Bayesian_Ridge

October 29, 2019

In [1]:
```
import pandas as pd
from matplotlib import pyplot
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy import stats
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import BayesianRidge
%matplotlib inline
```

In [2]:
```
df = pd.read_csv("student-por.csv", delimiter=";")
df.head()
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...
1	GP	F	17	U	GT3	T	1	1	at_home	other	...
2	GP	F	15	U	LE3	T	1	1	at_home	other	...
3	GP	F	15	U	GT3	T	4	2	health	services	...
4	GP	F	16	U	GT3	T	3	3	other	other	...

	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	4	3	4	1	1	3	4	0	11	11
1	5	3	3	1	1	3	2	9	11	11
2	4	3	2	2	3	3	6	12	13	12
3	3	2	2	1	1	5	0	14	14	14
4	4	3	2	1	2	5	0	11	13	13

[5 rows x 33 columns]

In [3]:
```
df.shape
```

Out[3]:
```
(649, 33)
```

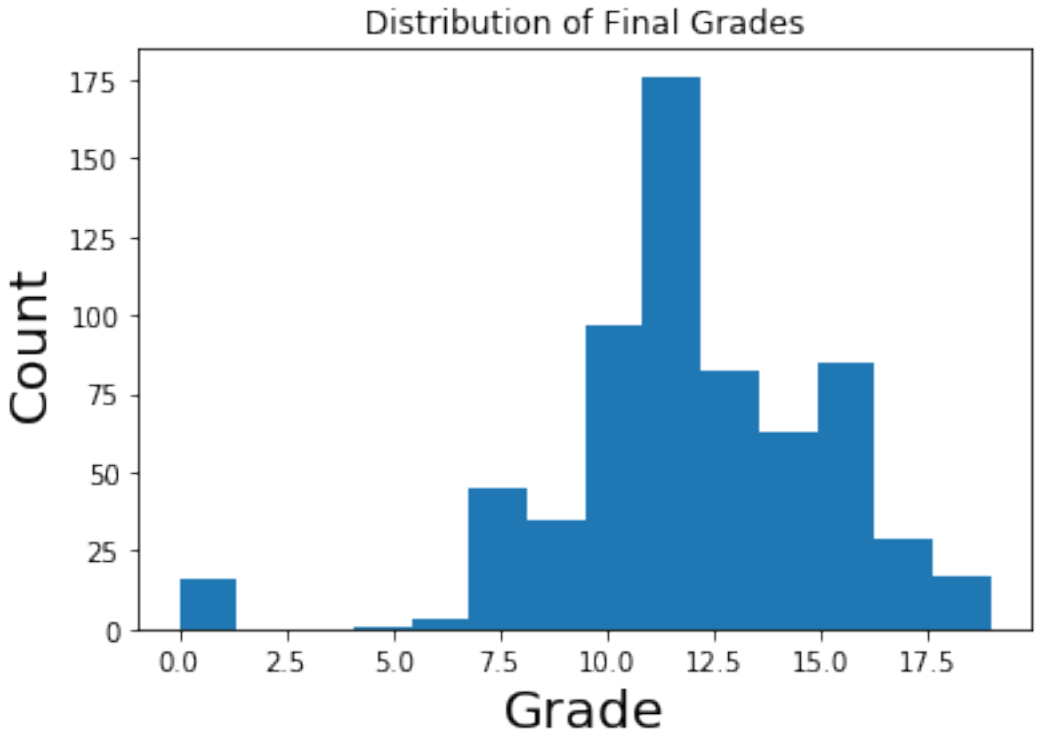
In [4]:
```
df.columns
```

Out[4]:
```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
```

