

[MUSIC] Hi, in this lesson we will talk about the very first steps a good data scientist takes when he is given a new data set. Mainly, exploratory data analysis or EDA in short. By the end of this lesson, you will know, what are the most important things from data understanding and exploration prospective we need to pay attention to. This knowledge is required to build good models and achieve high places on the leader board. We will first discuss what exploratory data analysis is and why we need it. We will then go through important parts of EDA process and see examples of what we can discover during EDA. Next we will take a look at the tools we have to perform exploration. What plots to draw and what functions from pandas and matplotlib libraries can be useful for us. We will also briefly discuss a very basic data set cleaning process that is convenient to perform while exploring the data. And finally we'll go through exploration process for the Springleaf competition hosted on Kaggle some time ago. In this video we'll start talking about Exploratory Data Analysis. What is Exploratory Data Analysis? It's basically a process of looking into the data, understanding it and getting comfortable with it. Getting comfortable with a task, probably always the first thing you do. To solve a problem, you need to understand a problem, and to know what you are given to solve it. In data science, complete data understanding is required to generate powerful features and to build accurate models. In fact while you explore the data, you build an intuition about it. And when the data is intuitive for you, you can generate hypothesis about possible new features and eventually find some insights in the data which in turn can lead to a better score. We will see the example of what EDA can give us later in this lesson. Well, one may argue that there is another way to go. Read the data from the hard drive, never look at it and feed the classifier immediately. They use some pretty advanced modeling techniques, like mixing, stacking, and eventually get a pretty good score on the leaderboard. Although this approach sometimes works, it cannot take you to the very top positions and let you win. Top solutions always use advanced and aggressive modeling. But usually they have something more than that. They incorporated insights from the data, and to find those insights, they did a careful EDA. While we need to admit the raw computations where all you can do is modeling and EDA will not help you to build a better model. It is usually the case when the data is anonymized, encrypted, pre-processed, and obfuscated

. But look it will any way need to perform EDA to realize that this is the case and you better spend more time on modeling and make a server busy for a month. One of the main EDA tools is Visualization. When we visualize the data, we immediately see the patterns. And with this, ask ourselves, what are those patterns? Why do we see them? And finally, how do we use those patterns to build a better model? It also can be another way around. Maybe we come up with a particular hypothesis about the data. What do we do? We test it by making a visualization. In one of the next videos, we'll talk about the main visualization tools we can use for exploration. Just as a motivation example, I want to tell you about the competition, alexander D'yakonov, a former top one at Kagel took part some time ago. The interesting thing about this competition is that you do not need to do any modeling, if you understood your data well. In that competition, the objective was to predict whether a person will use the promo that a company offers him. So each role correspond to a particular promo received by a person. There are features that describe the person, for example his age, sex, is he married or not and so on. And there are features that describe the promo, the target is 0 or 1, will he use the promo or not. But, among all the features there were two especially interesting. The first one is, the number of promos sent by the person before. And the second is the number of promos the person had to use before. So let's take a particular user, say with index 13, and sort the rows by number of promos sent column. And now let's take a look at the difference at column the number of used promos between two consecutive rows. It is written here in diff column. And look, the values in diff column in most cases equal the target values. And in fact, there is no magic. Just think about the meaning of the columns. For example, for the second row we see that the person used one promo already but he was sent only one before that time. And that is why we know that he used the first promo and thus we have an answer for the first row. In general, if before the current promo the person used n promos and before the next promo he used $n + 1$ promos then we realize that he used the current promo. And so the answer is 1. If we know that he used n promos before the next promo, exactly as before the current promo, then obviously he did not use the current promo and the answer is 0. Well, it's not clear what to do with the last row for every user, or when we have missing rows,

but you see the point. We didn't even run the classifier, and we have 80% accuracy already. This would not be possible if we didn't do an EDA and didn't look into the data. Also as a remark, I should say that the presented method works because of mistake made by the organizers during data preparation. These mistakes are called leaks, and in competitions we are usually allowed to exploit them. We'll see more of these examples later in this course. So in this video we discussed the main reasons for performing an EDA. That is to get comfortable with the data and to find insights in magic features. We also saw an example where EDA and the data understanding was important to get a better score. And finally, the point to take away. When you start a competition, you better start with EDA, and not with hardcore modelling. We've had a lot of things to talk about in this lesson. So let's move to the next video. [MUSIC] In this video, we'll go through and break down several important steps namely, the first, getting domain knowledge step, second, checking if data is intuitive, and finally, understanding how the data was generated. So let's start with the first step, getting the domain knowledge. If we take a look at the computations hosted in the Kaggle, well, you'll notice, they are rather diverse. Sometimes, we need to detect threats on three dimensional body scans, or predict real estate price, or classify satellite images. Computation can be on a very specific topic which we know almost nothing about, that is, we don't have a domain knowledge. Usually, we don't need to go too deep inside the field but it's preferable to understand what our aim is, what data we have, and how people usually tackle this kind of problems to build a baseline. So, our first step should probably be searching for the topic, Googling within Wikipedia, and making sure we understand the data. For example, let's say we start a new computation in which we need to predict advertisers cost. Our first step is to realize that the computation is about web advertisement. By looking and searching for the column names, using any search engine, we understand that the data was exported from Google AdWords system. And after reading several articles about Google AdWords, we get the meaning of the columns. We now know that impressions column contained a number of times a particular ad appeared before users, and clicks column is how many times the ad was clicked by the users, and of course, the number of clicks should be less or equal than the number of impression. In this video, we'll not go much further into the details about this data set, but you can open the supplementary reading material for a more detailed exploration. After we've learned some domain knowledge, it is necessary to check if the values in the data set are intuitive, and agree with our domain knowledge. For example, if there is a column with age data, we should expect the values rarely to be larger than 100. And for sure, no one ever lived more than 200 years. So, the values should be smaller than 200. But for some reason, we find this super huge value 336. Most probably, is just a typo but it should be 36 or 33, and the best we can do is manually

y correct it. But the other possibility is that it's not a human age, but some alien's age for which it's totally normal to live more than 300 years. To check that, we should probably read the data description one more time, ask on forums. Maybe the data is totally correct, and then we just misinterpret it. Now, take a look at our Google AdWords data set. We understood that the values in the clicks variable should be less or equal than the values in impressions column. And in our case, in the first row, we see zero impressions and three clicker. That sounds like a bug, right? In fact, it probably is, but differently to the example of person's age, it could be rather a regular error made by either data exporting script or another kind of algorithm. That is, the errors were made not at random, but there is some kind of logic why there is an error in that particular place. It means that these mistakes can be used to get a better score. For example, in our case, we could create a new feature, `is_incorrect`, and mark all the rows that have errors. Probably, our models will find this feature helpful. It is also very important to understand how the data was generated. What was the algorithm for sampling objects from the database? Maybe, the host sample get objects at random, or they over-sample the particular class, that is, they generated more examples of that class. For example, to make the data set more class balanced. In fact, only if you know how the data was generated, you can set up a proper validation scheme for models. Coming down for the correct validation pipeline is crucial, and we will discuss it later in this course. So, what can we possibly find out about generation processes? For example, we could find out the train and test set were generated with different algorithms. And if the test set is different to the train set, we cannot use part of the train set as a validation set, because this part will not be representative of test set. And so, we cannot evaluate our models using it. So once again, to set up a correct validation, we need to know underlying data generation processes. In the ad computation, we've discussed before, that all the symptoms of different train test sampling. Improving the model on validation set didn't result into improved public leader-board score. And more, the leader-board score was unexpectedly higher than the validation score. I was also visualizing various things while trying to understand what's happening, and every time, the plots for the train set were much different to the test set plots. This also could not happen if the train and test set were similar. And finally, it was suspicious that although the train period was more than ten times larger than the test period, the train set had much fewer rows. It was not straightforward, but this triangle on the left figure was the clue for me, and the puzzle was solved. I've adjusted the train set to match test set. The validation score became reliable, and the modeling could be commenced. You can find the entire task description and investigation in the written materials. So, in this video, we've discussed several important exploratory steps. First, we need to get domain knowledge about the task as it helps to better understand the problem and the data. Next, we need to check if the data is intuitive, and agrees with our domain knowledge. And finally, it is necessary to understand how the data was generated by organizers because otherwise, we cannot establish a proper validation for our models.[SOUND] In the previous video,

we were working with the data for which we had a nice description. That is, we knew what the features were, and the data was given us as the set without severe modifications. But, it's not always the case. The data can be anonymized, and obfuscated. In this video, we'll first discuss what is anonymized data, and why organizers decide to anonymize their data. And next, we will see what we as competitors can do about it. Sometimes we can decode the data, or if we can not we can try to guess, what is the type of feature. So, let's get to the discussion. Sometimes the organizers really want some information to be reviewed. So, they make an effort to export competition data, in a way one couldn't get while you're out of it. Yet all the features are preserved, and machinery model will be able to do it's job. For example, if a company wants someone to classify its document, but doesn't want to reveal the document's content. It can replace all the word occurrences with hash values of those words, like in the example you see here. In fact, it will not change a thing for a model based on bags of words. I will refer to Anonymized data as to any data which organizers intentionally changed. Although it is not completely correct, I will use this wording for any type of changes. In computations with tabular data, companies can try to hide information each column stores. Take a look at this data set. First, we don't have any meaningful names for the features. The names are replaced with some dummies, and we see some hash like values in columns x1 and x6. Most likely, organizers decided to hash some sensitive data. There are several things we can do while exploring the data in this case. First, we can try to decode or de-anonymize the data, in a legal way of course. That is, we can try to guess true meaning of the features. Sometimes de-anonymization is not possible, but what we almost surely can do, is to guess the type of the features, separating them into numeric, categorical, and so on. Then, we can try to find how features relate to each other. That can be a specific relation between a pair of features, or we can try to figure out if the features are grouped in some way. In this video we will concentrate on the first problem. In the next video we will discuss visualization tools, that we can use both for exploring individual features, and feature relations. Let's now get to an example how it was possible to decode the meaning of the feature in one local competition I took part. I want to tell you about a competition I took part. It was a local competition, and organizers literally didn't give competitors any information about a dataset. They just put the link to download data on the competition page, and nothing else. Let's read the data first

