```
#Rakesh and Zurain: Data preparation
#Wolff and Zurain: Data Analysis
#Attila and Wolff: Findings and model(s) evaluation and conclusion
```

Student Game Performance Analytics

Part 1: Data Preparation

```
#data read
import pandas as pd

train = pd.read_csv('/data/notebook_files/data/train.csv', nrows=1000000)
train.head(10)
#sorted_train = train.sort_values('session_id', ascending=True)
#sorted_train.tail(10)
```

	session_id	index	elapsed_time	event_name	name	level	page	room_coor_x	room_coor_y
0	20090312431273200	0	0	cutscene_click	basic	0	NaN	-413.991405	-159.314686
1	20090312431273200	1	1323	person_click	basic	0	NaN	-413.991405	-159.314686
2	20090312431273200	2	831	person_click	basic	0	NaN	-413.991405	-159.314686
3	20090312431273200	3	1147	person_click	basic	0	NaN	-413.991405	-159.314686
4	20090312431273200	4	1863	person_click	basic	0	NaN	-412.991405	-159.314686
5	20090312431273200	5	3423	person_click	basic	0	NaN	-412.991405	-157.314686
6	20090312431273200	6	5197	person_click	basic	0	NaN	478.485079	-199.971679
7	20090312431273200	7	6180	person_click	basic	0	NaN	503.355128	-168.619913
8	20090312431273200	8	7014	person_click	basic	0	NaN	510.733442	-157.720642
9	20090312431273200	9	7946	person_click	basic	0	NaN	512.048005	-153.743631

```
display(train[-1:])
events = train.event_name.unique()
levels = train.level.unique()
names = train.name.unique()
events
levels
names
```

	session_id	index	elapsed_time	event_name	name	level	page	room_coor_x	room_
999999	20100412013046310	38	83131	cutscene_click	basic	1	NaN	569.743818	33.24

```
array(['basic', 'undefined', 'close', 'open', 'prev', 'next'],
      dtype=object)
```

```
labels = pd.read_csv("/data/notebook_files/data/train_labels.csv")
print(labels.shape)
labels.head(10)
```

```
(424116, 2)
```

	session_id	correct
0	20090312431273200_q1	1
1	20090312433251036_q1	0
2	20090312455206810_q1	1
3	20090313091715820_q1	0
4	20090313571836404_q1	1
5	20090314035813970_q1	1
6	20090314121766812_q1	1
7	20090314221187252_q1	0
8	20090314363702160_q1	1
9	20090314441803444_q1	1

```
labels['session'] = labels.session_id.apply(lambda x: int(x.split("_")[0]))
labels['question'] = labels.session_id.apply(lambda x: int(x.split("_")[-1][1
#sorted_lables = labels.sort_values('session', ascending=True)
#sorted_lables.head()
labels.head()
print(labels['question'].unique())
```

	session_id	correct	session	question
0	20090312431273200_q1	1	20090312431273200	1
1	20090312433251036_q1	0	20090312433251036	1
2	20090312455206810_q1	1	20090312455206810	1
3	20090313091715820_q1	0	20090313091715820	1
4	20090313571836404_q1	1	20090313571836404	1

```
for index, row in labels.iterrows():
    if row['session'] == 20100412013046310:
        print(index)
        break
```

```
885
```

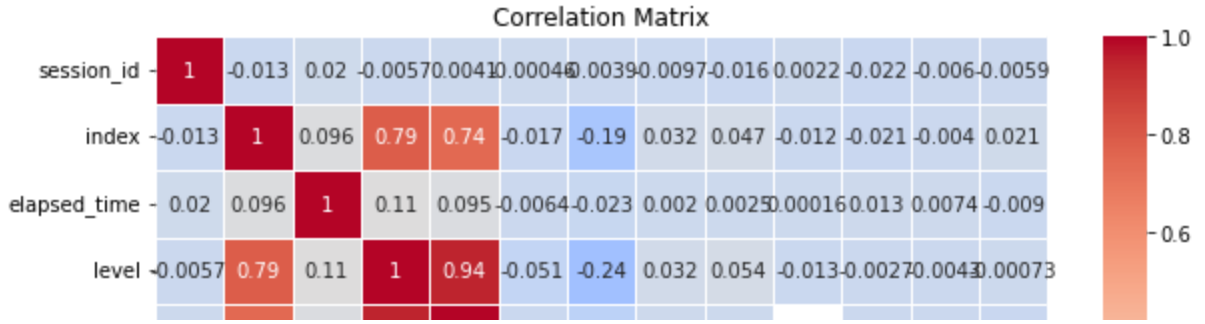
```
labels = labels[:885]
#labels.shape
```

```
train.describe()
```