```
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

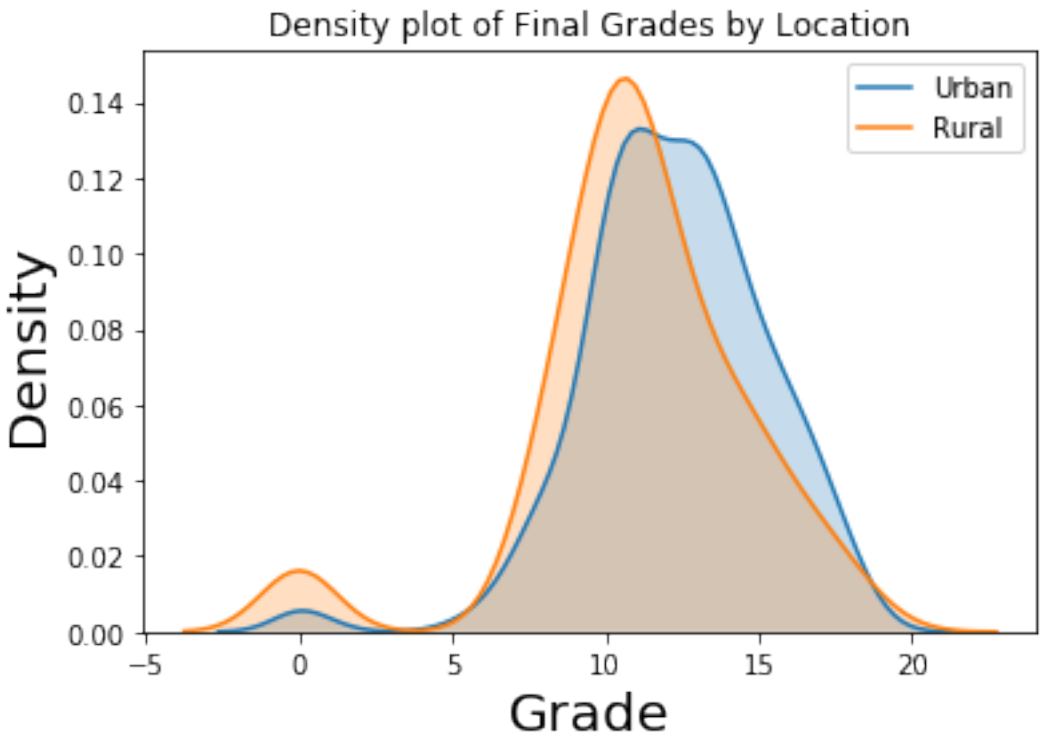
In [5]:
```
pyplot.hist(df["G3"], bins = 14)
pyplot.xlabel("Grade", fontsize=20)
pyplot.ylabel("Count", fontsize=20)
pyplot.title("Distribution of Final Grades")
```

Out[5]:
```
Text(0.5, 1.0, 'Distribution of Final Grades')
```

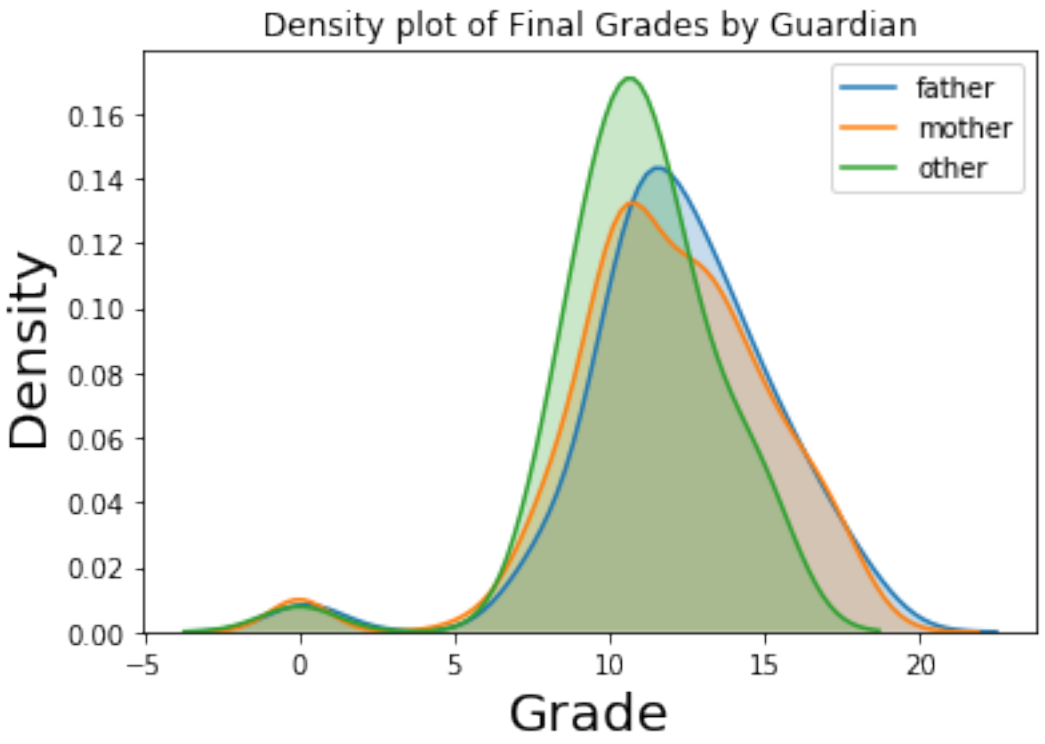


In [6]:
```
sns.kdeplot(df.loc[df["address"] == "U", "G3"], label="Urban", shade = True)
sns.kdeplot(df.loc[df["address"] == "R", "G3"], label="Rural", shade = True)
pyplot.xlabel("Grade", fontsize=20)
pyplot.ylabel("Density", fontsize=20)
pyplot.title("Density plot of Final Grades by Location")
```

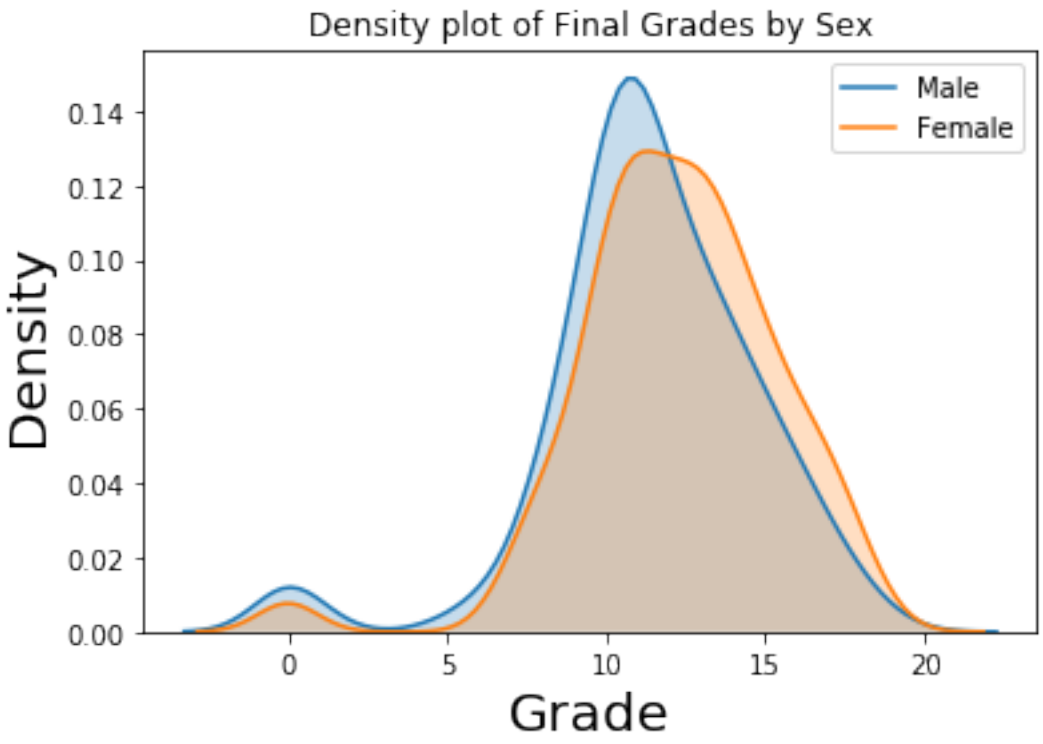
Out[6]:
```
Text(0.5, 1.0, 'Density plot of Final Grades by Location')
```