t, and basically what we see here is that the data is anonymized. The column names are like x something, and the values are hashes, and then the rest are numeric in here. But, well we don't know what they mean at all, and basically we don't know what we are to predict. We only know that it is a multi-class classification task, and we have four labels. So, as long as we don't know what the data is, we can probably build a quick baseline. Let's import Random Forest Classifier. Yeah, of course we need to drop target label from our data frame, as it is included in there. We'll fill null values with minus 999, and let's encode all the categorical features, that we can find by looking at the types. Property of our data frame. We will encode them with Label Encoder, and it is easier to do with function factorize from Pandas. Let's feed to Random Forest Classifier on our data. And let's plot the feature importance's, and what we see here is that feature X8 looks like an interesting one. We should probably investigate it a little bit deeper. If we take the feature X8, and print it's mean, and estimate the value. They turn out to be quite close to 0, and 1 respectively, and it looks like this feature was tendered skilled by the organizers. And we don't see here exactly 0, and exactly 1, because probably training test was concatenated when on the latest scale. If we concatenate training test, then the mean will be exactly 0, and the std will be exactly 1. Okay, so let's also see are there any other repeated values in these features? We can do it with a value counts function. Let's print first 15 rows of value counts out. And we can see that there are a lot of repeated values, they repeated a thousand times. All right, so we now know that this feature was standard scaled. Probably, we can try to scale it back. The original feature was multiplied by a number, and was shifted by a number. All we need to do is to find the shooting parameter, and the scaling parameter. But how do we do that, and it is really possible? Let's take unique values of the feature, and sort them. And let's print the difference between two consecutive numbers, in this sorted array. And look, it looks like the values are the same all the time. The distance between two consecutive unique values in this feature, was the same in the original data too. It was probably not 0.043 something, it was who knows, it could be 9 or 11 or 11.7, but it was the same between all the pairs, so assume that it was 1 because, well, 1 looks like a natural choice. Let's divide our feature by this number 0.043 something, and if we do it, yes, we see that

the differences become rather close to 1, they are not 1, only because of some numeric errors. So yes, if we divide our feature by this value, this is what you get. All right, so what else do we see here. We see that each number, it ends with the same values. Each positive number ends with this kind of value, and each negative with this, look. It looks like this fractional part was a part of the shifting parameter, let's just subtract it. And in fact if we subtract it, the data looks like an integers, actually. Like it was integer data, but again because of numeric errors, we see some weird numbers in here. Let's round the numbers, and that is what we get. This is actually on the first ten rows, not the whole feature. Okay, so what's next? What did we do so far? We found the scaling parameter, probably we were right, because the numbers became integers, and it's a good sign. We could be not right, because who knows, the scaling parameter could be 10 or 2 or again 11 and still the numbers will be integers. But, 1 looks like a good match. It couldn't be as random, I guess. But, how can we find the shifting parameter? We found only fractional part, can we find the other, and can we find the integer part, I mean? It's actually a hard question, because while you have a bunch of numbers in here, and you can probably build a hypothesis. It could be something, and the regular values for this something is like that, and we could probably scale it, shift it by this number. But it could be only an approximation, and not a hypothesis, and so our journey could really end up in here. But I was really lucky, and I will show it to you, so if you take your x8. I mean our feature, and print value counts, what we will see, we will see this number 11, 17, 18, something. And then if we scroll down we will see this, -1968, and it definitely looks like year of birth, right? Immediately I have a hypothesis, that this could be a text box where a person should enter his year of birth. And while most of the people really enter their year of birth, but one person entered zero. Or system automatically entered 0, when something wrong happened. And wow, that isn't the key. If we assume the value was originally 0, then the shifting parameter is exactly 9068, let's try it. Let's add 9068 to our data, and see the values. Again we will use value counts function, and we will sort sorted values. This is the minimum of the values, and in fact you see the minimum is 0, and all the values are not negative, and it looks really plausible. Take a look, 999, it's probably what people love to enter when they're asked to enter something, or this, 1899. It could be a default value for

this textbook, it occurred so many times. And then we see some weird values in here. People just put them at random. And then, we see some kind of distribution over the dates. That are plausible for people who live now, like 1980. Well maybe 1938, I'm not sure about this, and yes of course we see some days from the future, but for sure it looks like a year of birth, right? Well the question, how can we use this information for the competition? Well again for linear models, you probably could make a new feature like age group, or something like that. But In this particular competition, it was no way to use this for, to use this knowledge. But, it was really fun to investigate. I hope you liked the example, but usually is really hard to recognize anything sensible like a year of birth anonymous features. The best we can do is to recognize the type of the feature. Is it categorical, numeric, text, or something else? Last week we saw that each data type should be treated differently, and more treatment depends on the model we want to use. That is why to make a stronger model, we should know what data we are working with. Even though we cannot understand what the features are about, we should at least detect the types of variables in the data. Take a look at this example, we don't have any meaningful companies, but still we can deduce what the feature types are. So, x1 looks like text or physical recorded, x2 and x3 are binary, x4 is numeric, x5 is either categorical or numeric. And more, if it's numeric it could be something like event calendars, because the values are integers. When the number of columns in data set is small, like in our example, we can just bring the table, and manually sort the types out. But, what if there are thousand of features in the data set? Very useful functions to facilitate our exploration, function `dtypes` from pandas guesses the types for each column in the data frame. Usually it groups all the columns into three categories, `object`, `integer`, and so called `object` type. If `dtype` function assigned `object` type to a feature, this feature is most likely to be numeric. Integer typed features can be either binary encoded with a zero or one. Event counters, or even categorical, encoded with the label encoder. Sometimes this function returns a type named `object`. And it's the most problematic, it can be anything, even an irregular numeric feature with missing values filled with some text. Try it on your data, and also check out a very similar in full function from Pandas. To deal with object types, it is useful to print the data and literally look at it. It is useful to check unique values with `value_counts` function, and nulls location with

isnull function at times. In this lesson, we were discussing two things we can do with anonymized features. We saw that sometimes, it's possible to decode features, find out what this feature really means. It doesn't matter if we understand the meaning of the features or not, we should guess the feature types, in order to pre-process features accordingly to the type we have, and selected model class. In the next video, we'll see a lot of colorful plots, and talk about visualization, and other tools for exploratory data analysis. [SOUND] In the previous video, we've tried to decode anonymized features and guess their types. In fact, we want to do more. We want to generate new features and to find insights in a data. And in this lesson, we will talk about various visualizations that can help us with it. We will first to see what plots we can draw to explore individual features, and then we will get to exploration of feature relations. We'll explore pairs first and then we'll try to find feature groups in a dataset. First, there is no recipe how you find interesting things in the data. You should just spend some time looking closely at the data table, printing it, and examining. If we found something interesting, we then can take a closer look. So, EDA is kind of an art, but we have a bunch of tools for it which we'll discuss right now. The first, we can build histograms. Histograms split feature edge into bins and show how many points fall into each bin. Note that histograms may be misleading in some cases, so try to overwrite its number of bins when using it. Also, know that it aggregates in the data, so we cannot see, for example, if all the values are unique or there are a lot of repeated values. Let's see in other example. The first thing that I want to illustrate here is that histograms can confuse. Looking at this histogram, we could probably think that there are a lot of zero values in this feature. But in fact, if we take log arithm of the values and build histogram again, we'll clearly see that distribution is non-degenerate and there are many more distinct values than one. So my point is never make a conclusion based on a single plot. If you have a hypothesis, try to make several different plots to prove it. The second interesting thing here is that peak. What is it? It turns out that the peak is located exactly at the mean value of this feature. Seems like organizers filled the missing values with the mean values for us. So, now we understand that values were originally missing. How can we use this information? We can replace the missing values we found with not numbers, nulls again. For example, [inaudible] has a special algorithm that can fill missing values on its own and so, maybe [inaudible] will benefit from explicit missing values. Or we can fill the missing values with something other than feature mean, for example, with -999. Or we can generate a new feature which will indicate that the value was missing. This can be particularly useful for linear models. We can also build the plot where on X axis, we have a row index, and on the Y axis, we have feature values. It is convenient not to connect points with line segments but only draw them with circles. Now, if we observe horizontal lines on this kind of plot, we understand there are a

lot of repeated values in this feature. Also, note the randomness over the indices. That is, we see some horizontal patterns but no vertical ones. It means that the data is properly shuffled.

We can also color code the points according to their labels. Here, we see that the feature is quite good as it presumably gives a nice class separation. And also, we clearly see that the data is not shuffled here. It is, in fact, sorted by class label. It is useful to examine statistics with Pandas' describe function.

You can see examples of its output on the screenshot. It gives you information about mean, standard deviation, and several percentiles of the feature distribution. Of course, you can manually compute those statistics. In Pandas' nan type, you can find functions named by statistics they compute. Mean for mean value, var for variance, and so on, but it's really convenient to have them all in once. And finally, as we already discussed in the previous video, there is value_counts function to examine the number of occurrences of distinct feature values, and a function is null, which helps to find the missing values in the data. For example, you can visualize nulls patterns in the data as on the picture you see. So, here's the full list of functions we've discussed.

Make sure you remember each of them. To this end, we've discussed visualizations for individual features. And now, let's get to the next topic of our discussion, exploration of feature relations. It turns out that sometimes, it's hard to make conclusions looking at one feature at a time. So let's look at the pairs.

The best two here is a scatter plot. With it, we can draw one sequence of values versus another one. And usually, we plot one feature versus another feature. So each point on the figure corresponds to an object with the feature values shown by points position. If it's a classification task, it's convenient to color code the points with their labels like on this picture. The color indicates the class of the object. For regression, the heat map light coloring can be used, too. Or alternatively, the target value can be visualized by point size. We can effectively use scatter plots to check if the data distribution in the train and test sets are the same. In this example, the red points correspond to class zero, and the blue points to class one. And on top of red and blue points, we see gray points. They correspond to test set. We don't have labels for the test set, that is why they are gray. And we clearly see that the red points are mixed with part of the gray ones, and that that is good actually. But other gray points are located in the region where we don't have any training data, and that is bad. If you see some kind of discrepancy between colored and gray points distribution, you should probably stop and think if you're doing it right. It can be just a bug in the code, or it can be completely overfitted feature, or something else that is for sure not healthy. Now, take a look at this scatter plot. Say, we plot feature X_1 versus feature X_2 . What can we say about their relation? The right answer is X_2 is less or equal than $1 - X_1$. Just realize that the equation for the diagonal line is $X_1 + X_2 = 1$, and for all the points below the line, X_2 is less or equal than $1 - X_1$. So, suppose we found this relation between two features, how do we use this fact? Of course, it depends, but at least there are some obvious features to generate. For tree-based models, we can create new features like the difference or ratio between X_1 and X_2 . Now, take a

