null    object
 21  internet    486 non-null    object
 22  romantic    486 non-null    object
 23  famrel      486 non-null    int64 
 24  freetime    486 non-null    int64 
 25  goout       486 non-null    int64 
 26  Dalc        486 non-null    int64 
 27  Walc        486 non-null    int64 
 28  health      486 non-null    int64 
 29  absences    486 non-null    int64 
 30  grade       486 non-null    int64 
dtypes: int64(14), object(17)
memory usage: 121.5+ KB
```

No nulls!

In [153]:
```python
train.head()
```

Out[153]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
ID											
177	GP	F	17	U	GT3	T	2	2	other	other	course
368	GP	F	15	U	GT3	T	2	2	other	other	course
120	GP	M	17	U	GT3	T	1	2	at_home	services	other
230	MS	F	17	R	GT3	T	1	1	other	services	reputation
353	GP	F	18	U	LE3	T	2	2	other	other	home

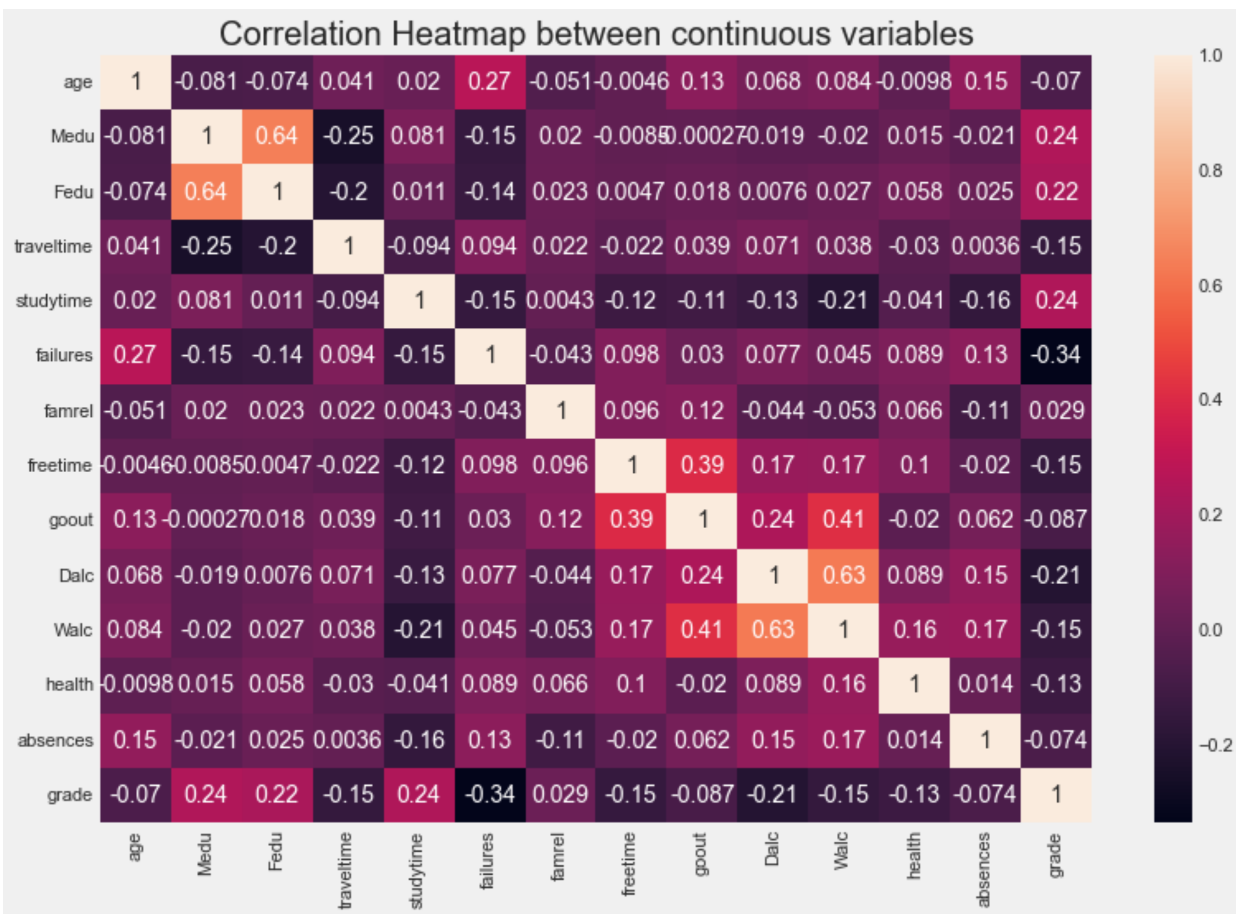
In [154]:
```python
all_features = train.columns.tolist()
num_features = train.describe().columns.tolist()
cat_features = [feat for feat in all_features if feat not in numerical
assert(len(all_features) == len(num_features) + len(cat_features))
train.describe()
```

Out[154]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel
count	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000
