# TELCO REPORT

Matteo Chianale

May 2023

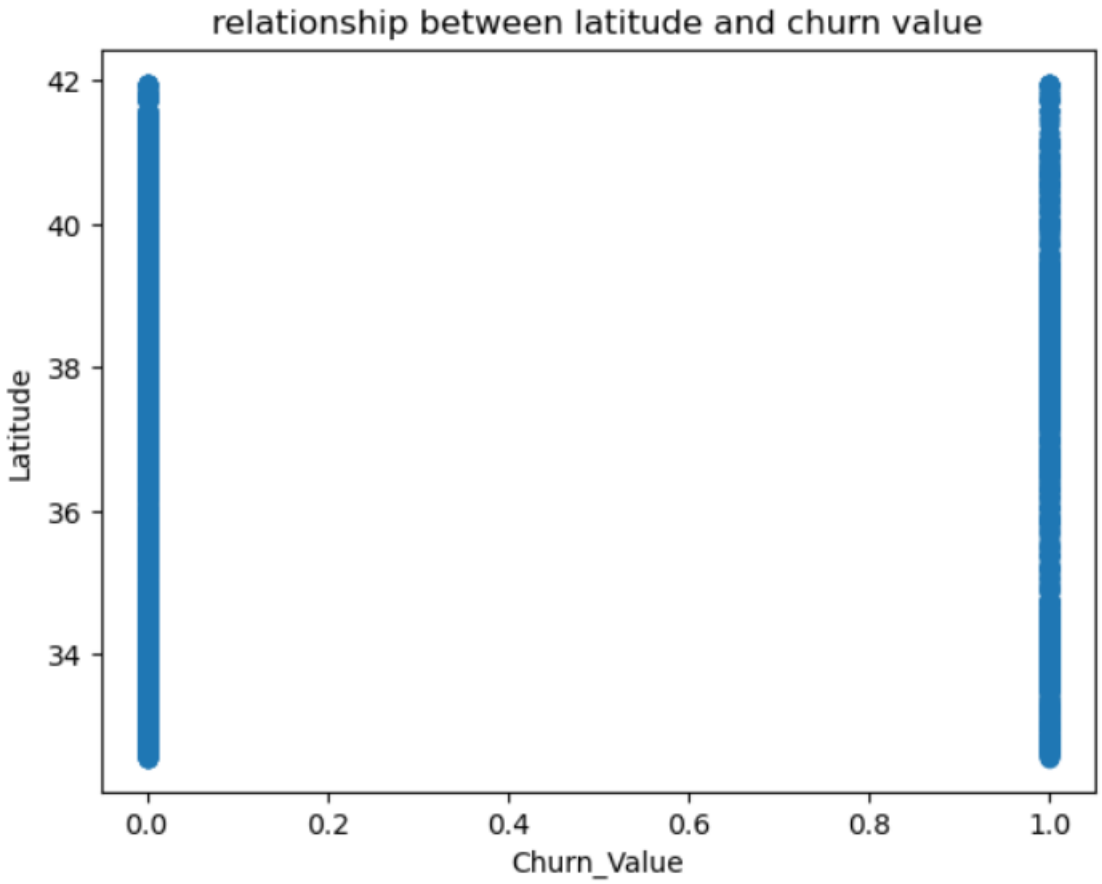
## 1 INTRODUCTION

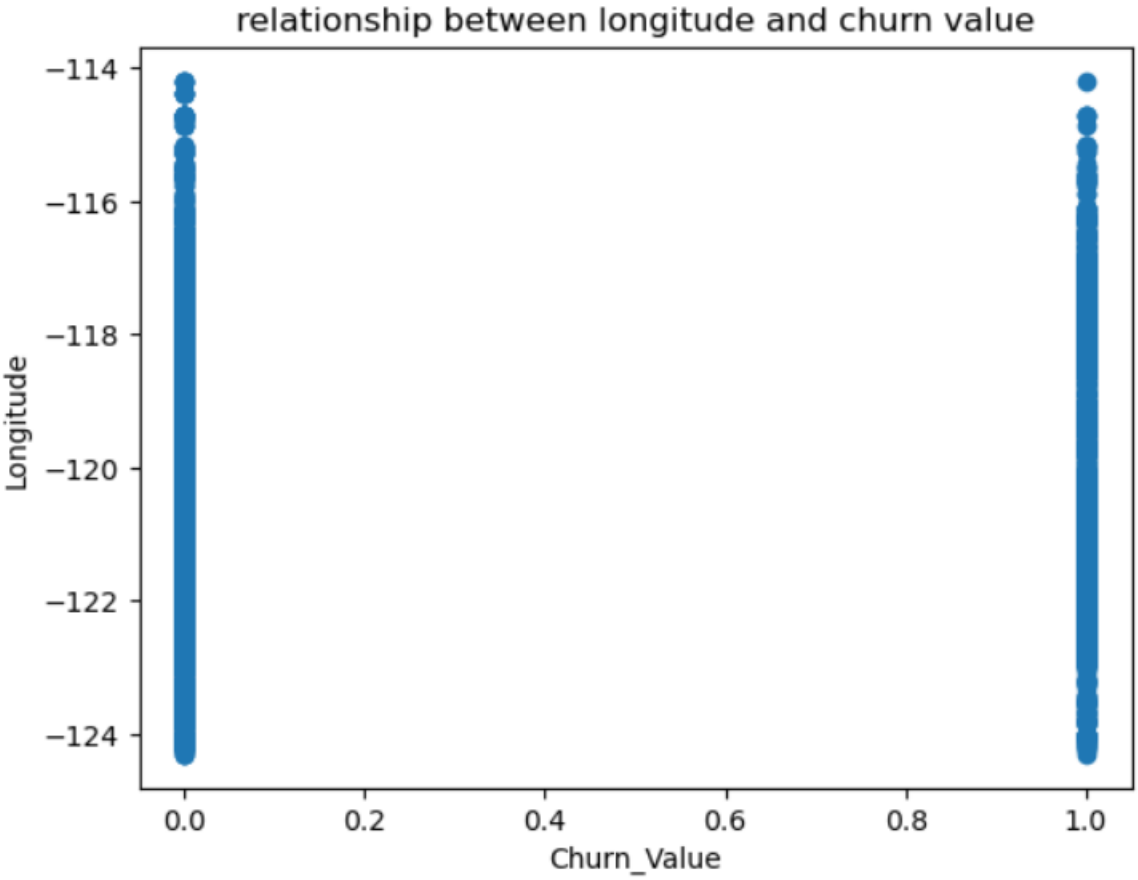
GITHUB : `https://github.com/mchianale/telco_churn.git`

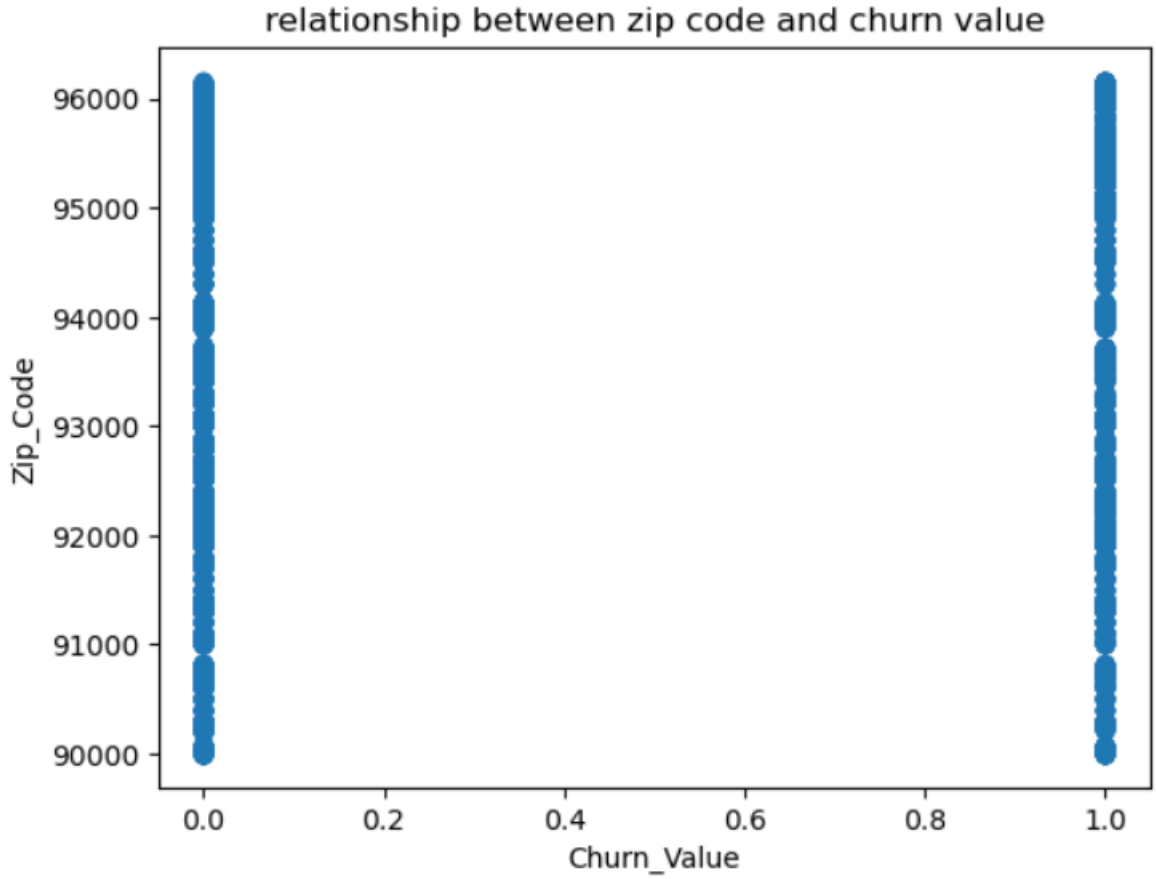
Telco Systems is a global leader in telecommunications, with over 40 years of experience in the design and development of high-performance network communications solutions. But accustomed to any company, some customers, even regulars, end up no longer buying from Telco, and those for various reasons. As part of this project, I will predict whether a customer will churn their subscription to a service based on their usage patterns, demographics and other characteristics.

## 2 DATASET

The main challenge of this project was to obtain a clean and relevant dataset. The majority of the code was to make the dataset readable and clear. The data being stored in a single excel file "Telco_customer_churn" with a large number of categories. I preferred to create functions that made it possible to generalize the conversion of Data, its overpressure and its information. I deleted all the elements corresponding to the location (City, Zip Code, latitude, longitude) because it was too complex to correlate with "Churn_Value" (City) or had no apparent correlation with "Churn_Value", according to the following three graphs:


relationship between latitude and churn value
Latitude
42
40
38
36
34
0.0
0.2
0.4
0.6
0.8
1.0
Churn_Value


relationship between longitude and churn value
−114
−116
−118
−120
−122
−124
Longitude
0.0
0.2
0.4
0.6
0.8
1.0
Churn_Value




For each of this graph, we can see there are no link between "Churn_Value" and each class (latitude, longitude, zip code). Even the customer churn or not, we have the same distribution for each class.
I also drop Churn Reason, Churn Label because it tells us directly if a customer churn or not, we can't use it

# 3 K-NEAREST NEIGHBORS

In our project we used the KNN model to predict the Churn Value of customers. It is a supervised classification algorithm based on the closest k examples (neighbors) of an input example to predict its class. The functioning of the KNN algorithm can be summarized in four steps:
- Calculate the distance between the input example and all learning examples (or reference examples).
- Select the k examples closest to the input example, using the distance calcu-

lated in the previous step.
- Determine the majority class from the selected k examples. That is, the class most represented in the neighbouring k.
- Predict the entry example class as the majority class determined in step 3.
To apply this model, I used the train_test_split function of sklearn.model_selection. I created X_train, X_test, y_test and y_train. I set text_size to 2%, so I can train the model with more Data.
For our first model prediction, we obtain an accuracy score of 76.8%. This means that the ratio of successful predictions to all predictions is 76.8%, which is good but not enough.

# 4 HYPERPARAMETER

In this last step, we wanted to increase the accuracy score obtained previously by playing with the parameters of the KNN model, which can be found directly on the site of the sklearn library:
`https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

Moreover, I used the GridSearchCV library to be able to test each combination of parameters that I set myself. Now we have this :

```
model = KNeighborsClassifier()
parameters = {
              'n_neighbors' : list(range(1,22, 2)),
              'metric' : ['euclidean', 'manhattan', 'minkowski'],
              'weights' : ['uniform', 'distance']
             }

grid_cv = GridSearchCV(model, parameters, scoring = make_scorer(accuracy_score))
grid_cv = grid_cv.fit(X_train, y_train)
```

- n_neighbors is a parameter that specifies the number of neighbors to consider when predicting the class of an input example. Here, I make variation of this number between 1 and 21 but just odd number
- metric specifies the distance used to measure the proximity between the examples. Specifically, metric indicates the distance measurement to be used to calculate the distance between the input example and the training set examples.
- Finally, weights specifies the weight to be assigned to each neighbour when calculating the class prediction. Indeed, weights indicate how the distances between the input example and its neighbors are weighted when predicting the class.

Finaly, we get this :

Our optimized KNN model is:

```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan', n_neighbors=9, weights='distance')
```

- metric = manhattan, so the distance used is the sum of the horizontal and vertical distances between two points in space
- n_neighbors is 9, so the 9 examples closest to the input example will be considered for class prediction.
- weights is put on distance. By consequence neighbors are weighted according to their distance to the input example. The nearest neighbours have a higher weight than the more distant neighbours.

In conclusion, by applying these parameters to our model, we obtain an accuracy score of 80% or a gain of 3% compared to before. It is better than earlier but not enough again.

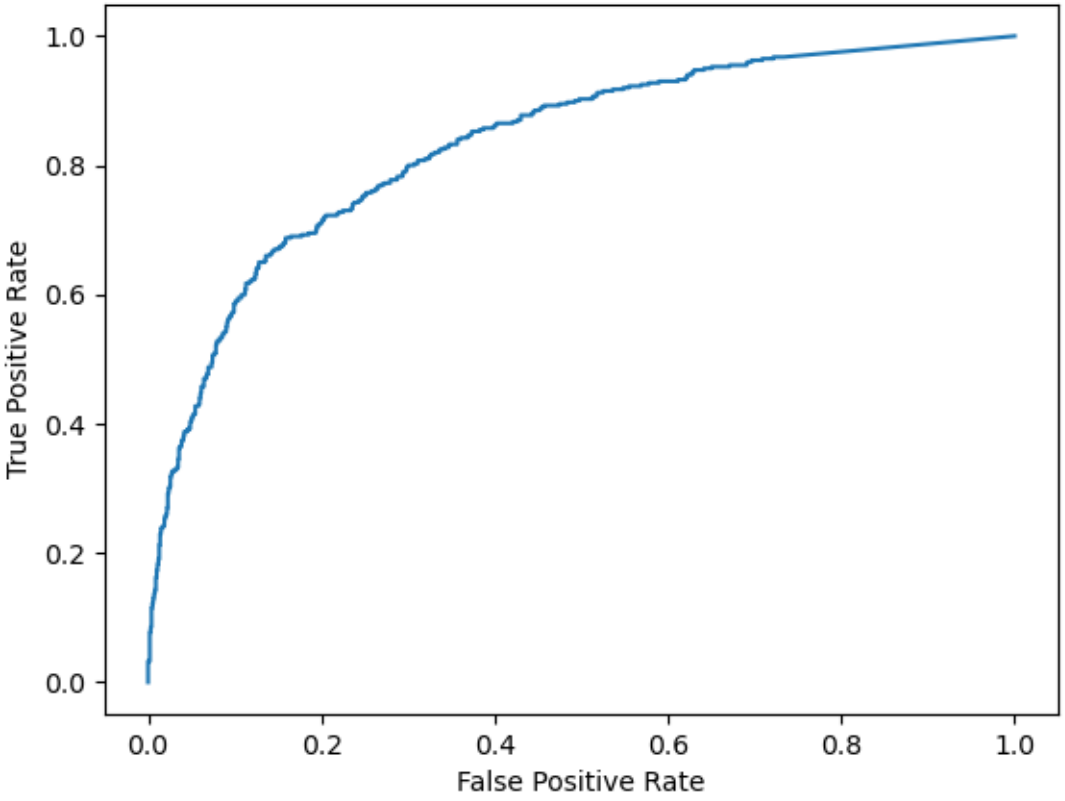
# 5 CONCLUSION

In this conclusion we analyse the result of my model:

First, we get the confusion matrix:

```
[[937  72]
 [205 195]]
```

We have 72 False Positive and 205 False Negative which is a lot. And we have a positive precision equals to 73%. p = TP/(TP+FP) = 195/(195 + 72) = 0.73 So our model is efficient, but still get errors. After, I plot a ROC Curve, which stands for “receiver operating characteristic” curve. This is a plot that displays the sensitivity and specificity of a logistic regression model.




The more the curve follows the top left corner of the graph, the better the model categorizes the data. As we can see from the above graph, this KNN model does a good job of classifying the data into categories, but maybe at the start of the curve, we don't have an efficient classification. To quantify this, we can calculate the AUC (area under the curve) which tells us which part of the graph is below the curve : auc = 0.83. The closer the AUC gets to 1, the better the model is. Here we have a sufficient value, not bad but not very efficient.

In conlusion our model fits well, but it could be better. So I can conclude that I maybe deleted the right data to use, or I didn't drop some Data which wasn't efficient in our case. It is also possible that relevant data exists for this project but that we did not have in our dataset. For example, maybe customer's age or job will play an important role because if you have a job which ask you to have a good telecommunication service, or less if you are a senior who generally use less this type of service. Or maybe add the data of a competing company

that tends to recover its customers.