Exercise 3

1. Search about major Kernel functions used in SVM. Why do we exploit kernels in SVM? Can we comment on where each specific kernel can be used? Consider the case where RBF is employed.

A:

Kernel Function is a method used to take data as input and transform into the required form of processing data. “Kernel” is used due to set of mathematical functions used in Support Vector Machine provides the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transformed to a linear equation in a higher number of dimension spaces. Basically, it returns the inner product between two points in a standard feature dimension.

Standard Kernel Function Equation:

$$K(\bar{x}) = 1, if ||\bar{x}|| <= 1$$
$$K(\bar{x}) = 0, Otherwise$$

Gaussian Kernel: It is used to perform transformation, when there is no prior knowledge about data.

$$K(x, y) = e^{-}(\frac{||x-y||^2}{2\sigma^2})$$

Gaussian Kernel Radial Basis Function (RBF): Same as above kernel function, adding radial basis method to improve the transformation.

$$K(x, y) = e^{-}(\gamma||x - y||^2)$$

Sigmoid Kernel: this function is equivalent to a two-layer, perceptron model of neural network, which is used as activation function for artificial neurons.

$$K(x, y) = tanh(\gamma.x^T y + r)$$

Polynomial Kernel: It represents the similarity of vectors in training set of data in a feature space over polynomials of the original variables used in kernel.

$$K(x, y) = tanh(\gamma.x^T y + r)^d, \gamma > 0$$

Linear Kernel: is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a large number of Features in a particular Data Set. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature.

2. In this section SVM model is used for classification on the mobile price dataset.

This dataset has 21 features and 2000 entries. The meanings of the features are given below.

battery_power: Total energy a battery can store in one time measured in mAh

blue: Has Bluetooth or not

clock_speed: speed at which microprocessor executes instructions

dual_sim: Has dual sim support or not

fc: Front Camera mega pixels

four_g: Has 4G or not

int_memory: Internal Memory in Gigabytes

m_dep: Mobile Depth in cm

mobile_wt: Weight of mobile phone

n_cores: Number of cores of processor

pc: Primary Camera mega pixels

px_height: Pixel Resolution Height

px_width: Pixel Resolution Width

ram: Random Access Memory in Mega Bytes

sc_h: Screen Height of mobile in cm

sc_w: Screen Width of mobile in cm

talk_time: longest time that a single battery charge will last when you are

three_g: Has 3G or not

touch_screen: Has touch screen or not

wifi: Has wifi or not

price_range: This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).

I run SVM with default settings on this dataset and the accuracy of the model is:

```
train accuracy: 0.97125
test accuracy: 0.8375
```

3. SVM with different kernel results:

I first used the sigmoid kernel instead of the default kernel(rbf) and the accuracy dropped severely.

```
train accuracy: 0.228125
test accuracy: 0.2075
```

With the polynomial kernel, the accuracy increased, using the default degree of poly=3

```
train accuracy: 0.99375
test accuracy: 0.8675
```

With increasing the degree of the polynomial kernel to 10, accuracy in the train model increased a little but the accuracy of the test model dropped severely, which can mean be interpreted as overfitting of the model

```
train accuracy: 0.9975
test accuracy: 0.6875
```

By using the linear kernel, the accuracy of the test and train dataset increased:

```
train accuracy: 0.959375
test accuracy: 0.93
```

By using rbf kernel and gamma Kernel coefficient set on scale:

```
train accuracy: 0.97125
test accuracy: 0.8375
```

By using rbf kernel and gamma Kernel coefficient set on auto:

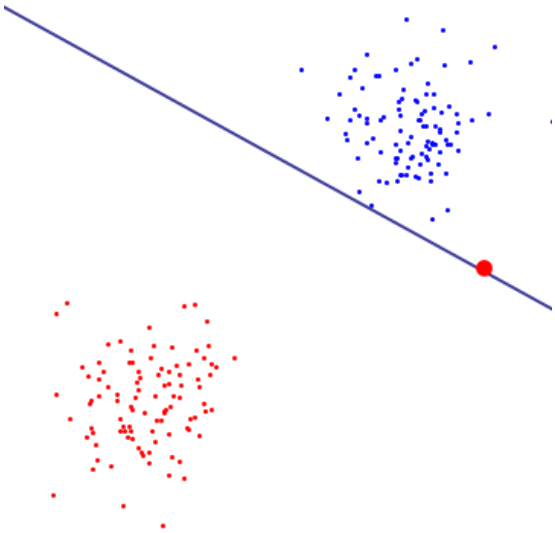
```
train accuracy: 0.91
test accuracy: 0.84
```

4.

In this section, I'm going to give a brief explanation of soft and hard margins for SVM and then we'll see the implementation results.

I would expect soft-margin SVM to be better even when training dataset is linearly separable. The reason is that in a hard-margin SVM, a single outlier can determine the boundary, which makes the classifier overly sensitive to noise in the data.

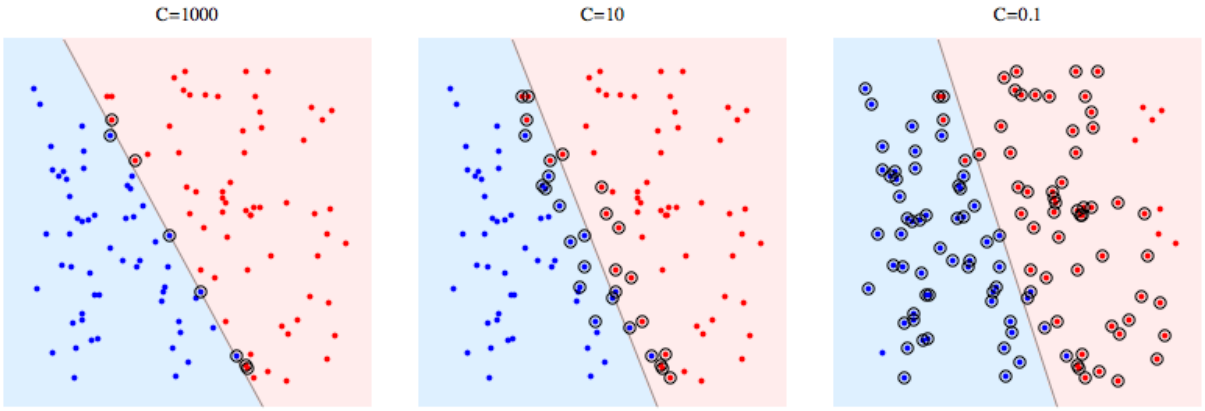
In the diagram below, a single red outlier essentially determines the boundary, which is the hallmark of overfitting



To get a sense of what soft-margin SVM is doing, it's better to look at it in the dual formulation, where you can see that it has the same margin-maximizing objective (margin could be negative) as the hard-margin SVM, but with an additional constraint that each Lagrange multiplier associated with support vector is bounded by C. Essentially this bounds the influence of any single point on the decision boundary, for derivation, see Proposition 6.12 in Cristianini/Shaw-Taylor's "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods".

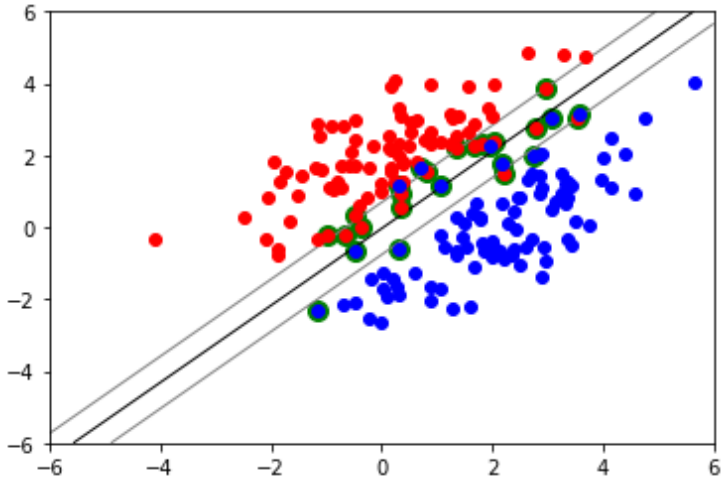
The result is that soft-margin SVM could choose decision boundary that has non-zero training error even if dataset is linearly separable, and is less likely to overfit.

Here's an example using libSVM on a synthetic problem. Circled points show support vectors. You can see that decreasing C causes classifier to sacrifice linear separability in order to gain stability, in a sense that influence of any single datapoint is now bounded by C.

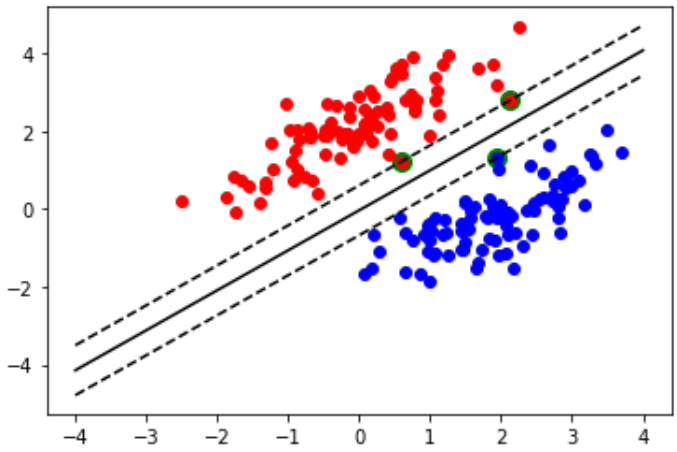


For hard margin SVM, support vectors are the points which are "on the margin". In the picture above, C=1000 is pretty close to hard-margin SVM, and you can see the circled points are the ones that will touch the margin (margin is almost 0 in that picture, so it's essentially the same as the separating hyperplane).

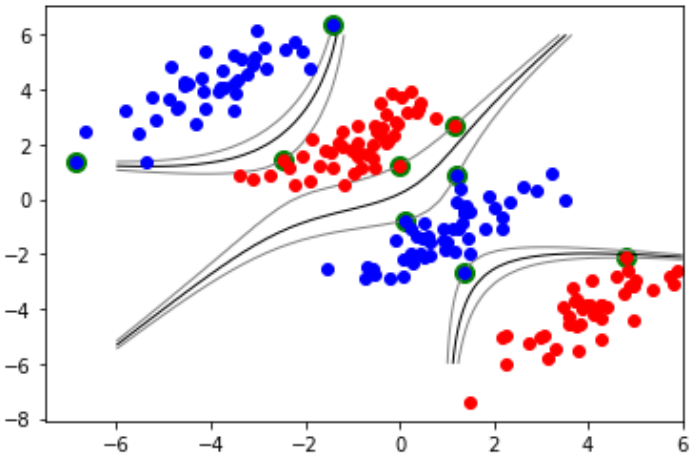
For implementation, with the help of a video in pythonprogramming.net about soft margin, the results for SVM with a soft margin is as shown in this plot:



For a hard margin model with a linear kernel:



And for a hard margin nonlinear:



By changing the parameter C and increasing it, we soften the margin and as a result I got better accuracy from my model with a linear kernel.

5. Let's investigate Feature Engineering.

Feature engineering is the process of transforming raw data to provide your algorithms with the most predictive inputs possible. Before seeing data, an

algorithm knows nothing of the problem at hand. Humans, though, can inject prior knowledge into the equation. If, for example, we know from experience that changes in user activity are more predictive than raw activity itself, we can introduce this information. Given the infinite number of potential features, it's often not computationally feasible for even the most sophisticated algorithms to do this on their own.