mean	16.718107	2.473251	2.283951	1.567901	1.930041	0.216049	3.958848
std	1.194818	1.149767	1.103710	0.742327	0.819380	0.599510	0.942881
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000
25%	16.000000	2.000000	1.000000	1.000000	1.000000	0.000000	4.000000
50%	17.000000	2.000000	2.000000	1.000000	2.000000	0.000000	4.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000
max	21.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000

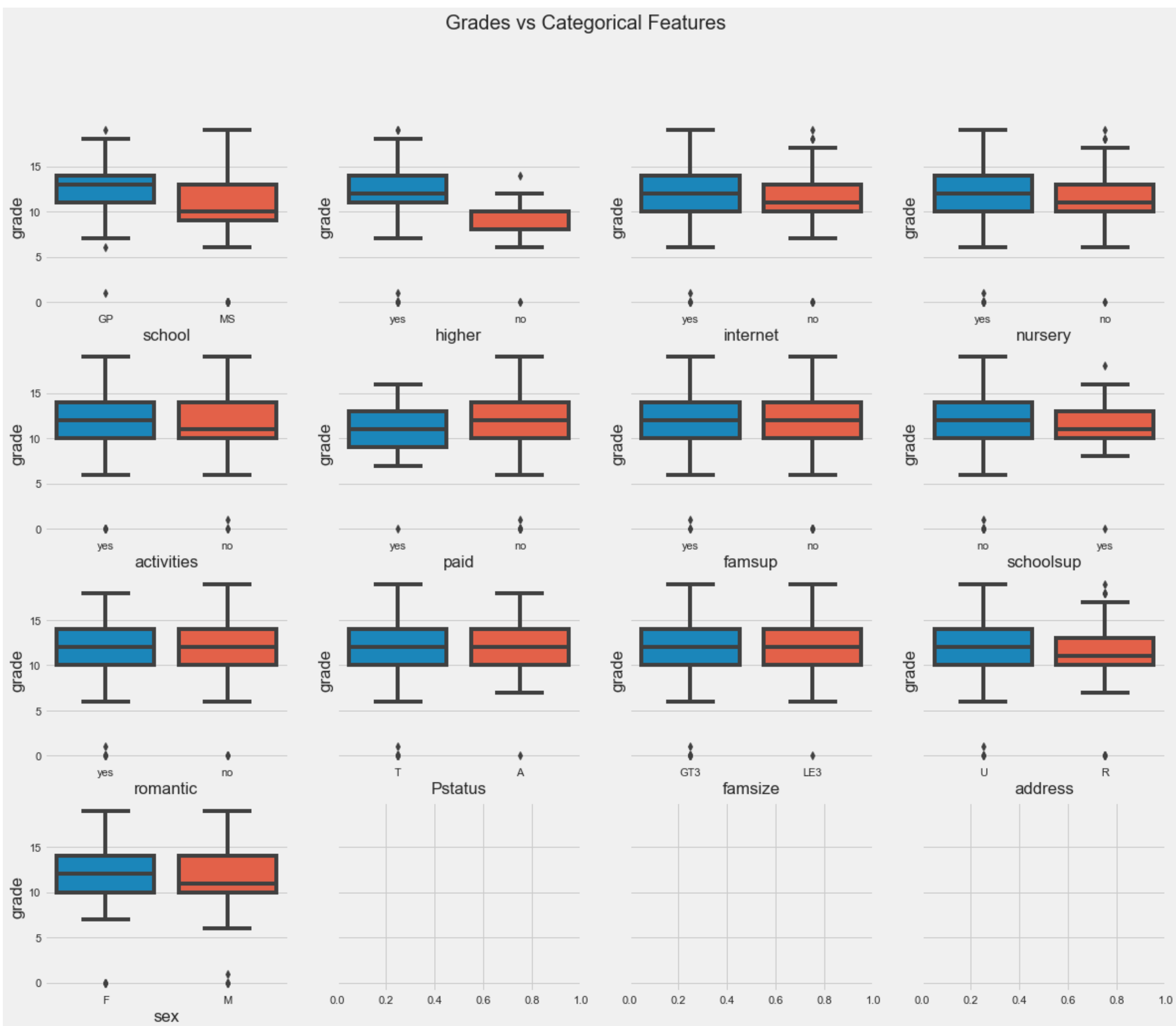
In [155]:
```
train[cat_features].nunique().sort_values(ascending=True)
```
Out[155]:
```
school        2
higher        2
nursery       2
activities    2
paid          2
famsup        2
schoolsup     2
internet      2
romantic      2
Pstatus       2
famsize       2
address       2
sex           2
guardian      3
reason        4
Fjob          5
Mjob          5
dtype: int64
```

In [156]:
```
plt.figure(figsize=(12,8))
sns.heatmap(train[num_features].corr(), annot=True)
plt.title("Correlation Heatmap between continuous variables")
```
Out[156]:
```
Text(0.5, 1.0, 'Correlation Heatmap between continuous variables')
```



In [157]:
```
plt.figure()
fig, axes = plt.subplots(4, 4, figsize=(18, 15), sharey=True)
fig.suptitle("Grades vs Categorical Features", fontsize=20)

