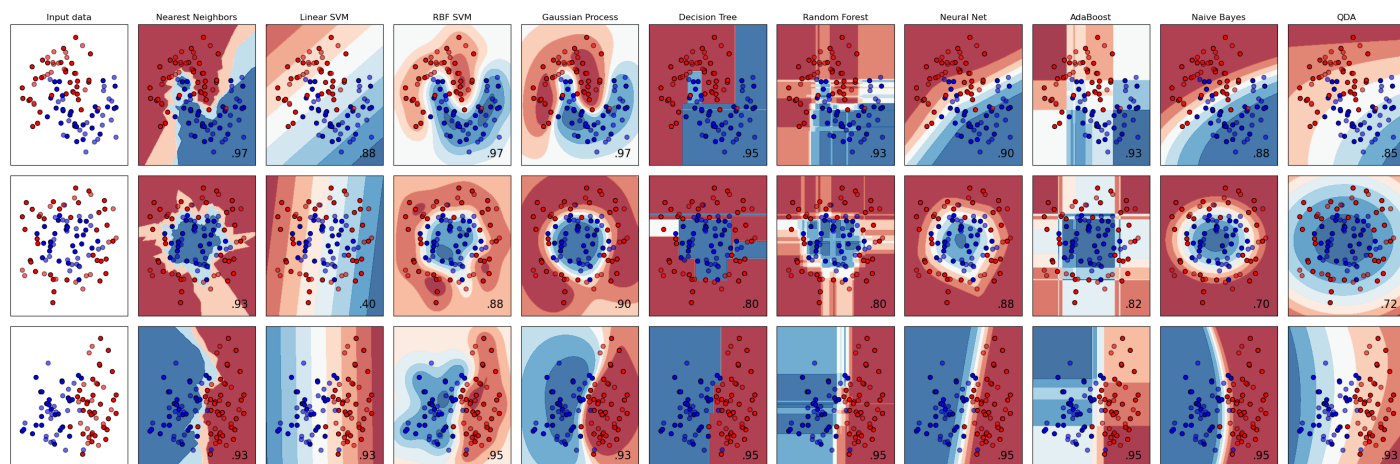


▼ Lab 4 - Machine Learning 1 - Classification

This lab will introduce the basic concepts behind machine learning and the tools that allow us to learn from data.

Machine learning is one of the main topics in modern AI and is used for many exciting applications that we will see in the coming weeks.



▼ Review

So far the two libraries that we have covered are Pandas (Week 2) which handles data frames.

```
import pandas as pd
```

And Altair (Week 3) which handles data visualization.

```
import altair as alt
```

We will continue building on these two libraries throughout the semester.

Let's start from a simple dataframe. We have been mainly loading data frames from files, but we can also create them directly from python dictionaries.

```

u = {
    "City" : ["New York", "Philadelphia", "Boston"],
    "Temperature" : [25.3, 30.1, 22.1],
    "Location" : [20.0, 15.0, 25.0],
    "Population" : [10000000, 1000000, 500000],
}
df = pd.DataFrame(d)
df

```

	City	Temperature	Location	Population
0	New York	25.3	20.0	10000000
1	Philadelphia	30.1	15.0	1000000
2	Boston	22.1	25.0	500000

Here are our columns.

```
df.columns
```

```
Index(['City', 'Temperature', 'Location', 'Population'], dtype='object')
```

We can make a graph by converting our dataframe into a chart . Remember we do this in three steps.

Charting

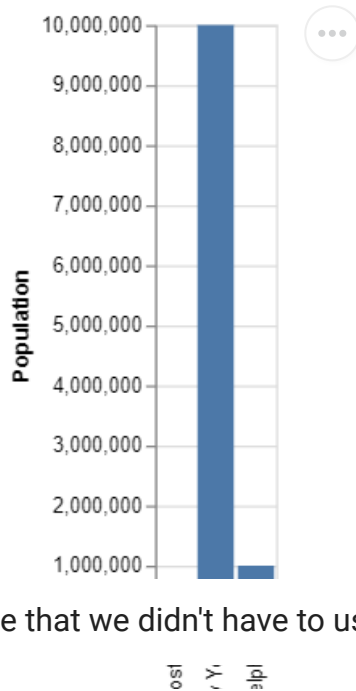
1. Chart - Convert a dataframe to a chart
2. Mark - Determine which type of chart we want
3. Encode - Say which columns correspond to which dimensions

One example is a bar chart.

```

chart = (alt.Chart(df)
        .mark_bar()
        .encode(x = "City",
                y = "Population"))
chart

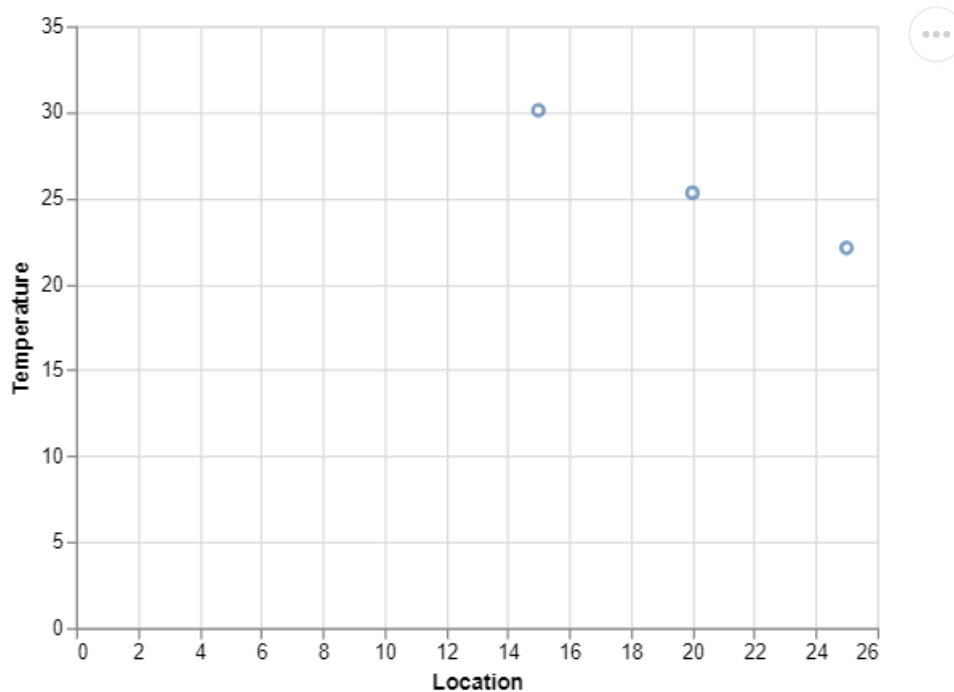
```



Notice that we didn't have to use all the columns, and it only showed the ones we specified.

Another example is a chart that shows the location and the temperature.

```
chart = (alt.Chart(df)
        .mark_point()
        .encode(x = "Location",
                y = "Temperature"))
chart
```

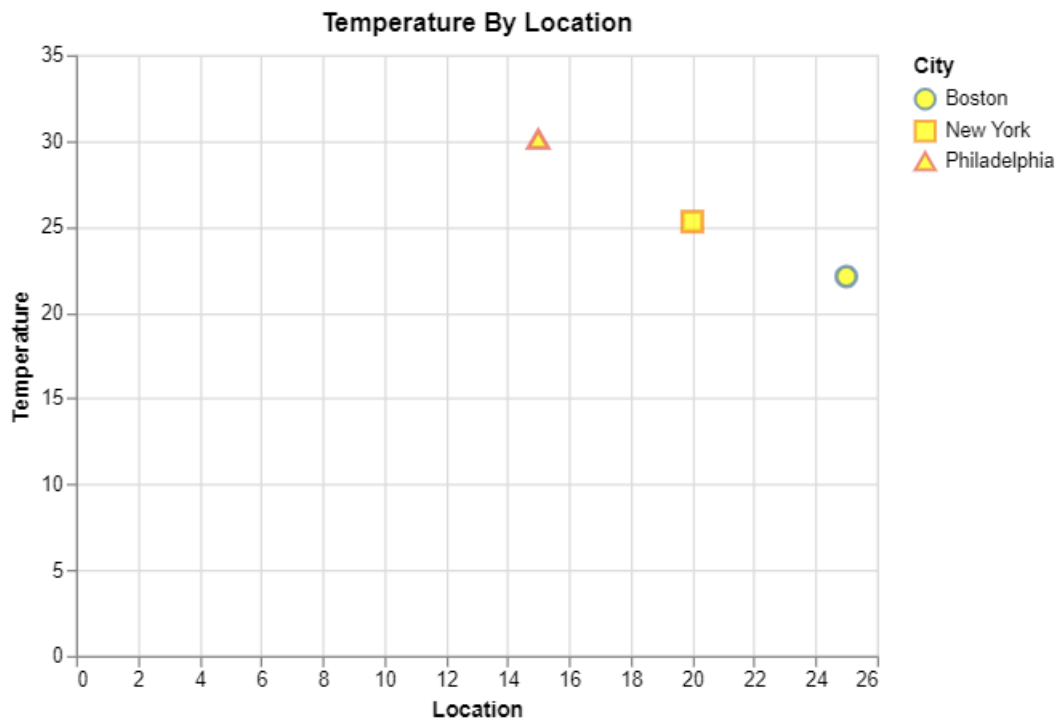


The library allows us to add special features. For instance, we can add a "Tooltip" where a mouse tells us which city it is.

```

chart = (alt.Chart(df)
        .mark_point(size=100, fill="yellow")
        .properties(title="Temperature By Location")
        .encode(x = "Location",
                y = "Temperature",
                shape = "City",
                color = "City",
                tooltip = ["City", "Temperature"])
        ))
chart

```



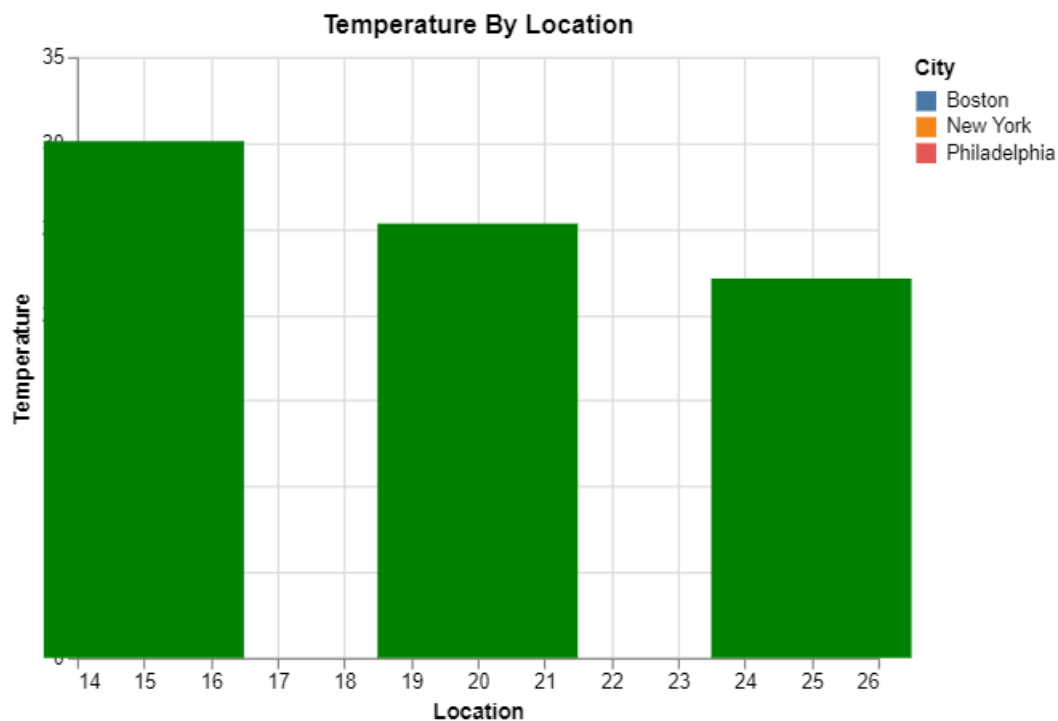
▼ Review Exercise

Make a bar chart that shows each city with its temperature and a tooltip of the city name.

```

# FILLME
chart = (alt.Chart(df)
        .mark_bar(size=100, fill="green")
        .properties(title="Temperature By Location")
        .encode(x = "Location",
                y = "Temperature",
                shape = "City",
                color = "City",
                tooltip = ["City", "Temperature"])
        ))
chart

```



▼ Unit A

▼ Machine Learning Data

For today's class we are going to take a break from our climate change data and work with a simplified set of starter data.

Our dataset is a Red versus Blue classification challenge. Let us take a look.

```
df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/simple.csv")  
df
```

	class	split	feature1	feature2
0	blue	train	0.368232	0.447353
1	blue	train	0.574324	0.382358
2	red	train	0.799023	0.849630
3	blue	train	0.778323	0.104591
4	red	train	0.824153	0.989757

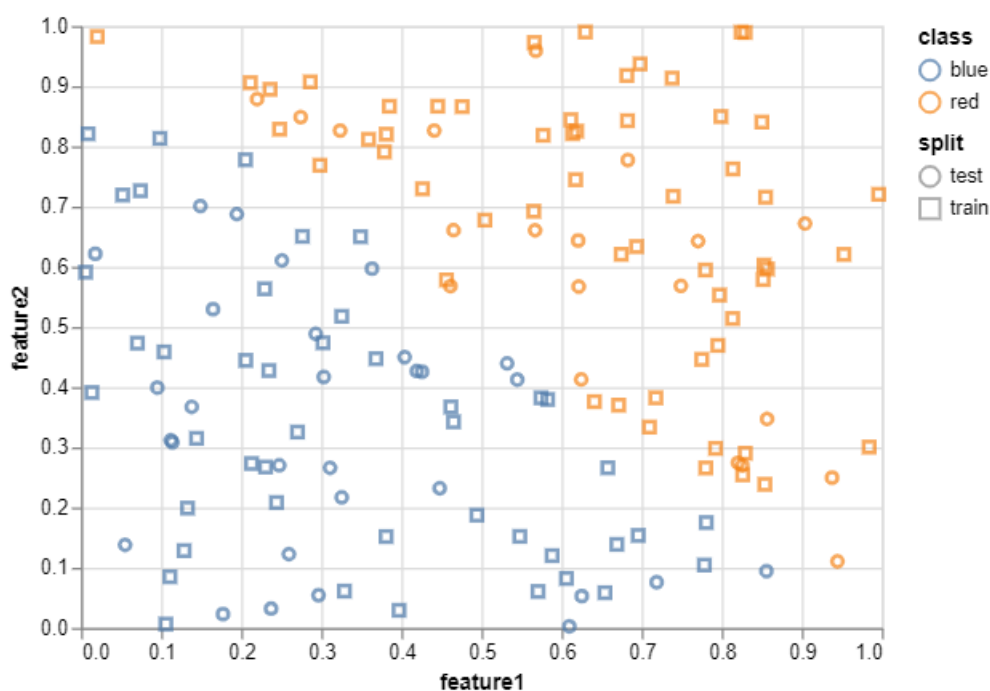
The first thing to do is to look at the columns.

```
df.columns
```

```
Index(['class', 'split', 'feature1', 'feature2'], dtype='object')
```

Here is what the data looks like.

```
chart = (alt.Chart(df)
        .mark_point()
        .encode(
            x = "feature1:Q",
            y = "feature2:Q",
            color = "class:N",
            shape = "split:N",
            tooltip = "class:N"
        ))
chart
```



First is `split`.

```
splits = df["split"].unique()  
splits  
  
array(['train', 'test'], dtype=object)
```

The two options here are `train` and `test`. This is an important distinction in machine learning.

- Train -> Points that we use to fit our machine learning model.
- Test -> Points that we use to predict with our machine learning model.

For example, if we were building a model for classifying types of birds from images, our Train split might be pictures of birds from a guide, whereas our Test split would be new pictures of birds in the wild that we want to classify.

Let us separate these out using a filter.

```
df_train = df.loc[df["split"] == "train"]  
df_test = df.loc[df["split"] == "test"]
```

Next is `class`.

```
classes = df_train["class"].unique()  
classes  
  
array(['blue', 'red'], dtype=object)
```

We can see there are two options, `red` and `blue`. This tells us the color associated with the point. For this exercise, our goal is going to be splitting up these two colors.

Finally we have `features`.

```
features = df_train[["feature1", "feature2"]].describe()  
features
```

	feature1	feature2
count	100.000000	100.000000
mean	0.508042	0.535118
std	0.269423	0.285280
min	0.005639	0.006402
25%	0.274920	0.296329

Features are the columns that we use in order to solve the challenge. The machine learning model gets to use the features in any way it wants in order to predict the class.

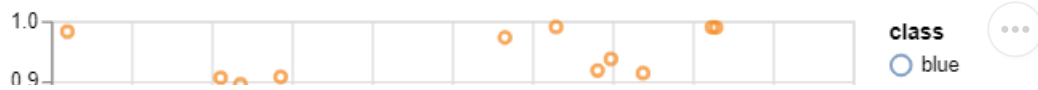
max 0.996007 0.999137

Let us now put everything together to draw a graph.

Charting

1. Chart - Just our training split.
2. Mark - Point mark to show each row
3. Encode - The features and the class.

```
chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class"
        ))
chart
```

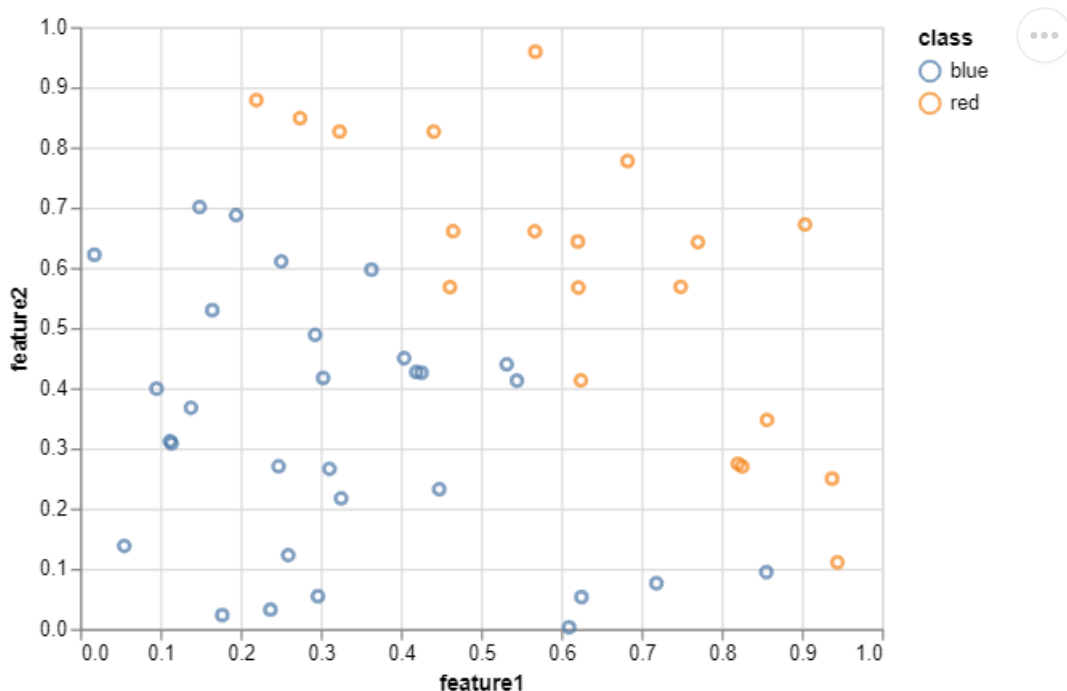
We can see that for this example the colors are split into two sides of the chart. Blue is at the bottom-left and red is at the top-right.



We can also look at the test split. The test split consists of the additional challenge points that our model needs to get correct. These points will follow a similar pattern, but have different features.



```
chart = (alt.Chart(df_test)
    .mark_point()
    .encode(
        x = "feature1",
        y = "feature2",
        color = "class"
    ))
chart
```



▼ Prediction

We are interested in using the features to predict the class (red/blue). We can do this by writing a function.

```
def predict(point):
    if point["feature1"] > 0.5:
        return "red"
```

```
else:  
    return "blue"
```

```
pt = {"feature1" : 0.3, "feature2" : 0.7}  
predict(pt)  
  
'blue'
```

We can apply this function using a variant of `map` from Module 1. The `apply` command will call our prediction for each point in test.

```
df_test["predict"] = df_test.apply(predict, axis=1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>
""Entry point for launching an IPython kernel.



```
df_test
```

	class	split	feature1	feature2	predict
100	blue	test	0.718804	0.075953	red
101	red	test	0.441072	0.826450	blue
102	red	test	0.820350	0.274501	red
103	red	test	0.620815	0.643749	red
104	blue	test	0.363353	0.596951	blue
105	blue	test	0.325463	0.216925	blue
106	red	test	0.465097	0.660846	blue
107	blue	test	0.194541	0.687478	blue
108	blue	test	0.250895	0.610575	blue
109	red	test	0.621408	0.567207	red
110	blue	test	0.177323	0.022961	blue
111	red	test	0.749021	0.568431	red
112	blue	test	0.310921	0.266022	blue
113	blue	test	0.447874	0.232126	blue
114	blue	test	0.296579	0.054368	blue
115	red	test	0.856608	0.347288	red
116	red	test	0.219821	0.878575	blue
117	red	test	0.624578	0.412857	red
118	red	test	0.682782	0.777404	red
119	blue	test	0.625190	0.052891	red
120	blue	test	0.855877	0.094495	red
121	blue	test	0.054964	0.137895	blue
122	red	test	0.566922	0.660628	red
123	red	test	0.770472	0.642557	red
124	blue	test	0.112091	0.312023	blue
125	blue	test	0.138122	0.367348	blue
126	red	test	0.944554	0.110551	red
127	blue	test	0.609665	0.002666	red
128	blue	test	0.425742	0.425577	blue
129	red	test	0.461263	0.567964	blue
130	blue	test	0.419118	0.426836	blue

131	blue	test	0.237319	0.032165	blue
132	blue	test	0.095227	0.399256	blue
133	red	test	0.274217	0.848751	blue
134	blue	test	0.544664	0.412633	red

Once we have made predictions, we can compute a score for how well our prediction did. We do this by comparing the `predict` with `class`.

```
correct = (df_test["predict"] == df_test["class"])
df_test["correct"] = correct
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>

Let us see how well we did. This graph puts everything together.

145	blue	test	0.259535	0.122557	blue
------------	------	------	----------	----------	------

```
chart = (alt.Chart(df_test)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class",
            fill = "predict",
            shape = "correct",
            tooltip = ["correct"]
        ))
chart
```



The outline of the point is blue / red based on the true class. Whereas the fill tells us our prediction. Mousing over the points will tell us whether they are correct or not.



Student question: How well did our predictions do?



```
# FILLME
```

```
df_test["correct"].mean()
```

0.76

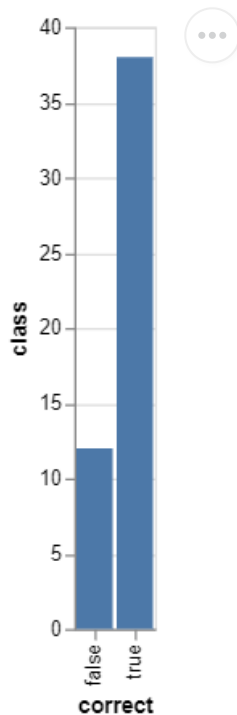
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

```
out = df_test.groupby(["correct"], as_index=False).count()
```

```
chart = (alt.Chart(out)
        .mark_bar()
        .encode(
            x="correct",
            y="class"
```

```
))
```





chart



▼ Group Exercise A

▼ Question 0

Who are other members of your group today?





```
# FILLME
Palak Shah, Sahar Sami and Aisha Bashir
dtypes = {
    "names": "Aisha Bashir", "Palak Shah", "Sahar Sami"
}
```

File "[<ipython-input-34-96a728b16baf>](#)", line 2
 Palak Shah, Sahar Sami and Aisha Bashir
 ^





SyntaxError: invalid syntax

SEARCH STACK OVERFLOW

What is something they are great at cooking?





```
# FILLME
dtypes = {
    "food": "indian food"
}
```

What are their favorite animals?

```
# FILLME
dtypes = {
    "favorite_animals": "labradors"
}
```

▼ Question 1

The `predict` function above is not able to fully separate the points into red/blue groups. Can you write a new function that gets all of the points correct?

```
# FILLME
def my_predict(point):
    if point['feature1'] + point['feature2'] > 1:
```

```

    return "red"
    return "blue"
df_test["predict"] = df_test.apply(predict, axis=1)

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>

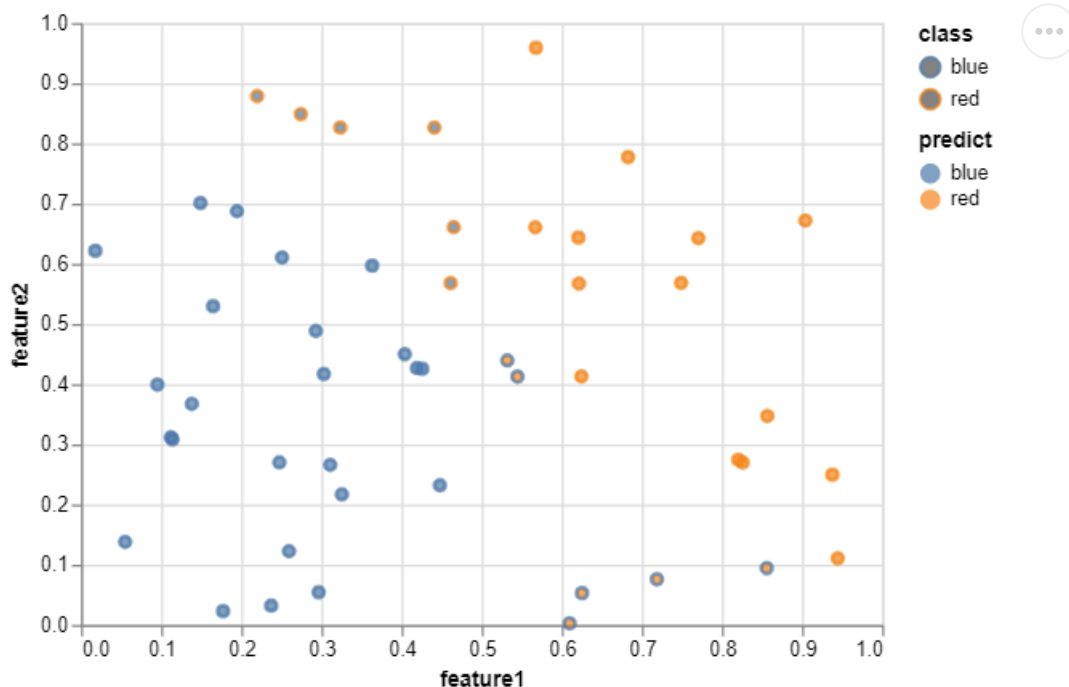


Redraw the graph above to show that you split up the points correctly.

```

# FILLME
chart = (alt.Chart(df_test)
    .mark_point()
    .encode(
        x = "feature1",
        y = "feature2",
        color = "class",
        fill = "predict",
    ))
chart

```



▼ Question 2

The dataset above is a bit easy. It seems like you can just separate the points with a line.

Next let us consider a harder example where the red and blue points form a circle.

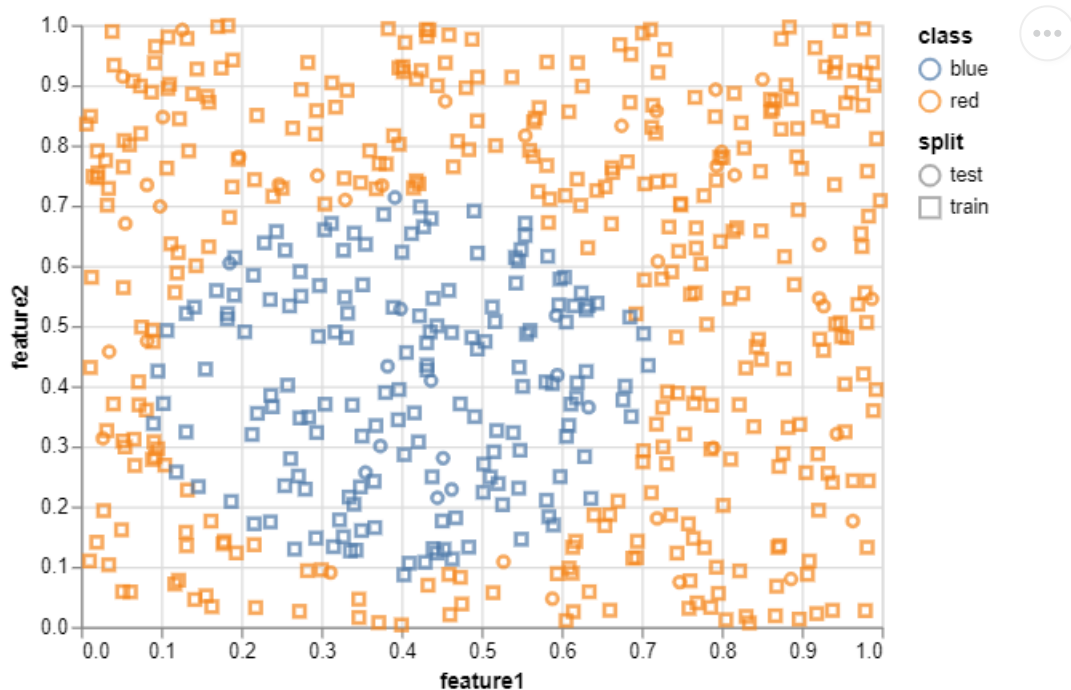
```
df2 = pd.read_csv("https://srush.github.io/BT-AI/notebooks/circle.csv")
```

```
df2_train = df2.loc[df2["split"] == "train"]
```

```
df2_test = df2.loc[df2["split"] == "test"]
```

Draw a chart with these points.

```
# FILLME
chart = (alt.Chart(df2)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class",
            shape = "split")
        )
chart
```



▼ Question 3

Try to write a function that separates the blue and the red points. How well can you do?

FILLME

def my_circle_predict(point):

if (point['feature1'] - 0.4)**2 + (point['feature2'] - 0.4)**2 > 0.05:

return "red"

return "blue"

pass

df2_test["predict"] = df2_test.apply(predict, axis=1)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>
import sys

Redraw the graph above to show that you split up the points correctly.

FILLME

chart = (alt.Chart(df2_test)

.mark_point()

.encode(

x = "feature1",

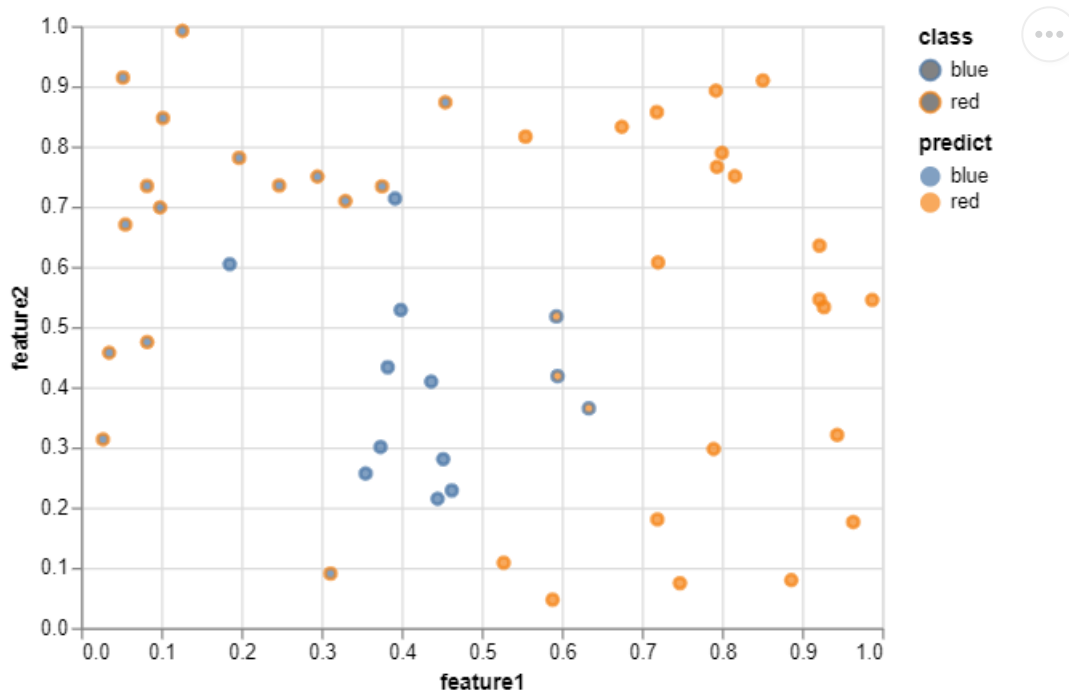
y = "feature2",

color = "class",

fill = "predict",

))

chart



▼ Unit B

Machine learning (ML) is a collection of method for automatically finding a way to split up points. It sounds really simple, but it turns out that splitting up points like this is really useful.

To explore this idea we are going to use the library Scikit-Learn. This is a standard toolkit for machine learning in Python.



One warning. The documentation for Scikit-Learn is a bit intimidating. If you look something up it might appear like this.

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Do not be scared though. Most of these options do not matter so much in practice. You can learn the important parts in 30 minutes.

Let us first import the library.

```
import sklearn.linear_model
```

We are going to use this formula for all our machine learning.

Model Fitting

1. Dataframe. Create your training data (This part you are an expert in!)
2. Fit. Create a model and give it training features
3. Predict. Use the model on test data.

Step 1. Create out data. (We did this already).

```
df_train
```

	class	split	feature1	feature2
0	blue	train	0.368232	0.447353
1	blue	train	0.574324	0.382358
2	red	train	0.799023	0.849630
3	blue	train	0.778323	0.104591
4	red	train	0.824153	0.989757
...
95	blue	train	0.325493	0.517772
96	blue	train	0.328865	0.061240
97	red	train	0.792355	0.298344
98	red	train	0.850158	0.840475
99	red	train	0.385003	0.866745

Step 2. Create our model and fit it to data.

First we pick a model type. We will mostly use this one.

```
sklearn.linear_model.LogisticRegression
```

```
sklearn.linear_model._logistic.LogisticRegression
```

However I really hate the name *logistic regression*. So let us rename this function to what it really is.

```
LinearClassification = sklearn.linear_model.LogisticRegression
model = LinearClassification()
```

Then we tell it which features to use as input (X) and what it goal is (y). Here we tell it to use `feature1` and `feature2` and to predict whether the point is `red`.

```
model.fit(X=df_train[["feature1", "feature2"]],
          y=df_train["class"] == "red")

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

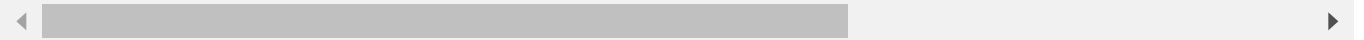
This is similar to Altair chart. Just tell it which columns to use.

