Predicting if student answer questions correctly

Decision Forests are a family of tree-based models including Random Forests and Gradient Boosted Trees. They are the best place to start when working with tabular data, and will often outperform (or provide a strong baseline) before you begin experimenting with neural networks.

Import Libraries

In [1]:
```
import tensorflow as tf
import tensorflow_addons as tfa
import tensorflow_decision_forests as tfdf

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
print("TensorFlow Decision Forests v" + tfdf.__version__)
print("TensorFlow Addons v" + tfa.__version__)
print("TensorFlow v" + tf.__version__)
```

```
TensorFlow Decision Forests v1.2.0
TensorFlow Addons v0.19.0
TensorFlow v2.11.0
```

Load the Dataset

Since the dataset is extensive, some individuals may encounter memory errors when attempting to read it from the CSV file. To mitigate this issue, we aim to optimize the memory usage by Pandas during the loading and storage of the dataset.

By default, when Pandas loads a dataset, it automatically infers the data types of different columns. Regardless of the maximum value stored in these columns, Pandas assigns int64 for numerical columns, float64 for float columns, and object dtype for string columns, among others.

To potentially reduce the memory footprint, we can downcast numerical columns to smaller types such as int8, int32, float32, etc., if their maximum values do not necessitate the larger types (e.g., int64, float64).

Similarly, Pandas automatically identifies string columns as object datatype. To optimize memory usage for string columns storing categorical data, we can specify their datatype as category.

Given the characteristics of this dataset, many columns can likely be downcast to smaller types. We will provide a dictionary of dtypes for columns to Pandas while reading the dataset, implementing these optimizations.

In [3]:
```
dtypes={
    'elapsed_time':np.int32,
    'event_name':'category',
    'name':'category',
    'level':np.uint8,
    'room_coor_x':np.float32,
    'room_coor_y':np.float32,
    'screen_coor_x':np.float32,
    'screen_coor_y':np.float32,
    'hover_duration':np.float32,
    'text':'category',
    'fqid':'category',
    'room_fqid':'category',
    'text_fqid':'category',
    'fullscreen':'category',
    'hq':'category',
    'music':'category',
    'level_group':'category'}

dataset_df = pd.read_csv('/input/predict-student-performance-from-game-play/train.csv', dtype=dtypes)
print("Full train dataset shape is {}".format(dataset_df.shape))
```

```
Full train dataset shape is (26296946, 20)
```

The data is composed of 20 columns and 26296946 entries.

In [4]:
```
# Display the first 5 entries

dataset_df.head()
```

Out[4]:

	session_id	index	elapsed_time	event_name	name	level	page	room_coor_x	room_coor_y	screen_coor_x	screen_coor_y	hove
0	20090312431273200	0	0	cutscene_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	
1	20090312431273200	1	1323	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	
2	20090312431273200	2	831	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	
3	20090312431273200	3	1147	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	
4	20090312431273200	4	1863	person_click	basic	0	NaN	-412.991394	-159.314682	381.0	494.0	

`session_id` uniquely identifies a user session.

Load the labels

The labels for the training dataset are stored in the train_labels.csv file. This file contains information on whether the user in a particular session answered each question correctly. Load the labels data by executing the following code.

In [5]:
```
labels = pd.read_csv('/input/predict-student-performance-from-game-play/train_labels.csv')
```

Each value in the session_id column is a combination of both the session and the question number. To enhance usability, we will split these into individual columns.

In [6]:
```
labels['session'] = labels.session_id.apply(lambda x: int(x.split('_')[0]) )
labels['q'] = labels.session_id.apply(lambda x: int(x.split('_')[-1][1:]) )
```

Let us take a look at the first 5 entries of `labels` using the following code:

In [7]:
```
labels.head(5)
```

Out[7]:

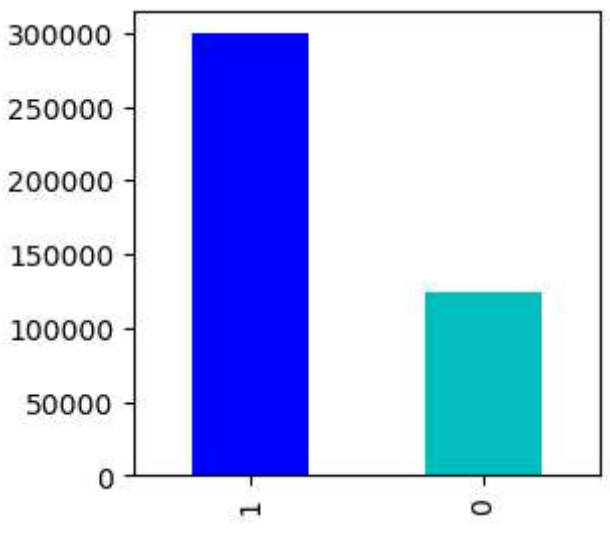
	session_id	correct	session	q
0	20090312431273200_q1	1	20090312431273200	1
1	20090312433251036_q1	0	20090312433251036	1
2	20090312455206810_q1	1	20090312455206810	1
3	20090313091715820_q1	0	20090313091715820	1
4	20090313571836404_q1	1	20090313571836404	1

Our goal is to train models for each question to predict the label "correct" for any input user session.

Bar chart for label column: correct

In [8]:
```
plt.figure(figsize=(3, 3))
plot_df = labels.correct.value_counts()
plot_df.plot(kind="bar", color=['b', 'c'])
```

Out[8]:
```
<AxesSubplot:>
```

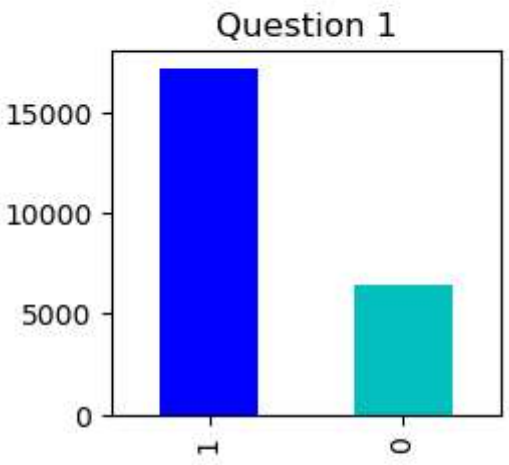


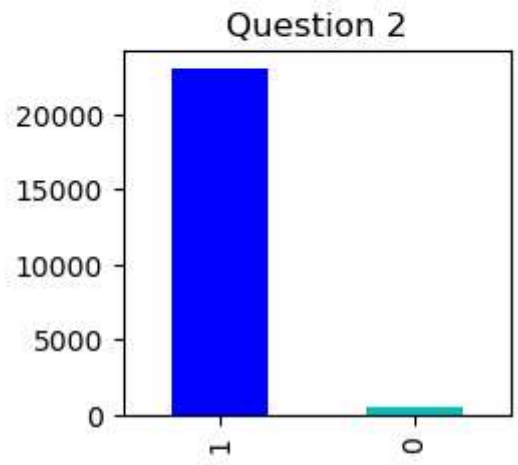
plot the values of the label column `correct` for each question.

