Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Module: Machine Learning
Date: 28/12/22
Name: Hamza Gabajiwala

Final Assignment

1. Feature Engineering

1.1 - Data Preprocessing

A. Listings

The *listings.csv* file consists of 7566 unique listings with 75 columns (features). These columns contain essential information about the host, the listing, and the neighbourhood.

- *Host*: About, Response Time, Acceptance & Response Rate, is a Superhost or is Verified…
- *Listing*: Location, Price, Amenities, Number of Beds & Bathrooms…
- *Neighbourhood*: Overview, Name …

Following are the steps done to preprocess the *listings*:

1. Removing NaN values

Firstly, I identified the number of NaN values in the dataset. In ***Figure 1.1,*** the red columns are empty, i.e. *licence*, *neighbourhood_group_cleansed*, *bathrooms*, and *calendar_updated*. We can drop these columns as they would not be contributing to predicting the review scores. Furthermore, I have removed the NaN values in the rows using the 'dropna' function. A total of 3210 rows are left after this operation. This is done instead of imputing values, as ~30% of the values are empty, and imputing values in the data can lead to a biased model.

2. Encoding

a. Label Encoding

I have used the **LabelEncoder** from sklearn.preprocessing to label encode categorical features such as *host_is_superhost*, *host_identity_verified*, and *instant_bookable*. That contains elements such as True or False. For Example *host_is_superhost* values: 't' & 'f'. These values are converted to '0' and '1'.

b. One Hot Encoding

Features include *room_type*, *nighbourhood_cleansed*, *host_response_time*, and *host_verifications*. For example, *room_type*: Entire home/apt, Private room, Shared room, and Hotel room. These values are encoded using the pandas get_dummies function that creates new columns of these respective labels with '1' and '0'.

3. Counting List values

The *amenities* column contains values stored as a list. The total items in the list are counted using literal_eval from ast and stored in a new column. For example, *amenities*: ['Oven', 'gym', 'Dryer'] to 3.

4. Text to Numeric

Columns such as *price*, *host_response_rate*, and *host_acceptance_rate* are numeric columns but stored with text characters such as '$', ',', and '%'. These characters are removed from the columns and are converted to integers.

5. Binning values

The target values, i.e. the review scores, are mainly above 4. Hence predicting them can give inaccurate results, as there is a significant data imbalance. Binning feature values are used when continuous variables are converted to categorised variables. I have grouped the values in a quantile range using 'pd.qcut'.

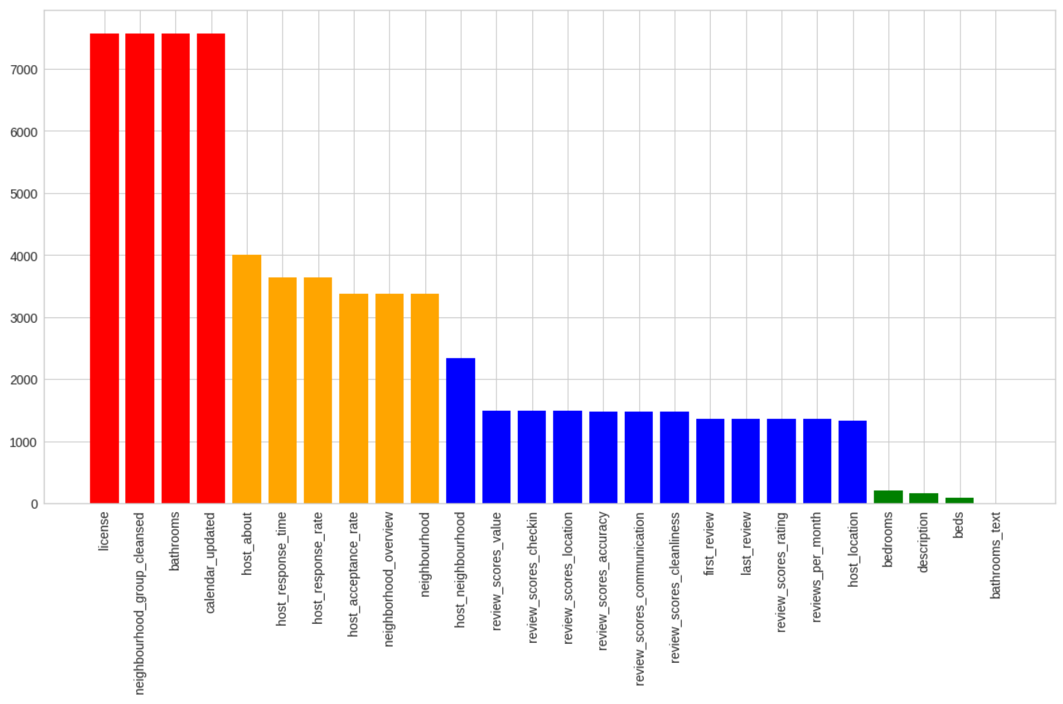


Figure 1.1: *NaN Values in Listings*

Figure 1.1.2 shows that the data is adequately balanced after the binning operation: no value is given extra importance. ***Table 1.1*** represents which value range categorises to which specific class. It can also be seen that the values mainly range above 4.5. Hence predicting these range values would be much better than continuous linear values.

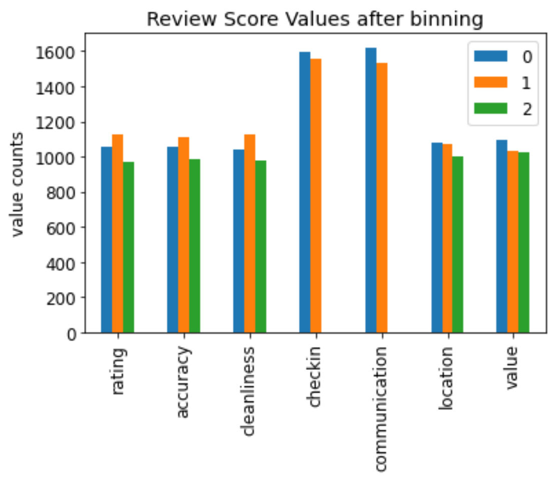


Figure 1.1.2: *Review Score values after binning*

B. Reviews

The *reviews.csv* file contains 243183 rows and six columns. Out of these columns, only the id and the comment are essential. The id will be used to map the listings in the *listings.csv* file. The following steps preprocess the reviews:

1. Drop Comments

Comments with a length of less than ten mainly contain emojis and one-word text. These comments are removed, leaving us with a total of 2839 rows dropped in the process.

2. Language of Comments

I have used an open-source library, ‘fasttext’, that contains a pre-trained model[1] to identify the language of the text passed to it.

a. The text is passed to the ‘fasttext’ model for prediction. It returns the predicted language of the text passed.
b. Only English comments are kept in the reviews. This leaves us with a total of 210482 rows.

3. Removing Emojis

Most of the online text is accompanied by emojis. These emojis also have a colour character indicating the skip type of it. These emojis are cleaned from the comments keeping the text intact.

Review Scores	*0*	*1*	*2*
Rating	[0.999, 4.71]	[4.71, 4.93]	[4.93, 5.0]
Cleanliness	[0.999, 4.697]	[4.697, 4.93]	[4.93, 5.0]
Checkin	[0.999, 4.94]	[4.94, 5.0]	*N/A*
Communication	[0.999, 4.95]	[4.95, 5.0]	*N/A*
Location	[0.999, 4.73]	[4.73, 4.92]	[4.92, 5.0]
Value	[0.999, 4.6]	[4.6, 4.82]	[4.82, 5.0]
Accuracy	[0.999, 4.8]	[4.8, 4.96]	[4.96, 5.0]

Table 1.1: *Grouping of the rating values*

4. TfIdf on Comments

a. Cleaning the text

First, the stop words from the comments are removed for the English language, using the stopwords library from nltk.corpus. Next, only textual data is kept, removing any numerical or special characters from the comments (10, $, <>). Furthermore, the text contains 'breakline' indicators that would be counted in the Tfidf Vectorizer; these 'breaklines' are removed. Lastly, the WordNetLemmatizer library is used from nltk.stem. This helps in grouping different forms of words into a single word. For Example, Books are changed to books.

b. Creating Tf Idf Vectors

A vectorise model is created using TfidfVectorizer, with the analyser set to word and the maximum number of features set to 25; it orders the features using the Term Frequency.

The ***Term Frequency*** is calculated in the following way:

$$TF_{(w,j)} = \frac{No.\ times\ term\ w\ appears\ in\ a\ document}{Total\ no.\ of\ terms\ w\ in\ the\ document}$$

The ***Inverse Document Frequency*** is calculated in the following way:

$$IDF_{(w)} = log\frac{Total\ no.\ documents}{No.\ of\ documents\ with\ term\ w\ in\ it}$$

Only a Maximum Number of Features of 25 are used as on increasing the number of features a few unnecessary words are included which are not essential in predicting the review score ratings. The vital word features used:

1. Location-based: Dublin, location, city, close, walk
2. Positive words: Clean, everything, good, great, lovely, nice, perfect, really, recommend
3. Accommodation-based: Apartment, Comfortable, easy, home, host, house, place, room, stay, well
4. Unnecessary word: Would

1.2 - Feature Selection

I have used ***Logistic Regression*** to visualise the importance of the features using the coefficients returned by the model. The features preprocessed in Section 1.1 are used.

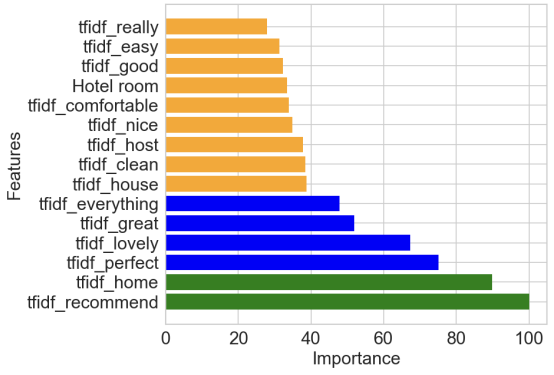


Figure 1.2: *Feature Importance for Review Score Rating*

I have used 25 *max_features* (words) from the *TfidfVectorizer*. It can be seen from ***Figure 1.2*** that *recommend*, *home*, *perfect, and lovely* from the Tf Idf Vectors and the *Hotel Room* from the listings data are considered essential features for predicting the ***Rating***. Similarly, for other review scores, the feature importance is calculated.

1.3 - Feature Scaling

Since most values lie in different ranges, ***Table 1.3*** shows the different ranges of values available in the data. I have used the ***Z-Score Normalisation*** to scale the values. The following normalisation technique was chosen to avoid outliers in the data and to scale the values for a specific feature to a standard scale. Below is the formula used:

$$x_i^{new} = \frac{x_i - \mu}{\sigma}$$

μ: *Mean*
σ: *Standard Deviation*

Column	*Min*	*Max*
Host Acceptance and Response rate	0	100
Amenities	0	76
Accommodates	1	16
Maximum & Minimum Nights	1	1125
Price	10	99149
Host Verifications	0	3

Table 1.3: *Different range of values available*

The below ***Figure 1.3*** and ***Figure 1.3.1*** displays the difference in normalising the data on the price feature. It can be noticed that there are a few (4) outliers visible because of the significant variation. On performing normalisation, the outliers are reduced, and the data lies in a small range.

