# Model_Building_Part_1

April 18, 2025

## 0.1 Importing libraries, loading data and displaying basic data info

[22]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

[23]:
```
# Load the dataset
current_path = os.getcwd()
parent_folder = os.path.dirname(current_path)
data_file_path = os.path.join(parent_folder, "Data", "Student_performance_data.
↪csv")

df = pd.read_csv(str(data_file_path))
```

[24]:
```
df.info()
```

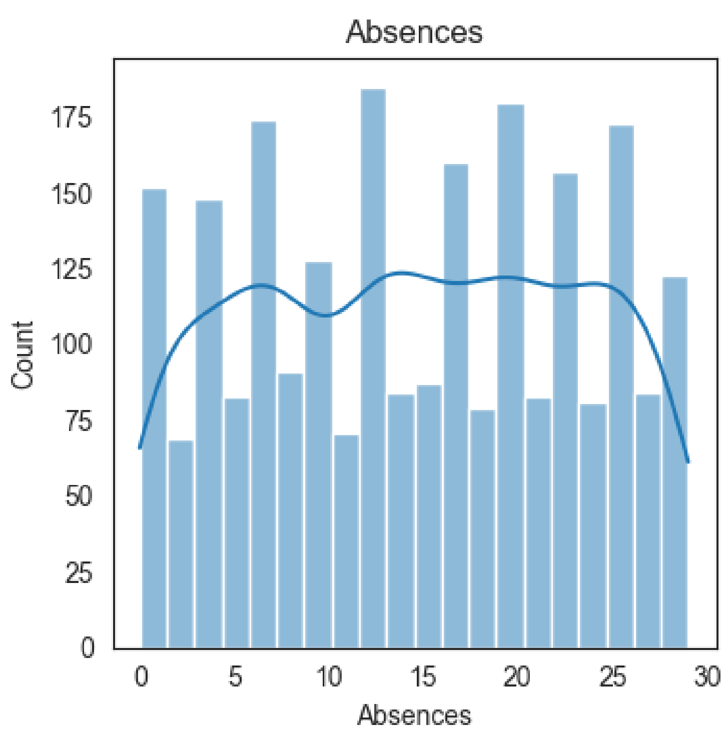
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   StudentID          2392 non-null   int64
 1   Age                2392 non-null   int64
 2   Gender             2392 non-null   int64
 3   Ethnicity          2392 non-null   int64
 4   ParentalEducation  2392 non-null   int64
 5   StudyTimeWeekly    2392 non-null   float64
 6   Absences           2392 non-null   int64
 7   Tutoring           2392 non-null   int64
 8   ParentalSupport    2392 non-null   int64
```

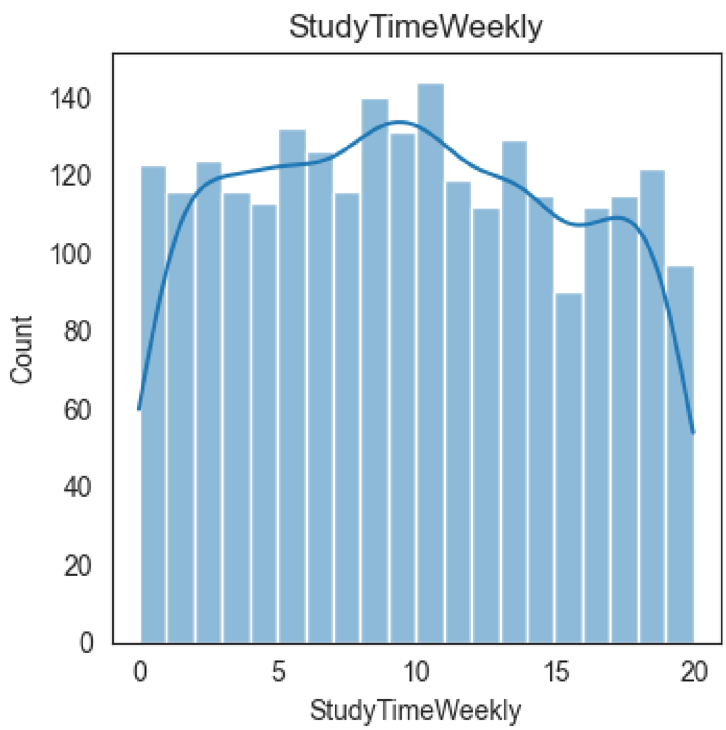
```
 9   Extracurricular    2392 non-null   int64
 10  Sports             2392 non-null   int64
 11  Music              2392 non-null   int64
 12  Volunteering       2392 non-null   int64
 13  GPA                2392 non-null   float64
 14  GradeClass         2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
```

## 0.2 Feature engineering

[25]:
```python
plt.figure(figsize=(4, 4))
sns.histplot(df['Absences'], kde=True, bins=20)
plt.title('Absences')
plt.tight_layout()
plt.show()

plt.figure(figsize=(4, 4))
sns.histplot(df['StudyTimeWeekly'], kde=True, bins=20)
plt.title('StudyTimeWeekly')
plt.tight_layout()
plt.show()
```






[26]:
```
df['Attendance'] = 1 - (df['Absences'] / 30) #basically just the opposite of␣
↪abscences data shows a max of 30 absences so divide by 30

df['Activity'] = df[['Extracurricular', 'Music', 'Sports', 'Volunteering']].
↪sum(axis=1) #all non academic activities, probably if students are more␣
↪involved they might do better academically

df['StudyTimeNorm'] = df['StudyTimeWeekly'] / 20 #study time massaged to be␣
↪between 0 and 1 (data shows a max of 20 hours per week so divide by 20
#here we define the ratios of used features to create the new feature
df['Engagement'] = (
    df['Attendance'] * 0.4 +
    df['Activity'] * 0.3 +
    df['StudyTimeNorm'] * 0.3
)
```

### 0.3 Checking for duplicates and viewing all input variables

[27]:
```
duplicates = df[df.duplicated()]
print(duplicates)
```

