1. Business Objective

The objective of my project is to help PetFinder.my accurately determine a pet photo's popularity score and to give these animals a higher chance of loving homes. PetFinder.my is Malaysia's leading animal welfare platform, (you can adopt a pet or find a home for your pet on this platform.) And they hope that pets with attractive photos can generate more interest and be adopted faster. But what makes a good picture? With the help of data science, you may be able to accurately determine a pet photo's appeal and even suggest improvements to give these rescue animals a higher chance of loving homes.

2. Data Ingestion

One interesting thing about this project is that this dataset has both image data and meta data. There are 9912 image data, with 1.06GB in size, are stored in a jpg format. In metadata, there are 12 basic features for each photo as well as the photo's Pawpularity score. The Pawpularity score are the values we are going to predict. They are continuous number between 0 and 100. These 12 features are labeled with the binary value 1 or 0, like whether they have a clear face, whether their eyes face front. (And the Id column is corresponding to the photo's file name.)

Use VIF to check if there any multicollinearity in the meta data:

```
#Use VIF to check the multicollinearity
predictor = train.columns[1:13]

print(predictor)

corr_matrix = train[predictor].corr()

from numpy.linalg import inv
inv_corr = inv(corr_matrix)

vif_data = pd.DataFrame()
vif_data["feature"] = predictor
vif_data["VIF"] = inv_corr.diagonal()
vif_data = vif_data.sort_values("VIF", ascending=False)
vif_data
```

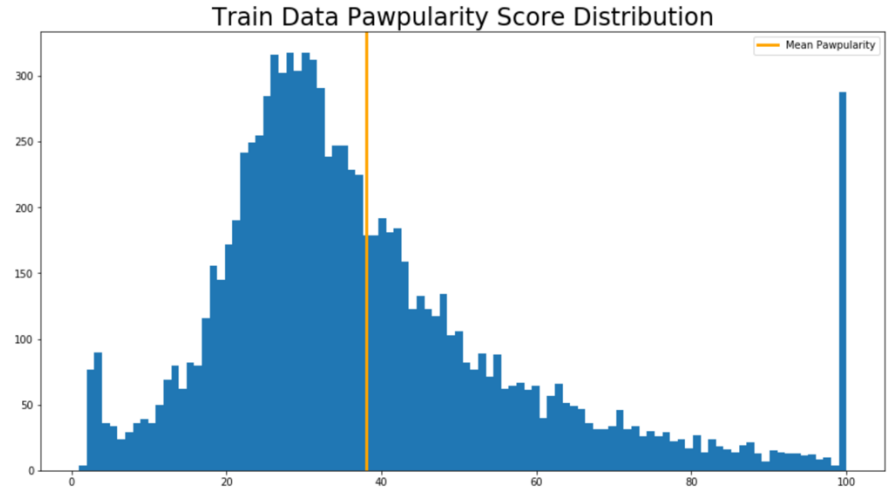
There are 27 pairs of duplicate images but with different popularity scores, which would affect the performance of prediction. So, I remove these images from the dataset.

```
def find_similar_images(threshold=0.90):
    # Number of Duplicate Images Found
    duplicate_counter = 1
    # Indices of Duplicate Images
    duplicate_idxs = set()
    # For each image in the train dataset
    for idx, phash in enumerate(tqdm(train['phash'])):
        # Compute the similarity to all other images
        for idx_other, phash_other in enumerate(train['phash']):
            # Similarity score is imply the percentage of equal bits
            similarity = (phash ==phash_other).mean()
            # Prevent self comparison, threshold similarity and ignore repetetive duplicate detection
            if idx != idx_other and similarity > threshold and not(duplicate_idxs.intersection([idx, idx_other])):
                # Update Duplicate Indices
                duplicate_idxs.update([idx, idx_other])
                # Get DataFrame rows
                row = train.loc[idx]
                row_other = train.loc[idx_other]
                # Plot Duplicate Images
                fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8,5))
                ax[0].imshow(imageio.imread(row['file_path']))
                ax[0].set_title(f'Idx: {idx}, Pawpularity: {row["Pawpularity"]}')
                ax[1].imshow(imageio.imread(row_other['file_path']))
                ax[1].set_title(f'Idx: {idx_other}, Pawpularity: {row_other["Pawpularity"]}')
                plt.suptitle(f'{duplicate_counter} | PHASH Similarity: {similarity:.3f}')
                plt.show()
                # Increase Duplicate Counter
                duplicate_counter += 1

    # Return Indices of Duplicates
    return duplicate_idxs
```

3. Visualizations

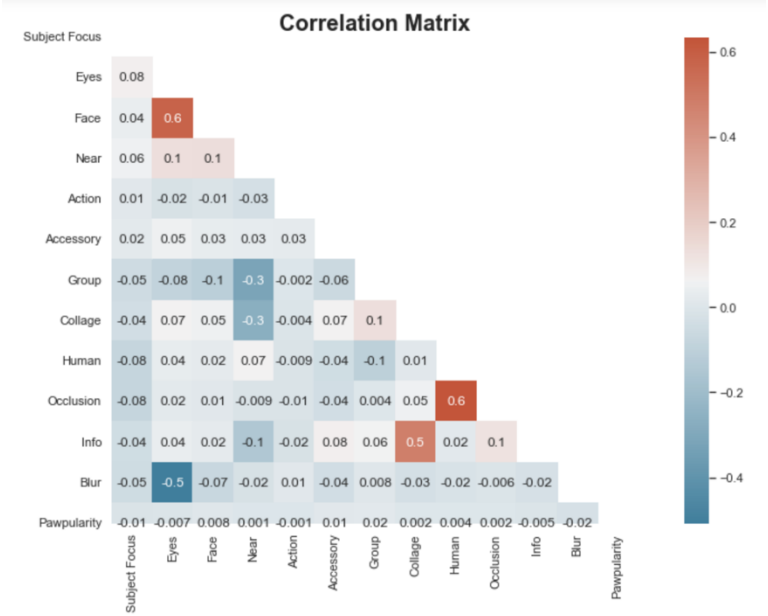
Train Data Pawpularity Score Distribution:

```
# Pawpularity Score Distribution
print('Pawpularity Statistics')
print(train['Pawpularity'].describe())
plt.figure(figsize=(15,8))
plt.title('Train Data Pawpularity Score Distribution', size=24)
plt.hist(train['Pawpularity'], bins=100)
plt.axvline(train['Pawpularity'].mean(), c='orange', ls='-', lw=3, label="Mean Pawpularity")
plt.legend()
#plt.show()
plt.savefig('fig1_1.pdf')
```



