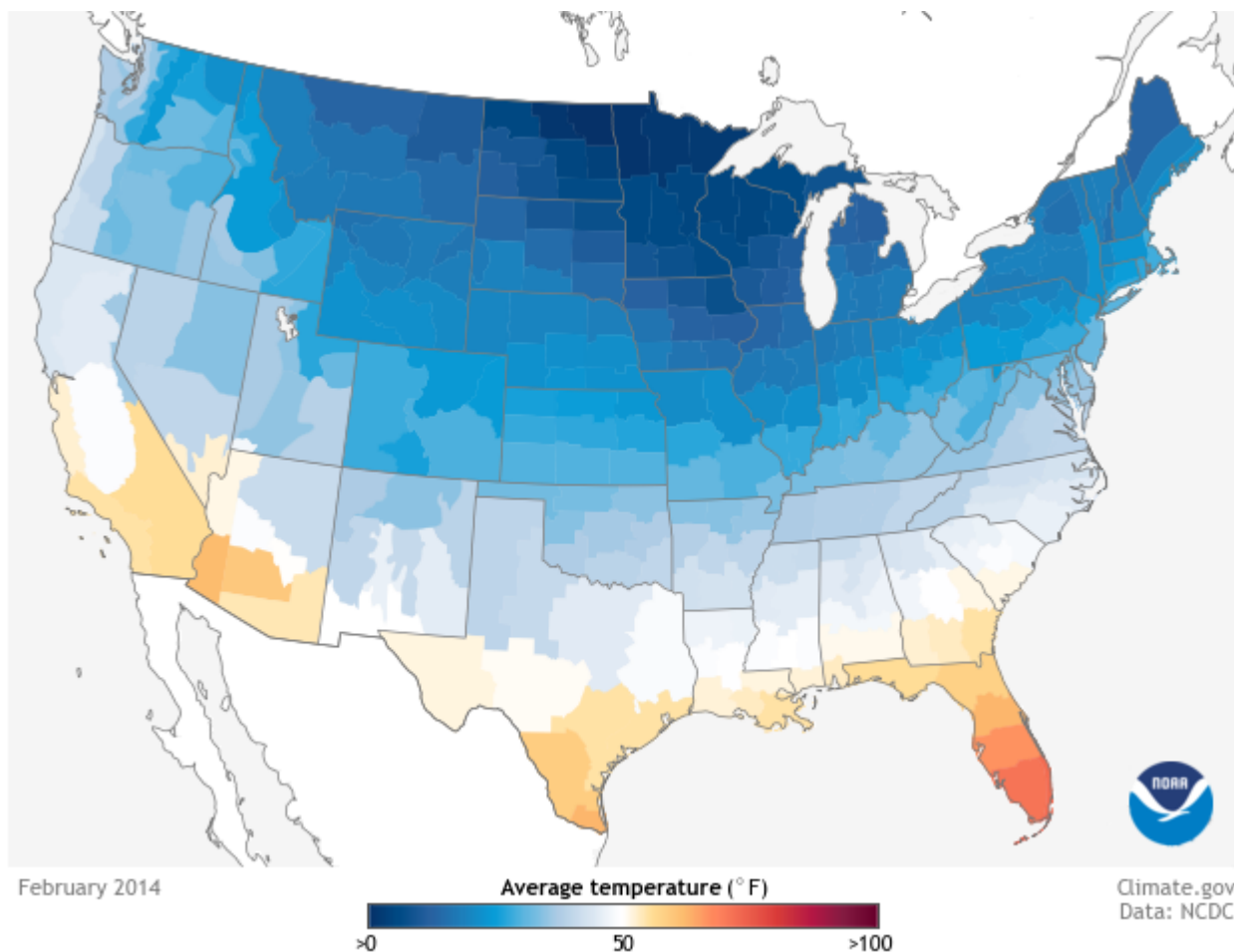


## ▾ Lab 5 - Machine Learning 2 - Features

This lab will continue our journey through applied machine learning. The main goal of the lab is to gain intuition for *features* and how better features allow us to capture more complex properties.

We will learn how features can be constructed for structured data and how they allow us to use simple models to make complex predictions.

We will also return to our climate change data and learn how to use past data to make future predictions.



## ▾ Review

We have learned 3 libraries so far. Pandas, Altair, and Scikit-Learn. We will use all three heavily

```
import pandas as pd
import altair as alt
import sklearn.linear_model
```

Reminder that our main sample dataset is a Red versus Blue classification challenge.

```
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
...	...	...	...	...
145	blue	test	0.259535	0.122557
146	red	test	0.937820	0.249618
147	blue	test	0.148987	0.700891
148	blue	test	0.531986	0.439514
149	red	test	0.567762	0.958953

150 rows × 4 columns

Machine learning data is always split into at least two groups. We get to look at the *train* data to *fit* our model. We then use the *test* data to evaluate our model.

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

We will focus on *classification*. Each data point has a *class* that we are trying to predict.

```
df_train["class"]
```

```
0    blue
1    blue
2     red
```

```

3     blue
4     red
...
95    blue
96    blue
97    red
98    red
99    red
Name: class, Length: 100, dtype: object

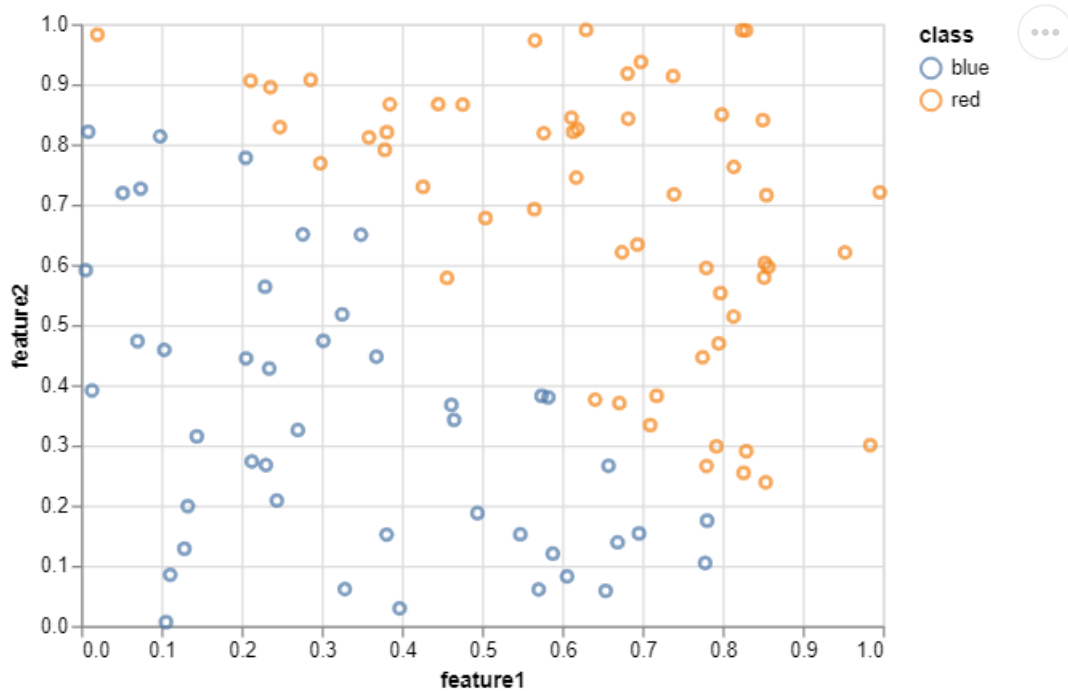
```

A point graph allows us to see our problem clearly. We need to separate the Reds from the Blues.

```

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

```



We can try this ourself by splitting the points doing it with one feature is not good enough.