Step 3. Predict. Once we have a model we can use it to predict the output classes of our model. This replaces the part where we did it manually.

```
df_test["predict"] = model.predict(df_test[["feature1", "feature2"]])
```

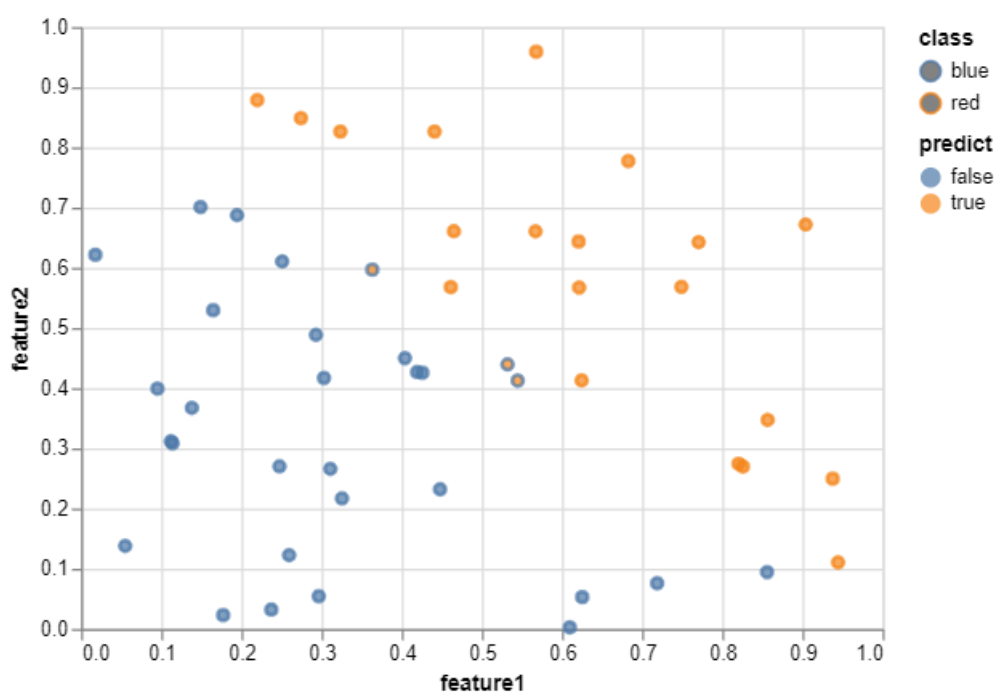
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>
 """Entry point for launching an IPython kernel.



We can see the graph that came out.

```
chart = (alt.Chart(df_test)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class",
            fill = "predict",
            tooltip = ["correct"]
        ))
chart
```



That's it! You have just built your first machine learning model.

▼ Details

What happened? How did the system know whether the output points would be red or blue?

The key idea is that behind the scenes the model uses the training data to learn a class for every possible point.

For instance, if we make up a feature value.

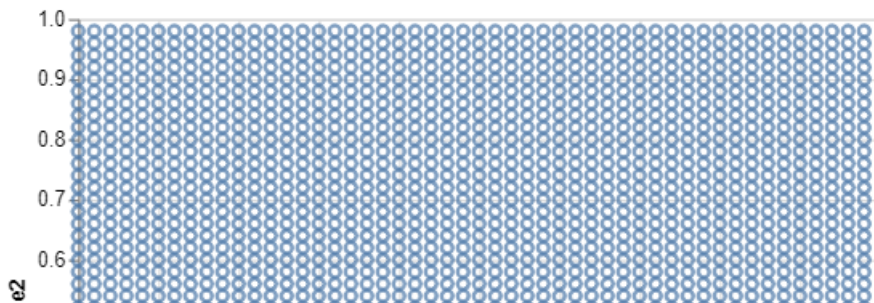
```
feature1 = 0.2  
feature2 = 0.5
```

Our model will produce an output prediction.

```
predict = model.predict([[feature1, feature2]])  
predict  
  
array([False])
```

In fact, we can even see what the model would do for any point. This dataframe has all possible points.

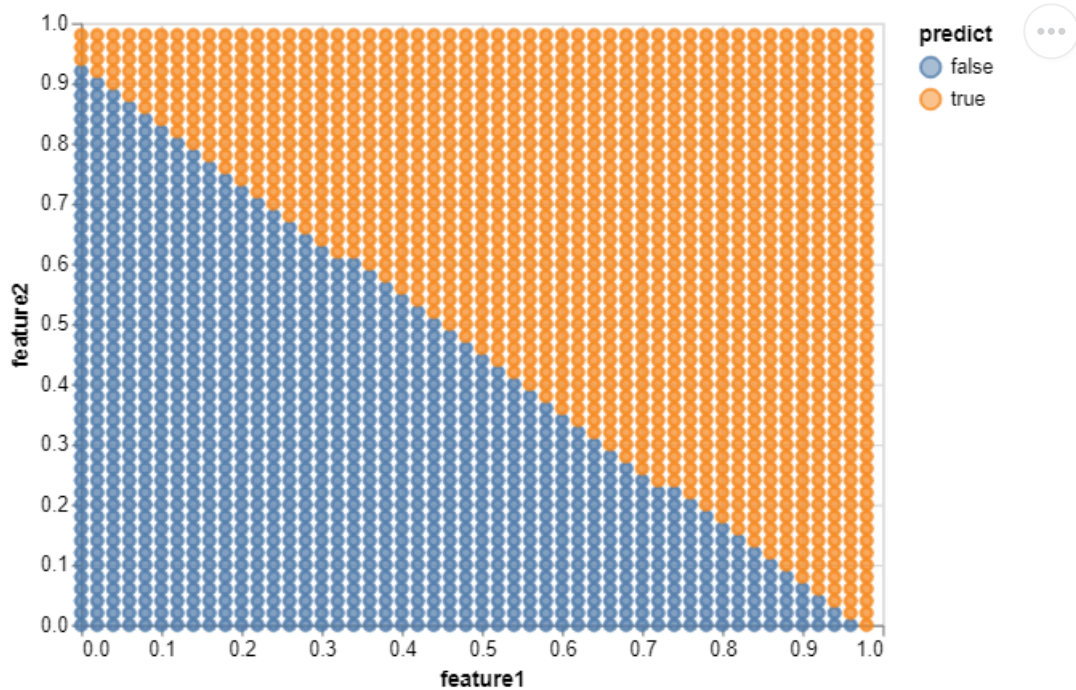
```
all_df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/all_points.csv")  
chart = (alt.Chart(all_df)  
        .mark_point()  
        .encode(  
            x = "feature1",  
            y = "feature2",  
        ))  
chart
```



Let us see what our model would do on each of them.

```
all_df["predict"] = model.predict(all_df[["feature1", "feature2"]])
```

```
chart = (alt.Chart(all_df)
    .mark_point()
    .encode(
        x = "feature1",
        y = "feature2",
        color="predict",
        fill = "predict",
    ))
chart
```



