In [13]:
```python
import pandas as pd

df = pd.read_csv('StudentPerformanceFactors.csv')
df.head()
```

Out[13]:

	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurri
0	23	84	Low	High	
1	19	64	Low	Medium	
2	24	98	Medium	Medium	
3	29	89	Low	Medium	
4	19	92	Medium	Medium	

In [15]:
```python
df.shape
df.columns
```

Out[15]:
```
Index(['Hours_Studied', 'Attendance', 'Parental_Involvement',
       'Access_to_Resources', 'Extracurricular_Activities', 'Sleep_Hours',
       'Previous_Scores', 'Motivation_Level', 'Internet_Access',
       'Tutoring_Sessions', 'Family_Income', 'Teacher_Quality', 'School_Typ
e',
       'Peer_Influence', 'Physical_Activity', 'Learning_Disabilities',
       'Parental_Education_Level', 'Distance_from_Home', 'Gender',
       'Exam_Score'],
      dtype='object')
```

In [17]:
```python
#creating a df copy -> df1
df1 = df.copy()
```

In [19]:
```python
df1.head(10)
```

Out[19]:

	Hours_Studied	Attendance	Parental_Involvement	Access_to_Resources	Extracurri
0	23	84	Low	High	
1	19	64	Low	Medium	
2	24	98	Medium	Medium	
3	29	89	Low	Medium	
4	19	92	Medium	Medium	
5	19	88	Medium	Medium	
6	29	84	Medium	Low	
7	25	78	Low	High	
8	17	94	Medium	High	
9	23	98	Medium	Medium	

In [21]:
```
df1.isnull().sum()
```

Out[21]:
```
Hours_Studied                  0
Attendance                     0
Parental_Involvement           0
Access_to_Resources            0
Extracurricular_Activities     0
Sleep_Hours                    0
Previous_Scores                0
Motivation_Level               0
Internet_Access                0
Tutoring_Sessions              0
Family_Income                  0
Teacher_Quality               78
School_Type                    0
Peer_Influence                 0
Physical_Activity              0
Learning_Disabilities          0
Parental_Education_Level      90
Distance_from_Home            67
Gender                         0
Exam_Score                     0
dtype: int64
```

In [31]:
```
#Dropping all missing values from 'Distance From Home'
df1.dropna(subset=['Distance_from_Home'],inplace=True)
```

In [33]:
```
df1.isnull().sum()
```

Out[33]:
```
Hours_Studied                  0
Attendance                     0
Parental_Involvement           0
Access_to_Resources            0
Extracurricular_Activities     0
Sleep_Hours                    0
Previous_Scores                0
Motivation_Level               0
Internet_Access                0
Tutoring_Sessions              0
Family_Income                  0
Teacher_Quality               76
School_Type                    0
Peer_Influence                 0
Physical_Activity              0
Learning_Disabilities          0
Parental_Education_Level      90
Distance_from_Home             0
Gender                         0
Exam_Score                     0
dtype: int64
```

In [35]:
```
df2= df1[['Distance_from_Home','Exam_Score','Hours_Studied','Sleep_Hours']].
```

In [37]:
```
#df2.groupby('Exam_Score').mean()
pd.set_option('display.max_rows', None)     # Show all rows
```

```
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.width', None)        # Ensure full width is shown
pd.set_option('display.max_colwidth', None) # Ensure columns are fully visib
df_grouped = df2.groupby(['Exam_Score','Distance_from_Home']).mean(numeric_c
df_grouped.head(150)
```

Out[37]:

Exam_Score	Distance_from_Home	Hours_Studied	Sleep_Hours
55	Near	3.000000	6.000000
56	Far	5.000000	7.000000
57	Far	14.000000	7.000000
	Moderate	7.000000	8.000000
	Near	6.000000	8.000000
58	Far	9.571429	6.571429
	Moderate	7.333333	7.222222
	Near	9.333333	7.166667
59	Far	12.000000	7.333333
	Moderate	9.875000	6.250000
	Near	12.631579	7.210526
60	Far	14.250000	6.750000
	Moderate	13.538462	7.538462
	Near	11.902439	6.902439
61	Far	17.350000	7.550000
	Moderate	13.529412	7.235294
	Near	13.407407	7.037037
62	Far	15.355556	7.088889
	Moderate	15.506173	7.024691
	Near	14.620438	7.138686
63	Far	18.045455	7.045455
	Moderate	16.588235	7.100840
	Near	16.303483	7.029851
64	Far	18.250000	6.857143
	Moderate	18.179191	6.965318
	Near	17.462406	6.988722
65	Far	19.853333	7.240000
	Moderate	18.919431	7.014218
	Near	17.963636	6.994805
66	Far	20.000000	6.797468
	Moderate	19.294118	7.031674

