CS589 - Mini project

Sakshi Doshi

Introduction

In this project, I try to do the Pawpularity Kaggle Challenge listed here. The main aim of this project is to analyse raw images and metadata to predict the popularity of pet photos. I have used several approaches to solve this project. I first start by pre-processing data by using different techniques such as reshaping, augmenting and feature engineering. I then train the data using different approaches such as: Using traditional regression models, using ensemble learning and deep learning. In addition to the models that need to be compulsorily implemented in the project, I have taken a step ahead and trained many more extra models. K-fold cross validation is used to give much more efficient results. The evaluation for these models is done by using the RMSE score value. This value is the square root of the variance of the residuals. It indicates how close are the model's prediction to the observed data points.

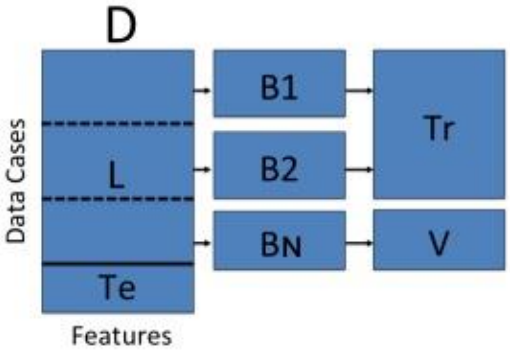
Q1

In order to validate the model, I use the RMSE score. The RMSE score is the square root of the variance of the residual error. It indicates the absolute fit of the model to the data. Basically, it tells us how the close are the observed data points to the model's predicted values. Here R-squared is a relative measure of fit and RMSE is the actual measure of fit. In my code, I use the mean_squared_error function from sklearn.metrics. in sklearn. The RMSE score will be our local test score. The formula for RMSE is

$$[\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}]$$

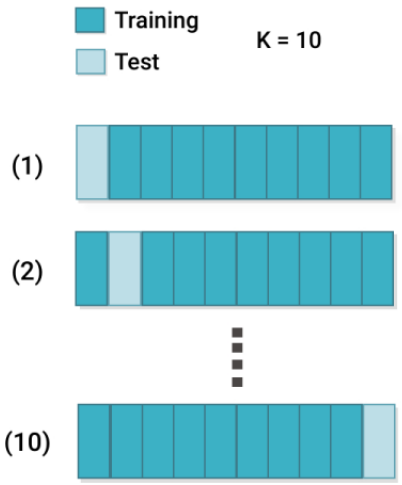
where $(\hat{y}_i)$ is the predicted value and (y_i) is the original value for each instance.

For evaluating the model performance, I use the scheme of K-fold cross validation. I take 10 folds and use them to cross validate and give me a better result.



Algorithm for K- fold cross validation :

1. Pick a number of folds – **k**. Usually, **k** is 5 or 10 but you can choose any number which is less than the dataset's length.
2. Split the dataset into **k** equal (if possible) parts (they are called folds)
3. Choose **k** – 1 folds which will be the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration of **cross-validation**, you must train a new model independently of the model trained on the previous iteration
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 **k** times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.



I have used k or cv = 10 for every model in this project. This means that there will be 10 folds in my training set and every time a new iteration is run, one of the 10 folds will act as validation set and the other nine will stay as the training sets. I have coded the entire K fold cross validation in my models. It was convenient as I just made a function and called it everywhere.

I used the training set to train the hyperparameters. The purpose of the validation set was to help me update the best hyper-paramters. Finally I trained the final model using both the training and the validation data and reported the final RMSE Value.

Q2:
In this competition, we have two types of input. The metadata and the image data. I have used metadata in each and every one of my models. Whereas, I have used image – data from model 20 onwards.

Preprocessing and Data Analysis of Metadata :

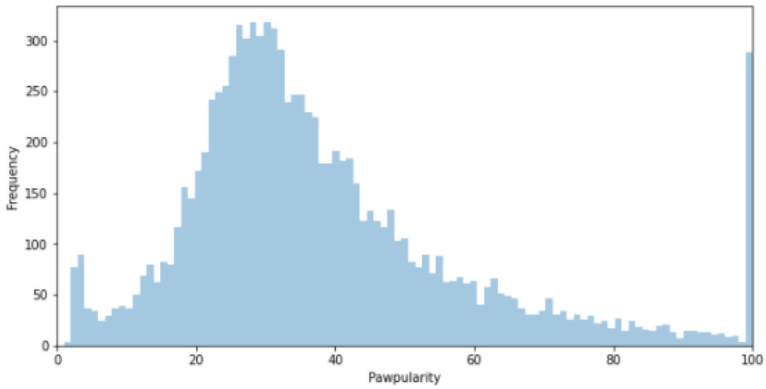
In order to use the metadata I first converted it into a dataframe and did some analysis on it. Listed below are some of the results that I thought were important in order to comprehend the data.

Here, we can see the shape of the training and testing files and different types of columns that we have.

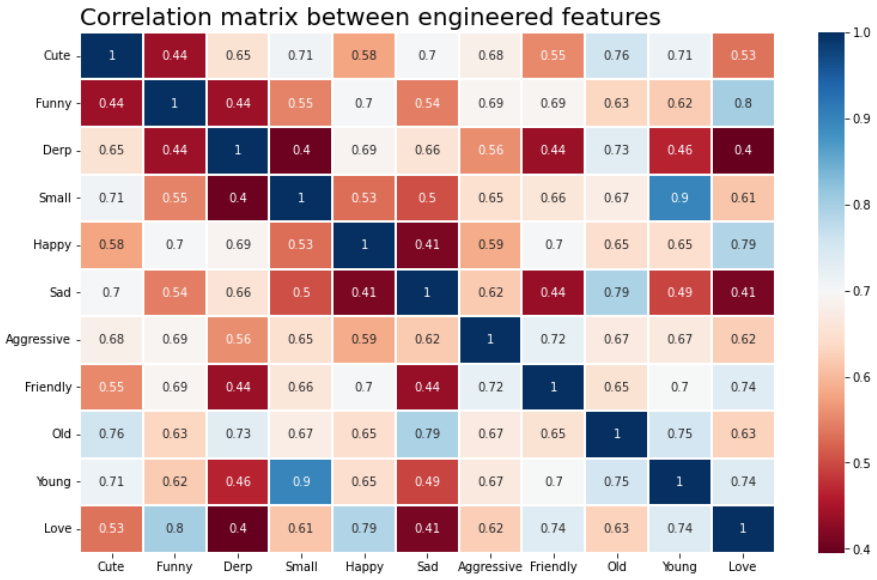
```
train_df dimensions:  (9912, 14)
train_df column names:  ['Id', 'Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Acce
ssory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blur', 'Pawpularity']

test_df dimensions:  (8, 13)
test_df column names:  ['Id', 'Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Acces
sory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blur']
```