sns.boxplot(ax=axes[0, 0], data=train, x='school', y='grade')
sns.boxplot(ax=axes[0, 1], data=train, x='higher', y='grade')
sns.boxplot(ax=axes[0, 2], data=train, x='internet', y='grade')
sns.boxplot(ax=axes[0, 3], data=train, x='nursery', y='grade', order=[
sns.boxplot(ax=axes[1, 0], data=train, x='activities', y='grade', orde
sns.boxplot(ax=axes[1, 1], data=train, x='paid', y='grade', order=['ye
sns.boxplot(ax=axes[1, 2], data=train, x='famsup', y='grade')
sns.boxplot(ax=axes[1, 3], data=train, x='schoolsup', y='grade')
sns.boxplot(ax=axes[2, 0], data=train, x='romantic', y='grade', order=
sns.boxplot(ax=axes[2, 1], data=train, x='Pstatus', y='grade')
sns.boxplot(ax=axes[2, 2], data=train, x='famsize', y='grade')
sns.boxplot(ax=axes[2, 3], data=train, x='address', y='grade')
sns.boxplot(ax=axes[3, 0], data=train, x='sex', y='grade')
```

Out[157]:
```
<AxesSubplot:xlabel='sex', ylabel='grade'>

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

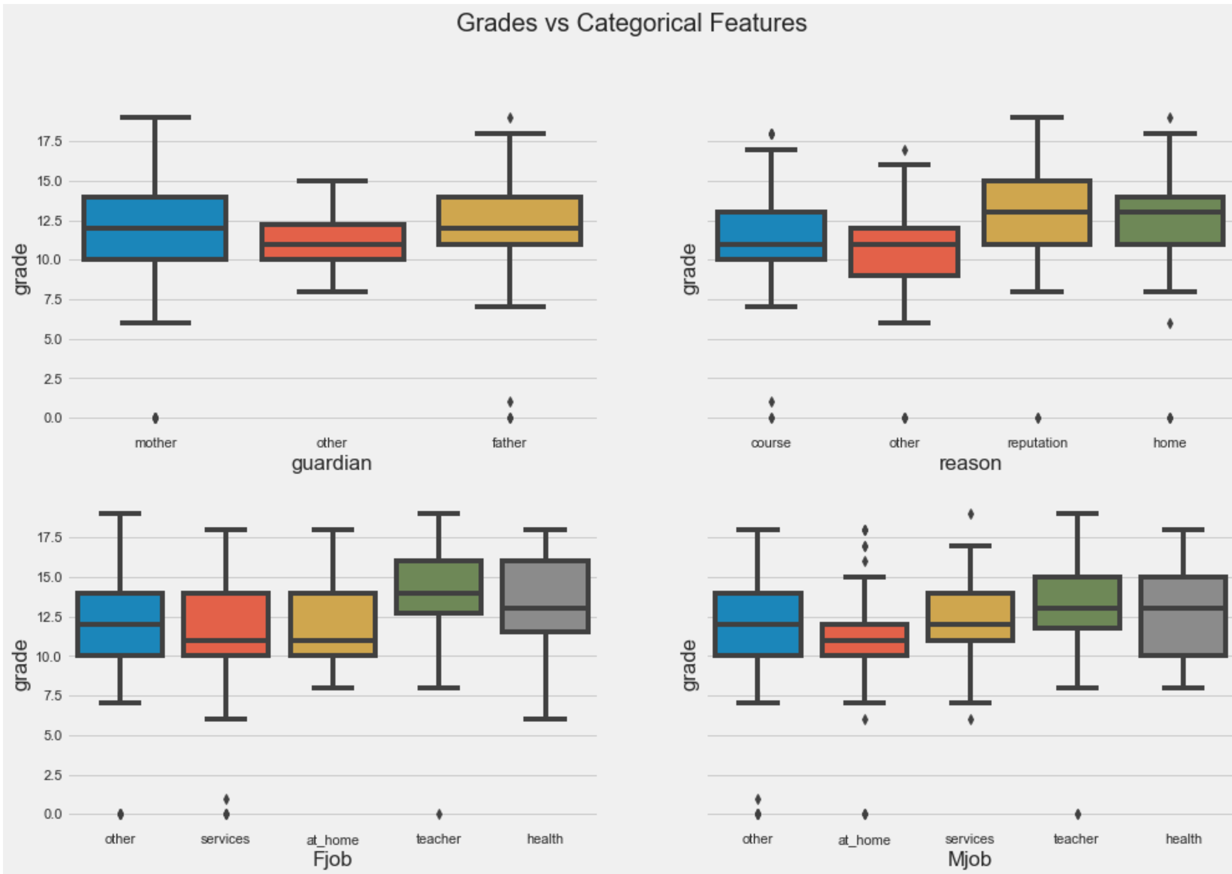


In [158]:

```
fig, axes = plt.subplots(2, 2, figsize=(15, 10), sharey=True)
fig.suptitle("Grades vs Categorical Features", fontsize=20)

sns.boxplot(ax=axes[0, 0], data=train, x='guardian', y='grade')
sns.boxplot(ax=axes[0, 1], data=train, x='reason', y='grade')
sns.boxplot(ax=axes[1, 0], data=train, x='Fjob', y='grade')
sns.boxplot(ax=axes[1, 1], data=train, x='Mjob', y='grade')
```

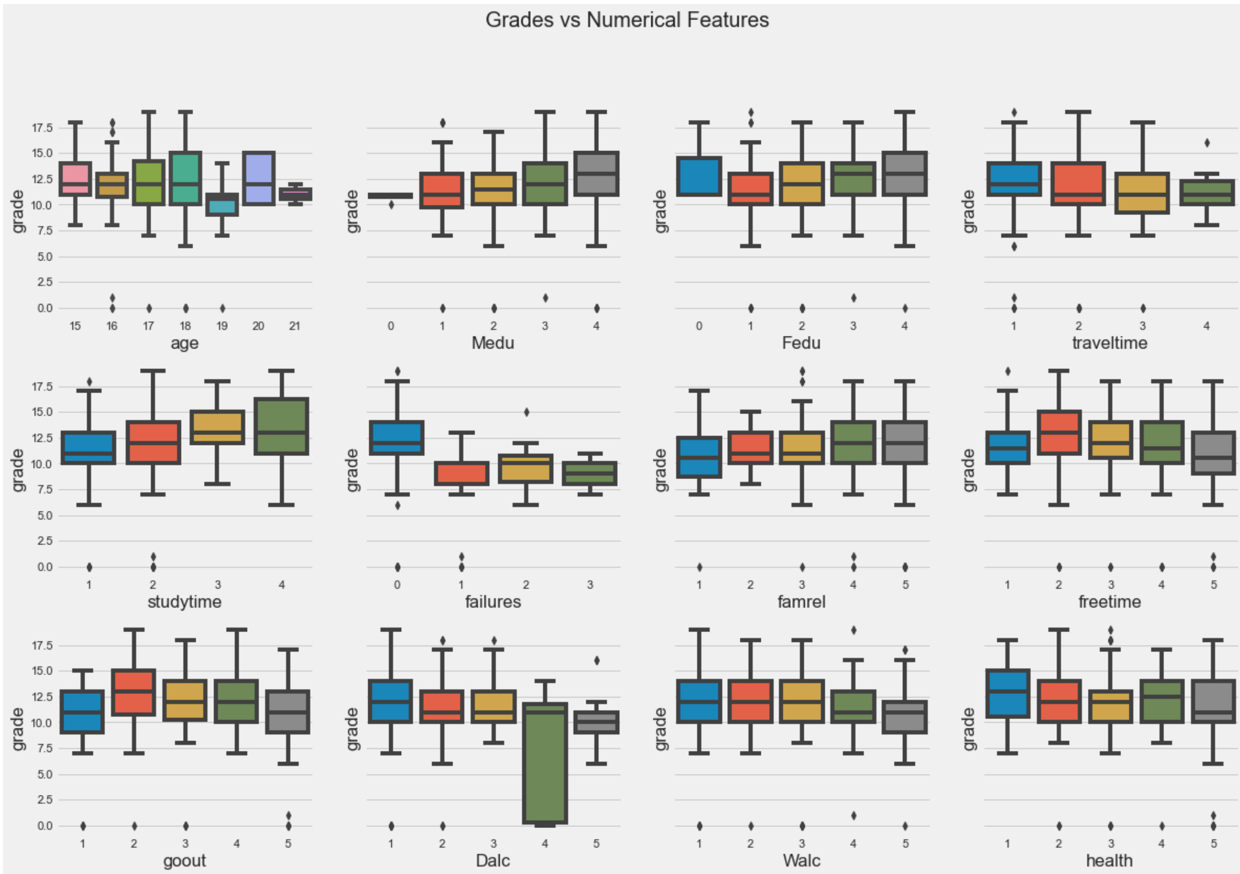
Out[158]:

```
<AxesSubplot:xlabel='Mjob', ylabel='grade'>
```