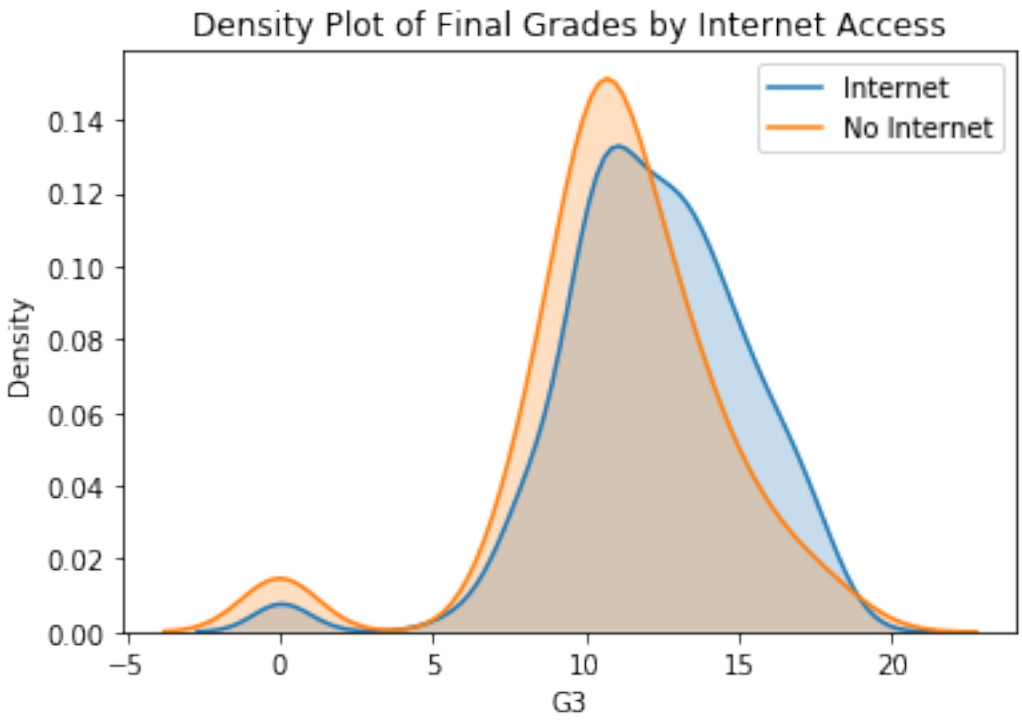

```
In [7]: sns.kdeplot(df.loc[df["guardian"] == "father", "G3"], label="father", shade = True)
        sns.kdeplot(df.loc[df["guardian"] == "mother", "G3"], label="mother", shade = True)
        sns.kdeplot(df.loc[df["guardian"] == "other", "G3"], label="other", shade = True)
        pyplot.xlabel("Grade", fontsize=20)
        pyplot.ylabel("Density", fontsize=20)
        pyplot.title("Density plot of Final Grades by Guardian")

Out[7]: Text(0.5, 1.0, 'Density plot of Final Grades by Guardian')
```




```
In [8]: sns.kdeplot(df.loc[df["sex"] == "M", "G3"], label="Male", shade = True)
        sns.kdeplot(df.loc[df["sex"] == "F", "G3"], label="Female", shade = True)
        pyplot.xlabel("Grade", fontsize=20)
        pyplot.ylabel("Density", fontsize=20)
        pyplot.title("Density plot of Final Grades by Sex")

Out[8]: Text(0.5, 1.0, 'Density plot of Final Grades by Sex')
```




```
In [9]: sns.kdeplot(df.loc[df['internet'] == 'yes', 'G3'], label = 'Internet', shade = True)
        sns.kdeplot(df.loc[df['internet'] == 'no', 'G3'], label = 'No Internet', shade = True)
        pyplot.xlabel('G3')
        pyplot.ylabel('Density')
        pyplot.title('Density Plot of Final Grades by Internet Access')

Out[9]: Text(0.5, 1.0, 'Density Plot of Final Grades by Internet Access')
```




```
In [10]: # Look at distribution of schools by address
         schools = df.groupby(['school'])['address'].value_counts()
         schools

Out[10]: school  address
         GP      U          345
                 R           78
         MS      R          119
                 U          107
         Name: address, dtype: int64

In [11]: #Find Correlations and sort
         df.corr()["G3"].sort_values()

Out[11]: failures     -0.393316
         Dalc         -0.204719
         Walc         -0.176619
         traveltime   -0.127173
         freetime     -0.122705
         age          -0.106505
         health       -0.098851
         absences     -0.091379
         goout        -0.087641
         famrel        0.063361
```

```
Fedu          0.211800
Medu          0.240151
studytime     0.249789
G1            0.826387
G2            0.918548
G3            1.000000
Name: G3, dtype: float64
```