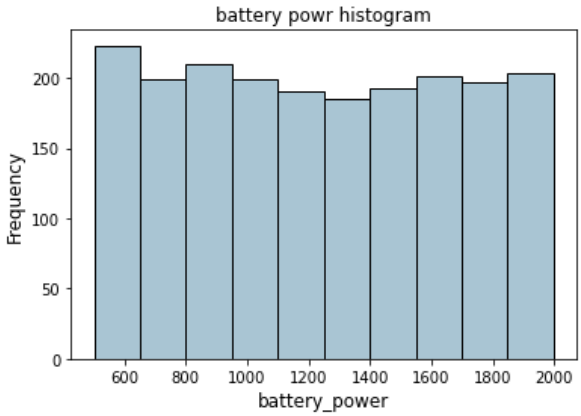
a)

Binning is a technique that accomplishes exactly what it sounds like. It will take a column with continuous numbers and place the numbers in "bins" based on ranges that we determine. This will give us a new categorical variable feature.

First, I tried two approaches with Fixed-Width Binning

Just like the name indicates, in fixed-width binning, we have specific fixed widths for each of the bins which are usually pre-defined by the user analyzing the data. Each bin has a pre-fixed range of values which should be assigned to that bin on the basis of some domain knowledge, rules or constraints. Binning based on rounding is one of the ways, where you can use the rounding operation which we discussed earlier to bin raw values.

This histogram plot shows the frequency of bins



With this approach, I binned the battery power feature once into two and once into 15 bins.

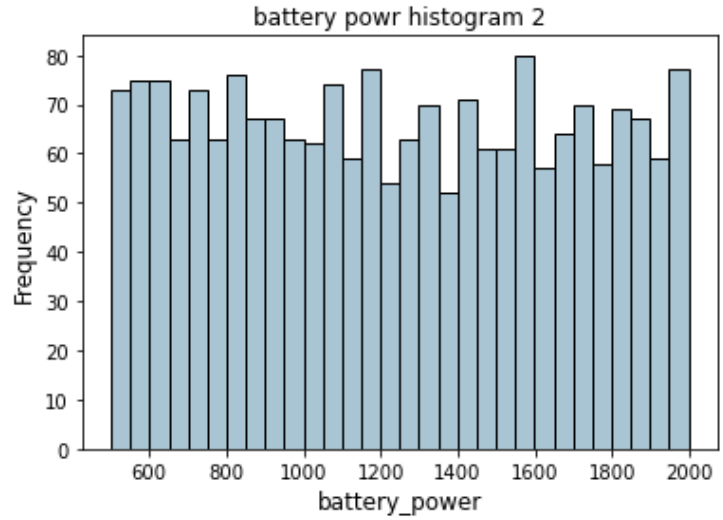
Another approach I took for binning is Adaptive Binning

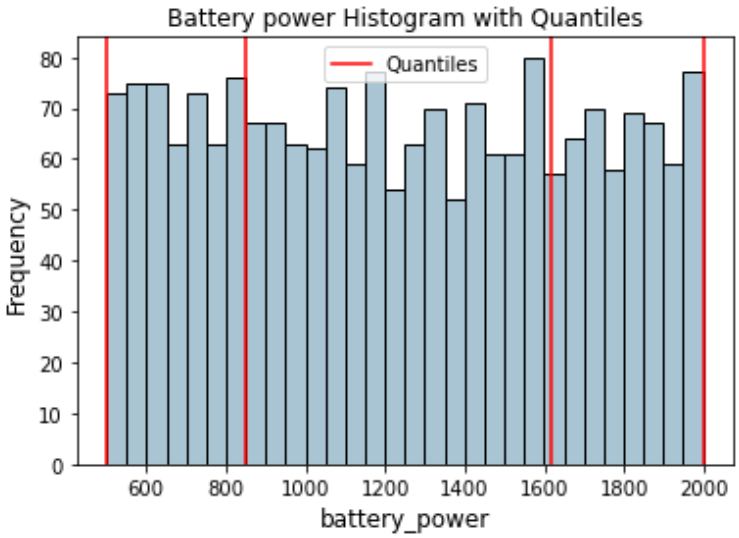
The drawback in using fixed-width binning is that due to us manually deciding the bin ranges, we can end up with irregular bins which are not uniform based on the number of data points or values which fall in each bin. Some of the bins might be densely populated and some of them might be sparsely populated or even empty! Adaptive binning is a safer strategy in these scenarios where we let the data speak for itself!

Quantile based binning is a good strategy to use for adaptive binning. Quantiles are specific values or cut-points which help in partitioning the continuous valued distribution of a specific numeric field into discrete contiguous bins or intervals.

Let’s take a look at the plot for this part:

In the first plot we can see same width bins, and in the second figure the quantiles are seen.





I binned the battery power into 3 bins with different sizes by this approach.

b)

to Convert categorical data to numerical Data we take two steps:

for example:

A “color” variable with the values: “red“, “green” and “blue“.

A “place” variable with the values: “first”, “second” and “third“.

Integer Encoding: As a first step, each unique category value is assigned an integer value. For example, “red” is 1, “green” is 2, and “blue” is 3.

This is called a label encoding or an integer encoding and is easily reversible.

For some variables, this may be enough.

The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.

For example, ordinal variables like the “place” example above would be a good example where a label encoding would be sufficient.

One-Hot Encoding: for categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

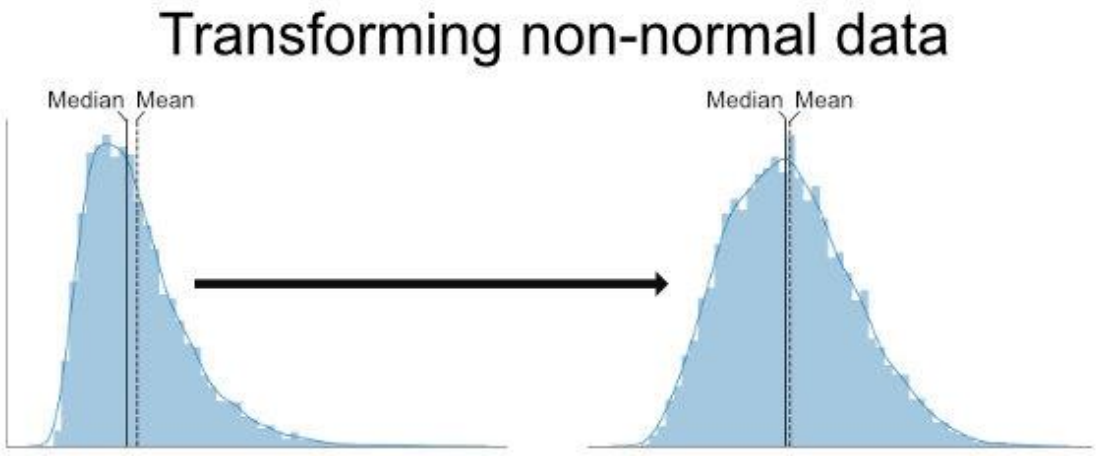
In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors. The binary variables are often called “dummy variables” in other fields, such as statistics.

In this section I used dummy variables for the categorical data columns 'blue','dual_sim','four_g','three_g','touch_screen','wifi'.

The data in these columns are integers so for I converted it into strings first to make dummies. This got me 39 features when the initial df had 21.

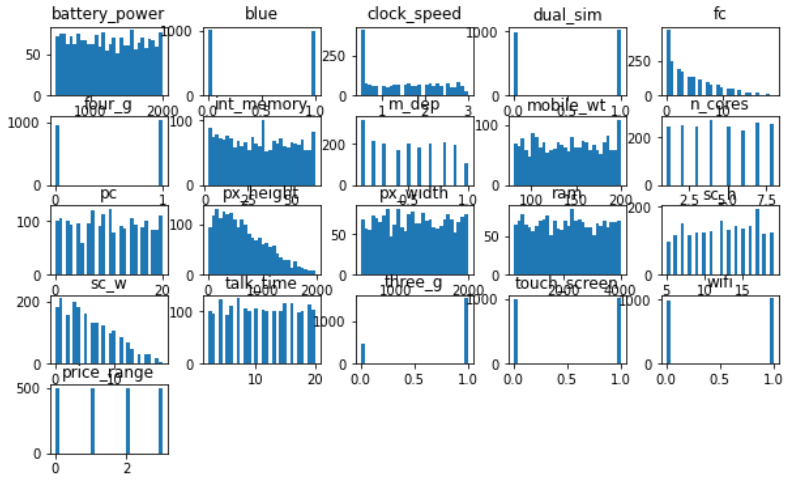
c)

One way to deal with non-normal data is to transform your data. Some ways to do this is Box-Cox, square root, and log transformation.



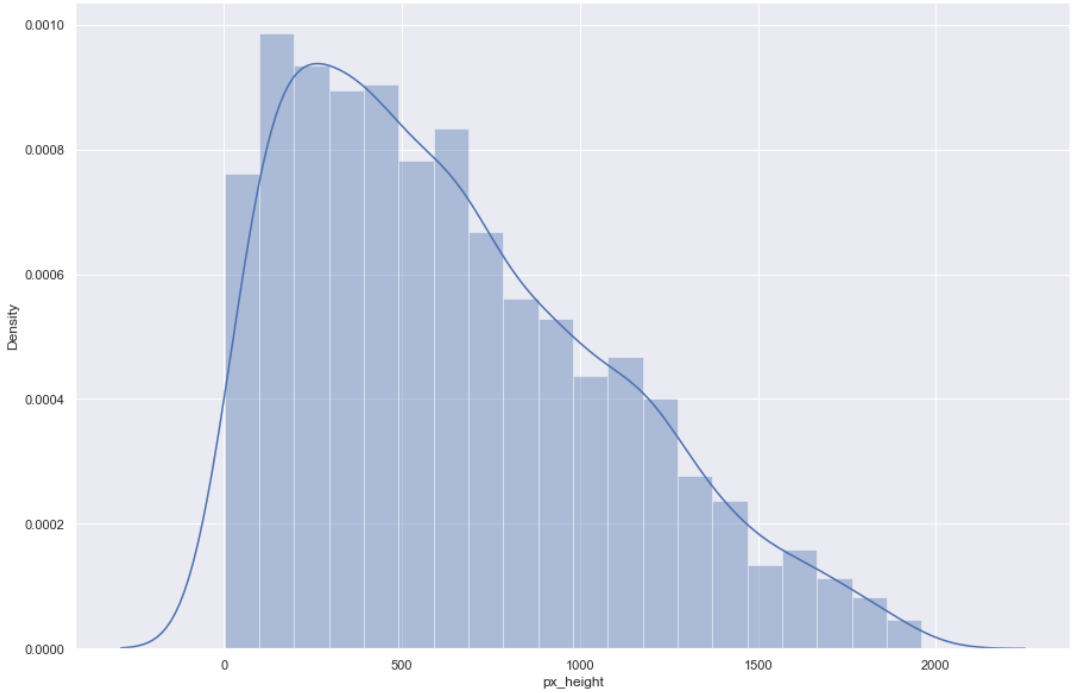
Log transformation is a data transformation method in which it replaces each variable x with a log(x). The choice of the logarithm base is usually left up to the analyst and it would depend on the purposes of statistical modeling.