	session_id	index	elapsed_time	level	page	room_coor_x	r
count	1.000000e+06	1000000.000000	1.000000e+06	1000000.000000	20577.000000	913257.000000	9
mean	2.009992e+16	656.770302	3.053446e+06	12.234207	3.174175	-53.998308	-
std	1.694178e+12	475.771925	1.618508e+07	6.483637	2.076302	517.629614	2
min	2.009031e+16	0.000000	0.000000e+00	0.000000	0.000000	-1986.058385	-
25%	2.010011e+16	291.000000	4.642658e+05	6.000000	1.000000	-350.621786	-
50%	2.010021e+16	602.000000	1.085220e+06	13.000000	3.000000	-9.124216	-
75%	2.010031e+16	908.000000	1.908770e+06	18.000000	5.000000	294.972010	2
max	2.010041e+16	4234.000000	4.293728e+08	22.000000	6.000000	1257.353920	5

```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = train.corr()
#Creating heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

Download




```
train = train.drop(columns = ['fullscreen', 'hq', 'music'])
```

```
CATS = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid']
NUMS = ['elapsed_time','level','page','room_coor_x', 'room_coor_y',
        'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

```
def feature_engineer(train):
    dfs = []
    for c in CATS:
        tmp = train.groupby(['session_id','level_group'])[c].agg('nunique')
        tmp.name = tmp.name + '_nunique'
        dfs.append(tmp)
    for c in NUMS:
        tmp = train.groupby(['session_id','level_group'])[c].agg('mean')
        dfs.append(tmp)
    for c in NUMS:
        tmp = train.groupby(['session_id','level_group'])[c].agg('std')
        tmp.name = tmp.name + '_std'
        dfs.append(tmp)
    df = pd.concat(dfs,axis=1)
    df = df.fillna(-1)
    df = df.reset_index()
    df = df.set_index('session_id')
    return df
```

```
df = feature_engineer(train)
print( df.shape )
df.head()
```

```
(2656, 22)
```

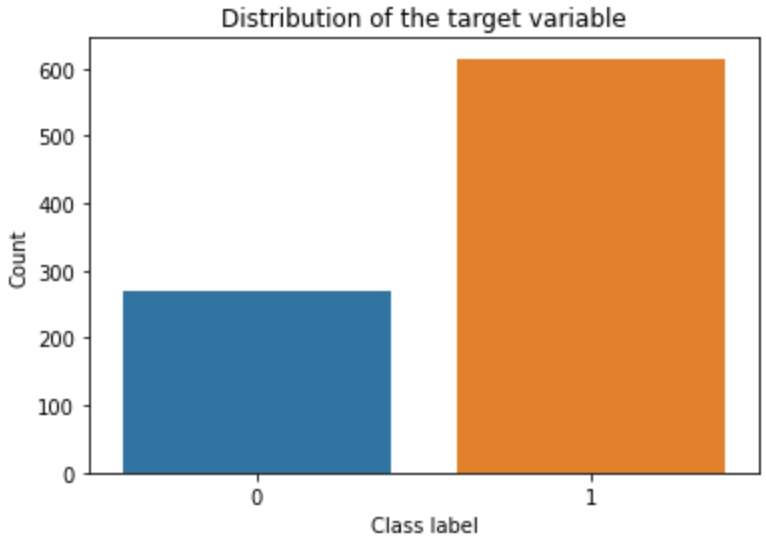
	level_group	event_name_nunique	name_nunique	fqid_nunique	room_fqid_nunique	t
session_id						
20090312431273200	0-4	10	3	30	7	1
20090312431273200	13-22	10	3	49	12	3
20090312431273200	5-12	10	3	39	11	2

Part 2: Data Analysis

```
df04 = df[df['level_group'] == "0-4"]
df512 = df[df['level_group'] == "5-12"]
df1322 = df[df['level_group'] == "13-22"]
```

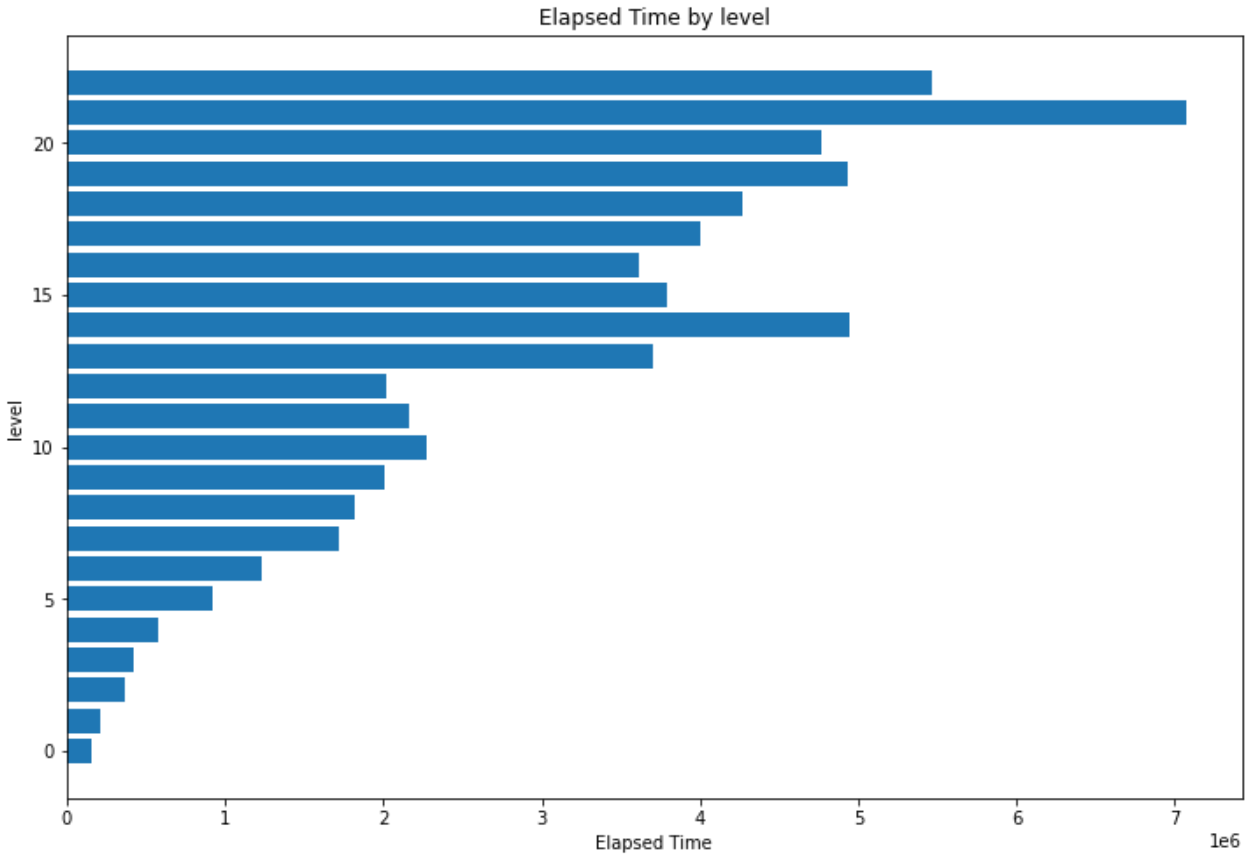
```
#Trying out some visualizations
sns.countplot(x='correct', data=labels)
plt.title("Distribution of the target variable")
plt.ylabel("Count")
plt.xlabel("Class label")
plt.show()
```

Download



```
#Visualizing how much time passes for each level
grouped_level=train.groupby(['level'])['elapsed_time'].mean().reset_index()
grouped_level.sort_values(inplace=True,by='elapsed_time')

plt.figure(figsize=(12,8))
plt.barh(y=grouped_level['level'],width=grouped_level['elapsed_time'])
plt.xlabel('Elapsed Time')
plt.ylabel('level')
plt.title('Elapsed Time by level')
plt.show()
```

Download



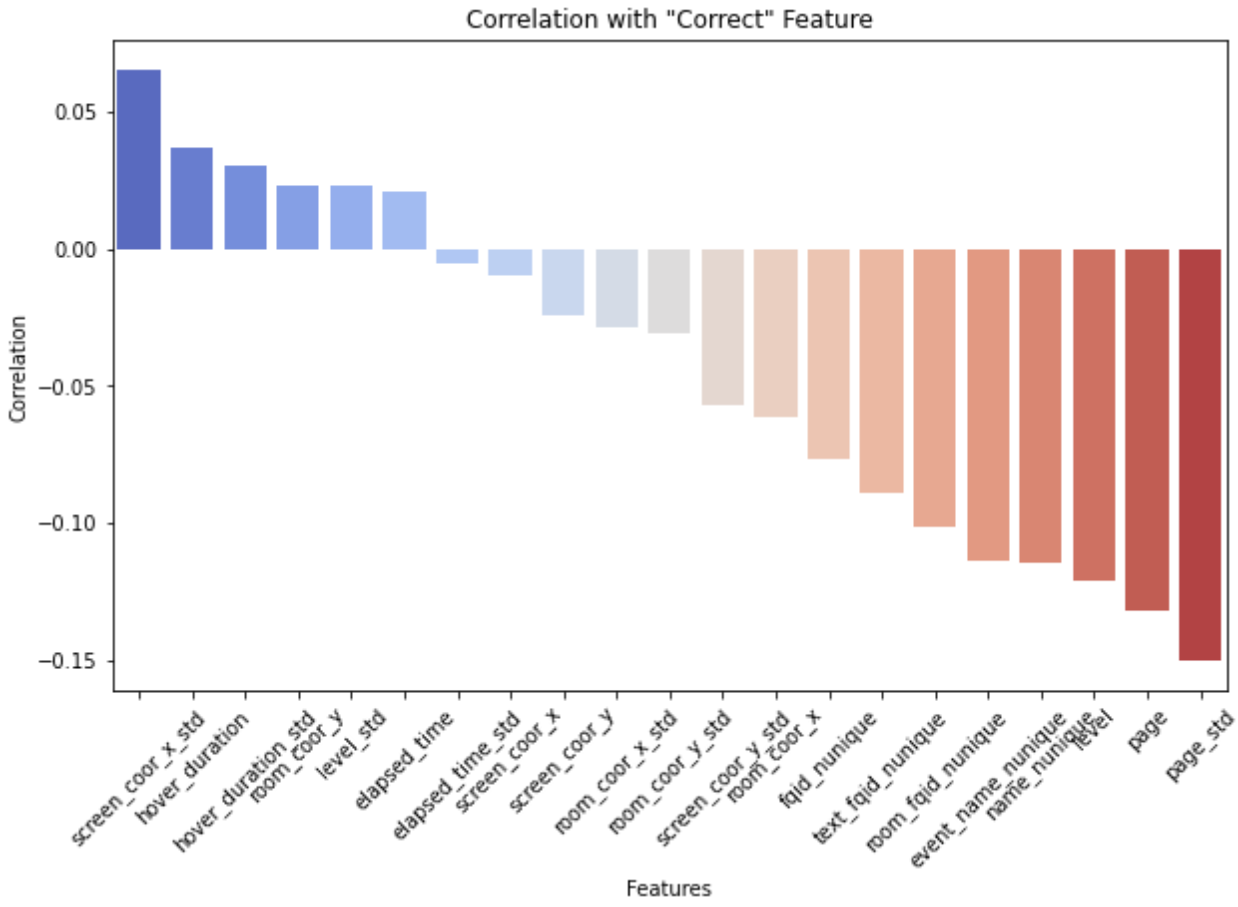

```
#Merging first 885 rows of training data with labels
#First with df04
df04_resized = df04[:885]
merged_df = df04_resized.assign(correct=labels['correct'].values)
merged_df.columns
```

```
Index(['level_group', 'event_name_nunique', 'name_nunique', 'fqid_nunique',
       'room_fqid_nunique', 'text_fqid_nunique', 'elapsed_time', 'level',
       'page', 'room_coor_x', 'room_coor_y', 'screen_coor_x', 'screen_coor_
       'hover_duration', 'elapsed_time_std', 'level_std', 'page_std',
       'room_coor_x_std', 'room_coor_y_std', 'screen_coor_x_std',
```

```
'screen_coor_y_std', 'hover_duration_std', 'correct'],
```

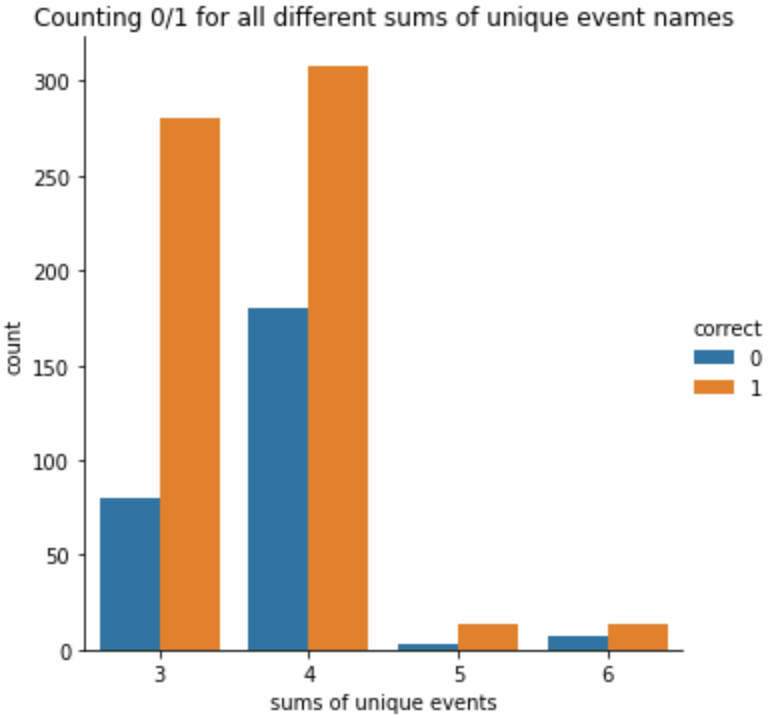
```
correlations = merged_df.corr()['correct'].drop('correct')
correlations = correlations.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x=correlations.index, y=correlations.values, palette='coolwarm')
plt.xticks(rotation=45)
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.title('Correlation with "Correct" Feature')

plt.show()
```

Download



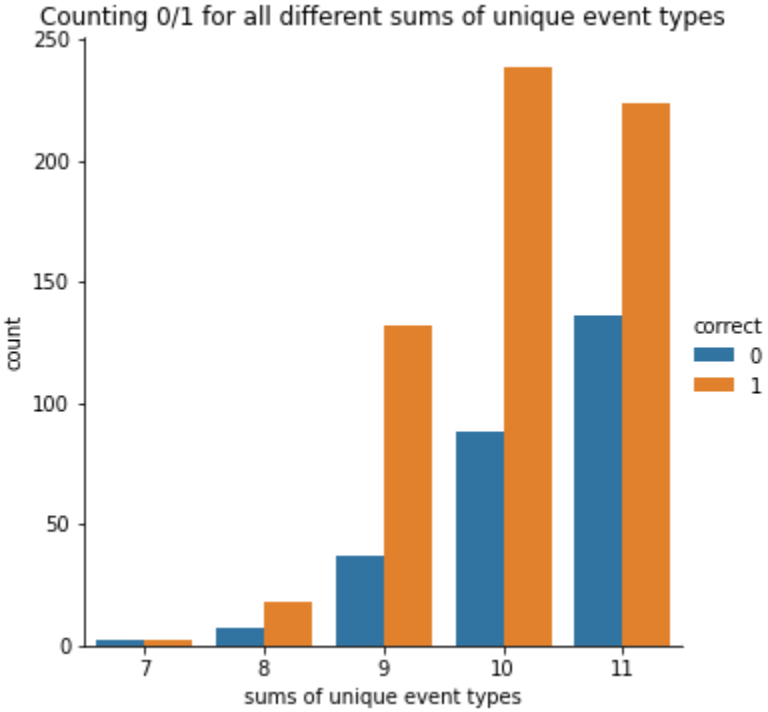

```
#Trying out some visualizations
#Counting 0/1 in each of the categories
sns.catplot(x='name_nunique', hue='correct', data = merged_df, kind='count')
plt.title('Counting 0/1 for all different sums of unique event names')
plt.xlabel("sums of unique events")
plt.show()
```

Download



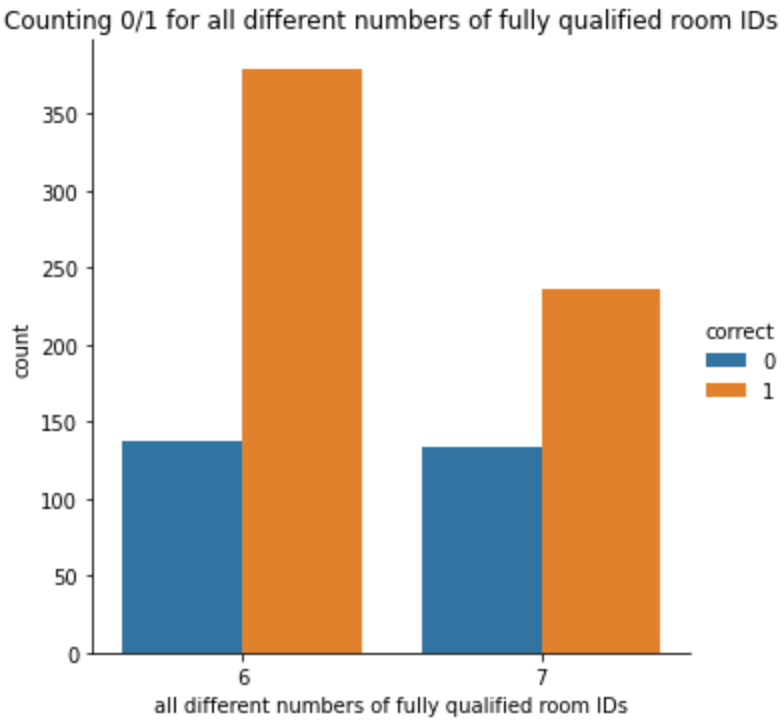

```
#Trying out some visualizations
#Counting 0/1 in each of the categories
sns.catplot(x='event_name_nunique', hue='correct', data = merged_df, kind='co
plt.title('Counting 0/1 for all different sums of unique event types')
plt.xlabel("sums of unique event types")
plt.show()
```

