In [44]:

```
# supresses future warnings
import warnings
warnings.simplefilter(action='ignore')

# Import the pandas library for df creation
import pandas as pd

# Import the NumPy library to use the random package
import numpy as np
import math
from scipy import stats

### For Splitting the Data set
from sklearn.model_selection import train_test_split


# Import the matplotlib library for plotting
import matplotlib.pyplot as plt

# Use the magic function to ensure plots render in a notebook
%matplotlib inline

## Converting Categorical Varaibles into Numerical Varaibles
from sklearn.preprocessing import LabelEncoder


# Import the seaborn library for plotting
import seaborn as sns

## For Materics Calculation
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [45]:

```
df = pd.read_csv(r'StudentsPerformance (1).csv')
```

In [46]:

```
testing = pd.read_csv(r'StudentsPerformance.csv')
```

In [47]:

```
df.head()
```

Out[47]:

	Student Number	Attendance (%)	Gender	Race/Ethnicity	Parental level of education	Lunch Type	Test Preparation Course	Mathematics	English	Biology	Physics	Chemistry
0	1	80	female	group B	bachelor's degree	standard	none	72	72	74	63	70
1	2	69	female	group C	some college	standard	completed	69	90	88	41	94
2	3	88	female	group B	master's degree	standard	none	90	95	93	61	78
3	4	65	male	group A	associate's degree	Special	none	47	57	44	49	96
4	5	70	male	group C	some college	standard	none	76	78	75	44	76

In [48]:

```
testing
```

Out[48]:

	Student Number	Attendance (%)	Gender	Race/Ethnicity	Parental level of education	Lunch Type	Test Preparation Course
0	1	80	female	group B	bachelor's degree	standard	none
1	2	69	female	group C	some college	standard	completed
2	3	88	female	group B	master's degree	standard	none
3	4	65	male	group A	associate's degree	Special	none
4	5	70	male	group C	some college	standard	none
...	...	...	...	...	...	...	...
995	996	94	female	group E	master's degree	standard	completed
996	997	89	male	group C	high school	Special	none
997	998	80	female	group C	high school	Special	completed
998	999	69	female	group D	some college	standard	completed
999	1000	88	female	group D	some college	Special	none

1000 rows × 7 columns

In [49]:

```
### Printing the Shape

print("df shape: {}".format(df.shape))
```

```
df shape: (1000, 13)
```

In [50]:

```
df.info()
```

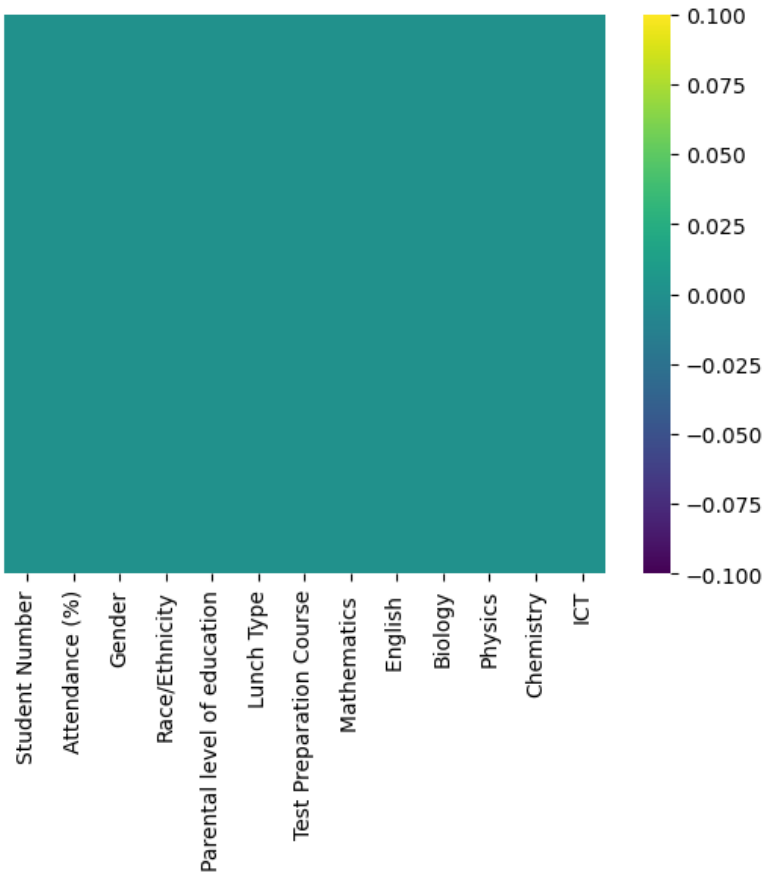
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 13 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Student Number               1000 non-null   int64 
 1   Attendance (%)               1000 non-null   int64 
 2   Gender                       1000 non-null   object
 3   Race/Ethnicity               1000 non-null   object
 4   Parental level of education  1000 non-null   object
 5   Lunch Type                   1000 non-null   object
 6   Test Preparation Course      1000 non-null   object
 7   Mathematics                  1000 non-null   int64 
 8   English                      1000 non-null   int64 
 9   Biology                      1000 non-null   int64 
 10  Physics                      1000 non-null   int64 
 11  Chemistry                    1000 non-null   int64 
 12  ICT                          1000 non-null   int64 
dtypes: int64(8), object(5)
memory usage: 101.7+ KB
```