```

def my_predict_feature1(point):
    if point["feature1"] > 0.5:
        return "red"
    else:
        return "blue"

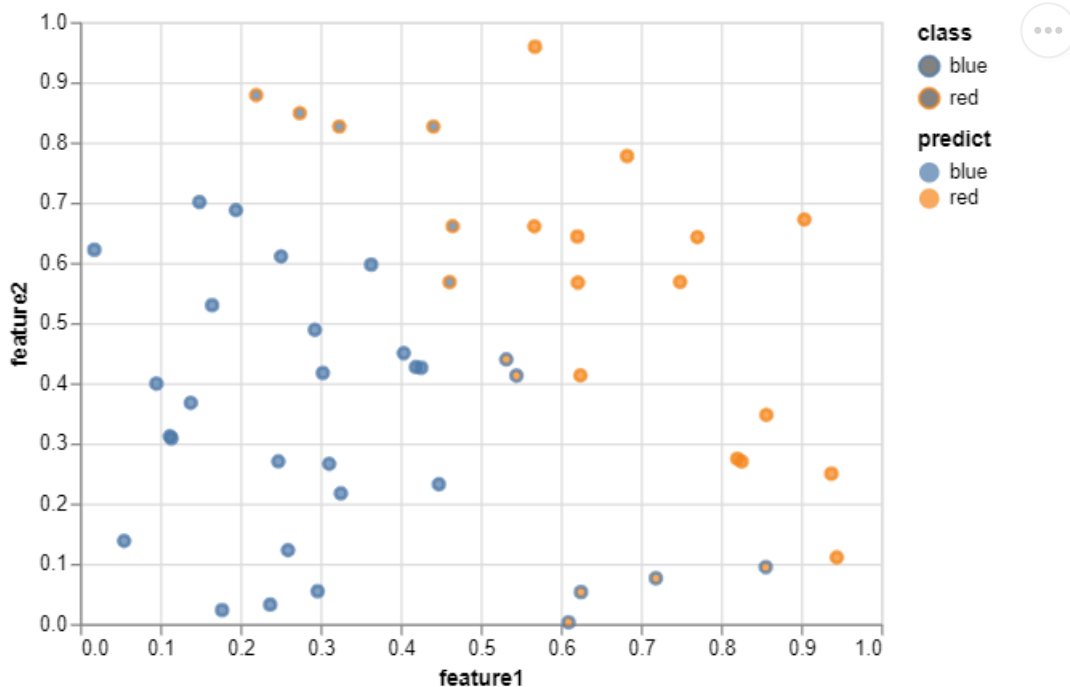
```

```
df_test["predict"] = df_test.apply(my_predict_feature1, 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.

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



But if we base it on two features we can get it right.

```
def my_predict_linear(point):
    if point["feature1"] + point["feature2"] < 1.0:
        return "blue"
    else:
        return "red"
```

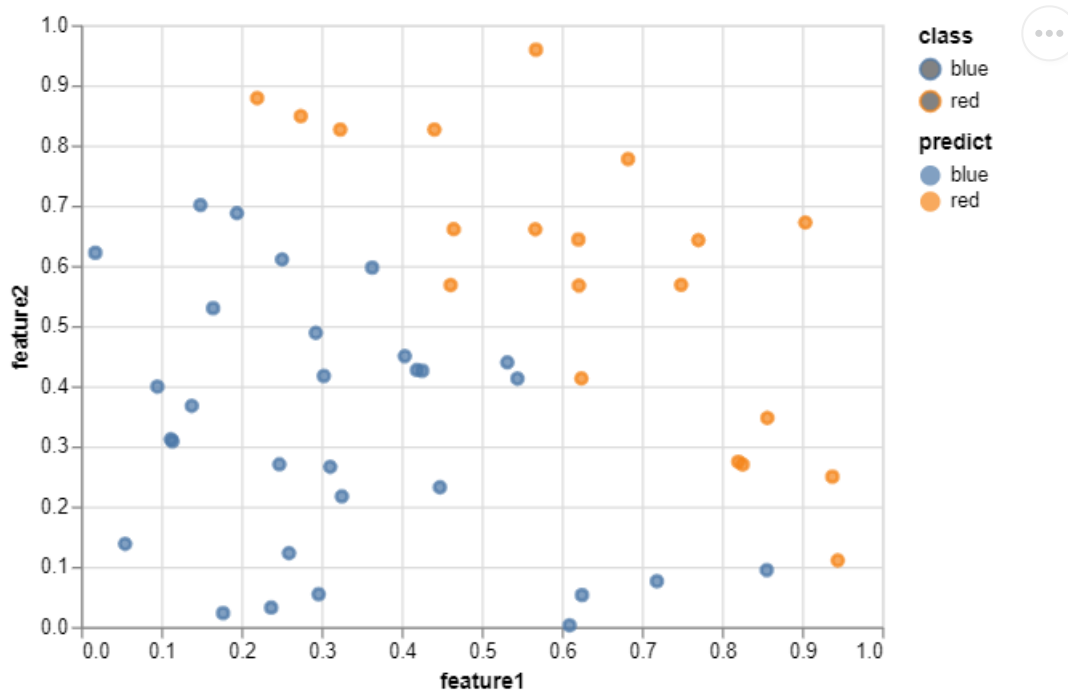
```
df_test["predict"] = df_test.apply(my_predict_linear, axis=1)
```

```
/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.

Notice how our cutoff used an addition between our two features. This allows us to form any line as the cutoff between the colors.

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



We are using the library *scikit-learn* for prediction. We are focusing on a method known as Linear Classification. (Note it has a silly name in the library, we rename it for simplicity.).

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

The main machine learning step is to fit our model to the features of the data.

```

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)

```

Linear Classification draws the best line to split the two classes. That is it tries out different combinations of adding together features and cutoffs to find the best one. (For now you can ignore how it does that.)

We can see this visually by trying out every possible point and seeing where the system puts them.

```

all_df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/all_points.csv")
all_df["predict"] = model.predict(all_df[["feature1", "feature2"]])

```

Here is a graph of out the points.

```

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

```



The goal of machine learning is to accurately predict the results of unseen data. We use the test data to make this possible.



## ▼ Review Exercise



Use the model to predict on the test data (using `model.predict`). What is the accuracy (number of points where the *prediction* is the same as the *true class*)?



# [COPY] [PASTE] [FILLME]

```
df_test["predict"] = model.predict(df_test[["feature1", "feature2"]])
df_test["is_red"] = df_test["class"] == "red"
df_test["correct"] = df_test["predict"] == df_test["is_red"]
df_test["correct"].mean()
```

File "<ipython-input-17-6452ef632ea5>", line 2

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

SyntaxError: invalid syntax

SEARCH STACK OVERFLOW

Can you graph the test data showing both the prediction and the true class?

# [COPY] [PASTE] [FILLME]

pass

## ▼ Unit A

## ▼ More Complex Data

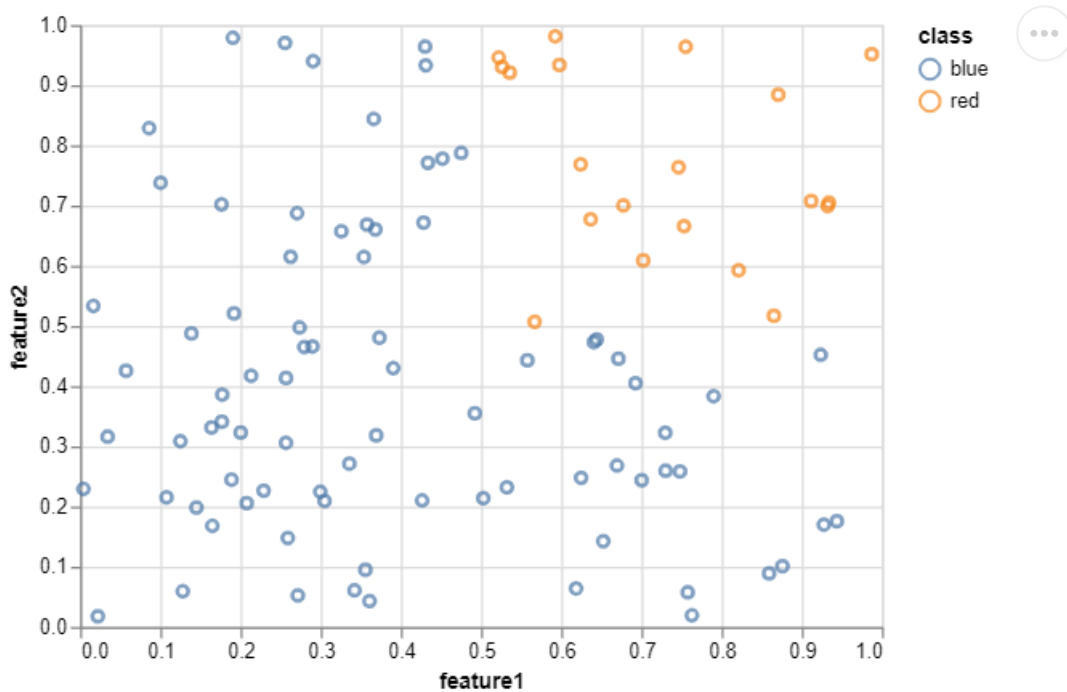
Our model is effective at drawing a line between data. However as we saw last time this only works if our data can be split easily with a line.


Let us load some more complex data.