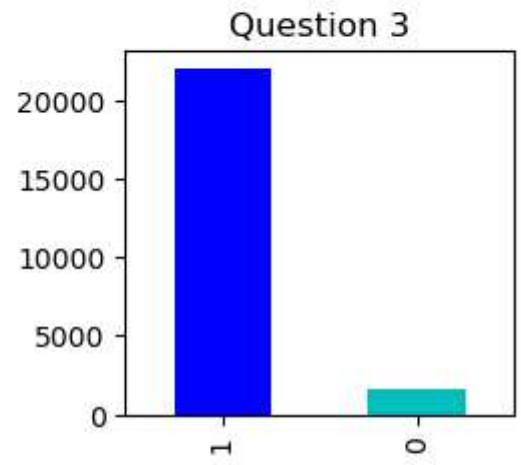
In [9]:
```
plt.figure(figsize=(10, 20))
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.suptitle("\"Correct\" column values for each question", fontsize=14, y=0.94)
for n in range(1,19):
    #print(n, str(n))
    ax = plt.subplot(6, 3, n)
```

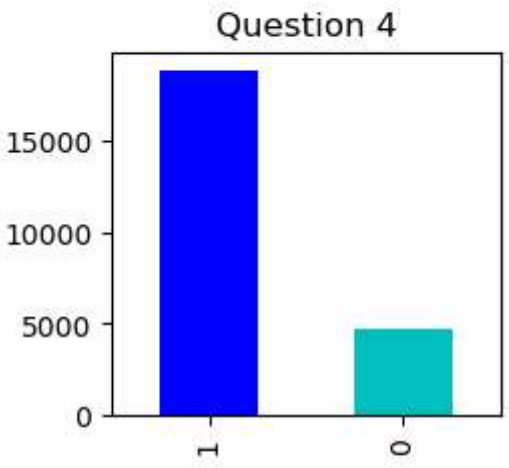
```
    # filter df and plot ticker on the new subplot axis
    plot_df = labels.loc[labels.q == n]
    plot_df = plot_df.correct.value_counts()
    plot_df.plot(ax=ax, kind="bar", color=['b', 'c'])

    # chart formatting
    ax.set_title("Question " + str(n))
    ax.set_xlabel("")
```

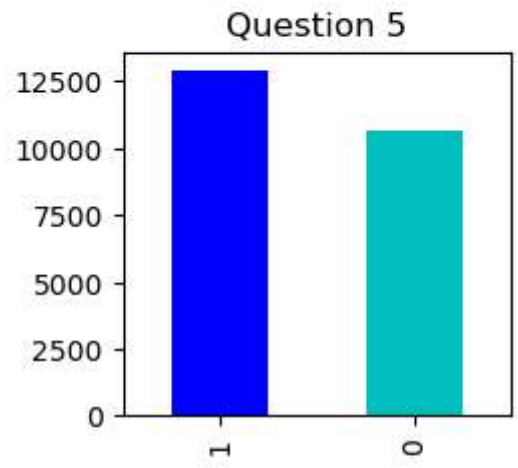
"Correct" column values for each question

