ricebowl

Release 1.0.0

Mar 06, 2021

Contents

CHAPTER 1

LICENSE

1.1 LICENSE

1.2 Need Help?

If you need any help with the project or documentation please contact: shivekchhabra@gmail.com

CHAPTER 2

Dataframe Pre-processing

2.1 Data-Preprocessing

Documentation of ricebowl data preprocessing. To use this simply do from ricebowl.processing import data_preproc and then use each function with data_preproc.<function>

2.1.1 read_csv

General function to read a csv file.

Parameters- Path of the csv file

Output- Dataframe

Usage:

```
df = read_csv(path)
```

2.1.2 read_excel

General function to read an excel file.

Parameters- Path of the excel file, Sheet name

Output- Dataframe

Usage:

```
df = read_excel(path,sheet_name)
```

2.1.3 reformat_col_headers

General function for formatting the column headers to lower case. All "spaces" and "-" are replaced by "_"

Parameters- Dataframe

Output- Dataframe with formatted column headers

Usage:

```
df = reformat_col_headers(df)
```

2.1.4 str_to_datetime

General function to convert string columns in date format to datetime. Add the names of the columns which need to be converted to datetime from string type.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated to datetime

Usage:

```
df = str_to_datetime(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.5 timestamp_to_datetime

General function to convert timestamp columns to datetime. Add the names of the columns which need to be converted to datetime from timestamp.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated to datetime

Usage:

```
df = timestamp_to_datetime(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.6 to_timestamp

General function to convert datetime/str columns in datetime format to timestamp. Add the names of the columns which need to be converted to timestamp.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated to timestamp

Usage:

```
df = to_timestamp(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.7 label_encode

General function to label encode the categorical columns. Add the names of the columns which need to be encoded.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated with encoded labels, label encoder object

Usage:

```
df = label_encode(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.8 one_hot_encode

General function to one-hot encode the categorical columns. Add the names of the columns which need to be encoded.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated with encoded labels

Usage:

```
df = one_hot_encode(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.9 dates_diff

General function to calculate the difference between 2 date columns.

Parameters- Dataframe, column 1, column 2, diff_type(Optional; Default='days'; Takes in 'days'/'weeks'/'months'/'years')

Output- Dataframe with a new column according to difference. For example if diff_type='weeks' then the new column will be of the name 'weeks'

Error Print- If a wrong "diff_type" is provided, prints an error message.

Usage:

```
df = dates_diff(df,col1,col2,diff_type='days')
```

2.1.10 drop_duplicates

General function to remove duplicate rows.

Parameters- Dataframe

Output- Dataframe without duplicate rows.

Usage:

```
df = drop_duplicates(df)
```

2.1.11 reset_index

General function to reset the index of the dataframe.

Parameters- Dataframe, Drop(True/False)

Output- Dataframe with a new index

Usage:

```
df = reset_index(df,drop=True)
```

2.1.12 to_dtype

General function to convert a column to a particular datatype.

Parameters- Dataframe, Data type, kwargs[column names]

Output- Dataframe with updated columns

Usage:

```
df = to_dtype(df, 'float', c1='col1', c2='col2'...., cn='col_n')
```

2.1.13 fill_mode

General function to fill null values with mode.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated. The null values in the columns will be filled with the mode of that column.

Usage:

```
df = fill_mode(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.14 fill_mean

General function to fill null values with mean.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with the columns updated. The null values in the columns will be filled with the mean of that column.

Usage:

```
df = fill_mean(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.15 melt

General function to melt data.

Parameters- Dataframe, Columns to melt(in the form of a list), New column name to be made after melting, Column name displaying values; Default- 'value'

Output- Dataframe with the columns updated. The data is melted.

Usage:

```
df = melt(df, ['col1','col2'...'col_n'], 'new_col_name_xyz', value)
```

2.1.16 split_columns

General function to make existing data a list of split values.

Parameters- Dataframe, Original Column, Separator to split on

Output- Dataframe with columns seperated. Example: if a column had dates like 2019-01-01 and we use this function with a separator '-', then the data will be modified to [2019,01,01]

Usage:

```
df = split_columns(df, 'column_name', separator='-')
```

2.1.17 remove_unwanted_chars

General function to remove unwanted characters from data.

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with unwanted characters removed. (like $,€,£,inr,¥,)

Usage:

```
df = remove_unwanted_chars(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.18 fill_num_abbreviations

General function to fill "million M", "billion B", "thousand k", "lakhs L", "crore cr".