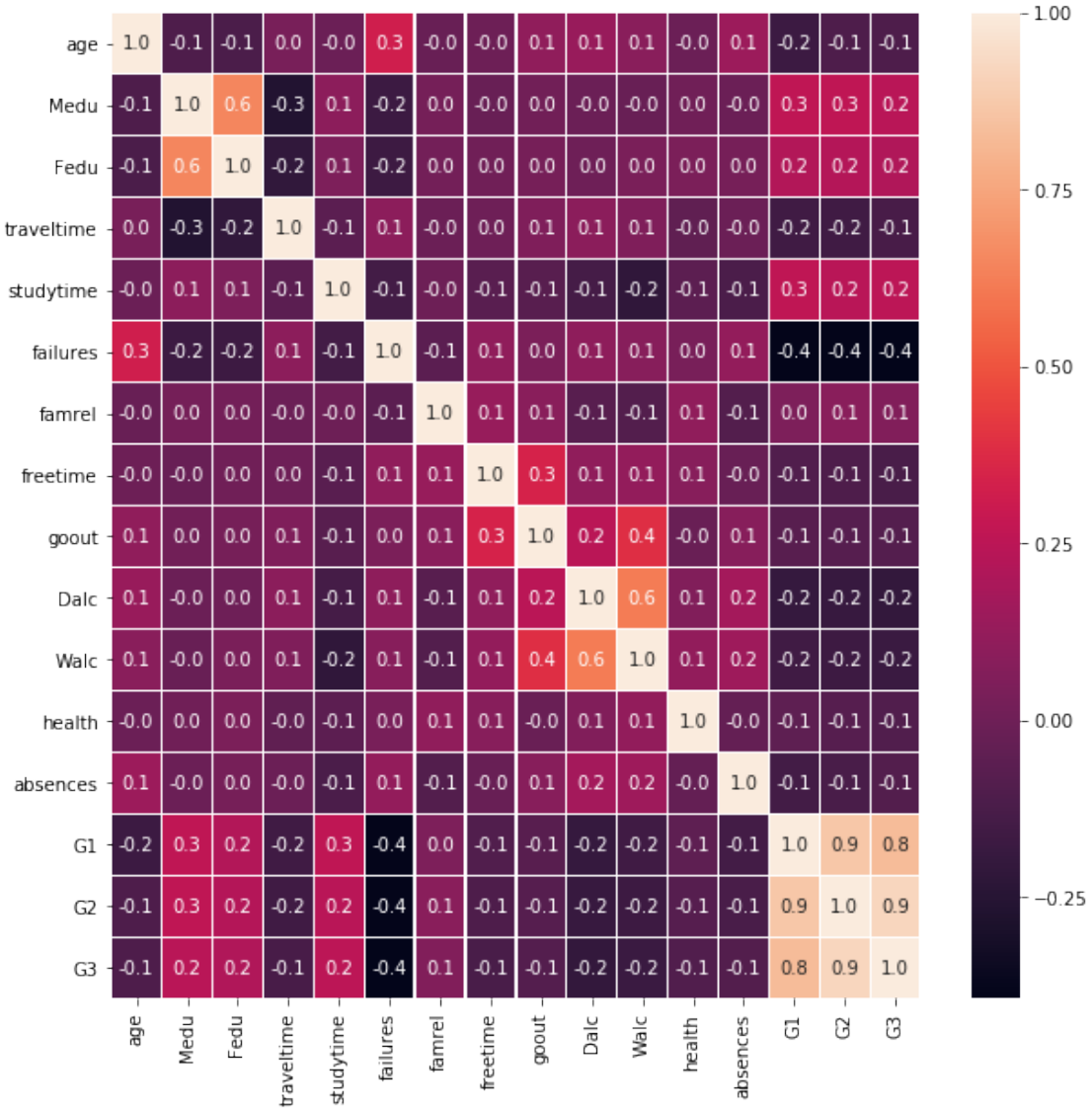
1 One-hot encoding

In [12]:
```
# Select only categorical variables
category_df = df.select_dtypes('object')

# One hot encode the variables
dummy_df = pd.get_dummies(category_df)

# Put the grade back in the dataframe
dummy_df['G3'] = df['G3']

# Find correlations with grade
dummy_df.corr()['G3'].sort_values()
```

Out[12]:
```
higher_no          -0.332172
school_MS          -0.284294
address_R          -0.167637
internet_no        -0.150025
Mjob_at_home       -0.136778
reason_other       -0.132577
sex_M              -0.129077
reason_course      -0.098305
romantic_yes       -0.090583
guardian_other     -0.080729
schoolsup_yes      -0.066405
activities_no      -0.059791
Mjob_other         -0.059251
famsup_no          -0.059206
paid_yes           -0.054898
Fjob_services      -0.053204
famsize_GT3        -0.045016
Fjob_at_home       -0.038904
nursery_no         -0.028752
Fjob_other         -0.005301
guardian_mother    -0.004415
Pstatus_T          -0.000754
Pstatus_A           0.000754
nursery_yes         0.028752
Mjob_services       0.038447
```

```
Fjob_health          0.039142
famsize_LE3          0.045016
reason_home          0.046537
guardian_father      0.051030
paid_no              0.054898
famsup_yes           0.059206
activities_yes       0.059791
schoolsup_no         0.066405
romantic_no          0.090583
Mjob_health          0.101244
Fjob_teacher         0.125916
sex_F                0.129077