Exam_Score	Distance_from_Home	Hours_Studied	Sleep_Hours
	Near	18.797297	7.056306
67	**Far**	20.464789	6.802817
	Moderate	19.872807	6.986842
	Near	19.851582	7.082725
68	**Far**	21.115385	6.987179
	Moderate	21.392070	7.039648
	Near	20.438753	7.042316
69	**Far**	22.440000	7.060000
	Moderate	21.158730	7.095238
	Near	20.843085	6.957447
70	**Far**	24.162162	6.945946
	Moderate	22.598684	7.098684
	Near	22.060519	7.051873
71	**Far**	22.761905	7.571429
	Moderate	23.974359	7.239316
	Near	22.865169	7.086142
72	**Far**	25.857143	6.666667
	Moderate	24.837500	6.937500
	Near	24.199005	7.208955
73	**Far**	26.666667	6.833333
	Moderate	27.037037	6.888889
	Near	25.607843	6.862745
74	**Far**	27.875000	6.875000
	Moderate	28.592593	6.777778
	Near	26.371429	6.542857
75	**Far**	31.000000	7.000000
	Moderate	31.071429	6.571429
	Near	29.266667	6.900000
76	**Moderate**	30.000000	6.250000
	Near	31.083333	6.833333
77	**Moderate**	39.000000	9.000000

Exam_Score	Distance_from_Home	Hours_Studied	Sleep_Hours
	Near	32.000000	7.250000
78	**Moderate**	19.000000	5.000000
	Near	38.333333	7.333333
79	**Far**	16.000000	7.000000
	Moderate	39.000000	10.000000
	Near	35.000000	7.000000
80	**Far**	21.500000	6.500000
	Moderate	18.000000	6.000000
	Near	15.000000	6.500000
82	**Moderate**	14.000000	5.000000
	Near	14.333333	6.666667
83	**Near**	22.000000	7.000000
84	**Far**	24.000000	7.000000
	Moderate	25.000000	7.000000
	Near	22.000000	9.000000
85	**Moderate**	24.000000	9.000000
86	**Moderate**	7.000000	10.000000
	Near	20.000000	6.666667
87	**Far**	7.000000	8.000000
	Near	11.000000	8.000000
88	**Moderate**	11.000000	7.000000
	Near	24.000000	8.500000
89	**Moderate**	20.000000	8.000000
	Near	14.000000	6.000000
91	**Moderate**	21.000000	9.000000
92	**Far**	31.000000	7.000000
	Near	1.000000	4.000000
93	**Near**	25.500000	7.000000
94	**Far**	19.000000	5.000000
	Moderate	26.500000	7.000000
	Near	25.000000	7.000000

Exam_Score	Distance_from_Home	Hours_Studied	Sleep_Hours
95	Near	19.500000	6.000000
96	Near	18.000000	7.000000
97	Near	20.333333	6.666667
98	Moderate	28.000000	9.000000
	Near	22.000000	6.000000
99	Far	23.000000	4.000000
	Near	14.000000	8.000000
100	Near	18.000000	4.000000
101	Moderate	27.000000	6.000000

In [39]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

RESEARCH QUESTION 1: RANDOM FOREST MODEL

Exam score < 70 (0) is a bad score while exam score >= 70 (1) is a good score

In [272…
```
dfr_1.loc[df1['Exam_Score'] < 70, 'ExamCode'] = 0
dfr_1.loc[df1['Exam_Score'] >= 70, 'ExamCode'] = 1
```

In [274…
```
dfr_2 = dfr_1[['ExamCode','Extracurricular_Activities','Tutoring_Sessions','
```

In [280…
```
dfr_2.loc[df1['Extracurricular_Activities'] == 'No', 'Extracurricular'] = 0
dfr_2.loc[df1['Extracurricular_Activities'] == 'Yes', 'Extracurricular'] = 1
dfr_2.head()
```

Out[280…

	ExamCode	Extracurricular_Activities	Tutoring_Sessions	Physical_Activity	Extracu
0	0.0	No	0	3	
1	0.0	No	2	4	
2	1.0	Yes	2	4	
3	1.0	Yes	1	4	
4	1.0	Yes	3	4	

In [282…
```
dfr_3 = dfr_1[['ExamCode','Extracurricular','Tutoring_Sessions','Physical_Ac
```

In [288…
```
dfr_3['ExamCode'].value_counts()
```

Out [288…
```
ExamCode
0.0    4927
1.0    1613
Name: count, dtype: int64
```

In [292…
```
X = dfr_3[['Extracurricular','Tutoring_Sessions','Physical_Activity']]

Y = dfr_3['ExamCode']
```

In [300…
```
#The following divides the dataset into 70% training and 30% testing.
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, r
```