```
df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/complex.csv")
df_train = df[df["split"] == "train"]
```

This data has all the red points in a the top-right corner. This makes it hard to separate.

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



 **Student question: Can you split the data yourself?**

```
#📄📄📄📄 FILLME
def my_predict(point):
    if True:
        return "blue"
    else:
        return "red"
```

```
df_test["predict"] = df_test.apply(my_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.



Let us now try to fit out data in the standard way.

```
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)
```

We can then predict on the train set to see how well it fit the data.

```
df_train["predict"] = model.predict(df_train[["feature1", "feature2"]])
df_train["correct"] = df_train["predict"] == (df_train["class"] == "red")

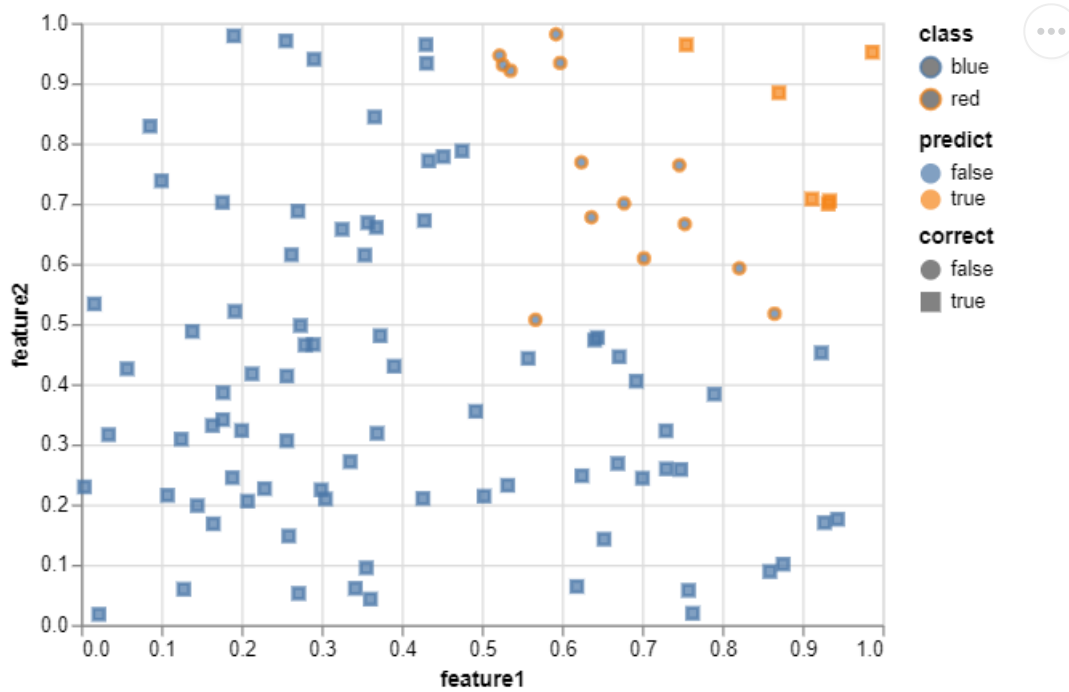
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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



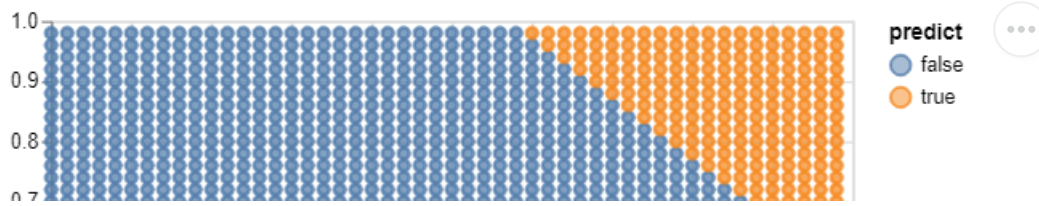
```
chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color="class",
            fill = "predict",
            shape = "correct",
        ))
chart
```



Unfortunately we can see that it did not fit well.

We can see that it did its best to try to find a line. However, it did not succeed in practice.

```
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
```



## ▼ Reasoning about Separation

Let us think about how we would separate this data manually.

Remember that our main tool was to create filters that let us divide up the points.

The upper right corner can be defined as the *and* of the right side and the upper side of our data.

```
feature1
filter = (df_train["feature1"] > 0.5) & (df_train["feature2"] > 0.5)
df_red = df_train.loc[filter]
```

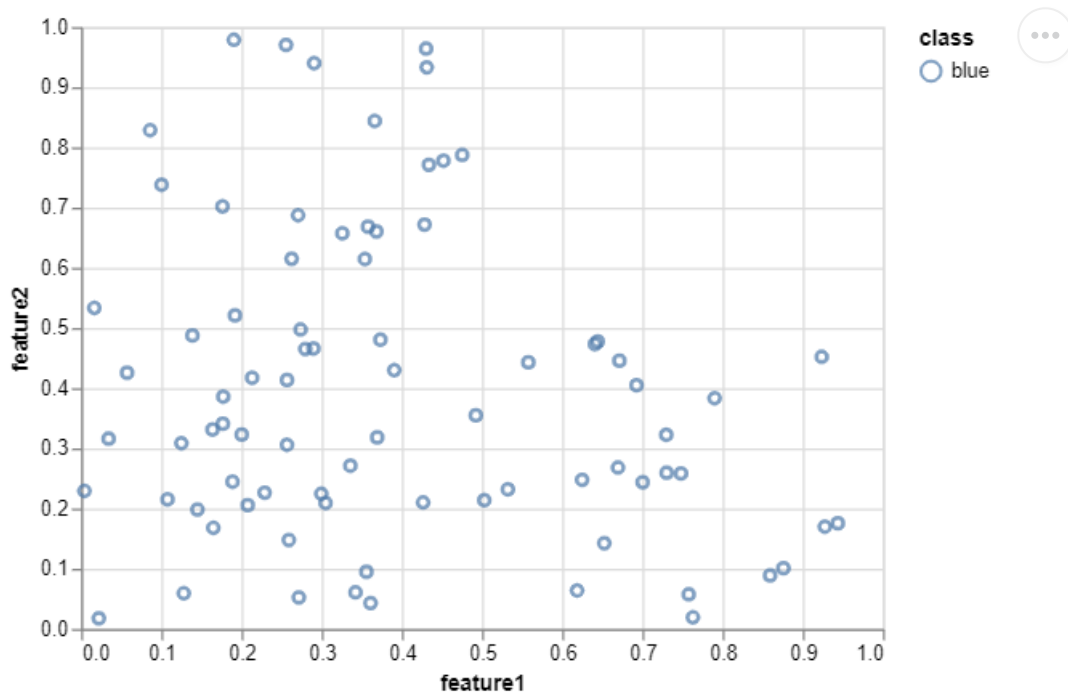
Here is what it looks like.

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

Conversely we separate the blue pts by the *or* of the the left side and the bottom side.