First let's take a look at the features

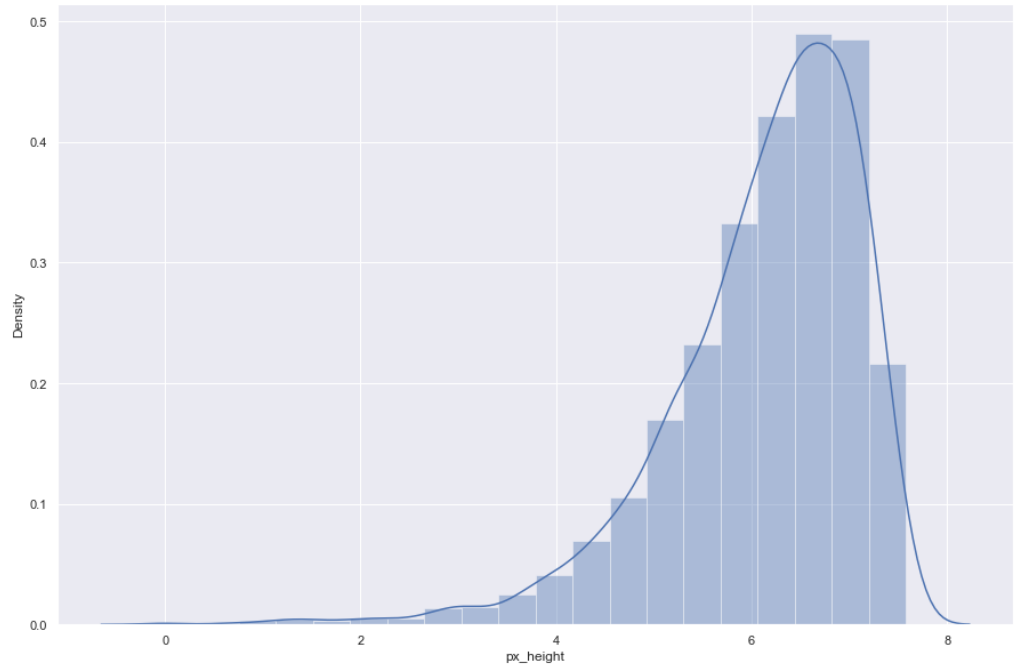


To get a better understanding of how the transformations change the columns, I applied both log and square root transform methods to the feature px_height

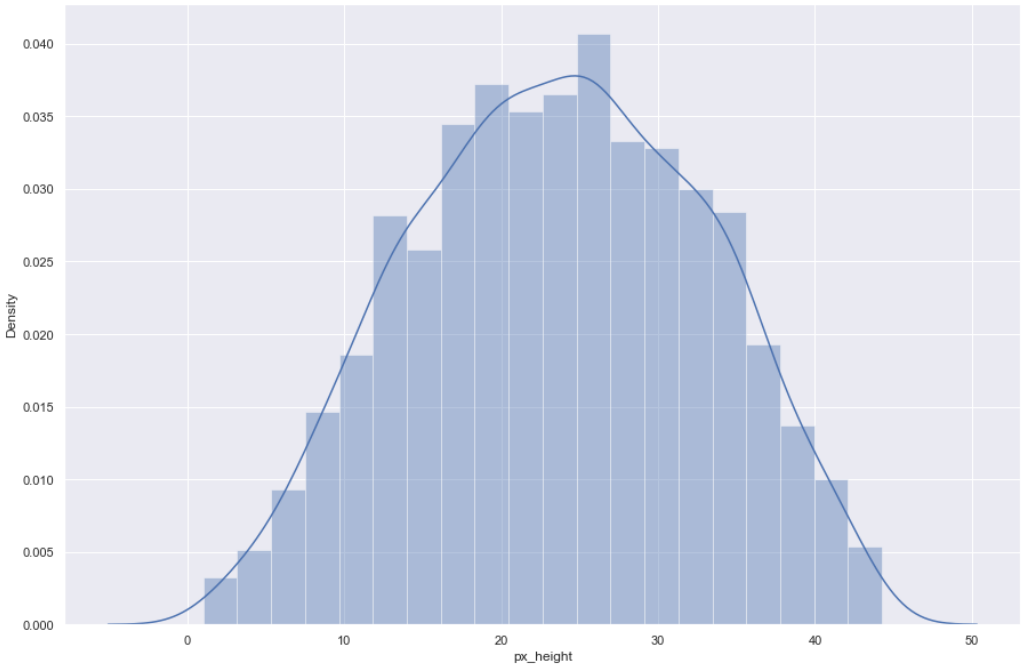
In this figure, we can see the distribution of px_height without any transformation:



Here, we can see the data after performing the log transformation on this column:



And we can see the data after performing the log transformation on this column:



Seems like the sqrt transform was more successful in dealing with this sort of non-normal data.

d)

In this section, I added a surface feature called sc_surface.

6. In this section, I use a SVM classification model with respect to the change made in each section of last part, and finally this model is run with all the feature changes made

By running SVM with default parameters on our data frame without any feature engineering changes, this is the result I've achieved:

```
train accuracy: 0.9748110831234257
test accuracy: 0.8442211055276382
```

The results with respect to the first binning I did is as below. Both test and train accuracy dropped a bit, which could be interpreted that the binning didn't make sense to improve performance.

```
train accuracy: 0.9502518891687658
test accuracy: 0.8115577889447236
```

The results of SVM with respect to the one hot encoding section are as shown below, which is quite similar to the one above:

```
train accuracy: 0.9521410579345088
test accuracy: 0.8190954773869347
```

By running the SVM model with respect to the square root transformation, the results got even better than the original which was expected as the data was closer to a normal distribution:

```
train accuracy: 0.9760705289672544
test accuracy: 0.8517587939698492
```

The model's result with respect to the added surface feature:

```
train accuracy: 0.9729219143576826
test accuracy: 0.8366834170854272
```

And finally, the SVM model with respect to all the feature engineering changes I made is:

```
train accuracy: 0.9250629722921915
test accuracy: 0.7638190954773869
```

7. Investigate various decision tree implementation algorithms. What is the main difference between them?

A:

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Some algorithms used in Decision Trees:

ID3 → (extension of D3)

C4.5 → (successor of ID3)

CART → (Classification and Regression Tree)
CHAID → (Chi-square automatic interaction detection Performs multi-level splits when

computing classification trees)
MARS → (multivariate adaptive regression splines)

Here in this table, we see a comparison of some of the Decision tree algorithms:

Methods	CART	C4.5	CHAID
Measure used to select input variable	Gini index; Twoing criteria	Entropy info-gain	Chi-square
Pruning	Pre-pruning using a single-pass algorithm	Pre-pruning using a single-pass algorithm	Pre-pruning using Chi-square test for independence
Dependent variable	Categorical/ Continuous	Categorical/ Continuous	Categorical
Input variables	Categorical/ Continuous	Categorical/ Continuous	Categorical/ Continuous
Split at each node	Binary; Split on linear combinations	Multiple	Multiple

For better understanding the main differences of these implementation methods, Decision Tree implementations differ primarily along these axes:

-The splitting criterion (i.e., how "variance" is calculated)

-Whether it builds models for regression (continuous variables, e.g., a score) as well as classification (discrete variables, e.g., a class label)

-Technique to eliminate/reduce over-fitting

-Whether it can handle incomplete data

8. In this section, I used decision tree for classification on the dataset with all the parameters set to default. The results for this classification:

```
dt train accuracy: 1.0
dt test accuracy: 0.7939698492462312
```

	precision	recall	f1-score
0	0.90	0.90	0.90
1	0.79	0.78	0.78
2	0.72	0.72	0.72
3	0.82	0.83	0.83
accuracy			0.81
macro avg	0.81	0.81	0.81
weighted avg	0.81	0.81	0.81

9. I explored different parameters of decision tree classifier and their effect on the results.

First, I changed the criterion default which was Gini impurity, to entropy and the results for accuracy improved:

```
dt train accuracy: 1.0
dt test accuracy: 0.8241206030150754
```

I also changed the splitter parameter which is the strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split. I changed the default from best to random:

```
dt train accuracy: 1.0
dt test accuracy: 0.8266331658291457
```

I changed the max_depth parameter from none to 10 and the result is as follows:

```
dt train accuracy: 0.9962216624685138
dt test accuracy: 0.8115577889447236
```

From a point on, increasing this parameter works as none.

And lastly, I changed min_samples_leaf parameter which is the minimum number of samples required to be at a leaf node. This parameter was originally set to 1 and I changed it to 10. As a result, the predictions accuracy went up for the test dataset.

```
dt train accuracy: 0.906801007556675
dt test accuracy: 0.8467336683417085
```

10. Look up pruning in Decision trees, why we need to prune decision trees and what benefits do we achieve?

A:

Pruning reduces the size of decision trees by removing parts of the tree that do not provide power to classify instances. Decision trees are the most susceptible out of all the machine learning algorithms to overfitting and effective pruning can reduce this likelihood.

11. For this section I changed the Complexity parameter used for Minimal Cost-Complexity Pruning from the default 0.0 into 0.03 and we can see the accuracy dropped

```
dt train accuracy: 0.7644836272040302
dt test accuracy: 0.7487437185929648
```

When I increased this parameter to 0.1, accuracy dropped more

```
dt train accuracy: 0.6486146095717884
dt test accuracy: 0.635678391959799
```

12. On this dataset I used both models Random Forest vs Decision tree both with default parameters.

Here we have the results for the Decision tree model:

```
dt train accuracy: 1.0
dt test accuracy: 0.8015075376884422
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       108
           1       0.76      0.79      0.77        99
           2       0.67      0.70      0.69        97
           3       0.84      0.81      0.82        94

    accuracy                           0.80       398
   macro avg       0.80      0.80      0.80       398
weighted avg       0.81      0.80      0.80       398
```

And the results for the Random Forest model:

```
rdf train accuracy: 1.0
rdf test accuracy: 0.8894472361809045
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       105
           1       0.88      0.85      0.87       107
           2       0.79      0.84      0.82        95
           3       0.91      0.91      0.91        91

    accuracy                           0.89       398
   macro avg       0.89      0.89      0.89       398
weighted avg       0.89      0.89      0.89       398
```

As it is obviously seen in the results, RF does a better job comparing to DT.

Usually, the decision tree model gives high importance to a particular set of features. But the random forest chooses features randomly during the training process. Therefore, it does not depend highly on any specific set of features. Therefore, the random forest can generalize over the data in a better way. This randomized feature selection makes random forest much more accurate than a decision tree.

13. Why do we still use Decision tree methods when there are so many new methods i.e., deep learning and neutral networks? (Interpretability)

A:

Neural networks are often compared to decision trees because both methods can model data that has nonlinear relationships between variables, and both can handle interactions between variables. However, neural networks have a number of drawbacks compared to decision trees.

Binary categorical input data for neural networks can be handled by using 0/1 (off/on) inputs, but categorical variables with multiple classes (for example, marital status or the state in which a person resides) are awkward to handle. Classifying a result into multiple categories usually is done by setting arbitrary value thresholds for discriminating one category from another. It would be difficult to devise a neural network to classify the location of residence into the 50 U.S. states. Classification trees, on the other hand, handle this type of problem naturally.

Neural networks do not present an easily-understandable model. When looking at a decision tree, it is easy to see that some initial variable divides the data into two categories and then other variables split the resulting child groups. This information is very useful to the researcher who is trying to understand the underlying nature of the data being analyzed.

A neural network is more of a “black box” that delivers results without an explanation of how the results were derived. Thus, it is difficult or impossible to explain how decisions were made based on the output of the network.

If a challenge is made to a decision based on a neural network, it is very difficult to explain and justify to non-technical people how decisions were made. In contrast, a decision tree is easily explained, and the process by which a particular decision “flows” through the decision tree can be readily shown.

It is difficult to incorporate a neural network model into a computer system without using a dedicated "interpreter" for the model. So, if the goal is to produce a program that can be distributed with a built-in predictive model, it is usually necessary to send along some additional module or library just for the neural network interpretation. In contrast, once a decision tree model has been built, it can be converted to if…then…else statements that can be implemented easily in most computer languages without requiring a separate interpreter.

14. We exploit some rules from the decision tree and use them. Search for some other rule induction algorithms i.e., Ripper and IREP, and explain these algorithms.

A:

In general, rule induction algorithms may be categorized as global and local. In global rule induction algorithms, the search space is the set of all attribute values, while in local rule induction algorithms the search space is the set of attribute-value pairs.

first is an example of a global rule induction algorithm called:

LEM1 *(Learning from Examples Module version 1):*

The algorithm LEM1, a component of the data mining system LERS (Learning from Examples using Rough Sets), is based on some rough set definitions [Pawlak, 1982], [Pawlak, 1991], [Pawlak et al., 1995]. Let B be a nonempty subset of the set A of all attributes. Let U denote the set of all cases. The indiscernibility relation IND(B) is a relation on U defined for x, y ∈ U by (x, y) ∈ IND(B) if and only if for both x and y the

values for all attributes from B are identical. The indiscernibility relation IND(B) is an equivalence relation.

Equivalence classes of IND(B) are called elementary sets of B. The family of all B-elementary sets will be denoted $B*$.

For a decision d we say that {d} depends on B if and only if $B* \leq \{d\}*$. A global covering (or relative reduction) of {d} is a subset B of A such that {d} depends on B and B is minimal in A. Thus, global coverings of {d} are computed by comparing partitions $B*$ with $\{d\}*$. On the basis of a global covering rules are computed using the dropping conditions technique [Michalski, 1983]. For a rule of the form C1 & C2 & ... & Cn → D , dropping conditions means scanning the list of all conditions, from the left to the right, with an attempt to drop any condition, checking against the decision table where the simplified rule does not violate consistency of the discriminant description.

LEM2 *(Learning from Examples Module, version 2):*

An idea of blocks of attribute-value pairs is used in the rule induction algorithm LEM2, another component of LERS. The option LEM2 of LERS is most frequently used since—in most cases—it gives better results. LEM2 explores the search space of attribute-value pairs. Its input data file is a lower or upper approximation of a concept (for definitions of lower and upper approximations of a concept see, e.g., [Grzymala-Busse, 1997]), so its input data file is always consistent. In general, LEM2 computes a local covering and then converts it into a rule set. We will quote a few definitions to describe the LEM2 algorithm [Chan and Grzymala-Busse, 1991], [Grzymala-Busse, 1992].

For an attribute-value pair (a, v) = t, a block of t, denoted by [t], is a set of all cases from U such that for attribute a have value v. Let B be a nonempty lower or upper approximation of a concept represented by a decision-value pair (d, w). Set B depends on a set T of attribute-value pairs t = (a, v) if and only if $\emptyset \neq [T] = \cap [t] \subseteq B, (t \in T)$.

Set T is a minimal complex of B if and only if B depends on T and no proper subset T0 of T exists such that B depends on T0 . Let T be a nonempty collection of nonempty sets of attribute-value pairs. Then T is a local covering of B if and only if the following conditions are satisfied:

1) each member T of ĩ is a minimal complex of B,

2) ∪ [T] = B (t∈ĩ), and ĩ is minimal, i.e., T has the smallest possible number of members.

The procedure LEM2 is presented below.

Procedure LEM2
(**input**: a set B,
output: a single local covering $\mathcal{T}$ of set B);
begin
$G := B$;
$\mathcal{T} := \emptyset$;
while $G \neq \emptyset$
begin
$T := \emptyset$;
$T(G) := \{t | [t] \cap G \neq \emptyset\}$;
while $T = \emptyset$ **or** $[T] \not\subseteq B$
begin
select a pair $t \in T(G)$ such that $|[t] \cap G|$ is maximum; if a tie occurs, select a pair $t \in T(G)$ with the smallest cardinality of $[t]$;
if another tie occurs, select first pair;
$T := T \cup \{t\}$;
$G := [t] \cap G$;
$T(G) := \{t | [t] \cap G \neq \emptyset\}$;
$T(G) := T(G) - T$;
end {while}
for each $t \in T$ **do**
if $[T - \{t\}] \subseteq$ B **then** $T := T - \{t\}$;
$\mathcal{T} := \mathcal{T} \cup \{T\}$;
$G := B - \cup_{T \in \mathcal{T}} [T]$;
end {while};
for each $T \in \mathcal{T}$ **do**
if $\bigcup_{S \in \mathcal{T} - \{T\}} [S] = B$ **then** $\mathcal{T} := \mathcal{T} - \{T\}$;
end {procedure}.

For a set X, |X| denotes the cardinality of X. The first step of the algorithm LEM2 is to compute all attribute-value pair blocks.

Obviously, in general, rule sets induced by LEM1 differ from rule sets induced by LEM2 from the same data sets.

AQ

Another rule induction algorithm, developed by R. S. Michalski and his collaborators in the early seventies, is an algorithm called AQ. Many versions of the algorithm have been developed, under different names [Michalski et al., 1986A], [Michalski et al., 1986B].

Let us start by quoting some definitions from [Michalski et al., 1986A], [Michalski et al., 1986B]. Let A be the set of all attributes, A = {A1, A2, ..., Ak}. A seed is a member of the concept, i.e., a positive case. A selector is an expression that associates a variable (attribute or decision) to a value of the variable, e.g., a negation of value, a disjunction of values, etc. A complex is a conjunction of selectors. A partial star G(e|e1) is a set of all complexes describing the seed e = (x1, x2, ..., xk) and not describing a negative case e1 = (y1, y2, ..., yk). Thus, the complexes of G(e|e1) are conjunctions of selectors of the form (Ai, ¬yi), for all i such that xi 6= yi. A star G(e|F) is constructed from all partial stars G(e|ei), for all ei ∈ F, and by conjuncting these partial stars by each other, using absorption law to eliminate redundancy. For a given concept C, a cover is a disjunction of complexes describing all positive cases from C and not describing any negative cases from F = U − C.

The main idea of the AQ algorithm is to generate a cover for each concept by computing stars and selecting from them single complexes to the cover.

The AQ algorithm demands computing conjuncts of partial stars. In the worst case, time complexity of this computation is O(nm), where n is the number of attributes and m is the number of cases. The authors of AQ suggest using the parameter MAXSTAR as a method of reducing the computational complexity. According to

this suggestion, any set, computed by conjunction of partial stars, is reduced in size if the number of its members is greater than MAXSTAR. Obviously, the quality of the output of the algorithm is reduced as well.

15. Can we use decision trees for time series problems?

A:

Yes, we can use Random Forest. It is an ensemble of decision tree algorithms which is a popular and effective ensemble machine learning algorithm.

It is widely used for classification and regression predictive modeling problems with structured (tabular) data sets, e.g., data as it looks in a spreadsheet or database table.

Random Forest can also be used for time series forecasting, although it requires that the time series dataset be transformed into a supervised learning problem first. It also requires the use of a specialized technique for evaluating the model called walk-forward validation, as evaluating the model using k-fold cross validation would result in optimistically biased results.

16. From this section forward, the implementations and predictions will be on a Bitcoin price dataset. I got this dataset from Investing.com and changed the date column's format to a more useful interpretable standard format and changed the M for million and k for thousand in the Vol. column to numbers.
Then I split the test from train, in a way that the test data is from the beginning of 2010 to the beginning of 2020 and the rest is split for the test dataset.

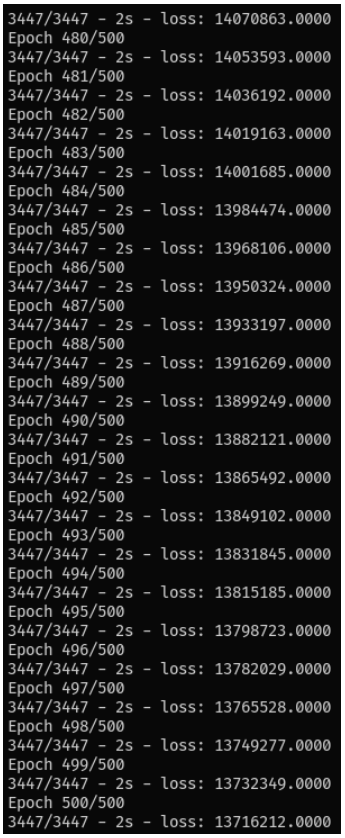
17. For this section, I use several methods for prediction on this dataset.

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems.

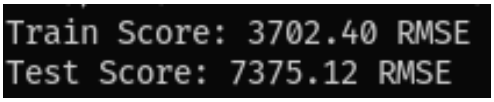
This is a behavior required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

The results from running LSTM (epoch 500) can be seen below, also RMSE for this model is evaluated:

```
Epoch 456/500
3447/3447 - 2s - loss: 14487250.0000
Epoch 457/500
3447/3447 - 2s - loss: 14468733.0000
Epoch 458/500
3447/3447 - 2s - loss: 14449943.0000
Epoch 459/500
3447/3447 - 2s - loss: 14431198.0000
Epoch 460/500
3447/3447 - 2s - loss: 14412711.0000
Epoch 461/500
3447/3447 - 2s - loss: 14394128.0000
Epoch 462/500
3447/3447 - 2s - loss: 14375636.0000
Epoch 463/500
3447/3447 - 2s - loss: 14357270.0000
Epoch 464/500
3447/3447 - 2s - loss: 14338872.0000
Epoch 465/500
3447/3447 - 2s - loss: 14320826.0000
Epoch 466/500
3447/3447 - 2s - loss: 14302372.0000
Epoch 467/500
3447/3447 - 2s - loss: 14284252.0000
Epoch 468/500
3447/3447 - 2s - loss: 14266086.0000
Epoch 469/500
3447/3447 - 2s - loss: 14248137.0000
Epoch 470/500
3447/3447 - 2s - loss: 14230039.0000
Epoch 471/500
3447/3447 - 2s - loss: 14212177.0000
Epoch 472/500
3447/3447 - 2s - loss: 14194334.0000
Epoch 473/500
3447/3447 - 2s - loss: 14176541.0000
Epoch 474/500
3447/3447 - 2s - loss: 14158818.0000
Epoch 475/500
3447/3447 - 2s - loss: 14141159.0000
Epoch 476/500
3447/3447 - 2s - loss: 14123614.0000
Epoch 477/500
3447/3447 - 2s - loss: 14105981.0000
Epoch 478/500
3447/3447 - 2s - loss: 14088300.0000
Epoch 479/500
3447/3447 - 2s - loss: 14070863.0000
```



RMSE for LSTM:



Next,

an ARIMA model is a class of statistical models for analyzing and forecasting time series data.

