Analysis of ISIC Melanoma Classification Metadata and Images

Table of Contents

- EDA of the metadata,
- Extracting basic image attributes like image size, colors etc.
- Creating new features from existing data

We will try answer questions like:

- How's the data looking?
- Do we have complete dataset?
- How's the target distribution looking? Is it balanced?
- What are the effects of scan site on outcome?
- Does age effects skin lesion type?
- Is there difference between female and male patients in terms of target?
- How many unique patient data we have and how many scans they had? Is it important?
- Is image quality, colors, size have meaningful impact on the outcome?

Import

In [1]:
```
!pip install -q efficientnet
!pip install -q plotly
!pip install -q pandas_summary
```

In [2]:
```
# loading packages

import pandas as pd
import numpy as np

#

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

#

import seaborn as sns
import plotly.express as px

#

import os
import random
import re
import math
import time

from tqdm import tqdm
from tqdm.keras import TqdmCallback


from pandas_summary import DataFrameSummary

import warnings


warnings.filterwarnings('ignore') # Disabling warnings for clearer outputs



seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
```

In [3]:
```
# Setting color palette.
orange_black = [
    '#fdc029', '#df861d', '#FF6347', '#aa3d01', '#a30e15', '#800000', '#171820'
]

# Setting plot styling.
plt.style.use('ggplot')
```

In [4]:
```
# Setting file paths for our notebook:

base_path = ''
train_img_path = 'jpeg/train/'
test_img_path = 'jpeg/test/'
img_stats_path = 'melanoma2020imgtabular'
```

Loading Data

Train data has 8 features, 33126 observations and Test data 5 features, 10982 observations.

Train Dataset Consists Of:

1. image name -> the filename of specific image for the train set
2. patient_id -> identifies the unique patient
3. sex -> gender of the patient
4. age_approx -> approx age of the patient at time of scanning
5. anatom_site_general_challenge -> location of the scan site
6. diagnosis -> information about the diagnosis
7. benign_malignant - indicates scan result if it's malignant or benign
8. target -> same as above but better for modelling since it's binary

Test Dataset Consists Of:

1. image name -> the filename of specific image for the train set
2. patient_id -> identifies the unique patient
3. sex -> gender of the patient
4. age_approx -> approx age of the patient at time of scanning
5. anatom_site_general_challenge -> location of the scan site

In [5]:
```
# Loading train and test data.
train = pd.read_csv(os.path.join(base_path, 'train.csv'))
test = pd.read_csv(os.path.join(base_path, 'test.csv'))
sample = pd.read_csv(os.path.join(base_path, 'sample_submission.csv'))
```

In [6]:
```
# Checking train and test columns/rows.

print(
    f'Train data has {train.shape[1]} features, {train.shape[0]} observations and Test data {test.shape[1]} features, {test.shape[0]} observatio
)
```

```
Train data has 8 features, 33126 observations and Test data 5 features, 10982 observations.
Train features are:
['image_name', 'patient_id', 'sex', 'age_approx', 'anatom_site_general_challenge', 'diagnosis', 'benign_malignant', 'target']
Test features are:
['image_name', 'patient_id', 'sex', 'age_approx', 'anatom_site_general_challenge']
```

In [7]:
```
# Renaming train/test columns:

train.columns = [
    'img_name', 'id', 'sex', 'age', 'location', 'diagnosis',
    'benign_malignant', 'target'
]
test.columns = ['img_name', 'id', 'sex', 'age', 'location']
```

In [8]:
```
# Taking 5 random samples from the train data:

train.sample(5)
```

Out[8]:

	img_name	id	sex	age	location	diagnosis	benign_malignant	target
8231	ISIC_2566319	IP_9258690	male	35.0	upper extremity	nevus	benign	0
21862	ISIC_6617618	IP_5350484	male	30.0	lower extremity	unknown	benign	0
3058	ISIC_1018102	IP_3723085	female	40.0	torso	unknown	benign	0
7474	ISIC_2334999	IP_7779275	female	45.0	upper extremity	unknown	benign	0
3470	ISIC_1151651	IP_9754730	female	55.0	head/neck	nevus	benign	0

In [9]:
```
# Taking 5 random samples from the test data:

test.sample(5)
```

Out[9]:

	img_name	id	sex	age	location
3444	ISIC_3217925	IP_2245284	female	35.0	torso
354	ISIC_0445097	IP_4987053	female	45.0	lower extremity
4091	ISIC_3807024	IP_1792727	male	50.0	lower extremity
2388	ISIC_2257432	IP_8996365	female	55.0	torso
6924	ISIC_6288943	IP_3329371	male	50.0	head/neck

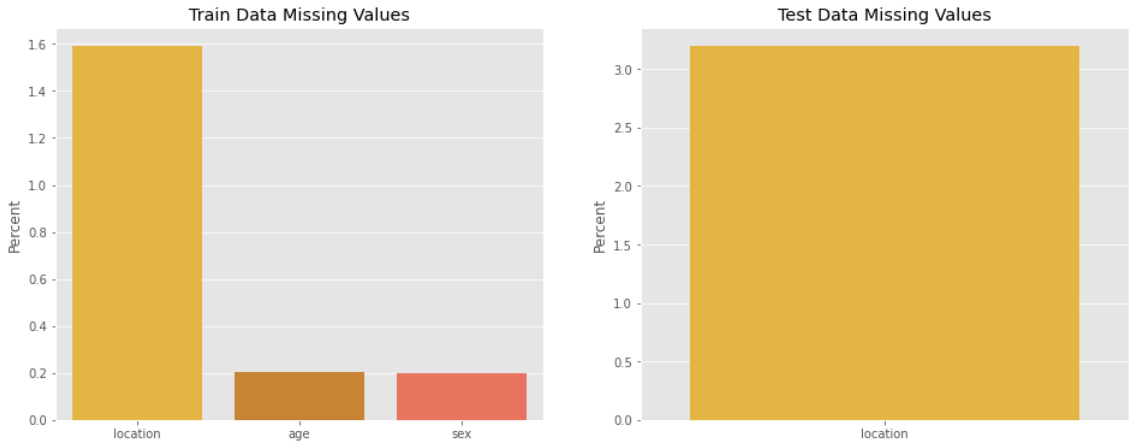
Missing Values

There is a small portion of missing values for age and sex, set 'unknown' for missing values.

In [10]:
```
# Checking missing values:

def missing_percentage(df):

    total = df.isnull().sum().sort_values(
        ascending=False)[df.isnull().sum().sort_values(ascending=False) != 0]
```

```
    percent = (df.isnull().sum().sort_values(ascending=False) / len(df) *
               100)[(df.isnull().sum().sort_values(ascending=False) / len(df) *
                     100) != 0]
    return pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])


missing_train = missing_percentage(train)
missing_test = missing_percentage(test)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))

sns.barplot(x=missing_train.index,
            y='Percent',
            data=missing_train,
            palette=orange_black,
            ax=ax[0])

sns.barplot(x=missing_test.index,
            y='Percent',
            data=missing_test,
            palette=orange_black,
            ax=ax[1])

ax[0].set_title('Train Data Missing Values')
ax[1].set_title('Test Data Missing Values')
```

Out[10]: Text(0.5, 1.0, 'Test Data Missing Values')



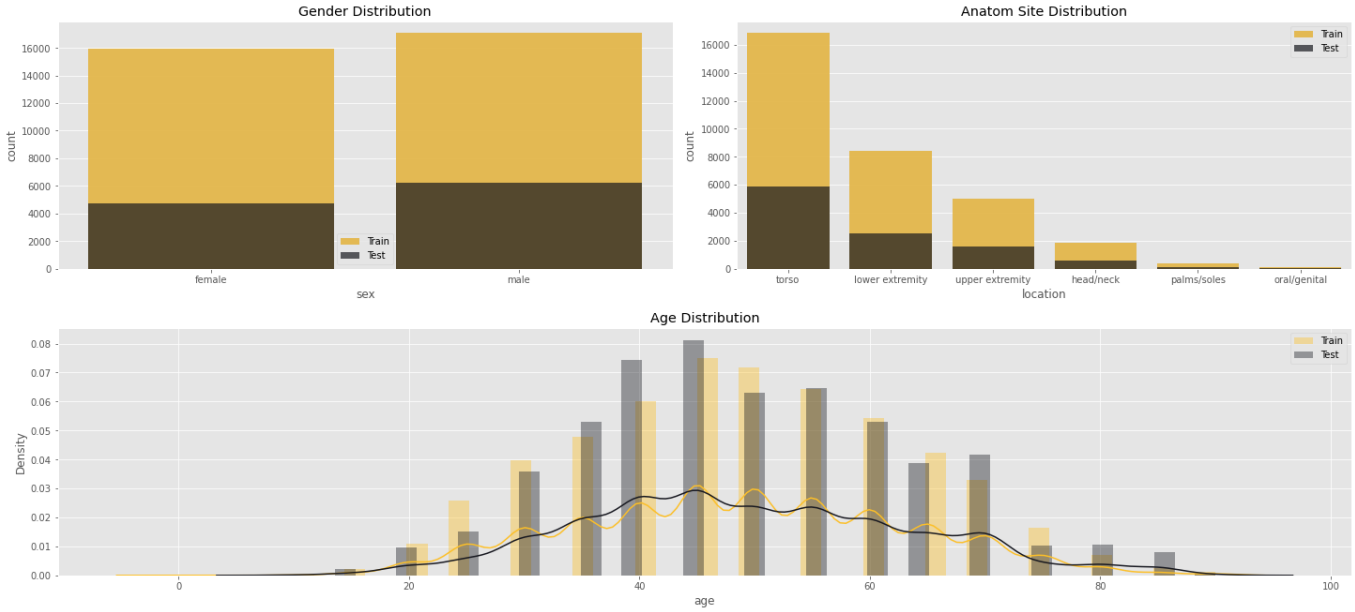
Checking Variables Before Imputing

Check variable distribution before imputing the missing ones.

