Part 2 Portuguese grades for both schools

In [ ]:
```
import pandas as pd #Importing necessary packages
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

In [ ]:
```
df = pd.read_csv('student-por.csv',sep=';')
df.head()
```

In [ ]:
```
df = df.drop(['G1','G2'],axis=1)
df.head()
#Dropping columns G1 and G2 in order to get a more precise prediction
```

In [ ]:
```
print('Number of rows:', len(df)) # Determine the how many rows are there
```

In [ ]:
```
df.isna().sum()
#Checking for Null values
```

In [ ]:
```
df.info()
```

In [ ]:
```
for var in ['traveltime','studytime','Medu','Fedu','famrel','freetime','goout','Dalc
    df[var] = df[var].astype('category')
df.info()
```

In [ ]:
```
num_col = df._get_numeric_data().columns
cat_col = list(set(df.columns)-set(num_col))
num_col = list(num_col)[0:3]
```

In [ ]:
```
nominal_col = ['higher','Fjob','address','guardian','school','paid','sex','Mjob','ac
ordinal_col = np.setdiff1d(cat_col,nominal_col)
```

In [ ]:
```
# change ordinal data to integer data format
for col in ordinal_col:
    df[col] = df[col].astype(int)
```

In [ ]:
```
# checking for erroneous data in categorical data
for col in df[cat_col]:
    print(col,df[col].unique())
```

In [ ]:
```
# checking for erroneous data in numerical data
df[num_col].describe()
```

In [ ]:
```
for i in df[num_col]:
    print(i,df[i].unique())
```

In [ ]:
```
# checking for erroneous data in G3
df['G3'].describe()
```

In [ ]:
```
df['G3'].unique()
```

Exploratory Data Analytics

In [ ]:
```
import seaborn as sns
sns.pairplot(df, hue = 'school')
```

In [ ]:
```
corrs = df.corr()
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corrs, annot=True, fmt='.2f', ax=ax)
plt.show()
```

In [ ]:
```
sns.boxplot(data=df, x='school', y='G3')
```

In [ ]:
```
df.groupby(by='school').mean()
```

Reducing skewness in numerical data

In [ ]:
```
skew_limit = 0.75

skew_cols = (df[num_col].skew()
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))

skew_cols # very high skewness in features
```

In [ ]:
```
field = "absences"
fig, (ax_original, ax_log1p) = plt.subplots(1, 2, figsize=(15, 5))

df[field].hist(ax=ax_original)

# Apply a log transformation (numpy syntax) to this column
df[field].apply(np.log1p).hist(ax=ax_log1p)

# Formatting of titles etc. for each subplot
ax_original.set(title='before np.log1p', ylabel='frequency', xlabel='value')
ax_log1p.set(title='after np.log1p', ylabel='frequency', xlabel='value')
fig.suptitle('Field "{}"'.format(field));
print('pop_orignial skewness: ',df[field].skew())
print('pop_log1p skewness: ',df[field].apply(np.log1p).skew())
# fall in skewness after log1p
```

In [ ]:
```
# apply log1p across all numerical columns

for col in df[skew_cols.index]:
# drop famrel as skewness increase after log1p
    df[col] = df[col].apply(np.log1p)
new_skew_cols = (df[skew_cols.index].skew()
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'}))

skew_cols['New_skew'] = new_skew_cols
skew_cols['Difference'] = abs(skew_cols['New_skew']) - abs(skew_cols['Skew'])
skew_cols
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
for col in num_col:
    df[col] = mm.fit_transform(df[[col]])
```

In [ ]:
```
df = pd.get_dummies(df, columns=nominal_col,drop_first=True)
df.head()
# df is the transformed dataest for ML, df is the orignal dataset
```

Polynomial Features

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
feature_cols = list(filter(lambda x: x!= 'G3', df.columns))

pf = PolynomialFeatures(degree=2, include_bias=False,)
X_pf = pf.fit_transform(df[feature_cols])
```

Train Test Split

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

df_train, df_test = train_test_split(df,
                                     train_size = 0.7,
                                     test_size = 0.3,
                                     random_state = 42)

feature_cols = list(filter(lambda x: x!= 'G3', df.columns))

X_train = df_train[feature_cols]
y_train = df_train['G3']

X_test = df_test[feature_cols]
y_test = df_test['G3']
```

In [ ]:
```
# splitting of Polynomial feautures
X_pf_train = X_pf[X_train.index]
y_pf_train = df['G3'][X_train.index]