In [51]:

```
# Checking the Missing Valåues
sns.heatmap(df.isnull(), yticklabels=False, cmap= "viridis")
```

Out[51]:

```
<AxesSubplot: >
```



In [52]:

```
df.describe()
```

Out[52]:

	Student Number	Attendance (%)	Mathematics	English	Biology	Physics	Chemistry	ICT
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	79.920000	66.08900	69.169000	68.054000	68.005000	67.604000	67.092000
std	288.819436	11.400244	15.16308	14.600192	15.195657	14.205449	14.864998	14.917777
min	1.000000	63.000000	0.00000	17.000000	10.000000	22.000000	10.000000	8.000000
25%	250.750000	69.000000	57.00000	59.000000	57.750000	59.000000	58.000000	57.000000
50%	500.500000	84.000000	66.00000	70.000000	69.000000	68.000000	68.000000	67.000000
75%	750.250000	90.000000	77.00000	79.000000	79.000000	78.000000	78.000000	77.000000
max	1000.000000	96.000000	100.00000	100.000000	100.000000	100.000000	100.000000	100.000000

In [53]:

```
def report(df):
    col = []
    d_type = []
    uniques = []
    n_uniques = []

    for i in df.columns:
        col.append(i)
        d_type.append(df[i].dtypes)
        uniques.append(df[i].unique()[:5])
        n_uniques.append(df[i].nunique())

    return pd.DataFrame({'Column': col, 'd_type': d_type, 'unique_sample': uniques, 'n_uniques': n_uniques})
```

In [54]:

```
report(df)
```

Out[54]:

	Column	d_type	unique_sample	n_uniques
0	Student Number	int64	[1, 2, 3, 4, 5]	1000
1	Attendance (%)	int64	[80, 69, 88, 65, 70]	21
2	Gender	object	[female, male]	2
3	Race/Ethnicity	object	[group B, group C, group A, group D, group E]	5
4	Parental level of education	object	[bachelor's degree, some college, master's deg...	7
5	Lunch Type	object	[standard, Special]	2
6	Test Preparation Course	object	[none, completed]	2
7	Mathematics	int64	[72, 69, 90, 47, 76]	81
8	English	int64	[72, 90, 95, 57, 78]	72
9	Biology	int64	[74, 88, 93, 44, 75]	77
10	Physics	int64	[63, 41, 61, 49, 44]	73
11	Chemistry	int64	[70, 94, 78, 96, 76]	76
12	ICT	int64	[80, 62, 48, 77, 66]	77