look at this scatter plot. It's hard to say what is the true relation between the features, but after all, our goal is not to decode the data here but to generate new features and get a better score. And this plot gives us an idea how to generate the features out of these two features. We see several triangles on the picture, so we could probably make a feature to each triangle a given point belongs, and hope that this feature will help. When you have a small number of features, you can plot all the pairwise scatter plots at once using scatter metrics function from Pandas. It's pretty handy. It's also nice to have histogram and scatter plot before the eyes at the same time as scatter plot gives you very vague information about densities, while histograms do not show feature interactions. We can also compute some kind of distance between the columns of our feature table and store them into a matrix of size number of features by a number of features. For example, we can compute correlation between the counts. It's the most common type of matrices people build, correlation metric. But we can compute other things than correlation. For example, how many times one feature is larger than the other? I mean, how many rows are there such that the value of the first feature is larger than the value of the second one? Or another example, we can compute how many distinct combinations the features have in the dataset. With such custom functions, we should build the metrics manually, and we can use matshow function from Matplotlib to visualize it like on the slide you see. If the metrics looks like a total mess like in here, we can run some kind of clustering like K-means clustering on the rows and columns of this matrix and reorder the features. This one looks better, isn't it? We actually came to the last topic of our discussion, feature groups. And it's what we see here. There are groups of very similar features, and usually, it's a good idea to generate new features based on the groups. Again, it depends, but maybe some statistics could collated over the group will work fine as features. Another visualization that helps to find feature groups is the following: We calculate the statistics of each feature, for example, mean value, and then plot it against column index. This plot can look quite random if the columns are shuffled. So, what if we sorted the columns based on this statistic? Feature and mean, in this case. It looks like it worked out. We clearly see the groups here. So, now we can take a closer look to each group and use the imagination to generate new features. And here is a list of all the functions we've just discussed. Pause the video and check if you remember the examples we saw. So, finally in this video, we we're talking about the tools and functions that help us with data exploration. For example, to explore features one by one, we can use histograms, plots, and we can also examine statistics. To explore a relation between the features, the best tool is a scatter plot. Scatter metrics combines several scatter plots and histograms on one figure. Correlation plot is useful to understand how similar the features are. And if we reorder the columns and rows of the correlation metrics, we'll probably find feature groups. And feature groups was the last topic we discussed in this lesson. We also saw a plot of sorted feature statistics and how it can reveal as feature groups. Well, of course, we've discussed only a fraction of helpful plots there are. With practice, you will develop and find your own tools further expl

oration.[MUSIC] Hi, in this video we will discuss a little bit of dataset cleaning and see how to check if dataset is shuffled. It is important to understand that the competition data can be only apart of the data organizers have. The organizers could give us a fraction of objects they have or a fraction of features. And that is why we can have some issues with the data. For example, we can encounter a feature which takes the same value for every object in both train and test set. This could be due to the sampling procedure. For example, the future is a year, and the organizers exported us only one year of data. So in the original data that the organizers have, this future is not constant, but in the competition data it is constant. And obviously, it is not useful for the models and just occupy some memory. So we are about to remove such constant features. In this example data set feature of zero is constant. It can be the case that the feature is constant on the train set but how is different values on the test set. Again, it is better to remove such features completely since it is constant during training. In our dataset feature is f1. What is the problem, actually? For example, my new model can assign some weight to this future, so this future will be a part of the prediction formula, and this formula will be completely unreliable for the objects with the new values of that feature. For example, for the last row in our data set. J row, even if categorical feature is not constant on the train path but there were values that present only in the test data, we need to handle this situation properly. We need to decide, do these new values matter much or not? For example, we can simulate this situation with a validation set and compare the quality of the predictions on the objects with the syn feature values and objects with the new feature values. Maybe we will decide to remove the feature or maybe we will decide to create a separate model for the object with a new feature values. Sometimes there are duplicated numerical features that these two columns are completely identical. In our example data set, these columns f2 and f3. Obviously, we should leave only one of those two features since the other one will not give any new information to the model and will only slow down training. From a number of features, it's easy to check if two columns are the same. We just can compare them element wise. We can also have duplicated categorical features. The problem is that the features

can be identical but their levels have different names. That is it can be possible to rename levels of one of the features and two columns will become identical. For example features f4 and f5. If we rename levels of the feature f5, C to A, A to B, and B to C. The result will look exactly as feature f4. But how do we find such duplicated features? Fortunately, it's quite easy, it will take us only one more line of code to find them. We need to label and code all the categorical features first, and then compare them as if they were numbers. The most important part here is label encoding. We need to do it right. We need to encode the features from top to bottom so that the first unique value we see gets label 1, the second gets 2 and so on. For example for feature f4, we will encode A with 1, B with 2 and C with 3. Now feature f5 will encode it differently C will be 1, A will be 2 and B will be 3. But after such encodings columns f4 and f5 turn out to be identical and we can remove one of them. Another important thing to check is if there are any duplicated rows in the train and test. Is to write a lot of duplicated rows that also have different target, it can be a sign the competition will be more like a roulette, and our validation will be different to public leader board score, and private standing will be rather random. Another possibility, duplicated rows can just be the result of a mistake. There was a competition where one row was repeated 100,000 times in the training data set. I'm not sure if it was intentional or not, but it was necessary to remove those duplicated rows to have a high score on the test set. Anyway, it's better to explain it to ourselves why do we observe such duplicated rows? This is a part of data understanding in fact. We should also check if train and test have common rows. Sometimes it can tell us something about data set generation process. And again we should probably think what could be the reason for those duplicates? Another thing we can do, we can set labels manually for the test rows that are present in the train set. Finally, it is very useful to check that the data set is shuffled, because if it is not then, there is a high chance to find data leakage. We'll have a special topic about data leakages later, but for now we'll just discuss that the data is shuffled. What we can do is we can plot a feature or target vector versus row index. We can optionally smooth the values using running average. On this slide rolling target value from pairs competition is plotted while mean target value is shown with dashed blue line. If the data was shuffled properly

y we would expect some kind of oscillation of the target values around the mean target value. But in this case, it looks like the end of the train set is much different to the start, and we have some patterns. Maybe the information from this particular plot will not advance our model. But once again, we should find an explanation for all extraordinary things we observe. Maybe eventually, we will find something that will lead us to the first place. Finally, I want to encourage you one more time to visualize every possible thing in a dataset. Visualizations will lead you to magic features. So this is the last slide for this lesson. Hope you've learned something new and excited about it. Here's a whole list of topics we've discussed. You can pause this video and try to remember what we were talking about and where. See you later. [MUSIC][MUSIC] So in this video, I will go through Springleaf data, it was a competition on Kaggle. In that competition, the competitors were to predict whether a client will respond to direct mail offer provided by Springleaf. So presumably, we'll have some features about client, some features about offer, and we'll need to predict 1 if he will respond and 0 if he will not, so let's start. We'll first import some libraries in here, define some functions, it's not very interesting. And finally, let's load the data and train our test one, and do a little bit of data overview. So the first thing we want to know about our data is the shapes of data tables, so let's bring the train shape, and test that test shape. What we see here, we have one 150,000 objects, both in train and test sets, and about 2000 features in both train and test. And what we see more than, we have one more feature in train, and as humans, just target can continue to move the train. So we should just keep it in mind and be careful, and drop this column when we feed our models. So let's examine training and test, so let's use this function had to print several rows of both. We see here we have ID column, and what's interesting here is that I see in training we have values 2, 4, 5, 7, and in test we have 1, 3, 6, 9. And it seems like they are not overlapping, and I suppose the generation process was as following. So the organizers created a huge data set with 300,000 rules, and then they sampled at random, rows for the train and for the test. And that is basically how we get this train and test, and we have this column IG, it is row index in this original huge file. Then we have something categor

ical,
then something numeric, numeric again, categorical, then
something that can be numeric or binary. But you see has decimal
part,
so I don't know why, then some very strange values in here,
and again, something categorical. And actually,
we have a lot of in between, and yeah, we have target as the last
column
of the train set, so let's move on. Probably another thing we want
to
check is whether we have not a numbers in our data set, like non-
ce values,
and we can do it in several ways. And one way we, let's compute
how many NaNs are there for
each object, for each row. So this is actually what we do here,
and we print only the values for
the first 15 rows. And so the row 0 has 25 NaNs, row 1 has 19 NaNs,
and so on, but what's interesting here,
six rows have 24 NaNs. It doesn't look like we got it in random,
it's really unlikely to
have these at random. So my hypothesis could be that
the row order has some structure, so the rows are not shuffled,
and
that is why we have this kind of pattern. And that means that we
probably could use row index as another feature for
our classifier, so that is it. And the same, we can do with columns,
so for each column, let's compute how
many NaNs are there in each column. And we see that ID has 0 NaNs,
then some 0s, and then we see that a lot of
columns have the same 56 NaNs. And that is again something really
strange, so either every column will have 56 NaNs, and so it's not
magic,
it's probably just how the things go. But if we know that there
are a lot
of columns, and every column have more different number of NaNs,
then it's
really unlikely to have a lot of columns nearer to each other in
the data
set with the same number of NaNs. So probably, our hypothesis could
be here that the column order is not random, so
we could probably investigate this. So we have about 2,000
columns in this data, and it's a really huge number of columns.
And it's really hard to work
with this data set, and basically we don't have any names,
so the data is only mice. As I told you,
the first thing we can do is to determine the types of the data,
so we will do it here. So we're first going to continue train and
test on a huge data frame like the organizers had,
it will have 300,000 rows. And then we'll first use
a unique function to determine how many unique
values each column has. And basically here we bring
several values of what we found, and it seems like there are five

e columns that have only one unique number. So we can drop the, basically what we have here, we just find them in this line, and then we drop them. So next we want to remove duplicated features, but first, for convenience, fill not a numbers with something that we can find easily later, and then we do the following. So we create another data frame of size, of a similar shape as the training set. What we do we take a column from train set, we apply a label encoder, as we discussed in a previous video, and we basically store it in this new train set. So basically we get another data frame which is train, but label encoded train set. And having this data frame, we can easily find duplicated features, we just start iterating the features with two iterators. Basically, one is fixed and the second one goes from the next feature to the end. Then we try to compare the columns, the two columns that we're standing at, right. And if they are element wise the same, then we have duplicated columns, and basically that is how we fill up this dictionary of duplicated columns. We see it here, so we found that variable 9 is duplicated for input 8, and variable 18 again is duplicated for variable 8, and so on, and so we have really a lot of duplicates in here. So this loop, it took some time, so I prefer to dump the results to disk, so we can easily restore them. So I do it here, and then I basically drop those columns that we found from the train test data frame. So yeah, in the second video, we will go through some features and do some work to data set. [MUSIC] So, let's continue exploration. We wanted to determine the types of variables, and to do that we will first use this nunique function to determine how many unique values again our feature have. And we use this dropna=False to make sure this function computes and accounts for nons. Otherwise, it will not count null as unique value. It will just unhit them. So, what we see here that ID has a lot of unique values again and then we have not so huge values in this series, right? So I have 150,000 elements but 6,000 unique elements. 25,000, it's not that a huge number, right? So, let's aggregate this information and do the histogram of the values from above. And it's not histogram of these exact values but but it's normalized values. So, we divide each value by the number of rows in the tree. It's the maximum value of unique values we could possibly have. So what we see here that there are a lot of features that have a few unique values and there are several that have a lot, but not so much, not as much as these. So these features have almost in every row unique value. So, let's actually explore these. So, ID essentially is having a lot of unique values. No problem with that. But what is this? So what we actually see here, they are integers. They are huge numbers but they're integers. Well, I would expect a real, nunique variable with real values to have a lot of unique values, not integer type

e variable. So, what could be our guess what these variables represent? Basically, it can be a counter again. But what else it could be? It could be a time in let's say milliseconds or nanoseconds or something like that. And we have a lot of unique values and no overlapping between the values because it's really unlikely to have two events or two rows in our data set having the same time, let's say it's time of creation and so on, because the time precision is quite good. So yeah, that could be our guess. So next, let's explore this group of features. Again with some manipulations, I found them and these are presented in this table.