Parameters- Dataframe, kwargs[column names]

Output- Dataframe with filled abbreviations. Example: 20k would be replaced by 20000

Usage:

```
df = fill_num_abbreviations(df, c1='col1', c2='col2' .... cn='col_n')
```

2.1.19 split_data

General function to split data for modeling purpose

Parameters- Data, Label, Test size (optional; default=0.3)

Output- xtrain, xtest, ytrain, ytest in array format.

Usage:

```
xtrain, xtest, ytrain, ytest = split_data(data, label, test_size=0.25)
```

2.1.20 find_corr

General function to find correlation excluding all null values

Parameters- Dataframe, method(optional; default='pearson')

Output- correlation data frame.

Usage:

```
corr = find_corr(df, method='spearman')
```

2.1.21 zscore_outliers

General function to find outliers in a random variable using zscore with a threshold of 2.5 for better results

Parameters- Datarame series

Output- List of outliers.

Usage:

```
outliers = zscore_outliers(df['xyz'])
```

2.1.22 standarization

General function to standardize the data using standard scaler

Parameters- Dataframe, list of columns to be converted

Output- Dataframe with updated columns

Usage:

```
df = standarization(data, list_of_cols=['xyz','abc'])
```

2.1.23 normalization

General function to normalize the data using min-max scaler

Parameters- Dataframe, list of columns to be converted

Output- Dataframe with updated columns

Usage:

```
df = normalization(data, list_of_cols=['xyz','abc'])
```

2.1.24 basic_stats

General function to get all the basic stats of the data

Parameters- Dataframe, file path to write the stats(Optional, default=None- prints on console)

Output- Basic stats in string format.

Usage:

```
stats = basic_stats(data, file = './xyz.txt')
```

CHAPTER 3

Image Pre-processing

3.1 Image Pre-Processing

Documentation of ricebowl image preprocessing. To use this simply do from ricebowl.processing import data_preproc and then use each function with image_preproc.<function>

3.1.1 read_image

Reads a single image from the path.

Parameters- Path of the image

Output- Image

Usage:

```
img=read_img('path of the image')
```

3.1.2 show_image

Displays the image with a customized title.

Parameters- Image, title(optional; Default-'img')

Output- Image (Press any key to continue/exit)

Usage:

```
img=show_image(img,title='my_title')
```

3.1.3 write_image

Writes the image at a given path location with the customized name.

Parameters- Path+<img_name.ext>, Image

Output- Image at the given location saved. (Please note: Checks for <if exists> have not been added. Same name will cause conflict)

Usage:

```
img=write_image('./foldername/image.png', img)
```

3.1.4 inverting

Inverts an image (Converts to negative)

Parameters- Image

Output- Inverted Image

Usage:

```
img=inverting(img)
```

3.1.5 gray_scale

Converts an image to grayscale

Parameters- Image

Output- Gray-Scale Image

Usage:

```
img=gray_scale(img)
```

3.1.6 resize

Resizes an image keeping the aspect ratio constant

Parameters- Image, Width, Height

Output- Resized Image

Usage:

```
img=resize(img, 100, 300)
```

3.1.7 gaussian_blurring

Blurs an image using gaussian blurring.

Parameters- Image, Kernel(Optional; Default- (21,21)

Output- Blurred Image

Usage:

```
img=gaussian_blurring(img, ksize=(21,21))
```

3.1.8 orb_features

Extracts features in the passed image

Parameters- Image

Output- Feature extracted image, Descriptor Matrix

Usage:

```
img,desc=orb_features(img)
```

3.1.9 get_images

Returns the array data of images and their labels (entire path)

Parameters- Path of the folder containing all images

Output- Data Matrix, Label array

Usage:

```
data,labels=get_images('./path')
```

3.1.10 dcm_to_png

Converts .dcm images to .png format (from the entire folder to another folder)

Parameters- Path of folder with input images in .dcm format, Path of folder to write the .png format

Output- Images written in the output folder. (Added print command showing how many images are written.)

Usage:

```
dcm_to_png('./input_folder_path/','./output_folder_path/')
```

3.1.11 denoise

Removing noise from a colored Image

Parameters- Input Image (colored)

Output- Noise removed image

Usage:

```
img=denoise(img)
```

3.1.12 binarization

Converting an image to binary using Otsu Thresholding

Parameters- Input Image

Output- Black and white image

Usage:

```
img=binarization(img)
```

3.1.13 erode

General function to erode text in Image (Performs morphological transformation - erosion)