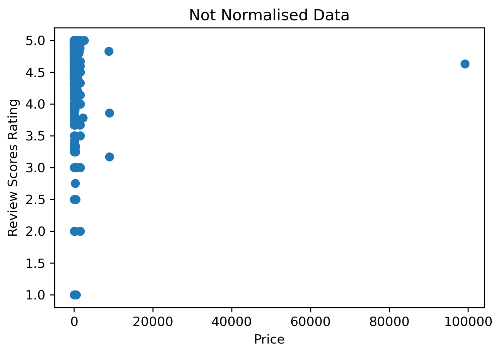


Figure 1.3: *Price Feature, not Normalised*

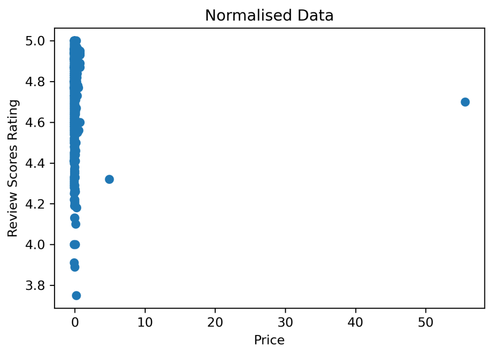


Figure 1.3.1: *Price Feature Normalised using Z-Score*

2. Methodology

Since this is now a classification problem; I have used the ***Logistic Regression*** and ***K-Nearest Neighbours*** model from sklearn.

2.1 - Logistic Regression:

This is a linear classification method with a supervised machine learning approach. The ***Logistic Regression*** model takes in a penalty. The *'l2'* (***Ridge Regression***) penalty is chosen for the model as it provided promising results compared to 'none' and 'l1' (***Lasso Regression***). This penalty (l2) helps reduce overfitting by reducing the fluctuations in the coefficients. The *l2* penalty is calculated in the following way:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + C\sum_{j=1}^{n} \theta_j^2$$

m: *Number of Samples*
x: *Features*
y: *Target Value*
C: *Amount of Inverse Regularisation*

2.2 - KNeighbors Classifier

The second model used is K-Nearest Neighbours, a simple and easy-to-implement classification problem. The weights used for the model are 'uniform'; in this, all the points in each neighbourhood are weighted equally.

3. Evaluation

The dataset consists of *3150* rows split into training and testing data. Where *20%* of the data is used for testing *630* rows. I have used ***cross_val_score*** from sklearn.model_selection to evaluate the model by varying the hyperparameters of the models.

3.1 - Logistic Regression

For this machine-learning model, the penalty is set to 'l2' (***Ridge Regression***) and varied the hyperparameter 'C' (Inverse Regularisation) values. The following values are chosen [0.01, 1.0, 2.0, 5.0, 10.0, 15.0] for training the model. ***Figure 3.1*** displays the accuracy for *review_scores_rating* on varying *C* values.

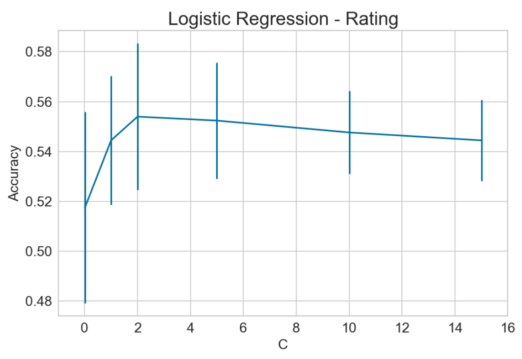


Figure 3.1: *Accuracy and MSE (Logistic Regression)*

3.2 - KNeighbors Classifier

For this machine learning model, I have evaluated the model for different 'n_neighbors'. The following n_neighbors are selected [1, 3, 5, 7, 9, 11]. As we keep increasing the 'n_neighbors', the accuracy increases, making the model simple by generalising the target values into larger clusters. ***Figure 3.2*** displays the accuracy for *review_scores_rating* on varying the *n_neighbor* values.

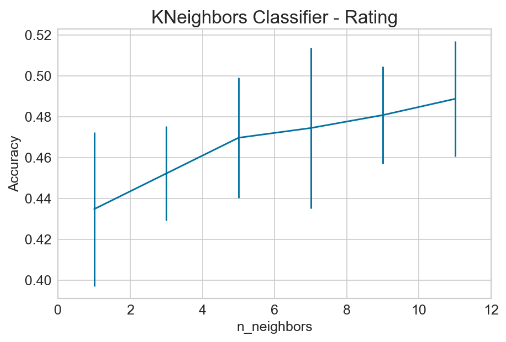


Figure 3.2: *Accuracy and MSE (KNN)*

3.3 - Baseline Model

For the baseline model, I have selected ***Dummy Classifier,*** with the strategy set to *most_frequent* and *uniform*. Firstly, the most frequent method was used to check if there was no class imbalance in the data. Secondly, the *uniform* method was used to check if the data was not uniformly distributed. On observing the accuracy and the mean squared error for both strategies, it is observed that the Most Frequent method gets a higher accuracy with a lower mean squared error. ***Figure 3.3*** displays the most frequent method performing better than the strategy in terms of accuracy.

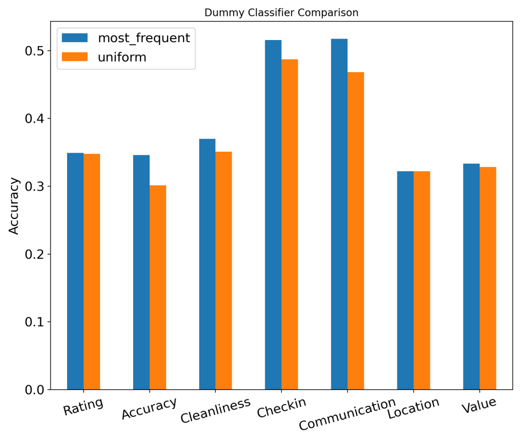


Figure 3.3: *Accuracy comparison of Dummy Classifier strategies*

3.4 - Hyperparameter Selection

The following table displays the hyperparameters selected for both models. For Logistic Regression, the *C* values are selected, similarly *n_neighbors* for the KNeighborsClassifier Model.

Review Scores	*C*	*n_neighbors*
Rating	5	7
Accuracy	1	3
Cleanliness	2	3
Checkin	10	5
Communication	5	7
Location	2	7
Value	5	7

Table 3.4: *Hyperparameter Selections*

4. Results

I have created ROC AUC Curves for the seven review scores to be predicted. ***Figure 4.1*** and ***Figure 4.2*** show that the models perform better than the baseline classifier, and the curves lie in the *True Positive Rate* (top left corner). ***Table 4.1*** displays the ***Accuracy*** of the models over different Review Scores. For the *review_score_rating,* three classes are predicted. The following scores represent these classes:

- ***0****: {0.999 to 4.71}*
- ***1****: {4.71, to 4.93}*
- ***2****: {4.93, to 5.0}*

4.1 - Logistic Regression

The *review_score_rating* gets an ***Accuracy*** of *0.61*. ***Figure 4.1*** shows that the ***Logistic Regression*** model performs better than the ***Baseline*** model (dotted line) and lies towards the *True Positive Rate*. ***Table 4.2*** displays the confusion matrix for the model.

4.2 - KNeighbors Classifier

The *review_score_rating* gets an ***Accuracy*** of *0.49*. From ***Figure 4.2***, it can be noted that the ***KNN*** model lies close to the ***Baseline*** model (dotted line). ***Table 4.3*** displays the confusion matrix for the model.

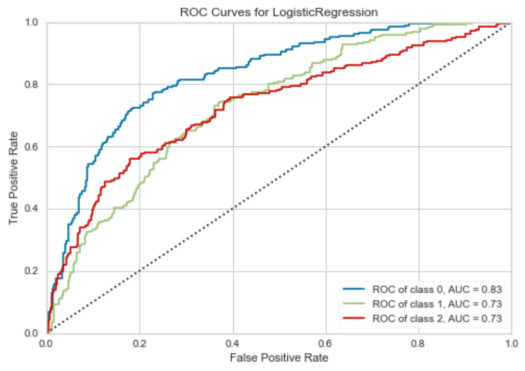


Figure 4.1: *ROC AUC Curve*

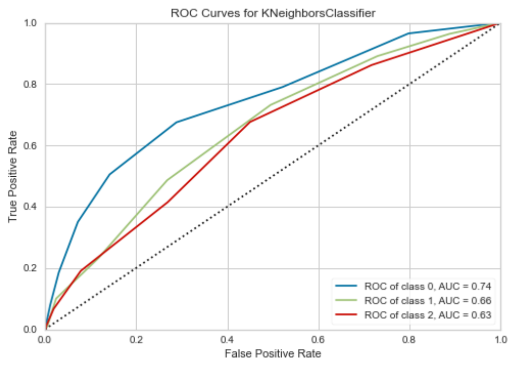


Figure 4.2: *ROC AUC Curve (KNN)*

4.3 - Feasibility Study

From ***Table 4.1***, it can be observed that the ***Logistic Regression*** outperforms both the ***K-Neighbors Classifier*** and the ***Baseline Model***. Logistic Regression is a better option for predicting the classes at which the value ranges.

Looking back at the *Feature Importance* of the features in ***Figure 1.2*** for *review_scores_rating*. We can see that the Tf Idf Vectors that create words such as *recommend, home, perfect, lovely,* and *great* are given almost 50% importance. Whilst features such as *Hotel room*, *host_is_superhost*, and *host_listings_count* from the listings data are given the importance of ~25%.

Review Scores	*Logistic Regression*	*KNN*	*Dummy Classifier*
Rating	**0.596**	0.493	0.349
Accuracy	**0.536**	0.446	0.346
Cleanliness	**0.534**	0.447	0.369
Checkin	**0.673**	0.587	0.515
Communication	**0.687**	0.612	0.517
Location	**0.519**	0.450	0.322
Value	**0.506**	0.484	0.333

Table 4.1: *Accuracy for the Models*

		Predicted		
		0	**1**	**2**
	0	150	31	19
Actual	**1**	45	135	40
	2	40	68	102

Table 4.2: *Confusion Matrix Logistic Regression*

		Predicted		
		0	**1**	**2**
	0	120	58	22
Actual	**1**	48	138	34
	2	61	99	50

Table 4.3: *Confusion Matrix KNN*

Feasibility

We cannot predict the exact review score values from the features. However, we can estimate where the review score values lie based on people's comments.