Mjob_teacher         0.134910
internet_yes         0.150025
address_U            0.167637
reason_reputation    0.170944
school_GP            0.284294
higher_yes           0.332172
G3                   1.000000
Name: G3, dtype: float64
```

In [13]:
```
f, ax = pyplot.subplots(figsize = (10, 10))
sns.heatmap(df.corr(), annot = True, linewidth = 0.3, fmt = ".1f", ax = ax)
pyplot.show()
```




```
In [14]: def format_data(df):
             # Targets are final grade of student
             labels = df['G3']

             # Drop the school and the grades from features
             df = df.drop(columns=['school', 'G1', 'G2'])

             # One-Hot Encoding of Categorical Variables
             df = pd.get_dummies(df)

             # Find correlations with the Grade
             most_correlated = df.corr().abs()['G3'].sort_values(ascending=False)
```

```
    # Maintain the top 6 most correlation features with Grade
    most_correlated = most_correlated[:8]

    df = df.loc[:, most_correlated.index]
    df = df.drop(columns = 'higher_no')

    # Split into training/testing sets with 25% split
    X_train, X_test, y_train, y_test = train_test_split(df, labels,
                                                        test_size = 0.25,
                                                        random_state=42)

    return X_train, X_test, y_train, y_test
```

In [15]:
```
X_train, X_test, y_train, y_test = format_data(df)
X_train.head()
```