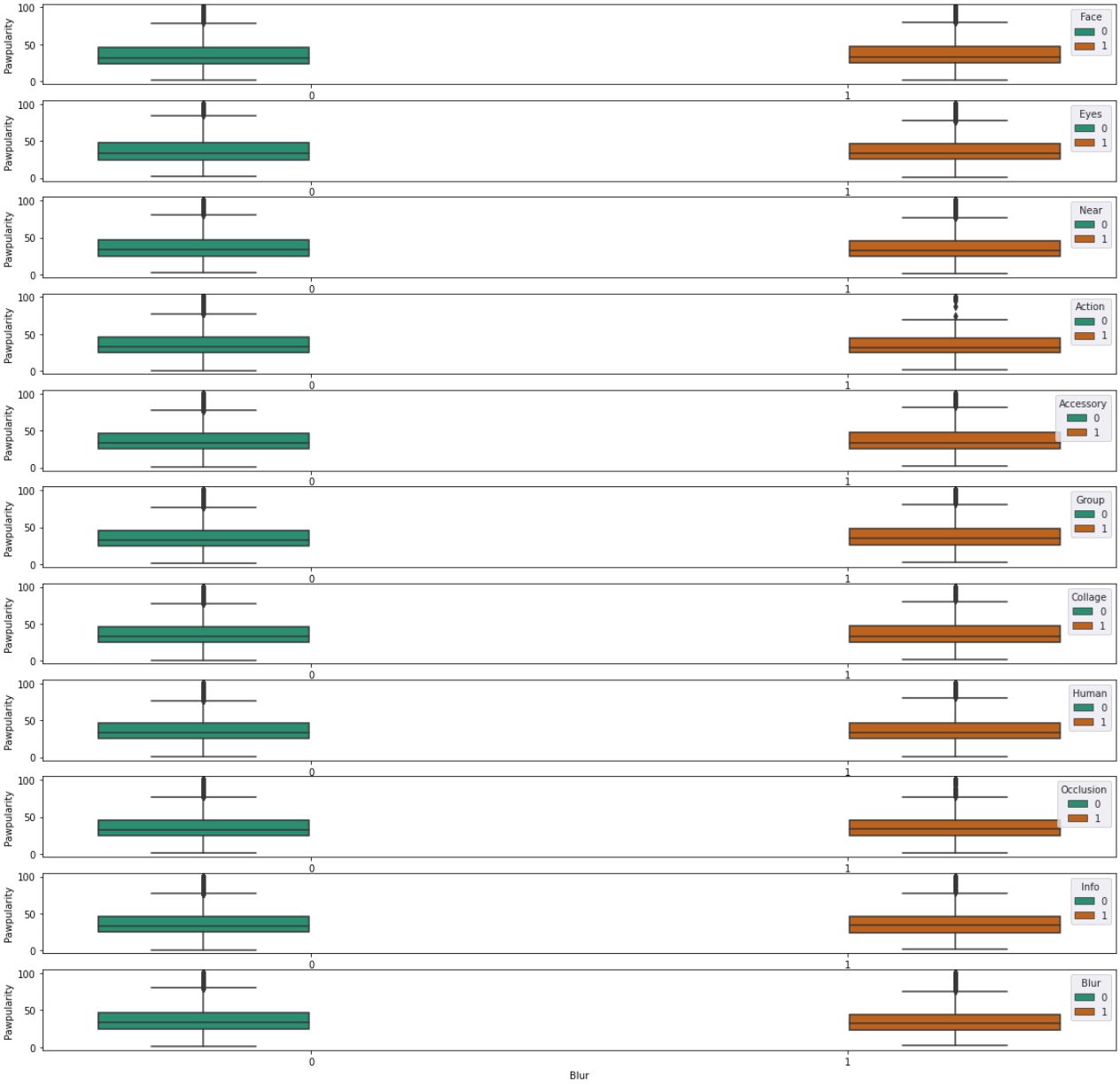
This graph shows the relationship between Popularity and the frequency of occurrence of that particular popularity number.



Here I have shown the co-relation matrix between different features of the pet animals



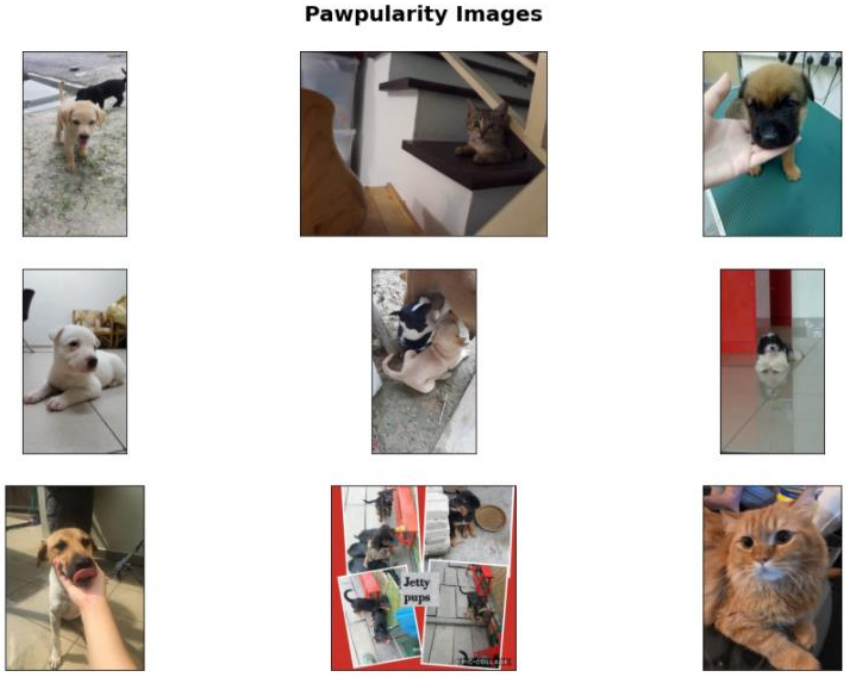
The figure belows shows us distribution of scores in respect to different features



Pre-processing and Analysis for the Image Data :

I have used different techniques for pre-processing the image data. However, before doing that I did some analysis on it in order to check how varied the data is and what kind of pre-processing and augmentation can be done on it in order to give me the most efficient results.

Here is an image of random popularity images.

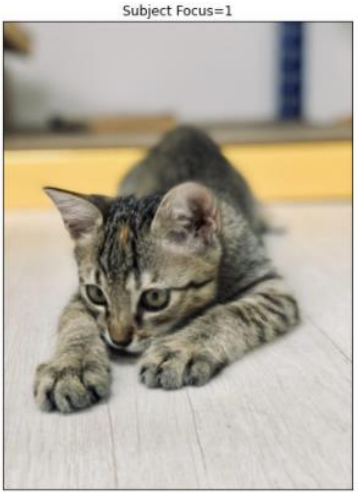


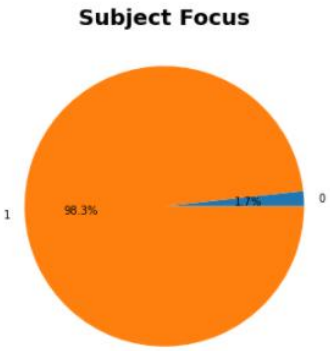
After that, I tried to check the images with the highest paw score.

Plotting the above image got me curious, hence I started exploring which features contributed the most for these high score images.

Here is a pie chart showing the top 3 features responsible for high score

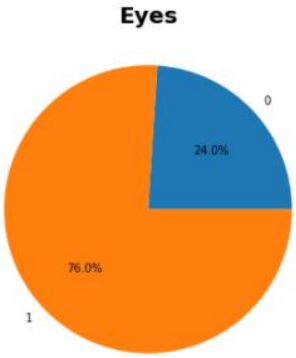
Subject Focus=1





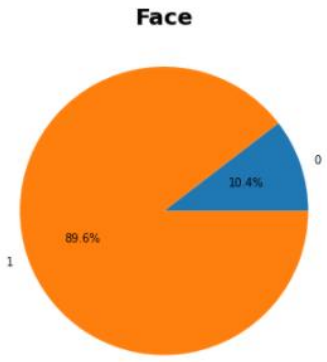
Subject Focus=0

Eyes=1



Eyes=0

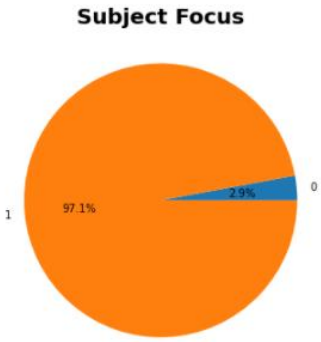
Face=1

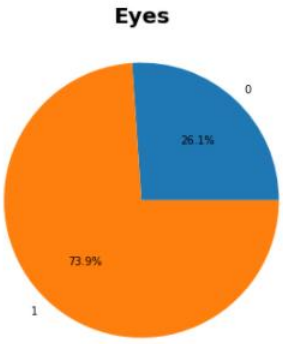


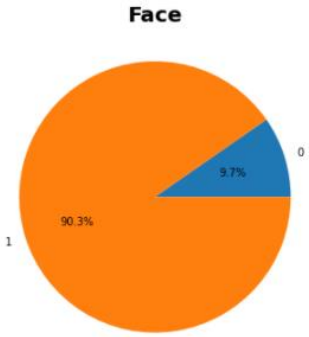
Face=0

Similarly , I tried to do the analyse the images with the lowest Paw score:

The features that contributed the most to low paw score are:







Prompt Feature Engineering and Pre-processing :

Now, for regression models (20-40) I tried to do prompt feature engineering. I did this in order to use Image data concatenated with metadata as an input to these models. I did this with the help of a framework called as CLIP by OpenAI.