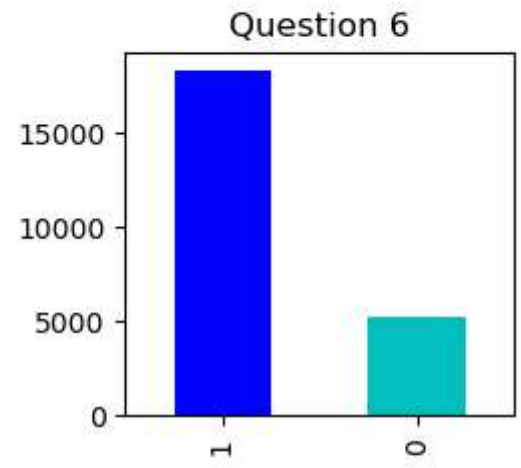


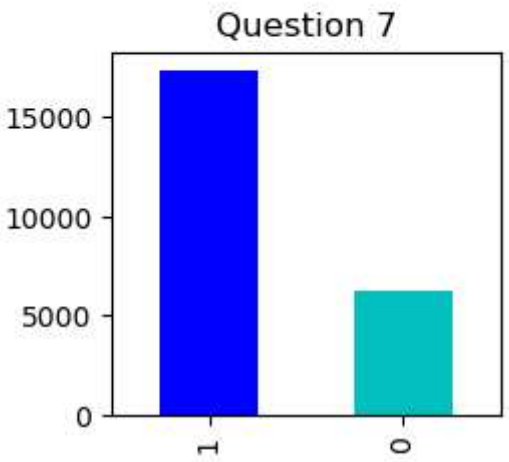


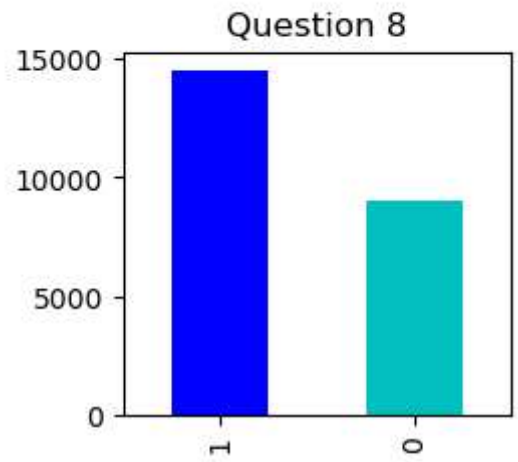


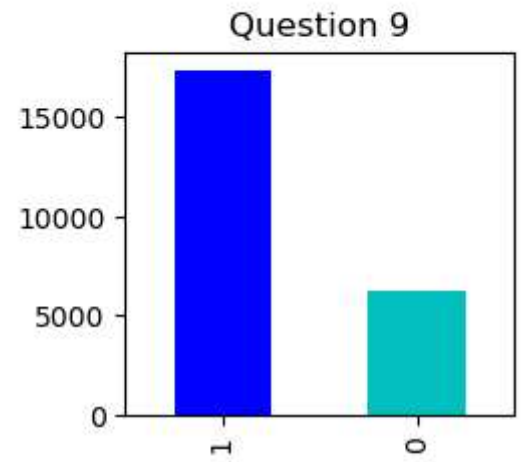


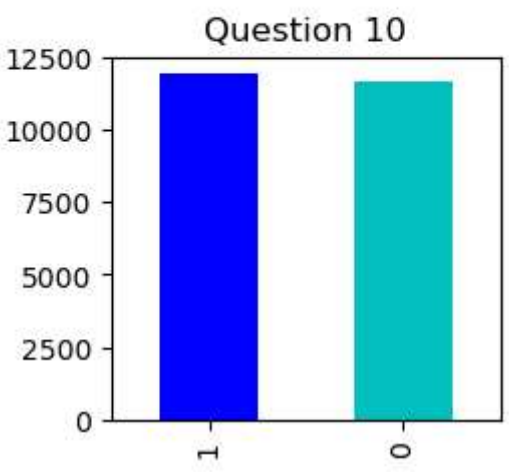


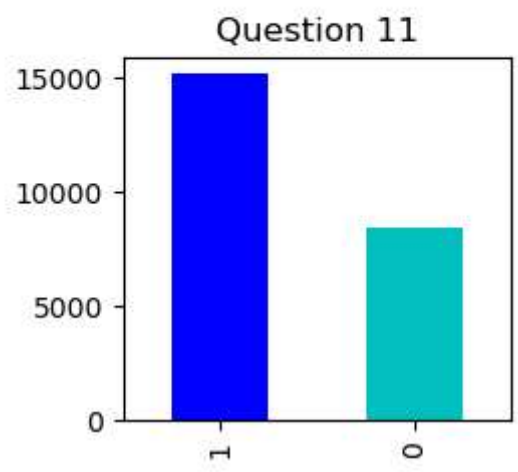


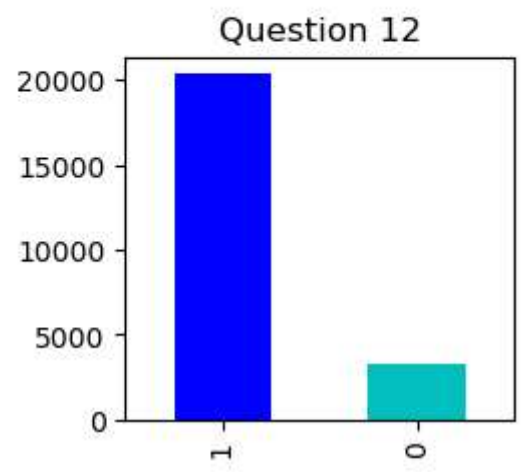


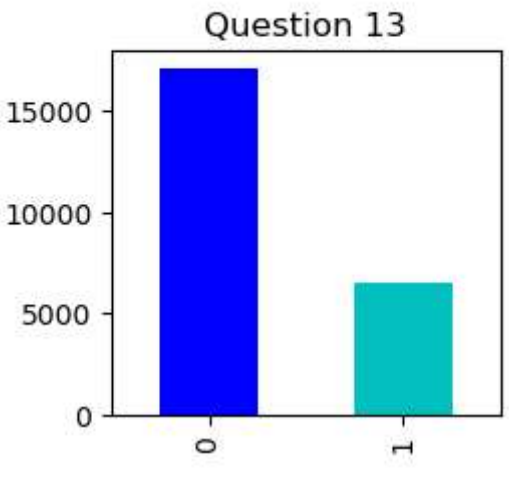


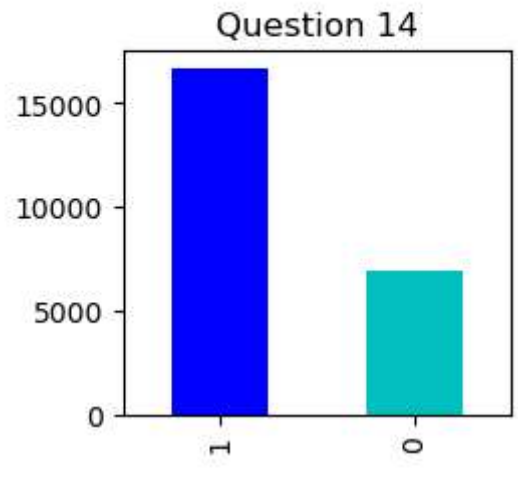


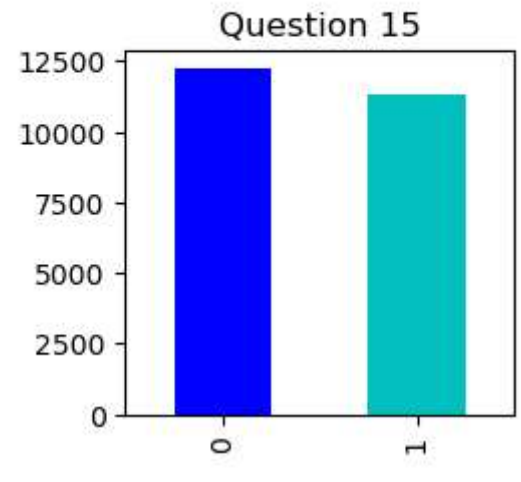


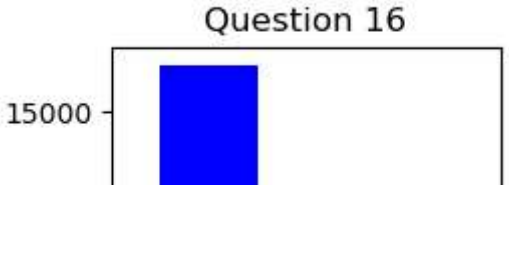


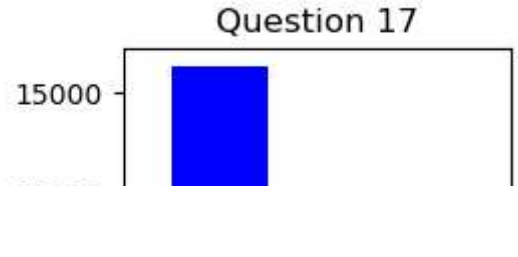


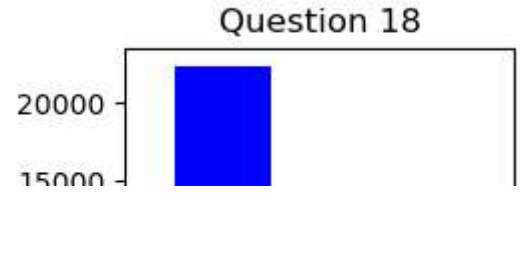


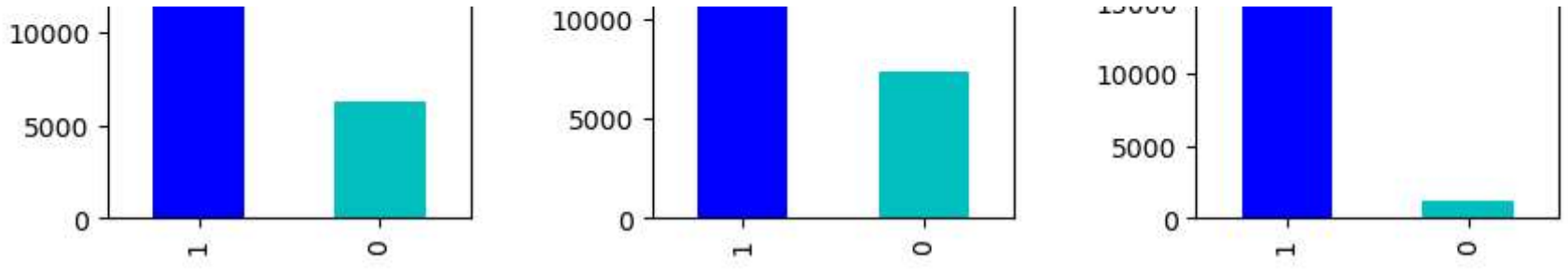












Prepare the dataset

the dataset presents the questions and data to us in the order of levels - level segments (represented by the column level_group) 0-4, 5-12, and 13-22. Our task is to predict the correctness of each segment's questions as they are presented. To accomplish this, we will generate basic aggregate features from the relevant columns, and additional features can be created to enhance model performance.

Firstly, we will create two separate lists containing the names of the Categorical columns and Numerical columns. We will exclude the columns fullscreen, hq, and music as they do not contribute useful information for this problem statement.

In [10]:
```
CATEGORICAL = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid']
NUMERICAL = ['elapsed_time','level','page','room_coor_x', 'room_coor_y',
       'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