It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

The results from running ARIMA can be seen below, RMSE for this model is evaluated and can be seen at the end:

```
predicted=6177.766744, expected=6186.200000
predicted=6213.592214, expected=5822.100000
predicted=5812.940998, expected=6468.900000
predicted=6534.949355, expected=6744.600000
predicted=6745.479359, expected=6678.900000
predicted=6690.029293, expected=6725.100000
predicted=6691.170438, expected=6373.400000
predicted=6416.799310, expected=6233.700000
predicted=6255.655616, expected=5890.400000
predicted=5877.409189, expected=6391.000000
predicted=6401.894853, expected=6412.500000
predicted=6384.223665, expected=6638.500000
predicted=6639.924321, expected=6800.500000
predicted=6767.883691, expected=6735.900000
predicted=6776.168075, expected=6857.400000
predicted=6857.960644, expected=6772.700000
predicted=6786.349157, expected=7332.300000
predicted=7350.989005, expected=7185.200000
predicted=7177.581621, expected=7361.200000
predicted=7378.400456, expected=7289.000000
predicted=7272.354428, expected=6863.100000
predicted=6907.979967, expected=6867.800000
predicted=6851.727974, expected=6917.600000
predicted=6928.417798, expected=6841.300000
predicted=6842.495027, expected=6850.900000
predicted=6818.273830, expected=6629.100000
predicted=6626.416261, expected=7085.600000
predicted=7093.023520, expected=7035.800000
predicted=7028.215921, expected=7230.800000
predicted=7240.711433, expected=7122.900000
predicted=7097.663166, expected=6833.500000
predicted=6869.985699, expected=6842.500000
predicted=6833.514307, expected=7112.900000
predicted=7128.754671, expected=7488.500000
predicted=7488.024123, expected=7503.800000
predicted=7484.259761, expected=7540.400000
predicted=7540.109038, expected=7678.900000
predicted=7694.869628, expected=7766.000000
predicted=7796.436690, expected=7746.900000
predicted=7748.844177, expected=8770.900000
predicted=8778.800272, expected=8629.000000
Test RMSE: 416.337
```

Next,

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

I tried to run XGBoost on this dataset, the code is there but I couldn't run the algorithm because of the problems caused by my system.

18. Because I couldn't implement that many models in the last part, I skipped this section.

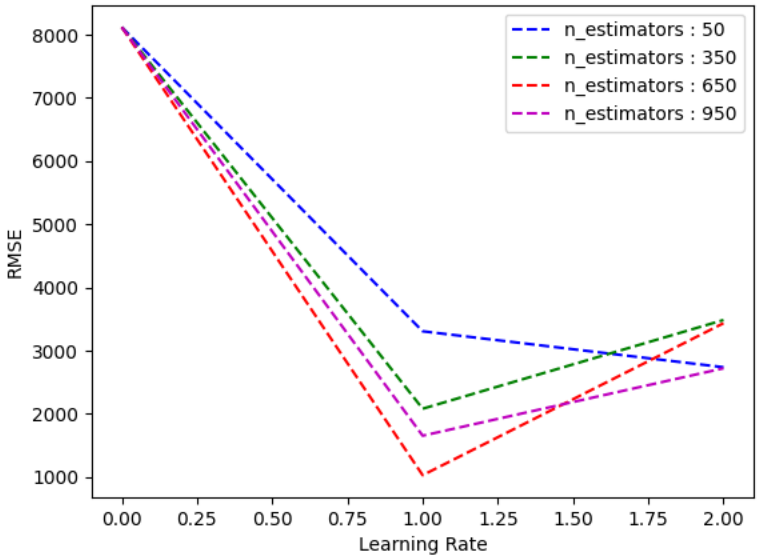
19. AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique that is used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances. Boosting is used to reduce bias as well as the variance for supervised learning. It works on the principle where learners are grown sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones.

For this part, I checked parameters for AdaBoost.

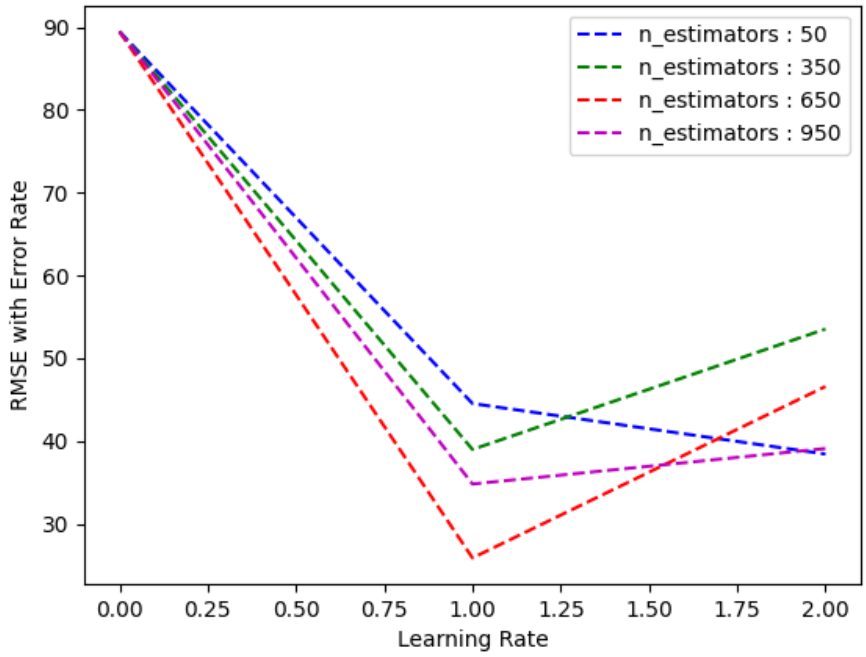
As the learning rate parameter is weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier.

The n_estimators parameter is maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

For the n_estimators parameter with default=50 and learning_rate parameter, I applied a for loop increasing the first parameter from 50 to 950 with step size 300 and the checked the RMSE for this model for different learning rates.



As asked in part 21 of this assignment, I set an upper bound and lower bound for the predictions if the predictions were within this bound, I assumed them to be true. The plot for this part can be seen below, RMSE with error rate:



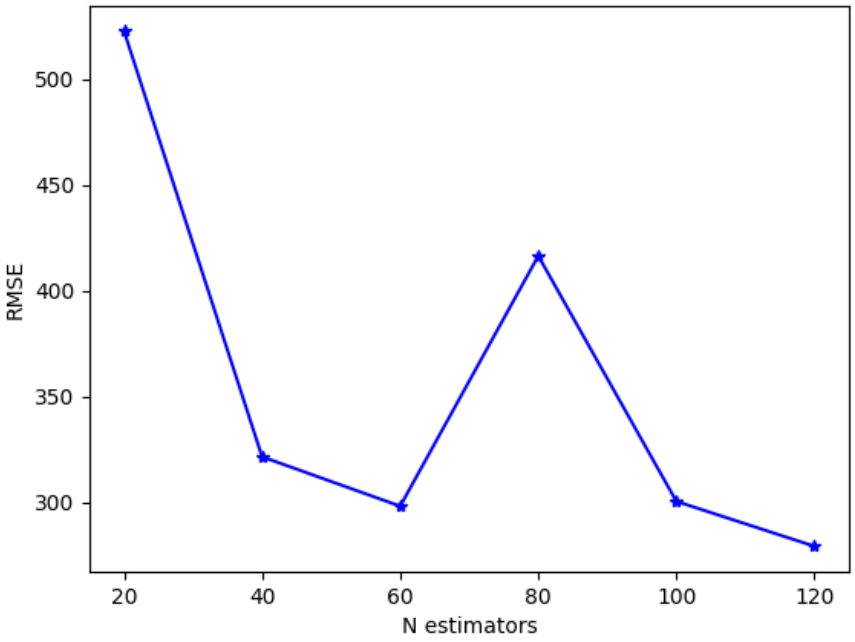
20. For this section, RF is run with different parameters.

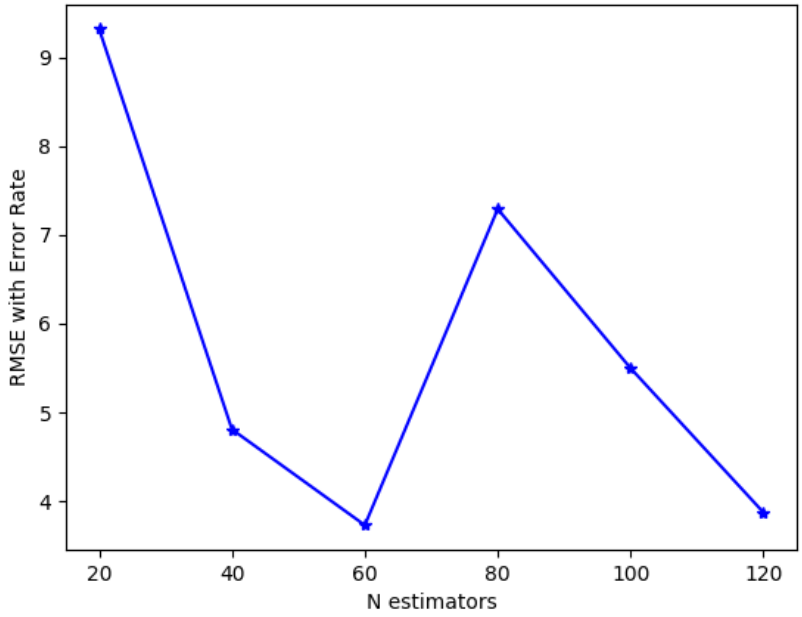
In this section I again checked the parameter n_estimators with default=100, for different values

This parameter indicates the number of trees in the forest.

Again, like before, RMSE is checked for this model and with a bound set for error rate.

The results can be seen below:





21. This section was shown in previous parts.

22.

In this section, the purpose was to check and predict whether the price in each step was increasing or decreasing. This required a little change in the dataset to label increasing steps yes and otherwise no. as a result of this change, one datapoint at the end of each of test and train is left out and not labeled.