Download



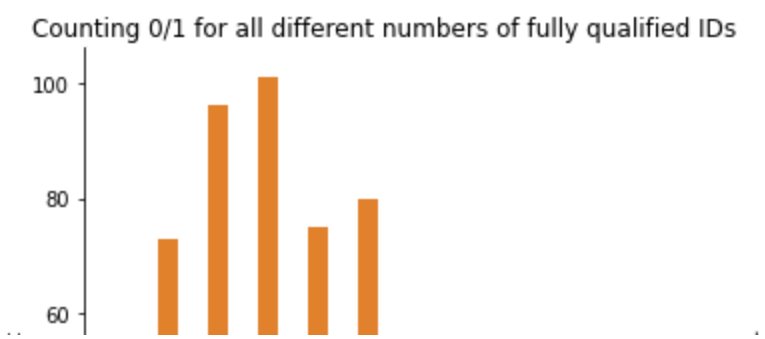
```
#Trying out some visualizations
#Counting 0/1 in each of the categories
sns.catplot(x='room_fqid_nunique', hue='correct', data = merged_df, kind='cou
plt.title('Counting 0/1 for all different numbers of fully qualified room IDs
plt.xlabel("all different numbers of fully qualified room IDs")
plt.show()
```

Download




```
#Trying out some visualizations
#Counting 0/1 in each of the categories
sns.catplot(x='fqid_nunique', hue='correct', data = merged_df, kind='count')
plt.title('Counting 0/1 for all different numbers of fully qualified IDs')
plt.xlabel("all different numbers of fully qualified IDs")
plt.show()
```

Download




```
#trying out Logit model
from sklearn.linear_model import LogisticRegression
#Creating model
logit_model = LogisticRegression()
#Formatting input training data
x_train = df04.drop('level_group', axis = 1)
x_train = x_train[:885]
y_train = labels.correct

length = int(len(x_train) * .8)
x_test = x_train[length:]
x_train = x_train[:length]
y_test = y_train[length:]
y_train = y_train[:length]

#print(x_train.describe())
#display(y_train)
#y_train.describe()

#Training the model
logit_model.fit(x_train,y_train)
#Prediction
y_pred = logit_model.predict(x_test)
y_pred
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1])
```

```
from sklearn.metrics import accuracy_score, classification_report

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generating a classification report
classification_rep = classification_report(y_test, y_pred, zero_division=1)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.7288135593220338
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        46
           1       0.74      0.98      0.84       131

    accuracy                           0.73       177
   macro avg       0.37      0.49      0.42       177
weighted avg       0.55      0.73      0.62       177
```

```
#Decision Tree model
from sklearn import tree
#Creating model
tree_model = tree.DecisionTreeClassifier()
#Formatting input training data
x_train = df04.drop('level_group', axis = 1)
x_train = x_train[:885]
y_train = labels.correct

length = int(len(x_train) * .8)
x_test = x_train[length:]
x_train = x_train[:length]
y_test = y_train[length:]
y_train = y_train[:length]

#print(x_train.describe())
display(y_train)
#y_train.describe()

#Training the model
tree_model.fit(x_train,y_train)
#Prediction
y_pred = tree_model.predict(x_test)
y_pred
```

```
0      1
1      0
2      1
```

```
3      0
4      1
      ..
703    1
704    1
705    1
706    0
707    1
Name: correct, Length: 708, dtype: int64
```

```
array([0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1])
```

```
# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generating a classification report
classification_rep = classification_report(y_test, y_pred, zero_division=1)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.5536723163841808
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.37      0.30        46
           1       0.74      0.62      0.67       131

    accuracy                           0.55       177
   macro avg       0.50      0.49      0.49       177
weighted avg       0.61      0.55      0.58       177
```

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
rf_classifier = RandomForestClassifier(n_estimators=750, max_features=2, min_
rf_classifier.fit(x_train, y_train)

predictions = rf_classifier.predict(x_test)
print(predictions)
```

```
[1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1]
```

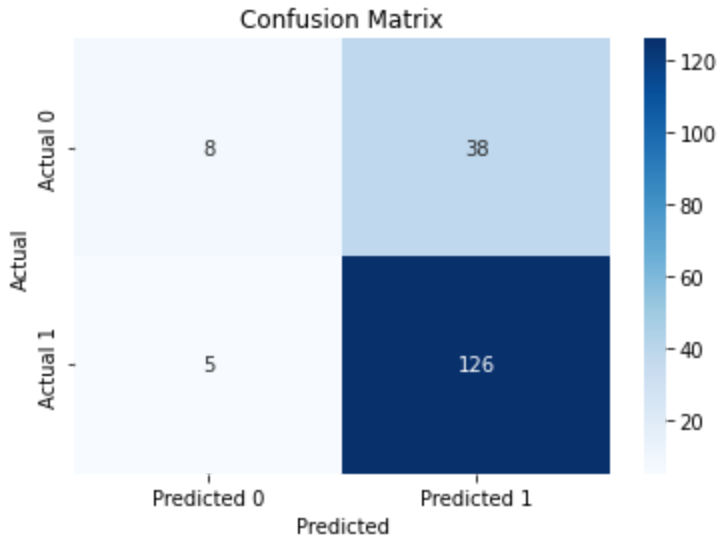
```
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Generating a classification report
classification_rep = classification_report(y_test, predictions, zero_division
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.7570621468926554
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.17      0.27        46
           1       0.77      0.96      0.85       131

    accuracy                           0.76       177
   macro avg       0.69      0.57      0.56       177
weighted avg       0.73      0.76      0.70       177
```

Part 3: Data Visualization

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions)

import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame from the confusion matrix
cm_df = pd.DataFrame(cm, index=['Actual 0', 'Actual 1'], columns=['Predicted

# Create a heatmap
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')

# Add labels, title, and axis ticks
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Show the plot
plt.show()
```