```
filter = (df_train["feature1"] < 0.5) | (df_train["feature2"] < 0.5)
df_blue = df_train.loc[filter]
```

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



## ▼ Features

We can create features as new functions that transform our data. Here we note that the feature should be aware of where the middle of the data is.

```
def mkupperfeature(f):
    if f <= 0.5:
        return -1.0
    else:
        return f
```

We can use standard pandas methods to create these new features.

```
df_train["feature3"] = df_train["feature1"].map(mkupperfeature)
df_train["feature4"] = df_train["feature2"].map(mkupperfeature)
```

```
/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.

```
/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>



Now we can fit our model. *Note* we are passing feature 3 and 4, not 1 and 2.

```
model.fit(X=df_train[["feature3", "feature4"]],
          y=df_train["class"] == "red")
df_train["predict"] = model.predict(df_train[["feature3", "feature4"]])
```

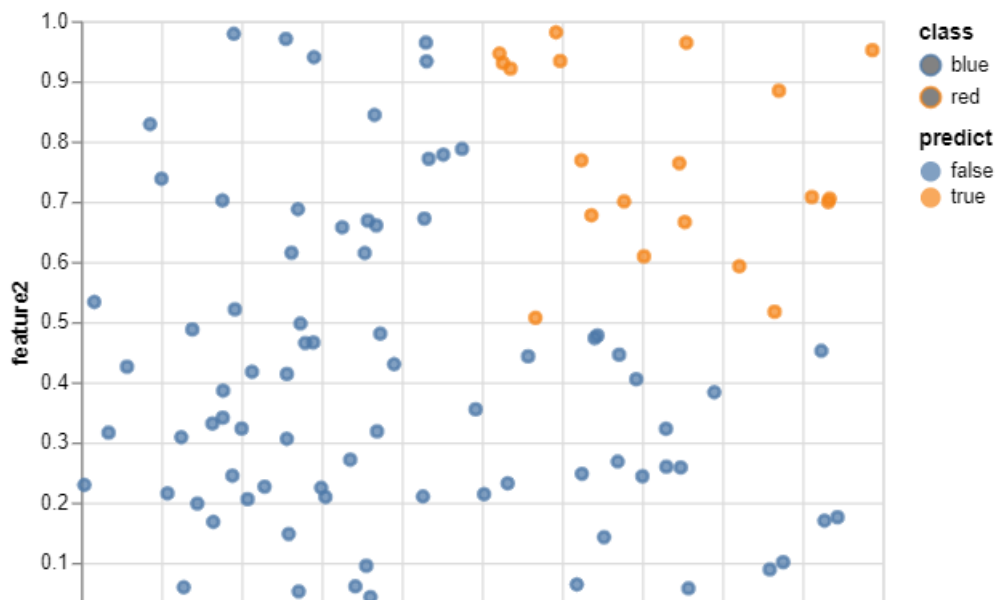
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>  
 This is separate from the ipykernel package so we can avoid doing imports until



Now it works!

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



Let's repeat that and look what happened. We first added new features that tell the model about the up/down and left/right regions.

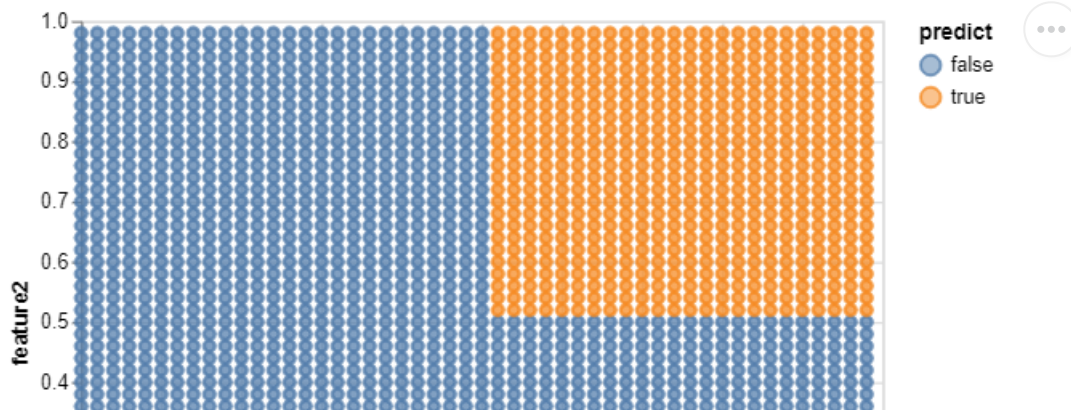
```
all_df["feature3"] = all_df["feature1"].map(mkupperfeature)
all_df["feature4"] = all_df["feature2"].map(mkupperfeature)
```

We then predict on these new features.

```
all_df["predict"] = model.predict(all_df[["feature3", "feature4"]])
```

And when we graph on the original features, it can draw fancy shapes.

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

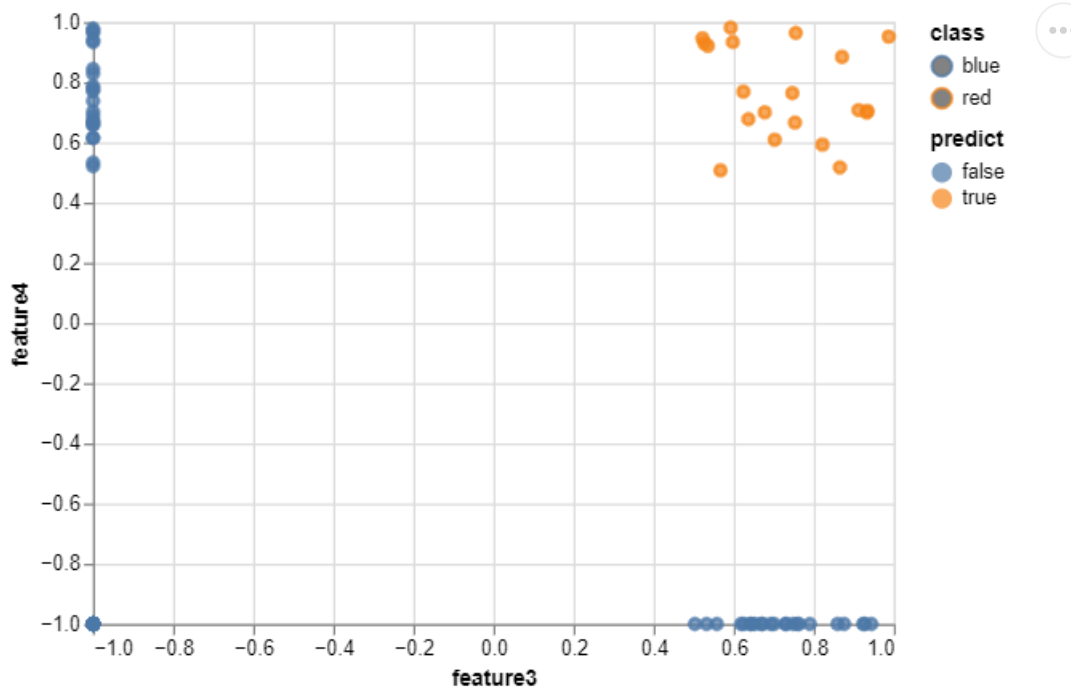


Why were able to draw a line that had a square shape?



The key trick is that the line is straight in the features that we gave the model. Features 3 and 4. If we graph them it looks like this.