The LSTM algorithm was used for this prediction and the results can be seen below:

```
Epoch 477/500
3447/3447 - 2s - loss: 0.2293
Epoch 478/500
3447/3447 - 2s - loss: 0.2297
Epoch 479/500
3447/3447 - 2s - loss: 0.2298
Epoch 480/500
3447/3447 - 2s - loss: 0.2295
Epoch 481/500
3447/3447 - 2s - loss: 0.2297
Epoch 482/500
3447/3447 - 2s - loss: 0.2298
Epoch 483/500
3447/3447 - 2s - loss: 0.2294
Epoch 484/500
3447/3447 - 2s - loss: 0.2296
Epoch 485/500
3447/3447 - 2s - loss: 0.2305
Epoch 486/500
3447/3447 - 2s - loss: 0.2294
Epoch 487/500
3447/3447 - 2s - loss: 0.2296
Epoch 488/500
3447/3447 - 2s - loss: 0.2299
Epoch 489/500
3447/3447 - 2s - loss: 0.2295
Epoch 490/500
3447/3447 - 2s - loss: 0.2297
Epoch 491/500
3447/3447 - 2s - loss: 0.2295
Epoch 492/500
3447/3447 - 2s - loss: 0.2299
Epoch 493/500
3447/3447 - 2s - loss: 0.2295
Epoch 494/500
3447/3447 - 2s - loss: 0.2296
Epoch 495/500
3447/3447 - 2s - loss: 0.2292
Epoch 496/500
3447/3447 - 2s - loss: 0.2299
Epoch 497/500
3447/3447 - 2s - loss: 0.2297
Epoch 498/500
3447/3447 - 2s - loss: 0.2302
Epoch 499/500
3447/3447 - 2s - loss: 0.2291
Epoch 500/500
3447/3447 - 2s - loss: 0.2293
(119,)
(119,)
Train Score: 0.48 RMSE
Test Score: 0.50 RMSE
```

23. Using the previous part of this exercise and adding that to the features, I run the prediction model again to see the change in results. It can be seen below:

```
3447/3447 - 3s - loss: 11922312.0000
Epoch 475/500
3447/3447 - 3s - loss: 11923591.0000
Epoch 476/500
3447/3447 - 3s - loss: 11924872.0000
Epoch 477/500
3447/3447 - 4s - loss: 11918895.0000
Epoch 478/500
3447/3447 - 4s - loss: 11918842.0000
Epoch 479/500
3447/3447 - 4s - loss: 11916318.0000
Epoch 480/500
3447/3447 - 4s - loss: 11915632.0000
Epoch 481/500
3447/3447 - 4s - loss: 11912186.0000
Epoch 482/500
3447/3447 - 4s - loss: 11917528.0000
Epoch 483/500
3447/3447 - 4s - loss: 11912809.0000
Epoch 484/500
3447/3447 - 4s - loss: 11907254.0000
Epoch 485/500
3447/3447 - 4s - loss: 12089608.0000
Epoch 486/500
3447/3447 - 4s - loss: 11945299.0000
Epoch 487/500
3447/3447 - 4s - loss: 11917309.0000
Epoch 488/500
3447/3447 - 4s - loss: 11896039.0000
Epoch 489/500
3447/3447 - 4s - loss: 11874696.0000
Epoch 490/500
3447/3447 - 4s - loss: 11868794.0000
Epoch 491/500
3447/3447 - 4s - loss: 11913947.0000
Epoch 492/500
3447/3447 - 4s - loss: 11873693.0000
Epoch 493/500
3447/3447 - 4s - loss: 11859046.0000
Epoch 494/500
3447/3447 - 4s - loss: 11830657.0000
Epoch 495/500
3447/3447 - 4s - loss: 11826167.0000
Epoch 496/500
3447/3447 - 4s - loss: 11816306.0000
Epoch 497/500
3447/3447 - 4s - loss: 11805883.0000
Epoch 498/500
3447/3447 - 4s - loss: 11812074.0000
Epoch 499/500
3447/3447 - 4s - loss: 11794357.0000
Epoch 500/500
3447/3447 - 4s - loss: 11787027.0000
```

As for the RMSE:

```
Train Score: 3432.05 RMSE
Test Score: 6317.72 RMSE
```

References

1. geeksforgeeks.org

2. stackoverflow.com

3. kdnuggets.com

4. Song, Yan-Yan, and L. U. Ying. "Decision tree methods: applications for classification and prediction." *Shanghai archives of psychiatry* 27.2 (2015): 130.

5. displayr.com

6. dtreg.com

7. Grzymala-Busse J.W. (2009) Rule Induction. In: Maimon O., Rokach L. (eds) Data Mining and Knowledge Discovery Handbook. Springer, Boston, MA.