5. Question 2

A. Logistic Regression

Following are the two situations where Logistic Regression would give inaccurate results:

1. Data Imbalance

If the dataset consists of imbalanced data for the target variable, represented in ***Figure 5.1***. In this data, the target value consists of 34 '1s' and only 6 '0s'. On predicting this Feature, the model can only predict for the majority class, i.e. 1, as shown in ***Figure 5.2***.

2. No Linear Correlation

If data have no Linear Correlation between each other, the model will not be able to predict the target variable accurately. For example, I have created two features and one target variable randomly. ***Figure 5.3*** displays the two features and the respective target values. On predicting the target values from the features, it can be noticed from the following plot, ***Figure 5.4***, that the values are predicted linearly and not in the scattered way as they were created.

B. KNN vs MLP

KNN Advantages

The model relies on a straightforward calculation to determine the data points that belong to a specific class. The algorithm calculates the distance between the new data point and all the data points in the dataset and then selects the K data points closest to the

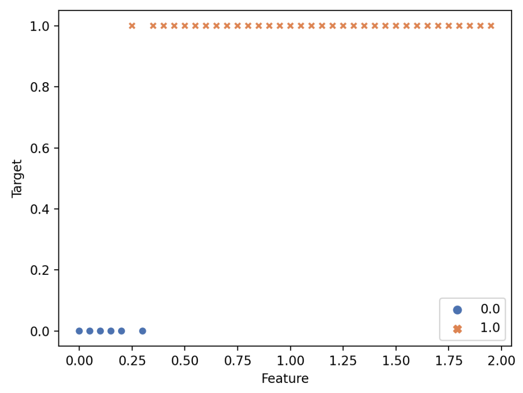


***Figure 5.1**: Data Imbalance Feature and Target Values*

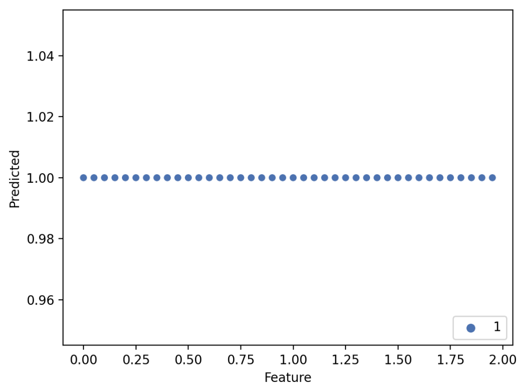


***Figure 5.2**: Prediction on Imbalanced Data*

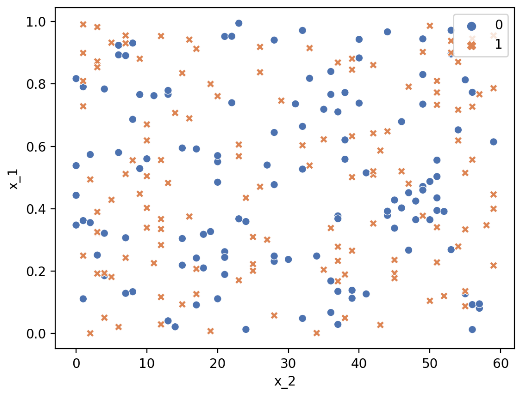


***Figure 5.3:** Features with the respective target*

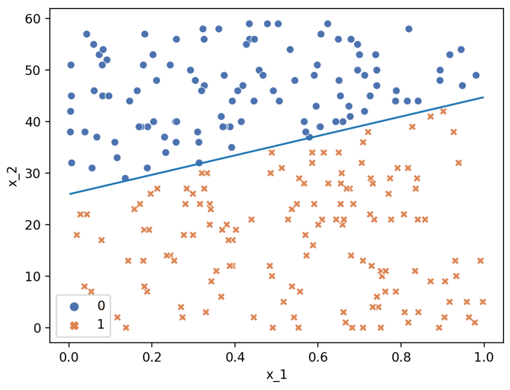


***Figure 5.4:** Features Predicted for the target value*

new data point. Furthermore, it relies on the entire dataset to make predictions. The algorithm can be used simply on a new dataset without additional training. Lastly, it can be used for both Classification and Regression. For *Classification*, the KNN model classifies the target variables by a majority vote of its neighbours. For *Regression*, the output is the property value for an object. This value is the average of its K Nearest Neighbors.

KNN Disadvantages

It can be slow at predicting the target class for a new data instance because it searches through the entire dataset to find the N-Neighbors. Furthermore, the algorithm is sensitive to irrelevant/outliers in the dataset. The data needs to be scaled appropriately to handle these outliers. Lastly, A model for the KNN algorithm is only created once a prediction is performed. An overhead when training data is enormous and prediction time is critical

MLP Advantages

MLP Classifiers can model complex relationships within the data. These classifiers can learn non-linear relationships between the input and output variables. They can also handle high-dimensional data like images, text and audio by learning distributed representations of the input data in the hidden layers. Furthermore, MLP Classifiers can be trained on large datasets and achieve high predictive accuracy on many real-world classification tasks. They can be used for predicting the target values quickly, as the model gets stored after training on the data.

Lastly, in MLP, several hyperparameters are available for tuning, such as the learning rate, activation function, number of hidden layers, amount of regularisation, dropout, and batch size.

MLP Disadvantages

If the dataset is large, it can take much time to train, especially it there are many hidden layers in the model. Due to the many hidden layers available in the model, it needs a lot of hyperparameter tuning to get satisfactory results. Furthermore, MLP models can be challenging to interpret as the model becomes complex with an increase in the number of hidden layers. Hence, making it work as a black box model does not provide a clear insight into its working.

C. *K-Fold Cross Validation*

In *K-Fold Cross Validation,* the dataset is resampled multiple times to train and evaluate the model on different subsets of the data, to understand how well the model generalises. When the model is trained on a single dataset split, it might perform poorly on other unseen data leading to an overly optimistic evaluation of the models' performance. Resampling the data multiple times across different folds can give a more reliable estimate of the model's performance on unseen data with different data splits.

For example, I have created a feature *X* using np.arange with values ranging between 0 and 1, and target *y* from feature *X*. ***Figure 5.5*** is created by randomly splitting the training data 5 times and predicting the target output for each iteration. The Mean Squared Error is calculated for these independent splits (folds). It can be seen that the MSE for each of the folds changes. It is understood that evaluating the model on a single train-test split is not a good option.

Selecting the Value of K

On selecting a small value of *k*, the model would be trained and evaluated on a limited number of folds, leading to a high bias, indicating that the model makes assumptions about the target variable. Nevertheless, on selecting a high value of *k*, the model will be trained on many folds, leading to a high variance, indicating that the model would overfit the data and be sensitive to small changes in the target variables.

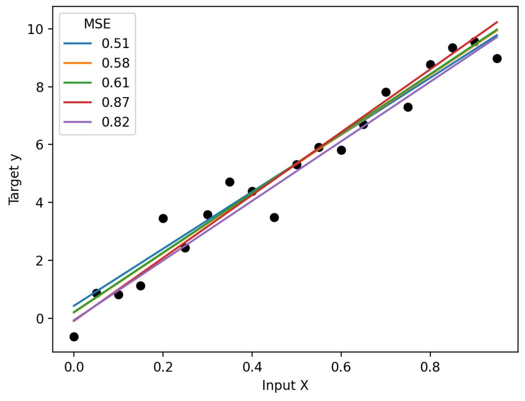
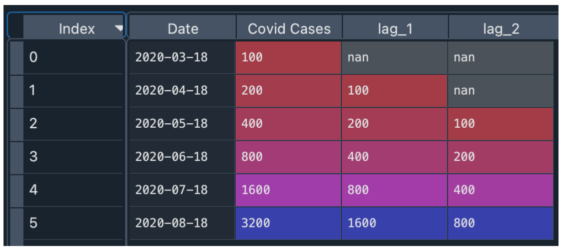


Figure 5.5: *Random data splits MSE*

D. *Time Series Data*

If we want to predict the value of a time series data at time *t+q,* q is the future value. We can use lagged output values to create features for the time series at time *t, t-1, and t-2*. These values contain valuable information to predict future values. For Example, if we want to create a feature of a times series to predict the number of covid cases per month. The lagged values are created by shifting the current value by one or two *steps*. From ***Figure 5.6,*** it can be seen that for the feature *lag_1,* the data is shifted by one step; similarly, for *lag_2,* it is shifted by two steps.



Index	Date	Covid Cases	lag_1	lag_2
0	2020-03-18	100	nan	nan
1	2020-04-18	200	100	nan
2	2020-05-18	400	200	100
3	2020-06-18	800	400	200
4	2020-07-18	1600	800	400
5	2020-08-18	3200	1600	800

Figure 5.6: *Time Series Features for Number of Covid Cases*

References

[1] FastText Language Identification https://fasttext.cc/docs/en/language-identification.html

Appendix

Code Link: [Github]

Main.py

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

from MLModels import LogisticRegressionModel, KNeighborsModel,
DummyClassifierModel
from MLModels import display_models_details

from sklearn.model_selection import cross_val_score

max_features = 25
name = "final_train_data"
if max_features != 25:
    name += f"_features-{max_features}"


def create_data():
    # LISTING DATA PRE-PROCESSING START
    # ---------------------------------
    from ListingsPreprocessing import ListingsPreprocessing
    listings_data = ListingsPreprocessing(auto_process=True,
dropNaN=True)
    # -------------------------------
    # LISTING DATA PRE-PROCESSING END

    # REVIEWS DATA PRE-PROCESSING START
    # ---------------------------------
    from ReviewsPreprocessing import ReviewsPreprocessing

    reviews_data = ReviewsPreprocessing(auto_process=True,
max_features=max_features)
    # -------------------------------
    # REVIEWS DATA PRE-PROCESSING END

    # MERGING DATA START
    # ------------------
    import pandas as pd
    FILE_PATH = ""

    reviews_polarity = pd.read_csv(f"{FILE_PATH}reviews_data.csv")
    listings_preprocessed = pd.read_csv(f"{FILE_PATH}listings_data.csv")
    merged_inner = pd.merge(left=listings_preprocessed, 
right=reviews_polarity,
                            left_on='id', right_on='listing_id')

    merged_inner.to_csv(f"{name}.csv")
    # ----------------
    # MERGING DATA END


create_new_data = False

if create_new_data:
    create_data()

# -----TARGET-----|
# rating---------0|
# accuracy-------1|
# cleanliness----2|
# checkin--------3|
# communication--4|
# location-------5|
# value----------6|
# ----------------|

# FINAL SELECTIONS
params = {
    # target
    0: {'C': 5,
        'n_neighbors': 7},  # rating
    1: {'C': 1,
        'n_neighbors': 3},  # accuracy
    2: {'C': 2,
        'n_neighbors': 3},  # cleanliness
    3: {'C': 10,
        'n_neighbors': 5},  # checkin
    4: {'C': 5,
        'n_neighbors': 7},  # communication
    5: {'C': 2,
        'n_neighbors': 7},  # location
    6: {'C': 5,
        'n_neighbors': 7},  # value
}

# stratified is a better approach for this problem statement
dummy_strats = ['most_frequent', 'stratified', 'uniform']
```

```
# PREDICTING 7 REVIEW SCORES
for target in [0, 1, 2, 3, 4, 5, 6]:
    Logi = LogisticRegressionModel(path=f"{name}.csv", target=target,
C=params[target]['C'])
    KNN = KNeighborsModel(path=f"{name}.csv", target=target,
n_neighbors=params[target]['n_neighbors'])
    Dummy = DummyClassifierModel(path=f"{name}.csv", target=target,
strategy=dummy_strats[2])
display_models_details()


# CROSS VALIDATION
"""
for target in [0, 1, 2, 3, 4, 5, 6]:
    # LOGISTIC REGRESSION START
    # -------------------------
    print('-'*10+'LOGISTIC REGRESSION'+'-'*10)
    C = [0.01, 1.0, 2.0, 5.0, 10.0, 15.0]

    mean_error = []
    std_error = []

    for c in C:
        Logi = LogisticRegressionModel(path=f"{name}.csv",
target=target, C=c)

        score = cross_val_score(Logi.model, Logi.Xtest, Logi.ytest,
cv=5, scoring='accuracy')
        mean_error.append(np.array(score).mean())
        std_error.append(np.array(score).std())

    fig = plt.figure()
    plt.errorbar(C, mean_error, yerr=std_error)
    plt.title(f'Logistic Regression - {Logi.predicting.capitalize()}',
fontsize=20)
    plt.xlim((min(C) - 1, max(C) + 1))
    plt.locator_params(axis='x', nbins=10)
    plt.xlabel('C', fontsize=15)
    plt.ylabel('Accuracy', fontsize=15)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.tight_layout()
    plt.savefig(f'LogisticRegression/CrossVal/{Logi.predicting}.png',
dpi=120)
```

```
    plt.show();
    # -----------------------
    # LOGISTIC REGRESSION END


    # KNN START
    # ---------
    print('-'*10+'KNN'+'-'*10)
    N = [1, 3, 5, 7, 9, 11]

    mean_error = []
    std_error = []

    for n in N:
        KNN = KNeighborsModel(path=f"{name}.csv", target=target,
n_neighbors=n)

        score = cross_val_score(KNN.model, KNN.Xtest, KNN.ytest, cv=5,
scoring='accuracy')
        mean_error.append(np.array(score).mean())
        std_error.append(np.array(score).std())

    fig = plt.figure()
    plt.rc('font', size=20)
    plt.errorbar(N, mean_error, yerr=std_error)
    plt.title(f'KNeighbors Classifier - {KNN.predicting.capitalize()}',
fontsize=20)
    plt.xlim((min(N) - 1, max(N) + 1))
    plt.locator_params(axis='x', nbins=10)
    plt.xlabel('n_neighbors', fontsize=15)
    plt.ylabel('Accuracy', fontsize=15)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.tight_layout()
    plt.savefig(f'KNeighborsCLassifier/CrossVal/{KNN.predicting}.png',
dpi=120)
    plt.show();
    # -------
    # KNN END


Dummy = DummyClassifierModel(target=target, strategy='uniform')
display_models_details()
"""
```

ListingsPreprocessing.py

```
import pandas as pd
from ast import literal_eval
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer(sparse_output=True)

FILE_PATH = ""

ID_FEATURE = ['id']

IMP_FEATURES = [
    'host_is_superhost','host_verifications','host_identity_verified',

'accommodates','minimum_nights','maximum_nights','instant_bookable','pri
ce',
    'host_response_time','host_response_rate','host_acceptance_rate',
    'host_listings_count','amenities',
    'bedrooms','beds',
    'room_type',
    'neighbourhood_cleansed',
]

PRED_FEATURES = [
    # PREDICTIONS
    'review_scores_rating', 'review_scores_accuracy',
    'review_scores_cleanliness', 'review_scores_checkin',
    'review_scores_communication', 'review_scores_location',
    'review_scores_value',
]


FEATURES = ID_FEATURE + IMP_FEATURES + PRED_FEATURES


ENCODE_COLUMNS = [
    'host_is_superhost',
    'host_identity_verified', 'instant_bookable',
]

BINARIZE_COLUMNS = [
    'room_type', 'neighbourhood_cleansed',
    'host_response_time', 'host_verifications'
]
```

```
label_encoder = LabelEncoder()

# SCALING LISTINGS DATA START
# ---------------------------
def z_score(x):
  return (x - x.mean())/(x.std())

def minmax(x):
  return (x - x.min()) / (x.max() - x.min())
# -------------------------
# SCALING LISTINGS DATA END


class ListingsPreprocessing:
  def __init__(self, feature_columns=None, auto_process=False,
dropNaN=False):
    if feature_columns is None:
        feature_columns = FEATURES
    self.listings = pd.read_csv(f"{FILE_PATH}listings.csv",
converters={'amenities': literal_eval,

'host_verifications': literal_eval})
    self.feature_columns = feature_columns

    self.dropNaN = dropNaN
    self.drop_columns()

    if auto_process:
      self.label_encode(columns=ENCODE_COLUMNS)
      self.count_vars('amenities', drop_column=True)
      # self.count_vars('host_verifications', drop_column=False)

      self.txt_to_numeric('$', ',', column='price')
      self.txt_to_numeric('%', column='host_response_rate')
      self.txt_to_numeric('%', column='host_acceptance_rate')

      self.bin_values()

      self.scaling_data = set(self.listings.keys()) -
set(ENCODE_COLUMNS) - set(BINARIZE_COLUMNS) - set(ID_FEATURE) -
set(PRED_FEATURES)
      self.scale_values(columns=self.scaling_data)
      self.one_hot_encode('room_type', 'neighbourhood_cleansed',
'host_response_time', 'host_verifications', pop=True)

      self.save_file()


  def drop_columns(self):

self.listings.drop(columns=self.listings.columns.difference(self.feature
_columns), axis=1, inplace=True)
      if self.dropNaN:
          self.drop_NaN()

  def drop_NaN(self):
      self.listings = self.listings.dropna()

  def label_encode(self, columns):
    for column in columns:
        if column == 'host_response_time':
            self.listings = self.listings.replace('within an hour', 3)
            self.listings = self.listings.replace('within a few hours',
2)
            self.listings = self.listings.replace('within a day', 1)
            self.listings = self.listings.replace('a few days or more',
0)
        else:
            self.listings[column] =
label_encoder.fit_transform(self.listings[column])

  def count_vars(self, column, drop_column=True):
    self.listings[f'{column}_count'] = range(0, len(self.listings))
    for index, row in self.listings.iterrows():
      self.listings[f'{column}_count'][index] = len(row[f'{column}'])
    if drop_column:
        self.listings.drop([f'{column}'], axis=1, inplace=True)

  def txt_to_numeric(self, *argv, **kwargs):
    for arg in argv:
      self.listings[kwargs['column']] =
self.listings[kwargs['column']].str.replace(arg, '')
    self.listings[kwargs['column']] =
pd.to_numeric(self.listings[kwargs['column']])

  def one_hot_encode(self, *argv, pop=False):
      for arg in argv:
          if type(self.listings[arg][0]) == type('str'):
              df = pd.get_dummies(self.listings[arg])
              self.listings = pd.concat([self.listings, df], axis=1)
              if pop:
                  self.listings.pop(arg)
          else:
              self.listings = self.listings.join(
                  pd.DataFrame.sparse.from_spmatrix(

mlb.fit_transform(self.listings.pop('host_verifications')),
                      index=self.listings.index,
                      columns=mlb.classes_))

  def bin_values(self):
    keys = ['review_scores_rating', 'review_scores_cleanliness',
            'review_scores_checkin', 'review_scores_communication',
            'review_scores_location', 'review_scores_value',
            'review_scores_accuracy']

    for key in keys:
        cuts = 3
        labels = [0, 1, 2]

        if key in ['review_scores_checkin',
'review_scores_communication']:
            cuts = 2
            labels = [0, 1]

    self.listings.review_scores_rating =
pd.qcut(self.listings.review_scores_rating, cuts, duplicates='drop',
labels=labels)

  def scale_values(self, columns=None, func=z_score):
       for f in columns:
           if f not in ['amenities', 'host_verifications', 'room_type',
'neighbourhood_cleansed', 'host_response_time']:
               self.listings[f] = func(self.listings[f])

  def save_file(self, file_name='listings_data.csv'):
    self.listings.to_csv(file_name)
```

ReviewsPreprocessing.py

```
import wget, os, re
import pandas as pd
from tqdm import tqdm
from os.path import exists

# LANGUAGE PROCESSOR
import fasttext

# WORD ANALYZING
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import TfidfVectorizer

FILE_PATH = ""

if not os.path.exists('/tmp/lid.176.bin'):
    print('Downloading model...')

wget.download("https://dl.fbaipublicfiles.com/fasttext/supervised-models
/lid.176.bin", out="/tmp/lid.176.bin")

PRETRAINED_MODEL_PATH = '/tmp/lid.176.bin'

model = fasttext.load_model(PRETRAINED_MODEL_PATH)

sid = SentimentIntensityAnalyzer()
lemmantizer = WordNetLemmatizer()

ID_FEATURE = ['listing_id']
IMP_FEATURES = ['comments']
FEATURES = ID_FEATURE + IMP_FEATURES


class ReviewsPreprocessing:
    def __init__(self, feature_columns=None, auto_process=False,
max_features=150, sentiment_analysis=False):
        if feature_columns is None:
            feature_columns = FEATURES

        self.reviews = pd.read_csv(f"{FILE_PATH}reviews.csv")
        self.feature_columns = feature_columns
        self.drop_columns()

        self.reviews_tfidf = None
        self.feature_names = None

        self.max_features = max_features
        self.stop_words_processed = False
        self.sentiment_analysis = sentiment_analysis

        self.polarity_df = pd.DataFrame([])
        self.tfidf_df = pd.DataFrame([])

        self.auto_process = auto_process
        if self.auto_process:
            self.drop_comments_with_len(_len=10)
            self.get_comment_lang()
            self.keep_comment_lang(lang='en')
            self.remove_emojis()
            self.remove_stopwords()
            self.calculate_comment_polarity()
            self.vectorize_words()
            self.listings_mean_data()
            self.merge_dataframes()
            self.save_file()

    def drop_columns(self):

self.reviews.drop(columns=self.reviews.columns.difference(self.feature_c
olumns), axis=1, inplace=True)

    def drop_comments_with_len(self, _len=10):
        self.reviews = self.reviews[self.reviews.comments.str.len() >
_len]

    def get_comment_lang(self):
        if not exists('reviews_lang.csv'):
            self.reviews['lang'] = range(0, len(self.reviews))
            for index, row in tqdm(self.reviews.iterrows(),
desc="Determining Language: ", total=self.reviews.shape[0]):
                if type(row['comments']) == type('s'):
                    predictions = model.predict(row['comments'])
                    l = predictions[0][0].split('__label__')[1]
                    if l != 'ceb' and l != 'nds' and l != 'war' and l !=
'wuu':
                        self.reviews['lang'][index] = l
            self.reviews.to_csv('reviews_lang.csv')
        else:
            self.reviews = pd.read_csv('reviews_lang.csv')

    def keep_comment_lang(self, lang='en'):
        self.reviews = self.reviews[self.reviews.lang == lang]

    def remove_emojis(self):
        self.reviews.astype(str).apply(lambda x: x.str.encode('ascii',
'ignore').str.decode('ascii'))

    def remove_stopwords(self):
        stop_words = set(stopwords.words('english'))

        self.reviews['comments'] = self.reviews['comments'].apply(
            lambda x: re.sub('[^a-zA-Z]', ' ', x))

        self.reviews['comments'] = self.reviews['comments'].apply(
            lambda x: re.sub(' br ', ' ', x))

        self.reviews['comments'] = self.reviews['comments'].apply(
            lambda x: " ".join([lemmantizer.lemmatize(item) for item in
x.lower().split() if item not in stop_words]))

        self.stop_words_processed = True

    def calculate_comment_polarity(self):
        if not exists('reviews_polarity.csv'):
            self.reviews['compound'] = range(0, len(self.reviews))
            self.reviews['pos'] = range(0, len(self.reviews))
            self.reviews['neu'] = range(0, len(self.reviews))
            self.reviews['neg'] = range(0, len(self.reviews))

            for index, row in tqdm(self.reviews.iterrows(),
desc="Calculating Sentiment: ",
                                   total=self.reviews.shape[0]):
                ss = sid.polarity_scores(row['comments'])

                self.reviews['compound'][index] = ss['compound']
                self.reviews['pos'][index] = ss['pos']
                self.reviews['neu'][index] = ss['neu']
                self.reviews['neg'][index] = ss['neg']

            self.reviews.to_csv('reviews_polarity.csv')
    def vectorize_words(self):
        self.reviews = pd.read_csv('reviews_polarity.csv')
        self.reviews = self.reviews.fillna('none', axis=1)

        _len = len(self.reviews)
        data_without_stopwords = self.reviews.comments[:_len]

        vectorizer = TfidfVectorizer(analyzer='word',
max_features=self.max_features)
        reviews_tfidf = vectorizer.fit_transform(data_without_stopwords)
        reviews_tfidf = reviews_tfidf.toarray()

        self.feature_names = vectorizer.get_feature_names()

        self.reviews_tfidf = pd.DataFrame(data=reviews_tfidf)

self.reviews_tfidf.to_csv(f'vectors_tfidf_{self.max_features}.csv',
index=False)

        self.reviews_tfidf =
pd.read_csv(f'vectors_tfidf_{self.max_features}.csv')
        self.reviews_tfidf.columns = [f"tfidf_{x}" for x in
self.feature_names]
        self.reviews_tfidf['listing_id'] = self.reviews['listing_id']

    def listings_mean_data(self):
        for key in self.feature_names:
            self.tfidf_df[f'tfidf_{key}'] =
self.reviews_tfidf.groupby(by="listing_id")[f'tfidf_{key}'].mean()

        self.tfidf_df.to_csv(f'vectors_tfidf_{self.max_features}.csv',
index=False)

        for key in ['compound', 'pos', 'neg', 'neu']:
            self.polarity_df[key] =
self.reviews.groupby(by="listing_id")[key].mean()

    def merge_dataframes(self):
        self.reviews = pd.merge(left=self.tfidf_df,
right=self.polarity_df, left_on='listing_id', right_on='listing_id')

    def save_file(self, file_name='reviews_data.csv'):
        self.reviews.to_csv(file_name)
```

MLModels.py

```
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate

# MODEL
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier

# PREPROCESSING
from sklearn.model_selection import train_test_split

# METRICS
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from yellowbrick.classifier import ROCAUC

FILE_PATH = ""

models_details = []

TARGET_LIST = ['review_scores_rating', 'review_scores_accuracy',
               'review_scores_cleanliness', 'review_scores_checkin',
               'review_scores_communication', 'review_scores_location',
               'review_scores_value']

Y_set = set(TARGET_LIST)
id_columns = ['Unnamed: 0.1', 'Unnamed: 0', 'id', 'listing_id', 'phone',
'has_identity_verified']
sentiment_remove = ['neu', 'pos', 'neg', 'compound']


class TrainingData:
    def __init__(self, path="final_train_data.csv", target=None):
        (self.X, self.y) = (None, None)
        (self.Xtrain, self.Xtest), (self.ytrain, self.ytest) = (None,
None), (None, None)
        self.model = None

        self.listings = pd.read_csv(path)
        self.needed_columns = list(
            set(self.listings.keys()) - set(id_columns)
            - set(sentiment_remove) - Y_set)

        self.target = target
        self.predicting =
TARGET_LIST[self.target].split('scores_')[1].upper()

        if target is None:
            raise ValueError("TARGET CANNOT BE NONE!")

        self.return_data()

    def return_data(self):
        self.X = self.listings[self.needed_columns]
        self.y = self.listings[TARGET_LIST[self.target]]

        self.Xtrain, self.Xtest, self.ytrain, self.ytest =
train_test_split(self.X, self.y, test_size=0.2,

random_state=1)
        return (self.Xtrain, self.Xtest), (self.ytrain, self.ytest)

    def train_data(self):
        print(f"TRAINING: {self.model}")
        print(f"PREDICTING: {self.predicting}")
        self.model.fit(self.Xtrain, self.ytrain)

    def print_metrics(self):
        model_score = self.model.score(self.Xtest, self.ytest)

        ypred = self.model.predict(self.Xtest)
        model_mean_error = mean_squared_error(self.ytest, ypred)

        models_details.append([f"{self.model} - {self.predicting}",
model_score, model_mean_error])

        print("-" * 10 + "CONFUSION-MATRIX" + "-" * 10)
        print(confusion_matrix(self.ytest, ypred))

        print("-" * 10 + "CLASSIFICATION-REPORT" + "-" * 10)
        print(classification_report(self.ytest, ypred))

    def plot_ROC_curve(self):
        visualizer = ROCAUC(self.model, encoder={0: '0',
                                                 1: '1',
                                                 2: '2', },
                              macro=False, micro=False)
        visualizer.fit(self.Xtrain, self.ytrain)
        visualizer.score(self.Xtest, self.ytest)

        self.model_name = str(self.model).split('(')[0]


visualizer.show(outpath=f"{self.model_name}/ROCAUC/{self.predicting}.png
")
        return visualizer

    def feature_importance(self):
        feature_importance = abs(self.model.coef_[0])
        feature_importance = 100.0 * (feature_importance /
feature_importance.max())


        top_features = pd.DataFrame({'feature_imp': feature_importance,
                                     'features': self.X.columns},
                                    columns=['feature_imp', 'features'])


        top_features = top_features.sort_values(by='feature_imp',
                                                ascending=False)


top_features.to_csv(f"{self.model_name}/FeatureImportance/{self.predicti
ng}.csv", index=False)
        top_features = top_features.head(15)

        col = []
        for val in top_features.feature_imp:
            if val < 10:
                col.append('red')
            elif 10 < val < 40:
                col.append('orange')
            elif 40 < val < 80:
                col.append('blue')
            else:
                col.append('green')


        # fig = plt.figure()
        # plt.barh(top_features.features, top_features.feature_imp,
color=col)
        # plt.xlabel('Importance', fontsize=18)
        # plt.ylabel('Features', fontsize=18)
        # plt.xticks(fontsize=18)
        # plt.yticks(fontsize=18)
        # plt.title(f'{self.predicting.title()} Feature Importance',
fontsize=18)
        # plt.tight_layout()
        #
plt.savefig(f"{self.model_name}/FeatureImportance/{self.predicting}.png"
,
        #             dpi=120)
        # plt.show();
        # plt.close();


def display_models_details():
    print(tabulate(models_details, headers=['Model', 'Score', 'Mean
Error']))


class LogisticRegressionModel(TrainingData):
    def __init__(self, path=None, target=None, penalty='l2', C=1.0):
        super().__init__(path=path, target=target)
        self.model = LogisticRegression(penalty=penalty, C=C)
        self.train_data()
        self.print_metrics()
        self.plot_ROC_curve()
        self.feature_importance()


class KNeighborsModel(TrainingData):
    def __init__(self, path=None, target=None, n_neighbors=5):
        super().__init__(path=path, target=target)
        self.model = KNeighborsClassifier(n_neighbors=n_neighbors,
                                          weights='uniform')
        self.train_data()
        self.print_metrics()
        self.plot_ROC_curve()


class DummyClassifierModel(TrainingData):
    def __init__(self, path=None, target=None,
strategy='most_frequent'):
        super().__init__(path=path, target=target)
        self.model = DummyClassifier(strategy=strategy)
        self.train_data()
        self.print_metrics()


"""

print(tabulate(models_details, headers=['Model', 'Score', 'Mean
Error']))


dfreq = DummyClassifier(strategy='most_frequent').fit(Xtrain, ytrain)
ypred = dfreq.predict(Xtest)
print_metrics(dfreq, Xtest, ytest, ypred)

dunif = DummyClassifier(strategy='uniform').fit(Xtrain, ytrain)
ypred = dunif.predict(Xtest)
print_metrics(dunif, Xtest, ytest, ypred)

dstrut = DummyClassifier(strategy='stratified').fit(Xtrain, ytrain)
ypred = dstrut.predict(Xtest)
print_metrics(dstrut, Xtest, ytest, ypred)


print(tabulate(models_details, headers=['Model', 'Score', 'Mean
Error']))
# print(tabulate(np.column_stack((list(X.keys()),
list(model.coef_[0]))), headers=['Feature', 'Score']))
"""
```

DataVisualization.py

```
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate

FILE_PATH = ""

listings = pd.read_csv(f"{FILE_PATH}listings.csv")
reviews = pd.read_csv(f"{FILE_PATH}reviews.csv")

print(f"Total listings: {len(listings.axes[0])}")
print(f"Features in listings: {len(listings.axes[1])}")

print(f"\nTotal reviews: {len(reviews.axes[0])}")
print(f"Features in reviews: {len(reviews.axes[1])}")


def get_nan_values(df, cert=None, drop_NaN=True):
    store_NaN = {}
    if cert != None:
        store_NaN[cert] = [df[cert].isna().sum()]
    else:
        for key in df.keys():
            store_NaN[key] = [df[key].isna().sum()]
    store_NaN = pd.DataFrame(store_NaN)

    if drop_NaN:
        store_NaN = store_NaN[store_NaN != 0]
        store_NaN = store_NaN.T.dropna()
    else:
        store_NaN = store_NaN.T
    return store_NaN


df = get_nan_values(listings, drop_NaN=True)
df = df.sort_values(by=[0], ascending=False)

col = []
for val in df[0]:
    if val < 1000:
        col.append('green')
    elif 1000 < val < 3000:
        col.append('blue')
    elif 3000 < val < 4500:
        col.append('orange')
    else:
        col.append('red')

fig = plt.figure(figsize=(20, 10))
plt.xticks(rotation=90, fontsize=14)
plt.yticks(fontsize=14)
plt.bar(list(df.T.keys()), df[0], color=col)
plt.show();

df = get_nan_values(reviews, drop_NaN=False)
df = df.sort_values(by=[0], ascending=False)

col = []
for val in df[0]:
    if val < 1000:
        col.append('green')
    elif 1000 < val < 3000:
        col.append('blue')
    elif 3000 < val < 4500:
        col.append('orange')
    else:
        col.append('red')

fig = plt.figure(figsize=(20, 10))
plt.xticks(rotation=90)
plt.bar(list(df.T.keys()), df[0], color=col, width=0.1)
plt.show();

host_about_NaN = get_nan_values(listings, cert='host_about')
print(host_about_NaN.T)

print(listings.host_response_time.value_counts())

listings['host_response_rate'] =
listings["host_response_rate"].str.replace("%", "")
listings['host_response_rate'] =
pd.to_numeric(listings['host_response_rate'])
listings['host_acceptance_rate'] =
listings["host_acceptance_rate"].str.replace("%", "")
listings['host_acceptance_rate'] =
pd.to_numeric(listings['host_acceptance_rate'])

keys = ['host_response_rate', 'host_acceptance_rate']
```

```
tabular_format = []

for key in keys:
    tabular_format.append([key, listings[key].min(),
listings[key].max(), listings[key].mean(), listings[key].median()])

print(tabulate(tabular_format, headers=['Name', 'Min', 'Max', 'Mean',
'Median']))

keys = ['review_scores_rating', 'review_scores_accuracy',
'review_scores_cleanliness',
        'review_scores_checkin', 'review_scores_communication',
        'review_scores_location', 'review_scores_value']

tabular_format = []

for key in keys:
    tabular_format.append([key, listings[key].min(),
listings[key].max(), listings[key].mean(), listings[key].median()])

print(tabulate(tabular_format, headers=['Name', 'Min', 'Max', 'Mean',
'Median']))


import matplotlib.pyplot as plt
import pandas as pd
listings = pd.read_csv("final_train_data.csv")
y = listings[[
'review_scores_rating',
'review_scores_accuracy',
'review_scores_cleanliness',
'review_scores_checkin',
'review_scores_communication',
'review_scores_location',
'review_scores_value'
]]
index = [
'rating',
'accuracy',
'cleanliness',
'checkin',
'communication',
'location',
'value'
]
```

```
# plt.figure()
plt.rcParams.update({'font.size': 12}) # must set in top
# fig = plt.figure(figsize=(20, 10))
df = pd.DataFrame({'0': [1052, 1058, 1042, 1593, 1620, 1080, 1094],
                   '1': [1125, 1108, 1128, 1557, 1530, 1071, 1030],
                   '2': [973, 984, 980, None, None, 999, 1026]},
index=index)
ax = df.plot.bar(rot=90, title='Review Score Values after binning')
ax.set_ylabel("value counts")


import pandas as pd
from ListingsPreprocessing import ListingsPreprocessing

listings_data = ListingsPreprocessing(auto_process=False, dropNaN=True)

ENCODE_COLUMNS = [
    'host_is_superhost',
    'host_identity_verified', 'instant_bookable',
]


listings_data.label_encode(columns=ENCODE_COLUMNS)
listings_data.count_vars('amenities', drop_column=True)

listings_data.txt_to_numeric('$', ',', column='price')
listings_data.txt_to_numeric('%', column='host_response_rate')
listings_data.txt_to_numeric('%', column='host_acceptance_rate')

training_data = pd.read_csv('final_train_data.csv')

new_store = pd.DataFrame([])
new_store['accommodates_not_normalized'] =
listings_data.listings['accommodates']
new_store['accommodates'] = training_data['accommodates']

new_store['price_not_normalized'] = listings_data.listings['price']
new_store['price'] = training_data['price']

new_store['price_not_normalized'] = listings_data.listings['price']
new_store['price'] = training_data['price']

new_store['host_response_rate_not_normalized'] =
listings_data.listings['host_response_rate']
new_store['host_response_rate'] = training_data['host_response_rate']
```

```
new_store['host_acceptance_rate_not_normalized'] =
listings_data.listings['host_acceptance_rate']
new_store['host_acceptance_rate'] =
training_data['host_acceptance_rate']

new_store['review_scores_rating'] =
listings_data.listings['review_scores_rating']

import matplotlib.pyplot as plt

col = ['host_acceptance_rate', 'price', 'host_response_rate']
i = 1

fig = plt.figure()
plt.scatter(new_store[col[i]], new_store.review_scores_rating)
name = " ".join(col[i].split('_')).title()
plt.xlabel(name)
plt.ylabel('Review Scores Rating')
plt.title('Normalised Data')
plt.savefig(f'{col[i]}.png', dpi=300)
plt.close(fig)

fig = plt.figure()
plt.scatter(new_store[f"{col[i]}_not_normalized"],
new_store.review_scores_rating)
plt.xlabel(name)
plt.ylabel('Review Scores Rating')
plt.title('Not Normalised Data')
plt.savefig(f'{col[i]}_not_normalised.png', dpi=300)
plt.close(fig)

import seaborn as sns

# START 3D PLOT
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(new_store.price, new_store.maximum_nights,
new_store.review_scores_rating_class, color='blue')
ax.set_xlabel('price')
ax.set_ylabel('maximum')
ax.set_zlabel('rating')
```

```
import pandas as pd
vectors = pd.read_csv('vectors_tfidf_25.csv')
new_store['tfidf_would'] = training_data['tfidf_would']
fig = plt.figure()
plt.scatter(new_store['tfidf_would'], new_store.review_scores_rating)
plt.show()


listings_data = ListingsPreprocessing(auto_process=True)


speed = [0.1, 17.5, 40, 48, 52, 69, 88]
lifespan = [2, 8, 70, 1.5, 25, 12, 28]
index = ['snail', 'pig', 'elephant',
         'rabbit', 'giraffe', 'coyote', 'horse']

import pandas as pd
acc1 = [0.349206,
       0.346032,
       0.369841,
       0.515873,
       0.51746,
       0.322222,
       0.333333,]

acc2 = [0.347619,
        0.301587,
        0.350794,
        0.487302,
        0.468254,
        0.322222,
        0.328571]

term = ['Rating', 'Accuracy', 'Cleanliness', 'Checkin',
        'Communication', 'Location', 'Value']

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
df = pd.DataFrame({'most_frequent': acc1,
                   'uniform': acc2}, index=term)
ax = df.plot.bar(rot=15, fontsize=15)
ax.legend(fontsize=15)
ax.set_ylabel('Accuracy', fontsize=15)
```

```
plt.title("Dummy Classifier Comparison")
plt.legend(loc='upper left', fontsize=15)
plt.grid(False)
plt.show();
```

Question2.py

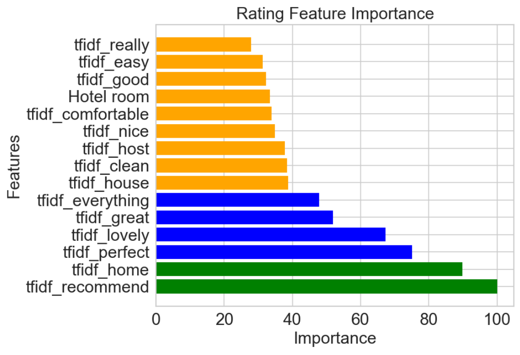
```python
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def binary_step(x, thresh=10):
  return np.where(x<thresh, 0, 1)


def decision_boundary(model):
    b = model.intercept_[0]
    w1, w2 = model.coef_[0]

    c = -b / w2
    m = -w1 / w2

    xd = np.linspace(X.x_1.min(), X.x_1.max())
    yd = m * xd + c

    return xd, yd


size = 250
# NO LINEAR CORELATION
X = pd.DataFrame()
X['x_1'] = np.random.rand(size, )
X['x_2'] = np.random.randint(0, 60, size=size)


y = binary_step(10 + np.random.normal(0.0,1.0,size))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)

model_no_linear_corr = LogisticRegression().fit(X_train, y_train)
ypred = model_no_linear_corr.predict(X)
```

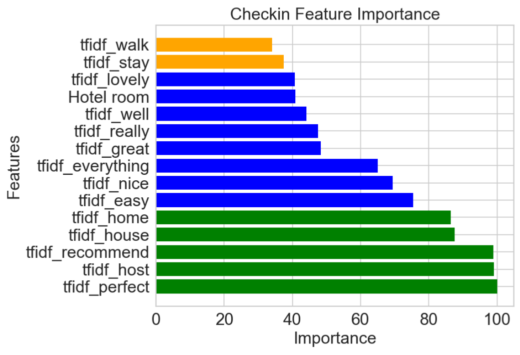
```
print("Logistic Regression - No Colinearity")
print(f"Model Score: {model_no_linear_corr.score(X, y)}")
print(confusion_matrix(y, model_no_linear_corr.predict(X)))
print(classification_report(y, model_no_linear_corr.predict(X)))


pred_actual_logi = pd.DataFrame({'x_1': X['x_1'],
                'x_2': X['x_2'],
              'Ground Truth': y,
              'Prediction': ypred})

# sns.scatterplot(x=X.x_2, y=X.x_1, style=y, hue=y, palette='deep')


xd, yd = decision_boundary(model_no_linear_corr)
for idx, val in enumerate(yd):
    if val < 0:
        yd[idx] = 0
    elif val > 50:
        yd[idx] = 50

sns.scatterplot(x=X.x_1, y=X.x_2, style=ypred, hue=ypred,
palette='deep')
plt.plot(xd, yd)


# DATA IMBALANCE
X = pd.DataFrame()
size = 2000
X['x_1'] = np.arange(0,2,0.05)

y = 10 * X.x_1 + np.random.normal(0.0,1.0,X.size)

for idx, val in enumerate(y):
    if val < 2:
        y[idx] = int(0)
    else:
        y[idx] = int(1)
y = y.astype('int')

model_data_imbalance = LogisticRegression().fit(X, y)

print("Logistic Regression - Data Imbalance")
print(f"Model Score: {model_data_imbalance.score(X, y)}")
print(confusion_matrix(y, model_data_imbalance.predict(X)))
print(classification_report(y, model_data_imbalance.predict(X)))

sns.scatterplot(x=X.x_1, y=y, style=y, hue=y, palette='deep')
plt.legend(loc='lower right')
plt.ylabel('Target')
plt.xlabel('Feature')

ypred = model_data_imbalance.predict(X)
sns.scatterplot(x=X.x_1, y=ypred, style=ypred, hue=ypred,
palette='deep')
plt.legend(loc='lower right')
ypred = model_data_imbalance.predict(X)
plt.ylabel('Predicted')
plt.xlabel('Feature')

pred_data_imbalance_logi = pd.DataFrame({'x_1': X['x_1'],
              'Ground Truth': y,
              'Prediction': model_data_imbalance.predict(X)})