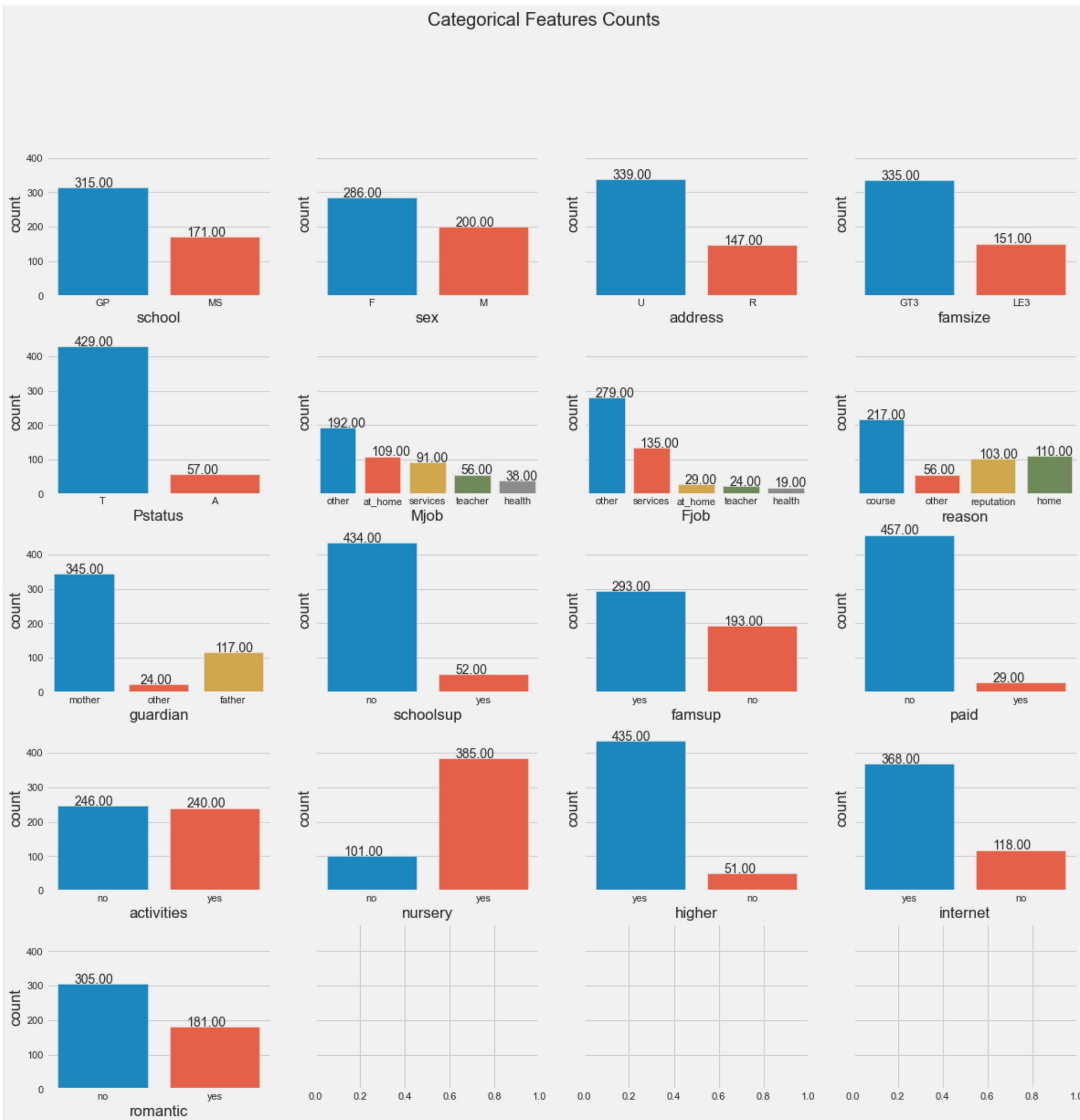
In [159]:
```
plt.figure()
fig, axes = plt.subplots(3, 4, figsize=(18, 12), sharey=True)
fig.suptitle("Grades vs Numerical Features", fontsize=20)

for i in range(3):
    for j in range(4):
        sns.boxplot(ax=axes[i, j], data=train, x=np.array(num_features
```

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



```
In [160]: # For unbalanced datasets
plt.figure()
fig, axes = plt.subplots(5, 4, figsize=(18, 18), sharey=True)
fig.suptitle("Categorical Features Counts", fontsize=20)

for i in range(4):
    for j in range(4):
        sns.countplot(ax=axes[i, j], data=train, x=np.array(cat_featur
        for p in axes[i,j].patches:
            axes[i,j].annotate('{:.2f}'.format(p.get_height()), (p.get
sns.countplot(ax=axes[4,0], data=train, x=cat_features[-1])
for p in axes[4,0].patches:
    axes[4,0].annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.1
```

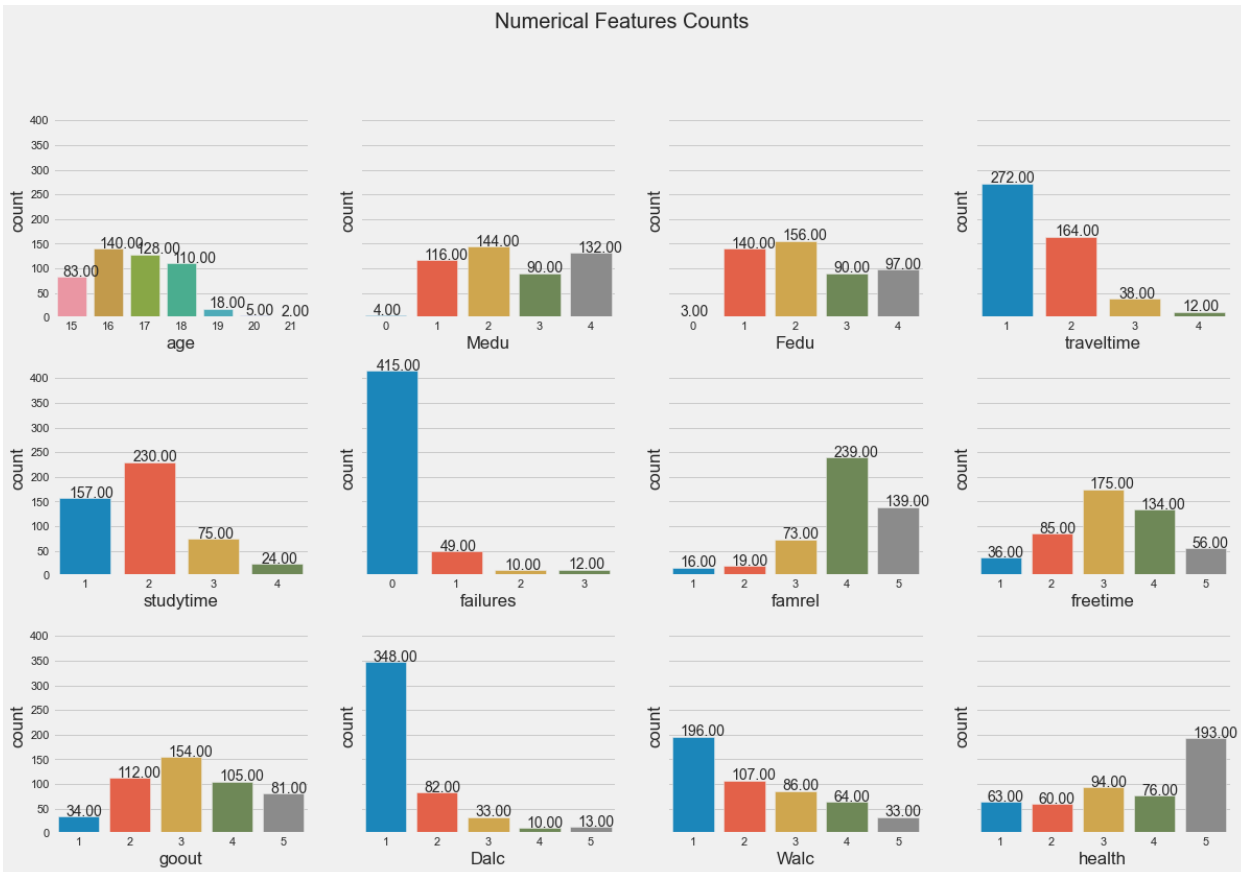
```
<Figure size 432x288 with 0 Axes>
```