So, what's interesting about this? Actually, if you take a look at the names. So the first one is 541. And the second one is 543. Okay. And then we have 1,081 and 1,082, so you see they are standing really close to each other. It's really unlikely that half of the row, if the column order was random, if the columns were shuffled. So, probably the columns are grouped together according to something and we could explore this something. And what's more interesting, if we take a look at the values corresponding to one row, then we'll find that'll say this value is equal to this value. And this value is equal to this value and this value, and this is basically the same value that we had in here. So, we have five features out of four of this having the same value. And if you examine other objects, some of them will have the same thing happening and some will not. So, you see it could be something that is really essential to the objects and it could be a nice feature that separates the objects from each other. And, it's something that we should really investigate and where we should really do some feature engineering. So, for say [inaudible], it will be really hard to find those patterns. I mean, it cannot find. Well, it will struggle to find that two features are equal or five features are equal. So, if we create or say feature that will calculate how many features out of these, how many features we have have the same value say for the object zero where we'll have the value five in this feature and something for other rows, then probably this feature could be discriminative. And then we can create other features, say we set it to one if the values in this column, this and this and this and this are the same and zero to otherwise, and so on. And basically, if you go through these rows, you will find that the patterns are different and sometimes the values are the same in different columns. So for example, for this row, we see that this value is equal to this value. And this value is different to previous ones but its equal to this one. And it's really fascinating, isn't it? And if it actually will work and improve the model, I will be happy. And another thing we see here is some strange values and they look like nons. I mean, it's something that a human typed in or a machine just autofilled. So, let's go further. Oh, yeah. And the last thing is just try to pick one variable from this group and see what values does it have. So, let's pick variable 15 and here's its values. And minus 999 is probably how we've filled in the nons. And yeah, we have 56 of them and all other values are non-negative, so probably it's counters. I mean, how many events happened in, I don't know, in the month or something like that. Okay. And finally, let's filter the columns and then separate columns into categorical and numeric. And it's really easy to do using this function `select_dtypes`. Basically, all the columns that

will have objects type, if you would use a function `dtypes`. We think of them as categorical variables. And otherwise, if they are assigned type integer or float or something like that, or numeric type then we will think of these columns as numeric columns. So, we can go through the features one-by-one as actually I did during the competition. Well, we have 2,000 features in this data set and it is unbearable to go through a feature one-by-one.

I've stopped at about 250 features. And you can find in my notebook and reading materials if you're interested. It's a little bit messy but you can see it. So, what we will do here, just several examples of what I was trying to investigate in data set, let's do the following. Let's take the number of columns, we computed them previously. So, we'll now work with only the first 42 columns and we'll create such metrics. And it looks like correlation matrices and all of that type of matrices like when we have the features along the y axis, features along the x axis. Basically, well, it's really huge. Yeah. And in this case, what we'll have as the values is the number or the fraction of elements of one feature that are greater than elements of the second feature. So, for example, this cell shows that all variables or all values in variable 50 are less than values and variable ID, which is expected. So, yeah. And it's opposite in here. So, if we see one in here it means that variable 45, for example, is always greater than variable 24. And, while we expect this metrics to be somehow random, if the count order was random. But, in here we see, for example, these kind of square. It means that every second feature is greater, not to the second but let's say $i+1$ feature is greater than the feature i . And, well it could be that this information is about, for example, counters in different periods of time. So, for example, the first feature is how many events happened in the first month. The second feature is how many events happened in the first two month and so kind of cumulative values. And, that is why one feature is always greater than the other. And basically, what information we can extract from this kind of metrics is that we have this group and we can generate new features and these features could be, for example, the difference between two consecutive features. That is how we will extract, for example, the number of events in each month. So, we'll go from cumulative values back to normal values. And, well linear models, say, neural networks, they could do it themselves but tree-based algorithms they could not. So, it could be really helpful. So, in attached to non-book in the reading materials you will see that a lot of these kind of patterns. So, we have one in here, one in here. The patterns, well, this is also a pattern, isn't it? And now we will just go through several variables that are different. So, for example, variable two and variable three are interesting. If you build a histogram of them, you will see something like that. And, the most interesting part here are these spikes. And you see, again, they're not random. There's something in there. So, if we take this variable two and build there, we'll, use this value count's function, we'll have value and how many times it occurs in this variable. We will see that the values, the top values, are 12, 24, 36, 60 and so on. So, they can be divided by 12 and well probably, this variable is somehow connected to time, isn't it? To hours. Well, and what can we do? We want to generate features so we will generate feature like the val

ue of these variable modular 12 or, for example, value of this variable integer division by 12. So, this could really help. In other competition, you could build a variable and see something like that again. And what happened in there, the organizers actually had quantized data. So, they only had data that in our case could be divided by 12. Say 12, 24 and so on. But, they wanted to kind of obfuscate the data probably and they added some noise.

And, that is why if you plot an histogram, you will still see the spikes but you will also see something in between the spikes.

And so, again, these features in that competition they work quite well and you could dequantize the values and it could really help. And the same is happening with variable 3 basically, 0, 12, 24 and so on. And variable 4, I don't have any plot for variable 4 itself in here but actually we do the same thing. So, we take variable 4, we create a new feature variable 4 modulus 50. And now, we plot this kind of histogram. What you see here is light green, there are actually two histograms in there. The first one for object from the class 0 and the second one for the objects from class 1. And one is depicted with light green and the second one is with dark green. And, you see these other values. And, you see only difference in these bar, but, you see the difference. So, it means that these new feature variable 4 modulus 50 can be really discriminative when it takes the value 0. So, one could say that this is kind of, well, I don't know how to say that., I mean, certain people would never do that. Like, why do we want to take away modular 50? But, you see sometimes this can really help. Probably because organizers prepare the data that way. So, let's get through categorical features. We have actually not a lot of them. We have some labels in here, some binary variables. I don't know what is this, this is probably is some problems with the encoding I have. And then, we have some time variables. This is actually not a time. Time. Not a time. Not a time. This is time. Whoa, this is interesting. This looks like cities, right? Or towns, I mean, city names. And, if you remember what features we can generate from geolocation, it's the place to generate it. And, then again, it was some time, some labels and once again, it's the states. Isn't it? So, again, we can generate some geographic features. But particularly interesting, the features are the date. Dates that we had in here. And basically, these are all the columns that I found having the data information. So, it was one of the best features for this competition actually. You could do the following, you could do a scatter plot between two date features to particular date features and found that they have some relation, and, one is always greater than another.

It means that probably these are dates of some events and one event is happening always after the first one. So, we can extract different features like the difference between these two dates.

And in this competition, it really helped a lot. So, be sure to do exploratory data analysis and extract all the powerful features like that. Otherwise, if you don't want to look into the data, you will not find something like that. And, it's really interesting. So, thank you for listening. Hi, everyone. In this video,

I will tell you about the specifics of Numerai Competition that was held throughout year 2016. Note that Numerai organizers changed the format in 2017. So, the findings I'm going to read will not work on new data. Let's state the problem. Participants were

e solving a binary classification task on a data set with 21 anonymized numeric features. Unusual part is that both train and test data sets have been updating every week. Data sets were also shuffled column-wise. So it was like a new task every week. Pretty challenging. As it turned out, this competition had a data leak. Organizers did not disclose any information about the nature of data set. But allegedly, it was some time series data with target variable highly dependent on transitions between time points. Think of something like predicting price change in stock market here. Means that, if we knew true order or had timestamp variable, we could easily get nearly perfect score. And therefore, we had to somehow reconstruct this order. Of course, approximately. But even a rough approximation was giving a huge advantage over other participants. The first and most important step is to find a nearest neighbor for every point in a data set, and add all 21 features from that neighbor to original point. Simple logistic regression of those 42 features, 21 from original, and 21 from neighboring points, allowed to get into top 10 on the leaderboard. Of course, we can get better scores with some Hardcore EDA. Let's start exploring correlation metrics of new 21 features. If group features with highest correlation coefficient next to each other, we'll get a right picture. This picture can help us in two different ways. First, we can actually fix some column order. So, weekly column shuffling won't affect our models. And second, we can clearly notice seven groups with three highly correlated features in each of them. So, the data actually has some non-trivial structure. Now, let's remember that we get new data sets every week. What is more? Each week, train data sets have the same number of points. We can assume that there is some connection between consecutive data sets. This is a little strange because we already have a time series. So, what's the connection between the data from different weeks? Well, if we find nearest neighbors from every point in current data set from previous data set, and plot distance distributions, we can notice that first neighbor is much, much closer than the second. So, we indeed have some connection between consecutive data sets. And it looks like we can build a bijective mapping between them. But let's not quickly jump into conclusions and do more exploration. Okay. We found a nearest neighbor in previous data set. What if we examine the distances between the neighboring objects at the level of individual features? We clearly have three different groups of seven features. Now remember, the sorted correlation matrix? It turns out that each of three highly correlated features belong to a different group. A perfect match. And if we multiply seven features from the first group by three, and seven features from the second group by two in the original data set, recalculate nearest neighbor-based features within the data sets, and re-train our models, we'll get a nice improvement. So, after this magic multiplications, of course, I'd tried other constants, our order approximation became a little better. Great. Now, let's move to the true relation. New data, weekly updates, all of it was a lie. Remember, how we were calculating neighbors between consecutive data sets? Well, we can forget about consecutiveness. Calculate neighbors between current data set, and the data set from two weeks ago or two months ago. No matter what, we will be getting pretty much the same distances. Why? The simplest answer is

that the data actually didn't change. And every week, we were getting the same data, plus a little bit of noise. And thus, we could find nearest neighbor in each of previous data sets, and average them all, successfully reducing the variance of added noise. After averaging, true order approximation became even better. I have to say that a little bit of test data actually did change from time to time. But nonetheless, most of the roles migrated from week to week. Because of that, it was possible to probe the whole public leader board which helped even further, and so on, and so on. Of course, there are more details regarding that competition, but they aren't very interesting. I wanted to focus on the process of reverse engineering. Anyway, I hope you like this kind of detective story and realized how important exploratory data analysis could be. Thank you for your attention and always pay respect to EDA. This isn't the rare case in competitions when you see people jumping down on leaderboard after revealing private results. So, we ask ourselves, what is happening out there?

There are two main reasons for these jumps. First, competitors could ignore the validation and select the submission which scored best against the public leaderboard. Second, is that sometimes competitions have no consistent public/private data split or they have too little data in either public or private leaderboard. Well, we as participants, can't influence competitions organization. We can certainly make sure that we select our most appropriate submission to be evaluated by private leaderboard. So, the broad goal of next videos is to provide you a systematic way to set up validation in a competition, and tackle most common validation problems. Let's quickly overview of the content of the next videos. First, in this video, we will understand the concept of validation and overfitting. In the second video, we will identify the number of splits that should be done to establish stable validation. In the third video, we will go through most frequent methods which are used to make train/test split in competitions. In the last video, we will discuss most often validation problems. Now, let me start to explain the concept for validation for those who may never heard of it. In the nutshell, we want to check if the model gives expected results on the unseen data. For example, if you've worked in a healthcare company which goal is to improve life of patients, we could be given the task of predicting if a patient will be diagnosed a particular disease in the near future. Here, we need to be sure that the model we train will be applicable in the future. And not just applicable, we need to be sure about what quality this model will have depending on the number of mistakes the model make. And on the predictive probability of a patient having this particular disease, we may want to decide to run special medical tests for the patient to clarify the diagnosis. So, we need to correctly understand the quality of our model. But, this quality can differ on train data from the past and on the unseen test data from the future. The model could just memorize all patients from the train data and be completely useless on the test data because we don't want this to happen. We need to check the quality of the model with the data we have and these checks are the validation. So, usually, we divide data we have into two parts, train part and validation part. We fit our model on the train part and check its quality on the validation part. Beside that, in the last example, our model

will be checked against the unseen data in the future and actually these data can differ from the data we have. So we should be ready for this. In competitions, we usually have the similar situation. The organizers of a competition give us the data in two chunks. First, train data with all target values. And second, test data without target values. As in the previous example, we should split the data with labels into train and validation parts. Furthermore, to ensure the competition spirit, the organizers split the test data into the public test set and the private test set. When we sent our submissions to the platform, we see the scores for the public test set while the scores for the private test set are released only after the end of the competition. This also ensures that we don't need the test set or in terms of a model do not overfit. Let me draw you an analogy with the disease projection, if we already divided our data into train and validation parts. And now, we are repeatedly checking our model against the validation set, some models, just by chance, will have better scores than the others. If we continue to select best models, modify them, and again select the best from them, we will see constant improvements in the score. But that doesn't mean we will see these improvements on the test data from the future. By repeating this over and over, we could just achieve the validation set or in terms of a competition, we could just cheat the public leaderboard. But again, if it overfit, the private leaderboard will let us down. This is what we call overfitting in a competition. Get an unrealistically good scores on the public leaderboard that later result in jumping down the private leaderboard. So, we want our model to be able to capture patterns in the data but only those patterns that generalize well between both train and test data. Let me show you this process in terms of underfitting and overfitting. So, to choose the best model, we basically want to avoid underfitting on the one side and overfitting on the other. Let's understand this concept on a very simple example of a binary classification test. We will be using simple models defined by formulas under the pictures and visualize the results of model's predictions. Here on the left picture, we can see that if the model is too simple, it can't capture underlined relationship and we will get poor results. This is called underfitting. Then, if we want our results to improve, we can increase the complexity of the model and that will probably find that quality on the training data is going down. But on the other hand, if we make too complicated model like on the right picture, it will start describing noise in the train data that doesn't generalize the test data. And this will lead to a decrease of model's quality. This is called overfitting. So, we want something in between underfitting and overfitting here. And for the purpose of choosing the most suitable model, we want to be able to evaluate our results. Here, we need to make a remark, that the meaning of overfitting in machine learning in general and the meaning of overfitting competitions in particular are slightly different. In general, we say that the model is overfitted if its quality on the train set is better than on the test set. But in competitions, we often say, that the models are overfitted only in case when quality on the test set will be worse than we have expected. For example, if you train gradient boosting decision tree in the competition is our area under a curve metric. We sometimes can ob

serve that the quality on the training data is close to one while on the test data, it could be less for example, near 0.9. In general sense, the models overfitted here but while we get area under curve was 0.9 on both validation and public/private test sets, we will not say that it is overfitted in the context of a competition. Let me illustrate this concept again in a bit different way. So, let's say for the purpose of model evaluation, we divided our data into two parts. Train and validation parts. Like we already did, we will derive model's complexity from low to high and look at the models here. Note, that usually, we understand error or loss is something which is opposite to model's quality or score. In the figure, the dependency looks pretty reasonable. For two simple models, we have underfitting which means higher error on both train and validation. For two complex models, we have overfitting which means low error on train but again high error on validation. In the middle, between them, if the perfect model's complexity, it has the lowest error on the validation data and thus we expect it to have the lowest error on the unseen test data. Note, that here the training error is always better than the test error which implies overfitting in general sense, but doesn't apply in the context of competitions. Well done. In this video, we define validation, demonstrated its purpose, and interpreted validation in terms of underfitting and overfitting. So, once again, in general, the validation helps us answer the question, what will be the quality of our model on the unseen data and help us select the model which will be expected to get the best quality on that test data. Usually, we are trying to avoid underfitting on the one side that is we want our model to be expressive enough to capture the patterns in the data. And we are trying to avoid overfitting on the other side, and don't make too complex model, because in that case, we will start to capture noise or patterns that doesn't generalize to the test data.[SOUND] In the previous video, we understood that validation helps us select a model which will perform best on the unseen test data. But, to use validation, we first need to split the data with given labels, training, and validation parts. In this video, we will discuss different validation strategies. And answer the questions. How many splits should we make and what are the most often methods to perform such splits. Loosely speaking, the main difference between these validation strategies is the number of splits being done. Here I will discuss three of them. First is holdout, second is K-fold, and third is leave-one-out. Let's start with holdout. It's a simple data split which divides data into two parts, training data frame, and validation data frame. And the important note here is that in any method, holdout included, one sample can go either to train or to validation. So the samples between train and the validation do not overlap, if they do, we just can't trust our validation. This is sometimes the case, when we have repeated samples in the data. And if we are, we will get better predictions for these samples and more optimistic all estimation overall. It is easy to see that t

these can prevent us from selecting best parameters for our model. For example, over fitting is generally bad. But if we have duplicated samples that present, and train, and test simultaneously and over feed, validation scores can deceive us into a belief that maybe we are moving in the right direction. Okay, that was the quick note about why samples between train and validation must not overlap. Back to holdout. Here we fit our model on the training data frame, and evaluate its quality on the validation data frame. Using scores from this evaluation, we select the best model. When we are ready to make a submission, we can retrain our model on our data with given labels. Thinking about using holdout in the competition. It is usually a good choice, when we have enough data. Or we are likely to get similar scores for the same model, if we try different splits. Great, since we understood what holdout is, let's move onto the second validation strategy, which is called K-fold. K-fold can be viewed as a repeated holdout, because we split our data into key parts and iterate through them, using every part as a validation set only once. After this procedure, we average scores over these K-folds. Here it is important to understand the difference between K-fold and usual holdout or bits of K-times. While it is possible to average scores they receive after K different holdouts. In this case, some samples may never get invalidation, while others can be there multiple times. On the other side, the core idea of K-fold is that we want to use every sample for validation only once. This method is a good choice when we have a minimum amount of data, and we can get either a sufficiently big difference in quality, or different optimal parameters between folds. Great, having dealt with K-fold, we can move on to the third validation strategy in our release. It is called leave-one-out. And basically it is a special case of Kfold when K is equal to the number of samples in our data. This means that it will iterate through every sample in our data. Each time one sample came in a slot minus one object is a train subset and one object left is a test subset. This method can be helpful if we have too little data and just enough model to entrain. So that there, validation strategies. Holdout, K-fold and leave-one-out. We usually use holdout or K-fold on shuffle data. By shuffling data we are trying to reproduce random trained validation split. But sometimes, especially if you do not have enough samples for some class, a random split can fail. Let's consider, for an example. We have binary classification tests and a small data set with eight samples. Four of class zero, and four

r of class one. Let's split data into four folds. Done, but notice, we are not always getting 0 and 1 in the same problem. If we'll use the second fold for validation, we'll get an average value of the target in the train of two third instead of one half. This can drastically change predictions of our model. What we need here to handle this problem is stratification. It is just the way to insure we'll get similar target distribution over different faults. If we split data into four faults with stratification, the average of each false target values will be equal to one half. It is easier to guess that significance of this problem is higher, first for small data sets, like in this example, second for unbalanced data sets. And for binary classification, that could be, if target average were very close to 0 or vice versa, very close to 1. And third, for multiclass classification tasks with huge amount of classes. For good classification data sets, stratification split will be quite similar to a simple shuffle split. That is, to a random split. Well done, in this video we have discussed different validation strategies and reasons to use each one of them. Let's summarize this all. If we have enough data, and we're likely to get similar scores and optimal model's parameters for different splits, we can go with Holdout. If on the contrary, scores and optimal parameters differ for different splits, we can choose KFold approach. And even, if we too little data, we can apply leave-one-out. The second big takeaway from this video for you should be stratification. It helps make validation more stable, and especially useful for small and unbalanced datasets. Great. In the next videos we will continue to comprehend validation at it's core. [SOUND] [MUSIC] Since we already know the main strategies for validation, we can move to more concrete examples. Let's imagine, we're solving a competition with a time series prediction, namely, we are to predict a number of customers for a shop for which they're due in next month. How should we divide the data into train and validation here? Basically, we have two possibilities. Having data frame first, we can take random rows in validation and second, we can make a time-based split, take everything before some date as a train and everything out there as a validation. Let's plan these two options next. Now, when you think about features you need to generate and the model you need to train, how complicated these two cases are? In the first block, we can just interpret between the previous and the next value to get our predictions. Very easy, but wait. Do we really have future information about the number of customers in the real world? Well, probably not. But does this mean that this validation is useless? Again, it doesn't. What it does mean really that if we make train validation split different fr