CLIP (Contrastive Language-Image Pre-Training) is a neural network trained on a variety of (image, text) pairs. It can be instructed in natural language to predict the most relevant text snippet, given an image, without directly optimizing for the task, similarly to the zero-shot capabilities of GPT-2 and 3. We found CLIP matches the performance of the original ResNet50 on ImageNet “zero-shot” without using any of the original 1.28M labeled examples, overcoming several major challenges in computer vision.
With some priors around what makes animals popular, this could create some interesting features. It's possible information around concepts could be extracted from an image, with well-defined prompts
We use the cosine similarity between language and image embeddings to extract features for modelling here.

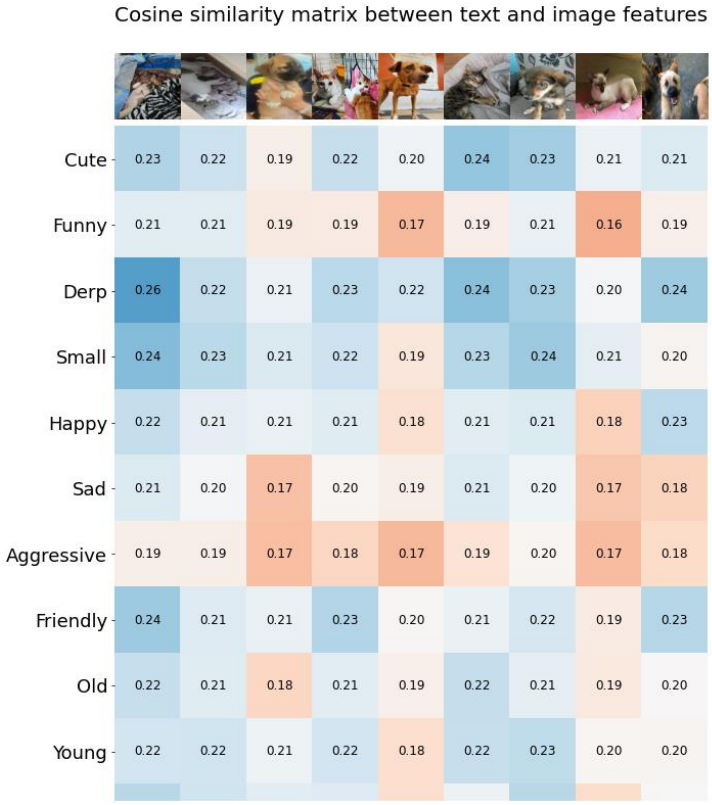
1. In order to do the task, we first try to load the dataset.

2. We then take a set of texts; in our case the “feature words”. Let’s call this Set S. Therefore, S = { 'Cute’, 'Funny', 'Derp', 'Small', 'Happy', 'Sad', 'Aggressive', 'Friendly', 'Old', 'Young', 'Love']}
3. We then tokenize the set S.

4. Once that is done, we encode both the images and the tokenized set S.

5. Once that is done, we extract the image features by normalizing the above encoding with dim = -1

6. With these features, we then plot a similarity matrix. (Diagram below)



7. Once that is done, we then use the similarity matrix in order to label the training dataset.

Apart from these, I try out some data augmentation for the images for other Deep learning tasks.

Data Augmentation:

```
1.      img = tf.io.decode_jpeg(tf.io.read_file(path), channels=3)
        img = tf.cast(img, dtype=tf.float32)
        img = tf.image.central_crop(img, 1.0)
        img = tf.image.resize(img, (224, 224))
```

```
2.  img = tf.image.random_flip_left_right(img)
      img = tf.image.random_brightness(img, 0.1)
      img = tf.image.random_saturation(img, 0.9, 1.1)
      img = tf.image.random_contrast(img, 0.9, 1.1)
```

As we can see, I have tried to augment the images by reshaping them and cropping them. I also tried to flip them and change their color properties by using functions such as brightness, saturation and contrast. I do this in order to make the typical features of the image prominent. This will help the model to recognize them very easily.

Q3.
In this section I have implemented a variety of different models. I tried to be as creative as I can.

1. First of all, I tried to use the metadata as input and implemented different types of traditional linear and extra regression models.
2. However, after running them, I realised I could use feature engineering on images, merge them with the metadata to give it as an input to these models (Feature engineering section included in Q2). I then ran these regression models on a completely different set of inputs.
3. Even after doing so much, I was not satisfied with my results.
4. In order to get more assurance, I used frameworks such as OPTUNA and AutoML which told me the best hyper-parameter settings for my regression models.
5. Lastly, I ran some models with metadata and feature engineered image data as the input along with the best set of hyper-parameters suggested by the above mentioned frameworks.
6. I used k-fold cross validation and rmse score evaluation for each and every model in this section.

Section 1:
In this section, I will start by enlisting all the regression models on whom I used the metadata as the input

---

Model 1:

1. Name of Model: Lasso Regression
   First we need to import the model from sklearn.linear_model.Lasso
   Lasso regression is a linear model in the sklearn library trained with L1 prior as a regularizer. The optimization formula for Lasso is:

   ```
   (1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
   ```

   Technically the Lasso model is optimizing the same objective function as the Elastic Net with l1_ratio = 0.1 (no L2 penalty).

2. Hyperparameters: alpha = 0.1, fit_intercept = True, max_iter = 1000, tol = 1e-4, selection = ‘cyclic’
3. Regularizer : Lasso
4. Local test score (RMSE): 20.5823

Model 2:

1. Name of model: Support Vector Regression

We import the model from sklearn.svm.SVR. Support Vector Regression (SVR) uses the same principle as SVM, but for regression problem. Thus, any hyperplane that satisfies our SVR should satisfy:

$$-a < Y - wx+b < +a$$

SVR acknowledges the presence of non-linearity in the data and provides a proficient prediction model

2. Hyperparameters: C=1.0, epsilon = 0.2, kernel ='rbf' ,gamma ='scale', tol = 1e-3,cache_size = 200
3. Regularizer = L2 or Ridge
4. Local test score (RMSE): 21.1368

Model 3:

1. Name of model: Ridge Regression

We need to import this model from sklearn.linear_model.Ridge. This model can be Linear least squares with l2 regularization. It also minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression problem where the loss function is the linear least squares function and regularization is given by the l2-norm. Ridge regression is also known as Tikhonov regularization. This estimator has built-in support for multi-variate regression.

2. Hyperparameters: alpha = 0.1, solver = 'auto', tol = '1e-3', verbose = 0, random_state = 0, check_input = True
3. Regularizer : Ridge
4. Local test score (RMSE): 20.5949

Model 4:

1. Name of model: Kernel Ridge Regression

   Kernel ridge regression (KRR) combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space.