For each categorical column, we will initially group the dataset by session_id and level_group. Subsequently, we will count the number of distinct elements in the column for each group and store this information temporarily.

For all numerical columns, we will group the dataset by session_id and level_group. Instead of counting the number of distinct elements, we will calculate the mean and standard deviation of the numerical column for each group and store these statistics temporarily.

Following this, we will concatenate the temporary data frames generated in the previous step for each column to create our new feature-engineered dataset.

In [11]:
```
def feature_engineer(dataset_df):
    dfs = []
    for c in CATEGORICAL:
        tmp = dataset_df.groupby(['session_id','level_group'])[c].agg('nunique')
        tmp.name = tmp.name + '_nunique'
        dfs.append(tmp)
    for c in NUMERICAL:
        tmp = dataset_df.groupby(['session_id','level_group'])[c].agg('mean')
        dfs.append(tmp)
    for c in NUMERICAL:
        tmp = dataset_df.groupby(['session_id','level_group'])[c].agg('std')
        tmp.name = tmp.name + '_std'
        dfs.append(tmp)
    dataset_df = pd.concat(dfs,axis=1)
    dataset_df = dataset_df.fillna(-1)
    dataset_df = dataset_df.reset_index()
    dataset_df = dataset_df.set_index('session_id')
    return dataset_df
```

In [12]:
```
dataset_df = feature_engineer(dataset_df)
print("Full prepared dataset shape is {}".format(dataset_df.shape))
```