X_pf_test = X_pf[X_test.index]
y_pf_test = df['G3'][X_test.index]
```

Model Kfold and evaluation

In [ ]:

```
from sklearn.model_selection import KFold, cross_val_predict
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV
from timeit import default_timer as timer # Calcuate time elapsed in training model

# define root mean sq error func
def rmse(ytrue, ypredicted):
    return np.sqrt(mean_squared_error(ytrue, ypredicted))

folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
```

In [ ]:

```
import statsmodels.api as sm
def diagnostic_plot(y_test, prediction):
    residual = y_test - prediction

    fig, axs = plt.subplots(2, 2,figsize=(15,10))
    fig.suptitle('Diagnostic plots')
    axs[0,0].plot([0, 15], [0, 20], ls="--", c="red", alpha=0.5)
    axs[0,0].scatter(y_test, prediction)
    axs[0,0].set_title('Truth-Prediction plot')
    axs[0,0].set_xlabel("Truth")
    axs[0,0].set_ylabel("Predictions")

    sm.qqplot(residual,fit=True,line= 's', ax=axs[0,1])
    axs[0,1].set_title('QQ plot')
    axs[0,1].set_xlabel("Theoretical Quantiles")
    axs[0,1].set_ylabel("Sample Quantiles")

    sns.residplot(prediction,y_test,
                              lowess=True,
                              scatter_kws={'alpha': 0.5},
                              line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8},
                              ax=axs[1,0])
    axs[1,0].set_title('Residual plot')
    axs[1,0].set_xlabel("Predicted")
    axs[1,0].set_ylabel("Residual")

    sqrt_standardized_residual=np.sqrt(np.abs(residual))
    sns.regplot(prediction, sqrt_standardized_residual,
                scatter=True,
                lowess=True,
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.5},
                ax=axs[1,1])
    axs[1,1].set_title('Scale-Location plot')
    axs[1,1].set_xlabel("Predicted")
    axs[1,1].set_ylabel("Sqrt Standarized residuals")
```

Dummy Model

In [ ]:

```
average_G3 = np.mean(y_test)
average_G3
```

In [ ]:

```
start = timer()
```

```
dummy_prediction =[]
for row in range(len(y_test)):
    dummy_prediction.append(average_G3)

print("R2 score: ",r2_score(y_test, dummy_prediction))
print('RMSE: ', rmse(y_test,dummy_prediction))

end = timer()
dummy_time = end - start
print('Time Elapsed (sec):', dummy_time)
```

Base Linear Regression

In [ ]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm_cv = GridSearchCV(estimator = lm,
                        param_grid = {},
                        cv = folds)
start = timer()
# fit the model
lm_cv.fit(X_train, y_train)

end = timer()
lm_time = end - start
print('Time Elapsed (sec):', lm_time)
```

In [ ]:
```
lm_prediction = lm_cv.predict(X_test)

print("R2 score: ",r2_score(y_test, lm_prediction))
print('RMSE: ', rmse(y_test,lm_prediction))
```

In [ ]:
```
diagnostic_plot(y_test, lm_prediction)
```

Linear Regression using polynomial features

In [ ]:
```
lm = LinearRegression()
lm_pf_cv = GridSearchCV(estimator = lm,
                        param_grid = {},
                        cv = folds)
start = timer()
# fit the model
lm_pf_cv.fit(X_pf_train, y_pf_train)

end = timer()
lm_pf_time = end - start
print('Time Elapsed (sec):', lm_pf_time)
```

In [ ]:
```
lm_pf_prediction = lm_pf_cv.predict(X_pf_test)

print("R2 score: ",r2_score(y_pf_test, lm_pf_prediction))
print('RMSE: ', rmse(y_pf_test,lm_pf_prediction))
```

In [ ]:
```
diagnostic_plot(y_test, lm_pf_prediction)
```

Lasso Regression

In [ ]:
```
alphas = np.geomspace(0.001, 10, 30)
```

In [ ]:
```
from sklearn.linear_model import LassoCV

start = timer()
lasso_estimator = LassoCV(alphas=alphas,
                          cv=5).fit(X_train,y_train)
lasso_prediction = lasso_estimator.predict(X_test)

print('Alpha param: ', lasso_estimator.alpha_)
print("R2 score: ",r2_score(y_test, lasso_prediction))
print('RMSE: ', rmse(y_test,lasso_prediction))

end = timer()
lasso_time = end - start
print('Time Elapsed (sec):', lasso_time)
```