   The form of the model learned by KRR is identical to support vector regression (SVR). However, different loss functions are used: KRR uses squared error loss while support vector regression uses epsilon-insensitive loss, both combined with l2 regularization. In contrast to SVR, fitting a KRR model can be done in closed-form and is typically faster for medium-sized datasets. On the other hand, the learned model is non-sparse and thus slower than SVR, which learns a sparse model for epsilon > 0, at prediction-time.

2. Hyperparameters: kernel = 'linear', gamma ='1.0', degree = 3, coef0 = 1
3. Regularizer = L2 or Ridge
4. Local test score (RMSE): 22.4245

Model 5:

1. Name of model: Elastic Net Regression

   Elastic Net Regression is nothing but linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

If you are interested in controlling the L1 and L2 penalty separately, keep in mind that this is equivalent to:

```
a * ||w||_1 + 0.5 * b * ||w||_2^2
```

where:

```
alpha = a + b and l1_ratio = a / (a + b)
```

2. Hyperparameters: random_state = 0, alpha = 1, l1_ratio = 0.5,fit_intercept = True, max_iter = 1000, tol = 1e-4, selection = 'cyclic'
3. Regularizer : Ridge and Lasso
4. Local test score (RMSE): 20.5909

Model 6:

1. Name of model: XGBoost Regression

   Extreme **Gradient** Boosting, or XGBoost for short, is an efficient open-source implementation of the gradient boosting algorithm. ... The two main reasons to use XGBoost are execution speed and model performance. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems.

   The objective function contains loss function and a regularization term. It tells about the difference between actual values and predicted values, i.e how far the model results are from the real values. The most common loss functions in XGBoost for regression problems is reg:linear, and that for binary classification is reg:logistics.

2. Hyperparameters n_estimators=1000, max_depth=5, eta=0.01, subsample=0.7, colsample_bytree=0.6
3. Local test score (RMSE): 20.6785

Model 7:

1. Name of model: LGBM Regression

   LightGBM extends the gradient boosting algorithm by adding a type of automatic feature selection as well as focusing on boosting examples with larger gradients. This can result in a dramatic speedup of training and improved predictive performance.

   It uses two types of techniques which are gradient Based on side sampling or GOSS and Exclusive Feature bundling or EFB. .So LGBM can be thought of as gradient boosting trees with the combination for EFB and GOSS.

2. Hyperparameters : 'num_leaves': 31, 'objective': 'binary' , boosting_type = ‘gbdt’, max_depth = -1, learning_rate = 0.1, n_estimators = 100, colsample_bytree = 1,reg_alpha = 0.1, n_jobs = -1
3. Regularizer = L1 or Lasso
4. Local test score (RMSE): 20.6787

Model 8:

1. Name of model: Linear Regression
   This model is called as Ordinary least squares Linear Regression. Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

   A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

2. Hyperparameters : fit_intercept = True, normalize = False, positive =False
3. Regularizer = Lasso or L1
4. Local test score (RMSE): 20.5950

Model 9:

1. Name of model: Extra Trees Regressor
   Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees. It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble

   This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

2. Hyperparameters = n_estimators=100, random_state = 0, mini_samples_split = 2, mini_samples_leaf = 1, max_features = ‘auto’, min_impurity_decrease = 0,bootstrap = False
3. Local test score (RMSE): 20.9539

Model 10:

1. Name of model: Decision Tree Regression

The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

2. Hyperparameters: random_ state = 0, max_depth = 2
3. Local test score (RMSE): 20.9597

Model 11:

1. Name of model: Random Forest Regression
   A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap = True (default), otherwise the whole dataset is used to build each tree.

2. Hyperparameters = n_estimators=1000, max_depth=2, random_state = 0
3. Local test score (RMSE): 20.5893

Model 12:

1. Name of model: Gaussian Processing Regression
   Gaussian process regression (GPR) is a nonparametric, Bayesian approach to regression that is making waves in the area of machine learning. GPR has several benefits, working well on small datasets and having the ability to provide uncertainty measurements on the predictions.

   Gaussian process regression is nonparametric (*i.e.* not limited by a functional form), so rather than calculating the probability distribution of parameters of a specific function, GPR calculates the probability distribution over all admissible functions that fit the data. However, similar to the above, we specify a prior (on the function space), calculate the posterior using the training data, and compute the predictive posterior distribution on our points of interest.

$$f^*|X, y, X^* \sim \mathcal{N}(\bar{f}^*, \Sigma^*)$$
$$\bar{f}^* = \mu^* + K(X^*, X)[K(X, X) + \sigma_n^2 I]^{-1}(y - \mu)$$
$$\Sigma^* = K(X^*, X^*) - K(X^*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X^*)$$

Predictive posterior distribution for GPR [1]

2. Hyperparameters : random_state = 0, kernel = DotProduct() + WhiteKernel(), alpha=1e -10,optimizer=fmin_l_bfgs_b", n_restarts_optimizer=0, normalize_y=False, copy_X_trainbool=True
3. Local test score (RMSE): 20.8996

Model 13:

1. Name of model: Gradient Boosting Regression

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

Gradient boosting is a greedy algorithm and can overfit a training dataset quickly.It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.

2. Hyperparameters: random_state = 0 , loss= squared_error, learning_rate= 0.1, n_estimators = 100,subsample = 1.0, criterion =friedman_mse
3. Local test score (RMSE): 20.6273

Model 14:

1. Name of model: AdaBoost Regression

   An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases. One can even say that it is a type of ensemble learning.

2. Hyperparameters= base_estimator= Decision Tree Regressor (max_depth = 3), n_estimators=100,learning_rate=1.0,loss='linear',random_state=0
3. Local test score (RMSE): 20.9727

Model 15:

1. Name of model: K-Neighbours Regression
   Regression based on k-nearest neighbors.

   The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

The basic nearest neighbors regression uses uniform weights: that is, each point in the local neighborhood contributes uniformly to the classification of a query

point. Under some circumstances, it can be advantageous to weight points such that nearby points contribute more to the regression than faraway points. This can be accomplished through the weights keyword. The default value, weights = 'uniform', assigns equal weights to all points. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied, which will be used to compute the weights.

2. Hyperparameters : n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, pint=2, metrics='minkowski', n_jobs=-1
3. Local test score (RMSE): 22.1728

Model 16:

1. Name of model: Cat Boost Regression
   CatBoost builds upon the theory of decision trees and gradient boosting. The main idea of boosting is to sequentially combine many weak models (a model performing slightly better than random chance) and thus through greedy search create a strong competitive predictive model. Because gradient boosting fits the decision trees sequentially, the fitted trees will learn from the mistakes of former trees and hence reduce the errors. This process of adding a new function to existing ones is continued until the selected loss function is no longer minimized.

In the growing procedure of the decision trees, CatBoost does not follow similar gradient boosting models. Instead, CatBoost grows oblivious trees, which means that the trees are grown by imposing the rule that all nodes at the same level, test the same predictor with the same condition, and hence an index of a leaf can be calculated with bitwise operations. The oblivious tree procedure allows for a simple fitting scheme and efficiency on CPUs, while the tree structure operates as a regularization to find an optimal solution and avoid overfitting.

2. Hyperparameters : iterations = 200 , learning_ rate = 0.01
3. Local test score (RMSE): 20.5823

Model 17:

1. Name of model: Random Forest Quantile Regression
   In random forest quantile regression, in order to estimate F(Y=y|x)=q , each target value in `y_train` is given a weight. Formally, the weight given to `y_train[j]` while estimating the quantile is

$$\frac{1}{T}\sum_{t=1}^{T}\frac{1(y_j \in L(x))}{\sum_{i=1}^{N} 1(y_i \in L(x))}$$

   where L(x) denotes the leaf that x falls into.

Informally, what it means that for a new unknown sample, we first find the leaf that it falls into at each tree. Then for each `(X, y)` in the training data, a weight is given

to `y` at each tree in the following manner.If it is in the same leaf as the new sample, then the weight is the fraction of samples in the same leaf. If not, then the weight is zero.These weights for each `y` are summed up across all trees and averaged. Now since we have an array of target values and an array of weights corresponding to these target values, we can use this to measure empirical quantile estimates.

2. Hyperparameters : random_state = 0, mini_samples_split = 10, n_estimators = 1000
3. Local test score (RMSE): 20.7249

Model 18:

1. Name of model: Mondrian Tree Regression
   Mondrian forests can be grown in an incremental/online fashion and remarkably, the distribution of online Mondrian forests is the same as that of batch Mondrian forests. Mondrian forests achieve competitive predictive performance comparable with existing online random forests and periodically re-trained batch random forests, while being more than an order of magnitude faster, thus representing a better computation vs accuracy tradeoff.

2. Hyperparameters : random_state = 1, max_depth = 2
3. Local test score (RMSE): 20.5948

Model 19:

1. Name of model: Extra Trees Quantile Regression
   Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees. It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble. However in extra trees quantile regressor, the outcome is in the form of quantiles. With this algorithm we are able to plot prediction intervals.

2. Hyperparameters n_estimators=100, random_state = 0
3. Local test score (RMSE): 20.8027

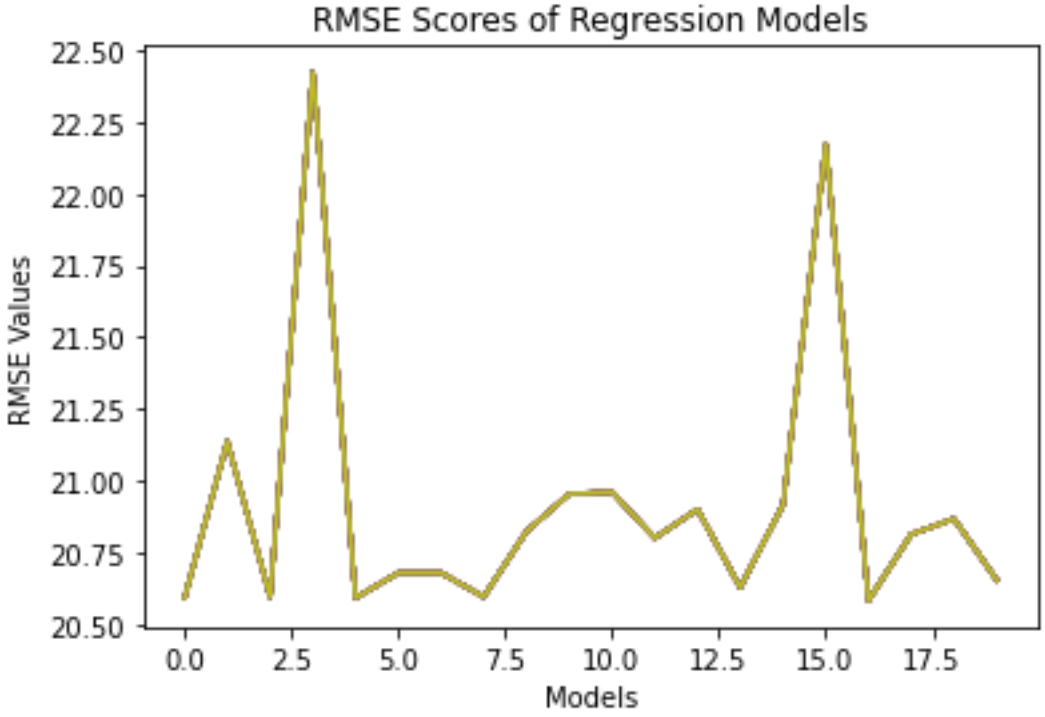
Model 20:

1. Name of model: Decision Tree Quantile Regression
   It is fairly straightforward to extend a standard decision tree to provide predictions at percentiles. When a decision tree is fit, the trick is to store not only the sufficient statistics of the target at the leaf node such as the mean and variance but also all the target values in the leaf node. At prediction, these are used to compute empirical quantile estimates.

> Let's say, the parameter `min_samples_leaf` is set to 5, then for a new sample `X`, the 5 samples in the leaf are given equal weight while determining Y|XY|X at different quantiles. If `min_samples_leaf` is set to 1, then the expectation equals the quantile at every percentile.

2. Hyperparameters : random_ state = 0
3. Local test score (RMSE): 20.9597

Here is a graph which shows the comparison of all the models I have executed till now. The order of the models is the same as I have written them down.



Section 2:
In this section, I use the metadata and the feature engineered image data using CLIP by OpenAI. I use both the kinds of data as an input to the same models listed above. I have kept the parameters same in order to get easy comparison between the two. Here are the results:

Model Number	Model Name	Local Test Score (RMSE)
21	Lasso Regression	20.6560
22	Support Vector Regression	21.1075
23	Ridge Regression	20.1168
24	Kernel Ridge Regression	19.8965
25	Elastic Net Regression	20.6551
26	XGB Regression	19.9026
27	LGBM Regression	19.8621

28	Linear Regression	19.8898
29	Extra Trees Regression	20.0786
30	Decision Tree Regression	28.3805
31	Random Forest Regression	20.2561
32	Gaussian Process Regressor	58.9553
33	Gradient Boosting Regression	20.0408
34	AdaBoost Regression	21.0614
35	KNeighbours Regression	21.6573
36	Cat Boost Regression	20.1504
37	Random Forest Quantile Regression	19.9973
38	Mondrian Tree Regression	20.6286
39	Extra Trees Quantile Regression	20.0792
40	Decision Tree Quantile Regressor	28.3805

Section 3:
In this section, I will jot the down the models that have been tuned using external hyper-parameter tuning frameworks

Model 41
I used Auto-ML(FLAML) for finding the best hyper-parameters for this model. FLAML (a fast and lightweight autoML library)is a python package that can tell us the best-fit machine learning model for low computation. Thus, it removes the burden of the manual process of choosing the best model and best parameter. Hence to solve this problem, Microsoft built an AutoML system which is mainly focused on: Model selection, Hyperparameter tuning, Feature engineering and Neural architecture search.

In this model, I tried to pass all the regression models used above and give it as an input to this model. This model then ran all the models for different iterations and different hyper-parameters to try and figure out which one works best.

> Settings : 'estimator_list':['xgboost','rf','lgbm','catboost'], 'log_file_name':'pp.log', 'task':'regression','metric':'rmse', 'time_budget':360, 'seed':2021
> Suggested Model and its hyper- parameters = LGBM Model with parameters as : colsample_bytree=0.9391605649208727, learning_rate=0.19814546649229234, max_bin=1023, min_child_samples=57, n_estimators=38, num_leaves=23, reg_alpha=0.0010079442745583322, reg_lambda=1.6292994966548278, verbose=-1
> Local Test Score with suggest model:

Model 42
In this model, I will be using XGB Regressor with OPTUNA. Optuna is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It features an imperative, *define-by-run* style user API. OPTUNA uses terms study and trial as follows:

- Study: optimization based on an objective function
- Trial: a single execution of the objective function

1. Model Name: OPTUNA with XGB Regressor
2. Best Suggested Hyper-parameters:
3. 'learning_rate': 0.18646897021392203,
4. 'reg_lambda': 19.999343440533227,
5. 'reg_alpha': 6.8173777242809015,
6. 'subsample': 0.11550972696210626,
7. 'colsample_bytree': 0.6442992796524804,
8. 'max_depth': 2}

9. Local Test Score: 19.8953

Model 43

In this model I will be using Random forest Regressor with Optuna.