```
Full prepared dataset shape is (70686, 22)
```

Basic exploration of the prepared dataset

In [13]:
```
dataset_df.head()
```

Out[13]:

session_id	level_group	event_name_nunique	name_nunique	fqid_nunique	room_fqid_nunique	text_fqid_nunique	elapsed_time	
20090312431273200	0-4	10	3	30	7	17	8.579356e+04	1
20090312431273200	13-22	10	3	49	12	35	1.040601e+06	17
20090312431273200	5-12	10	3	39	11	24	3.572052e+05	8
20090312433251036	0-4	11	4	22	6	11	9.763342e+04	1
20090312433251036	13-22	11	6	73	16	43	2.498852e+06	17

5 rows × 22 columns

In [14]:
```
dataset_df.describe()
```

Out[14]:

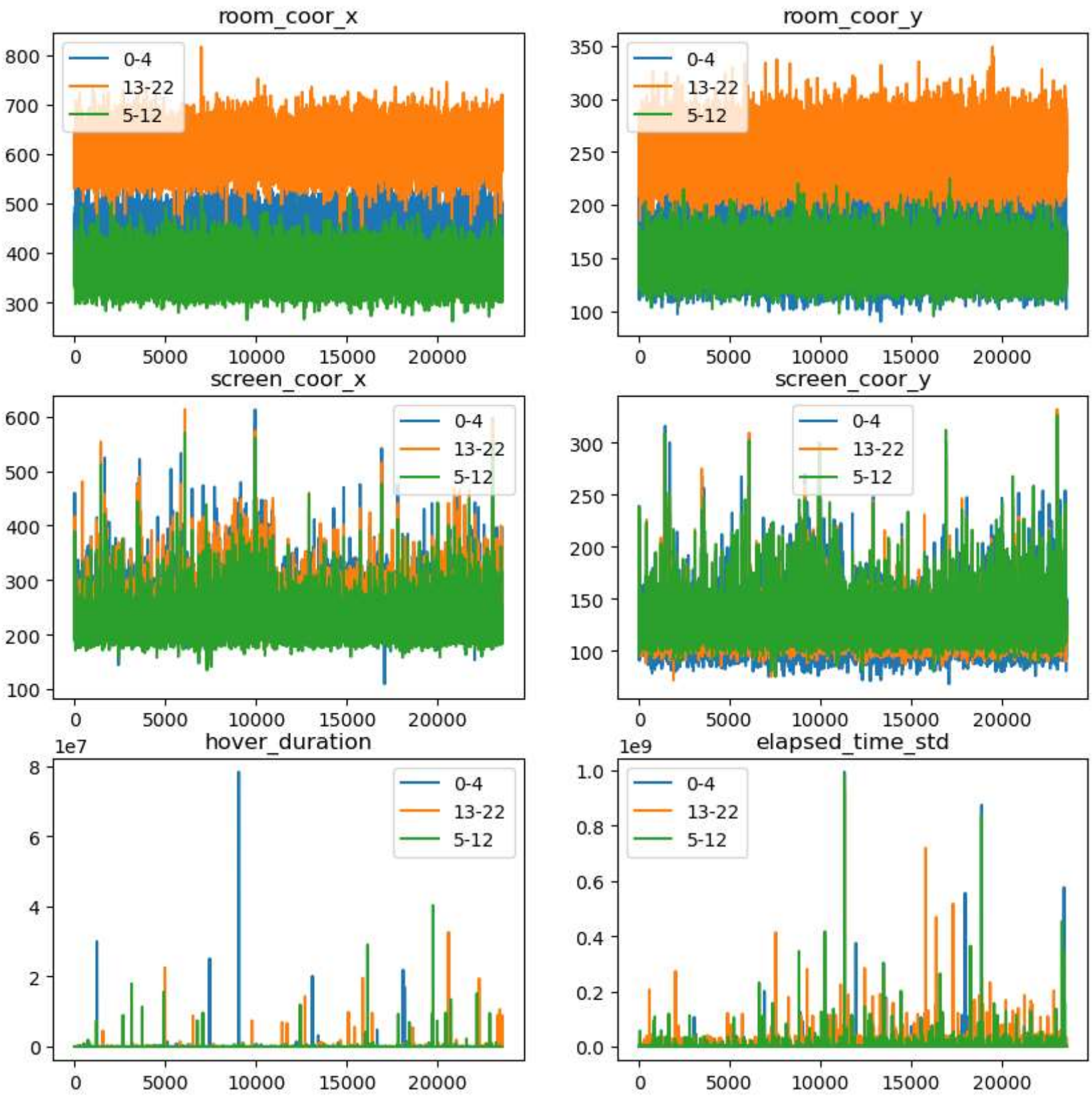
	event_name_nunique	name_nunique	fqid_nunique	room_fqid_nunique	text_fqid_nunique	elapsed_time	level	page
count	70686.000000	70686.000000	70686.000000	70686.000000	70686.000000	7.068600e+04	70686.000000	70686.000000
mean	10.390021	3.885324	40.468650	10.268908	23.838412	2.811806e+06	9.311221	1.439995
std	0.832923	0.799293	13.020929	2.928351	8.989095	2.289877e+07	6.523788	2.350494
min	7.000000	3.000000	18.000000	5.000000	8.000000	4.591262e+02	0.923372	-1.000000
25%	10.000000	3.000000	25.000000	7.000000	15.000000	1.642777e+05	2.051613	-1.000000
50%	11.000000	4.000000	43.000000	11.000000	23.000000	5.882137e+05	8.140526	1.000000
75%	11.000000	4.000000	51.000000	12.000000	32.000000	1.351434e+06	17.592593	4.000000
max	11.000000	6.000000	81.000000	17.000000	48.000000	1.191886e+09	20.222101	6.000000

8 rows × 21 columns

Numerical data distribution

plot some numerical columns and their value against each level_group:

In [15]:
```
figure, axis = plt.subplots(3, 2, figsize=(10, 10))

for name, data in dataset_df.groupby('level_group'):
    axis[0, 0].plot(range(1, len(data['room_coor_x_std'])+1), data['room_coor_x_std'], label=name)
    axis[0, 1].plot(range(1, len(data['room_coor_y_std'])+1), data['room_coor_y_std'], label=name)
    axis[1, 0].plot(range(1, len(data['screen_coor_x_std'])+1), data['screen_coor_x_std'], label=name)
    axis[1, 1].plot(range(1, len(data['screen_coor_y_std'])+1), data['screen_coor_y_std'], label=name)
    axis[2, 0].plot(range(1, len(data['hover_duration'])+1), data['hover_duration_std'], label=name)
    axis[2, 1].plot(range(1, len(data['elapsed_time_std'])+1), data['elapsed_time_std'], label=name)


axis[0, 0].set_title('room_coor_x')
axis[0, 1].set_title('room_coor_y')
axis[1, 0].set_title('screen_coor_x')
axis[1, 1].set_title('screen_coor_y')
axis[2, 0].set_title('hover_duration')
axis[2, 1].set_title('elapsed_time_std')

for i in range(3):
    axis[i, 0].legend()
    axis[i, 1].legend()

plt.show()
```



split the dataset into training and testing datasets:

In [16]:
```
def split_dataset(dataset, test_ratio=0.20):
    USER_LIST = dataset.index.unique()
    split = int(len(USER_LIST) * (1 - 0.20))
    return dataset.loc[USER_LIST[:split]], dataset.loc[USER_LIST[split:]]

train_x, valid_x = split_dataset(dataset_df)
print("{} examples in training, {} examples in testing.".format(
    len(train_x), len(valid_x)))
```