```
Empty DataFrame
Columns: [StudentID, Age, Gender, Ethnicity, ParentalEducation, StudyTimeWeekly,
```

```
Absences, Tutoring, ParentalSupport, Extracurricular, Sports, Music,
Volunteering, GPA, GradeClass, Attendance, Activity, StudyTimeNorm, Engagement]
Index: []
```

[28]:
```python
#lists input variables
all_vars = df.columns
outputvar_name = 'GradeClass'
inputvar_names = all_vars.drop(outputvar_name).tolist()

df_inputs = df[inputvar_names]
df_output = df[outputvar_name]

print(f'there are {len(inputvar_names)} Input variables')
df_inputs
```

```
there are 18 Input variables
```

[28]:
```
      StudentID  Age  Gender  Ethnicity  ParentalEducation  StudyTimeWeekly  \
0          1001   17       1          0                  2        19.833723
1          1002   18       0          0                  1        15.408756
2          1003   15       0          2                  3         4.210570
3          1004   17       1          0                  3        10.028829
4          1005   17       1          0                  2         4.672495
…             …    …       …          …                  …              …
2387       3388   18       1          0                  3        10.680555
2388       3389   17       0          0                  1         7.583217
2389       3390   16       1          0                  2         6.805500
2390       3391   16       1          1                  0        12.416653
2391       3392   16       1          0                  2        17.819907

      Absences  Tutoring  ParentalSupport  Extracurricular  Sports  Music  \
0            7         1                2                0       0      1
1            0         0                1                0       0      0
2           26         0                2                0       0      0
3           14         0                3                1       0      0
4           17         1                3                0       0      0
…            …         …                …                …       …      …
2387         2         0                4                1       0      0
2388         4         1                4                0       1      0
2389        20         0                2                0       0      0
2390        17         0                2                0       1      1
2391        13         0                2                0       0      0

      Volunteering       GPA  Attendance  Activity  StudyTimeNorm  Engagement
0                0  2.929196    0.766667         1       0.991686    0.904173
1                0  3.042915    1.000000         0       0.770438    0.631131
2                0  0.112602    0.133333         0       0.210528    0.116492
3                0  2.054218    0.533333         1       0.501441    0.663766
```

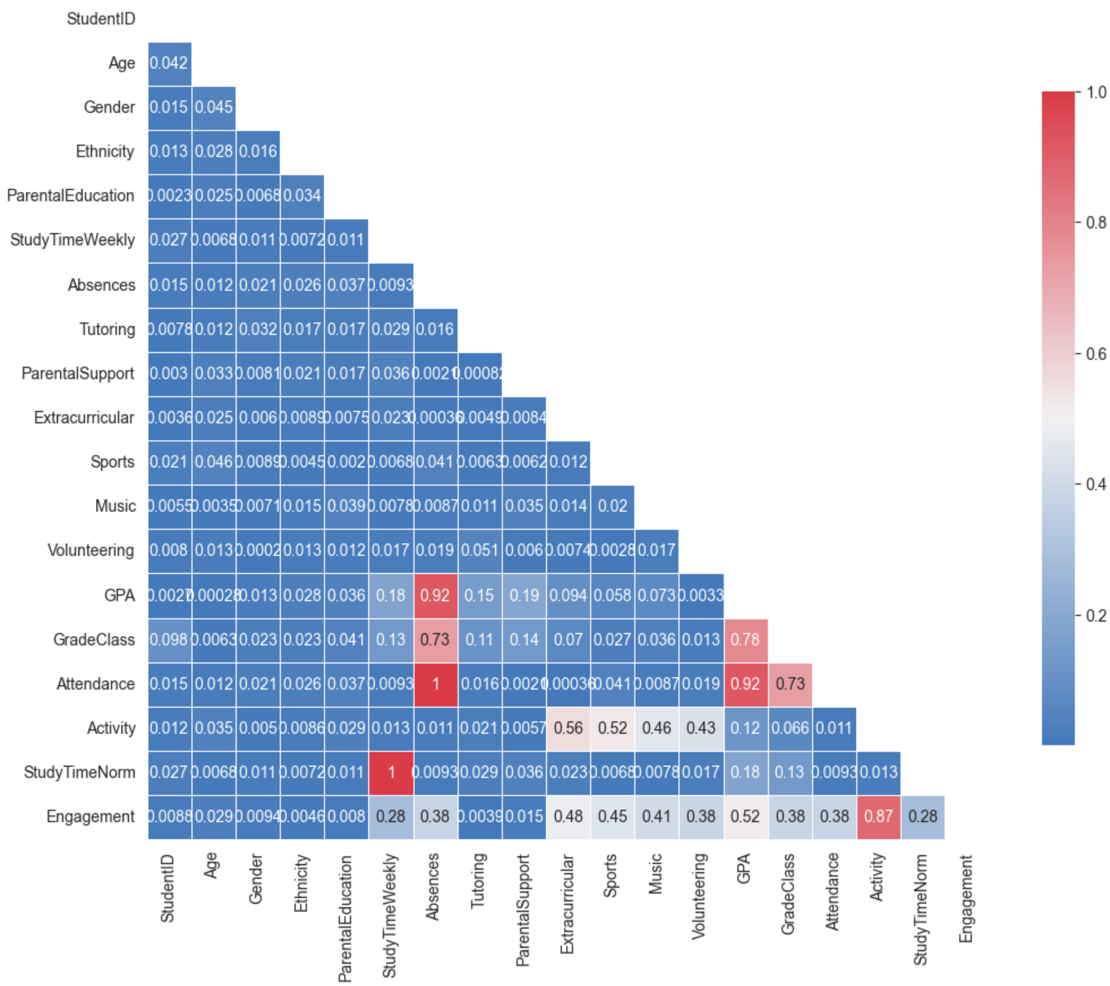
```
4                0  1.288061   0.433333       0       0.233625   0.243421
…             …      …       …        …           …         …
2387             0  3.455509   0.933333       1       0.534028   0.833542
2388             0  3.279150   0.866667       1       0.379161   0.760415
2389             1  1.142333   0.333333       1       0.340275   0.535416
2390             0  1.803297   0.433333       2       0.620833   0.959583
2391             1  2.140014   0.566667       1       0.890995   0.793965