8. machinelearningmastery.com

9. scikit-learn.org

10. pandas.pydata.org

11. medium.com

12. kaggle.com

13. towardsdatascience.com

14. pythonprogramming.net

15.sandipanweb.wordpress.com

16.towardsdatascience.com

17.delftstack.com

18.marsja.se

19.analyticsvidhya.com

20.machinelearningmastery.com

21.https://www.kaggle.com /bitcoin-price-prediction

Appendix

```
# -*- coding: utf-8 -*-
"""

@author: Sayeh

Spyder Editor
"""

#%% Imorting data and packaging

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as ss
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_sco
re, cross_val_predict, StratifiedKFold
from sklearn.metrics import auc, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.svm import SVC
from numpy import linalg
import cvxopt
import cvxopt.solvers
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from datetime import datetime
import csv
from xgboost import XGBRegressor
from statsmodels.tsa.arima.model import ARIMA
# Load libraries
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
import math

df1 = pd.read_csv (r'C:\Users\Asus\Desktop\Data Mining\03_assignment\data\
mobile price\train.csv')

#*************************WORKING WITH FIRST(mobile price) DATASET*******
*****************
#%%Understanding data for the second dataset

#to get a better understanding of how our data looks
df1.shape
df1.info() #no null data here
df1.describe()
df1.head(10)
df1.columns
#duplicated datapoints
df1.duplicated(keep=False).any() #no duplicates found

#deleting outliers

for cols in df1.columns:
    if df1[cols].dtype == 'int64' or df1[cols].dtype == 'float64':
        upper_range = df1[cols].mean() + 3 * df1[cols].std()
        lower_range = df1[cols].mean() - 3 * df1[cols].std()

        indexs = df1[(df1[cols] > upper_range) | (df1[cols] < lower_range)
].index
        df1 = df1.drop(indexs)

#dropping not useful datapoints
df1 = df1.drop(df1[df1['px_height'] == 0.0].index)

#%%Understanding data for the second dataset for questions of bitcoin data
set
df2 = pd.read_csv (r'C:\Users\Asus\Desktop\Data Mining\03_assignment\data\
Bitcoin 16-25\Bitcoin_Historical_Data_Investing.com.csv')

#to get a better understanding of how our data looks
df2.shape
df2.info() #no null data here
df2.describe()
df2.head(10)
df2.columns
#duplicated datapoints
df2.duplicated(keep=False).any() #no duplicates found
```

```
df2.isna()
#columns with null data
df2.isna().sum()/len(df2) #no null data

#%% correlation
df1.corr().price_range.sort_values()

corr = df1.corr()
corr
f, ax = plt.subplots(figsize=(15, 15))

sns.heatmap(corr, square = True ,annot = True)

#%% train and test split

y = df1["price_range"].values
x_data=df1.drop(["price_range"],axis=1)
x = (x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,ra
ndom_state=1)

#%% svm classifier 1

svm=SVC(kernel='rbf',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

#%% svm classifier 2 kernel sigmoid

svm=SVC(kernel='sigmoid',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

#%% svm classifier 2 kernel poly deg3

svm=SVC(kernel='poly', degree=3,random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

#%% svm classifier 2 kernel poly deg 10
```

```
svm=SVC(kernel='poly', degree=10,random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

#%% svm classifier 2 kernel linear

svm=SVC(kernel='linear',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

#%% svm classifier kernel rbf and gamma Kernel coefficient set on scale

svm=SVC(kernel='rbf',random_state=None, gamma='scale')
svm.fit(x_train,y_train)

y_pred = svm.predict(x_test)

print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred, pos_label='posi
tive', average='micro'))
# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred, pos_label='positive',
 average='micro'))

#%% svm classifier kernel rbf and gamma Kernel coefficient set on auto

svm=SVC(kernel='rbf',random_state=None, gamma='auto')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred, pos_label='posi
tive', average='micro'))
# Model Recall: what percentage of positive tuples are labelled as such?
```

```
print("Recall:",metrics.recall_score(y_test, y_pred, pos_label='positive',
 average='micro'))

#this part of the code is extra and for better understanding margins
#%% an example of soft margin hard margin and visialization for svm with p
lots
#I got this part of the code from https://pythonprogramming.net/soft-
margin-kernel-cvxopt-svm-machine-learning-tutorial/

def linear_kernel(x1, x2):
    return np.dot(x1, x2)

def polynomial_kernel(x, y, p=3):
    return (1 + np.dot(x, y)) ** p

def gaussian_kernel(x, y, sigma=5.0):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

class SVM(object):

    def __init__(self, kernel=linear_kernel, C=None):
        self.kernel = kernel
        self.C = C
        if self.C is not None: self.C = float(self.C)

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Gram matrix
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = self.kernel(X[i], X[j])

        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
```

```
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
        print("%d support vectors out of %d points" % (len(self.a), n_samp
les))

        # Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
        self.b /= len(self.a)

        # Weight vector
        if self.kernel == linear_kernel:
            self.w = np.zeros(n_features)
            for n in range(len(self.a)):
                self.w += self.a[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None

    def project(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
                    s += a * sv_y * self.kernel(X[i], sv)
                y_predict[i] = s
```

```
            return y_predict + self.b

    def predict(self, X):
        return np.sign(self.project(X))

if __name__ == "__main__":
    import pylab as pl

    def gen_lin_separable_data():
        # generate training data in the 2-d case
        mean1 = np.array([0, 2])
        mean2 = np.array([2, 0])
        cov = np.array([[0.8, 0.6], [0.6, 0.8]])
        X1 = np.random.multivariate_normal(mean1, cov, 100)
        y1 = np.ones(len(X1))
        X2 = np.random.multivariate_normal(mean2, cov, 100)
        y2 = np.ones(len(X2)) * -1
        return X1, y1, X2, y2

    def gen_non_lin_separable_data():
        mean1 = [-1, 2]
        mean2 = [1, -1]
        mean3 = [4, -4]
        mean4 = [-4, 4]
        cov = [[1.0,0.8], [0.8, 1.0]]
        X1 = np.random.multivariate_normal(mean1, cov, 50)
        X1 = np.vstack((X1, np.random.multivariate_normal(mean3, cov, 50))
)
        y1 = np.ones(len(X1))
        X2 = np.random.multivariate_normal(mean2, cov, 50)
        X2 = np.vstack((X2, np.random.multivariate_normal(mean4, cov, 50))
)
        y2 = np.ones(len(X2)) * -1
        return X1, y1, X2, y2

    def gen_lin_separable_overlap_data():
        # generate training data in the 2-d case
        mean1 = np.array([0, 2])
        mean2 = np.array([2, 0])
        cov = np.array([[1.5, 1.0], [1.0, 1.5]])
        X1 = np.random.multivariate_normal(mean1, cov, 100)
        y1 = np.ones(len(X1))
        X2 = np.random.multivariate_normal(mean2, cov, 100)
        y2 = np.ones(len(X2)) * -1
        return X1, y1, X2, y2
```

```
def split_train(X1, y1, X2, y2):
    X1_train = X1[:90]
    y1_train = y1[:90]
    X2_train = X2[:90]
    y2_train = y2[:90]
    X_train = np.vstack((X1_train, X2_train))
    y_train = np.hstack((y1_train, y2_train))
    return X_train, y_train

def split_test(X1, y1, X2, y2):
    X1_test = X1[90:]
    y1_test = y1[90:]
    X2_test = X2[90:]
    y2_test = y2[90:]
    X_test = np.vstack((X1_test, X2_test))
    y_test = np.hstack((y1_test, y2_test))
    return X_test, y_test

def plot_margin(X1_train, X2_train, clf):
    def f(x, w, b, c=0):
        # given x, return y such that [x,y] in on the line
        # w.x + b = c
        return (-w[0] * x - b + c) / w[1]

    pl.plot(X1_train[:,0], X1_train[:,1], "ro")
    pl.plot(X2_train[:,0], X2_train[:,1], "bo")
    pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

    # w.x + b = 0
    a0 = -4; a1 = f(a0, clf.w, clf.b)
    b0 = 4; b1 = f(b0, clf.w, clf.b)
    pl.plot([a0,b0], [a1,b1], "k")

    # w.x + b = 1
    a0 = -4; a1 = f(a0, clf.w, clf.b, 1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, 1)
    pl.plot([a0,b0], [a1,b1], "k--")

    # w.x + b = -1
    a0 = -4; a1 = f(a0, clf.w, clf.b, -1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, -1)
    pl.plot([a0,b0], [a1,b1], "k--")

    pl.axis("tight")
```

```
        pl.show()

    def plot_contour(X1_train, X2_train, clf):
        pl.plot(X1_train[:,0], X1_train[:,1], "ro")
        pl.plot(X2_train[:,0], X2_train[:,1], "bo")
        pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

        X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace(-6,6,50))
        X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2
))])
        Z = clf.project(X).reshape(X1.shape)
        pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='low
er')
        pl.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1, orig
in='lower')
        pl.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1, orig
in='lower')

        pl.axis("tight")
        pl.show()

    def test_linear():
        X1, y1, X2, y2 = gen_lin_separable_data()
        X_train, y_train = split_train(X1, y1, X2, y2)
        X_test, y_test = split_test(X1, y1, X2, y2)

        clf = SVM()
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
        correct = np.sum(y_predict == y_test)
        print("%d out of %d predictions correct" % (correct, len(y_predict
)))

        plot_margin(X_train[y_train==1], X_train[y_train==-1], clf)

    def test_non_linear():
        X1, y1, X2, y2 = gen_non_lin_separable_data()
        X_train, y_train = split_train(X1, y1, X2, y2)
        X_test, y_test = split_test(X1, y1, X2, y2)

        clf = SVM(polynomial_kernel)
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
```

```
        correct = np.sum(y_predict == y_test)
        print("%d out of %d predictions correct" % (correct, len(y_predict
)))

        plot_contour(X_train[y_train==1], X_train[y_train==-1], clf)

    def test_soft():
        X1, y1, X2, y2 = gen_lin_separable_overlap_data()
        X_train, y_train = split_train(X1, y1, X2, y2)
        X_test, y_test = split_test(X1, y1, X2, y2)

        clf = SVM(C=1000.1)
        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)
        correct = np.sum(y_predict == y_test)
        print("%d out of %d predictions correct" % (correct, len(y_predict
)))

        plot_contour(X_train[y_train==1], X_train[y_train==-1], clf)


    #test_linear()
    #test_non_linear()
    #test_soft()



#%% for matgin soft and hard

svm=SVC(C=1.0,kernel='linear',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

svm=SVC(C=10.0,kernel='linear',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))


svm=SVC(C=100.0,kernel='linear',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
```

```
print("test accuracy:",svm.score(x_test,y_test))


## FEATURE ENGINEERING
#%%binning on battery power

#****equal width binning*****
fig, ax = plt.subplots()
df1['battery_power'].hist(color='#A9C5D3', edgecolor='black',
                          grid=False)
ax.set_title('battery powr histogram', fontsize=12)
ax.set_xlabel('battery_power', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
#bin1
df1['battery_power_bin1'] = np.array(np.floor(np.array(df1['battery_power'
]) / 1000.))
df1[['battery_power', 'battery_power_bin1']].iloc[1071:1076]
df1['battery_power_bin1'].value_counts()
#df1 = df1.drop(columns=['battery_power'])

#bin2
df1['battery_power_bin2'] = np.array(np.floor(np.array(df1['battery_power'
]) / 100.))
df1['battery_power_bin2']
df1[['battery_power', 'battery_power_bin2']].iloc[1071:1076]
df1['battery_power_bin2'].value_counts()
#*****bin3 addaptive******
fig, ax = plt.subplots()
df1['battery_power'].hist(bins=30, color='#A9C5D3',
                            edgecolor='black', grid=False)
ax.set_title('battery powr histogram 2', fontsize=12)
ax.set_xlabel('battery_power', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)

#quantiled histogram bin3
fig, ax = plt.subplots()
df1['battery_power'].hist(bins=30, color='#A9C5D3',
                            edgecolor='black', grid=False)

quantile_list = [0, .25, .75, 1.]
quantiles = df1['battery_power'].quantile(quantile_list)
quantiles

for quantile in quantiles:
    qvl = plt.axvline(quantile, color='r')
```

```
ax.legend([qvl], ['Quantiles'], fontsize=10)
ax.set_title('Battery power Histogram with Quantiles',
             fontsize=12)
ax.set_xlabel('battery_power', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)

quantile_labels = ['0-25Q', '25-75Q', '75-100Q']
df1['battery_power_bin3'] = pd.qcut(df1['battery_power'],q=quantile_list)


df1[['battery_power', 'battery_power_bin1' ,'battery_power_bin2','battery_
power_bin3']].iloc[1071:1076]
#df1 = df1.drop(columns=['battery_power_bin3','battery_power_bin1' ,'batte
ry_power_bin2'])

#%%one hot encoding and dummies

columns = ['blue','dual_sim','four_g','three_g','touch_screen','wifi']
df1["blue"]=df1["blue"].astype(str)
df1["dual_sim"]=df1["dual_sim"].astype(str)
df1["four_g"]=df1["four_g"].astype(str)
df1["three_g"]=df1["three_g"].astype(str)
df1["touch_screen"]=df1["touch_screen"].astype(str)
df1["wifi"]=df1["wifi"].astype(str)


dummies_feature = pd.get_dummies(df1[columns])
dummies_feature.head()
dummies_feature.shape
df1 = pd.concat([df1, dummies_feature], axis=1)
df1.head()
df1 = df1.drop(columns=columns)
df1.head()
df1.info()
df1.shape

#%% transformation methods

#distribution plot for all features
df1.hist(grid=False,figsize=(10, 6),bins=30)

#applying log transform method for px_ height
df1['px_height_log'] = np.log(df1["px_height"])
#applying square root transform method for px_ height
df1['px_height_sqrt'] = np.sqrt(df1["px_height"])
```

```
df1 = df1.drop(columns=['px_height'])

#px_height pre transformation
sns.set(rc={'figure.figsize':(15,10)})
sns.distplot(df1["px_height"],kde_kws={"label": 'px_height'}, bins=20)

#px_height log transformation
sns.set(rc={'figure.figsize':(15,10)})
sns.distplot(np.log(df1["px_height"]),kde_kws={"label": 'px_height_log'},
bins=20)

#px_height sqrt transformation
sns.set(rc={'figure.figsize':(15,10)})
sns.distplot(np.sqrt(df1["px_height"]),kde_kws={"label": 'px_height_sqrt'}
, bins=20)

#%% adding a surface feature

df1['sc_surface'] = df1['sc_h'] * df1['sc_w']

#%%svm for the feature engineering part
# I ran each targeted cell for feature engineering and ran this svm model
afterwards to get results
svm=SVC(kernel='rbf',random_state=None)
svm.fit(x_train,y_train)
print("train accuracy:",svm.score(x_train,y_train))
print("test accuracy:",svm.score(x_test,y_test))

print(classification_report(y_pred, y_test))

#%% Decision tree classifier

dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
y_pred = dtc.predict(x_test)
print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