In [11]:

```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 9))

# Creating a grid:

grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Gender Distribution')

sns.countplot(train.sex.sort_values(ignore_index=True),
              alpha=0.9,
              ax=ax1,
              color='#fdc029',
              label='Train')
sns.countplot(test.sex.sort_values(ignore_index=True),
              alpha=0.7,
              ax=ax1,
              color='#171820',
              label='Test')
ax1.legend()

# Customizing the second grid.

ax2 = fig.add_subplot(grid[0, 2:])

# Plot the countplot.

sns.countplot(train.location,
              alpha=0.9,
              ax=ax2,
              color='#fdc029',
              label='Train',
              order=train['location'].value_counts().index)
sns.countplot(test.location,
              alpha=0.7,
              ax=ax2,
              color='#171820',
```

```
            label='Test',
            order=test['location'].value_counts().index), ax2.set_title(
                'Anatom Site Distribution')

ax2.legend()

# Customizing the third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Age Distribution')

# Plot the histogram.

sns.distplot(train.age, ax=ax3, label='Train', color='#fdc029')
sns.distplot(test.age, ax=ax3, label='Test', color='#171820')

ax3.legend()

plt.show()
```



Imputing Missing Data

In [12]:
```
# Filling missing anatom site values with 'unknown' tag:

for df in [train, test]:
    df['location'].fillna('unknown', inplace=True)
```

In [13]:
```
# Double checking:

ids_train = train.location.values
ids_test = test.location.values
ids_train_set = set(ids_train)
ids_test_set = set(ids_test)

location_not_overlap = list(ids_train_set.symmetric_difference(ids_test_set))
n_overlap = len(location_not_overlap)
if n_overlap == 0:
    print(
        f'There are no different body parts occuring between train and test set...'
    )
else:
    print('There are some not overlapping values between train and test set!')
```

```
There are no different body parts occuring between train and test set...
```

In [14]:
```
# Filling age and sex with appropriate values.

train['sex'].fillna(train['sex'].mode()[0], inplace=True)

train['age'].fillna(train['age'].median(), inplace=True)
```

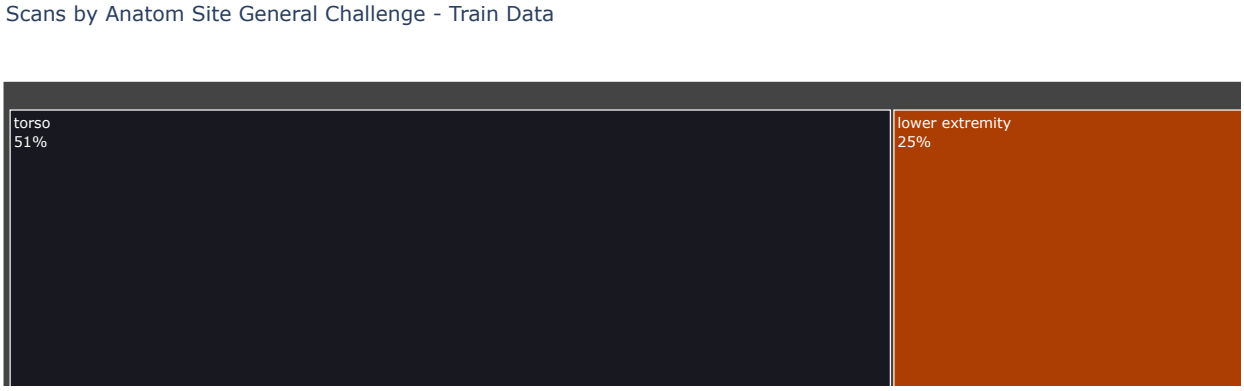
In [15]:
```
# Checking missing value counts:

print(
    f'Train missing value count: {train.isnull().sum().sum()}\nTest missing value count: {train.isnull().sum().sum()}'
)
```

```
Train missing value count: 0
Test missing value count: 0
```

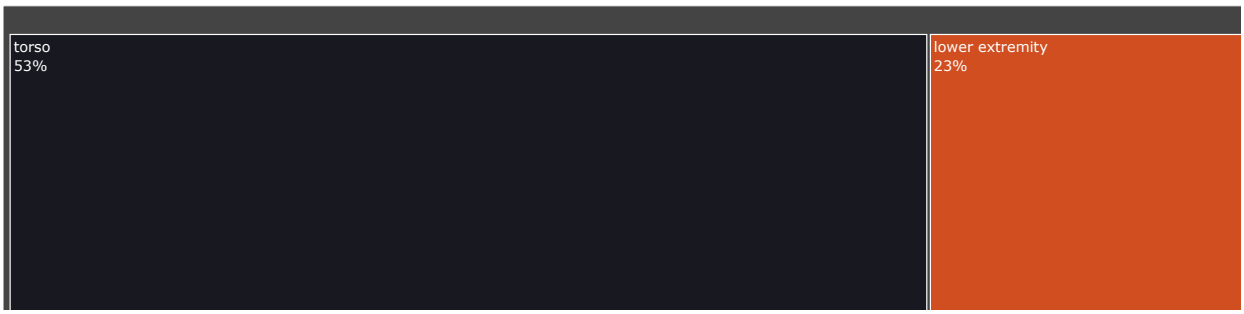
Exploring the Data

Scans by Anatomy Site

In [16]:
```
# Train data:

cntstr = train.location.value_counts().rename_axis('location').reset_index(
    name='count')

fig = px.treemap(cntstr,
                 path=['location'],
                 values='count',
                 color='count',
                 color_continuous_scale=orange_black,
                 title='Scans by Anatom Site General Challenge - Train Data')

fig.update_traces(textinfo='label+percent entry')
fig.show()
```



In [17]:
```
# Test data:

cntste = test.location.value_counts().rename_axis('location').reset_index(
    name='count')

fig = px.treemap(cntste,
                 path=['location'],
                 values='count',
                 color='count',
                 color_continuous_scale=orange_black,
                 title='Scans by Anatom Site General Challenge - Test Data')

fig.update_traces(textinfo='label+percent entry')
fig.show()
```



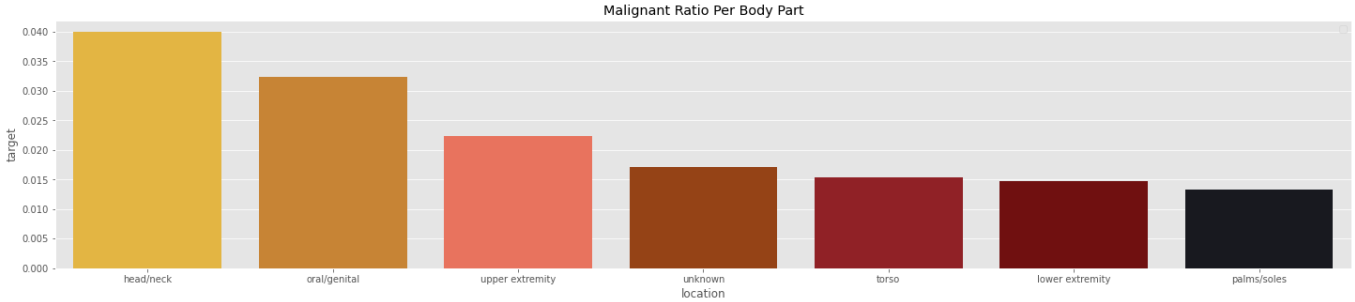
Body Part Ratio by Gender and Target

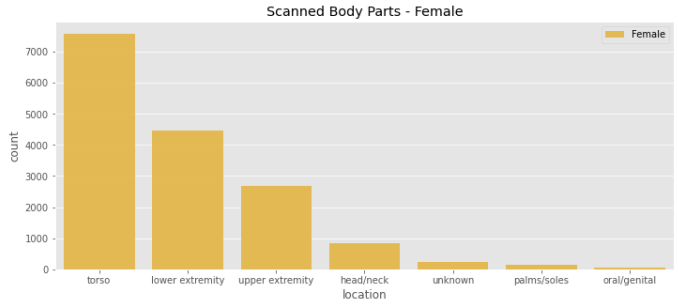
Some body parts are more likely to be malignant, head/neck comes first with followed by oral/genital and upper extremity. Scanned body part locations are similar in order between males and females with small differences on distribution.

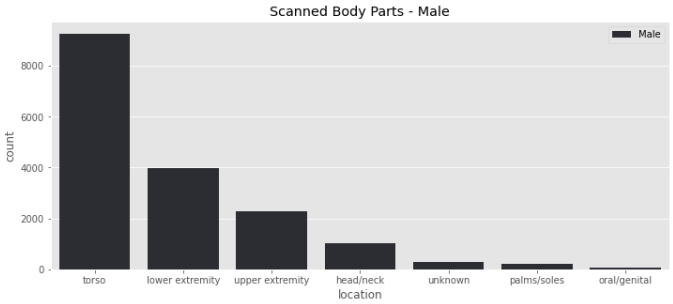
In [18]:

```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 9))
# Creating a grid
grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[1, :2])
# Set the title.
ax1.set_title('Scanned Body Parts - Female')

# Plot:

sns.countplot(
    train[train['sex'] == 'female'].location.sort_values(ignore_index=True),
    alpha=0.9,
    ax=ax1,
    color='#fdc029',
    label='Female',
    order=train['location'].value_counts().index)
ax1.legend()

# Customizing the second grid.

ax2 = fig.add_subplot(grid[1, 2:])

# Set the title.

ax2.set_title('Scanned Body Parts - Male')

# Plot.

sns.countplot(
    train[train['sex'] == 'male'].location.sort_values(ignore_index=True),
    alpha=0.9,
    ax=ax2,
    color='#171820',
    label='Male',
    order=train['location'].value_counts().index)

ax2.legend()

# Customizing the third grid.

ax3 = fig.add_subplot(grid[0, :])

# Set the title.

ax3.set_title('Malignant Ratio Per Body Part')

# Plot.

loc_freq = train.groupby('location')['target'].mean().sort_values(
    ascending=False)
sns.barplot(x=loc_freq.index, y=loc_freq, palette=orange_black, ax=ax3)

ax3.legend()

plt.show()
```

```
No handles with labels found to put in legend.
```



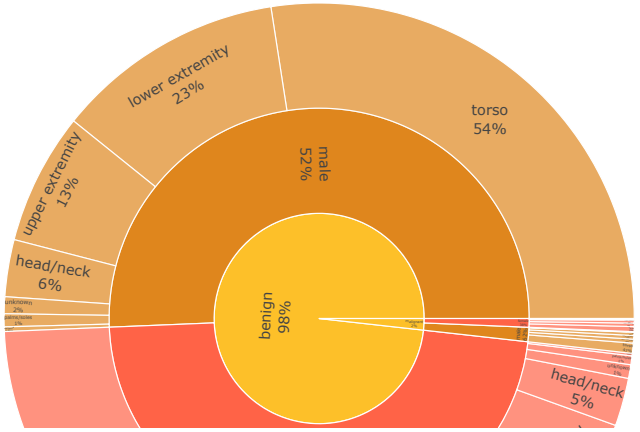




A General Look With Sunburst Chart

- Only 2% of our targets are malignant
- On malignant images males are dominant with 62%
- Gender wise benign images are more balance 52-48% male female ratio
- Malignant image scan locations differs based on the patients gender:
  - Meanwhile the torso is most common location in males it's almost half of the scans meanwhile in females it's 39%
  - Lower extremity is more common with female scans than males 18% males vs 26% females
  - Again upper extremity malignant scans is common with females than males (23- 17%)
- Benign image scan locations more similar between male and female patients.

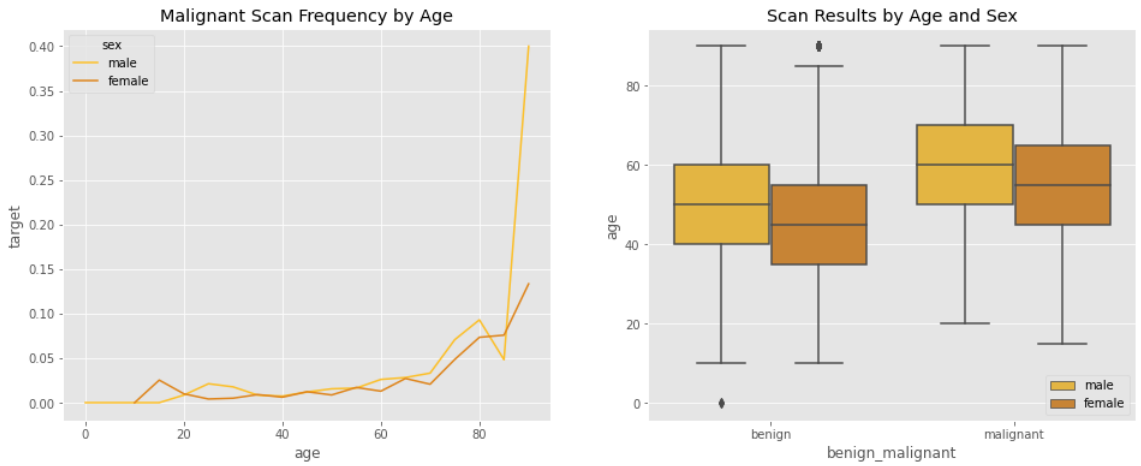
In [19]:
```
# Plotting interactive sunburst:

fig = px.sunburst(data_frame=train,
                  path=['benign_malignant', 'sex', 'location'],
                  color='sex',
                  color_discrete_sequence=orange_black,
                  maxdepth=-1,
                  title='Sunburst Chart Benign/Malignant > Sex > Location')

fig.update_traces(textinfo='label+percent parent')
fig.update_layout(margin=dict(t=0, l=0, r=0, b=0))
fig.show()
```



Age and Scan Result Relations

Age looks pretty decent factor on scan result. Getting malignant scan result with elderly age is more possible than young patients. There is spike for both genders after age of 85, if we look distribution of ages there isn't much of 80+ patients and it can be the reason of this spike but we can safely say it's more likely to be malignant scan after age of 60.

In [20]:
```
# Plotting age vs sex vs target:

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
sns.lineplot(x='age',
             y='target',
             data=train,
             ax=ax[0],
             hue='sex',
             palette=orange_black[:2],
             ci=None)
sns.boxplot(x='benign_malignant',
            y='age',
            data=train,
            ax=ax[1],
            hue='sex',
            palette=orange_black)

plt.legend(loc='lower right')

ax[0].set_title('Malignant Scan Frequency by Age')
ax[1].set_title('Scan Results by Age and Sex')

plt.show()
```



Age #2

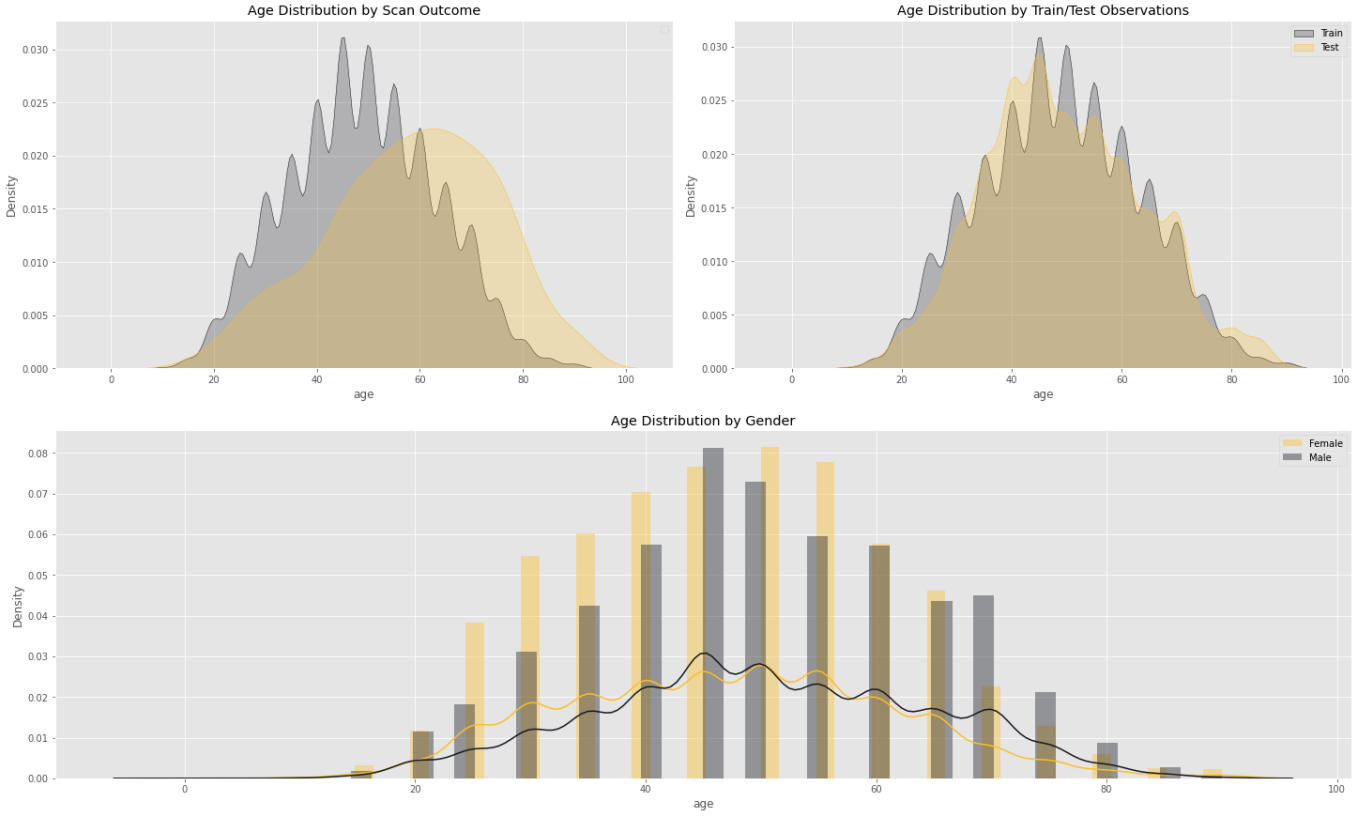
Age seems evenly distributed on both train and test datasets, we can see small bumps at age 75+ and around 40.

Older people are more likely to get malignant scan results. There are more female patients in younger ages this trend changes with the older patients.

In [21]:
```
# Creating a customized chart and giving in figsize etc.

# Plotting age dist vs target and age dist vs datasets

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Age Distribution by Scan Outcome')

# Plot

ax1.legend()

sns.kdeplot(train[train['target'] == 0]['age'],
            shade=True,
            ax=ax1,
            color='#171820',
            label='Benign')
sns.kdeplot(train[train['target'] == 1]['age'],
            shade=True,
            ax=ax1,
            color='#fdc029',
            label='Malignant')

# Customizing second grid.

ax2 = fig.add_subplot(grid[0, 2:])
```

```
# Set the title.

ax2.set_title('Age Distribution by Train/Test Observations')

# Plot.

sns.kdeplot(train.age, label='Train', shade=True, ax=ax2, color='#171820')
sns.kdeplot(test.age, label='Test', shade=True, ax=ax2, color='#fdc029')

ax2.legend()

# Customizing third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Age Distribution by Gender')

# Plot

sns.distplot(train[train.sex == 'female'].age,
             ax=ax3,
             label='Female',
             color='#fdc029')
sns.distplot(train[train.sex == 'male'].age,
             ax=ax3,
             label='Male',
             color='#171820')
ax3.legend()

plt.show()
```

```
No handles with labels found to put in legend.
```



Unique Patients and Their Scan Images

It looks like we have multiple scan images per patient, actual unique patient counts are much lower than images on both datasets. We can get more information about patients age like when he had his first scan and his last scan. We can get interesting insights like:

- Most of the malignant results are found around first 20 scans. Of course there can be control scans after the diagnosis...
- Scan numbers are similar in first 100 scans but we have 200+ scan images for **one particular patient** in dataset, it's pretty interesting since we don't have this case in our training data. We should be careful about this and it can effect our model.
- Most of the malignant cases are under 20 images but in general we can say it's more likely to be malignant result if there are more scan images.

In [22]:

```
print(
    f'Number of unique Patient ID\'s in train set: {train.id.nunique()}, Total: {train.id.count()}\nNumber of unique Patient ID\'s in test set:
)
```

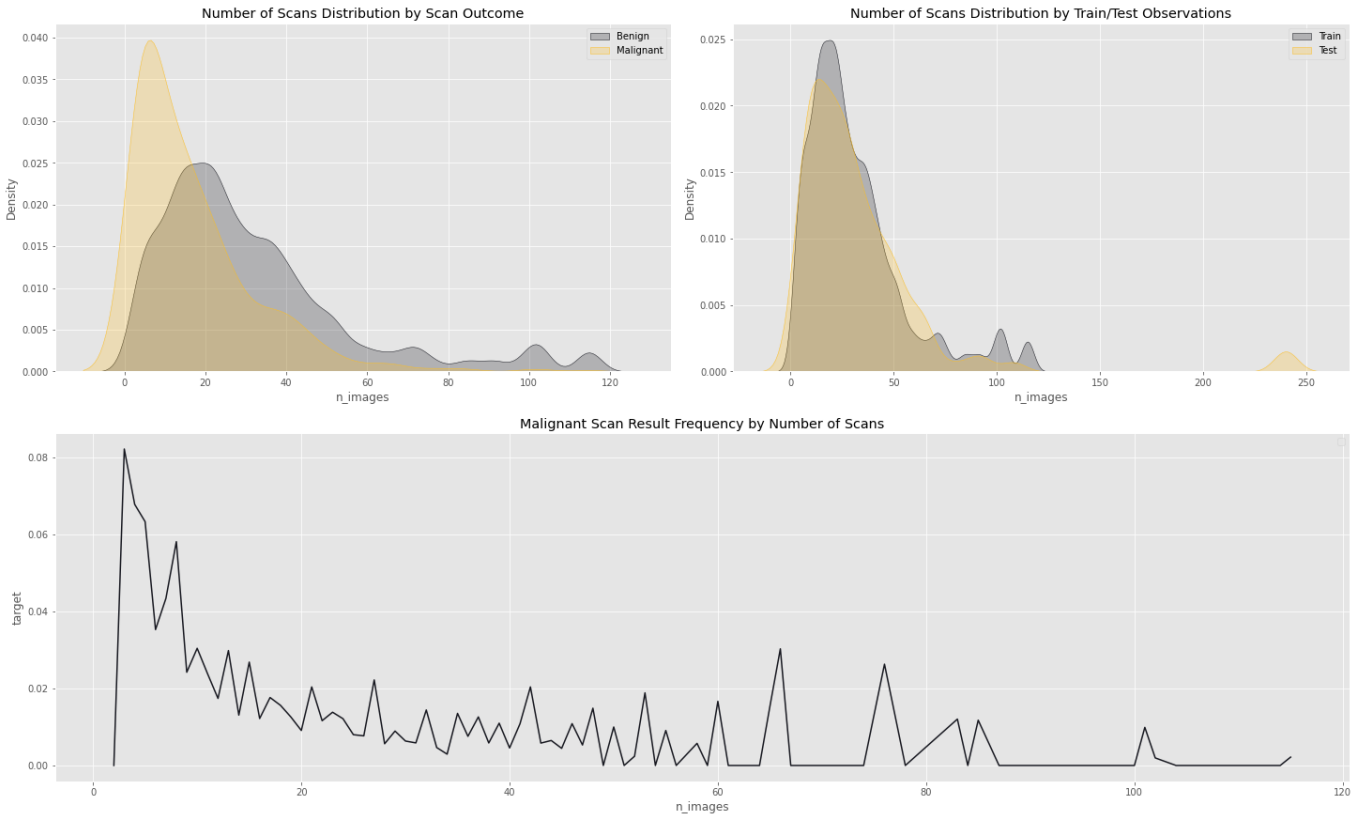
```
Number of unique Patient ID's in train set: 2056, Total: 33126
Number of unique Patient ID's in test set: 690, Total: 10982
```

In [23]:
```
train['age_min'] = train['id'].map(train.groupby(['id']).age.min())
train['age_max'] = train['id'].map(train.groupby(['id']).age.max())

test['age_min'] = test['id'].map(test.groupby(['id']).age.min())
test['age_max'] = test['id'].map(test.groupby(['id']).age.max())
```

In [24]:
```
train['n_images'] = train.id.map(train.groupby(['id']).img_name.count())
test['n_images'] = test.id.map(test.groupby(['id']).img_name.count())
```

In [25]:
```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Number of Scans Distribution by Scan Outcome')

# Plot

sns.kdeplot(train[train['target'] == 0]['n_images'],
            shade=True,
            ax=ax1,
            color='#171820',
            label='Benign')
sns.kdeplot(train[train['target'] == 1]['n_images'],
            shade=True,
            ax=ax1,
            color='#fdc029',
            label='Malignant')

ax1.legend()

# Customizing the second grid.

ax2 = fig.add_subplot(grid[0, 2:])

# Set the title.

ax2.set_title('Number of Scans Distribution by Train/Test Observations')

# Plot

sns.kdeplot(train.n_images, label='Train', shade=True, ax=ax2, color='#171820')
sns.kdeplot(test.n_images, label='Test', shade=True, ax=ax2, color='#fdc029')
ax2.legend()

# Customizing the third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Malignant Scan Result Frequency by Number of Scans')

# Plot

z = train.groupby('n_images')['target'].mean()
sns.lineplot(x=z.index, y=z, color='#171820', ax=ax3)
ax3.legend()

plt.show()
```

```
No handles with labels found to put in legend.
```



Loading Image Meta Features

This is the part where we get basic info directly from images themselves.

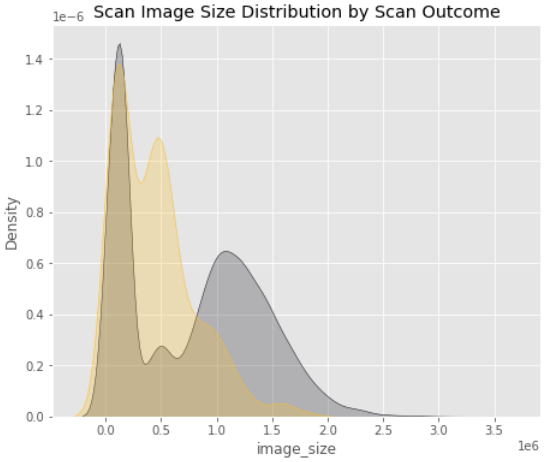
In [27]:
```
# Getting image sizes by using os:

for data, location in zip([train, test], [train_img_path, test_img_path]):
    images = data['img_name'].values
    sizes = np.zeros(images.shape[0])
    for i, path in enumerate(tqdm(images)):
        sizes[i] = os.path.getsize(os.path.join(location, f'{path}.jpg'))

    data['image_size'] = sizes
```

```
100%|██████████| 33126/33126 [00:00<00:00, 43773.35it/s]
100%|██████████| 10982/10982 [00:00<00:00, 52960.54it/s]
```

Image Sizes

We can see some kind of relation between size and target.

In [28]:
```
# Plotting image sizes:

fig, ax = plt.subplots(1, 2, figsize=(16, 6))

sns.kdeplot(train[train['target'] == 0]['image_size'],
            shade=True,
            ax=ax[0],
            color='#171820',
            label='Benign')
sns.kdeplot(train[train['target'] == 1]['image_size'],
            shade=True,
            ax=ax[0],
            color='#fdc029',
            label='Malignant')

sns.kdeplot(train.image_size,
            label='Train',
            shade=True,
            ax=ax[1],
            color='#171820')
sns.kdeplot(test.image_size,
            label='Test',
            shade=True,
            ax=ax[1],
            color='#fdc029')

ax[0].set_title('Scan Image Size Distribution by Scan Outcome')
ax[1].set_title('Scan Image Size Distribution by Train/Test Observations')

plt.show()
```



Getting Image Attributes

You can get these attributes by using the code below, I commented it out here and imported it as a data becasue it's time consuming process.