[2392 rows x 18 columns]
```

### 0.4 Heatmap that examines the correlation between inputs and GradeClass

[29]:
```
def CorrPlot(df, dropDuplicates = True, figsize = (8, 6)):
    # df = df.corr()
    df = np.abs(df.corr())
    # Exclude duplicate correlations by masking upper right values
    if dropDuplicates:
        mask = np.zeros_like(df, dtype=bool)
        mask[np.triu_indices_from(mask)] = True
    # Set background color / chart style
    sns.set_style(style = 'white')
    # Set up  matplotlib figure
    f, ax = plt.subplots(figsize=figsize)
    # Add diverging colormap from red to blue
    cmap = sns.diverging_palette(250, 10, as_cmap=True)
    # Draw correlation plot with or without duplicates
    if dropDuplicates:
        sns.heatmap(df, mask=mask, cmap=cmap,
                    annot=True,
                    square=True,
                    linewidth=.5, cbar_kws={"shrink": .75}, ax=ax)
    else:
        sns.heatmap(df, cmap=cmap,
                square=True,
                annot=True,
                linewidth=.5, cbar_kws={"shrink": .5}, ax=ax)
```

[30]:
```
CorrPlot(df, figsize = (12, 10))
```

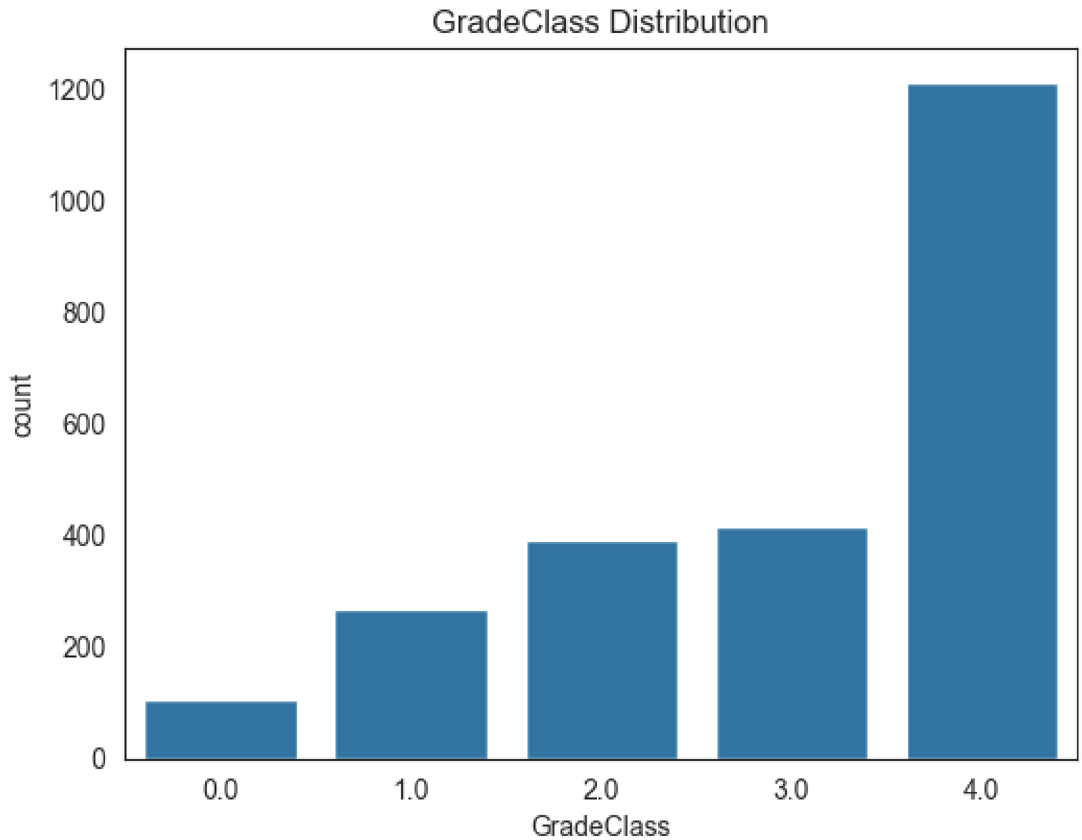


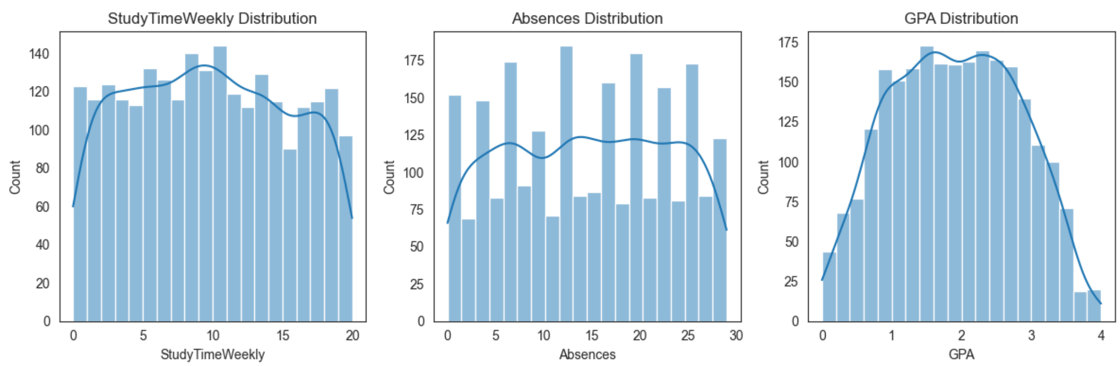

## 0.5 Univariate analysis

[31]:
```
#plots gradeClass
sns.countplot(x='GradeClass', data=df)
plt.title('GradeClass Distribution')
plt.show()