In [161]:

```
# Basically also categorical
plt.figure()
fig, axes = plt.subplots(3, 4, figsize=(18, 12), sharey=True)
fig.suptitle("Numerical Features Counts", fontsize=20)

for i in range(3):
    for j in range(4):
        sns.countplot(ax=axes[i, j], data=train, x=np.array(num_featur
        for p in axes[i,j].patches:
            axes[i,j].annotate('{:.2f}'.format(p.get_height()), (p.get
```

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



In [ ]:

Preprocessing Data

In [162]:
```
# One Hot Encoding

train = pd.get_dummies(data=train, columns=cat_features)
train.head()
```

Out[162]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	he
ID												
177	17	2	2	1	2	0	5	4	5	1	2	
368	15	2	2	1	4	0	5	1	2	1	1	
120	17	1	2	2	2	0	4	4	4	4	5	
230	17	1	1	3	1	1	5	2	1	1	2	
353	18	2	2	1	2	0	4	3	3	1	1	

In [166]:
```
from sklearn.model_selection import train_test_split

X = train.drop('grade', axis=1)
y = train['grade']

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.25
```

In [180]:
```
all_feats = X.columns.tolist()
```

ML Algos

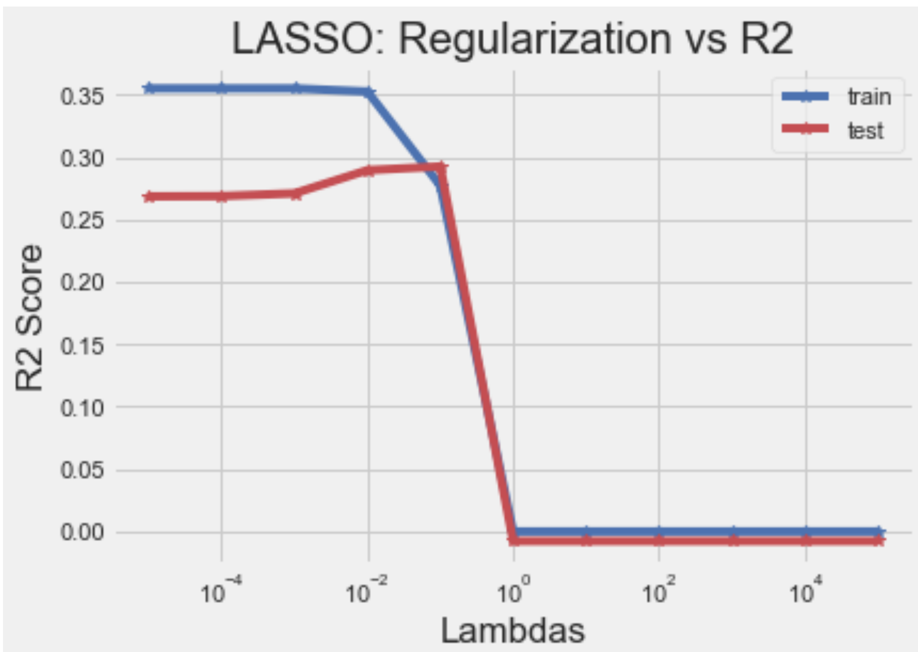
All categorical variables vs grades

LASSO

In [170]:
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')

lmbdas = np.logspace(-5,5,11)

train_accuracy = np.zeros(len(lmbdas))
test_accuracy = np.zeros(len(lmbdas))

for i, lmbda in enumerate(lmbdas):

    lasso_reg = linear_model.Lasso(alpha = lmbda, random_state = 1)
    lasso_reg.fit(train_X, train_y)
```

```
    # check accuracy
    train_accuracy[i] = lasso_reg.score(train_X, train_y)
    test_accuracy[i] = lasso_reg.score(val_X, val_y)

plt.semilogx(lmbdas, train_accuracy,'*-b', label='train')
plt.semilogx(lmbdas, test_accuracy,'*-r', label='test')
plt.title("LASSO: Regularization vs R2")
plt.ylabel("R2 Score")
plt.xlabel("Lambdas")
plt.legend()

max_acc = max(test_accuracy)
max_index = np.argmax(test_accuracy)
print("Optimal index:", max_index, "\nBest test accuracy:", max_acc, "
```

```
Optimal index: 4
Best test accuracy: 0.2925656129127092
Optimal Lambda: 0.1
```



In [189]:
```
lasso_opt = linear_model.Lasso(alpha = 0.1, random_state = 1)
lasso_opt.fit(train_X, train_y)
val_pred = lasso_opt.predict(val_X)
rms = mean_squared_error(val_y, val_pred, squared=False)
print('intercept: ', lasso_opt.intercept_)
for i in (list(zip(train_X[all_feats], lasso_opt.coef_))):
    print(i, sep='\n')
print(f"RMSE for Lasso w/ lambda=0.1: {rms}")
```

```
intercept:  11.263111952382127
('age', 0.007665992961792124)
('Medu', 0.09154899692650595)
('Fedu', 0.3194599031810628)
('traveltime', -0.0)
('studytime', 0.3658945241837787)
('failures', -1.213622864931187 5)
```

```
('famrel', -0.0)
('freetime', -0.1459677636654572)
('goout', -0.0)
('Dalc', -0.20469914170031145)
('Walc', -0.10882354410367069)
('health', -0.10270330802915616)
('absences', -0.0156339630826885)
('school_GP', 0.9991504231565367)

('school_MS', -8.561889427194004e-17)
('sex_F', 0.0)
('sex_M', -0.0)
('address_R', -0.13901479938458197)
('address_U', 0.0)
('famsize_GT3', -0.0)
('famsize_LE3', 0.0)
('Pstatus_A', 0.0)
('Pstatus_T', -0.0)
('Mjob_at_home', -0.0)
('Mjob_health', 0.0)
('Mjob_other', -0.0)
('Mjob_services', 0.0)
('Mjob_teacher', 0.0)
('Fjob_at_home', 0.0)
('Fjob_health', 0.0)
('Fjob_other', 0.0)
('Fjob_services', -0.13262504929577432)
('Fjob_teacher', 0.0)
('reason_course', -0.0)
('reason_home', 0.0)
('reason_other', -0.0)
('reason_reputation', 0.0)
('guardian_father', 0.0)
('guardian_mother', -0.0)
('guardian_other', 0.0)
('schoolsup_no', 0.03812318915924627)
('schoolsup_yes', -1.0202664923734771e-15)
('famsup_no', 0.0)
('famsup_yes', -0.0)
('paid_no', 0.0)
('paid_yes', -0.0)
('activities_no', -0.0)
('activities_yes', 0.0)
('nursery_no', 0.0)
('nursery_yes', -0.0)
('higher_no', -0.7849127881503086)
('higher_yes', 0.0)
('internet_no', -0.14590414686884653)
('internet_yes', 0.0)
('romantic_no', 0.0)
('romantic_yes', -0.0)
RMSE for Lasso w/ lambda=0.1: 2.6443510578879295
```

The variables with the most weights are:

```
 ('age', 0.007666)
('Medu', 0.09155)
('Fedu', 0.31946)
('studytime', 0.3659)
('failures', -1.2136)
('freetime', -0.14597)
('Dalc', -0.2047)
('Walc', -0.1088)
('health', -0.1027)
('absences', -0.0156)
('school_GP', 0.999)
('address_R', -0.139)
('Fjob_services', -0.13262)
('schoolsup_no', 0.0381)
('higher_no', -0.7849)
('internet_no', -0.1459)
```

Decision Trees

In [196]:

```
from sklearn.tree import DecisionTreeRegressor
def sort_tuple(tup):
    tup.sort(key = lambda x: x[1])
    return tup
```

In [200]:
```
num_trees = np.logspace(1,4,4,dtype=int)

for i, num in enumerate(num_trees):

    tree = DecisionTreeRegressor(max_depth = num, random_state=1)
    tree.fit(train_X, train_y)

    val_pred = tree.predict(val_X)
    train_pred = tree.predict(train_X)

    rmseTrain = mean_squared_error(train_y, train_pred, squared=False)
    print("\nTrain RMSE for num_est =", num, rmseTrain)

    rmse = mean_squared_error(val_y, val_pred, squared=False)
    print("\nRMSE for num_est =", num, rmse)
```

```
Train RMSE for num_est = 10 1.042764022368624

RMSE for num_est = 10 3.6622110826562535

Train RMSE for num_est = 100 0.0

RMSE for num_est = 100 3.896194023512235

Train RMSE for num_est = 1000 0.0

RMSE for num_est = 1000 3.896194023512235

Train RMSE for num_est = 10000 0.0

RMSE for num_est = 10000 3.896194023512235
```

In [208]:
```
num_trees = np.linspace(1,10,10,dtype=int)

print("num, RMSE Train, RMSE Test")
for i, num in enumerate(num_trees):

    tree = DecisionTreeRegressor(max_depth = num, random_state=1)
    tree.fit(train_X, train_y)

    val_pred = tree.predict(val_X)
    train_pred = tree.predict(train_X)

    rmseTrain = mean_squared_error(train_y, train_pred, squared=False)
    rmse = mean_squared_error(val_y, val_pred, squared=False)

    print(num, rmseTrain, rmse)
```

```
num, RMSE Train, RMSE Test
1 2.9609602632514105 2.900716483128254
2 2.825005562309739 2.972739729935821
3 2.6268359738163984 2.9010292805004543
4 2.4414600186526036 2.8025815839200128
5 2.2396671536532664 3.1263538602685332
6 1.9671367732646547 3.1186851194230583
7 1.7368429539963608 3.6003768571396093
8 1.5380737636037596 3.2671948623611486
9 1.277112858915956 3.936851614577229
10 1.042764022368624 3.6622110826562535
```

In [207]:
```
tree_opt = DecisionTreeRegressor(max_depth = 4, random_state=1)
tree_opt.fit(train_X, train_y)

# Feature Importance
importances = list(tree_opt.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, im

f = sort_tuple(feature_importances)
for i in f[-5:]:
    # top 5 most important features
    print(i, sep='\n')
```

```
('activities_no', 0.06)
('guardian_father', 0.07)
('higher_yes', 0.08)
('school_MS', 0.15)
('failures', 0.39)
```

Random Forest

In [190]:
```
from sklearn.ensemble import RandomForestRegressor

num_est = np.logspace(1,4,4, dtype=int)

for i, num in enumerate(num_est):

    rf = RandomForestRegressor(n_estimators = num, random_state = 1)
    rf.fit(train_X, train_y)
    val_pred = rf.predict(val_X)

    rmse = mean_squared_error(val_y, val_pred, squared=False)
    print("\nRMSE for num_est =", num, rmse)
```

```
RMSE for num_est = 10 2.4878063284485896

RMSE for num_est = 100 2.586409626122351

RMSE for num_est = 1000 2.582233138894813

RMSE for num_est = 10000 2.5761228805427234
```

In [209]:
```
print("Best Algo is RF with n_est = 10")
rf = RandomForestRegressor(n_estimators = 10, random_state = 1)
rf.fit(train_X, train_y)

# Feature Importance
importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, im

f = sort_tuple(feature_importances)
for i in f[-5:]:
    # top 5 most important features
    print(i, sep='\n')
```

```
Best Algo is RF with n_est = 10
('Walc', 0.04)
('Medu', 0.05)
('goout', 0.06)
('absences', 0.06)
('failures', 0.19)
```

In [216]:
```
# source: https://towardsdatascience.com/hyperparameter-tuning-the-ran
# Narrowing down parameters for hyperparameter tuning with GridSearchC
from pprint import pprint
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(200, stop = 2000, num = 10
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2
000]}
```

In [220]:
```
rf = RandomForestRegressor()
# 3 fold CV for run time
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = r
                               n_iter = 100, cv = 3, verbose=2, random
rf_random.fit(train_X, train_y)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

---------------------------------------------------------------------
------
AttributeError                           Traceback (most recent call
last)
Input In [220], in <cell line: 7>()
      3 rf_random = RandomizedSearchCV(estimator = rf, param_distribu
tions = random_grid,
      4                                n_iter = 100, cv = 3, verbose=
2, random_state=1, n_jobs = -1)
      5 rf_random.fit(train_X, train_y)
----> 7 rf_random.best_params()