1. Model Name : OPTUNA with Random forest Regressor
2. Hyperparameters : 'criterion': 'mse',

'bootstrap': 'True',
'max_depth': 90,
'max_features': 'sqrt',
'max_leaf_nodes': 982,
'n_estimators': 500}

3. Local Test Score: 19.7130

Q4.

Section 4:

For the ensemble task, I used a voting regressor, bagging regressor, and the Mondrian forest regressor.

Model 44

I used a Voting Regressor in this model. In order to import a Voting Regressor, we use the model available at sklearn.ensemble.VotingRegressor. A voting regressor is an ensemble meta-estimator that fits several base regressors, each on the whole dataset. Then it averages the individual predictions to form a final prediction.

1. The Base classifiers I used for this model are: Catboost Regressor, XGB Regressor, Gradient Boosting Regressor
2. Their combination in ensemble: First the Catboost regressor will be executed. After that the XGB and then Gradient Boosting Regressor.

Since I am using a Voting Regressor, every model will be used 100% and then its result will be averaged out with the results of other base regression models.

3. Hyperparameters:
   Cat Boost Regressor: iterations = 200, learning_rate = 0.01
   XGB Regressor = n_estimators = 1000, max_depth = 5, eta = 0.01, subsample = 0.7, colsample_bytree = 0.6
   Gradient Boosting Regressor = random_state = 0

4. Local test score:20.0365

Model 45

In this model, I used the Voting regressor as well.

1. The Base classifiers I used for this model are: Lasso Regressor, XGB Regressor, Support Vector Regressor
2. Their combination in ensemble: Since I am using a Voting Regressor, every model will be used 100% and then its result will be averaged out with the results of other base regression models.
3. Hyperparameters:
   Lasso: alpha = 0.1
   XGB Regressor = = n_estimators = 1000, max_depth = 5, eta = 0.01, subsample = 0.7, colsample_bytree = 0.6
   Support Vector Regressor = C=1.0, epsilon=0.2, kernel ='rbf'
4. Local test score: 20.37099

Model 46

In this model, I used the Voting regressor with the Base classifiers as: CatBoost Regressor, Random Forest Regressor, Lasso Regressor. I was inspired to use these classifiers as they had the lowest RMSE score in the above mentioned models from 1 to 20

1. Base classifiers: CatBoost Regressor, Random Forest Regressor, Lasso Regressor.
2. Their combination in ensemble: Since I am using a Voting Regressor, every model will be used 100% and then its result will be averaged out with the results of other base regression models.
3. Hyperparameters:
   Lasso: alpha = 0.1
   Cat Boost Regressor: iterations = 200, learning_rate = 0.01
   Random Forest Regressor = max_depth = 2, random_state = 0
4. Local test score: 20.581638

Model 47:

I used Bagging Regressor in this model.

A Bagging regressor is an ensemble meta-estimator that fits base regressor each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box

estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

1. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting .
2. If samples are drawn with replacement, then the method is known as Bagging .
3. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces .
4. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches.

Base estimator: Cat Boost Regressor

Hyperparameters : n_estimators = 10, random_state = 0

Local Test Score : 20.8041

Model 48
In this model, I make use of the Mondrian Forest Regressor. The Mondrian Forest Regressor is an ensemble of MondrianTreeRegressors. The variance in the prediction of the different types of trees in this regressor is reduced by averaging the predictions from all the trees.

Base Estimator : Mondrian Tree Regressor
Hyperparameters : random_state = 1, max_depth = 2
Local Test Score : 20.5939

Q5

Section 5:
In this section, I will be exploring deep learning techniques in order to solve the Pawpularity contest.

Model 49 :

1. This model is executed using the Convolutional Neural Networks and Cat Boost Regressor. The layers in this model are as follows:

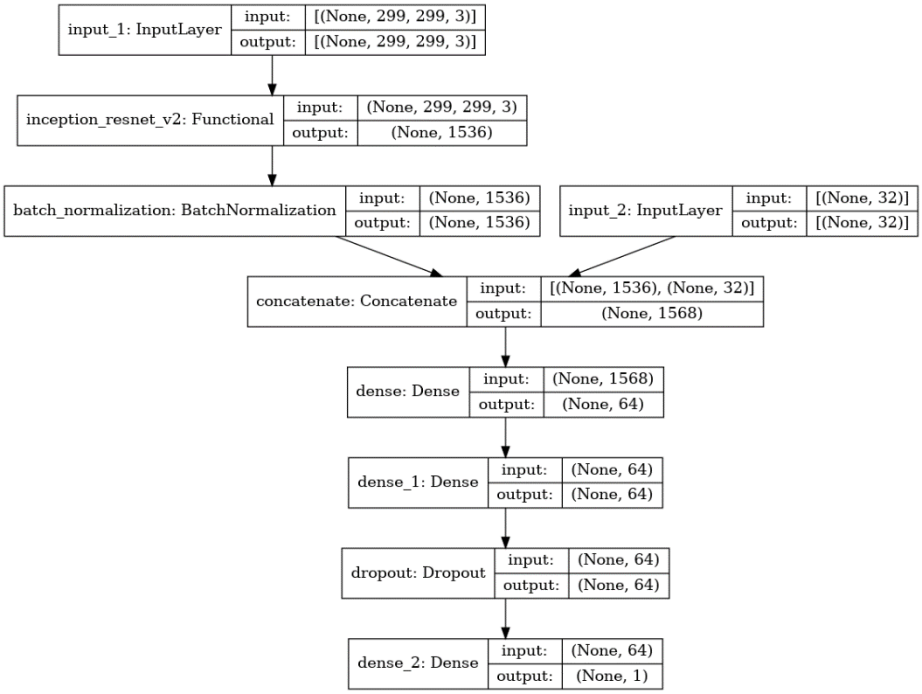
```
nn.add(Conv2D(8 , (3,3), (2,2), activation='relu', padding='same', input_shape
=(128,128,3)))
nn.add(Conv2D(16, (3,3), (2,2), activation='relu', padding='same'))
nn.add(Conv2D(32, (3,3), (2,2), activation='relu', padding='same'))
nn.add(Flatten())
nn.add(Dense(units=128, activation='relu'))
nn.add(Dropout(rate=0.5, seed=2021))
nn.add(Dense(units=1))
```

2. I used Adam Optimizer with the loss as MSE. The total number of parameters over here are 1,064,865. Out of these, all are trainable parameters.

   Once done with Neural Networks, I combined the output with Cat Boost Regressor.
3. Hyper-parameters for the Cat Boost Regressor:
   Learning Rate = 0.01
   Iterations = 200

4. Local Test Score : 20.6919

Model 50:
In this model, I try to use a pretrained model from the Keras-pretrained-models collection called as InceptionResnetV2. The layers of the model are as I have mentioned it below:



Model diagram

The model uses both metadata and Image data as input.