om train/test split, then we are going to create a useless model. And here, we get to the main rule of making a reliable validation. We should, if possible, set up validation to mimic train/test split, but that's a little later. Let's go back to our example. On the second picture, for most of test point, we have neither the next value nor the previous one. Now, let's imagine we have a pool of different models trained on different features, and we selected the best model for each type of validation. Now, the question, will these models differ? And if they will, how significantly? Well, it is certain that if you want to predict what will happen a few points later, then the model which favor features like previous and next target values will perform poorly. It happens because in this case, we just don't have such observations for the test data. But we have to give the model something in the feature value, and it probably will be not numbers or missing values. How much experience that model have with these type of situations? Not much. The model just won't expect that and quality will suffer. Now, let's remember the second case. Actually, here we need to rely more on the time trend. And so, the features, which is the model really we need here, are more like what was the trend in the last couple of months or weeks? So, that shows that the model selected as the best model for the first type of validation will perform poorly for the second type of validation. On the opposite, the best model for the second type of validation was trained to predict many points ahead, and it will not use adjacent target values. So, to conclude this comparison, these models indeed differ significantly, including the fact that most useful features for one model are useless for another. But, the generated features are not the only problem here. Consider that actual train/test split is time-based, here is the question. If we carefully generate features that are drawing attention to time-based patterns, we'll get a reliable validation with a random-based split. Let me say this again in another words. If we'll create features which are useful for a time-based split and are useless for a random split, will be correct to use a random split to select the model? It's a tough question. Let's take a moment and think about it. Okay, now let's answer this. Consider the case when target falls a linear train. In the first block, we see the exact case of randomly chosen validation. In the second, we see the same time-based split as we consider before. first, let's notice that in general, model predictions will be close to targets mean value calculated using train data. So in the first block, if the validation points will be closer to this mean value compared to test points, we'll get a better score in validation than on test. But in the second case, the validation points are roughly as far as the test points from target mean value. And so, in the second case, validation score will be more similar to the test score. Great, as we just found out, in the case of incorrect validation, not only features, but the value target can lead to unrealistic estimation of the score. Now, that example was quite similar to what you may encounter while solving real competitions. Numerous competitions use time-based split namely: the Rossmann Store Sales competition, the Grupo Bimbo Inventory Demand competition and others. So, to quickly summarize this valuable example we just have discussed, different splitting strategies can differ significantly, namely: in generated features, in

the way the model will rely on that features, and in some kind of target leak. That means, to be able to find smart ideas for feature generation and to consistently improve our model, we absolutely want to identify train/test split made by organizers, including the competition, and reproduce it. Let's now categorize most of these splitting strategies and competitions, and discuss examples for them. Most splits can be united into three categories: a random split, a time-based split and the id-based split. Let's start with the most basic one, the random split. Let's start, the most common way of making a train/test split is to split data randomly by rows. This usually means that the rows are independent of each other. For example, we have a test of predicting if a client will pay off alone. Each row represents a person, and these rows are fairly independent of each other. Now, let's consider that there is some dependency, for example, within family members or people which work in the same company. If a husband can pay a credit probably, his wife can do it too. That means if by some misfortune, a husband will be present in the training data and his wife will be present in the test data. We probably can explore this and devise a special feature for that case. For in such possibilities, and realizing that kind of features is really interesting. More in this case and others I will mention here, comes in the next lesson of our course. So again, that was a random split. The second method is a time-based split. We already discussed the unit example of the split in the beginning of this video. In that case, we generally have everything before a particular date as a training data, and the rating after date as a test data. This can be a signal to use special approach to feature generation, especially to make useful features based on the target. For example, if we are to predict a number of customers for the shop for each day in the next week, we can come up with something like the number of customers for the same day in the previous week, or the average number of customers for the past month. As I mentioned before, this split is widespread enough. It was used in a Rossmann store sales competition and in the Grupo Bimbo inventory demand competition, and in other's competitions. A special case of validation for the time-based split is a moving window validation. In the previous example, we can move the date which divides train and validation. Successively using week after week as a validation set, just like on this picture. Now, having dealt with the random and the time-based splits, let's discuss the ID-based split. ID can be a unique identifier of user, shop, or any other entity. For example, let's imagine we have to solve a task of music recommendations for completely new users. That means, we have different sets of users in train and test. If so, we probably can make a conclusion that features based on user's history, for example, how many songs user listened in the last week, will not help for completely new users. As an example of ID-based split, I want to tell you a bit about the Caterpillar to pricing competition. In that competition, train/test split was done on some category ID, namely, tube ID. There is an interesting case when we should employ the ID-based split, but IDs are hidden from us. Here, I want to mention two examples of competitions with hidden ID-based split. These include Intel and MumbaiODT Cervical Cancer Screening competition, and The Nature Conservancy fisheries monitoring competition. In the first competi-

on, we had to classify patients into three classes, and for each patient, we had several photos. Indeed, photos of one patient belong to the same class. Again, sets of patients from train and test did not overlap. And we should also ensure these in the training regulations split. As another example, in The Nature Conservancy fisheries monitoring competition, there were photos of fish from several different fishing boats. Again, fishing boats and train and test did not overlap. So one could easily overfit if you would ignore risk and make a random-based split. Because the IDs were not given, competitors had to derive these IDs by themselves. In both these competitions, it could be done by clustering pictures. The easiest case was when pictures were taken just one after another, so the images were quite similar. You can find more details of such clustering in the kernels of these competitions. Now, having in these two main standalone methods, we also need to know that they sometimes may be combined. For example, if we have a task of predicting sales in a shop, we can choose a split in date for each shop independently, instead of using one date for every shop in the data. Or another example, if we have search queries from multiple users, is using several search engines, we can split the data by a combination of user ID and search engine ID. Examples of competitions with combined splits include the Western Australia Rental Prices competition by Deloitte and their qualification phase of data science game 2017. In the first competition, train/test was split by a single date, but the public/private split was made by different dates for different geographic areas. In the second competition, participants had to predict whether a user of online music service will listen to the song. The train/test split was made in the following way. For each user, the last song he listened to was placed in the test set, while all other songs were placed in the train set. Fine. These were the main splitting strategies employed in the competitions. Again, the main idea I want you to take away from this lesson is that your validation should always mimic train/test split made by organizers. It could be something non-trivial. For example, in the Home Depot Product Search Relevance competition, participants were asked to estimate search relevancy. In general, data consisted of search terms and search results for those terms, but test set contained completely new search terms. So, we couldn't use either a random split or a search term-based split for validation. First split favored more complicated models, which led to overfitting while second split, conversely, to underfitting. So, in order to select optimal models, it was crucial to mimic the ratio of new search terms from train/test split. Great. This is it. We just demonstrated major data splitting strategies employed in competitions. Random split, time-based split, ID-based split, and their combinations. This will help us build reliable validation, make a useful decisions about feature generation, and in the end, select models which will perform best on the test data. As the main point of this video, remember the general rule of making a reliable validation. Set up your validation to mimic the train/test split of the competition.[SOUND] Hi and welcome back. In the previous videos we discussed the concept of validation and overfitting. And discussed how to chose validation strategy based on the properties of data we have. And

finally we learned to identify data split made by organizers. After all this work being done, we honestly expect that the relation will, in a way, substitute a leaderboard for us. That is the score we see on the validation will be the same for the private leaderboard. Or at least, if we improve our model and validation, there will be improvements on the private leaderboard. And this is usually true, but sometimes we encounter some problems here. In most cases these problems can be divided into two big groups. In the first group are the problems we encounter during local validation. Usually they are caused by inconsistency of the data, a widespread example is getting different optimal parameters for different faults. In this case we need to make more thorough validation. The problems from the second group, often reveal themselves only when we send our submissions to the platform. And observe that scores on the validation and on the leaderboard don't match. In this case, the problem usually occurs because we can't mimic the exact train test split on our validation. These are tough problems, and we definitely want to be able to handle them. So before we start, let me provide an overview of this video. For both validation and submission stages we will discuss main problems, their causes, how to handle them. And then, we'll talk a bit about when we can expect a leaderboard shuffle. Let's start with discussion of validation stage problems. Usually, they attract our attention during validation. Generally, the main problem is a significant difference in scores and optimal parameters for different train validation splits. Let's start with an example. So we can easily explain this problem. Consider that we need to predict sales in a shop in February. Say we have target values for the last year, and, usually, we will take last month in the validation. This means January, but clearly January has much more holidays than February. And people tend to buy more, which causes target values to be higher overall. And that means squared error of our predictions for January will be greater than for February. Does this mean that the model will perform worse for February? Probably not, at least not in terms of overfitting. As we can see, sometimes this kind of model behavior can be expected. But what if there is no clear reason why scores differ for different folds? Let's identify several common reasons for this and see what we can do about it. The first hypotheses we should consider is that we have too little data. For example, consider a case w

When we have a lot of patterns and trends in the data. But we do not have enough samples to generalize these patterns well. In that case, a model will utilize only some general patterns. And for each train validation split, these patterns will partially differ. This indeed, will lead to a difference in scores of the model. Furthermore, validation samples will be different each time only increasing the dispersion of scores for different folds. The second type of this, is data is too diverse and inconsistent. For example, if you have very similar samples with different target variance, a model can confuse them. Consider two cases, first, if one of such examples is in the train while another is in the validation. We can get a pretty high error for the second sample. And the second case, if both samples are in validation, we will get smaller errors for them. Or let's remember another example of diverse data we have already discussed a bit earlier. I'm talking about the example of predicting sales for January and February. Here we have the nature or the reason for the differences in scores. As a quick note, notice that in this example, we can reduce this diversity a bit if we will validate on the February from the previous year. So the main reasons for a difference in scores and optimal model parameters for different folds are, first, having too little data, and second, having too diverse and inconsistent data. Now let's outline our actions here. If we are facing this kind of problem, it can be useful to make more thorough validation. You can increase K in KFold, but usually 5 folds are enough. Make KFold validation several times with different random splits. And average scores to get a more stable estimate of model's quality. The same way we can choose the best parameters for the model if there is a chance to overfit. It is useful to use one set of KFold splits to select parameters and another set of KFold splits to check model's quality. Examples of competitions which required extensive validation include the Liberty Mutual Group Property Inspection Prediction competition and the Santander Customer Satisfaction competition. In both of them, scores of the competitors were very close to each other. And thus participants tried to squeeze more from the data. But do not overfit, so the thorough validation was crucial. Now, having discussed validation stage problems, let's move on to submission stage problems. Sometimes you can diagnose these problems