AttributeError: 'RandomizedSearchCV' object has no attribute 'best_pa
rams'
```

In [221]:
```
rf_random.best_params_
```

```
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samp
les_leaf=4, min_samples_split=10, n_estimators=1200; total time=   1.
7s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_sampl
es_leaf=1, min_samples_split=2, n_estimators=800; total time=   1.4s
[CV] END bootstrap=True, max_depth=30, max_features=auto, min_samples
_leaf=4, min_samples_split=5, n_estimators=1800; total time=   4.5s
[CV] END bootstrap=True, max_depth=110, max_features=auto, min_sample
s_leaf=1, min_samples_split=2, n_estimators=1600; total time=   5.0s
[CV] END bootstrap=False, max_depth=40, max_features=sqrt, min_sample
s_leaf=4, min_samples_split=5, n_estimators=1600; total time=   2.2s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_sample
s_leaf=1, min_samples_split=5, n_estimators=1800; total time=   2.7s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_sample
s_leaf=1, min_samples_split=2, n_estimators=600; total time=   2.2s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples
_leaf=1, min_samples_split=10, n_estimators=2000; total time=   3.6s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_sampl
es_leaf=2, min_samples_split=5, n_estimators=1200; total time=   2.3s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples
```

In [222]:
```
best_params = {'n_estimators': 200,
               'min_samples_split': 5,
               'min_samples_leaf': 1,
               'max_features': 'sqrt',
               'max_depth': 90,
               'bootstrap': False}

rf = RandomForestRegressor(**best_params, random_state = 1)
rf.fit(train_X, train_y)
val_pred = rf.predict(val_X)

rmse = mean_squared_error(val_y, val_pred, squared=False)
print("\nRMSE for num_est =", num, rmse)

# Feature Importance
importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, im

f = sort_tuple(feature_importances)
for i in f[-5:]:
    # top 5 most important features
    print(i, sep='\n')
```

```
RMSE for num_est = 10 2.5041265632727976
('Walc', 0.04)
('absences', 0.04)
('school_GP', 0.04)
('higher_yes', 0.04)
('failures', 0.1)
```

In [224]:
```python
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [80, 90, 100],
    'max_features': [7, 8],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [3, 4, 5, 6, 7],
    'n_estimators': [100, 150, 200, 250, 300]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 10, n_jobs = -1, verbose = 2)

grid_search.fit(train_X, train_y)
```

```
Fitting 10 folds for each of 450 candidates, totalling 4500 fits
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=150; total time=   0.3s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=150; total time=   0.5s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=200; total time=   0.4s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=200; total time=   0.4s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=200; total time=   0.3s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
eaf=1, min_samples_split=3, n_estimators=250; total time=   0.4s
[CV] END bootstrap=False, max_depth=80, max_features=7, min_samples_l
```

In [225]:
```python
grid_search.best_params_
```

Out[225]:
```
{'bootstrap': False,
 'max_depth': 80,
 'max_features': 8,
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 300}
```

In [233]:
```
best_params = {'bootstrap': False,
               'max_depth': 80,
               'max_features': 8,
               'min_samples_leaf': 1,
               'min_samples_split': 6,
               'n_estimators': 300}

rf = RandomForestRegressor(**best_params, random_state = 1)
rf.fit(train_X, train_y)
val_pred = rf.predict(val_X)

rmse = mean_squared_error(val_y, val_pred, squared=False)
print("\nRMSE for num_est =", num, rmse)

# Feature Importance
importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, im

f = sort_tuple(feature_importances)
for i in f[-5:]:
    # top 5 most important features
    print(i, sep='\n')
```

```
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=100; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=150; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=150; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=150; total time=   0.2s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=200; total time=   0.3s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=200; total time=   0.3s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=250; total time=   0.4s
[CV] END bootstrap=False, max_depth=100, max_features=8, min_samples_
leaf=3, min_samples_split=7, n_estimators=250; total time=   0.4s
```

DNN

In [280]:
```
num_feats = len(all_feats)
tf.random.set_seed(1)
def create_DNN(unit):
    """creating DNN architechture """
    model = keras.Sequential()
    model.add(keras.layers.Dense(unit, input_dim=num_feats, activatior
    model.add(keras.layers.Dense(unit, activation='relu'))
    model.add(keras.layers.Dropout(0.5))
    model.add(keras.layers.Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics
    return model
```

In [281]:
```
from keras.wrappers.scikit_learn import KerasRegressor
model=KerasRegressor(build_fn=create_DNN)
```

In [282]:
```
params = {'batch_size':[20, 40, 60, 80, 100],
          'epochs':[100, 200, 300, 400],
          'unit':[5, 10, 15, 20, 25]
           }
gs = GridSearchCV(estimator=model, param_grid=params, cv=10)
gs_result = gs.fit(train_X, train_y)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 46.0353 -
root_mean_squared_error: 6.7849
Epoch 285/300
5/5 [==============================] - 0s 2ms/step - loss: 44.8603 -
root_mean_squared_error: 6.6978
Epoch 286/300
5/5 [==============================] - 0s 2ms/step - loss: 51.1428 -
root_mean_squared_error: 7.1514
Epoch 287/300
5/5 [==============================] - 0s 2ms/step - loss: 38.5023 -
root_mean_squared_error: 6.2050
Epoch 288/300
5/5 [==============================] - 0s 2ms/step - loss: 43.3371 -
root_mean_squared_error: 6.5831
Epoch 289/300
5/5 [==============================] - 0s 2ms/step - loss: 47.4798 -
root_mean_squared_error: 6.8906
Epoch 290/300
5/5 [==============================] - 0s 2ms/step - loss: 47.3309 -
root_mean_squared_error: 6.8797
```

In [286]:
```
best_params=gs_result.best_params_
accuracy=gs_result.best_score_

best_params
```

Out[286]:
```
{'batch_size': 80, 'epochs': 400, 'unit': 25}
```

DNN 2

In [299]:
```
def create_DNN2(unit):
    """creating DNN architechture """
    model = keras.Sequential()
    model.add(keras.layers.Dense(unit, input_dim=num_feats, activation
    model.add(keras.layers.Dense(unit, activation='relu'))
    model.add(keras.layers.Dropout(0.6))
    model.add(keras.layers.Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics
    return model

model=KerasRegressor(build_fn=create_DNN2)

params = {'batch_size':[10, 20, 30],
          'epochs':[100, 200],
          'unit':[30, 50, 70]
           }
gs = GridSearchCV(estimator=model, param_grid=params, cv=10)
gs_result = gs.fit(train_X, train_y)
```