```
56547 examples in training, 14139 examples in testing.
```

Select a Model

There are several tree-based models for you to choose from.

- RandomForestModel
- GradientBoostedTreesModel
- CartModel
- DistributedGradientBoostedTreesModel

We can list all the available models in TensorFlow Decision Forests using the following code:

In [17]:
```
tfdf.keras.get_all_models()
```

Out[17]:
```
[tensorflow_decision_forests.keras.RandomForestModel,
 tensorflow_decision_forests.keras.GradientBoostedTreesModel,
 tensorflow_decision_forests.keras.CartModel,
 tensorflow_decision_forests.keras.DistributedGradientBoostedTreesModel]
```

To get started, we'll work with a Gradient Boosted Trees Model. This is one of the well-known Decision Forest training algorithms.

A Gradient Boosted Decision Tree is a set of shallow decision trees trained sequentially. Each tree is trained to predict and then "correct" for the errors of the previously trained trees.

How can I configure a tree-based model?

TensorFlow Decision Forests provides good defaults for you (e.g., the top-ranking hyperparameters on our benchmarks, slightly modified to run in a reasonable time). If you would like to configure the learning algorithm, you will find many options to explore and achieve the highest possible accuracy.

Training

We will train a model for each question to predict if the question will be answered correctly by a user. There are a total of 18 questions in the dataset. Hence, we will be training 18 models, one for each question.

To support our training loop, we need to provide a few data structures to store the trained models, predictions on the validation set, and evaluation scores for the trained models.

We will create these using the following code:

In [18]:
```
# Fetch the unique list of user sessions in the validation dataset. We assigned
# `session_id` as the index of our feature engineered dataset. Hence fetching
# the unique values in the index column will give us a list of users in the
# validation set.
VALID_USER_LIST = valid_x.index.unique()

# Create a dataframe for storing the predictions of each question for all users
# in the validation set.
# For this, the required size of the data frame is:
# (no: of users in validation set  x no of questions).
# We will initialize all the predicted values in the data frame to zero.
# The dataframe's index column is the user `session_id`s.
prediction_df = pd.DataFrame(data=np.zeros((len(VALID_USER_LIST),18)), index=VALID_USER_LIST)

# Create an empty dictionary to store the models created for each question.
models = {}

# Create an empty dictionary to store the evaluation score for each question.
evaluation_dict ={}
```

Before training the data, we need to understand how level_groups and questions are associated with each other.

In this game, the first quiz checkpoint (i.e., questions 1 to 3) comes after finishing levels 0 to 4. So, for training questions 1 to 3, we will use data from the level_group 0-4. Similarly, we will use data from the level_group 5-12 to train questions from 4 to 13 and data from the level_group 13-22 to train questions from 14 to 18.

We will train a model for each question and store the trained model in the models dictionary.

In [19]:
```
# Iterate through questions 1 to 18 to train models for each question, evaluate
# the trained model and store the predicted values.
for q_no in range(1,19):

    # Select level group for the question based on the q_no.
    if q_no<=3: grp = '0-4'
    elif q_no<=13: grp = '5-12'
    elif q_no<=22: grp = '13-22'
    print("### q_no", q_no, "grp", grp)


    # Filter the rows in the datasets based on the selected level group.
    train_df = train_x.loc[train_x.level_group == grp]
    train_users = train_df.index.values
    valid_df = valid_x.loc[valid_x.level_group == grp]
    valid_users = valid_df.index.values

    # Select the labels for the related q_no.
    train_labels = labels.loc[labels.q==q_no].set_index('session').loc[train_users]
    valid_labels = labels.loc[labels.q==q_no].set_index('session').loc[valid_users]

    # Add the label to the filtered datasets.
    train_df["correct"] = train_labels["correct"]
    valid_df["correct"] = valid_labels["correct"]

    # There's one more step required before we can train the model.
    # We need to convert the datatset from Pandas format (pd.DataFrame)
    # into TensorFlow Datasets format (tf.data.Dataset).
    # TensorFlow Datasets is a high performance data loading library
    # which is helpful when training neural networks with accelerators like GPUs and TPUs.
    # We are omitting `level_group`, since it is not needed for training anymore.
    train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_df.loc[:, train_df.columns != 'level_group'], label="correct
    valid_ds = tfdf.keras.pd_dataframe_to_tf_dataset(valid_df.loc[:, valid_df.columns != 'level_group'], label="correct

    # We will now create the Gradient Boosted Trees Model with default settings.
    # By default the model is set to train for a classification task.
    gbtm = tfdf.keras.GradientBoostedTreesModel(verbose=0)
    gbtm.compile(metrics=["accuracy"])

    # Train the model.
    gbtm.fit(x=train_ds)

    # Store the model
    models[f'{grp}_{q_no}'] = gbtm

    # Evaluate the trained model on the validation dataset and store the
    # evaluation accuracy in the `evaluation_dict`.
```

```
    inspector = gbtm.make_inspector()
    inspector.evaluation()
    evaluation = gbtm.evaluate(x=valid_ds,return_dict=True)
    evaluation_dict[q_no] = evaluation["accuracy"]

    # Use the trained model to make predictions on the validation dataset and
    # store the predicted values in the `prediction_df` dataframe.
    predict = gbtm.predict(x=valid_ds)
    prediction_df.loc[valid_users, q_no-1] = predict.flatten()
```