In [ ]:
```
print("Number of non-zero coeff: ", sum(lasso_estimator.coef_ != 0))
print("Mean coeff: ",sum(abs(lasso_estimator.coef_))/sum(lasso_estimator.coef_ != 0)
```

In [ ]:
```
print("\u0332".join('Feature_coefficient (Top)'),'\n')
features = []
coeffs = []
coeffs_abs = []
for feature, coeff in zip(X_test.columns, lasso_estimator.coef_):
    features.append(feature)
    coeffs.append(coeff)
    coeffs_abs.append(abs(coeff))

pd.DataFrame({'Features': features, 'Coefficients': coeffs ,'abs_coeffs':coeffs_abs}
```

In [ ]:
```
diagnostic_plot(y_test, lasso_prediction)
```

Ridge Regression

In [ ]:
```
alphas = np.geomspace(1, 100, 30)
```

In [ ]:
```
from sklearn.linear_model import RidgeCV

start = timer()
ridge_estimator = RidgeCV(alphas=alphas,
                          cv=5).fit(X_train,y_train)
ridge_prediction = ridge_estimator.predict(X_test)

print('Alpha param: ', ridge_estimator.alpha_)
print("R2 score: ",r2_score(y_test, ridge_prediction))
print('RMSE: ', rmse(y_test,ridge_prediction))

end = timer()
```

```
ridge_time = end - start
print('Time Elapsed (sec):', ridge_time)
```

In [ ]:

```
print("Number of non-zero coeff: ", sum(ridge_estimator.coef_ != 0))
print("Mean coeff: ",sum(abs(ridge_estimator.coef_))/sum(ridge_estimator.coef_ != 0)
```

In [ ]:

```
print("\u0332".join('Feature_coefficient (Top)'),'\n')
features = []
coeffs = []
coeffs_abs = []
for feature, coeff in zip(X_test.columns, ridge_estimator.coef_):
    features.append(feature)
    coeffs.append(coeff)
    coeffs_abs.append(abs(coeff))

pd.DataFrame({'Features': features, 'Coefficients': coeffs ,'abs_coeffs':coeffs_abs}
```

In [ ]:

```
diagnostic_plot(y_test, ridge_prediction)
```

Elasticnet Regression

In [ ]:

```
from sklearn.linear_model import ElasticNetCV
alphas = np.geomspace(0.01, 1, 30)
l1_ratios = np.linspace(0.1, 0.9, 9)

start = timer()
elasticNetCV = ElasticNetCV(alphas=alphas,
                            l1_ratio=l1_ratios,
                            cv=5).fit(X_train,y_train)
elasticNet_prediction = elasticNetCV.predict(X_test)

print('Alpha param: ', elasticNetCV.alpha_)
print('l1 ratio param: ', elasticNetCV.l1_ratio_)
print("R2 score: ",r2_score(y_test, elasticNet_prediction))
print('RMSE: ', rmse(y_test,elasticNet_prediction))

end = timer()
EN_time = end - start
print('Time Elapsed (sec):', EN_time)
```

In [ ]:

```
print("Number of non-zero coeff: ", sum(elasticNetCV.coef_ != 0))
print("Mean coeff: ",sum(abs(elasticNetCV.coef_))/sum(elasticNetCV.coef_ != 0))
```

In [ ]:

```
print("\u0332".join('Feature_coefficient (Top)'),'\n')
features = []
coeffs = []
coeffs_abs = []
for feature, coeff in zip(X_test.columns, elasticNetCV.coef_):
    features.append(feature)
    coeffs.append(coeff)
    coeffs_abs.append(abs(coeff))

pd.DataFrame({'Features': features, 'Coefficients': coeffs ,'abs_coeffs':coeffs_abs}
```

In [ ]:
```
diagnostic_plot(y_test, elasticNet_prediction)
```

Stepwise regression - Forward

In [ ]:
```
def forward_regression(X, y,
                       threshold_in,
                       verbose=False):
    initial_list = []
    included = list(initial_list)
    while True:
        changed=False
        exc = list(set(X.columns)-set(included))
        pval = pd.Series(index=exc)
        for column in exc:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[column]]))).f
            pval[column] = model.pvalues[column]
        best = pval.min()
        if best < threshold_in:
            best_feature = pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:17} with p-value {:.6}'.format(best_feature, best))

        if not changed:
            break

    return included

# Starting with all features, features are removed iteratively based on the p-value
def backward_regression(X, y,
                        threshold_out,
                        verbose=False):
    included=list(X.columns)
    while True:
        changed=False
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:17} with p-value {:.6}'.format(worst_feature, worst_pv
        if not changed:
            break
    return included
```