numeric_cols = ['StudyTimeWeekly', 'Absences', 'GPA']

plt.figure(figsize=(12, 4))
for i, col in enumerate(numeric_cols):
    plt.subplot(1, 3, i+1)
    sns.histplot(df[col], kde=True, bins=20)
    plt.title(f'{col} Distribution')
plt.tight_layout()
plt.show()
```



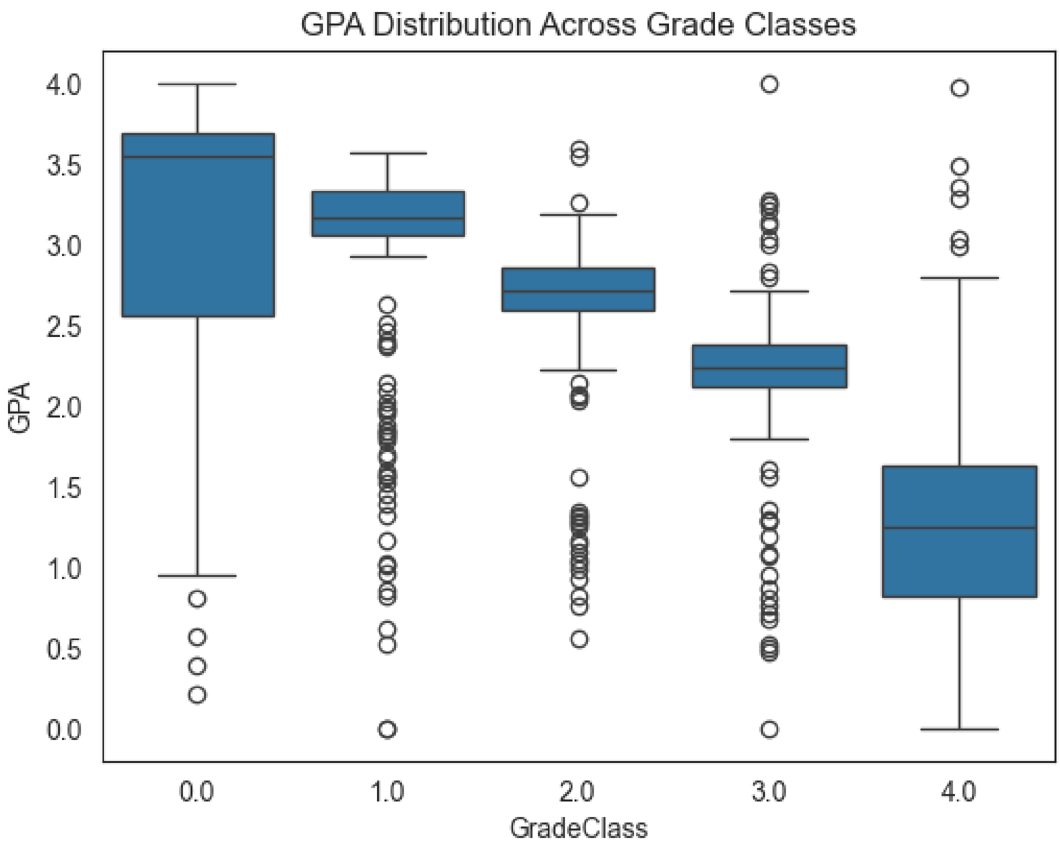



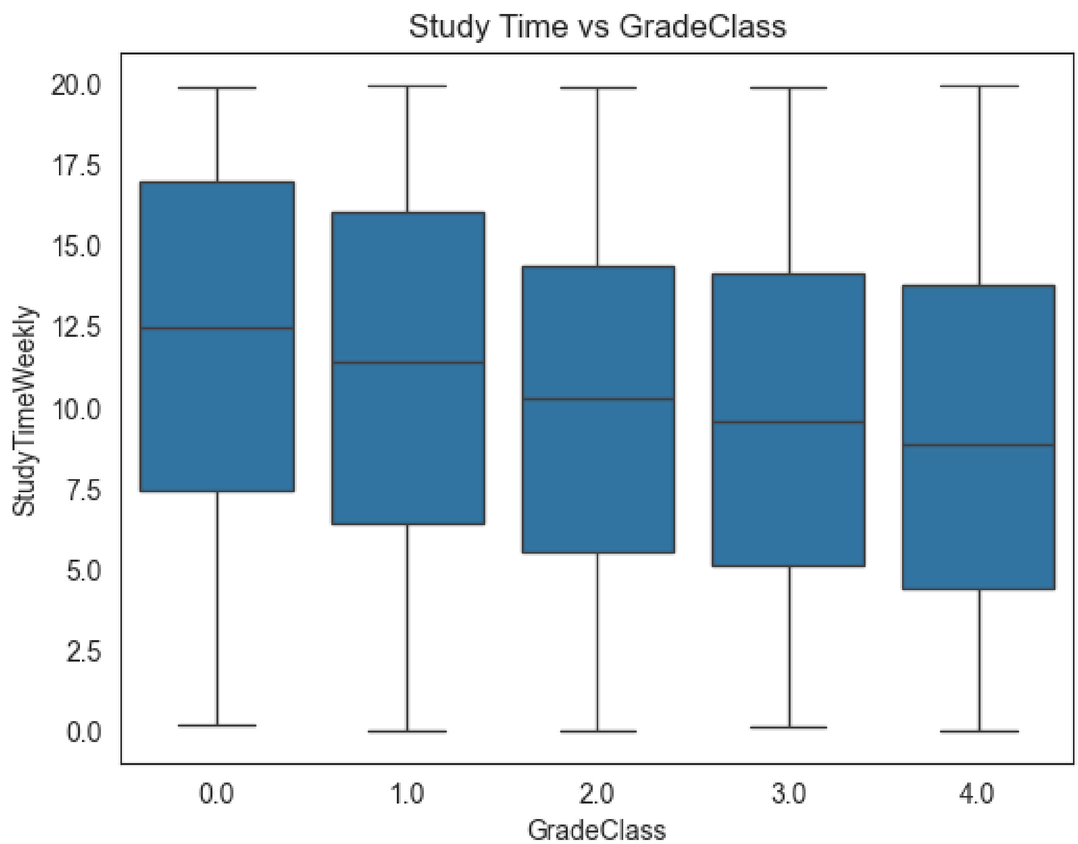

## 0.6 Bivariate analysis

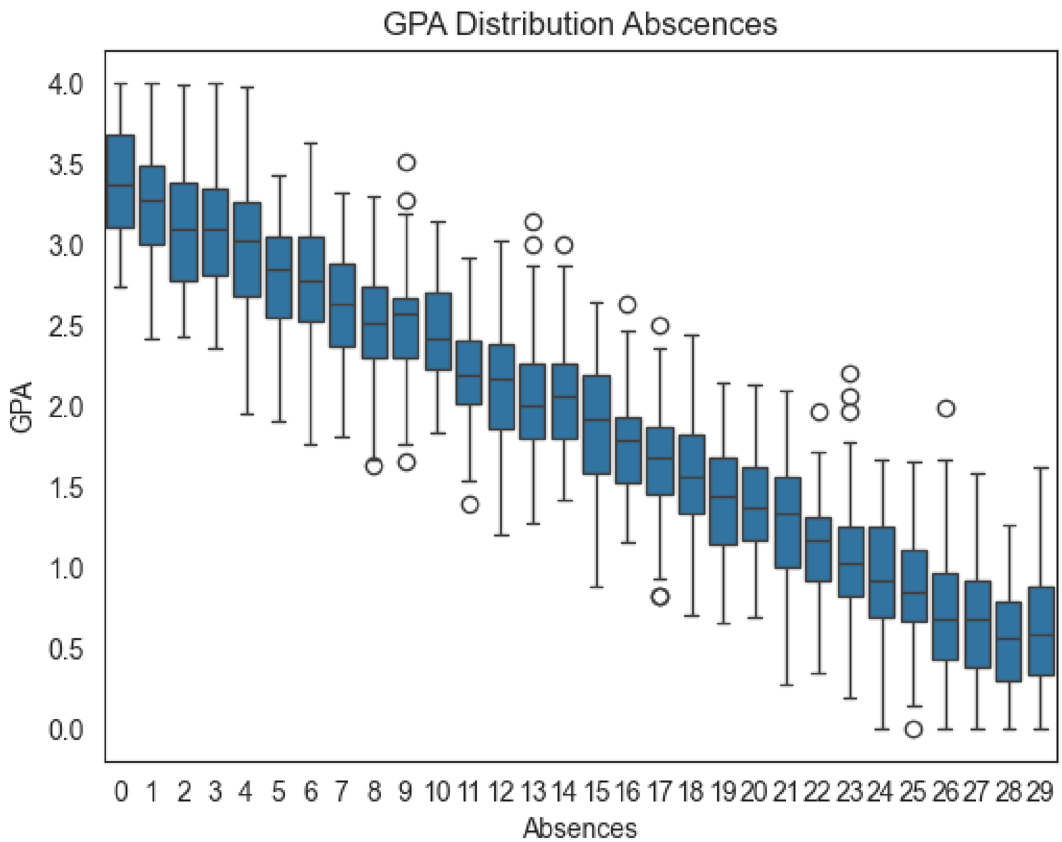
[32]:
```
# Boxplot of GPA across GradeClass
sns.boxplot(x='GradeClass', y='GPA', data=df)
plt.title("GPA Distribution Across Grade Classes")
plt.show()
```