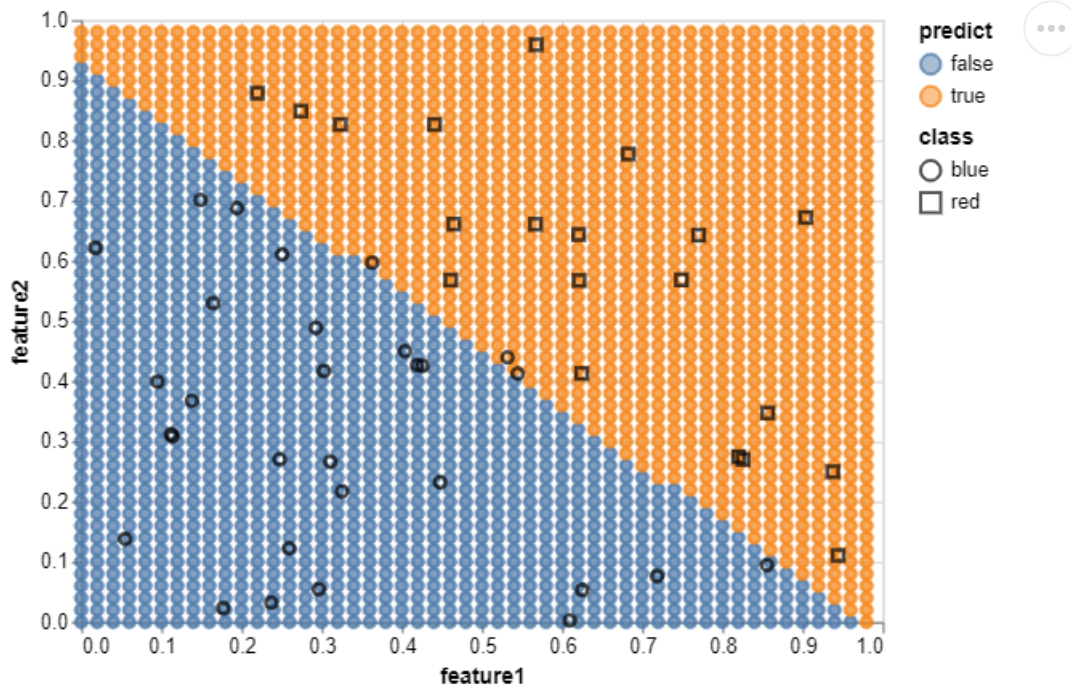
This makes sense as it corresponds to the same line that we saw on our original chart.

```
chart2 = (alt.Chart(df_test)
    .mark_point(color="black")
    .encode(
        x = "feature1",
```

```

    y = "feature2",
    shape = "class",
))
chart = chart + chart2
chart

```



▼ Other Data.

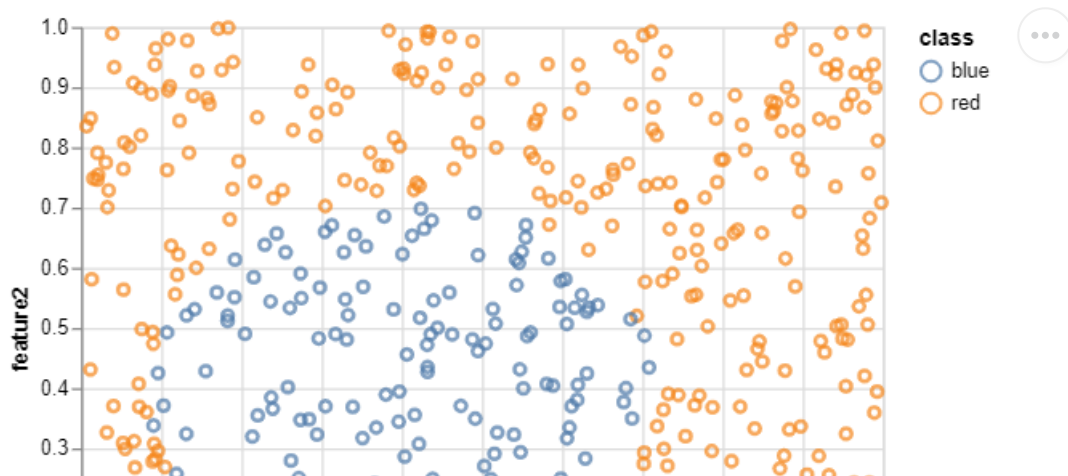
So is machine learning magic? Can we just give any data and have it learn a separator for us?

Well let's try the circle dataset.

```

chart = (alt.Chart(df2_train)
    .mark_point()
    .encode(
        x = "feature1",
        y = "feature2",
        color = "class",
    ))
chart

```



First we fit.

```
model.fit(X=df2_train[["feature1", "feature2"]],
          y=df2_train["class"] == "red")
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

Then we predict.

```
df2_test["predict"] = model.predict(df2_test[["feature1", "feature2"]])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>
 """Entry point for launching an IPython kernel.

df2_test

	class	split	feature1	feature2	predict
500	blue	test	0.594712	0.418443	True
501	red	test	0.102121	0.847092	True
502	red	test	0.082418	0.474960	True
503	red	test	0.197240	0.780904	True
504	red	test	0.789593	0.297249	True
505	blue	test	0.462743	0.228536	True
506	red	test	0.126399	0.991937	True
507	blue	test	0.391861	0.713452	True
508	blue	test	0.633829	0.364800	True
509	blue	test	0.382835	0.433166	True
510	red	test	0.527502	0.108264	True
511	red	test	0.719390	0.180239	True
512	red	test	0.793643	0.765896	True
513	red	test	0.052198	0.914220	True
514	red	test	0.963775	0.176075	True
515	blue	test	0.437035	0.409321	True
516	red	test	0.375655	0.733664	True
517	red	test	0.675004	0.832558	True
518	blue	test	0.185455	0.604339	True
519	blue	test	0.355151	0.256612	True
520	blue	test	0.451964	0.280519	True
521	red	test	0.329907	0.709427	True
522	red	test	0.927210	0.533049	True
523	red	test	0.027352	0.313297	False
524	red	test	0.886499	0.079478	True
525	red	test	0.921658	0.635214	True
526	red	test	0.098637	0.698637	True
527	red	test	0.792375	0.892601	True
528	red	test	0.295041	0.749920	True
529	red	test	0.747440	0.074464	True
530	blue	test	0.399063	0.528276	True

531	blue	test	0.373780	0.300631	True
532	red	test	0.921765	0.545698	True
533	red	test	0.816113	0.750664	True
534	red	test	0.588545	0.046874	True
535	red	test	0.055169	0.670168	True
536	red	test	0.035069	0.457245	False
537	red	test	0.851052	0.909606	True
538	red	test	0.943769	0.320757	True
539	red	test	0.554810	0.816288	True
540	red	test	0.718627	0.857093	True
541	red	test	0.799872	0.789397	True
542	blue	test	0.444945	0.214648	True
543	blue	test	0.503204	0.517170	True

Finally we graph.

```
all_df["predict"] = model.predict(all_df[["feature1", "feature2"]])
```

```
531    blue    test    0.373780    0.300631    True
```

```
chart = (alt.Chart(all_df)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color="predict",
            fill = "predict",
        ))
chart
```



Unfortunately this result no good. The model did not learn about the circle. In fact it learned something completely wrong.



We can debug the problem by looking at how we created our model.



This line of code, says create `Linear` model. Linear in this case implies that the model can only use a line to split the points.



```
model = LinearClassification()
```

This model couldn't even learn about the circle if it wanted to.

In the group exercise you will explore some other possible models that can get around some of these limitations.