Out[15]:

	G3	failures	higher_yes	studytime	Medu	Fedu	Dalc
213	11	0	1	2	4	4	1
43	10	0	1	1	2	2	1
42	15	0	1	2	4	4	1
73	14	0	1	1	3	1	2
494	9	0	0	2	1	2	1

In [16]:
```
# Rename variables in train and teste
X_train = X_train.rename(columns={'higher_yes': 'higher_edu',
                                  'Medu': 'mother_edu',
                                  'Fedu': 'father_edu',
                                  "Dalc": "workday_alcohol_consumption"})

X_test = X_test.rename(columns={'higher_yes': 'higher_edu',
                                  'Medu': 'mother_edu',
                                  'Fedu': 'father_edu',
                                  "Dalc": "workday_alcohol_consumption"})
```

In [17]:
```
X_train.head()
```

Out[17]:

	G3	failures	higher_edu	studytime	mother_edu	father_edu	\
213	11	0	1	2	4	4	
43	10	0	1	1	2	2	
42	15	0	1	2	4	4	
73	14	0	1	1	3	1	
494	9	0	0	2	1	2	

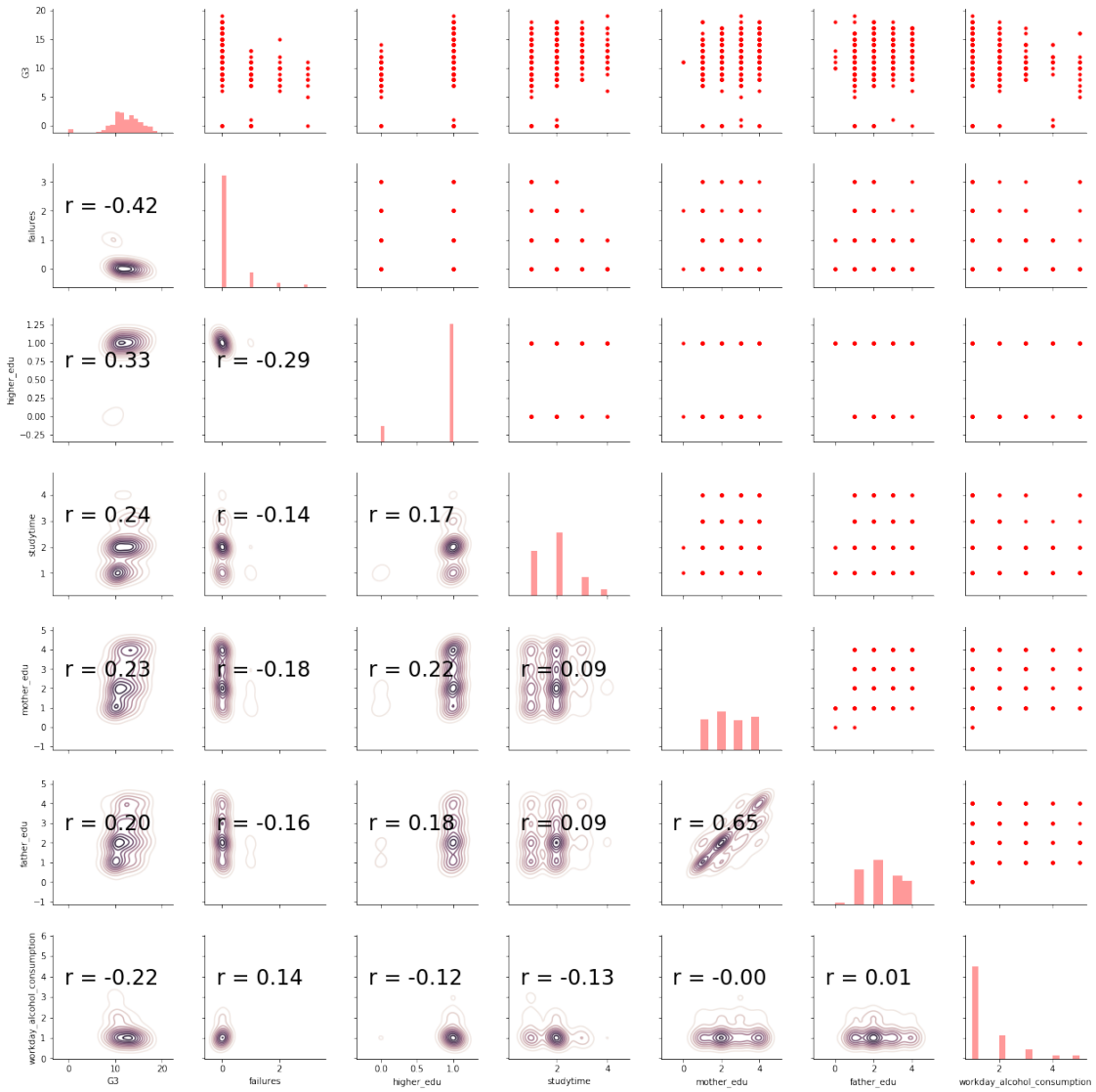
	workday_alcohol_consumption
213	1
43	1
42	1
73	2
494	1