In [29]:
```
# from keras.preprocessing import image

# for data, location in zip([train, test],[train_img_path, test_img_path]):
#     images = data['img_name'].values
#     reds = np.zeros(images.shape[0])
#     greens = np.zeros(images.shape[0])
#     blues = np.zeros(images.shape[0])
#     mean = np.zeros(images.shape[0])
#     x = np.zeros(images.shape[0], dtype=int)
#     y = np.zeros(images.shape[0], dtype=int)
#     for i, path in enumerate(tqdm(images)):
#         img = np.array(image.load_img(os.path.join(location, f'{path}.jpg')))

#         reds[i] = np.mean(img[:,:,0].ravel())
#         greens[i] = np.mean(img[:,:,1].ravel())
#         blues[i] = np.mean(img[:,:,2].ravel())
#         mean[i] = np.mean(img)
#         x[i] = img.shape[1]
#         y[i] = img.shape[0]

#     data['reds'] = reds
#     data['greens'] = greens
#     data['blues'] = blues
#     data['mean_colors'] = mean
#     data['width'] = x
#     data['height'] = y

# train['total_pixels']= train['width']*train['height']
# test['total_pixels']= test['width']*test['height']
# train['res'] = train['width'].astype(str) + 'x' + train['height'].astype(str)
# test['res'] = test['width'].astype(str) + 'x' + test['height'].astype(str)
```

In [30]:
```
# train.to_csv('train_mean_colorres.csv',
#              columns=['reds','greens','blues','mean_colors','width','height','total_pixels'])
# test.to_csv('test_mean_colorres.csv',
#             columns=['reds','greens','blues','mean_colors','width','height','total_pixels'])
```

In [31]:
```
# Loading color data:
train_attr = pd.read_csv('archive/train_mean_colorres.csv')
test_attr = pd.read_csv('archive/test_mean_colorres.csv')
```

In [32]:
```
train_attr.head()
```

Out[32]:

	reds	greens	blues	mean_colors	width	height	total_pixels
0	212.935898	138.914175	157.742255	169.864109	6000	4000	24000000
1	217.292550	165.093667	130.881285	171.089167	6000	4000	24000000
2	199.941121	130.227057	145.908743	158.692307	1872	1053	1971216
3	119.690968	62.614725	58.724976	80.343556	1872	1053	1971216
4	226.279244	173.174914	152.425131	183.959763	6000	4000	24000000

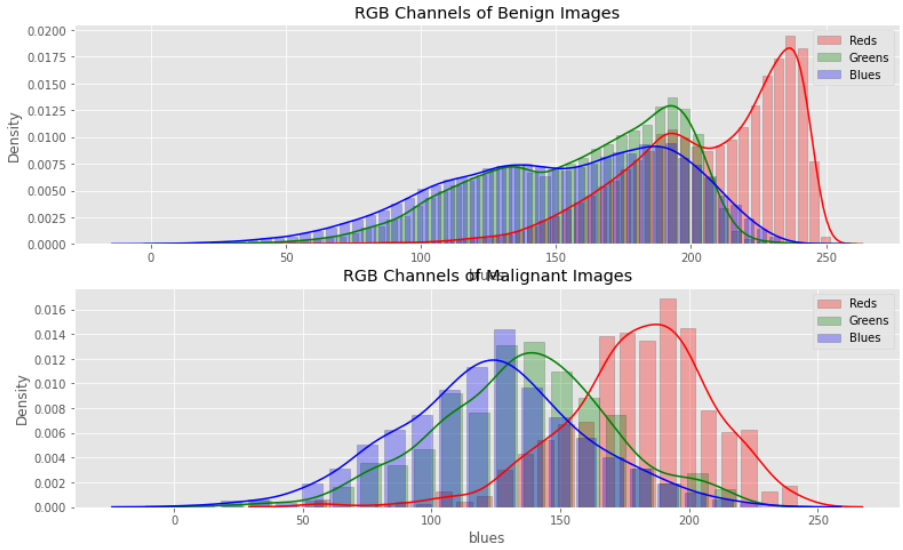
In [33]:
```
train = pd.concat([train, train_attr], axis=1)
test = pd.concat([test, test_attr], axis=1)

train['res'] = train['width'].astype(str) + 'x' + train['height'].astype(str)
test['res'] = test['width'].astype(str) + 'x' + test['height'].astype(str)
```

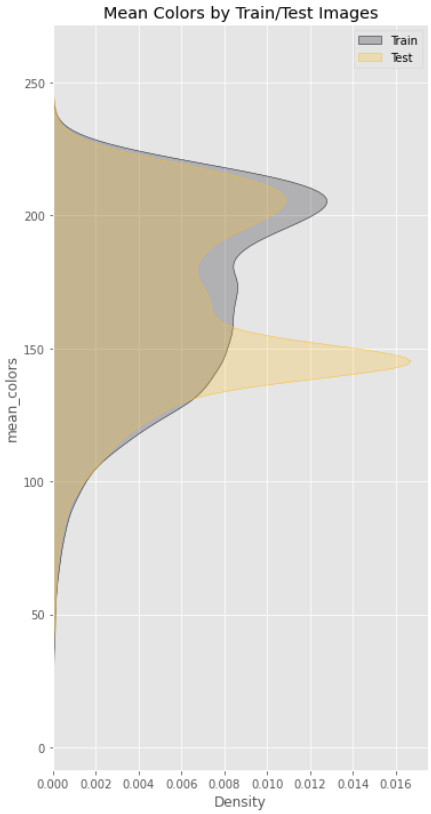
Image Colors and Their Effects on Results

In [34]:

```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('RGB Channels of Benign Images')

# Plot.

sns.distplot(train[train['target'] == 0].reds,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='red',
             kde=True,
             ax=ax1,
             label='Reds')
sns.distplot(train[train['target'] == 0].greens,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='green',
             kde=True,
             ax=ax1,
             label='Greens')
sns.distplot(train[train['target'] == 0].blues,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='blue',
             kde=True,
             ax=ax1,
             label='Blues')

ax1.legend()

# Customizing the second grid.

ax2 = fig.add_subplot(grid[1, :2])

# Set the title.

ax2.set_title('RGB Channels of Malignant Images')

# Plot

sns.distplot(train[train['target'] == 1].reds,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='red',
             kde=True,
             ax=ax2,
             label='Reds')
sns.distplot(train[train['target'] == 1].greens,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='green',
             kde=True,
             ax=ax2,
             label='Greens')
sns.distplot(train[train['target'] == 1].blues,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='blue',
             kde=True,
             ax=ax2,
             label='Blues')
ax2.legend()

# Customizing the third grid.

ax3 = fig.add_subplot(grid[:, 2])

# Set the title.
```

```
ax3.set_title('Mean Colors by Train/Test Images')

# Plot

sns.kdeplot(train.mean_colors,
            shade=True,
            label='Train',
            ax=ax3,
            color='#171820',
            vertical=True)
sns.kdeplot(test.mean_colors,
            shade=True,
            label='Test',
            ax=ax3,
            color='#fdc029',
            vertical=True)
ax3.legend()

plt.show()
```





How are the Image Sizes Affecting Targets in Our Data

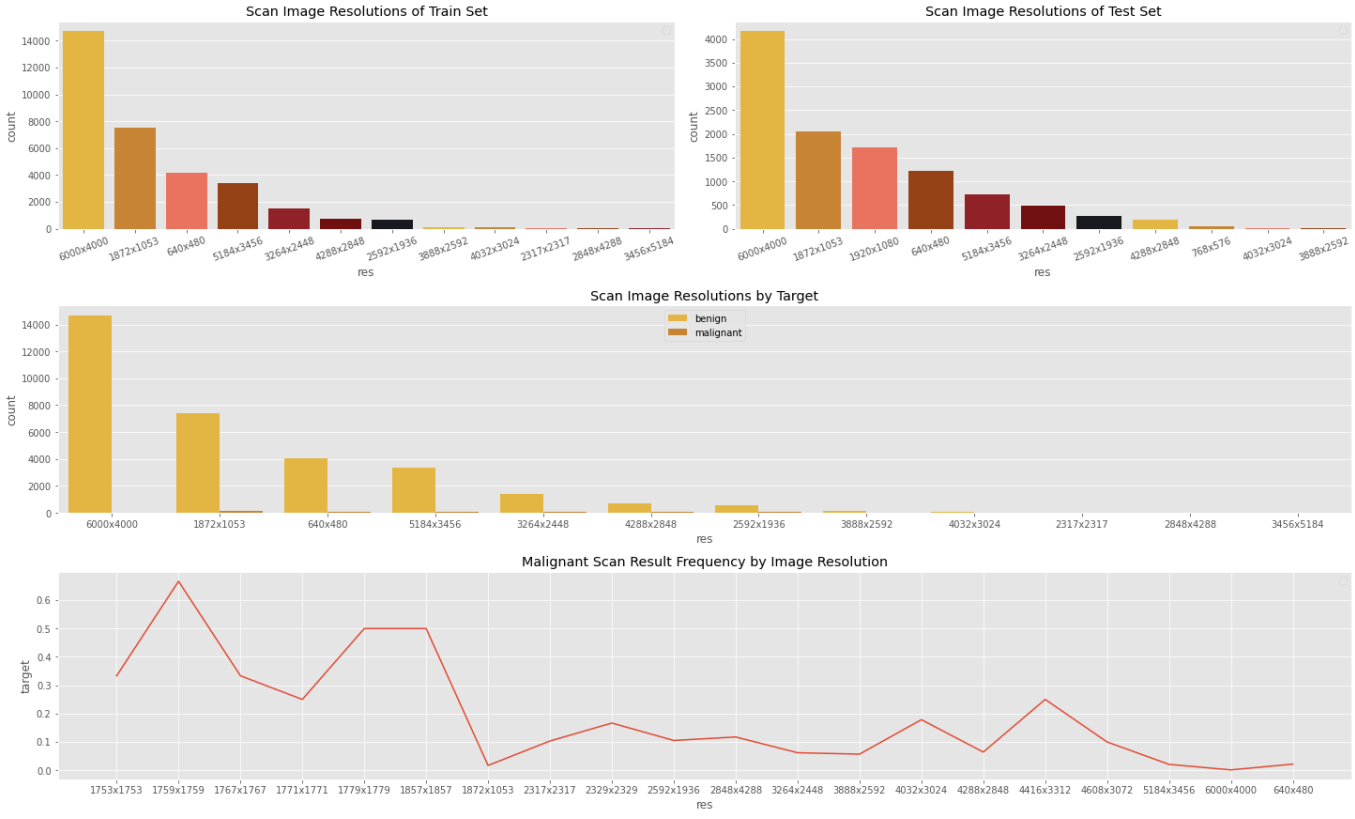
We have whole 1920x1080 set in test data which is not present in train data. That can have huge impact on final scores, which is to be noted in the models. We might want to leave out image size related info in your models or regularize your models to smooth that effect. It can cause overfitting because of high correlation between image sizes and target, but these correlation might not be the case in test set (most likely) so keep that in mind.