```
# Barplot: StudyTimeWeekly vs GradeClass
sns.boxplot(x='GradeClass', y='StudyTimeWeekly', data=df)
plt.title("Study Time vs GradeClass")
plt.show()

# Boxplot of GPA across Abscenses
sns.boxplot(x='Absences', y='GPA', data=df)
plt.title("GPA Distribution Abscences")
plt.show()
```




Study Time vs GradeClass
StudyTimeWeekly
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
1.0
2.0
3.0
4.0
GradeClass



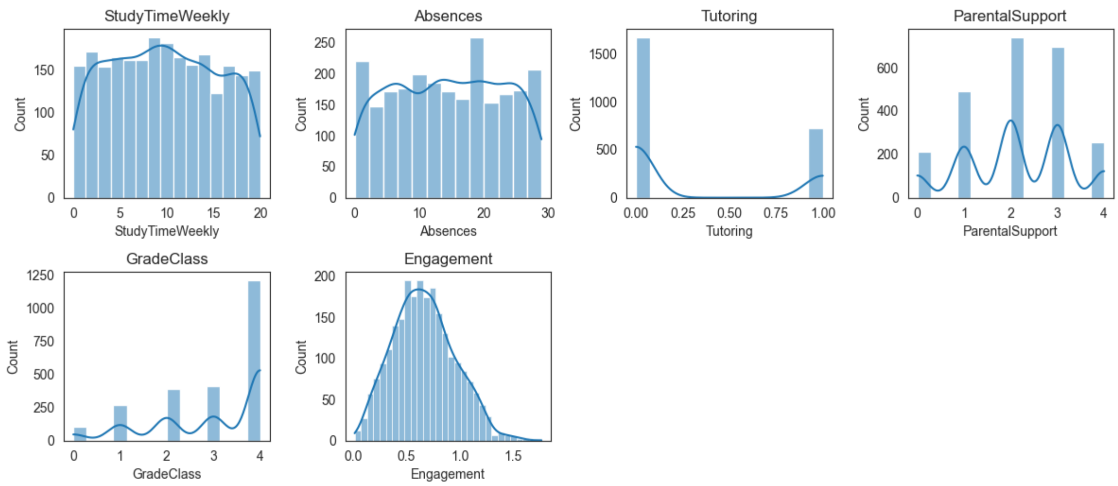

## 0.7 Dropping data that wont have an effect on GradeClass

drop GPA because it directly determines grade class

[33]:
```
df_cleaned = df.drop(columns=[
    'Age', 'Gender', 'Ethnicity', 'ParentalEducation',
    'Extracurricular', 'Music', 'Volunteering',
    'Sports', 'StudentID', 'GPA', 'StudyTimeNorm', 'Activity', 'Attendance'
])
```

## 0.8 Displays all data that wasnt dropped

[34]:
```
plt.figure(figsize=(12, 10))
for i, col in enumerate(df_cleaned.columns):
    plt.subplot(4, 4, i+1)
    sns.histplot(df_cleaned[col], kde=True)
    plt.title(col)
plt.tight_layout()
plt.show()
```



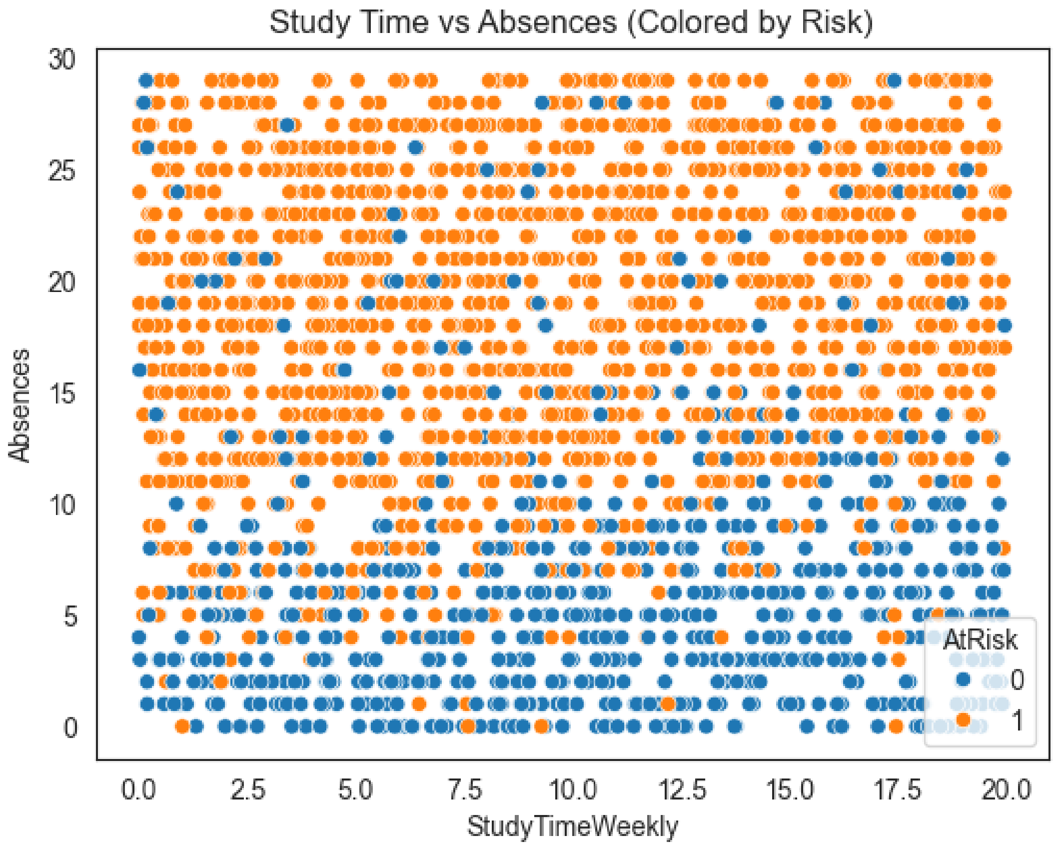

## 0.9 Create a function to map GradeClass values to at risk or not

[35]:
```
def convert_to_risk(x):
    if x >= 3:
        return 1
    else:
        return 0