# DUMMY CLASSIFIER
model_dummy = DummyClassifier(strategy='most_frequent').fit(X, y)

print("Dummy Classifier - Data Imbalance")
print(f"Model Score: {model_dummy.score(X, y)}")
print(confusion_matrix(y, model_dummy.predict(X)))
print(classification_report(y, model_dummy.predict(X)))

pred_data_imbalance_dummy = pd.DataFrame({'x_1': X['x_1'],
              'Ground Truth': y,
              'Prediction': model_dummy.predict(X)})


# HOLD OUT METHOD
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd

X = np.arange(0,1,0.05).reshape(-1, 1)
y = 10 * X + np.random.normal(0.0,1.0,X.size).reshape(-1, 1)
```

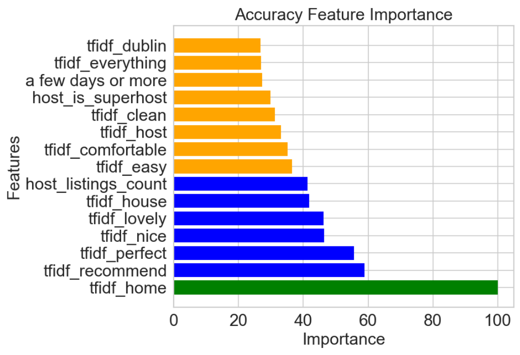
```
intercept = []
slope = []
mean_error = []

for i in range(5):
    Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.2)

    model = LinearRegression().fit(Xtrain, ytrain)
    ypred = model.predict(Xtest)

    intercept.append(model.intercept_[0])
    slope.append(model.coef_[0][0])
    mean_error.append(mean_squared_error(ytest, ypred))

    print('Intercept: {:.2f}\nSlope: {:.2f}\nSquared Error:
{:.2f}'.format(model.intercept_[0],

model.coef_[0][0],

mean_squared_error(ytest, ypred)))
    print('\n\n')

    y_vals = model.intercept_ + X * model.coef_
    plt.plot(X, y_vals, label='{:.2f}'.format(mean_squared_error(ytest,
ypred)))

vals = pd.DataFrame({
    'intercept': intercept,
    'slope': slope,
    'mean_error': mean_error})

plt.scatter(X, y, c='black')
plt.xlabel('Input X')
plt.ylabel('Target y')
plt.legend(title="MSE", fancybox=True)
plt.show();

import numpy as np
arr = np.arange(3.0, 5.5, 0.05)