Download

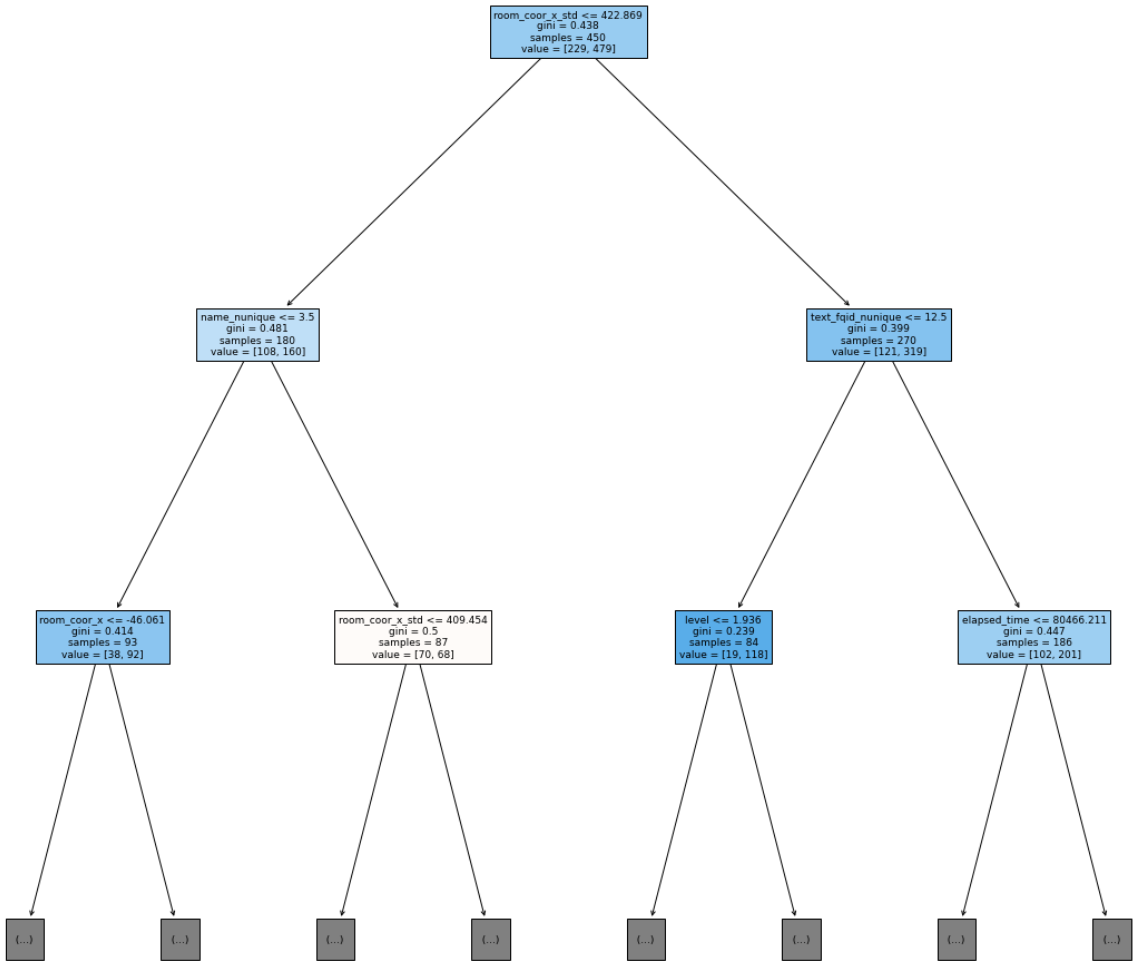



```
#One single decision tree in our forest
plt.figure(figsize=(20,20))
tree.plot_tree(rf_classifier.estimators_[0], feature_names=x_train.columns, f
```

```
[Text(558.0, 951.3000000000001, 'room_coor_x_std <= 422.869\ngini = 0.438\n
 Text(279.0, 679.5, 'name_nunique <= 3.5\ngini = 0.481\nsamples = 180\nvalu
 Text(139.5, 407.70000000000005, 'room_coor_x <= -46.061\ngini = 0.414\nsam
 Text(69.75, 135.89999999999998, '\n  (...)  \n'),
 Text(209.25, 135.89999999999998, '\n  (...)  \n'),
 Text(418.5, 407.70000000000005, 'room_coor_x_std <= 409.454\ngini = 0.5\ns
 Text(348.75, 135.89999999999998, '\n  (...)  \n'),
```