df_cleaned['AtRisk'] = df_cleaned['GradeClass'].apply(convert_to_risk)
```

## 0.10 More data visualisation

[36]:
```
sns.scatterplot(x='StudyTimeWeekly', y='Absences', hue='AtRisk',␣
 ↪data=df_cleaned)
plt.title("Study Time vs Absences (Colored by Risk)")
plt.show()
```




### 0.11 Outlier treatment

[37]:
```
def remove_outliers(df, cols):  #function to remove outliers
    for col in cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)

        IQR = Q3 - Q1

        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower) & (df[col]<= upper)]

    return df


df_cleaned = remove_outliers(df_cleaned, ['Absences', 'StudyTimeWeekly',␣
↪'ParentalSupport', 'Engagement'])  #calls the function made above
```

### 0.12 Splitting data into features and a target

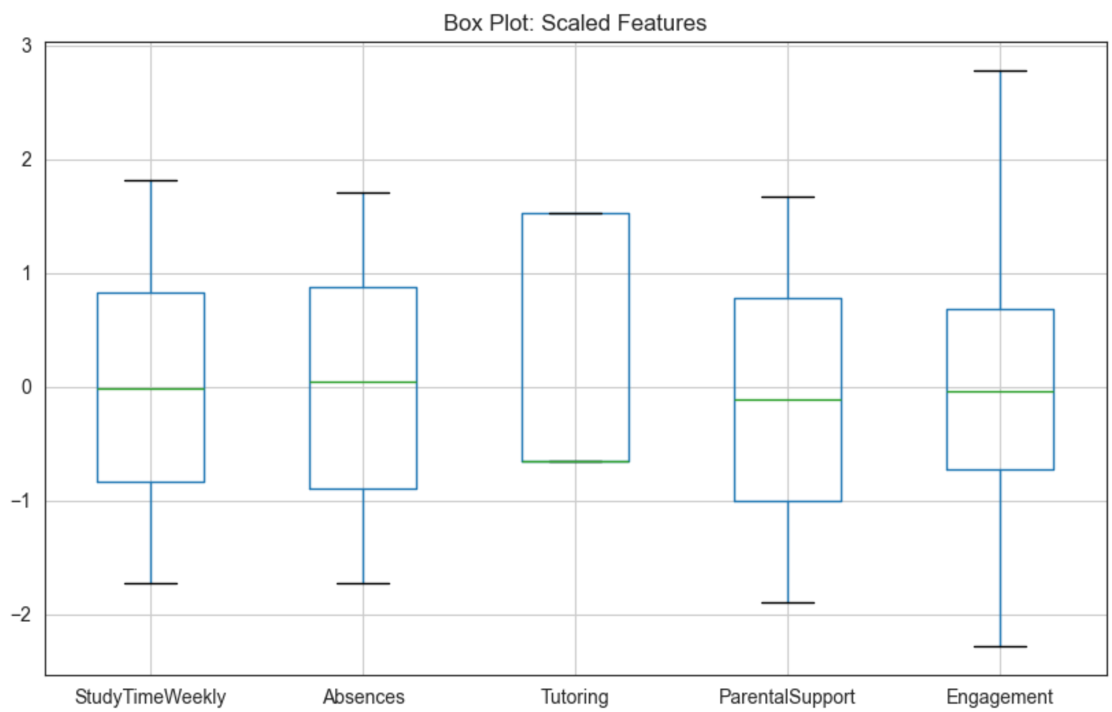
[38]:
```
X = df_cleaned.drop(columns=['GradeClass', 'AtRisk'])
y = df_cleaned['AtRisk'] #AtRisk = D or F
```

### 0.13 Scaling input data using standard scaler

[39]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

### 0.14 Viewing scaled data

[40]:
```
#converts data to df
x_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
#displays scaled features
fig, ax = plt.subplots(figsize=(10, 6))
boxplot = x_scaled_df.boxplot(vert = 1, ax=ax)
_ = ax.set_title(f'Box Plot: Scaled Features')
```



## 0.15 spliting data into training and testing data

[45]:
```
# Get the original indices
original_indices = np.arange(len(X_scaled))
# Split indices along with the data
X_train, X_test, y_train, y_test, idx_train, idx_test = train_test_split(
    X_scaled, y, original_indices, test_size=0.2, random_state=95
)
```

Save the Train and est data to csv files

[ ]:
```
# Convert to Series and save
train_index = pd.Series(idx_train)
test_index = pd.Series(idx_test)