```
chart = (alt.Chart(df_train)
    .mark_point()
    .encode(
        x = "feature3:Q",
        y = "feature4:Q",
        color="class",
        fill = "predict",
    ))
chart
```







The key idea is that when we train on the new features. We can be *non-linear* in the original features.

## ▼ Group Exercise A

For this exercise we will try to learn classifiers for the following data sets. This will let us experiment with different features and their usefulness.

### ▼ Question 0

Who are other members of your group today?

#    FILLME  
Kera McGovern, Lyra D'Souza

Do they prefer cats or dogs?

#    FILLME  
pass

Do they prefer summer or winter?

#    FILLME  
pass

Do they prefer sweet or savoury breakfast?

#    FILLME  
pass

### ▼ Question 1

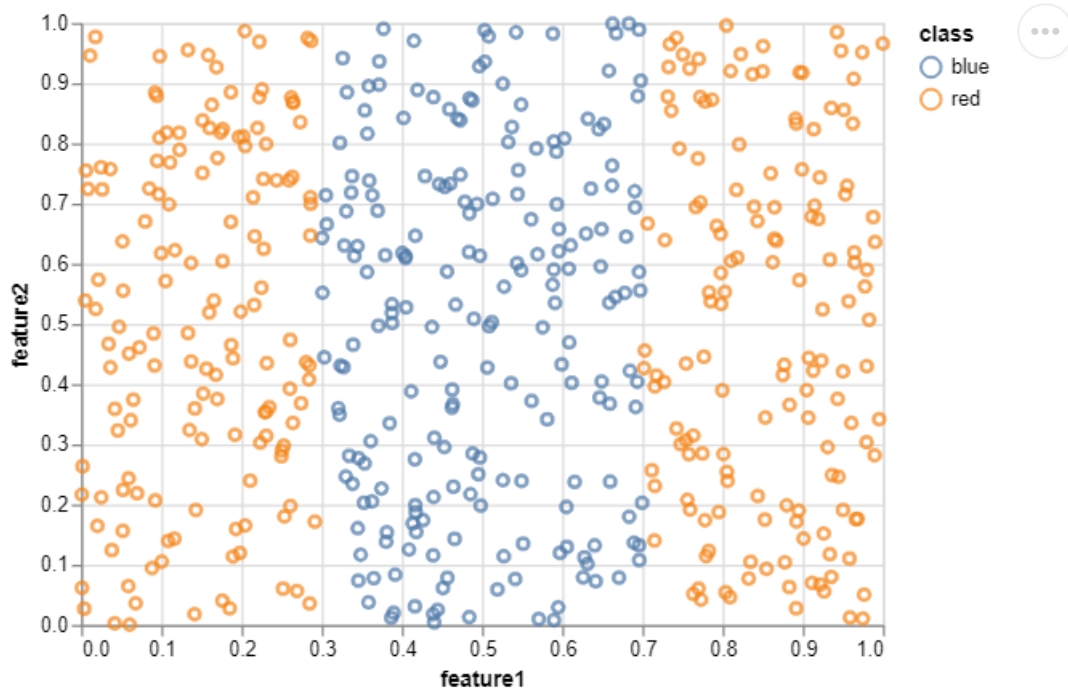
For this first question, the data is separated into two parts based on how far it is from the center



```

dt = pd.read_csv("https://srush.github.io/BI-AI/notebooks/center.csv")
df_train = df.loc[df["split"] == "train"]
df_test = df.loc[df["split"] == "test"]
chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class"
        ))
chart

```



Construct new features that allow you to separate the data into parts.

```

# FILLME
def mkextreme(f):
    if f < 0.3 or f >= 0.7:
        return "red"
    else:
        return "blue"

df_train["feature3"] = df_train["feature"].map(mkextreme)

```

```

-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key, method, tolerance)
    2897         try:
-> 2898             return self._engine.get_loc(casted_key)
    2899         except KeyError as err:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'feature'

```

The above exception was the direct cause of the following exception:

```

KeyError                                Traceback (most recent call last)
^ 2 frames

```

## ▼ Question 2

```

2899         except KeyError as err:

```

Now the data is split into a circle around the point (0.4, 0.4).

```

2902         if tolerance is not None:

```

```

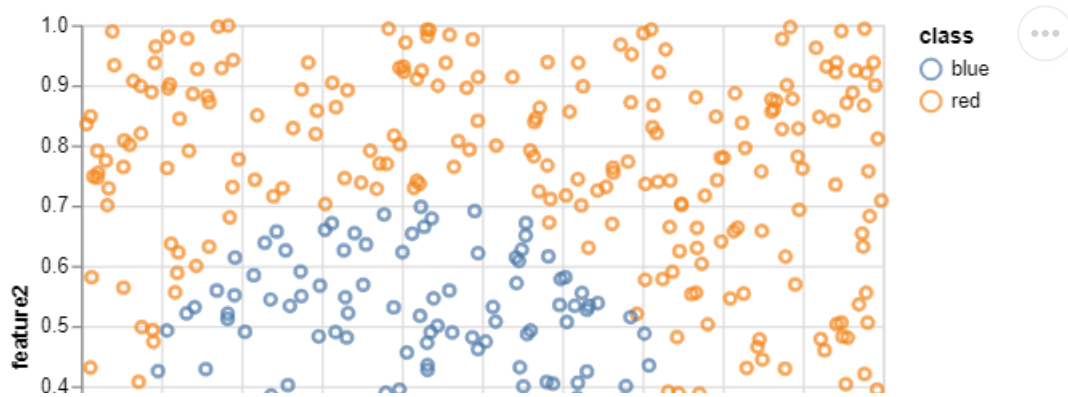
df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/circle.csv")
df_train = df.loc[df["split"] == "train"]
df_test = df.loc[df["split"] == "test"]

```

```

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

```



Use the formula for distance from a point  $\sqrt{(x - 0.4)^2 + (y - 0.4)^2}$  to define new features for this problem.



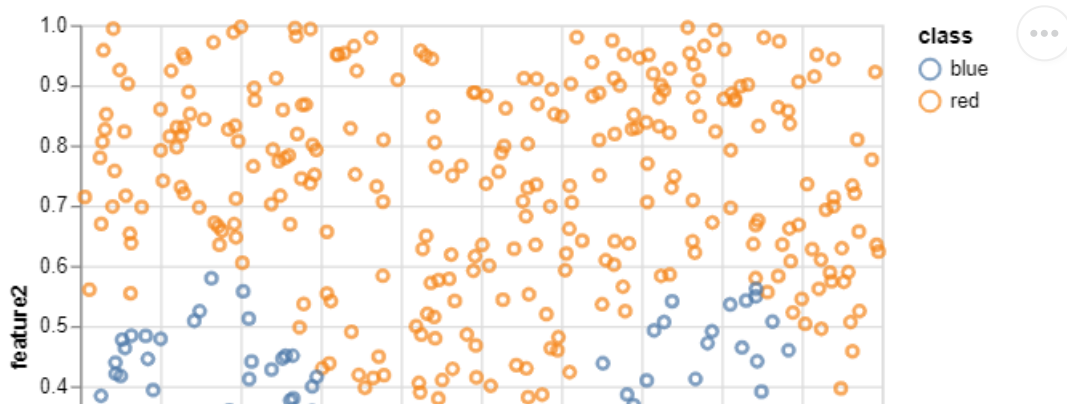
```
# FILLME
```

```
def feature_circle(point):
    eq = (point["feature1"]-0.4)**2 + (point["feature2"]-0.4)**2
    if eq > 0.09:
        return "red"
    else:
        return "blue"
```

## ▼ Question 3

Finally consider a problem with a periodic (repeating) curve.

```
df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/periodic.csv")
df_train = df.loc[df["split"] == "train"]
df_test = df.loc[df["split"] == "test"]
chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color = "class"
        ))
chart
```



Define a new feature based on the formula  $\sin(\text{feature1} * 10)$ . (Sin is in the math library).



```
import math
math.sin
```

```
<function math.sin>
```

Use this function to create a new feature (feature 3). Show that training with feature 1,2, and 3 will yield an accurate value

```
# FILLME
```

## ▼ Unit B

## ▼ Real World Data

Now we will apply the methods we learned in the previous section to a real-world classification problem. We will return to the temperature classification problem from Week 3

```
temp_df = pd.read_csv("https://srush.github.io/BT-AI/notebooks/Temperatures.csv", index_col=0)
```

We convert the temperature to Fahrenheit for easier visualization.

```
temp_df["Temp"] = (temp_df["Temperature"] * 9/5) + 32
```

To make things simpler, we filter to the US in the years 2000 and 2001

```
filter = ((temp_df["Country"] == "United States") &
          (temp_df["Date"].dt.year >= 2000) & (temp_df["Date"].dt.year <= 2001))
df = temp_df.loc[filter]
```

Just as a reminder here are our columns.