▼ Group Exercise B

▼ Question 1

The linear model we used above could only draw lines to separate red and blue. Let us consider a new model.

```
import sklearn.neighbors
neighbor_model = sklearn.neighbors.KNeighborsClassifier(1)
```

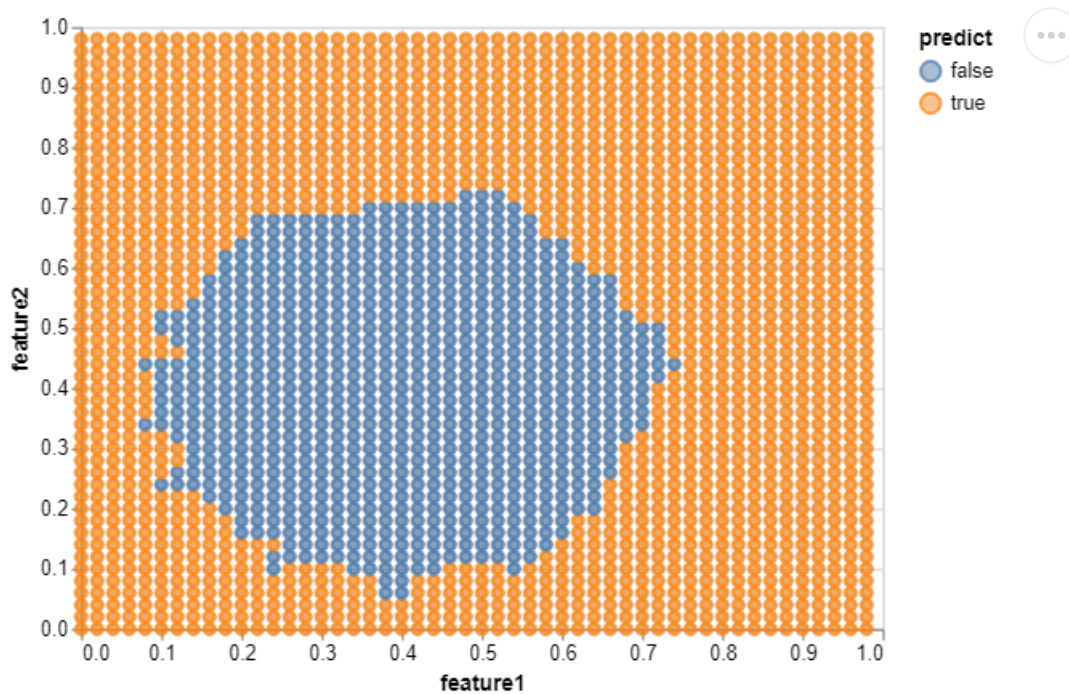
The neighbor model takes a different approach. Instead of producing a line, it memorizes all the points in training and predicts based on how close a test example is.

For this question, you should :

1. Fit the neighbor model to the circle data.
2. Predict on `a11_df`.

3. Graph the resulting shape.

```
# FILLME
neighbor_model.fit(X=df2_train[["feature1", "feature2"]],
                  y=df2_train["class"] == "red")
all_df["predict"] = neighbor_model.predict(all_df[["feature1", "feature2"]])
chart = (alt.Chart(all_df)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color="predict",
            fill = "predict",
        ))
chart
```



It will not be perfect but it should be much closer to the circle shape of the data.

▼ Question 2

So far all of our datasets have had 2 features. For this dataset there are three features (feature1, feature2, feature3).

```
df3 = pd.read_csv("https://srush.github.io/BT-AI/notebooks/three.csv")
```

Split the dataset into train and test, and then fit the linear model `model1` to all three of these features.

    FILLME

```
df3_train = df3[df3["split"] == "train"]
df3_test = df3[df3["split"] == "test"]
model3 = LinearClassification()
model3.fit(x=df3_train[["feature1", "feature2", "feature3"]],
          y=df3_train["class"] == "red")
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-68-d22fa4e222dc> in <module>()
      4 model3 = LinearClassification()
      5 model3.fit(x=df3_train[["feature1", "feature2", "feature3"]],
----> 6          y=df3_train["class"] == "red")
```





TypeError: fit() got an unexpected keyword argument 'x'

SEARCH STACK OVERFLOW

How many points in test does the model get correct?

    FILLME

```
df3_test["predict"] = model13.predict(df3_test[["feature1", "feature2", "feature3"]])
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-69-beb24920f77b> in <module>()
      1 #     FILLME
----> 2 df3_test["predict"] = model13.predict(df3_test[["feature1", "feature2",
"feature3"]])
```

NameError: name 'model13' is not defined

SEARCH STACK OVERFLOW

▼ Question 3

It turns out that for `df3` you only need two of the features to achieve high accuracy. Make a graph for each pair of features (three graphs total).

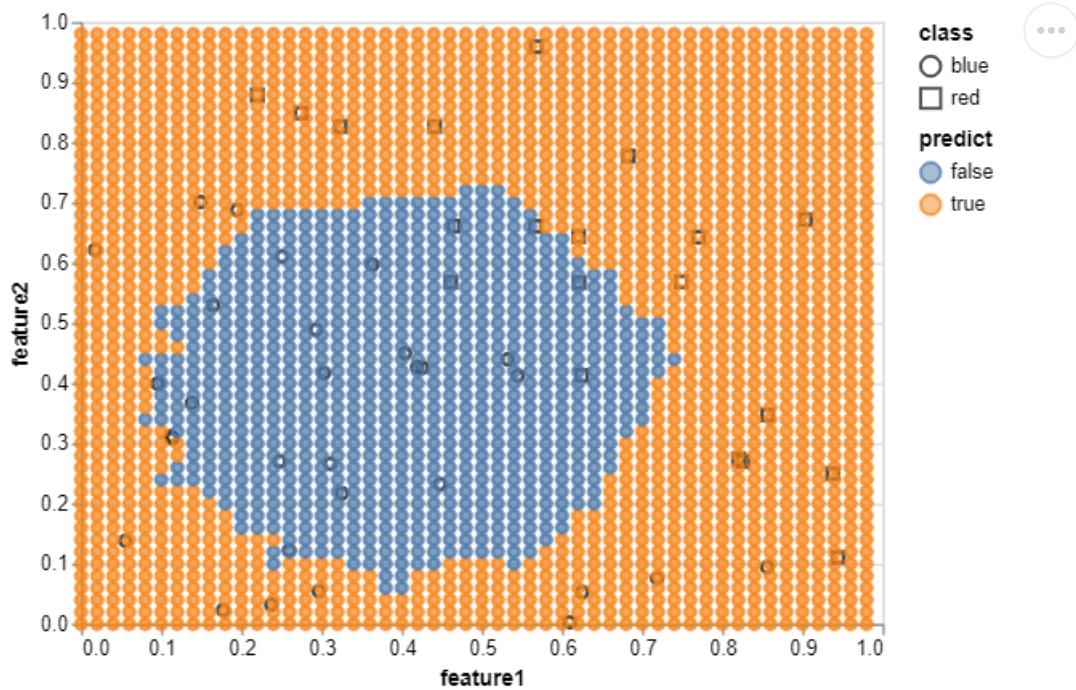
    FILLME

```
chart1 = (alt.Chart(all_df)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color="predict",
```

```

        fill = "predict",
    ))
chart2 + chart

```



Which are the two features that you need? Try fitting `model1` to just those two.

```

# FILLME
chart1 = (alt.Chart(all_df)
    .mark_point()
    .encode(
        x = "feature1",
        y = "feature2",
        color="predict",
        fill = "predict",
        model = 'predict'
    ))
chart2

```