train_index_path = os.path.join(parent_folder, "Data", "train_index.csv")
test_index_path = os.path.join(parent_folder, "Data", "test_index.csv")

train_index.to_csv(train_index_path, index=False, header=False)
test_index.to_csv(test_index_path, index=False, header=False)

print("Part 1: Train and test indices saved to CSV.")
```

```
Part 1: Train and test indices saved to CSV.
```

## 0.16 implementing basic ML models

### 0.16.1 Logistic regression

[47]:
```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
print("Logistic Regression Report:\n", classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Report:
               precision    recall  f1-score   support

           0       0.90      0.80      0.85       147
           1       0.92      0.96      0.94       329

    accuracy                           0.91       476
   macro avg       0.91      0.88      0.89       476
weighted avg       0.91      0.91      0.91       476
```

### 0.16.2 Random Forest

[48]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=101)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))
```

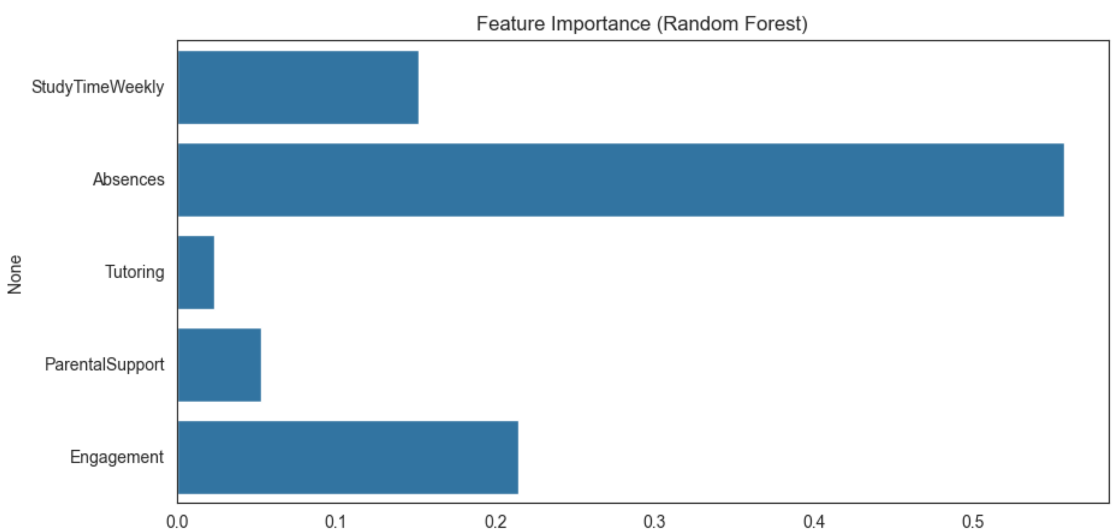
```
Random Forest Report:
               precision    recall  f1-score   support

           0       0.90      0.79      0.84       147
           1       0.91      0.96      0.93       329

    accuracy                           0.91       476
   macro avg       0.90      0.87      0.89       476
weighted avg       0.91      0.91      0.91       476
```

### 0.16.3 Random Forest importance

[49]:
```
importances = rf.feature_importances_
feature_names = X.columns
plt.figure(figsize=(10, 5))
sns.barplot(x=importances, y=feature_names)
plt.title('Feature Importance (Random Forest)')
plt.show()
```



### 0.16.4 XGBoost

[51]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
print('XGBoost:\n', classification_report(y_test, y_pred_xgb))
```

```
XGBoost:
               precision    recall  f1-score   support

           0       0.88      0.75      0.81       147
           1       0.89      0.95      0.92       329

    accuracy                           0.89       476
   macro avg       0.89      0.85      0.87       476
weighted avg       0.89      0.89      0.89       476


c:\Users\Ruan\AppData\Local\Programs\Python\Python312\Lib\site-
packages\xgboost\training.py:183: UserWarning: [13:14:12] WARNING: C:\actions-
runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```