Parameters- Input Image

Output- Eroded image

Usage:

```
img=erode(img)
```

3.1.14 find_contours

Finding contours of an image

Parameters- Input Image

Output- Array of image contours

Usage:

```
img=find_contours(img)
```

3.1.15 sharpen

Sharpens an image

Parameters- Input Image

Output- Sharpened image

Usage:

```
img=sharpen(img)
```

3.1.16 edging

Finding edges of an image (canny)

Parameters- Input Image

Output- Edges are outlined in the image

Usage:

```
img=edging(img)
```

3.1.17 autorotate

Auto rotating an image according to the degree of skewness identified

Parameters- Input Image

Output- De-skewed image

Usage:

```
img=autorotate(img)
```

3.1.18 image_enhancer

General function to adjust the image's contrast

Parameters- Input Image

Output- Image with increased contrast

Usage:

```
img=image_enhancer(img)
```

3.1.19 remove_shadow

General function to remove shadows from image

Parameters- Input Image

Output- Image with shadows removed

Usage:

```
img=remove_shadow(img)
```

CHAPTER 4

Modeling

4.1 Modeling

Documentation of ricebowl modeling. To use this simply do from ricebowl.modeling import choose_model and then use each function with choose_model.<function>

Please note all these are basic ML models and are set to be used with default parameters. This is solely done to achieve a base model result in shorter time for a variety of different models.

4.2 Classification Models:

These are the available classification models and their function names. Please make sure to do any preprocessing beforehand using processing module from ricebowl.

4.2.1 random_forest_classifier

General function for random forest classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = random_forest_classifier(training_data, training_label, test_data)
```

4.2.2 decision_tree_classifier

General function for decision tree classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = decision_tree_classifier(training_data, training_label, test_data)
```

4.2.3 svm_classifier

General function for support vector machine classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = svm_classifier(training_data, training_label, test_data)
```

4.2.4 extra_tree_classifier

General function for extra-tree classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = extra_tree_classifier(training_data, training_label, test_data)
```

4.2.5 gaussian_classifier

General function for gaussian classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = gaussian_classifier(training_data, training_label, test_data)
```

4.2.6 logistic_classifier

General function for logistic-regression i.e. classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = logistic_classifier(training_data, training_label, test_data)
```

4.2.7 logistic_cv_classifier

General function for logistic regression with cross validation i.e. classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = logistic_cv_classifier(training_data, training_label, test_data)
```

4.2.8 bernoulli_classifier

General function for bernoulli classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = bernoulli_classifier(training_data, training_label, test_data)
```

4.2.9 multinomial_classifier

General function for multinomial classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = multinomial_classifier(training_data, training_label, test_data)
```

4.2.10 sgd_classifier

General function for stochastic gradient descent classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = sgd_classifier(training_data, training_label, test_data)
```

4.2.11 passive_aggressive_classifier

General function for passive-aggressive classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = passive_aggressive_classifier(training_data, training_label, test_data)
```

4.2.12 ridge_classifier

General function for ridge classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = ridge_classifier(training_data, training_label, test_data)
```

4.2.13 mlp_classifier

General function for multi-layer-perceptron classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = mlp_classifier(training_data, training_label, test_data)
```

4.2.14 adaboost_classifier

General function for adaboost classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = adaboost_classifier(training_data, training_label, test_data)
```

4.2.15 bagging_classifier

General function for bagging classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = bagging_classifier(training_data, training_label, test_data)
```

4.2.16 lda_classifier

General function for linear-discriminant-analysis classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = lda_classifier(training_data, training_label, test_data)
```

4.2.17 qda_classifier

General function for quadratic-discriminant-analysis classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = qda_classifier(training_data, training_label, test_data)
```

4.2.18 knn_classifier

General function for k-nearest-neighbour classification.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate accuracy, f1, confusion matrix and a classification report.

Usage:

```
ypred = knn_classifier(training_data, training_label, test_data)
```

4.3 Regression Models:

These are the available regression models and their function names. Please make sure to do any preprocessing beforehand using processing module from ricebowl.

4.3.1 knn_regressor

General function for k-nearest-neighbour regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = knn_regressor(training_data, training_label, test_data)
```

4.3.2 linear_regressor

General function for linear regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = linear_regressor(training_data, training_label, test_data)
```

4.3.3 ransac_regressor

General function for ransac regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = ransac_regressor(training_data, training_label, test_data)
```

4.3.4 ARD_regressor

General function for ARD regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = ARD_regressor(training_data, training_label, test_data)
```

4.3.5 huber_regressor

General function for huber regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = huber_regressor(training_data, training_label, test_data)
```

4.3.6 sgd_regressor

General function for stochastic-gradient-descent regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = sgd_regressor(training_data, training_label, test_data)
```

4.3.7 theilsen_regressor

General function for theilsen regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = theilsen_regressor(training_data, training_label, test_data)
```