In [55]:

```
df.columns
```

Out[55]:

```
Index(['Student Number', 'Attendance (%)', 'Gender', 'Race/Ethnicity',
       'Parental level of education', 'Lunch Type', 'Test Preparation Course',
       'Mathematics ', 'English ', 'Biology', 'Physics', 'Chemistry', 'ICT'],
      dtype='object')
```

In [56]:

```
# Remove trailing whitespaces from column names
df.columns = df.columns.str.strip()
```

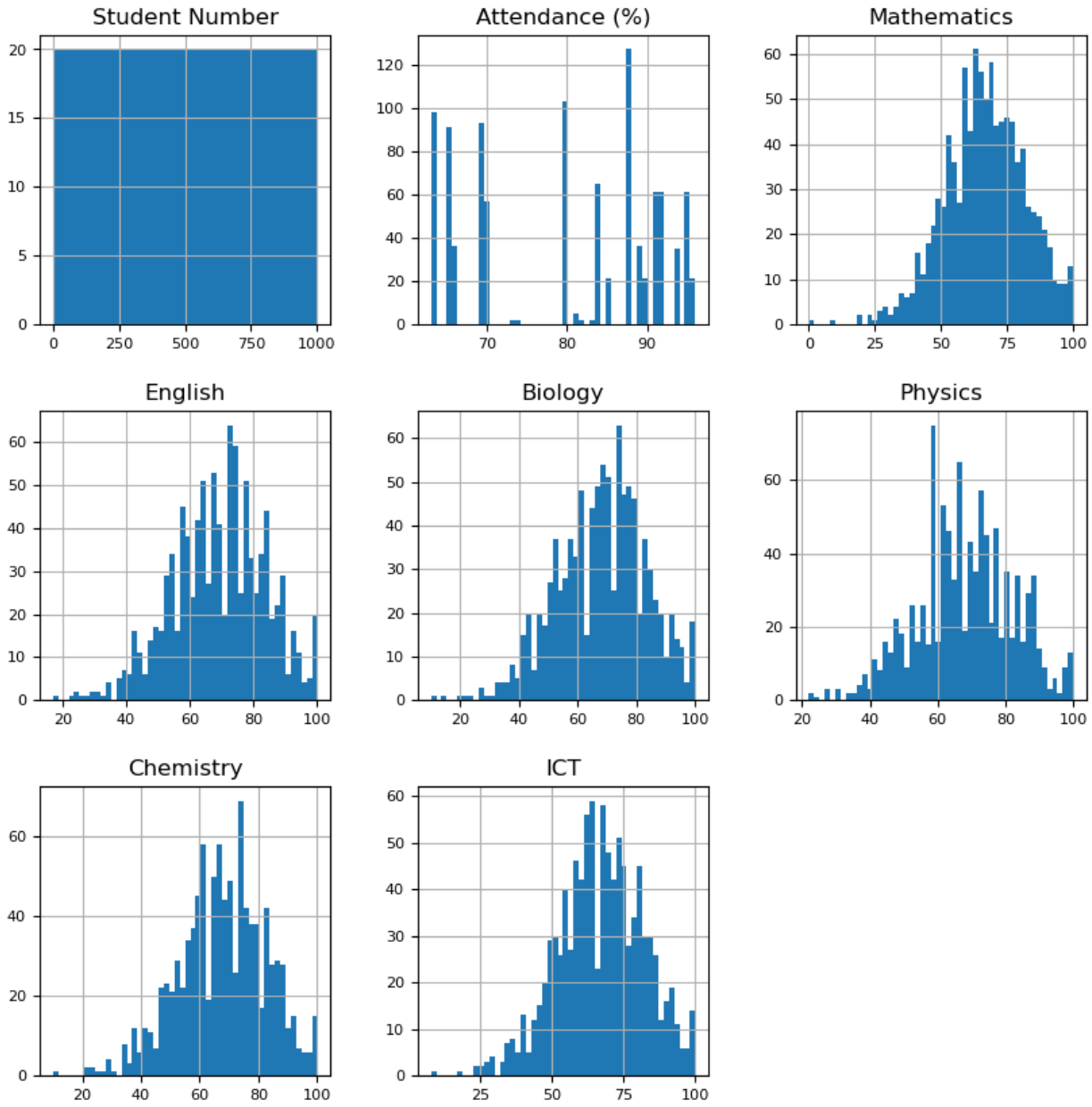
In [57]:

```
## Checking for Duplicate Values
duplicate = df.duplicated()
print(duplicate.sum())
```

```
0
```