```
Epoch 136/200
17/17 [==============================] - 0s 2ms/step - loss: 18.0993
- root_mean_squared_error: 4.2543
Epoch 137/200
17/17 [==============================] - 0s 1ms/step - loss: 17.8102
- root_mean_squared_error: 4.2202
Epoch 138/200
17/17 [==============================] - 0s 2ms/step - loss: 17.9674
- root_mean_squared_error: 4.2388
Epoch 139/200
17/17 [==============================] - 0s 1ms/step - loss: 17.7287
- root_mean_squared_error: 4.2105
Epoch 140/200
17/17 [==============================] - 0s 2ms/step - loss: 19.7838
- root_mean_squared_error: 4.4479
Epoch 141/200
17/17 [==============================] - 0s 2ms/step - loss: 18.8718
- root_mean_squared_error: 4.3442
Epoch 142/200
17/17 [==============================] - 0s 2ms/step - loss: 18.5522
```

In [300]:
```
gs_result.best_params_
```

Out[300]:
```
{'batch_size': 20, 'epochs': 100, 'unit': 50}
```

Kaggle Submission

In [244]:
```
test = pd.read_csv("BUDStest.csv", index_col = 0)
test = pd.get_dummies(data=test, columns=cat_features)
test.head()
```

Out[244]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	he
ID												
620	17	1	1	1	2	0	4	3	4	1	2	
571	17	3	4	1	3	0	4	4	5	1	3	
215	17	2	2	2	1	0	4	4	4	2	3	
123	18	3	2	1	3	0	4	3	3	5	1	
46	18	2	4	1	2	1	2	3	2	1	3	

Random Forest Submission

In [240]:
```
best_params = {'bootstrap': False,
                'max_depth': 80,
                'max_features': 8,
                'min_samples_leaf': 1,
                'min_samples_split': 6,
                'n_estimators': 300}

rf = RandomForestRegressor(**best_params, random_state = 1)
rf.fit(train_X, train_y)
grade = rf.predict(test)
```

In [247]:
```
my_submission = pd.DataFrame({'ID': test.index, 'grade': grade})
# you could use any filename. We choose submission here
my_submission.to_csv('submission.csv', index=False)
```

DNN Submission

In [288]:
```
best_params = {'batch_size': 80, 'epochs': 400, 'unit': 25}


def create_best_DNN(unit):
    """creating DNN architechture """
    model = keras.Sequential()
    model.add(keras.layers.Dense(unit, input_dim=num_feats, activation
    model.add(keras.layers.Dense(unit, activation='relu'))
    model.add(keras.layers.Dropout(0.5))
    model.add(keras.layers.Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics
    return model

DNN = create_best_DNN(25)
DNN.fit(train_X, train_y, epochs=400, batch_size=80, verbose=1, valida
score = DNN.evaluate(val_X, val_y, verbose=1)
```

```
root_mean_squared_error: 3.5606 - val_loss: 7.6033 - val_root_mean_sq
uared_error: 2.7574
Epoch 280/400
5/5 [==============================] - 0s 15ms/step - loss: 11.5954 -
root_mean_squared_error: 3.4052 - val_loss: 8.1153 - val_root_mean_sq
uared_error: 2.8487
Epoch 281/400
5/5 [==============================] - 0s 8ms/step - loss: 14.6323 -
root_mean_squared_error: 3.8252 - val_loss: 8.6872 - val_root_mean_sq
uared_error: 2.9474
Epoch 282/400
5/5 [==============================] - 0s 9ms/step - loss: 12.2309 -
root_mean_squared_error: 3.4973 - val_loss: 8.2291 - val_root_mean_sq
uared_error: 2.8686
Epoch 283/400
5/5 [==============================] - 0s 8ms/step - loss: 12.5928 -
root_mean_squared_error: 3.5486 - val_loss: 7.5136 - val_root_mean_sq
uared_error: 2.7411
Epoch 284/400
```

In [296]:
```
grade = DNN.predict(test)
grade = grade.reshape(len(grade))
```

In [297]:
```
my_submission = pd.DataFrame({'ID': test.index, 'grade': grade})
# you could use any filename. We choose submission here
my_submission.to_csv('submission.csv', index=False)
```

DNN Submission 2

In [332]:
```python
#{'batch_size': 20, 'epochs': 100, 'unit': 50}

def create_best_DNN2(unit):
    """creating DNN architechture """
    model = keras.Sequential()
    model.add(keras.layers.Dense(unit, input_dim=num_feats, activatior
    model.add(keras.layers.Dense(unit, activation='relu'))
    model.add(keras.layers.Dense(unit, activation='relu'))
    model.add(keras.layers.Dropout(0.5))
    model.add(keras.layers.Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics
    return model

DNN = create_best_DNN2(60)
DNN.fit(train_X, train_y, epochs=100, batch_size=20, verbose=1, valida
score = DNN.evaluate(val_X, val_y, verbose=1)
```

```
Epoch 21/100
19/19 [==============================] - 0s 4ms/step - loss: 13.5862
- root_mean_squared_error: 3.6860 - val_loss: 6.7765 - val_root_mean_
squared_error: 2.6032
Epoch 22/100
19/19 [==============================] - 0s 3ms/step - loss: 13.8045
- root_mean_squared_error: 3.7154 - val_loss: 7.4718 - val_root_mean_
squared_error: 2.7335
Epoch 23/100
19/19 [==============================] - 0s 4ms/step - loss: 13.0117
- root_mean_squared_error: 3.6072 - val_loss: 6.9234 - val_root_mean_
squared_error: 2.6312
Epoch 24/100
19/19 [==============================] - 0s 3ms/step - loss: 12.7710
- root_mean_squared_error: 3.5737 - val_loss: 6.8737 - val_root_mean_
squared_error: 2.6218
Epoch 25/100
19/19 [==============================] - 0s 3ms/step - loss: 12.6497
- root_mean_squared_error: 3.5566 - val_loss: 7.0375 - val_root_mean_
squared_error: 2.6528
```

In [333]:
```python
score
```

Out[333]:
```
[7.08127498626709, 2.6610665321350098]
```

In [314]:
```python
grade = DNN.predict(test)
grade = grade.reshape(len(grade))
```

In [315]:
```python
my_submission = pd.DataFrame({'ID': test.index, 'grade': grade})
# you could use any filename. We choose submission here
my_submission.to_csv('submission.csv', index=False)
```

For next time

Tensorboard and early stopping. Batch Norm? Callback? How to grid search faster?

```
In [ ]:
```