in the process of doing careful. But still, often you encounter these type of problems only when you submit your solution to the platform. But then again, is your friend when it comes down to finding the root of the problem. Generally speaking, there are two cases of these issues. In the first case, leaderboard score is consistently higher or lower than validation score. In the second, leaderboard score is not correlated with validation score at all. So in the worst case, we can improve our score on the validation. While, on the contrary, score on the leaderboard will decrease. As you can imagine, these problems can be much more trouble. Now remember that the main rule of making a reliable validation, is to mimic a train tests pre made by organizers. I won't lie to you, it can be quite hard to identify and mimic the exact train tests here. Because of that, I highly you to start submitting your solutions right after you enter the competition. It's good to start exploring other possible roots of this problem. Let's first sort out causes we could observe during validation stage. Recall, we already have different model scores on different folds during validation. Here it is useful to see a leaderboard as another validation fold. Then, if we already have different scores in KFold, getting a not very similar result on the leaderboard is not surprising. More we can calculate mean and standard deviation of the validation scores and estimate if the leaderboard score is expected. But if this is not the case, then something is definitely wrong. There could be two more reasons for this problem. The first reason, we have too little data in public leaderboard, which is pretty self explanatory. Just trust your validation, and everything will be fine. And the second train and test data are from different distributions. Let me explain what I mean when I talk about different distributions. Consider a regression test of predicting people's height by their photos on Instagram. The blue line represents the distribution of heights for men, while the red line represents the distribution of heights for women. As you can see, these distributions are different. Now let's consider that the training data consists only of women, while the test data consists only of men. Then all model predictions will be around the average height for women. And the distribution of these predictions

will be very similar to that for the train data. No wonder that our model will have a terrible score on the test data. Now, because our course is a practical one, let's take a moment and think what you can do if you encounter these in a competition. Okay, let's start with a general approach to such problems. At the broadest level, we need to find a way to tackle different distributions in train and test. Sometimes, these kind of problems could be solved by adjusting your solution during the training procedure. But sometimes, this problem can be solved only by adjusting your solution through the leaderboard. That is through leaderboard probing. The simplest way to solve this particular situation in a competition is to try to figure out the optimal constant prediction for train and test data. And shift your predictions by the difference. Right here we can calculate the average height of women from the train data. Calculating the average height of men is a bit trickier. If the competition's metric is means squared error, we can send two constant submissions, write down the simple formula. And find out that the average target value for the test is equal to 70 inches. In general, this technique is known as leaderboard probing. And we will discuss it in the topic about. So now we know the difference between the average target values for the train and the test data, which is equal to 7 inches. And as the third step of adjusting our submission to the leaderboard we could just try to add 7 to all predictions. But from this point it is not validational it is a leaderboard probing and list. Yes we probably could discover this during exploratory data analysis and try to make a correction in our validation scheme. But sometimes it is not possible without leaderboard probing, just like in this example. A competition which has something similar is the Quora question pairs competition. There, distributions of the target from train and test were different. So one could get a good improvement of a score adjusting his predictions to the leaderboard. But fortunately, this case is rare enough. More often, we encounter situations which are more like the following case. Consider that now train consists not only of women, but mostly of women, and test, vice versa. Consists not only of men, but mostly of men. The main strategy to deal with these kind of situations is simple. Again, remember to mimic the train test split. If the test consists mostly of Men, force the validation to have the same distribution. In that case, you ensure that your validation will be fair. This is true for getting raw scores and

optimal parameters correctly. For example, we could have quite different scores and optimal parameters for women's and men's parts of the data set. Ensuring the same distribution in test and validation helps us get scores and parameters relevant to test. I want to mention two examples of this here. First the Data Science Game Qualification Phase: Music recommendation challenge. And second, competition with CTR prediction which we discussed earlier in the data topic. Let's start with the second one, do you remember the problem, we have a test of predicting CTR. So, the train data, which basically was the history of displayed ads obviously didn't contain ads which were not shown. On the contrary, the test data consisted of every possible ad. Notice this is the exact case of different distributions in train and test. And again, we need to set up our validation to mimic test here. So we have this huge bias towards showing that in the train and to set up a correct validation. We had to complete the validation set with rows of not shown ads. Now, let's go back to the first example. In that competition, participants had to predict whether a user will listen to a song recommended by assistant. So, the test contained only recommended songs. But train, on the contrary, contained both recommended songs and songs users selected themselves. So again, one could adjust his validation by 50 renowned songs selected by users. And again, if we will not account for that fact, then improving our model on actually selected songs can result in the validation score going up. But it doesn't have to result and the same improvements for the leaderboard. Okay let's conclude this overview of handling validation problems for the submission stage. If you have too little data in public leaderboard, just trust your validation. If that's not the case, make sure that you did not overfit. Then check if you made correct train/test split, as we discussed in the previous video. And finally, check if you have different distributions in train and test. Great, let's move on to the next point of this video. For now, I hope you did everything all right. First, you did extensive validation. Second, you choose a correct splitter strategy for train validation split. And finally, you ensured the same distributions in validation and testing. But sometimes you have to expect leaderboard shuffle anyway, and not just for you, but for everyone. First, for those who've never heard of it, a leaderboard shuffle happens when participants position some public and

private leaderboard drastically different. Take a look at this screenshot from the two sigma financial model in challenge competition. The green and the red arrows mean how far a team moved. For example, the participant who finished the 3rd on the private leaderboard was the 392nd on the public leaderboard. Let's discuss three main reasons for that shuffle, randomness, too little data, and different public, private distributions. So first, randomness, this is the case when all participants have very similar scores. This can be either a very good score or a very poor one. But the main point here is that the main reason for differences in scores is randomness. To understand this a bit more, let's go through two quick examples here. The first one is the Liberty Mutual Group, Property Inspection Prediction competition. In that competition, scores of competitors were very close. And though randomness didn't play a major role in that competition, still many people overfit on the public leaderboard. The second example, which is opposite to the first is the TWO SIGMA Financial Model and Challenge competition. Because the financial data in that competition was highly unpredictable, randomness played a major role in it. So one could say that the leaderboard shuffle there was among the biggest shuffles on KFold platform. Okay, that was randomness, the second reason to expect leaderboard shuffle is too little data overall, and in private test set especially. An example of this is the Restaurant Revenue Prediction Competition. In that competition, trained set consisted of less than 200 gross. And this set consisted of less than 400 gross. So as you can see shuffle here was more than expected. Last reason of leaderboard shuffle could be different distributions between public and private test sets. This is usually the case with time series prediction, like the Rossmann Stores Sales competition. When we have a time based split, we usually have first few weeks in public leaderboard, and next few weeks in private leaderboards. As people tend to adjust their submission to public leaderboard and overfit, we can expect worse results on private leaderboard. Here again, trust your validation and everything will be fine. Okay, let's go over reasons for leaderboard shuffling. Now let's conclude both this video and the entire validation topic. Let's start with the video. First, if you have big dispersion of scores on validation stage we should do extensive validation. That means every score from different KFold splits, and team model on one split while

evaluating score on the other. Second, if submission do not match local validation score, we should first, check if we have too little data in public leaderboard. Second, check if we did not overfit, check if you chose correct splitting strategy. And finally, check if training and test data have different distributions. You can expect leaderboard shuffle because of three key things, randomness, little amount of data, and different public/private test distributions. So that's it, in this topic we defined validation and its connection to overfitting. Described common validation strategies. Demonstrated major data splitting strategies. And finally analyzed and learned how to tackle main validation problems. Remember this, and it will absolutely help you out in competitions. Make sure you understand the main idea of validation well. That is, you need to mimic the trained test split. [MUSIC]

Hi everyone. In this section, we will talk about a very sensitive topic data leakage or more simply, leaks. We'll define leakage in a very general sense as an unexpected information in the data that allows us to make unrealistically good predictions. For the time being, you may have think of it as of directly or indirectly adding ground truths into the test data. Data leaks are very, very bad. They are completely unusable in real world. They usually provide way too much signal and thus make competitions lose its main point, and quickly turn them into a leak hunt increase. People often are very sensitive about this matter. They tend to overreact. That's completely understandable. After spending a lot of time on solving the problem, a sudden data leak may render all of that useless. It is not a pleasant position to be in. I cannot force you to turn the blind eye but keep in mind, there is no ill intent whatsoever. Data leaks are the result of unintentional errors, accidents. Even if you find yourself in a competition with an unexpected data leak close to the deadline, please be more tolerant. The question of whether to exploit the data leak or not is exclusive to machine learning competitions. In real world, the answer is obviously a no, nothing to discuss. But in a competition, the ultimate goal is to get a higher leaderboard position. And if you truly pursue that goal, then exploit the leak in every way possible. Further in this section, I will show you the main types of data leaks that could appear during solving a machine learning problem. Also focus on a competition specific leak exploitation technique leaderboard probing. Finally, you will find special videos dedicated to the most interesting and non-trivial data leaks. I will start with the most typical data leaks that may occur in almost every problem. Time series is our first target. Typically, future picking. It is common sense not to pick into the future like, can we use stock market's price from day after tomorrow to predict price for tomorrow? Of course not. However, direct usage of future information in incorrect time splits still exist. When you enter a time series competition at first, check train, public, and private splits. If even one of them is not on time, then you found a data leak. In such case, unrealistic

features like prices next week will be the most important. But even when split by time, data still contains information about future. We still can access the rows from the test set. We can have future user history in CTR task, some fundamental indicators in stock market predictions tasks, and so on. There are only two ways to eliminate the possibility of data leakage. It's called competitions, where one can not access rows from future or a test set with no features at all, only IDs. For example, just the number and instrument ID in stock market prediction, so participants create features based on past and join them themselves. Now, let's discuss something more unusual. Those types of data leaks are much harder to find. We often have more than just train and test files. For example, a lot of images or text in archive. In such case, we can't access some meta information, file creation date, image resolution etcetera. It turns out that this meta information may be connected to target variable. Imagine cats versus dogs classification. What if cat pictures were taken before dog? Or taken with a different camera? Because of that, a good practice from organizers is to erase the meta data, resize the pictures, and change creation date. Unfortunately, sometimes we will forget about it. A good example is Truly Native competition, where one could get nearly perfect scores using just the dates from zip archives. Another type of leakage could be found in IDs. IDs are unique identifiers of every row usually used for convenience. It makes no sense to include them into the model. It is assumed that they are automatically generated. In reality, that's not always true. ID may be a hash of something, probably not intended for disclosure. It may contain traces of information connected to target variable. It was a case in Caterpillar competition. A link ID as a feature slightly improve the result. So I advise you to pay close attention to IDs and always check whether they are useful or not. Next is row order. In trivial case, data may be shuffled by target variable. Sometimes simply adding row number or relative number, suddenly improves this course. Like, in Telstra Network Disruptions competition. It's also possible to find something way more interesting like in TalkingData Mobile User Demographics competition. There was some kind of row duplication, rows next to each other usually have the same label. This is it with a regular type of leaks. To sum things up, in this video, we embrace the concept of data leak and cover data leaks from future picking, meta data, IDs, and row order. [SOUND] Now, I will tell you about a competition-specific technique tightly connected with data leaks. It's leaderboard probing. There are actually two types of leaderboard probing. The first one is simply extracting all ground truth from public part of the leaderboard. It's usually pretty harmless, only a little more of straining data. It is also a relatively easy to do and I have a submission change on the small set of rows so that you can unambiguously calculate ground truth for those rows from leaderboard score. I suggest checking out the link to Alek Trott's post in additional materials. He thoroughly explains how