In [58]:

```
## Checking the frequency of Numerical Varaible
df.hist(figsize=(10, 10), bins=50, xlabelsize=8, ylabelsize=8);
```



In [59]:

```
df
```

Out[59]:

	Student Number	Attendance (%)	Gender	Race/Ethnicity	Parental level of education	Lunch Type	Test Preparation Course	Mathematics	English	Biology	Physics	Chemistı
0	1	80	female	group B	bachelor's degree	standard	none	72	72	74	63	7
1	2	69	female	group C	some college	standard	completed	69	90	88	41	9
2	3	88	female	group B	master's degree	standard	none	90	95	93	61	7
3	4	65	male	group A	associate's degree	Special	none	47	57	44	49	9
4	5	70	male	group C	some college	standard	none	76	78	75	44	7
...	...	...	...	...	...	...	...	...	...	...	...	
995	996	94	female	group E	master's degree	standard	completed	88	99	95	79	7
996	997	89	male	group C	high school	Special	none	62	55	55	67	6
997	998	80	female	group C	high school	Special	completed	59	71	65	69	7
998	999	69	female	group D	some college	standard	completed	68	78	77	86	7
999	1000	88	female	group D	some college	Special	none	77	86	86	47	8

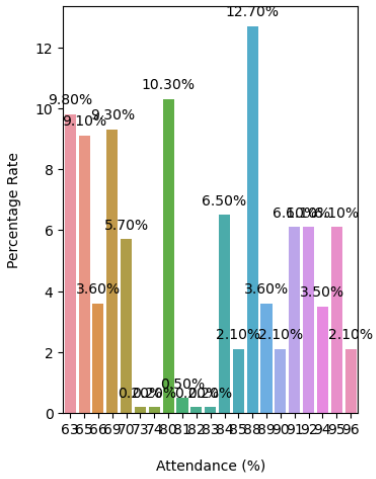
1000 rows × 13 columns

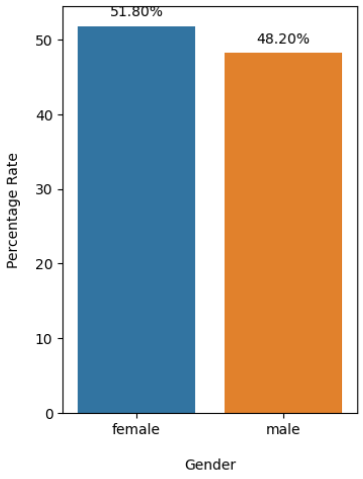
In [60]:

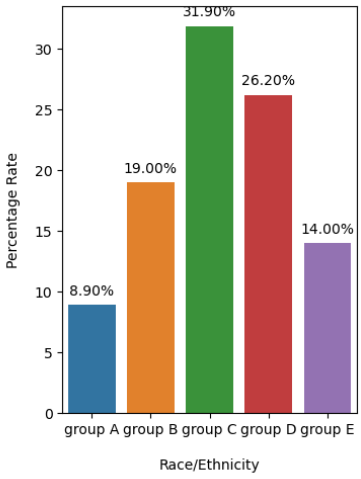
```
def add_data_labels(ax, spacing = 5):

    # For each bar: Place a label
    for rect in ax.patches:
        # Get X and Y placement of label from rect.
        y_value = rect.get_height()
        x_value = rect.get_x() + rect.get_width() / 2

        # Number of points between bar and label. Change to your liking.
        space = spacing
        # Vertical alignment for positive values
        va = 'bottom'

        # If value of bar is negative: Place label below bar
        if y_value < 0:
            # Invert space to place label below
            space *= -1
            # Vertically align label at top
            va = 'top'

        # Use Y value as label and format number with one decimal place
        label = "{:.2f}%".format(y_value)

        # Create annotation
        plt.annotate(
            label,                      # Use `label` as label
            (x_value, y_value),         # Place label at end of the bar
            xytext = (0, space),        # Vertically shift label by `space`
            textcoords = "offset points", # Interpret `xytext` as offset in points
            ha = 'center',              # Horizontally center label
            va = va)                    # Vertically align label differently for positive and negative values
```