In [18]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(486, 7)
(163, 7)
```

In [19]:
```
# Calculate correlation coefficient
def corrfunc(x, y, **kws):
    r, _ = stats.pearsonr(x, y)
    ax = pyplot.gca()
    ax.annotate("r = {:.2f}".format(r),
                xy=(.1, .6), xycoords=ax.transAxes,
                size = 24)

cmap = sns.cubehelix_palette(light=1, dark = 0.1,
                             hue = 0.5, as_cmap=True)

sns.set_context(font_scale=2)

# Pair grid set up
g = sns.PairGrid(X_train)

# Scatter plot on the upper triangle
g.map_upper(pyplot.scatter, s=10, color = 'red')

# Distribution on the diagonal
g.map_diag(sns.distplot, kde=False, color = 'red')

# Density Plot and Correlation coefficients on the lower triangle
g.map_lower(sns.kdeplot, cmap = cmap)
g.map_lower(corrfunc);
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tup
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




```
In [20]: def evaluate(X_train, X_test, y_train, y_test):
             # Names of models
             model_name_list = ['Linear Regression', "Bayesian Ridge"]

             X_train = X_train.drop(columns='G3')
             X_test = X_test.drop(columns='G3')

             # Instantiate the models
             model1 = LinearRegression()
             model2 = BayesianRidge(compute_score=True)

             # Dataframe for results
             results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)
```

```
        # Train and predict with each model
        for i, model in enumerate([model1, model2]):
            model.fit(X_train, y_train)
            predictions = model.predict(X_test)
            coeff = model.coef_
            print("{} Coefficients: ".format(model_name_list[i]), coeff)
            # Metrics
            mae = numpy.mean(abs(predictions - y_test))
            rmse = numpy.sqrt(numpy.mean((predictions - y_test) ** 2))

            # Insert results into the dataframe
            model_name = model_name_list[i]
            results.loc[model_name, :] = [mae, rmse]

        # Median Value Baseline Metrics
        baseline = numpy.mean(y_train)
        baseline_mae = numpy.mean(abs(baseline - y_test))
        baseline_rmse = numpy.sqrt(numpy.mean((baseline - y_test) ** 2))

        results.loc['Baseline', :] = [baseline_mae, baseline_rmse]

        return results
```

```
In [21]: results = evaluate(X_train, X_test, y_train, y_test)

Linear Regression Coefficients:  [-1.65372778  1.82066221  0.52901785  0.26480722  0.14815365 -0
Bayesian Ridge Coefficients:  [-1.61060828  1.57819002  0.53404974  0.27681423  0.15421188 -0.47


In [22]: print(results)

                     mae     rmse
Linear Regression  2.10147  2.81613
Bayesian Ridge     2.09346  2.81068
Baseline            2.3698  3.20491


In [ ]:
```