# LAGGED OUTPUT
```

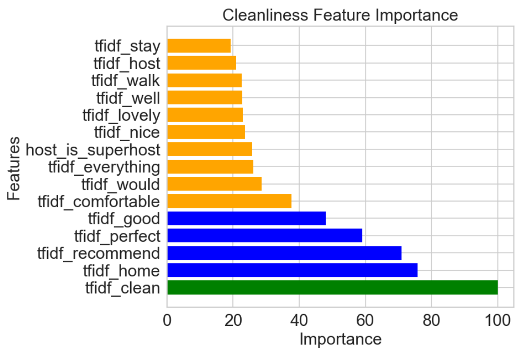
```
import pandas as pd
data = pd.DataFrame({
    'Date': ['2020-03-18',
             '2020-04-18',
             '2020-05-18',
             '2020-06-18',
             '2020-07-18',
             '2020-08-18',
             ],
    'Covid Cases': [ 100,
                    200,
                    400,
                    800,
                    1600,
                    3200
        ],
    'lag_1': [ None,
              100,
              200,
              400,
              800,
              1600,
        ],
    'lag_2': [None,
              None,
              100,
              200,
              400,
              800,
        ],
    })
```

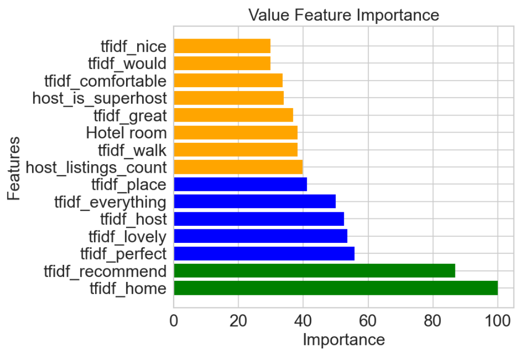
Feature Importance for the Review Scores

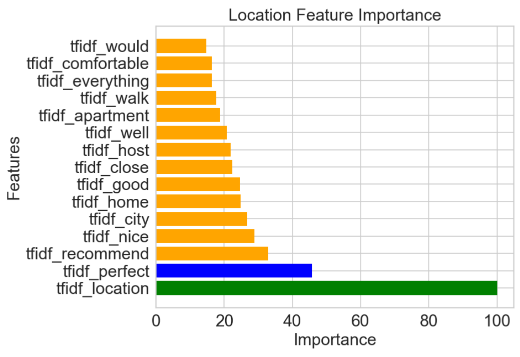


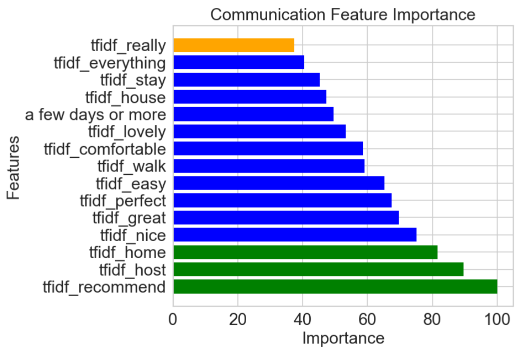




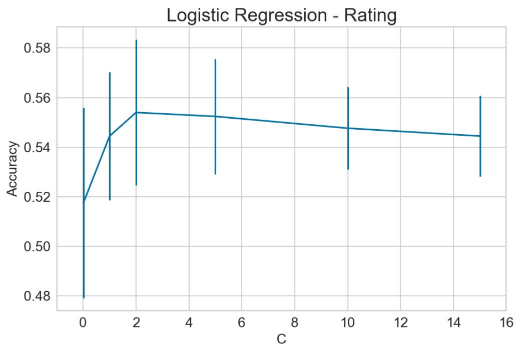


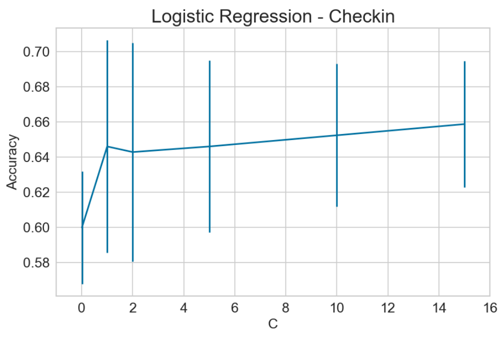


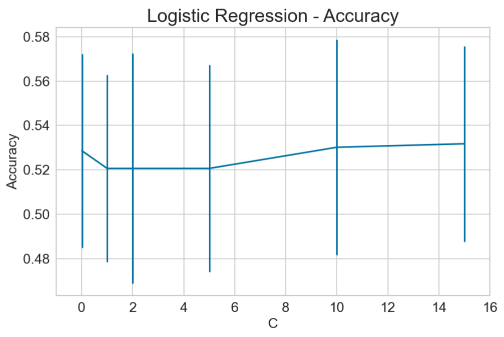


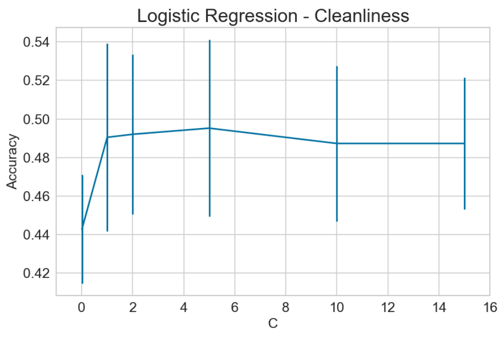


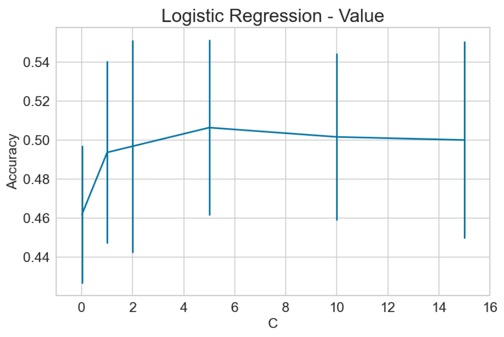
Cross Validation for Logistic Regression

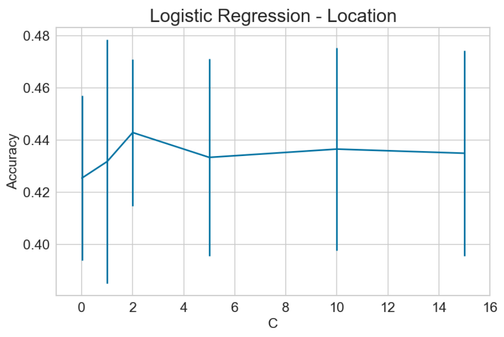


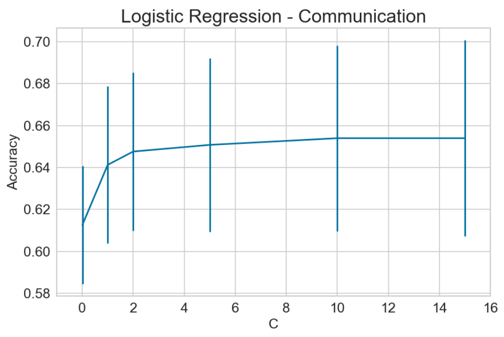






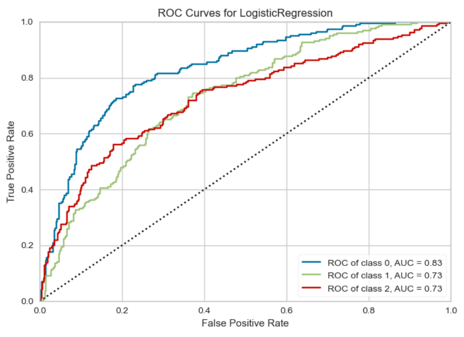




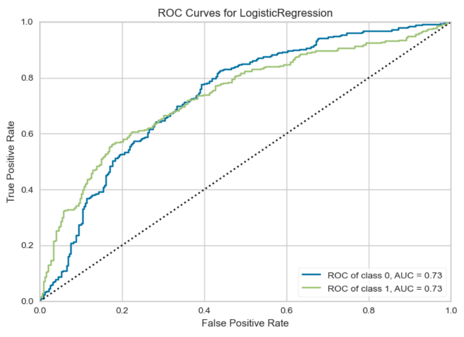


ROC Curve for Logistic Regression

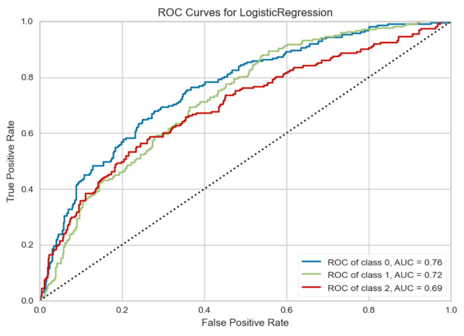
Rating



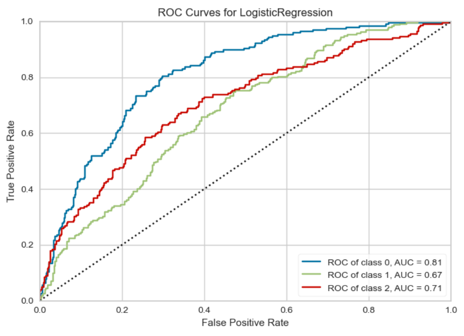
Checkin



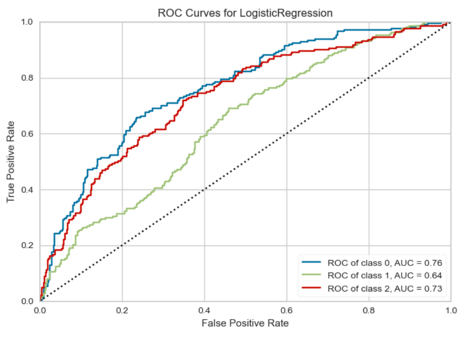
Accuracy



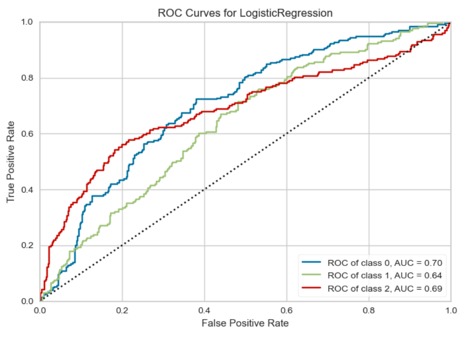
Cleanliness



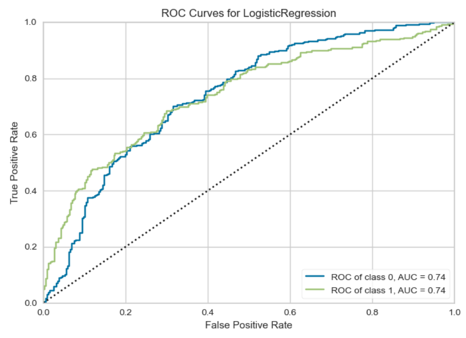
Value



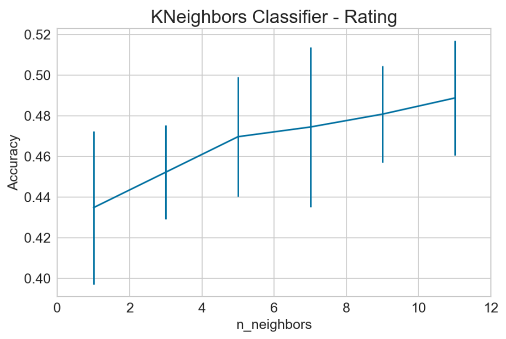
Location

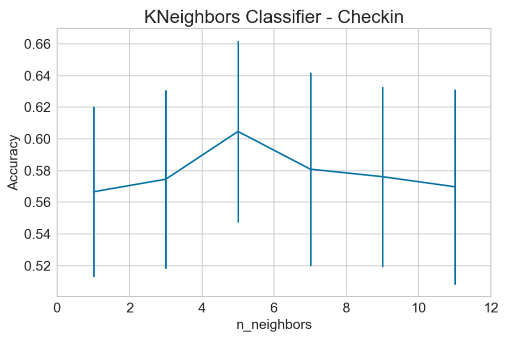


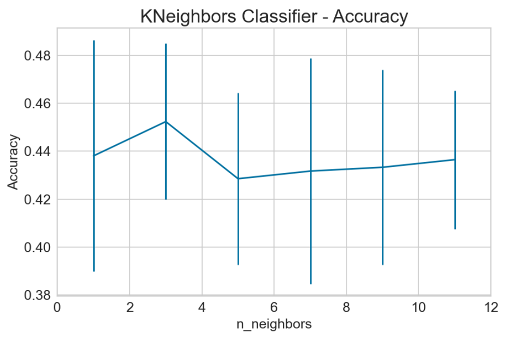
Communication

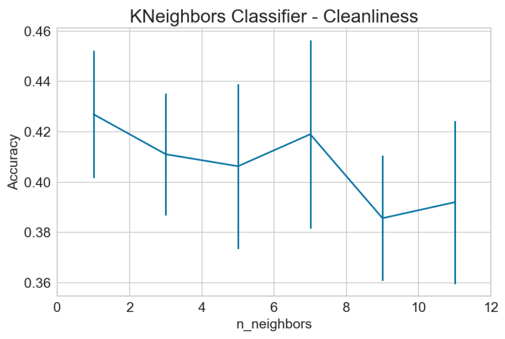


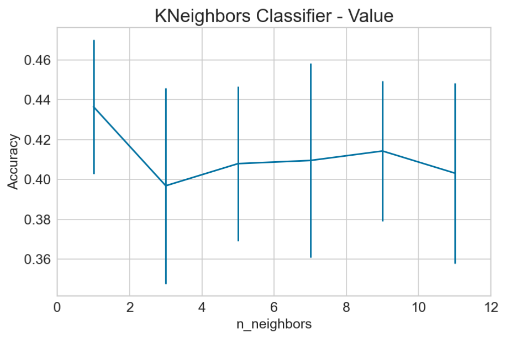
Cross Validation for K-Nearest Neighbors

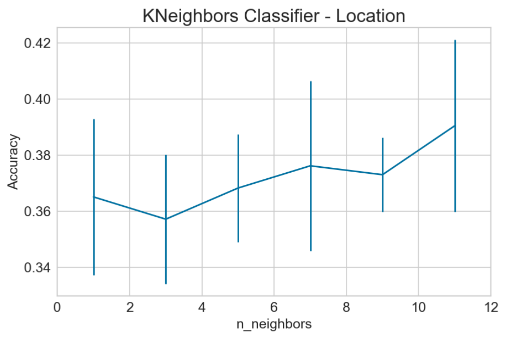


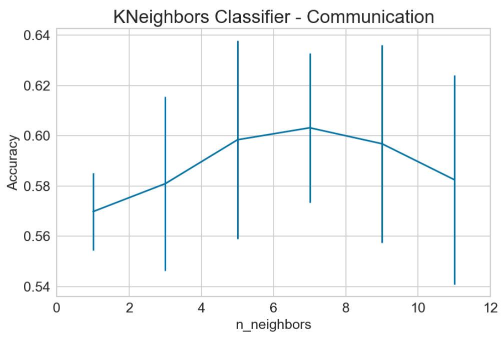






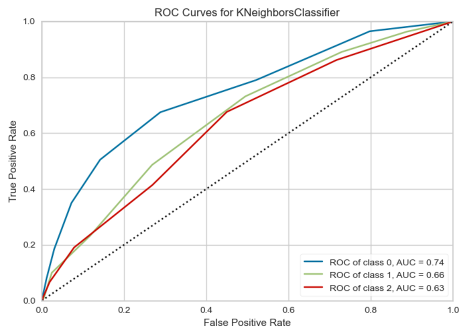




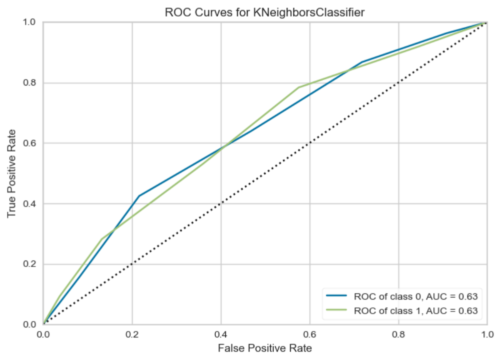


ROC Curve for K-Nearest Neighbors

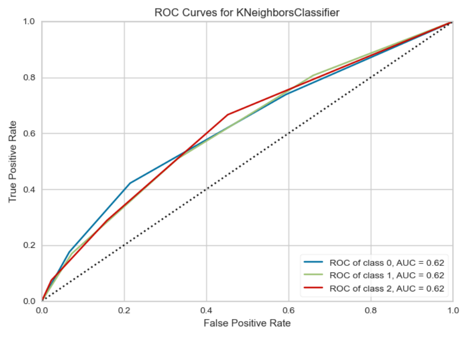
Rating



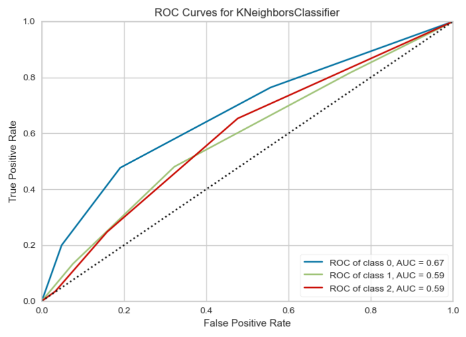
Checkin



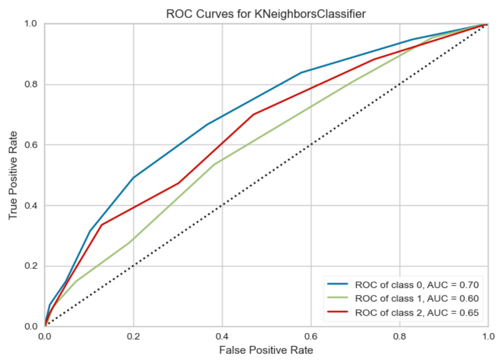
Accuracy



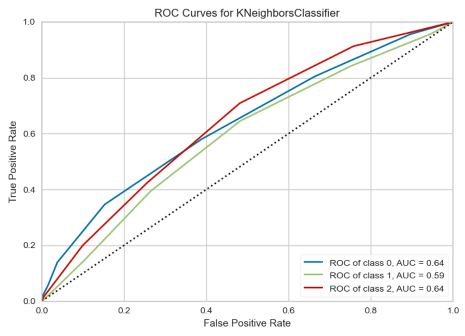
Cleanliness



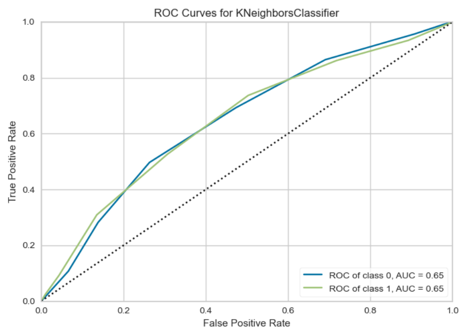
Value



Location



Communication


ROC Curves for KNeighborsClassifier
True Positive Rate
False Positive Rate
ROC of class 0, AUC = 0.65
ROC of class 1, AUC = 0.65