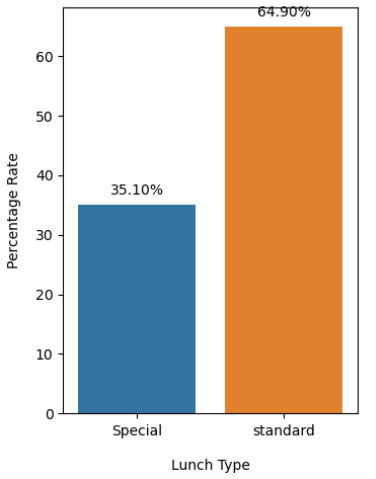
In [61]:

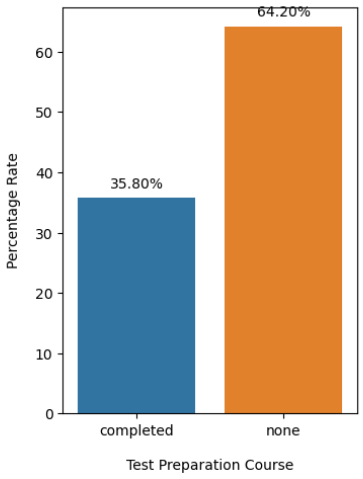
```
# Univariate Plot Analysis of Ordered categorical variables vs Percentage Rate
category_list = [ 'Attendance (%)', 'Gender', 'Race/Ethnicity',
        'Lunch Type', 'Test Preparation Course']
counter = 1

plt.figure(figsize = (15, 12))

for col_list in category_list:

    series = round(((df[col_list].value_counts(dropna = False))/
                    (len(df[col_list])) * 100), 2)

    plt.subplot(2, 3, counter)
    ax = sns.barplot(x = series.index, y = series.values, order = series.sort_index().index)
    plt.xlabel(col_list, labelpad = 15)
    plt.ylabel('Percentage Rate', labelpad = 10)

    # Call Custom Function
    add_data_labels(ax)

    counter += 1

del category_list, counter, ax

plt.subplots_adjust(hspace = 0.3)
plt.subplots_adjust(wspace = 0.5)
plt.show()
```