to do it very efficiently with minimum amount of submissions. Our main focus will be on another type of leaderboard probing. Remember the purpose of public, private split. It's supposed to protect private part of test set from information extraction. It turns out that it's still vulnerable. Sometimes, it's possible to submit predictions in such a way that will give out information about private data. It's all about consistent categories. Imagine, a chunk of data with the same target for every row. Like in the example, rows with the same IDs have the same target. Organizers split it into public and private parts. But we still know that that particular chunk has the same label for every row. After setting all the predictions close to 0 in our submission for that particular chunk of data, we can expect two outcomes. The first one is when score improved, it means that ground truth in public is 0. And it also means that ground truth in private is 0 as well. Remember, our chunk has the same labels. The second outcome is when the score became worse. Similarly, it means that ground truth in both public and private is 1. Some competitions indeed have that kind of categories. Categories that with high certainty have the same label. You could have encountered those type of categories in Red Hat and West Nile competitions. It was a key for winning. With a lot of submissions, one can explore a good part of private test set. It's probably the most annoying type of data leak. It's mostly technical and even if it's released close to the competition deadline, you simply won't have enough submissions to fully exploit it. Furthermore, this is on the tip of the iceberg. When I say consistent category, I do not necessarily mean that this category has the same target. It could be consistent in different ways. The definition is quite broad. For example, target label could simply have the same distribution for public and private parts of data. It was the case in Quora Question Pairs competition. In that competition there was a binary classification task being evaluated by log loss metric. What's important target were able had different distributions in train and test, but allegedly the same and private and public parts of these data. And because of that, we could benefit a lot via leaderboard probing. Treating the whole test set as a consistent category. Take a look at the formula on the slide. This logarithmic loss for submission with constant predictions C big. Where N big is the real number of rows, N_1 big is the number of rows with target one. And L big is the leader board score given by that constant prediction. From this equation, we can calculate N_1 divided by N or in other words,

the true ratio of once in the test set. That knowledge was very beneficial. We could use it rebalance training data points to have the same distribution of target variable as in the test set. This little trick gave a huge boost in leaderboard score. As you can see, leaderboard probing is a very serious problem that could occur under a lot of different circumstances. I hope that someday it will become completely eradicated from competitive machine learning. Now, finally, I like to briefly walk through the most peculiar and interesting competitions with data leakage. And first, let's take a look at Truly Native competition from different point of view. In this competition, participants were asked to predict whether the content in an HTML file is sponsored or not. As was already discussed in previous video, there was a data leak in archive dates. We can assume that sponsored and non-sponsored HTML files were gotten during different periods of time. So do we really get rid of data leak after erasing archive dates? The answer is no. Texts in HTML files may be connected to dates in a lot of ways. From explicit timestamps to much more subtle things, like news contents. As you've probably already realized, the real problem was not metadata leak, but rather data collection. Even without meta-information, machine learning algorithms will focus on actually useless features. The features that only act as proxies for the date. The next example is Expedia Hotel Recommendations, and that competition, participants worked with logs of customer behavior. These include what customers searched for, how they interacted with search results, and clicks or books, and whether or not the search result was a travel package. Expedia was interested in predicting which hotel group a user is going to book. Within the logs of customer behavior, there was a very tricky feature. At distance from users seeking their hotel. Turned out, that this feature is actually a huge data leak. Using this distance, it was possible to reverse engineer two coordinates, and simply map ground truth from train set to the test set. I strongly suggest you to check out the special video dedicated to this competition. I hope that you will find it very useful because the approaches and methods of exploiting data leaks were extremely nontrivial. And you will find a lot of interesting tricks in it. The next example is from Flavours of Physics competition. It was a pretty complicated problem dealing with physics at Large Hadron Collider. The special thing about

that competition was that signal was artificially simulated. Organizers wanted a machine learning solution for something that has never been observed. That's why the signal was simulated. But simulation cannot be perfect and it's possible to reverse engineer it. Organizers even created special statistical tests in order to punish the models that exploit simulation flaws. However, it was in vain. One could bypass the tests, fully exploit simulation flaws, and get a perfect score on the leaderboard. The last example is going to cover pairwise tasks. Where one needs to predict whether the given pair of items are duplicates or not, like in Quora question pairs competition. There is one thing common to all the competitions with pairwise tasks. Participants are not asked to evaluate all possible pairs. There is always some nonrandom subsampling, and this subsampling is the cause of data leakage. Usually, organizers sample mostly hard-to-distinguish pairs. Because of that, of course, imbalance in item frequencies. It results in more frequent items having the higher possibility of being duplicates. But that's not all. We can create a connectivity matrix N times N , where N is the total number of items. If item I and item J appeared in a pair then we place 1 in I, J and J, I positions. Now, we can treat the rows in connectivity matrix as vector representations for every item. This means that we can compute similarities between those vectors. This trick works for a very simple reason. When two items have similar sets of neighbors they have a high possibility of being duplicates. This is it with data leaks. I hope you got the concept and found a lot of interesting examples. Thank you for your attention. [SOUND] Hi, everyone. In this video, I will tell you how I and my teammates, Stanislav Smirnov solved Kaggle Expedia hotel recommendations competition. Personally, one of my favorites, probably among top five most interesting competitions I've ever participated in. I'll state the problem now. So, if you came here right after Data Leaks lesson, it should already be familiar to you. Anyway, in that competition, we worked with lots of customer behavior. These include what customers searched for, how they interacted with search results, clicks or books, and whether or not the search result was a travel package, and Expedia was interested in predicting which hotel group a user is going to book. Important thing here is prediction target the hotel group. In other words, characteristics of actual hotel, remember it. As it turned out, this competition had a very non-trivial and extremely hard to exploit data leak. From the first glance, data leak was pretty straightforward. We had a destination distance among the feature. It's a distance from user city to an actual hotel he clicked on booked. And, as I said earlier, our prediction target is a characteristic of an actual hotel. Furthermore, destination distance was very precise so unique user city and destination dist

ance pairs corresponded to unique hotels. Putting two and two together, we can treat user city and destination distance pair as a proxy to our target. When in this set, we encountered such pair already present in train set, we could simply take a label from there as our prediction. It worked nearly perfect for the pairs present in both train and test. However, nearly half of test set consisted from new pairs without a match from train set. This way we had to go deeper. But, how exactly can we improve our solution? Well, there are two different ways. First, one is to create current features on corteges similar to user city and destination distance pair. For example, like how many hotels of which group there are for user city, hotel country, hotel city triplet. Then, we could train some machine learning model on such features. Another way is to somehow find more matches. For that purpose, we need to find true coordinates of users cities and hotel cities. From that, to guess it was destination distance feature, it was possible to find good approximation for the coordinates of actual hotels. Let's find out how to do it. First of all, we need to understand how to calculate the distance. Here, we work with geographical coordinates so the distances are geodesic. It's done via Haversine formula, not a pleasant one. Now, suppose that we know true coordinates of three points and distances from fourth point with unknown coordinates to each of them, if you write down a system of three equations, one for each distance, we can unambiguously solve it and get true coordinates for the fourth point. Now, we have four points with known coordinates. I think you get the idea. So, at first, by hook or by crook, we reverse engineer true coordinate of three big cities. After that, we can iteratively find coordinates of more and more cities. But as you can see from the picture, some cities ended up in oceans. It means that our algorithm is not very precise. A rounding error accumulates after every iteration and everything starts to fall apart. We get some different method and indeed we can do better. Just compare this picture with the previous one. It's obviously much more accurate. Remember how in iterative method we solved a system of three equations to unambiguously find coordinates of fourth unknown point. But why limit ourselves with three equations? Let's create a giant system of equations from all known distances with true coordinates being done on variables. We end up with literally hundreds or thousands of equations and tens of thousands of unknown variables. Good thing it's very sparse. We can apply special methods from SciPy to efficiently solve such a system. In the end, after solving that system of equations, we end up with a very precise coordinates for both hotel cities and user cities. But as you remember, we're predicting a type of a hotel. Using city coordinates and destination distance, it's possible to find an approximation of true coordinates of an actual hotel. When we fix user city and draw a circumference around it with the radius of destination distance, it's obvious that true hotel location must be somewhere on that circumference. Now, let's fix some hotel city and draw such circumferences from all users cities to that fixed hotel cities and draw them for every given destination distance. After doing so, we end up with pictures like the ones on the slide. A city contains a limited number of hotels so the intuition here is that hotels actually are on the intersection points and the more circumferences intersect in such

point, the higher the probability of a hotel being in that point. As you can see, the pictures are beautiful but pretty messy. It's impossible to operate in terms of singular points. However, there are explicit clusters of points and this information can be of use. We can do some kind of integration. For every city, let's create a grid around its center. Something like 10 kilometers times 10 kilometers with step size of 100 meters. Now, using training data, for every cell in the grid, we can count how many hotels of which type are present there. If a circumference goes through a cell, we give plus one to the hotel type corresponding to that circumference. During inference, we also draw a circumference based on destination distance feature. We see from what degree its cells it went through and use information from those cells to create features like a sum of all counters, average of all counters, maximum of all counters and so on. Great. We have covered the part of feature engineering. Note that all the features directly used target label. We cannot use them as is in training. We should generate them in out-of-fold fashion for training data. So we had training data for years 2013 and 2014. To generate features for year 2014, we used labelled data from year 2013 and vice versa, used the year 2014 to generate features for the year 2013. For the test features, which was from year 2015, we naturally used all training data. In the end, we calculated a lot of features and serve them into Xgboost model. After 16 hours of training for the course, we got our results. We ended up on third position on public leader-boards and forth on private. We did good, but we still did not fully exploit data leakage. If you check the leaderboard, you'll notice the difference in scores between first place and the rest. Under speculation, the winner did extraordinary. Although, in general, his methods were very similar to ours. He was able to extract way more signal. Finally, I hope you enjoyed my story. As you can see, sometimes working with data leakage could be very interesting and challenging. You may develop some unusual skills and broaden your horizons. Thank you for your attention.