In [35]:

```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=4, nrows=3, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Scan Image Resolutions of Train Set')

# Plot.

tres = train.res.value_counts().rename_axis('res').reset_index(name='count')
tres = tres[tres['count'] > 10]
sns.barplot(x='res', y='count', data=tres, palette=orange_black, ax=ax1)
plt.xticks(rotation=20)

ax1.legend()

# Customizing the second grid.
```

```
ax2 = fig.add_subplot(grid[0, 2:])

# Set the title.

ax2.set_title('Scan Image Resolutions of Test Set')

# Plot

teres = test.res.value_counts().rename_axis('res').reset_index(name='count')
teres = teres[teres['count'] > 10]
sns.barplot(x='res', y='count', data=teres, palette=orange_black, ax=ax2)
plt.xticks(rotation=20)
ax2.legend()

# Customizing the third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Scan Image Resolutions by Target')

# Plot.

sns.countplot(x='res',
              hue='benign_malignant',
              data=train,
              order=train.res.value_counts().iloc[:12].index,
              palette=orange_black,
              ax=ax3)
ax3.legend()

# Customizing the last grid.

ax4 = fig.add_subplot(grid[2, :])

# Set the title.

ax4.set_title('Malignant Scan Result Frequency by Image Resolution')

# Plot.

res_freq = train.groupby('res')['target'].mean()
res_freq = res_freq[(res_freq > 0) & (res_freq < 1)]
sns.lineplot(x=res_freq.index, y=res_freq, palette=orange_black, ax=ax4)
ax4.legend()

plt.show()
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```



The Mysterious Images

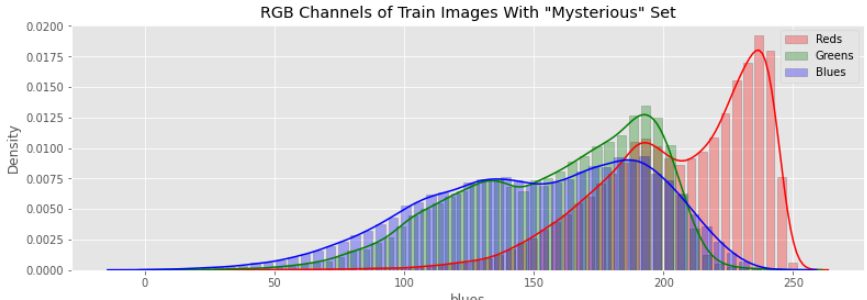
In last part we found out a new set of images with the resolution of 1920x1080 and they aren't present in train data at all. So we can assume these images weren't selected randomly for this competition. Down below I'm going to compare them with the rest of data.

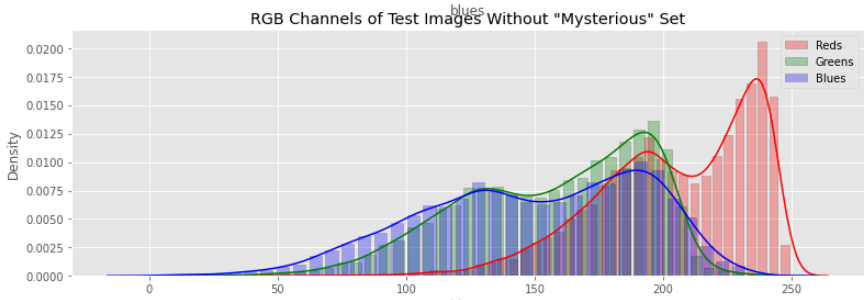
- It looks like without the 1920x1080 set mean colors are much more similar between train and test.
- Again image size distribution gets closer between train and test without the mystery set.

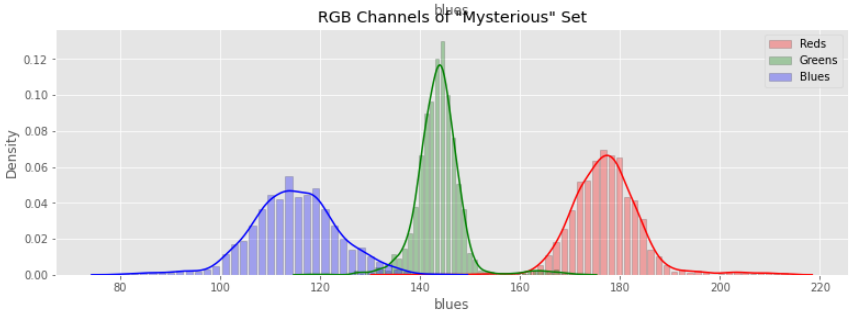
In [36]:

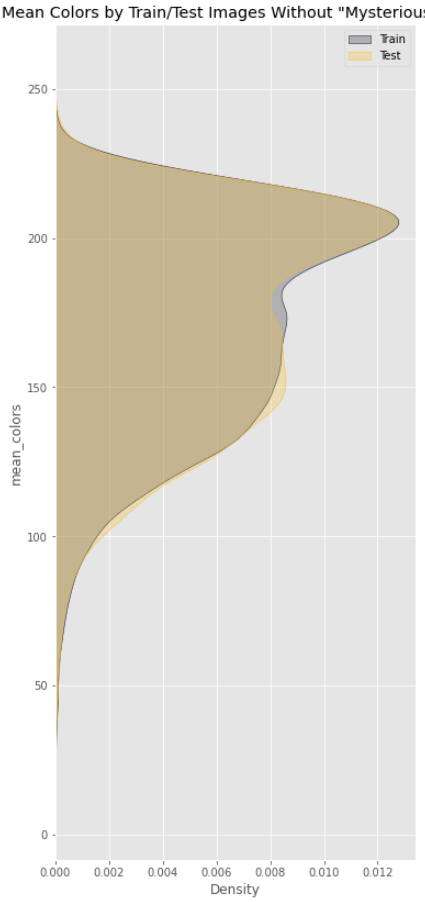
```
# Creating a customized chart and giving in figsize etc.

fig = plt.figure(constrained_layout=True, figsize=(20, 14))

# Creating a grid

grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('RGB Channels of Train Images With "Mysterious" Set')

# Plot.

sns.distplot(train.reds,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='red',
             kde=True,
             ax=ax1,
             label='Reds')
sns.distplot(train.greens,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='green',
             kde=True,
             ax=ax1,
             label='Greens')
sns.distplot(train.blues,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='blue',
             kde=True,
             ax=ax1,
             label='Blues')

ax1.legend()

# Customizing the second grid.

ax2 = fig.add_subplot(grid[1, :2])

# Set the title.

ax2.set_title('RGB Channels of Test Images Without "Mysterious" Set')

# Plot

sns.distplot(test[test['res'] != '1920x1080'].reds,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='red',
             kde=True,
             ax=ax2,
             label='Reds')
sns.distplot(test[test['res'] != '1920x1080'].greens,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='green',
             kde=True,
             ax=ax2,
             label='Greens')
sns.distplot(test[test['res'] != '1920x1080'].blues,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='blue',
             kde=True,
             ax=ax2,
             label='Blues')
ax2.legend()
```

```
# Customizing the third grid.

ax3 = fig.add_subplot(grid[:, 2])

# Set the title.

ax3.set_title('Mean Colors by Train/Test Images Without "Mysterious" Set')

# Plot

sns.kdeplot(train.mean_colors,
            shade=True,
            label='Train',
            ax=ax3,
            color='#171820',
            vertical=True)
sns.kdeplot(test[test['res'] != '1920x1080'].mean_colors,
            shade=True,
            label='Test',
            ax=ax3,
            color='#fdc029',
            vertical=True)
ax3.legend()

# Customizing the last grid.

ax2 = fig.add_subplot(grid[2, :2])

# Set the title.

ax2.set_title('RGB Channels of "Mysterious" Set')