print(classification_report(y_pred, y_test))

#print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

#%% DT different parameters criterion

dtc = DecisionTreeClassifier(criterion='entropy')
```

```
dtc.fit(x_train,y_train)
#y_pred = dtc.predict(x_test)

print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

#%% DT different parameters splitter

dtc = DecisionTreeClassifier(splitter='random')
dtc.fit(x_train,y_train)
#y_pred = dtc.predict(x_test)

print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

#%% DT different parameters min_samples_leaf

dtc = DecisionTreeClassifier(min_samples_leaf=10)
dtc.fit(x_train,y_train)
#y_pred = dtc.predict(x_test)

print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

#%% DT different parameters max_depth

dtc = DecisionTreeClassifier(max_depth=10)
dtc.fit(x_train,y_train)
#y_pred = dtc.predict(x_test)

print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

#%%pruning DT


dtc = DecisionTreeClassifier(ccp_alpha=0.1)
dtc.fit(x_train,y_train)
#y_pred = dtc.predict(x_test)

print("dt train accuracy:",dtc.score(x_train,y_train))
print("dt test accuracy:",dtc.score(x_test,y_test))

#%% Random forest classifier
```

```
rdf = RandomForestClassifier()
rdf.fit(x_train,y_train)
y_pred = rdf.predict(x_test)
print("rdf train accuracy:",rdf.score(x_train,y_train))
print("rdf test accuracy:",rdf.score(x_test,y_test))

print(classification_report(y_pred, y_test))









#*************************WORKING WITH SECOND(Bitcoin) DATASET***********
**************
#%% For questions 16,17,19,20,22,23


def RMSE(y_test, y_pred, error_rate):

    # y_test_with_error_rate = y_test.copy()
    error_rate_list = np.array([
        y_test*(1-error_rate), y_test*(1+error_rate)
    ]).reshape(len(y_test), 2)

    indexes_1 = np.where(
        y_pred <= error_rate_list[:, 1]
    )[0]
    indexes_2 = np.where(
        y_pred >= error_rate_list[:, 0]
    )[0]
    indexes = set(indexes_1).intersection(list(indexes_2))

    for index in indexes:
        y_pred[index] = y_test[index]

    rmse = np.sqrt(
        abs(
```

```
            np.average(y_test) - np.average(y_pred)
        )
    )
    return rmse




def q_16():
    def mdy_to_ymd(d):
        return datetime.strptime(d, '%b %d, %Y').strftime('%Y-%m-%d')

    with open('Bitcoin 16-25/Bitcoin.csv') as csv_file:
        file = csv.reader(csv_file, delimiter=',')
        file = list(file)
        file = np.array(file)

        for i in range(1, len(file)):
            for j in range(len(file[0])):
                if j == 0:
                    file[i, j] = mdy_to_ymd(file[i, j])
                elif j == 5:
                    if 'M' in file[i, j]:
                        file[i, j] = float(file[i, j].replace('%', '').rep
lace('M', '')) * 10**6
                    if 'K' in file[i, j]:
                        file[i, j] = float(file[i, j].replace('%', '').rep
lace('K', '')) * 10**3
                elif j == 6:
                    file[i, j] = float(file[i, j].replace('%', '').replace
('%', ''))
                else:
                    file[i, j] = np.float64(file[i, j].replace(',', ''))
                    # file[i, j] = file[i, j].replace(',', '').astype(np.f
loat)
                    # print(file[i, j])
                    # print(type(file[i, j]))
        file = np.flip(file, 0)

        while len(np.where(file == '-')[0]) != 0:
            index = np.where(file == '-')[0][0]
            file = np.delete(file, index, 0)

        train_data = None
```

```
        test_data = None
        index = np.where(file[:, 0] == '2020-01-01')[0][0]
        train_data = file[:index, 1:]
        test_data = file[index+1:-2, 1:]

    train_data = train_data.astype(np.float64)
    train_data = train_data.round(decimals=2)
    test_data = test_data.astype(np.float64)
    test_data = test_data.round(decimals=2)

    return train_data, test_data


def q_17(X_train, X_test, y_train, y_test):

    def lstm(X_train, X_test, y_train, y_test):

        X_train = y_train[1:]
        y_train = y_train[:-1]

        X_test = y_test[1:]
        y_test = y_test[:-1]

        X_test = np.reshape(X_test, (len(X_test), 1, 1))
        X_train = np.reshape(X_train, (len(X_train), 1, 1))

        # create and fit the LSTM network
        model = Sequential()
        look_back = len(X_train[0])
        model.add(LSTM(4, input_shape=(look_back, 1)))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.fit(X_train, y_train, epochs=500, batch_size=1, verbose=2)

        # make predictions
        trainPredict = model.predict(X_train)
        testPredict = model.predict(X_test)

        # calculate root mean squared error
        # print(testPredict)
        trainScore = metrics.mean_squared_error(y_train, trainPredict[:,0]
, squared=False)
        print('Train Score: %.2f RMSE' % (trainScore))
        testScore = metrics.mean_squared_error(y_test, testPredict[:,0], s
quared=False)
```

```
        print('Test Score: %.2f RMSE' % (testScore))

    def arima(X_train, X_test, y_train, y_test):
        train, test = y_train, y_test
        history = [x for x in train]
        predictions = list()
        # walk-forward validation
        for t in range(len(test)):
            model = ARIMA(history, order=(5,1,0))
            model_fit = model.fit()
            output = model_fit.forecast()
            yhat = output[0]
            predictions.append(yhat)
            obs = test[t]
            history.append(obs)
            print('predicted=%f, expected=%f' % (yhat, obs))

        rmse = metrics.mean_squared_error(test, predictions, squared=False
)
        print('Test RMSE: %.3f' % rmse)

    def XGboost(X_train, X_test, y_train, y_test):
        def xgboost_forecast(train, testX):
            # transform list into array
            train = np.asarray(train)
            # split into input and output columns
            trainX, trainy = train[:-1], train[-1]
            # fit model
            model = XGBRegressor(objective='reg:squarederror', n_estimator
s=1000)
            model.fit(trainX, trainy)
            # make a one-step prediction
            yhat = model.predict([testX])
            return yhat[0]

        predictions = list()
        # split dataset
        train, test = y_train, y_test
        # seed history with training dataset
        history = [x for x in train]
        # step over each time-step in the test set
        for i in range(len(test)-1):
            # split test row into input and output columns
            testX, testy = test[i], test[i+1]
            # fit model on history and make a prediction
```

```
            yhat = xgboost_forecast(history, testX)
            # store forecast in list of predictions
            predictions.append(yhat)
            # add actual observation to history for the next loop
            history.append(test[i])
            # summarize progress
            print('>expected=%.1f, predicted=%.1f' % (testy, yhat))

        # estimate prediction error
        # error = mean_absolute_error(test[:, -1], predictions)
        rmse = metrics.mean_squared_error(test[:, -
1], predictions, squared=False)
        print('Test RMSE: %.3f' % rmse)


    # lstm(X_train, X_test, y_train, y_test)
    # arima(X_train, X_test, y_train, y_test)
    # XGboost(X_train, X_test, y_train, y_test)




def q_19(X_train, X_test, y_train, y_test):

    print('19')
    y_train = y_train.astype('int')
    y_train.round(decimals=2)
    y_test = y_test.astype('int')
    y_test.round(decimals=2)


    result = {}
    for n_estimators in range(50, 1000, 300):
        result[n_estimators] = [[], []]
        for learning_rate in range(1, 4):
            # Create adaboost classifer object
            abc = AdaBoostClassifier(
                n_estimators=n_estimators,
                learning_rate=learning_rate,
            )
            # Train Adaboost Classifer
            model = abc.fit(X_train, y_train)

            #Predict the response for test dataset
```

```
            y_pred = model.predict(X_test)

            # Model Accuracy, how often is the classifier correct?
            rmse = metrics.mean_squared_error(y_test, y_pred, squared=Fals
e)
            result[n_estimators][0].append(rmse)

            print('---------------------------')
            print(f"Model RMSE : {rmse}")

            rmse = RMSE(y_test, y_pred, error_rate=0.05)
            result[n_estimators][1].append(rmse)

            print(f"Model RMSE : {rmse}")

    colors = [
        'b--',
        'g--',
        'r--',
        'm--',
        'y--',
        'b*-',
        'g*-',
        'r*-',
        'm*-',
        'y*-',
    ]
    color_index = 0
    for key, value in result.items():
        label = f'n_estimators : {key}'
        plt.plot(value[0], colors[color_index], label=label)
        plt.legend()
        color_index += 1

    plt.xlabel('Learning Rate')
    plt.ylabel('RMSE')

    plt.figure()
    color_index = 0
    for key, value in result.items():
        label = f'n_estimators : {key}'
        plt.plot(value[1], colors[color_index], label=label)
        plt.legend()
        color_index += 1
```

```
    plt.xlabel('Learning Rate')
    plt.ylabel('RMSE with Error Rate')

    plt.show()


def q_20(X_train, X_test, y_train, y_test):

    print('20')
    y_train = y_train.astype('int')
    y_train.round(decimals=2)
    y_test = y_test.astype('int')
    y_test.round(decimals=2)


    result = [[], []]
    for n_estimators in range(20, 130, 20):
        # Create random forest classifer object
        abc = RandomForestClassifier(
            n_estimators=n_estimators,
        )
        # Train Adaboost Classifer
        model = abc.fit(X_train, y_train)

        #Predict the response for test dataset
        y_pred = model.predict(X_test)

        # Model Accuracy, how often is the classifier correct?
        print('----------------------------')

        rmse = metrics.mean_squared_error(y_test, y_pred, squared=False)
        result[0].append(rmse)
        print(f"Model RMSE : {rmse}")

        rmse = RMSE(y_test, y_pred, error_rate=0.05)
        result[1].append(rmse)
        print(f"Model RMSE : {rmse}")

    colors = [
        'b*-',
        'g--',
        'r--',
        'm--',
        'y--',
        'b*-',
```

```
        'g*-',
        'r*-',
        'm*-',
        'y*-',
    ]


    n_estimators_list = list(
        range(20, 140, 20)
    )

    color_index = 0
    plt.plot(n_estimators_list, result[0], colors[color_index])
    color_index += 1
    plt.xlabel(f'N estimators')
    plt.ylabel('RMSE')

    plt.figure()
    color_index = 0
    plt.plot(n_estimators_list, result[1], colors[color_index])
    color_index += 1
    plt.xlabel(f'N estimators')
    plt.ylabel('RMSE with Error Rate')

    plt.show()


def q_22(X_train, X_test, y_train, y_test):

    yp_test = []
    for i in range(len(y_test)-1):
        if y_test[i+1] >= y_test[i]:
            yp_test.append(1)
        else:
            yp_test.append(0)
    yp_test = np.array(yp_test)

    x_test = y_test.copy()[:-1]
    x_test = np.reshape(x_test, (len(x_test), 1, 1))

    # yp_test = np.reshape(yp_test, (len(yp_test), 1, 1))

    yp_train = []
    for i in range(len(y_train)-1):
        if y_train[i+1] >= y_train[i]:
```

```
            yp_train.append(1)
        else:
            yp_train.append(0)
    yp_train = np.array(yp_train)

    x_train = y_train.copy()[:-1]

    x_train = np.reshape(x_train, (len(x_train), 1, 1))

    # yp_train = np.reshape(yp_train, (len(yp_train), 1, 1))


    # create and fit the LSTM network
    model = Sequential()
    look_back = 1
    model.add(LSTM(4, input_shape=(1, look_back)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(x_train, yp_train, epochs=500, batch_size=1, verbose=2)

    # make predictions
    trainPredict = model.predict(x_train)
    testPredict = model.predict(x_test)

    # invert predictions
    # trainPredict = scaler.inverse_transform(trainPredict)
    # yp_train = scaler.inverse_transform([yp_train])
    # testPredict = scaler.inverse_transform(testPredict)
    # yp_test = scaler.inverse_transform([yp_test])
    # calculate root mean squared error
    print(testPredict[:,0].shape)
    print(yp_test.shape)
    trainScore = metrics.mean_squared_error(yp_train, trainPredict[:,0], s
quared=False)
    print('Train Score: %.2f RMSE' % (trainScore))
    testScore = metrics.mean_squared_error(yp_test, testPredict[:,0], squa
red=False)
    print('Test Score: %.2f RMSE' % (testScore))



def q_23(X_train, X_test, y_train, y_test):

    yp_test = []
    for i in range(len(y_test)-1):
```

```
        if y_test[i+1] >= y_test[i]:
            yp_test.append(1)
        else:
            yp_test.append(0)
    yp_test = np.array(yp_test)

    X_test = X_test[:-1]
    y_test = y_test[:-1]
    print(X_test.shape)
    X_test = np.c_[X_test, yp_test]

    # x_test = y_test.copy()[:-1]
    X_test = np.reshape(X_test, (len(X_test), len(X_test[0]), 1))

    # yp_test = np.reshape(yp_test, (len(yp_test), 1, 1))

    yp_train = []
    for i in range(len(y_train)-1):
        if y_train[i+1] >= y_train[i]:
            yp_train.append(1)
        else:
            yp_train.append(0)
    yp_train = np.array(yp_train)
    X_train = X_train[:-1]
    y_train = y_train[:-1]
    X_train = np.c_[X_train, yp_train]

    # x_train = y_train.copy()[:-1]

    X_train = np.reshape(X_train, (len(X_train), len(X_train[0]), 1))

    # yp_train = np.reshape(yp_train, (len(yp_train), 1, 1))


    # create and fit the LSTM network
    model = Sequential()
    look_back = len(X_train[0])
    model.add(LSTM(4, input_shape=(look_back, 1)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=500, batch_size=1, verbose=2)

    # make predictions
    trainPredict = model.predict(X_train)
    testPredict = model.predict(X_test)
```

```
    # calculate root mean squared error
    print(testPredict)
    trainScore = metrics.mean_squared_error(y_train, trainPredict[:,0], sq
uared=False)
    print('Train Score: %.2f RMSE' % (trainScore))
    testScore = metrics.mean_squared_error(y_test, testPredict[:,0], squar
ed=False)
    print('Test Score: %.2f RMSE' % (testScore))


#%% now lets investigate

if __name__ == "__main__":

    train_data, test_data = q_16()

    # q_17(
    #     X_train=train_data[:, 1:],
    #     X_test=test_data[:, 1:],
    #     y_train=train_data[:, 0],
    #     y_test=test_data[:, 0],
    # )

    # q_19(
    #     X_train=train_data[:, 1:],
    #     X_test=test_data[:, 1:],
    #     y_train=train_data[:, 0],
    #     y_test=test_data[:, 0],
    # )

    # q_20(
    #     X_train=train_data[:, 1:],
    #     X_test=test_data[:, 1:],
    #     y_train=train_data[:, 0],
    #     y_test=test_data[:, 0],
    # )

    # q_22(
    #     X_train=train_data[:, 1:],
    #     X_test=test_data[:, 1:],
    #     y_train=train_data[:, 0],
    #     y_test=test_data[:, 0],
    # )
    q_23(
```

```
        X_train=train_data[:, 1:],
        X_test=test_data[:, 1:],
        y_train=train_data[:, 0],
        y_test=test_data[:, 0],
    )
```