```
df.columns
```

```
Index(['Date', 'Temperature', 'TemperatureUncertainty', 'City', 'Country',
       'Latitude', 'Longitude', 'Temp'],
      dtype='object')
```

## ▼ Example: Predicting Temperature

We will start with the question of predicting when the temperature goes over 70 degrees.

```
df["class"] = df["Temp"] > 70
```

```
/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.



To make our splits we divide the data arbitrarily into a train and test set.

```
def mksplit(city):
    if city[0] > "M":
        return "test"
    else:
        return "train"
df["split"] = df["City"].map(mksplit)
```

```
/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>



```
# What factors into the temperature of a city?
```

```
# Latitude ()
```

```
# Altitude / Sea Level (Flipped Simple)
# Longitude
# Proximity to Ocean
# Data -> Time of the year
#
```

The main question of interest will be "Which features should we use?". We want features that best separate the data into warm and cold groups.

## ▼ Try 1

A natural first attempt is to use location features. We can use the Longitude (east-west) and Latitude (north-south) to start.

```
df["feature1"] = df["Longitude"]
df["feature2"] = df["Latitude"]
```

```
/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.
```

```
/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>



We then split the data into train and test sets.

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

The different classes are True and False, i.e. is it over 70 for that city in that month.

```
df_train["class"]
```

```
170235    False
170238    False
170241     True
170244    False
170247    False
```

```

...
4827795    False
4827798    False
4827801    False
4827804     True
4827807    False
Name: class, Length: 176, dtype: bool

```

Now we follow our standard method to fit the data and see how well it does at separating on the train set.

```

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

```

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: 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>  
after removing the cwd from sys.path.

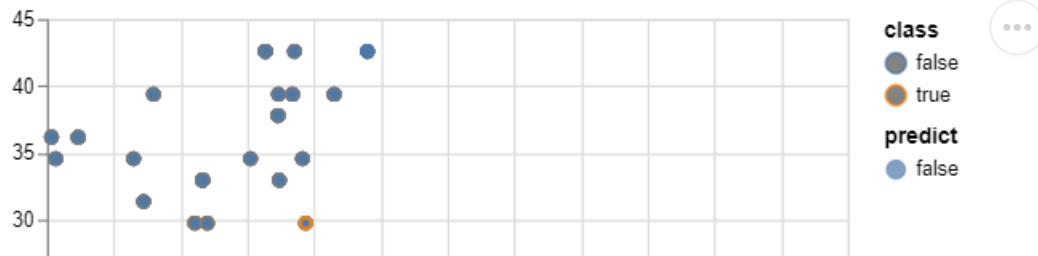


Unfortunately the model does not do that well at all!

```

chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature1",
            y = "feature2",
            color="class",
            fill = "predict",
            tooltip = "City"
        ))
chart

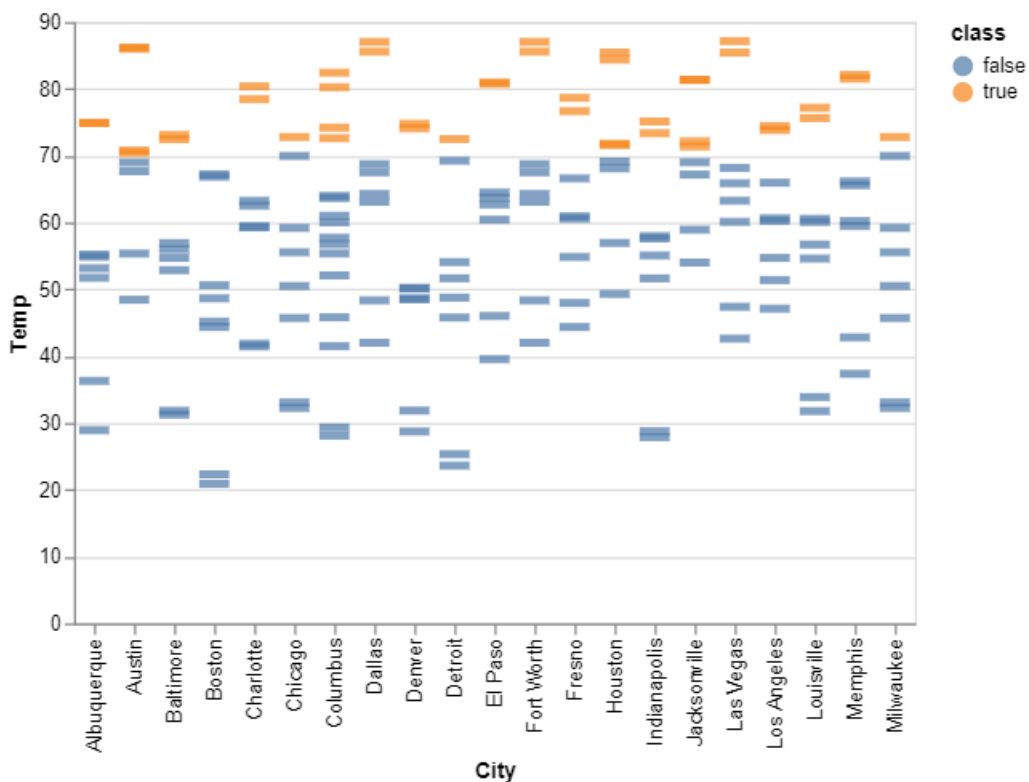
```



## ▼ Try 2

Let us look at the data to see what is happening. Why is it not enough to just look at Longitude and Latitude?

```
chart = (alt.Chart(df_train)
    .mark_tick(thickness=4)
    .encode(
        x = "City",
        y = "Temp",
        color="class",
        fill = "class",
        tooltip = "City",
        shape="City"
    ))
chart
```

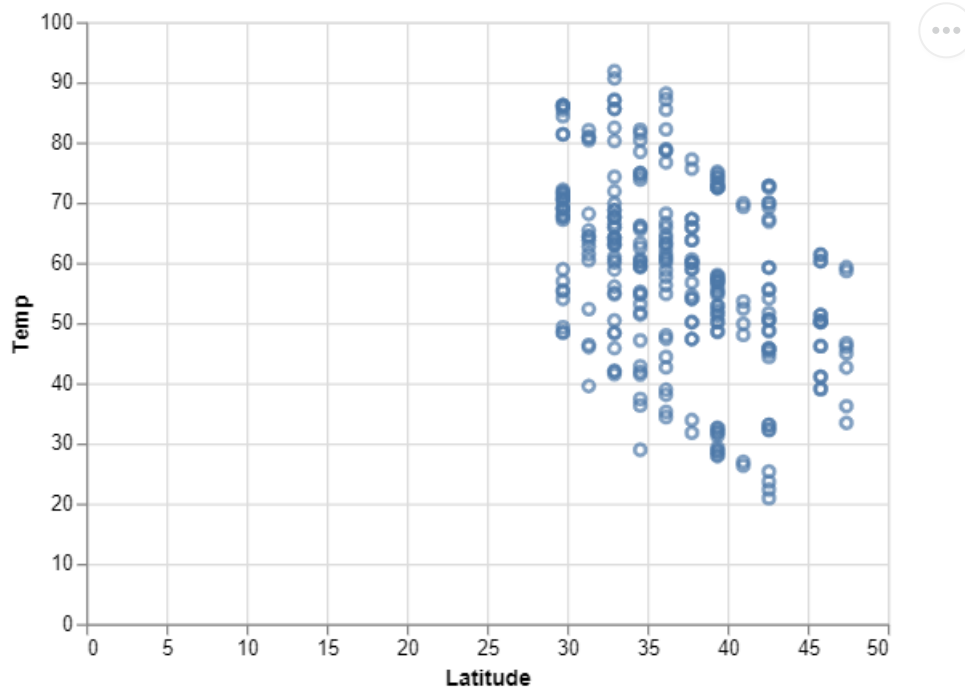




The problem seems to be seasons. The same city will have very different average temperatures in different months.