# Plot

sns.distplot(test[test['res'] == '1920x1080'].reds,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='red',
             kde=True,
             ax=ax2,
             label='Reds')
sns.distplot(test[test['res'] == '1920x1080'].greens,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='green',
             kde=True,
             ax=ax2,
             label='Greens')
sns.distplot(test[test['res'] == '1920x1080'].blues,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.3
             },
             color='blue',
             kde=True,
             ax=ax2,
             label='Blues')
ax2.legend()

plt.show()
```







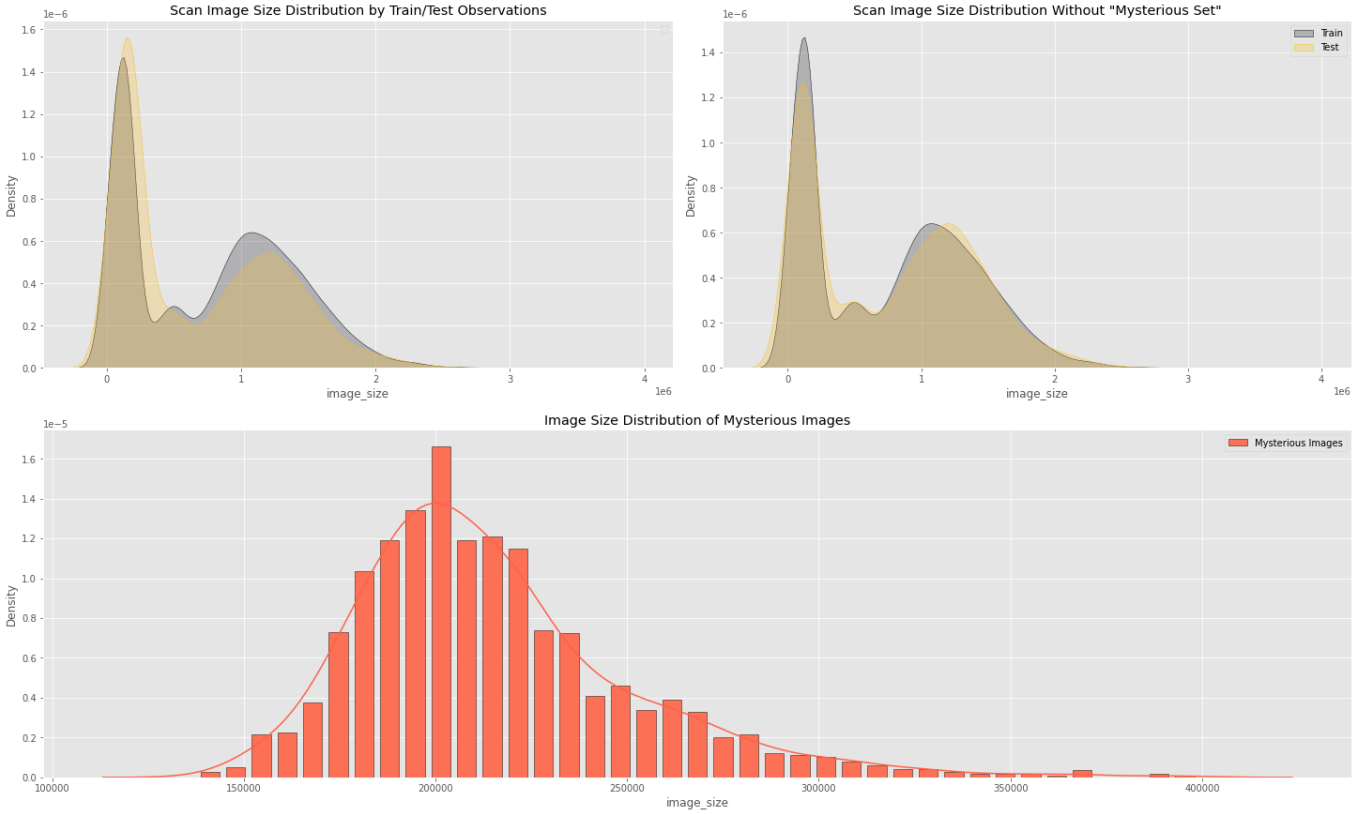


In [37]:

```
# Creating a customized chart and giving in figsize etc.

# Plotting age dist vs target and age dist vs datasets

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Scan Image Size Distribution by Train/Test Observations')

# Plot

ax1.legend()

sns.kdeplot(train['image_size'],
            shade=True,
            ax=ax1,
            color='#171820',
            label='Train')
sns.kdeplot(test['image_size'],
            shade=True,
            ax=ax1,
            color='#fdc029',
            label='Test')

# Customizing second grid.

ax2 = fig.add_subplot(grid[0, 2:])

# Set the title.

ax2.set_title('Scan Image Size Distribution Without "Mysterious Set"')

# Plot.

sns.kdeplot(train.image_size,
            label='Train',
            shade=True,
            ax=ax2,
            color='#171820')
sns.kdeplot(test[test['res'] != '1920x1080'].image_size,
            label='Test',
            shade=True,
            ax=ax2,
```

```
            color='#fdc029')
ax2.legend()

# Customizing third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Image Size Distribution of Mysterious Images')

# Plot

sns.distplot(test[test['res'] == '1920x1080'].image_size,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.9
             },
             color='#FF6347',
             kde=True,
             ax=ax3,
             label='Mysterious Images')
ax3.legend()

plt.show()
```

```
No handles with labels found to put in legend.
```



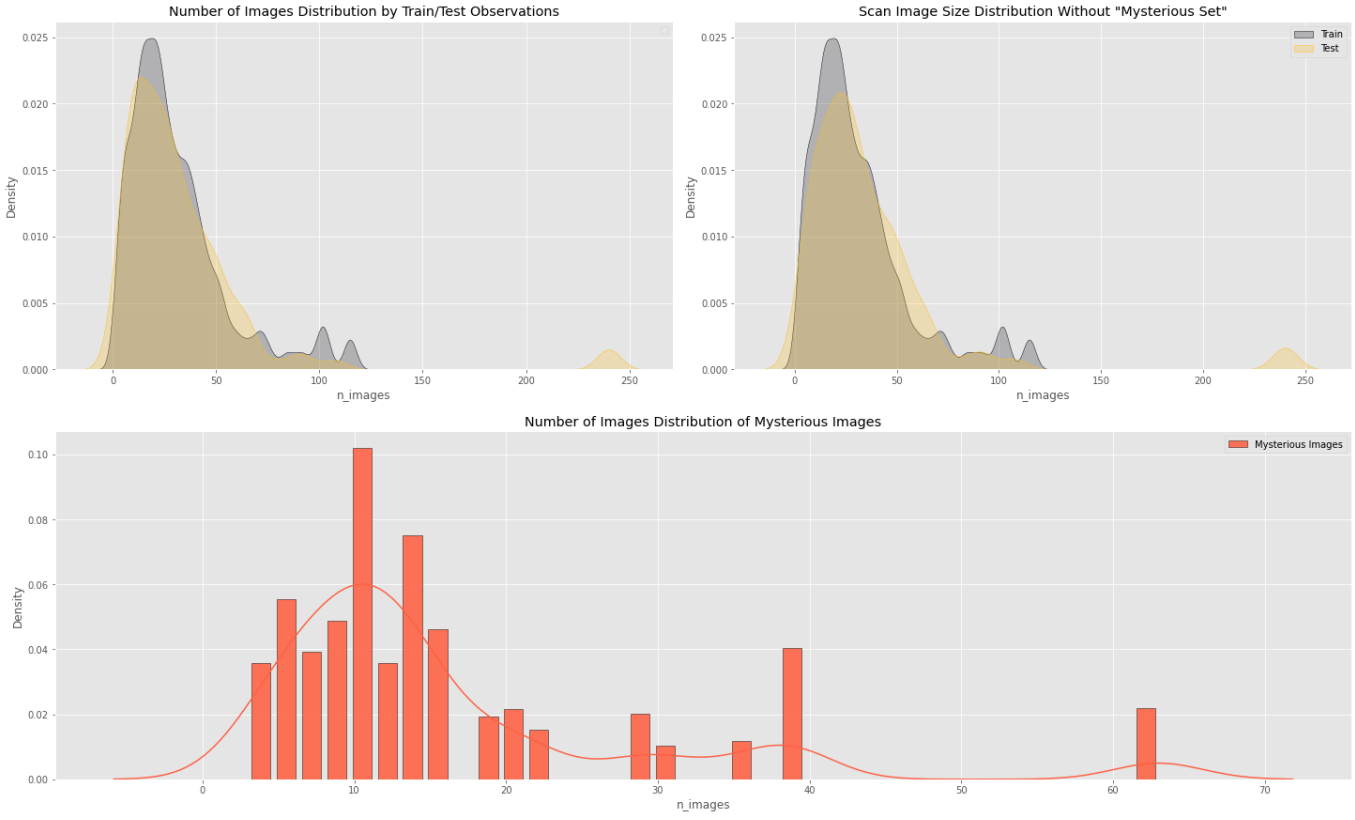
I was curious about if these 1920x1080 images belong to high scan patients including 200+ one but it seems these observations are grouped around 10 scans.

In [38]:

```
# Creating a customized chart and giving in figsize etc.

# Plotting age dist vs target and age dist vs datasets

fig = plt.figure(constrained_layout=True, figsize=(20, 12))

# Creating a grid

grid = gridspec.GridSpec(ncols=4, nrows=2, figure=fig)

# Customizing the first grid.

ax1 = fig.add_subplot(grid[0, :2])

# Set the title.

ax1.set_title('Number of Images Distribution by Train/Test Observations')

# Plot

ax1.legend()

sns.kdeplot(train['n_images'],
            shade=True,
            ax=ax1,
            color='#171820',
```

```
            label='Train')
sns.kdeplot(test['n_images'],
            shade=True,
            ax=ax1,
            color='#fdc029',
            label='Test')

# Customizing second grid.

ax2 = fig.add_subplot(grid[0, 2:])

# Set the title.

ax2.set_title('Scan Image Size Distribution Without "Mysterious Set"')

# Plot.

sns.kdeplot(train.n_images,
            label='Train',
            shade=True,
            ax=ax2,
            color='#171820')
sns.kdeplot(test[test['res'] != '1920x1080'].n_images,
            label='Test',
            shade=True,
            ax=ax2,
            color='#fdc029')
ax2.legend()

# Customizing third grid.

ax3 = fig.add_subplot(grid[1, :])

# Set the title.

ax3.set_title('Number of Images Distribution of Mysterious Images')