Local Test Score : 18.8117

Model 51 :
In this model, I make use of the pre-trained Efficient Net Model.
Layers in the model:

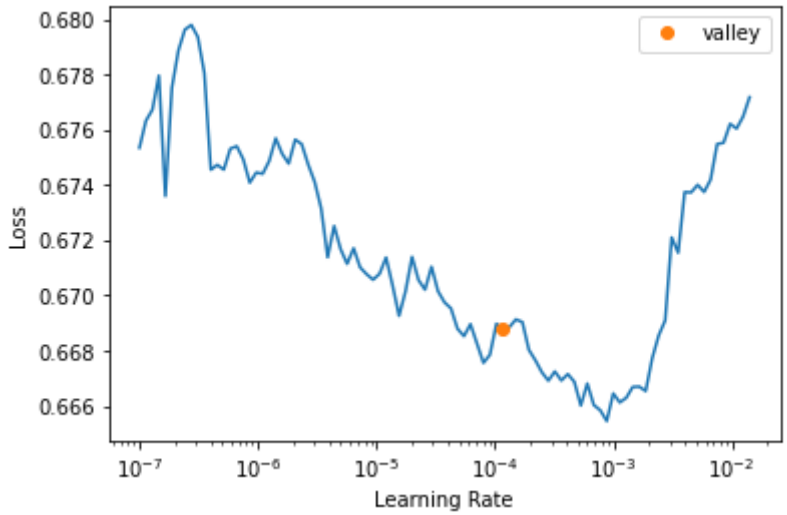
```
Input(shape=(224, 224, 3)),
    EfficientNet_model,
    BatchNormalization(),
    Dropout(0.2),
    Dense(units=64, activation="relu"),
    Dense(units=1, activation="relu")
```

I ran the model for 25 epochs and also used early stopping.

Local Test Score: 18.434

Model 52 :
In this model, I tried to use the SWIN Transformer. The SWIN Transformer is a type of vision Transformer. It generates feature maps hierarchically by merging image patches. It tries to do this usually in the deeper layers. It has linear computation complexity to input image size due to computation of self- attention only within each local window. This model gave me the best result possible.

In this process, we use a pretrained SWIN Transformer model and pass it through K – fold cross validation. The loss function I will use is BCEWithLogitsLossFlat()
It is a loss function from FastAI and it tries to flatten the input and the Target.
Other hyper-parameters : seed=999, y_block=RegressionBlock, num_workers=8



I use the API from FastAI in order to suggest me a learning rate. As we can see, the s uggested LR is `valley=0.00011673145490931347`

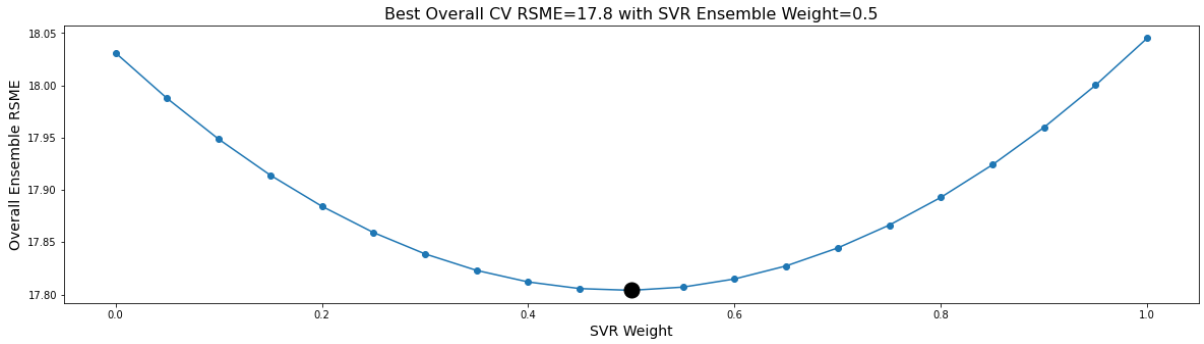
Local Test Score: 17.2606

Model 53:
In this model, use a combination of SVR and SWIN Transformer. I make use of RAPIDS. The RAPIDS suite of open-source software libraries gives the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs.I try to use the SWIM Transformer and SVR together to make a DL enhanced model.
Parameters for SVR are C = 20.0, kernel = ‘rbf’ This model uses both metadata and Image Data as input.

Steps in this process:
Step 1: Train Neural Network Model and calculate the result
Step 2: Train RAPIDS SVR Model in order to give us better and efficient prediction

Step 3: Average both the predictions and give out one common prediction

Local Test Score: 17.8437



Section 6:

In this section we will discover different types of transfer learning approaches:

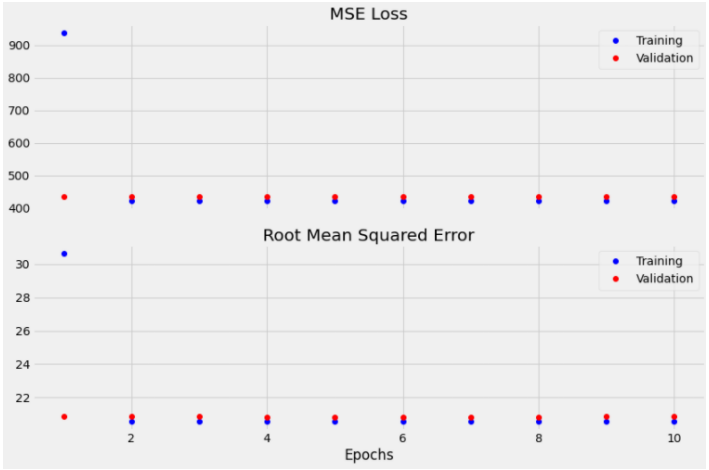
Model 54 :

In this model I try to use ResNetV2. This is a pretrained model on the ImageNet dataset. The layers in this model are as given below:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_flip (RandomFlip)     (None, 224, 224, 3)       0
_________________________________________________________________
resnet50v2 (Functional)      (None, 1000)              25613800
_________________________________________________________________
batch_normalization (BatchNo (None, 1000)              4000
_________________________________________________________________
top_dropout (Dropout)        (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 32)                32032
_________________________________________________________________
score (Dense)                (None, 1)                 33
=================================================================
Total params: 25,649,865
Trainable params: 34,065
Non-trainable params: 25,615,800
_________________________________________________________________
```

I used the Adam Optimizer. The learning rate was learnt by the model with the help of Exponential Decay Scheduler.

Local Test Score : 20.8457



Model 55 :
In this model I try to use EfficientNetB7_Top_ImageNet. This is a pretrained model on the ImageNet dataset. The layers in this model are as given below:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_flip_2 (RandomFlip)   (None, 600, 600, 3)       0
_________________________________________________________________
efficientnetb7 (Functional)  (None, 1000)              66658687
_________________________________________________________________
batch_normalization_2 (Batch (None, 1000)              4000
_________________________________________________________________
top_dropout (Dropout)        (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                32032
_________________________________________________________________
score (Dense)                (None, 1)                 33
=================================================================
Total params: 66,694,752
Trainable params: 34,065
Non-trainable params: 66,660,687
_________________________________________________________________
```