4.3.8 passive_aggressive_regressor

General function for passive aggressive regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = passive_aggressive_regressor(training_data, training_label, test_data)
```

4.3.9 mlp_regressor

General function for multi-layered-perceptron regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = mlp_regressor(training_data, training_label, test_data)
```

4.3.10 adaboost_regressor

General function for adaboost regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = adaboost_regressor(training_data, training_label, test_data)
```

4.3.11 random_forest_regressor

General function for random-forest regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = random_forest_regressor(training_data, training_label, test_data)
```

4.3.12 decision_tree_regressor

General function for decision tree regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = decision_tree_regressor(training_data, training_label, test_data)
```

4.3.13 svm_regressor

General function for svm regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = svm_regressor(training_data, training_label, test_data)
```

4.3.14 bagging_regressor

General function for bagging regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = bagging_regressor(training_data, training_label, test_data)
```

4.3.15 extra_tree_regressor

General function for extra tree regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = extra_tree_regressor(training_data, training_label, test_data)
```

4.3.16 lasso_regressor

General function for Lasso regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = lasso_regressor(training_data, training_label, test_data)
```

4.3.17 ridge_regressor

General function for Ridge regression.

Parameters- training data, training label, test data

Please note these parameters can be in the form of a list/ numpy array/ pandas dataframes.

Output- Predicted values in the form of a dataframe series. These can then be used as is or with metrics module to generate rmse, r2 score and mape.

Usage:

```
ypred = ridge_regressor(training_data, training_label, test_data)
```

CHAPTER 5

Metrics

5.1 Metrics

Documentation of ricebowl metrics. To use this simply do from ricebowl import metrics and then use each function with metrics.<function>

Please note all these are basic metrics outputted in a string format for your convenience.

5.1.1 classifier_outputs

General function for producing classification metric outputs.

Parameters- y_test(expected output), y_pred(observed output), f1_average(average parameter for calculating f1 score. Optional; Default= 'micro') Please note these parameters can be in the form of a list/ numpy array/ pandas series.

Output- Single string object containing accuracy, f1, confusion matrix and a classification report.

Usage:

```
output = classifier_outputs(y_test, y_pred, f1_average='micro')
print(output)
```

5.1.2 regression_outputs

General function for producing regression metric outputs.

Parameters- y_test(expected output), y_pred(observed output) Please note these parameters can be in the form of a list/ numpy array/ pandas series.

Output- Single string object containing r2 score, mape and rmse.

Usage:

```
output = regression_outputs(y_test, y_pred)
print(output)
```

CHAPTER 6

Plotting

6.1 Plotting

Documentation of ricebowl plotting. To use this simply do from ricebowl import plotting and then use each function with plotting.<function>

Please note all these are basic plotting graphs and will be plotted. You will have to press 'q' for another graph.

6.1.1 pairplot

General function to get a pair plot of the entire data. Hue can be modified to get the data along a single categorical column. You can see all types of information using this graph.

Parameters- Dataframe, hue(optional ; default=None), columns to plot for(optional ; default=['ALL'])

Output- Graph is plotted

Usage:

```
pairplot(df, hue='species', cols=['a',b'])
```

6.1.2 distribution

General function to get plots for all columns passed. Press 'q' for next figure. You can check the distribution of the data using these graphs

Parameters- Dataframe, kwargs[column names]

Output- Graph is plotted

Usage:

```
distribution(df, c1='xyz', c2='abc')
```

6.1.3 plot

General function to plot relationship between 2 random variables. x,y have input types as list/df series.

Parameters- x, y, xlabel(optional ; default='x'), ylabel(optional ; default='y') Please note: x and y can be either list or df series.

Output- Graph is plotted

Usage:

```
plot(x, y, xlabel='fruits', ylabel='prices')
```

6.1.4 scatter

General function to plot a scatterplot of the data

Parameters- data, x(optional ; default=None), y(optional ; default=None)

Output- Graph is plotted

Usage:

```
scatter(data, 'length', 'width')
```

6.1.5 box

General function to plot a boxplot and check the outliers.

Parameters- data

Output- Graph is plotted

Usage:

```
box(data)
```

6.1.6 pie_chart

General function to plot a pie chart.

Parameters- data, column name, title(optional; default- column name), labels(list; optional; default='None'), convert to label encoded format (True/False; default- False)

Output- Graph is plotted

Usage:

```
pie_chart(data, 'gender', title='Sex ratio', labels=["Male","Female"], convert=True)
```