# Plot

sns.distplot(test[test['res'] == '1920x1080'].n_images,
             hist_kws={
                 'rwidth': 0.75,
                 'edgecolor': 'black',
                 'alpha': 0.9
             },
             color='#FF6347',
             kde=True,
             ax=ax3,
             label='Mysterious Images')
ax3.legend()

plt.show()
```

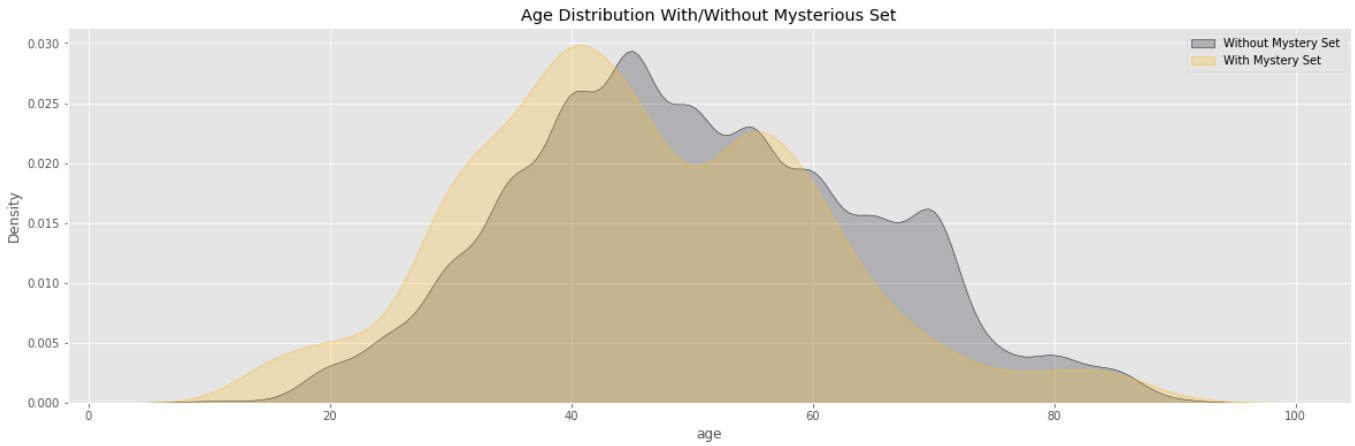
```
No handles with labels found to put in legend.
```



In [39]:

```
fig, ax = plt.subplots(figsize=(20, 6))
```

```
sns.kdeplot(test[test['res'] != '1920x1080'].age,
            shade=True,
            label='Without Mystery Set',
            color='#171820',
            )
sns.kdeplot(test[test['res'] == '1920x1080'].age,
            shade=True,
            label='With Mystery Set',
            color='#fdc029',
            )

plt.legend(loc='upper right')

ax.set_title('Age Distribution With/Without Mysterious Set')


plt.show()
```



Looks like our 1920x1080 set images consisting little bit younger patients than the rest.

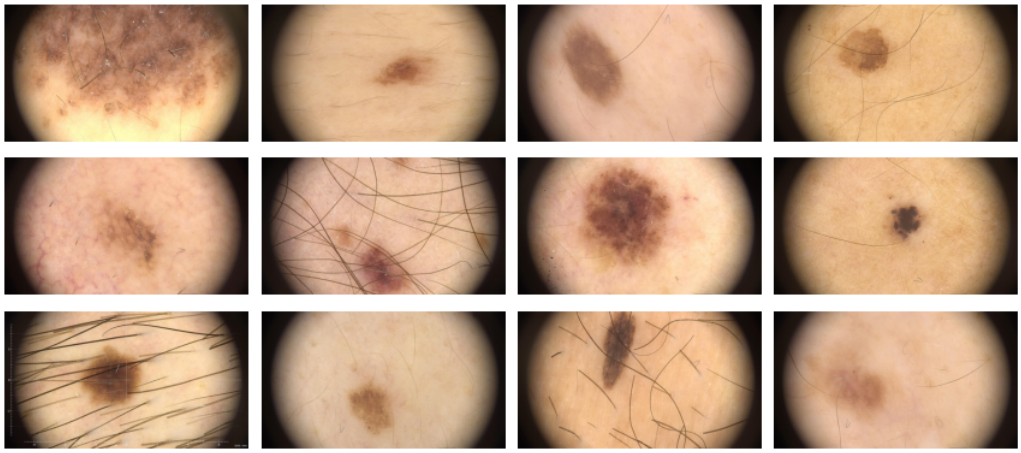
Visual Inspection of Mysterious Image Set

When we look at both samples we can see that 1920x1080 images are coming from a 'imaging device' with black circle around the images? In general this isn't the case with the rest of the test image samples.

In [40]:
```
mystery = test[test['res'] == '1920x1080']
mystimages = mystery['img_name'].values

nonmystery = test[test['res'] != '1920x1080']
nonmystimages = nonmystery['img_name'].values

random_myst_images = [np.random.choice(mystimages+'.jpg') for i in range(12)]
random_nmyst_images = [np.random.choice(nonmystimages+'.jpg') for i in range(12)]

# Location of test images
img_dir = 'jpeg/test'
```

In [41]:
```
plt.figure(figsize=(12,6))
for i in range(12):

    plt.subplot(3, 4, i + 1)
    img = plt.imread(os.path.join(img_dir, random_myst_images[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.suptitle('Sample Images From Mysterious Test Set', fontsize=14)
plt.tight_layout()
```

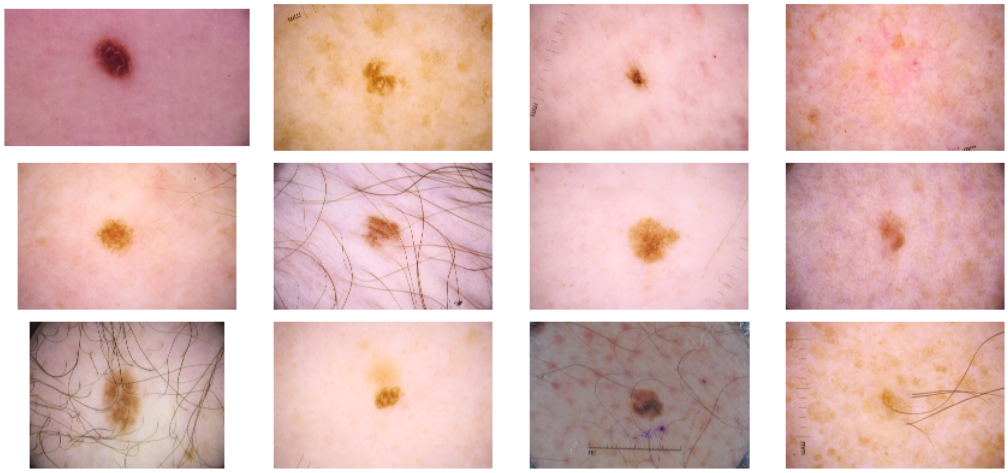
Sample Images From Mysterious Test Set



In [42]:

```
plt.figure(figsize=(12,6))
for i in range(12):

    plt.subplot(3, 4, i + 1)
    img = plt.imread(os.path.join(img_dir, random_nmyst_images[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.suptitle('Sample Images From Rest of the Test Set', fontsize=14, y=1.05)
plt.tight_layout()
```

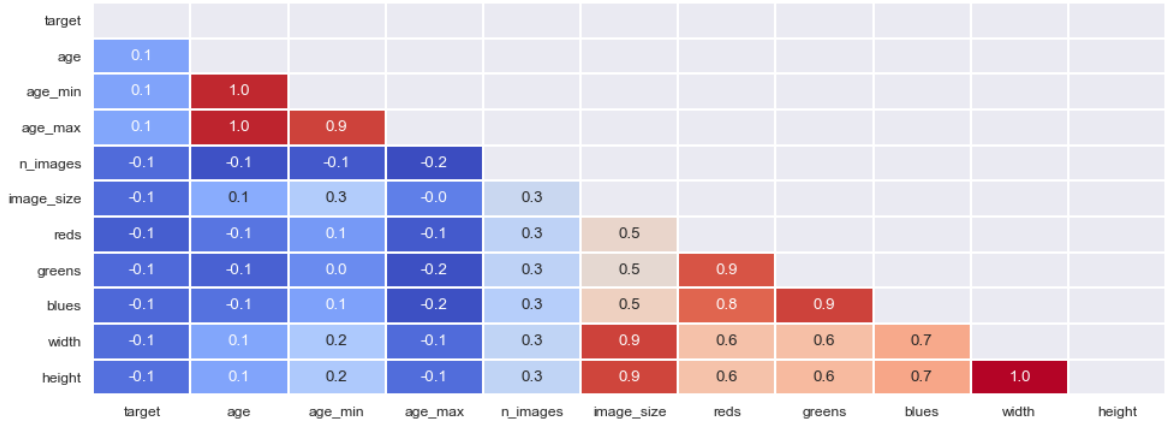
Sample Images From Rest of the Test Set



Correlations Between Features

In [43]:

```
# Display numerical correlations between features on heatmap.

sns.set(font_scale=1.1)
correlation_train = train[['target','age','age_min',
 'age_max',
 'n_images',
 'image_size',
 'reds',
 'greens',
 'blues',
 'width',
 'height',
 ]].corr()
mask = np.triu(correlation_train.corr())
plt.figure(figsize=(16, 6))
sns.heatmap(correlation_train,
            annot=True,
            fmt='.1f',
            cmap='coolwarm',
            mask=mask,
            linewidths=1,
            cbar=False)

plt.show()
```


target
age 0.1
age_min 0.1 1.0
age_max 0.1 1.0 0.9
n_images -0.1 -0.1 -0.1 -0.2
image_size -0.1 0.1 0.3 -0.0 0.3
reds -0.1 -0.1 0.1 -0.1 0.3 0.5
greens -0.1 -0.1 0.0 -0.2 0.3 0.5 0.9
blues -0.1 -0.1 0.1 -0.2 0.3 0.5 0.8 0.9
width -0.1 0.1 0.2 -0.1 0.3 0.9 0.6 0.6 0.7
height -0.1 0.1 0.2 -0.1 0.3 0.9 0.6 0.6 0.7 1.0
target age age_min age_max n_images image_size reds greens blues width height