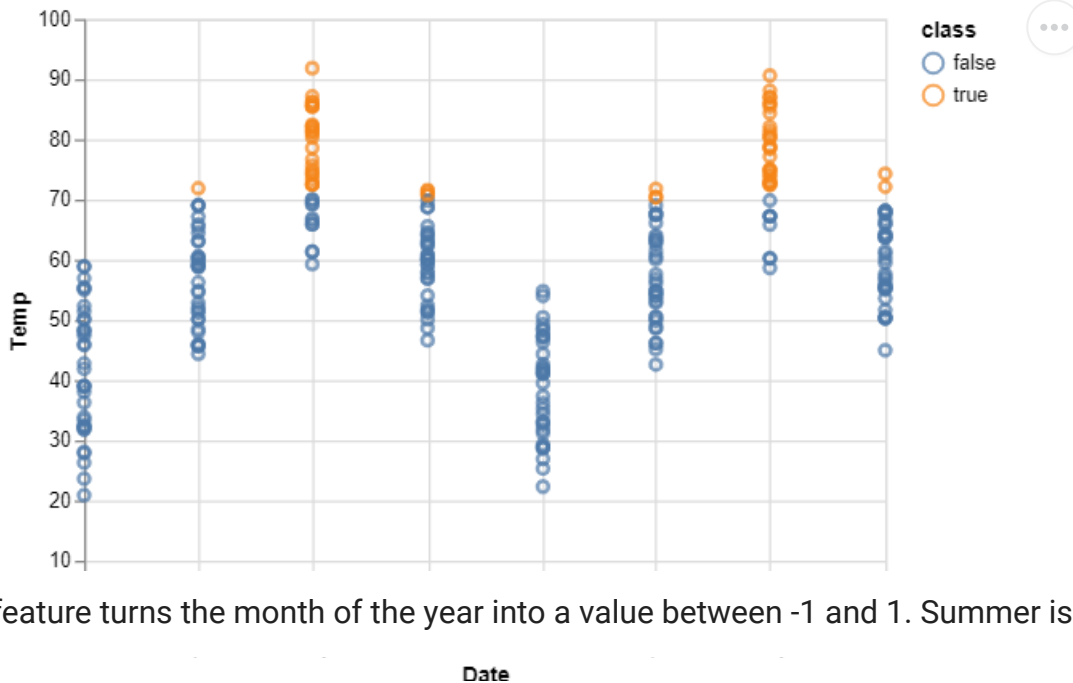
We can see this by looking at points grouped by latitude.

```
chart = alt.Chart(df).mark_point().encode(
    y = "Temp",
    x = "Latitude",
    tooltip=["City", "Country"],
)
chart
```



Latitude is clearly a useful feature, lower latitude means higher temperature. But it is not enough alone. We need to be able to separate out by season as well.

```
chart = alt.Chart(df).mark_point().encode(
    y = "Temp",
    x = "Date:T",
    color = "class",
    tooltip=["City", "Country"],
)
chart
```



This feature turns the month of the year into a value between -1 and 1. Summer is 1 and winter is 0.

```
def mkfeature(time):
    return -math.cos(((time.month - 1) / 11.0) * 2 * math.pi)
```

We can include this feature with latitude.

```
df["feature3"] = df["Date"].map(mkfeature)
df["feature4"] = df["Latitude"]
```

```
/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.
```

```
/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>

Apply the standard ML steps.

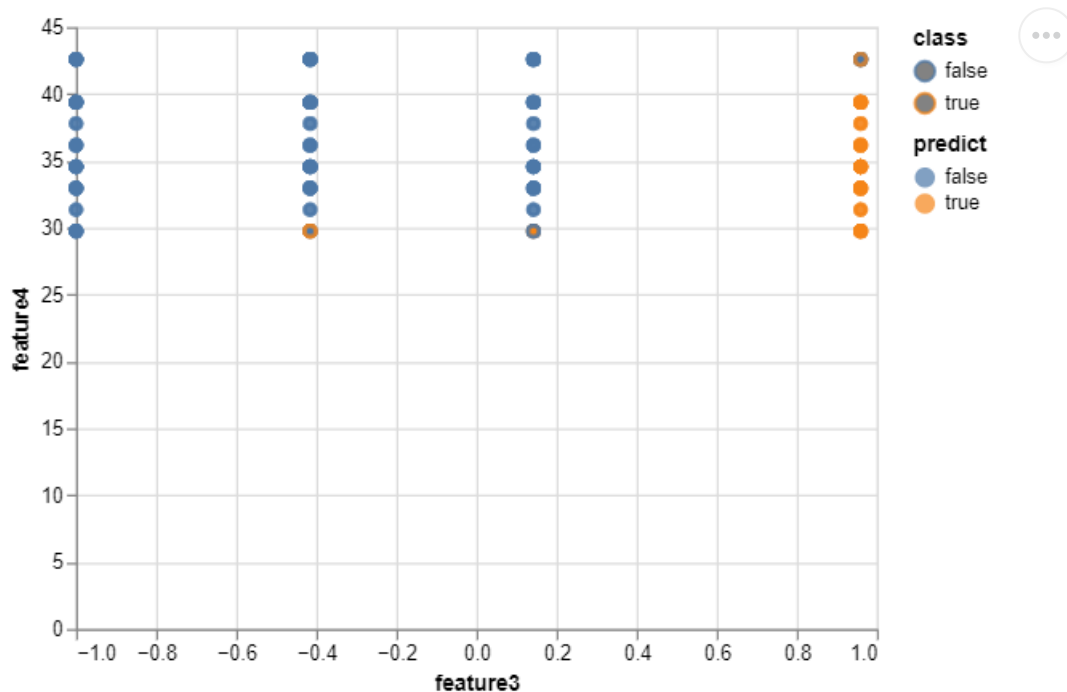
```
df_train = df.loc[df["split"] == "train"]
df_test = df.loc[df["split"] == "test"]
model.fit(X=df_train[["feature3", "feature4"]],
          y=df_train["class"])
df_train["predict"] = model.predict(df_train[["feature3", "feature4"]])
```

/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>

And now we can plot our graph. Here the x axis is the seasonal value and the y axis is the latitude. Mostly the model is able to now predict correctly.

```
chart = (alt.Chart(df_train)
        .mark_point()
        .encode(
            x = "feature3",
            y = "feature4",
            color="class",
            fill = "predict",
            tooltip = "City"
        ))
chart
```



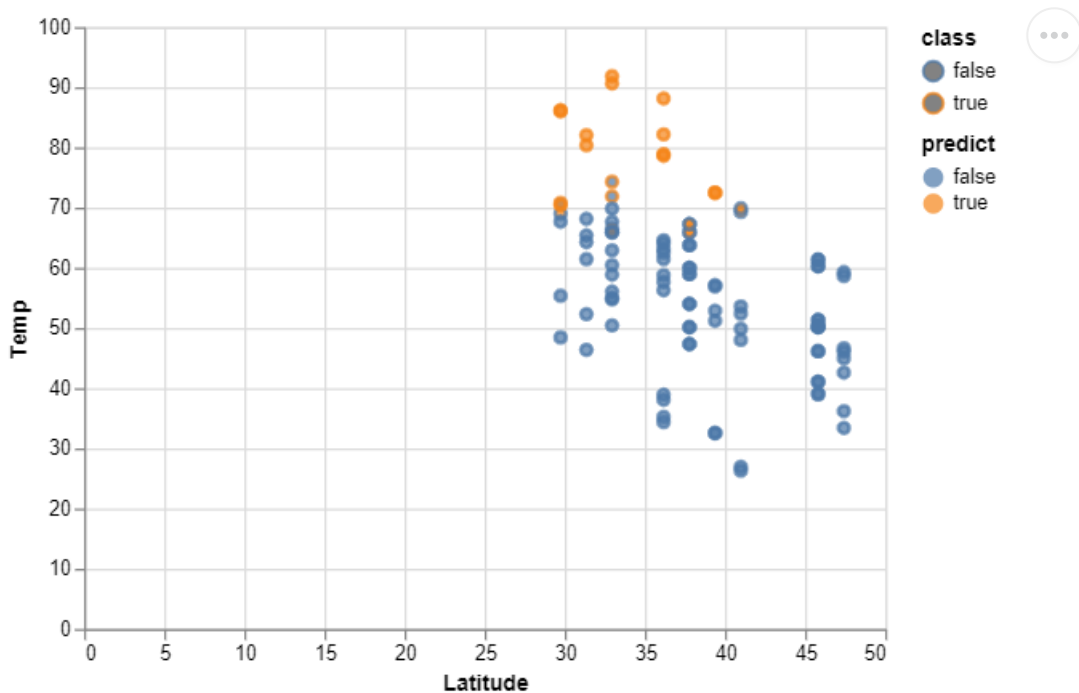
Checking on the test data we can see that the model use both features to make its predictions.