```
### q_no 1 grp 0-4
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
[INFO 2023-10-21T22:00:54.551720665+00:00 kernel.cc:1214] Loading model from path /tmp/tmpjx4g_0kr/model/ with prefix 448beaebeaf640dd
[INFO 2023-10-21T22:00:54.563735076+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" built
[INFO 2023-10-21T22:00:54.563829909+00:00 kernel.cc:1046] Use fast generic engine
WARNING: AutoGraph could not transform <function simple_ml_inference_op_with_handle at 0x7fc467c52710> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: could not get source code
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
5/5 [==============================] - 1s 12ms/step - loss: 0.0000e+00 - accuracy: 0.7286
5/5 [==============================] - 0s 13ms/step
### q_no 2 grp 0-4
[INFO 2023-10-21T22:01:00.438810644+00:00 kernel.cc:1214] Loading model from path /tmp/tmpg9nyahgl/model/ with prefix caa37d6a31424af0
[INFO 2023-10-21T22:01:00.441415714+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 12ms/step - loss: 0.0000e+00 - accuracy: 0.9743
5/5 [==============================] - 0s 12ms/step
### q_no 3 grp 0-4
[INFO 2023-10-21T22:01:03.799086222+00:00 kernel.cc:1214] Loading model from path /tmp/tmp54t5h21p/model/ with prefix b65639d16ca74f96
[INFO 2023-10-21T22:01:03.801817951+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.9351
5/5 [==============================] - 0s 12ms/step
### q_no 4 grp 5-12
[INFO 2023-10-21T22:01:10.168376107+00:00 kernel.cc:1214] Loading model from path /tmp/tmpjq6f8tau/model/ with prefix a5d35b7a1ccf4da8
[INFO 2023-10-21T22:01:10.181249717+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" built
[INFO 2023-10-21T22:01:10.181304396+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.7957
5/5 [==============================] - 0s 12ms/step
### q_no 5 grp 5-12
[INFO 2023-10-21T22:01:14.458160475+00:00 kernel.cc:1214] Loading model from path /tmp/tmpsfydt_an/model/ with prefix bd2859db3b7d4383
[INFO 2023-10-21T22:01:14.464772939+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.6319
5/5 [==============================] - 0s 12ms/step
### q_no 6 grp 5-12
[INFO 2023-10-21T22:01:19.637097678+00:00 kernel.cc:1214] Loading model from path /tmp/tmpmcmesgnb/model/ with prefix 36c054acf7874a4b
[INFO 2023-10-21T22:01:19.645516974+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.7885
5/5 [==============================] - 0s 12ms/step
### q_no 7 grp 5-12
[INFO 2023-10-21T22:01:23.608355434+00:00 kernel.cc:1214] Loading model from path /tmp/tmp4gwbxdtz/model/ with prefix 3395ba5a2c77471b
[INFO 2023-10-21T22:01:23.613544221+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" built
[INFO 2023-10-21T22:01:23.613580539+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.7456
5/5 [==============================] - 0s 12ms/step
### q_no 8 grp 5-12
[INFO 2023-10-21T22:01:26.951182113+00:00 kernel.cc:1214] Loading model from path /tmp/tmpwdbb2bpp/model/ with prefix 34af9638ba404aaa
[INFO 2023-10-21T22:01:26.954292506+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.6355
5/5 [==============================] - 0s 12ms/step
### q_no 9 grp 5-12
[INFO 2023-10-21T22:01:31.140619458+00:00 kernel.cc:1214] Loading model from path /tmp/tmpv2pk7b0q/model/ with prefix ba48e9839e65493f
[INFO 2023-10-21T22:01:31.146977359+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.7632
5/5 [==============================] - 0s 12ms/step
### q_no 10 grp 5-12
```

```
[INFO 2023-10-21T22:01:36.365585399+00:00 kernel.cc:1214] Loading model from path /tmp/tmpq6aq5567/model/ with prefix 4
1c525e340d447e0
[INFO 2023-10-21T22:01:36.374606773+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" buil
t
[INFO 2023-10-21T22:01:36.374649306+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.6109
5/5 [==============================] - 0s 12ms/step
### q_no 11 grp 5-12
[INFO 2023-10-21T22:01:40.807195876+00:00 kernel.cc:1214] Loading model from path /tmp/tmpfihkxkgq/model/ with prefix f
a71f43f7c66460e
[INFO 2023-10-21T22:01:40.814880237+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 12ms/step - loss: 0.0000e+00 - accuracy: 0.6522
5/5 [==============================] - 0s 16ms/step
### q_no 12 grp 5-12
[INFO 2023-10-21T22:01:44.333076762+00:00 kernel.cc:1214] Loading model from path /tmp/tmphzobz4w4/model/ with prefix b
3977165202d4798
[INFO 2023-10-21T22:01:44.336673953+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 12ms/step - loss: 0.0000e+00 - accuracy: 0.8695
5/5 [==============================] - 0s 12ms/step
### q_no 13 grp 5-12
[INFO 2023-10-21T22:01:49.574052667+00:00 kernel.cc:1214] Loading model from path /tmp/tmprrlk4y6g/model/ with prefix 2
540fb2e1fb5449a
[INFO 2023-10-21T22:01:49.582974463+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" buil
t
[INFO 2023-10-21T22:01:49.583017776+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.7218
5/5 [==============================] - 0s 13ms/step
### q_no 14 grp 13-22
[INFO 2023-10-21T22:01:54.059239681+00:00 kernel.cc:1214] Loading model from path /tmp/tmpvdmlnpaz/model/ with prefix 5
badf521f0c54c8d
[INFO 2023-10-21T22:01:54.066392567+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.7337
5/5 [==============================] - 0s 14ms/step
### q_no 15 grp 13-22
[INFO 2023-10-21T22:01:59.399329445+00:00 kernel.cc:1214] Loading model from path /tmp/tmpm47dqgue/model/ with prefix 2
784ca8141554a1d
[INFO 2023-10-21T22:01:59.408196804+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.6166
5/5 [==============================] - 0s 12ms/step
### q_no 16 grp 13-22
[INFO 2023-10-21T22:02:02.908800684+00:00 kernel.cc:1214] Loading model from path /tmp/tmpii5v9yco/model/ with prefix 4
b48d016f9e74ad3
[INFO 2023-10-21T22:02:02.911412758+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" buil
t
[INFO 2023-10-21T22:02:02.911443023+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.7486
5/5 [==============================] - 0s 13ms/step
### q_no 17 grp 13-22
[INFO 2023-10-21T22:02:06.895415711+00:00 kernel.cc:1214] Loading model from path /tmp/tmpgbsch6dr/model/ with prefix 3
4f946dae89e4d1c
[INFO 2023-10-21T22:02:06.90046158+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.7027
5/5 [==============================] - 0s 15ms/step
### q_no 18 grp 13-22
[INFO 2023-10-21T22:02:18.045037355+00:00 kernel.cc:1214] Loading model from path /tmp/tmpiuhnxlsm/model/ with prefix 5
9daf4db054841d3
[INFO 2023-10-21T22:02:18.070350071+00:00 abstract_model.cc:1311] Engine "GradientBoostedTreesQuickScorerExtended" buil
t
[INFO 2023-10-21T22:02:18.070381574+00:00 kernel.cc:1046] Use fast generic engine
5/5 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.9510
5/5 [==============================] - 0s 12ms/step
```

Inspect the Accuracy of the models.

We have trained a model for each question. Now, let us check the accuracy of each model and the overall accuracy for all the models combined.

Note: Due to the imbalanced label distribution, we cannot solely rely on the accuracy score to assess model performance.

In [20]:
```python
for name, value in evaluation_dict.items():
  print(f"question {name}: accuracy {value:.4f}")