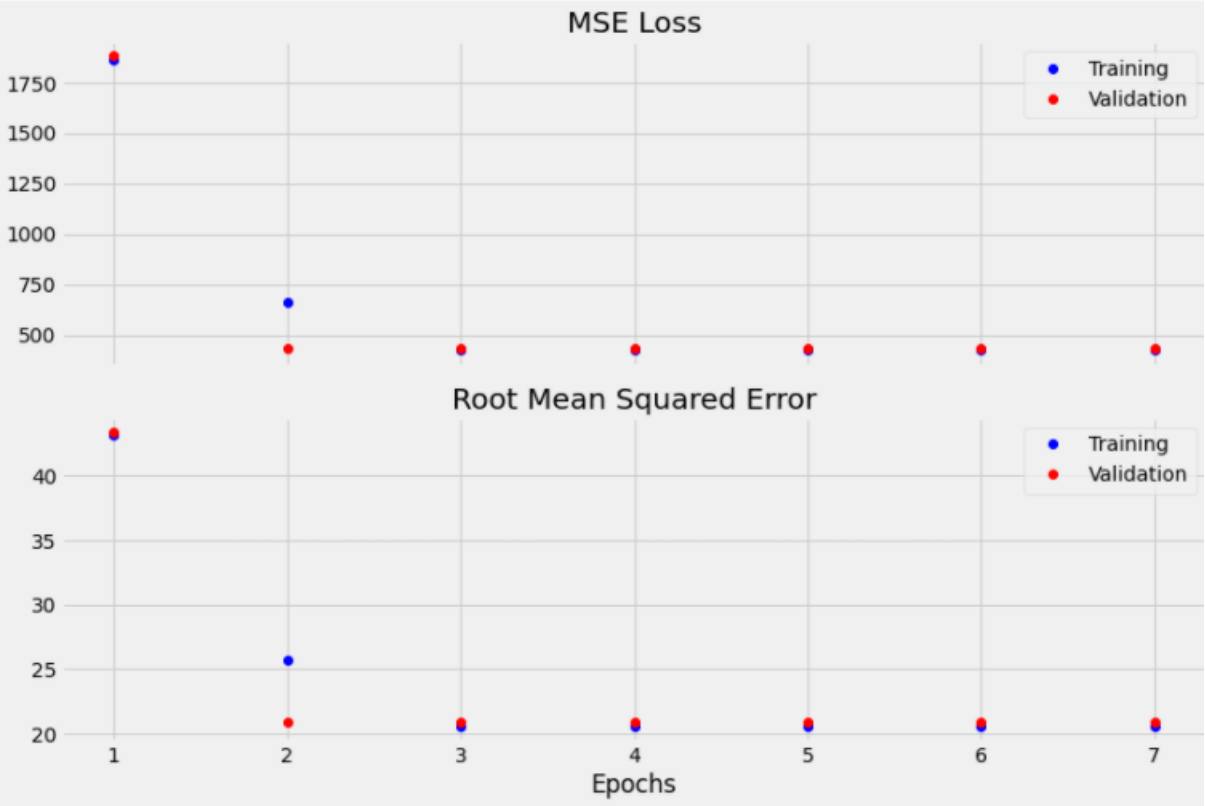
I used the Adam Optimizer. The learning rate was learnt by the model with the help of Exponential Decay Scheduler.
Local Test Score : 19.3403
Due to computation issues I only ran this model for 4 epochs. But I am sure if I did run for more, I would get a lower RMSE Score.

Model **56** :

In this model, I used MobileNet_1. This model was not trained on Image Net data set first. The layers in this model we as follows:

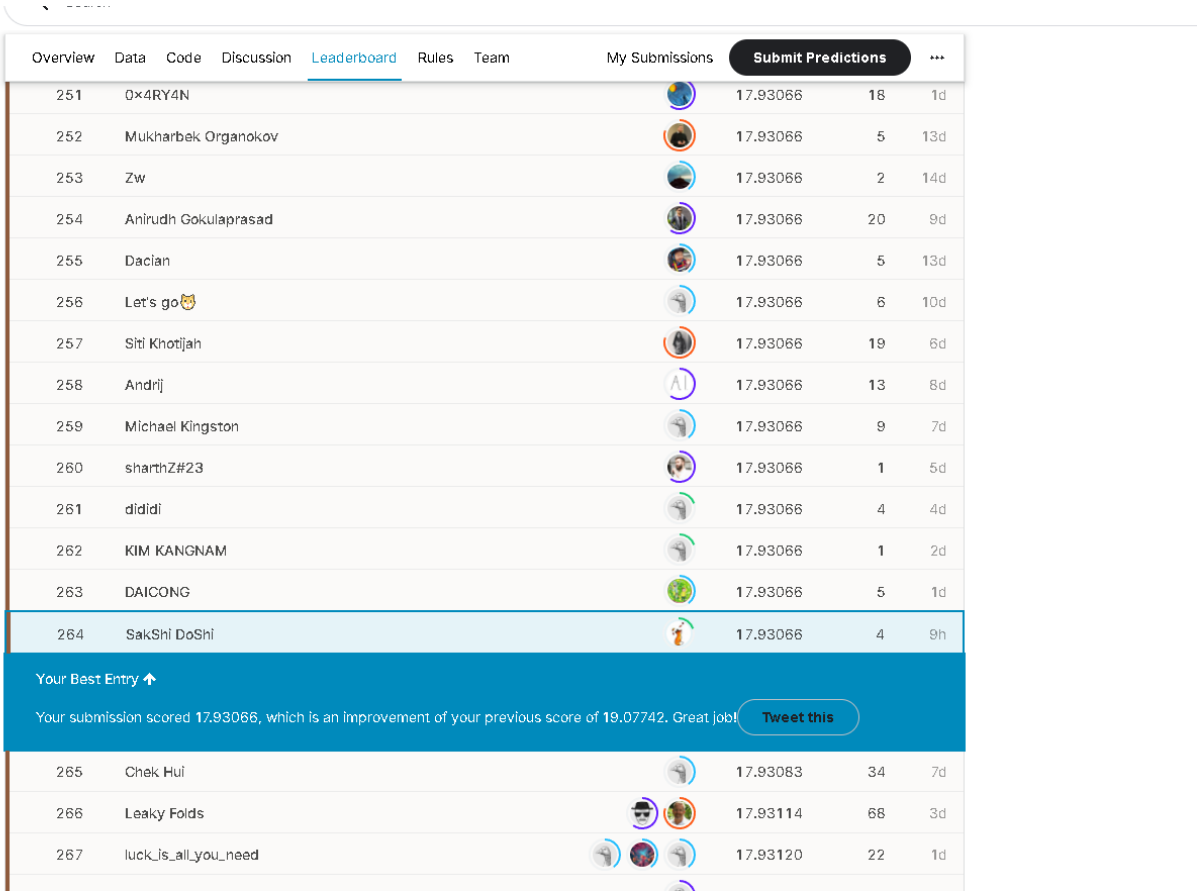
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_flip_3 (RandomFlip)   (None, 224, 224, 3)       0
_________________________________________________________________
mobilenet_1.00_224 (Function (None, 1000)              4253864
_________________________________________________________________
batch_normalization_3 (Batch (None, 1000)              4000
_________________________________________________________________
top_dropout (Dropout)        (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                32032
_________________________________________________________________
score (Dense)                (None, 1)                 33
=================================================================
Total params: 4,289,929
Trainable params: 34,065
Non-trainable params: 4,255,864
_________________________________________________________________
```

I used the Adam Optimizer. The learning rate was learnt by the model with the help of Exponential Decay Scheduler.
Local Test Score : 20.8603



Q6. I submitted my best performing model of SWIN Transformer and got a score of 17.93066

Q7.



Overview Data Code Discussion Leaderboard Rules Team My Submissions Submit Predictions

251	0×4RY4N	17.93066	18	1d
252	Mukharbek Organokov	17.93066	5	13d
253	Zw	17.93066	2	14d
254	Anirudh Gokulaprasad	17.93066	20	9d
255	Dacian	17.93066	5	13d
256	Let's go	17.93066	6	10d
257	Siti Khotijah	17.93066	19	6d
258	Andrij	17.93066	13	8d
259	Michael Kingston	17.93066	9	7d
260	sharthZ#23	17.93066	1	5d
261	dididi	17.93066	4	4d
262	KIM KANGNAM	17.93066	1	2d
263	DAICONG	17.93066	5	1d
264	SakShi DoShi	17.93066	4	9h

Your Best Entry

Your submission scored 17.93066, which is an improvement of your previous score of 19.07742. Great job! Tweet this

265	Chek Hui	17.93083	34	7d
266	Leaky Folds	17.93114	68	3d
267	luck_is_all_you_need	17.93120	22	1d

I got the rank of 264 with a score of 17.93066

My Kaggle identifier is : SakShi DoShi

Lastly, I would like to thank all the Kaggle community, Prof Madalina and Tas for helping through all the problems I faced.