In [302…
```
Y_train.value_counts()
```

Out [302…
```
ExamCode
0.0    3453
1.0    1125
Name: count, dtype: int64
```

In [304…
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50, verbose=0, bootstrap=True, max
clf = clf.fit(X_train,Y_train)
```

In [306…
```
Y_test.value_counts()
```

Out [306…
```
ExamCode
0.0    1474
1.0     488
Name: count, dtype: int64
```

Random Forests Tuning

In [308…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

depths=[1,2,3,4,5]
estimators=[50,100,150,200,250]
min_samples_split: [2, 5, 10]


best_mean_score = 0
best_params = {'n_estimators': None, 'max_depth': None}

kf =KFold(n_splits=5, shuffle=True, random_state=0)


for estimator in estimators:
    for depth in depths:
        clf = RandomForestClassifier(n_estimators=estimator, verbose=0, boot
        CVscores= cross_val_score(clf, X_train, Y_train, scoring='accuracy',
        print('With n_estimators={} and with max_depth={}, the Cross validat
```

```
        #Check if the current combination has a higher mean score

        if CVscores.mean() > best_mean_score:
            best_mean_score = CVscores.mean()
            best_params['n_estimators'] = estimator
            best_params['max_depth'] = depth

print(f"\nBest parameters: {best_params}")
print(f"Best mean score: {best_mean_score}")
```

```
With n_estimators=50 and with max_depth=1, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=50 and with max_depth=2, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=50 and with max_depth=3, the Cross validation scores mean
is 0.7538206027632615:
With n_estimators=50 and with max_depth=4, the Cross validation scores mean
is 0.7518543441429832:
With n_estimators=50 and with max_depth=5, the Cross validation scores mean
is 0.7522910253656907:
With n_estimators=100 and with max_depth=1, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=100 and with max_depth=2, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=100 and with max_depth=3, the Cross validation scores mean
is 0.7538206027632615:
With n_estimators=100 and with max_depth=4, the Cross validation scores mean
is 0.7531655809292004:
With n_estimators=100 and with max_depth=5, the Cross validation scores mean
is 0.7514171856730378:
With n_estimators=150 and with max_depth=1, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=150 and with max_depth=2, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=150 and with max_depth=3, the Cross validation scores mean
is 0.7538206027632615:
With n_estimators=150 and with max_depth=4, the Cross validation scores mean
is 0.7531655809292004:
With n_estimators=150 and with max_depth=5, the Cross validation scores mean
is 0.7511988450616842:
With n_estimators=200 and with max_depth=1, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=200 and with max_depth=2, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=200 and with max_depth=3, the Cross validation scores mean
is 0.7538206027632615:
With n_estimators=200 and with max_depth=4, the Cross validation scores mean
is 0.7531655809292004:
With n_estimators=200 and with max_depth=5, the Cross validation scores mean
is 0.7516360035316295:
With n_estimators=250 and with max_depth=1, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=250 and with max_depth=2, the Cross validation scores mean
is 0.7542577612332069:
With n_estimators=250 and with max_depth=3, the Cross validation scores mean
is 0.7538206027632615:
With n_estimators=250 and with max_depth=4, the Cross validation scores mean
is 0.7531655809292004:
With n_estimators=250 and with max_depth=5, the Cross validation scores mean
is 0.7518543441429832:

Best parameters: {'n_estimators': 50, 'max_depth': 1}
Best mean score: 0.7542577612332069
```

Train Data

In [328…
```
from sklearn.ensemble import RandomForestClassifier

# generate 100 decision trees
# verbose 0 (silent), 1 (progress bar) or 2 (one line per tree) you just say
# bootstrap=True , you are drawing with replacement, meaning that some data

clf = RandomForestClassifier(n_estimators=50, verbose=0, bootstrap=True, max
clf = clf.fit(X_train,Y_train)
```

In [330…
```
print(clf.feature_importances_)
```

```
[0.34 0.4  0.26]
```

The above is perhaps a powerful result: this indicates the relative power of each of the included features in classifying the students.

According to this, therefore, the relative contribution of each is:\

Extracurricular','Tutoring_Sessions','Physical_Activity']]

- Extracurricular: 34%
- Tutoring_Sessions: 40%
- Physical Activity: 26%

In [332…
```
accuracy = clf.score(X_train,Y_train) # make it a percentage and round to 2
print("The Random forest is {:.2f}% accurate for train dataset".format(accur
```

```
The Random forest is 75.43% accurate for train dataset
```

In [334…
```
from sklearn.metrics import confusion_matrix
predictions= clf.predict(X_train)
confusion_matrix(Y_train, predictions)
```