```
df_test["predict"] = model.predict(df_test[["feature3", "feature4"]])
```

/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.

```
chart = alt.Chart(df_test).mark_point().encode(
    y = "Temp",
    x = "Latitude",
    color="class",
    fill = "predict",
    tooltip=["City", "Country"],
)
chart
```



## ▼ Pretty Chart

Just for fun we can chart this data on a map.

```
from vega_datasets import data
us_cities_df = df.loc[df["Country"] == "United States"]
```

We just look at one month in the data.

```
df["predict"] = model.predict(df[["feature3", "feature4"]])
df_summer = df[(df["Date"].dt.month==4) & (df["Date"].dt.year==2001)]
```

```
/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.



Draw the background map.

```
states = alt.topo_feature(data.us_10m.url, feature='states')
background = alt.Chart(states).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=500,
    height=300
).project('albersUsa')
```

Plot our points, predictions, and the true answer on the map.

```
points = alt.Chart(df_summer).mark_point(size=100).encode(
    longitude='Longitude',
    latitude='Latitude',
    fill="Temp",
    color="predict",
    shape="class",
    tooltip=['City', "Temp"]
)
chart = background + points
chart
```



## ▼ Example 2: Adding Other Countries



For the group activity today, you will extend these ideas to other countries. In particular we will consider 3 others.



```
filter = (((temp_df["Country"] == "Canada") |
            (temp_df["Country"] == "France") |
            (temp_df["Country"] == "Brazil"))) &
            ((temp_df["Date"].dt.year == 2000) & (temp_df["Date"].dt.month==7)))
df = temp_df.loc[filter]
```

df

	Date	Temperature	TemperatureUncertainty	City	Country	Latitude	Long
<b>1307225</b>	2000-07-01	18.871	0.294	Calgary	Canada	50.63	-
<b>2169112</b>	2000-07-01	16.863	0.286	Edmonton	Canada	53.84	-
<b>2845616</b>	2000-07-01	19.064	0.336	Hamilton	Canada	42.59	
<b>3837556</b>	2000-07-01	19.082	0.304	Kingston	Canada	44.20	
<b>4920948</b>	2000-07-01	18.732	0.289	Montreal	Canada	45.81	
...	...	...	...	...	...	...	...
<b>579069</b>	2000-07-01	16.200	0.265	Strasbourg	France	49.03	
<b>582308</b>	2000-07-01	21.607	0.187	Toulon	France	42.59	
.....							

Classification and features are the same.

```
df["class"] = df["Temp"] > 70
df["feature3"] = df["Date"].map(mkfeature)
df["feature4"] = df["Latitude"]
```

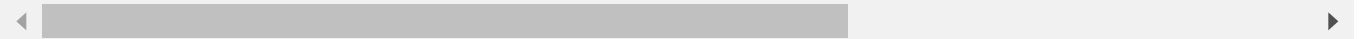
/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.

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.  
 /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>  
 /usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:3: 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>  
 This is separate from the ipykernel package so we can avoid doing imports until



The model and predictions are the same.

```
model.fit(X=df[["feature3", "feature4"]],
          y=df["class"])
df["predict"] = model.predict(df[["feature3", "feature4"]])
```

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:4: 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

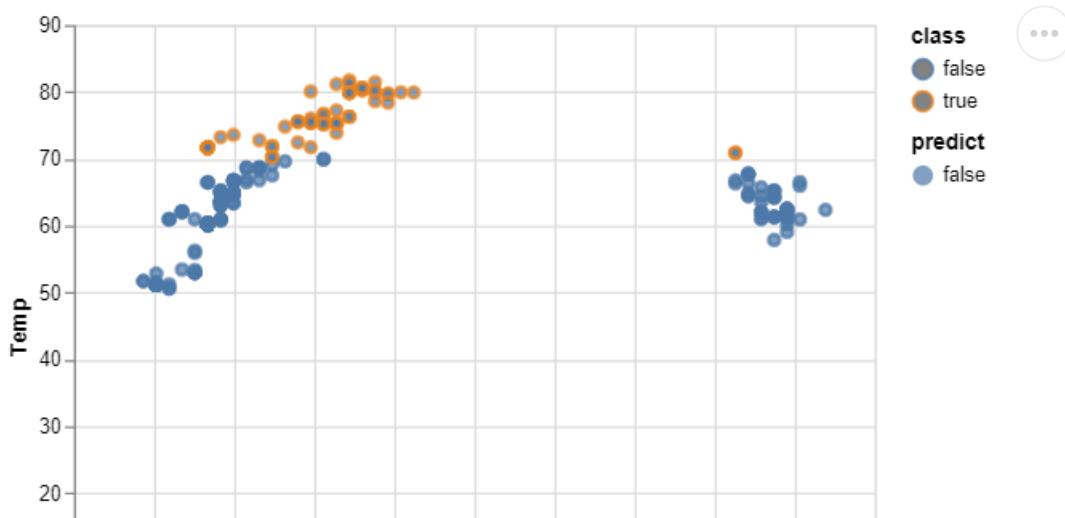
See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>  
 after removing the cwd from sys.path.



However, suddenly this approach does not work!

```
chart = alt.Chart(df).mark_point().encode(
    y = "Temp",
    x = "Latitude",
    color="class",
    fill = "predict",

    tooltip=["City", "Country"],
)
chart
```



## ▼ Group Exercise B

Latitude

For this group exercise you will puzzle through what is going wrong when we apply the approach to other countries on the map.

## ▼ Question 1

The model predicts the cities in Brazil mostly incorrectly. What is going wrong with this approach?

 FILLME

Canada and France have a high latitude while Brazil has low latitude. Brazil's data is differ

## ▼ Question 2

Modify one of the features in the model to help it correctly predict the Brazil cities.

# FILLME  
pass

## ▼ Question 3

Now consider a comparison of just France and Canada



```

filter = (((temp_df["Country"] == "France") |
            (temp_df["Country"] == "Canada")) &
            ((temp_df["Date"].dt.year == 2000) & (temp_df["Date"].dt.month==1)))
df = temp_df.loc[filter]
df["class"] = df["Temp"] > 25

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: 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\_
"""

```



Surprisingly France is at a very northern Latitude, almost as high as Canada.

Draw a chart like above that shows what happens when you try to run our standard model on this data.

```

# FILLME
def formula_predict(point):
    if math.sqrt((point["feature1"]-.4)**2 + (point["feature2"]-.4)**2) < .315:
        return 1
    else:
        return -1

df_train["feature3"] = df.apply(formula_predict, axis=1)
df_train["feature4"] = df.apply(formula_predict, axis=1)

model.fit(X=df_train[["feature3", "feature4"]],
          y=df_train["class"] == "red")
df_train["predict"] = model.predict(df_train[["feature3", "feature4"]])

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

```

```
File "<ipython-input-133-426e0d80f406>", line 3
    if math.sqrt((point["feature1"]-.4)**2 + (point["feature2"]-.4)**2) < .315:
```

Propose a different feature and add it to the model to mostly correctly classify this dataset. In particular, how does the classifier split France and Canada?

SEARCH STACK OVERFLOW

#     FILLME