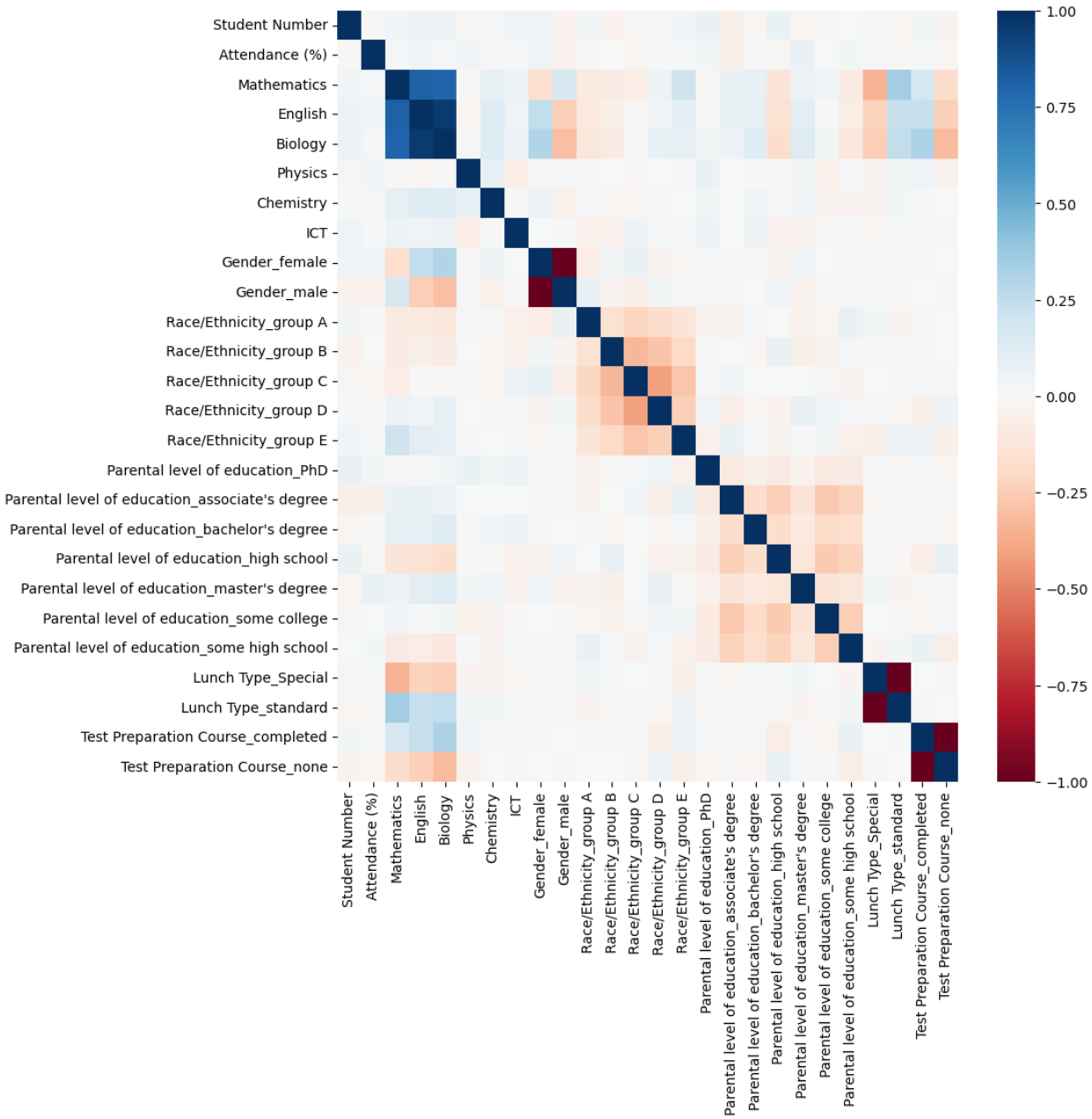


In [62]:

```
# Convert categorical variables to numeric representation
df_encoded = pd.get_dummies(df)

# Compute the correlation matrix
correlation = df_encoded.corr()

# Visualize the correlation matrix
plt.figure(figsize=(10, 10))
sns.heatmap(correlation, annot=False, fmt='.2f', annot_kws={'size': 10}, cmap='RdBu')
```

Out[62]:

```
<AxesSubplot: >
```



In [63]:

```
df_encoded.corr(method ="pearson")
```

Out[63]:

	Student Number	Attendance (%)	Mathematics	English	Biology	Physics	Chemistry	ICT	Gender_female	Gen
Student Number	1.000000	-0.003636	0.036827	0.048390	0.057848	-0.012999	0.010662	0.043070	0.043038	
Attendance (%)	-0.003636	1.000000	0.016255	0.032100	0.022087	0.037262	0.017687	0.014352	0.040649	
Mathematics	0.036827	0.016255	1.000000	0.817580	0.802642	-0.006606	0.088049	0.057329	-0.167982	
English	0.048390	0.032100	0.817580	1.000000	0.954598	-0.015844	0.124562	0.043282	0.244313	
Biology	0.057848	0.022087	0.802642	0.954598	1.000000	-0.004958	0.120366	0.057781	0.301225	
Physics	-0.012999	0.037262	-0.006606	-0.015844	-0.004958	1.000000	0.078795	-0.071683	-0.008681	
Chemistry	0.010662	0.017687	0.088049	0.124562	0.120366	0.078795	1.000000	-0.003835	0.052280	
ICT	0.043070	0.014352	0.057329	0.043282	0.057781	-0.071683	-0.003835	1.000000	0.006892	
Gender_female	0.043038	0.040649	-0.167982	0.244313	0.301225	-0.008681	0.052280	0.006892	1.000000	
Gender_male	-0.043038	-0.040649	0.167982	-0.244313	-0.301225	0.008681	-0.052280	-0.006892	-1.000000	
Race/Ethnicity_group A	0.025542	-0.007052	-0.091977	-0.096274	-0.110714	0.016957	0.018495	-0.050685	-0.071001	
Race/Ethnicity_group B	-0.043171	0.007651	-0.084250	-0.060283	-0.078254	0.000727	-0.019689	-0.047438	0.028466	
Race/Ethnicity_group C	-0.015024	-0.016660	-0.073387	-0.003074	-0.010203	-0.012481	-0.016415	0.059810	0.063368	
Race/Ethnicity_group D	0.011297	0.009173	0.050071	0.035177	0.082032	0.006838	0.019095	0.017219	-0.030566	
Race/Ethnicity_group E	0.033714	0.007891	0.205855	0.106712	0.089077	-0.006637	0.004935	-0.006935	-0.020302	
Parental level of education_PhD	0.069661	0.024225	-0.008486	0.011475	0.023712	0.064155	0.024260	0.061283	0.011664	
Parental level of education_associate's degree	-0.058327	-0.058951	0.063342	0.064370	0.062249	0.000521	0.000131	0.000371	0.011562	
Parental level of education_bachelor's degree	-0.017108	-0.007473	0.082937	0.092403	0.121016	-0.004797	0.034812	0.049063	-0.003376	
Parental level of education_high school	0.068062	-0.008506	-0.126295	-0.147699	-0.178235	-0.003018	-0.004379	-0.042817	-0.035409	
Parental level of education_master's degree	-0.017752	0.063959	0.060417	0.106452	0.125693	0.026216	0.034953	-0.041679	0.046188	
Parental level of education_some college	-0.014190	0.001471	0.037603	0.012140	0.029303	-0.047753	-0.023838	-0.006216	0.006949	
Parental level of education_some high school	0.000980	0.025638	-0.079633	-0.075670	-0.101621	0.014481	-0.030768	0.008984	-0.013958	
Lunch Type_Special	0.021799	0.014725	-0.345373	-0.227614	-0.243062	-0.025050	-0.028627	-0.016622	0.013342	
Lunch Type_standard	-0.021799	-0.014725	0.345373	0.227614	0.243062	0.025050	0.028627	0.016622	-0.013342	
Test Preparation Course_completed	0.028029	0.015677	0.177702	0.241780	0.312946	0.040872	-0.001577	0.014418	-0.006028	
Test Preparation Course_none	-0.028029	-0.015677	-0.177702	-0.241780	-0.312946	-0.040872	0.001577	-0.014418	0.006028	

26 rows × 26 columns

Linear Regression

In [64]:

```
ures = ['Student Number','Attendance (%)', 'Gender', 'Race/Ethnicity', 'Parental level of education', 'Lunch Typ
et_variables = ['Mathematics', 'English', 'Biology', 'Physics', 'Chemistry', 'ICT']
```