print("\nAverage accuracy", sum(evaluation_dict.values())/18)
```

```
question 1: accuracy 0.7286
question 2: accuracy 0.9743
question 3: accuracy 0.9351
question 4: accuracy 0.7957
question 5: accuracy 0.6319
question 6: accuracy 0.7885
question 7: accuracy 0.7456
question 8: accuracy 0.6355
question 9: accuracy 0.7632
question 10: accuracy 0.6109
question 11: accuracy 0.6522
question 12: accuracy 0.8695
question 13: accuracy 0.7218
question 14: accuracy 0.7337
question 15: accuracy 0.6166
question 16: accuracy 0.7486
question 17: accuracy 0.7027
question 18: accuracy 0.9510

Average accuracy 0.7558526065614488
```

Visualize the model

One benefit of tree-based models is that we can easily visualize them. The default number of trees used in Random Forests is 300.

Let us pick one model from the models dictionary and select a tree to display below.

In [21]:
```
tfdf.model_plotter.plot_model_in_colab(models['0-4_1'], tree_idx=0, max_depth=3)
```

Out[21]:

Variable importances

Variable importances generally indicate how much a feature contributes to the model predictions or quality. There are several ways to identify important features using TensorFlow Decision Forests. Let us pick one model from the models dictionary and inspect it.

Here is a list of available Variable Importances for Decision Trees:

In [22]:
```
inspector = models['0-4_1'].make_inspector()

print(f"Available variable importances:")
for importance in inspector.variable_importances().keys():
  print("\t", importance)
```

```
Available variable importances:
	 INV_MEAN_MIN_DEPTH
	 NUM_AS_ROOT
	 SUM_SCORE
	 NUM_NODES
```

As an example, let us display the important features for the Variable Importance NUM_AS_ROOT.

The larger the importance score for NUM_AS_ROOT, the more impact it has on the outcome of the model for Question 1 (i.e., model["0-4_1"]).

By default, the list is sorted from the most important to the least. From the output, you can infer that the feature at the top of the list is used as the root node in the most number of trees in the gradient-boosted trees than any other feature.

In [23]:
```
# Each line is: (feature name, (index of the feature), importance score)
inspector.variable_importances()["NUM_AS_ROOT"]
```

Out[23]:
```
[("room_fqid_nunique" (1; #16), 9.0),
 ("level" (1; #7), 5.0),
 ("name_nunique" (1; #9), 5.0),
 ("text_fqid_nunique" (1; #21), 5.0),
 ("hover_duration_std" (1; #6), 4.0),
 ("page" (1; #10), 4.0),
 ("screen_coor_x_std" (1; #18), 4.0),
 ("elapsed_time" (1; #1), 2.0),
 ("page_std" (1; #11), 2.0),
 ("room_coor_x" (1; #12), 2.0),
 ("event_name_nunique" (1; #3), 1.0),
 ("room_coor_y_std" (1; #15), 1.0)]
```

Threshold-Moving for Imbalanced Classification

Since the values of the column correct are fairly imbalanced, using the default threshold of 0.5 to map the predictions into classes 0 or 1 can result in poor performance. In such cases, to improve performance, we will calculate the F1 score for a certain range of thresholds and try to find the best threshold, i.e., the threshold with the highest F1 score. We will then use this threshold to map the predicted probabilities to class labels 0 or 1.

Please note that we are using the F1 score since it is a better metric than accuracy to evaluate problems with class imbalance.

In [24]:
```
# Create a dataframe of required size:
# (no: of users in validation set x no: of questions) initialized to zero values
# to store true values of the label `correct`.
true_df = pd.DataFrame(data=np.zeros((len(VALID_USER_LIST),18)), index=VALID_USER_LIST)
```

```
for i in range(18):
    # Get the true labels.
    tmp = labels.loc[labels.q == i+1].set_index('session').loc[VALID_USER_LIST]
    true_df[i] = tmp.correct.values

max_score = 0; best_threshold = 0

# Loop through threshold values from 0.4 to 0.8 and select the threshold with
# the highest `F1 score`.
for threshold in np.arange(0.4,0.8,0.01):
    metric = tfa.metrics.F1Score(num_classes=2,average="macro",threshold=threshold)
    y_true = tf.one_hot(true_df.values.reshape((-1)), depth=2)
    y_pred = tf.one_hot((prediction_df.values.reshape((-1))>threshold).astype('int'), depth=2)
    metric.update_state(y_true, y_pred)
    f1_score = metric.result().numpy()
    if f1_score > max_score:
        max_score = f1_score
        best_threshold = threshold

print("Best threshold ", best_threshold, "\tF1 score ", max_score)
```

```
Best threshold  0.6300000000000002      F1 score  0.6738222
```

Result

use the `best_threshold` calculate in the previous cell

In [25]:

```
import jo_wilder
env = jo_wilder.make_env()
iter_test = env.iter_test()

limits = {'0-4':(1,4), '5-12':(4,14), '13-22':(14,19)}

for (test, sample_submission) in iter_test:
    test_df = feature_engineer(test)
    grp = test_df.level_group.values[0]
    a,b = limits[grp]
    for t in range(a,b):
        gbtm = models[f'{grp}_{t}']
        test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_df.loc[:, test_df.columns != 'level_group'])
        predictions = gbtm.predict(test_ds)
        mask = sample_submission.session_id.str.contains(f'q{t}')
        n_predictions = (predictions > best_threshold).astype(int)
        sample_submission.loc[mask,'correct'] = n_predictions.flatten()

    env.predict(sample_submission)
```

```
This version of the API is not optimized and should not be used to estimate the runtime of your code on the hidden test
set.
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
```

In [26]:
```
! head result.csv
```

```
session_id,correct
20090109393214576_q1,1
20090109393214576_q2,1
20090109393214576_q3,1
20090109393214576_q4,1
20090109393214576_q5,0
20090109393214576_q6,1
20090109393214576_q7,1
20090109393214576_q8,0
20090109393214576_q9,1
```