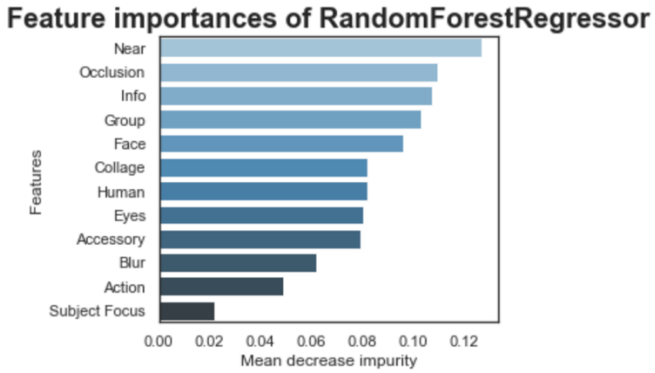
Correlation matrix between the features of meta data:

```
corr_matrix = train[predictor].corr()
fig = plt.figure()
plt.figure(figsize=(15,8))
sns.set_theme(style="white")
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr_matrix, annot=True, fmt='.1g', cmap=cmap,
            mask=mask, square=True)
plt.title('Correlation Matrix', fontsize=20, fontweight='bold')
#plt.show()
plt.savefig('fig1_4.pdf')
```



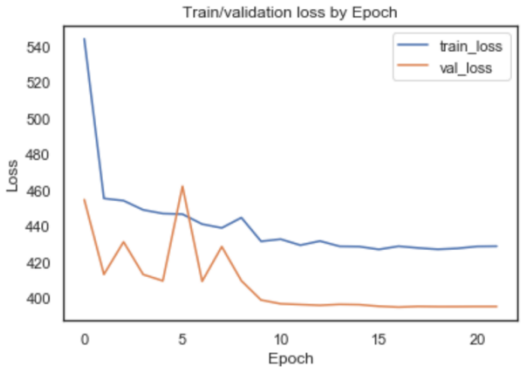
Feature importances of RandomForestRegressor:

```
importances = best_rfr.best_estimator_.feature_importances_

feature_names = X_train.columns
forest_importances = pd.DataFrame(importances, columns=["FI"], index=feature_names)
forest_importances = forest_importances.sort_values("FI", ascending=False)

fig = plt.figure(figsize=(12,8))
fig, ax = plt.subplots()
sns.barplot(data=forest_importances, x = "FI",
            y=forest_importances.index, ax=ax,
            palette="Blues_d")
ax.set_title("Feature importances of RandomForestRegressor",
             fontsize=20, fontweight='bold')
ax.set_xlabel("Mean decrease impurity")
ax.set_ylabel("Features")
fig.tight_layout()
#plt.show()
plt.savefig('fig3_1.pdf')
```



Train/Validation loss by epoch in CNN model:

```
plt.figure()
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
#plt.xticks(range(0,60))
plt.title("Train/validation loss by Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.savefig('fig3_3.pdf')
```



4. Machine Learning

a. Random forest regression on metadata

- Tuned three parameters: bootstrap, max_features, number of trees and max_depth.
- The best values for these four parameters are: True, log2, 400, 10.

- Train RMSE: 20.4.
- Validation RMSE: 20.74.

```
rfr = RandomForestRegressor(random_state=8)
param_grid = {
            "n_estimators" : [500, 600, 700],
            "max_features" : ["log2", "sqrt", 0.33],
            "bootstrap"    : [True, False]
        }

grid_rfr = GridSearchCV(
    rfr,
    param_grid,
    cv = 5,
    verbose=1,
    n_jobs=-1)

best_rfr = grid_rfr.fit(X_train, y_train)
```

b. CNN on image data
   - Basic CNN model
   - Pretrained model (EfficientNetB0.h5)
   - Train RMSE stabilized at 20.68.
   - Validation RMSE stabilized at 19.88.

```
inputs=keras.Input(shape=(image_width,image_height,3))
x=inputs
x=keras.layers.Conv2D(filters=16,kernel_size=3,strides=2,padding='same',activation='relu')(x)
x=keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x=keras.layers.Conv2D(filters=32,kernel_size=3,strides=2,padding='same',activation='relu')(x)
x=keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x=keras.layers.Conv2D(filters=64,kernel_size=3,strides=2,padding='same',activation='relu')(x)
x=keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x=keras.layers.Conv2D(filters=128,kernel_size=3,strides=2,padding='same',activation='relu')(x)
x=keras.layers.Flatten()(x)
x=keras.layers.Dense(128, activation = "relu")(x)
x=keras.layers.Dropout(0.2)(x)
output = tf.keras.layers.Dense(1)(x)  #default activation function is linear.
model = tf.keras.Model(inputs=inputs, outputs=output)
```

5. Deliverable

https://github.com/wr2007apple/thedataincubator/blob/main/Capstone_Project-Submit.ipynb