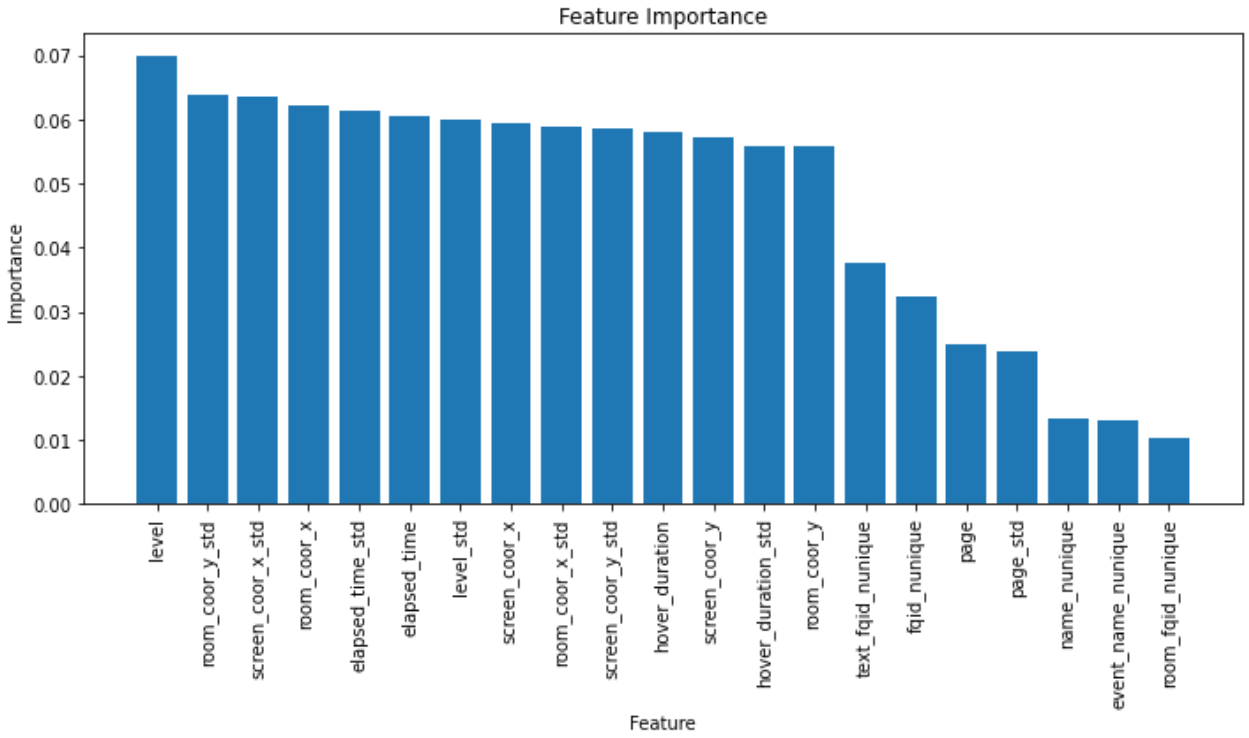
```
Text(488.25, 135.89999999999998, '\n  (...)  \n'),
Text(837.0, 679.5, 'text_fqid_nunique <= 12.5\ngini = 0.399\nsamples = 270
Text(697.5, 407.70000000000005, 'level <= 1.936\ngini = 0.239\nsamples = 8
Text(627.75, 135.89999999999998, '\n  (...)  \n'),
Text(767.25, 135.89999999999998, '\n  (...)  \n'),
Text(976.5, 407.70000000000005, 'elapsed_time <= 80466.211\ngini = 0.447\n
Text(906.75, 135.89999999999998, '\n  (...)  \n'),
Text(1046.25, 135.89999999999998, '\n  (...)  \n')]
```

Download



```
importances = rf_classifier.feature_importances_
importances = importances.astype(float)

#getting the feature list
feature_names = x_train.columns.tolist()


sorted_indices = np.argsort(importances)[::-1]
sorted_importances = importances[sorted_indices]
sorted_feature_names = [feature_names[i] for i in sorted_indices]

# Create a bar plot of feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(len(importances)), sorted_importances, tick_label=sorted_featur
plt.xticks(rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.tight_layout()
plt.show()
```

Download



```
##For when all data is added
##


for question in range(1, 19):
    if question < 4:
        group = '0-4'
    elif question < 14:
        group = '5-12'
    elif question < 23:
        group = '13-22'

    #get group specific variables
    traindf = x_train.loc[train.level_group == group]
    #vailddf = x_vaild.loc[train.level_group == group]
    #train y, figure out

    #train the model

    #predict on the valid data
```