In [65]:

```
df.columns
```

Out[65]:

```
Index(['Student Number', 'Attendance (%)', 'Gender', 'Race/Ethnicity',
       'Parental level of education', 'Lunch Type', 'Test Preparation Course',
       'Mathematics', 'English', 'Biology', 'Physics', 'Chemistry', 'ICT'],
      dtype='object')
```

In [66]:

```
data_encoded = pd.get_dummies(df[features])
```

In [67]:

```
### Splitting the data

X_train, X_test, y_train, y_test = train_test_split(data_encoded, df[target_variables], test_size=0.2, random_st
```

In [68]:

```
print("X_train shape: {}".format(X_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_train shape: {}".format(y_train.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (800, 20)
X_test shape: (200, 20)
y_train shape: (800, 6)
y_test shape: (200, 6)
```

In [69]:

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[69]:

▾ LinearRegression

LinearRegression()

In [70]:

```
predictions = model.predict(X_test)
```

In [91]:

```
print(predictions)
```

```
[[67.04375681 74.53469621 74.54285324 66.65155119 70.05558993 68.16077565]
 [69.44625476 78.49646788 80.55581856 66.95482026 66.06944154 66.40291002]
 [65.28140228 73.59335291 73.20653839 69.03704761 70.08724392 63.9254222 ]
 ...
 [56.58217601 69.77180697 69.97061246 67.11730503 67.56058293 68.37261824]
 [74.98086645 82.3761443  86.51417575 66.83032721 68.61571385 67.96119123]
 [58.22561336 67.88165979 67.82768127 66.32475768 67.74619942 68.51526251]]
```

In [94]:

```
predictions.shape
```

Out[94]:

```
(1000, 6)
```

In [84]:

```
# Calculate evaluation metrics
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

# Print the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2) Score:", r2)
```

```
Mean Squared Error (MSE): 207.8476846423782
Mean Absolute Error (MAE): 11.347642354873287
R-squared (R2) Score: 0.07280864000051816
```

Within Book Checking

In [2]:

```
def predict_grades(features):
    # Load the dataset into a Pandas DataFrame
    data = pd.read_csv('StudentsPerformance (1).csv')  # Replace 'your_dataset.csv' with the actual filename/pa

    # Remove trailing whitespaces from column names
    data.columns = data.columns.str.strip()

    # Select the relevant columns for training and prediction
    selected_features = ['Student Number', 'Attendance (%)', 'Gender', 'Race/Ethnicity', 'Parental level of edu
    target_variables = ['Mathematics', 'English', 'Biology', 'Physics', 'Chemistry', 'ICT']

    # Convert categorical variables into numerical representations (one-hot encoding)
    data_encoded = pd.get_dummies(data[selected_features])

    # Create a linear regression model and fit it to the entire dataset
    model = LinearRegression()
    model.fit(data_encoded, data[target_variables])

    # Prepare input data for prediction
    input_data = pd.DataFrame([features], columns=selected_features)
    input_data_encoded = pd.get_dummies(input_data)

    # Check for missing columns in input_data_encoded
    missing_cols = set(data_encoded.columns) - set(input_data_encoded.columns)
    for col in missing_cols:
        input_data_encoded[col] = 0

    # Reorder columns to match the order in the trained model
    input_data_encoded = input_data_encoded[data_encoded.columns]

    # Make predictions
    predictions = model.predict(input_data_encoded)

    return dict(zip(target_variables, predictions[0]))
```

In [6]:

```
input_features = {
    'Student Number' : 1000,
    'Attendance (%)': 80,
    'Gender': 'male',
    'Race/Ethnicity': 'group A',
    'Parental level of education': "Master's degree",
    'Lunch Type': 'standard',
    'Test Preparation Course': 'completed'
}

predicted_grades = predict_grades(input_features)
print(predicted_grades)
```

```
{'Mathematics': 73.64616894371855, 'English': 71.65526093799535, 'Biology': 71.52027500422251, 'Ph
ysics': 70.44515302233094, 'Chemistry': 69.05951792276925, 'ICT': 66.43941201376248}
```

In [ ]:

```
```