Out[334…
```
array([[3453,    0],
       [1125,    0]])
```

TEST DATA

In [336…
```
from sklearn.ensemble import RandomForestClassifier

# generate 100 decision trees
# verbose 0 (silent), 1 (progress bar) or 2 (one line per tree) you just say
# bootstrap=True , you are drawing with replacement, meaning that some data
```

```
clf = RandomForestClassifier(n_estimators=250, verbose=0, bootstrap=True, ma
clf = clf.fit(X_test,Y_test)
```

In [338…

```
predictions = clf.predict(X_test)
confusion_matrix(Y_test, predictions)
```

Out[338…

```
array([[1474,    0],
       [ 488,    0]])
```

In [340…

```
ccuracy=clf.score(X_test,Y_test)
print("The Random forest is {:.2f}% accurate for test dataset".format(accura
```

```
The Random forest is 75.43% accurate for test dataset
```

Since both the training and test accuracy are the same (75.43%), it indicates that the model is neither overfitting (i.e., performing too well on the training data but poorly on the test data) nor underfitting (i.e., not capturing enough patterns in both datasets). This suggests balanced performance and reliable generalization to new, unseen data.

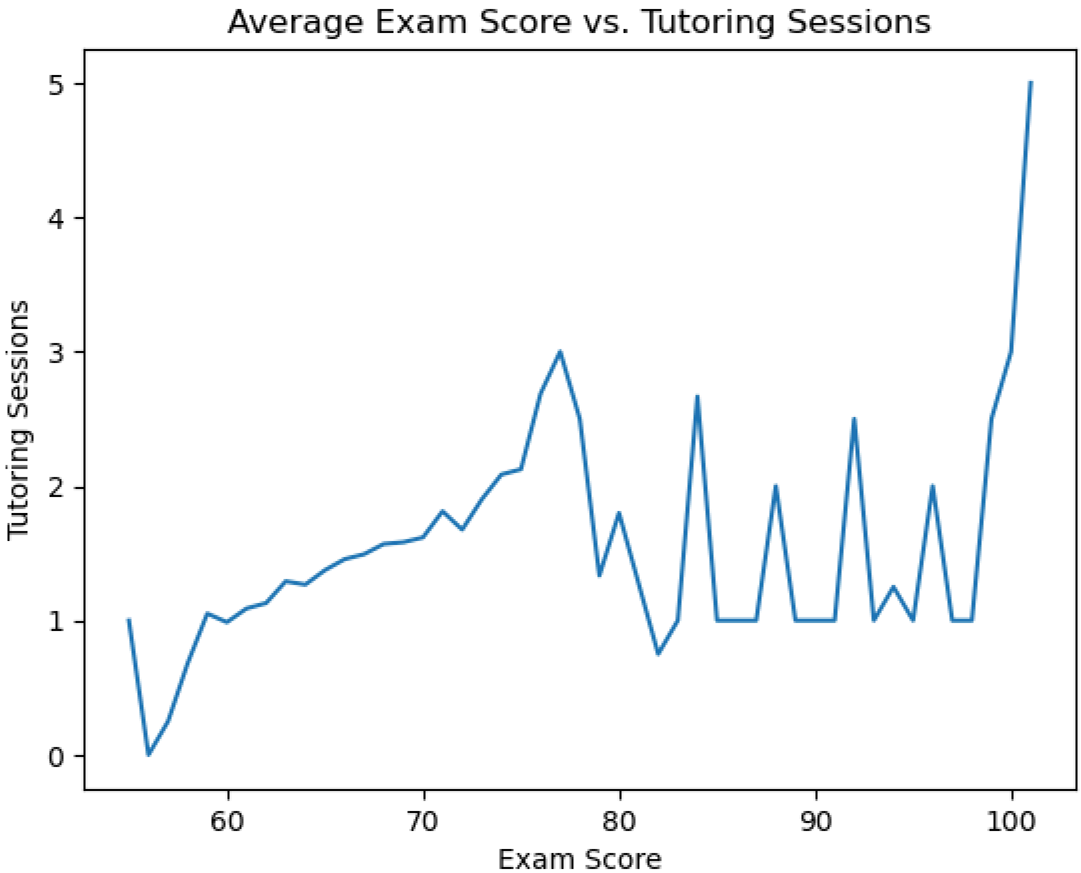
VISUALIZATIONS

In [68]:

```
df3= df1[['Tutoring_Sessions','Extracurricular_Activities','Physical_Activit
```

In [70]:

```
df3['Tutoring_Sessions'].groupby(df3['Exam_Score']).mean().plot(kind='line')
plt.xlabel('Exam Score')
plt.ylabel('Tutoring Sessions')
plt.title('Average Exam Score vs. Tutoring Sessions')
plt.show()
```


Average Exam Score vs. Tutoring Sessions
Tutoring Sessions
Exam Score
0
1
2
3
4
5
60
70
80
90
100