In [ ]:
```
start = timer()
forward_regression_features = forward_regression(X_train,y_train, threshold_in=0.05,
forward_regression_features
```

In [ ]:
```
lm = LinearRegression()
lm_forward = GridSearchCV(estimator = lm,
                          param_grid = {},
                          cv = folds)
```

```
# fit the model
lm_forward.fit(X_train[forward_regression_features], y_train)

end = timer()
lm_forward_time = end - start
print('Time Elapsed (sec):', lm_forward_time)
```

In [ ]:

```
lm_forward_prediction = lm_forward.predict(X_test[forward_regression_features])

print("R2 score: ",r2_score(y_test, lm_forward_prediction))
print('RMSE: ', rmse(y_test,lm_forward_prediction))
```

In [ ]:

```
diagnostic_plot(y_test, lm_forward_prediction)
```

Stepwise regression - Backward

In [ ]:

```
start = timer()
backward_regression_features = backward_regression(X_train,y_train, threshold_out=0.
```

In [ ]:

```
lm = LinearRegression()
lm_backward = GridSearchCV(estimator = lm,
                           param_grid = {},
                           cv = folds)

# fit the model
lm_backward.fit(X_train[backward_regression_features], y_train)

end = timer()
lm_backward_time = end - start
print('Time Elapsed (sec):', lm_backward_time)
```

In [ ]:

```
lm_backwards_prediction = lm_backward.predict(X_test[backward_regression_features])

print("R2 score: ",r2_score(y_test, lm_backwards_prediction))
print('RMSE: ', rmse(y_test,lm_backwards_prediction))
```

In [ ]:

```
diagnostic_plot(y_test, lm_backwards_prediction)
```

Random Forest Regressor

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(criterion = 'mse')
```

In [ ]:

```
# Number of features to consider at every split
n_features = len(df.columns)
max_features = ['auto', 'sqrt', 'log2']
# Maximum number of levels in tree
max_depth = [3]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
```

```
min_samples_leaf = [1, 2, 4]
# Create the random grid
param_test = {'max_features': max_features,
           'max_depth': max_depth,
           'min_samples_split': min_samples_split,
           'min_samples_leaf': min_samples_leaf}
print(param_test)
```

In [ ]:

```
rf_model = GridSearchCV(estimator = rf_model,
                        param_grid = param_test,
                        cv = folds)

# fit the model
start = timer()
rf_model.fit(X_train, y_train)

end = timer()
rf_time = end - start
print('Time Elapsed (sec):', rf_time)
```

In [ ]:

```
rf_model_prediction = rf_model.predict(X_test)

print('Best param: ', rf_model.best_params_)
print("R2 score: ",r2_score(y_test, rf_model_prediction))
print('RMSE: ', rmse(y_test,rf_model_prediction))
```

In [ ]:

```
diagnostic_plot(y_test, rf_model_prediction)
```

In [ ]:

```
feat_df = pd.DataFrame({'Feature':X_train.columns, 'Importance':rf_model.best_estima
feat_df = feat_df.sort_values(by = 'Importance', ascending=False)
feat_df.head()
```

In [ ]:

```
g = sns.barplot(data=feat_df, x='Feature', y= 'Importance')
for item in g.get_xticklabels():
    item.set_rotation(90)
```

In [ ]:

```
# Import tools needed for visualization
from sklearn.tree import export_graphviz


# Extract the small tree
tree_small = rf_model.best_estimator_.estimators_[5]
# Save the tree as a png image
export_graphviz(tree_small, out_file = 'portree.dot', feature_names = X_train.column
```

In [ ]:

```
from IPython.display import Image
Image(filename = 'portree.png')
```

In [ ]:

```
models_pred = [dummy_prediction, lm_prediction, lm_pf_prediction, lasso_prediction,
time_elapsed = [dummy_time, lm_time, lm_pf_time, lasso_time, ridge_time, EN_time, lm
rmse_vals = []
for pred in models_pred:
    rmse_vals.append(rmse(y_test, pred))
```

```
R2_score = []
for pred in models_pred:
    R2_score.append(r2_score(y_test, pred))

labels = ['Dummy', 'Linear', 'Linear + PF', 'Ridge', 'Lasso', 'ElasticNet','Stepwise

eval_df = pd.DataFrame({'RMSE':rmse_vals, 'R2 Score':R2_score, 'Time Elapsed':time_e
eval_df
```

In [ ]: