**Github Link**: https://github.com/AbhinavThippani/FeatureEngineeringCA.git

**Wiki Content**

# Data Creation

```

# importing libraries

from GoogleNews import GoogleNews

from newspaper import Article

from newspaper import Config

import pandas as pd

import nltk

# configuring browser

user\_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36'

config = Config()

config.browser\_user\_agent = user\_agent

# searching the autonomous articles

googlenews=GoogleNews(start='01/01/2010',end='04/09/2021')

googlenews.search('Autonomous Cars')

result=googlenews.result()

df=pd.DataFrame(result)

# fetching the articles

for i in range(2,29):

googlenews.getpage(i)

result=googlenews.result()

df=pd.DataFrame(result)

# removing unavaialble articles

update\_df = df.drop([42,112,187,200,228])

# getting the full article and storing in excel

lst = []

for ind in update\_df.index:

d={}

article = Article(update\_df['link'][ind],config=config)

article.download()

article.parse()

article.nlp()

d['Date']=update\_df['date'][ind]

d['Media']=update\_df['media'][ind]

d['Title']=article.title

d['Article']=article.text

d['Summary']=article.summary

lst.append(d)

news\_df=pd.DataFrame(lst)

news\_df.to\_excel("articles.xlsx")

# reading bbc dataset

df\_bbc = pd.read\_csv('Y:\\Masters\_Content\\Feature\_Engineering\\Lab1\\Data\\News\_dataset.csv', sep=';')

# creating autonomous category

news\_df['Category'] = 'autonomous car'

# extracting the required columns and creating a new dataframe

df\_auto = news\_df[['Article','Summary','Category']]

df\_auto.rename(columns = {'Article':'File\_Name','Summary':'Content'},inplace = True)

# removing tech articles

df\_bbc = df\_bbc[df\_bbc['Category'] != 'tech'].reset\_index(drop = True)

# concatenating bbc with autonomous articles

frames = [df\_auto,df\_bbc]

df\_final\_news = pd.DataFrame(pd.concat(frames,ignore\_index = True))

# saving the dataframe to csv

df\_final\_news = df\_final\_news.to\_csv('Y:\\Masters\_Content\\Feature\_Engineering\\Lab1\\Data\dataset.csv')

```

# EDA

```

# importing libraries

import pandas as pd

import matplotlib.pyplot as plt

import pickle

import seaborn as sns

sns.set\_style("whitegrid")

import altair as alt

## Code for hiding warnings

import warnings

warnings.filterwarnings("ignore")

# loading dataset

df\_path = "Y:/Masters\_Content/Feature\_Engineering/Lab1/Data/"

df\_path2 = df\_path + 'dataset.csv'

df = pd.read\_csv(df\_path2)

# Number of articles in each category

bars = alt.Chart(df).mark\_bar(size=50).encode(

x=alt.X("Category"),

y=alt.Y("count():Q", axis=alt.Axis(title='Number of articles')),

tooltip=[alt.Tooltip('count()', title='Number of articles'), 'Category'],

color='Category'

)

text = bars.mark\_text(

align='center',

baseline='bottom',

).encode(

text='count()'

)

(bars + text).interactive().properties(

height=300,

width=700,

title = "Number of articles in each category",

)

```

![categories\_articles](https://user-images.githubusercontent.com/90108087/132138814-29d1ff73-b89d-429a-a98b-7beadfa754ef.png)

```

# % of articles in each category

df['id'] = 1

df2 = pd.DataFrame(df.groupby('Category').count()['id']).reset\_index()

bars = alt.Chart(df2).mark\_bar(size=50).encode(

x=alt.X('Category'),

y=alt.Y('PercentOfTotal:Q', axis=alt.Axis(format='.0%', title='% of Articles')),

color='Category'

).transform\_window(

TotalArticles='sum(id)',

frame=[None, None]

).transform\_calculate(

PercentOfTotal="datum.id / datum.TotalArticles"

)

text = bars.mark\_text(

align='center',

baseline='bottom',

#dx=5 # Nudges text to right so it doesn't appear on top of the bar

).encode(

text=alt.Text('PercentOfTotal:Q', format='.1%')

)

(bars + text).interactive().properties(

height=300,

width=700,

title = "% of articles in each category",

)

```

![categories\_percent](https://user-images.githubusercontent.com/90108087/132138817-ef2dfaf4-fced-4e9e-be06-5231020cc35a.png)

```

# News length by category

df['News\_length'] = df['Content'].str.len()

plt.figure(figsize=(12.8,6))

sns.distplot(df['News\_length']).set\_title('News length distribution');

```

![Height\_bar](https://user-images.githubusercontent.com/90108087/132138818-5c32ef5e-b969-4a7d-b008-6644843e09fe.png)

```

# Let's remove from the 95% percentile onwards to better appreciate the histogram:

quantile\_95 = df['News\_length'].quantile(0.95)

df\_95 = df[df['News\_length'] < quantile\_95]

plt.figure(figsize=(12.8,6))

sns.distplot(df\_95['News\_length']).set\_title('News length distribution');

```

![bar\_length](https://user-images.githubusercontent.com/90108087/132138813-c7876a31-6600-4677-956f-9e6de2049b4b.png)

```

# Let's now plot a boxplot:

plt.figure(figsize=(12.8,6))

sns.boxplot(data=df\_95, x='Category', y='News\_length');

```

![plotting\_boxes](https://user-images.githubusercontent.com/90108087/132138819-c6fcd4ae-594f-4e02-8272-254ff03a3d52.png)

```

# We'll save the dataset:

with open('News\_dataset.pickle', 'wb') as output:

pickle.dump(df, output)

```

# Feature Engineering

```

# importing libraries

import pickle

import pandas as pd

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_selection import chi2

import numpy as np

# First of all we'll load the dataset:

path\_df = r"Y:\Masters\_Content\Feature\_Engineering\Lab1\Codes\News\_dataset.pickle"

with open(path\_df, 'rb') as data:

df = pickle.load(data)

# 1. Text cleaning and preparation

# 1.1. Special character cleaning

df['Content\_Parsed\_1'] = df['Content'].str.replace("\r", " ")

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace("\n", " ")

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace(" ", " ")

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace('"', '')

# 1.2. Upcase/downcase

# Lowercasing the text

df['Content\_Parsed\_2'] = df['Content\_Parsed\_1'].str.lower()

# 1.3. Punctuation signs

punctuation\_signs = list("?:!.,;")

df['Content\_Parsed\_3'] = df['Content\_Parsed\_2']

for punct\_sign in punctuation\_signs:

df['Content\_Parsed\_3'] = df['Content\_Parsed\_3'].str.replace(punct\_sign, '')

# 1.4. Possessive pronouns

df['Content\_Parsed\_4'] = df['Content\_Parsed\_3'].str.replace("'s", "")

# 1.5. Stemming and Lemmatization

# Saving the lemmatizer into an object

wordnet\_lemmatizer = WordNetLemmatizer()

df = df.dropna().reset\_index(drop = True)

nrows = len(df)

lemmatized\_text\_list = []

for row in range(0, nrows):

# Create an empty list containing lemmatized words

lemmatized\_list = []

# Save the text and its words into an object

text = df.loc[row]['Content\_Parsed\_4']

text\_words = text.split(" ")

# Iterate through every word to lemmatize

for word in text\_words:

lemmatized\_list.append(wordnet\_lemmatizer.lemmatize(word, pos="v"))

# Join the list

lemmatized\_text = " ".join(lemmatized\_list)

# Append to the list containing the texts

lemmatized\_text\_list.append(lemmatized\_text)

1.6. Stop words

# Loading the stop words in english

stop\_words = list(stopwords.words('english'))

df['Content\_Parsed\_6'] = df['Content\_Parsed\_5']

for stop\_word in stop\_words:

regex\_stopword = r"\b" + stop\_word + r"\b"

df['Content\_Parsed\_6'] = df['Content\_Parsed\_6'].str.replace(regex\_stopword, '')

df['Complete\_Filename'] = df['File\_Name'] + '-' + df['Category']

list\_columns = ["File\_Name", "Category", "Complete\_Filename", "Content", "Content\_Parsed\_6"]

df = df[list\_columns]

df = df.rename(columns={'Content\_Parsed\_6': 'Content\_Parsed'})

# 2. Label coding

category\_codes = {

'business': 0,

'entertainment': 1,

'politics': 2,

'sport': 3,

'autonomous car': 4

}

# Category mapping

df['Category\_Code'] = df['Category']

df = df.replace({'Category\_Code':category\_codes})

3. Train - test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['Content\_Parsed'],

df['Category\_Code'],

test\_size=0.15,

random\_state=8)

4. Text representation

# Parameter election

ngram\_range = (1,2)

min\_df = 10

max\_df = 1.

max\_features = 300

tfidf = TfidfVectorizer(encoding='utf-8',

ngram\_range=ngram\_range,

stop\_words=None,

lowercase=False,

max\_df=max\_df,

min\_df=min\_df,

max\_features=max\_features,

norm='l2',

sublinear\_tf=True)

features\_train = tfidf.fit\_transform(X\_train).toarray()

labels\_train = y\_train

print(features\_train.shape)

features\_test = tfidf.transform(X\_test).toarray()

labels\_test = y\_test

print(features\_test.shape)

# We can use the Chi squared test in order to see what unigrams and bigrams are most correlated with each category:¶

from sklearn.feature\_selection import chi2

import numpy as np

for Product, category\_id in sorted(category\_codes.items()):

features\_chi2 = chi2(features\_train, labels\_train == category\_id)

indices = np.argsort(features\_chi2[0])

feature\_names = np.array(tfidf.get\_feature\_names())[indices]

unigrams = [v for v in feature\_names if len(v.split(' ')) == 1]

bigrams = [v for v in feature\_names if len(v.split(' ')) == 2]

print("# '{}' category:".format(Product))

print(" . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-5:])))

print(" . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-2:])))

print("")

# Let's save the files we'll need in the next steps:

# X\_train

with open('Pickles/X\_train.pickle', 'wb') as output:

pickle.dump(X\_train, output)

# X\_test

with open('Pickles/X\_test.pickle', 'wb') as output:

pickle.dump(X\_test, output)

# y\_train

with open('Pickles/y\_train.pickle', 'wb') as output:

pickle.dump(y\_train, output)

# y\_test

with open('Pickles/y\_test.pickle', 'wb') as output:

pickle.dump(y\_test, output)

# df

with open('Pickles/df.pickle', 'wb') as output:

pickle.dump(df, output)

# features\_train

with open('Pickles/features\_train.pickle', 'wb') as output:

pickle.dump(features\_train, output)

# labels\_train

with open('Pickles/labels\_train.pickle', 'wb') as output:

pickle.dump(labels\_train, output)

# features\_test

with open('Pickles/features\_test.pickle', 'wb') as output:

pickle.dump(features\_test, output)

# labels\_test

with open('Pickles/labels\_test.pickle', 'wb') as output:

pickle.dump(labels\_test, output)

# TF-IDF object

with open('Pickles/tfidf.pickle', 'wb') as output:

pickle.dump(tfidf, output)

```

# Model Training

### Random Forest

```

# importing libraries

import pickle

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from pprint import pprint

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.model\_selection import ShuffleSplit

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Loading Data

path\_df = "Pickles/df.pickle"

with open(path\_df, 'rb') as data:

df = pickle.load(data)

# features\_train

path\_features\_train = "Pickles/features\_train.pickle"

with open(path\_features\_train, 'rb') as data:

features\_train = pickle.load(data)

# labels\_train

path\_labels\_train = "Pickles/labels\_train.pickle"

with open(path\_labels\_train, 'rb') as data:

labels\_train = pickle.load(data)

# features\_test

path\_features\_test = "Pickles/features\_test.pickle"

with open(path\_features\_test, 'rb') as data:

features\_test = pickle.load(data)

# labels\_test

path\_labels\_test = "Pickles/labels\_test.pickle"

with open(path\_labels\_test, 'rb') as data:

labels\_test = pickle.load(data)

# Cross-Validation for Hyperparameter tuning

rf\_0 = RandomForestClassifier(random\_state = 8)

print('Parameters currently in use:\n')

pprint(rf\_0.get\_params())

# Randomized Search Cross Validation

# n\_estimators

n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)]

# max\_features

max\_features = ['auto', 'sqrt']

# max\_depth

max\_depth = [int(x) for x in np.linspace(20, 100, num = 5)]

max\_depth.append(None)

# min\_samples\_split

min\_samples\_split = [2, 5, 10]

# min\_samples\_leaf

min\_samples\_leaf = [1, 2, 4]

# bootstrap

bootstrap = [True, False]

# Create the random grid

random\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

pprint(random\_grid)

# First create the base model to tune

rfc = RandomForestClassifier(random\_state=8)

# Definition of the random search

random\_search = RandomizedSearchCV(estimator=rfc,

param\_distributions=random\_grid,

n\_iter=50,

scoring='accuracy',

cv=3,

verbose=1,

random\_state=8)

# Fit the random search model

random\_search.fit(features\_train, labels\_train)

print("The best hyperparameters from Random Search are:")

print(random\_search.best\_params\_)

print("")

print("The mean accuracy of a model with these hyperparameters is:")

print(random\_search.best\_score\_)

# Grid Search Cross Validation

# Create the parameter grid based on the results of random search

bootstrap = [False]

max\_depth = [30, 40, 50]

max\_features = ['sqrt']

min\_samples\_leaf = [1, 2, 4]

min\_samples\_split = [5, 10, 15]

n\_estimators = [800]

param\_grid = {

'bootstrap': bootstrap,

'max\_depth': max\_depth,

'max\_features': max\_features,

'min\_samples\_leaf': min\_samples\_leaf,

'min\_samples\_split': min\_samples\_split,

'n\_estimators': n\_estimators

}

# Create a base model

rfc = RandomForestClassifier(random\_state=8)

# Manually create the splits in CV in order to be able to fix a random\_state (GridSearchCV doesn't have that argument)

cv\_sets = ShuffleSplit(n\_splits = 3, test\_size = .33, random\_state = 8)

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator=rfc,

param\_grid=param\_grid,

scoring='accuracy',

cv=cv\_sets,

verbose=1)

# Fit the grid search to the data

grid\_search.fit(features\_train, labels\_train)

print("The best hyperparameters from Grid Search are:")

print(grid\_search.best\_params\_)

print("")

print("The mean accuracy of a model with these hyperparameters is:")

print(grid\_search.best\_score\_)

best\_rfc = grid\_search.best\_estimator\_

best\_rfc

# Model fit and performance

best\_rfc.fit(features\_train, labels\_train)

rfc\_pred = best\_rfc.predict(features\_test)

# Training accuracy

print("The training accuracy is: ")

print(accuracy\_score(labels\_train, best\_rfc.predict(features\_train)))

# Test accuracy

print("The test accuracy is: ")

print(accuracy\_score(labels\_test, rfc\_pred))

# Classification report

print("Classification report")

print(classification\_report(labels\_test,rfc\_pred))

```

![Algo\_CR](https://user-images.githubusercontent.com/90108087/132138811-1769cc34-1226-445d-a240-86c05c142b30.JPG)

```

# Confusion matrix

aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category\_Code')

conf\_matrix = confusion\_matrix(labels\_test, rfc\_pred)

plt.figure(figsize=(12.8,6))

sns.heatmap(conf\_matrix,

annot=True,

xticklabels=aux\_df['Category'].values,

yticklabels=aux\_df['Category'].values,

cmap="Blues")

plt.ylabel('Predicted')

plt.xlabel('Actual')

plt.title('Confusion matrix')

plt.show()

```

![Algo\_naive\_CM](https://user-images.githubusercontent.com/90108087/132138812-ac3fa199-16a0-40ff-8724-30e522ce25d9.png)

```

# Let's save the model and this dataset:

with open('Models/best\_rfc.pickle', 'wb') as output:

pickle.dump(best\_rfc, output)

with open('Models/df\_models\_rfc.pickle', 'wb') as output:

pickle.dump(df\_models\_rfc, output)

```

## Support Vector Machine

```

# importing libraries

import pickle

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn import svm

from pprint import pprint

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.model\_selection import ShuffleSplit

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# loading data

# Dataframe

path\_df = "Pickles/df.pickle"

with open(path\_df, 'rb') as data:

df = pickle.load(data)

# features\_train

path\_features\_train = "Pickles/features\_train.pickle"

with open(path\_features\_train, 'rb') as data:

features\_train = pickle.load(data)

# labels\_train

path\_labels\_train = "Pickles/labels\_train.pickle"

with open(path\_labels\_train, 'rb') as data:

labels\_train = pickle.load(data)

# features\_test

path\_features\_test = "Pickles/features\_test.pickle"

with open(path\_features\_test, 'rb') as data:

features\_test = pickle.load(data)

# labels\_test

path\_labels\_test = "Pickles/labels\_test.pickle"

with open(path\_labels\_test, 'rb') as data:

labels\_test = pickle.load(data)

# Cross-Validation for Hyperparameter tuning

svc\_0 =svm.SVC(random\_state=8)

print('Parameters currently in use:\n')

pprint(svc\_0.get\_params())

# Randomized Search Cross Validation

C = [.0001, .001, .01]

# gamma

gamma = [.0001, .001, .01, .1, 1, 10, 100]

# degree

degree = [1, 2, 3, 4, 5]

# kernel

kernel = ['linear', 'rbf', 'poly']

# probability

probability = [True]

# Create the random grid

random\_grid = {'C': C,

'kernel': kernel,

'gamma': gamma,

'degree': degree,

'probability': probability

}

pprint(random\_grid)

# First create the base model to tune

svc = svm.SVC(random\_state=8)

# Definition of the random search

random\_search = RandomizedSearchCV(estimator=svc,

param\_distributions=random\_grid,

n\_iter=50,

scoring='accuracy',

cv=3,

verbose=1,

random\_state=8)

# Fit the random search model

random\_search.fit(features\_train, labels\_train)

print("The best hyperparameters from Random Search are:")

print(random\_search.best\_params\_)

print("")

print("The mean accuracy of a model with these hyperparameters is:")

print(random\_search.best\_score\_)

# Grid Search Cross Validation

# Create the parameter grid based on the results of random search

C = [.0001, .001, .01, .1]

degree = [3, 4, 5]

gamma = [1, 10, 100]

probability = [True]

param\_grid = [

{'C': C, 'kernel':['linear'], 'probability':probability},

{'C': C, 'kernel':['poly'], 'degree':degree, 'probability':probability},

{'C': C, 'kernel':['rbf'], 'gamma':gamma, 'probability':probability}

]

# Create a base model

svc = svm.SVC(random\_state=8)

# Manually create the splits in CV in order to be able to fix a random\_state (GridSearchCV doesn't have that argument)

cv\_sets = ShuffleSplit(n\_splits = 3, test\_size = .33, random\_state = 8)

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator=svc,

param\_grid=param\_grid,

scoring='accuracy',

cv=cv\_sets,

verbose=1)

# Fit the grid search to the data

grid\_search.fit(features\_train, labels\_train)

print("The best hyperparameters from Grid Search are:")

print(grid\_search.best\_params\_)

print("")

print("The mean accuracy of a model with these hyperparameters is:")

print(grid\_search.best\_score\_)

best\_svc = grid\_search.best\_estimator\_

# Model fit and performance

best\_svc.fit(features\_train, labels\_train)

svc\_pred = best\_svc.predict(features\_test)

# Training accuracy

print("The training accuracy is: ")

print(accuracy\_score(labels\_train, best\_svc.predict(features\_train)))

# Test accuracy

print("The test accuracy is: ")

print(accuracy\_score(labels\_test, svc\_pred))

# Classification report

print("Classification report")

print(classification\_report(labels\_test,svc\_pred))

```

![Report\_Values](https://user-images.githubusercontent.com/90108087/132138821-de71911e-e3d2-44d5-ba1a-7b3a5af47341.JPG)

```

# confusion matrix

aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category\_Code')

conf\_matrix = confusion\_matrix(labels\_test, svc\_pred)

plt.figure(figsize=(12.8,6))

sns.heatmap(conf\_matrix,

annot=True,

xticklabels=aux\_df['Category'].values,

yticklabels=aux\_df['Category'].values,

cmap="Blues")

plt.ylabel('Predicted')

plt.xlabel('Actual')

plt.title('Confusion matrix')

plt.show()

```

![categories\_data](https://user-images.githubusercontent.com/90108087/132138816-3abd59d7-a2e6-48f2-aca9-aa373253437d.png)

```

Let's save the model and this dataset:

with open('Models/best\_svc.pickle', 'wb') as output:

pickle.dump(best\_svc, output)

with open('Models/df\_models\_svc.pickle', 'wb') as output:

pickle.dump(df\_models\_svc, output)

```

## Multinomial Naïve Bayes

```

# importing libraries

import pickle

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from pprint import pprint

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.model\_selection import ShuffleSplit

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# loading data

# Dataframe

path\_df = "Pickles/df.pickle"

with open(path\_df, 'rb') as data:

df = pickle.load(data)

# features\_train

path\_features\_train = "Pickles/features\_train.pickle"

with open(path\_features\_train, 'rb') as data:

features\_train = pickle.load(data)

# labels\_train

path\_labels\_train = "Pickles/labels\_train.pickle"

with open(path\_labels\_train, 'rb') as data:

labels\_train = pickle.load(data)

# features\_test

path\_features\_test = "Pickles/features\_test.pickle"

with open(path\_features\_test, 'rb') as data:

features\_test = pickle.load(data)

# labels\_test

path\_labels\_test = "Pickles/labels\_test.pickle"

with open(path\_labels\_test, 'rb') as data:

labels\_test = pickle.load(data)

# Cross-Validation for Hyperparameter tuning

mnbc = MultinomialNB()

# Model fit and performance

mnbc.fit(features\_train, labels\_train)

mnbc\_pred = mnbc.predict(features\_test)

# Training accuracy

print("The training accuracy is: ")

print(accuracy\_score(labels\_train, mnbc.predict(features\_train)))

# Test accuracy

print("The test accuracy is: ")

print(accuracy\_score(labels\_test, mnbc\_pred))

# Classification report

print("Classification report")

print(classification\_report(labels\_test,mnbc\_pred))

```

![Sets\_values](https://user-images.githubusercontent.com/90108087/132138823-fbb8fa2b-397e-43e2-94c0-56b5e6e11d48.JPG)

```

aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category\_Code')

conf\_matrix = confusion\_matrix(labels\_test, mnbc\_pred)

plt.figure(figsize=(12.8,6))

sns.heatmap(conf\_matrix,

annot=True,

xticklabels=aux\_df['Category'].values,

yticklabels=aux\_df['Category'].values,

cmap="Blues")

plt.ylabel('Predicted')

plt.xlabel('Actual')

plt.title('Confusion matrix')

plt.show()

```

![predict\_matrix](https://user-images.githubusercontent.com/90108087/132138820-3e61f1b0-0b7a-491f-b441-60c8c43cdcc3.png)

```

# Let's save the model and this dataset:

with open('Models/best\_mnbc.pickle', 'wb') as output:

pickle.dump(mnbc, output)

with open('Models/df\_models\_mnbc.pickle', 'wb') as output:

pickle.dump(df\_models\_mnbc, output)

```