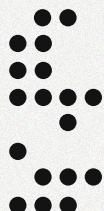




# Mayor información por comentario

Uso de **NLP** para bases más robustas y mejor toma de decisiones



Matías Karmelic



# ¿Por qué un modelo NLP?



## Miles de comentarios

En la industria digital, se reciben miles de comentarios para productos digitales al día.

Nos parece importante categorizar de mejor manera esta información, que es clave para la toma de decisiones.



## Sin categorización

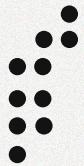
Las plataformas que reciben estos comentarios, generalmente incluyen una categorización base de 1 a 5 estrellas, o nota.



## +Información

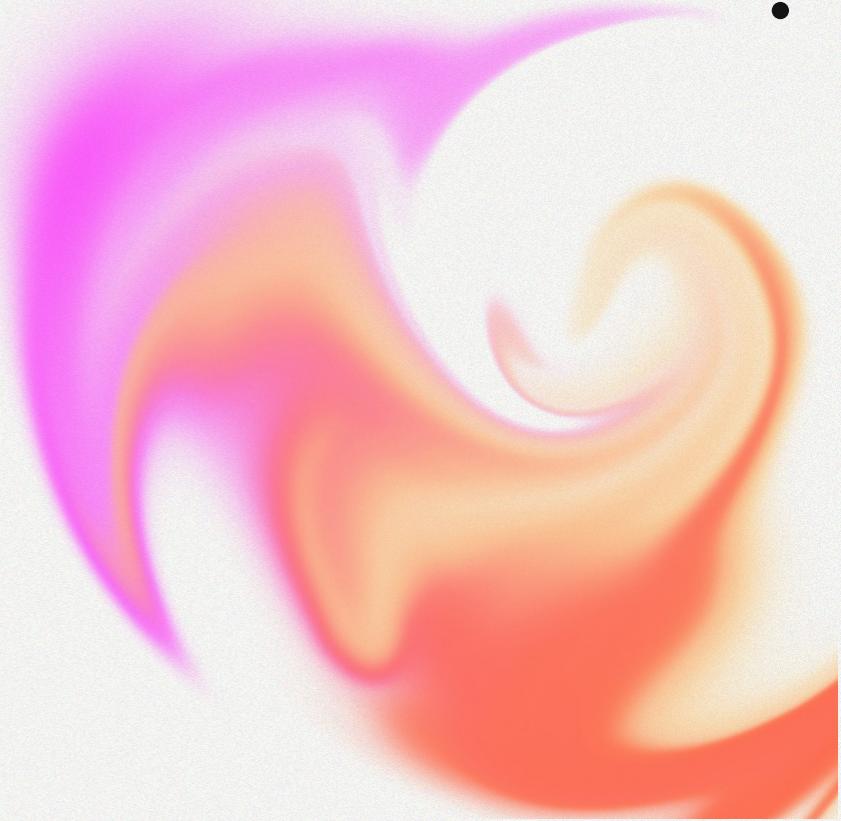
Más información significa mejores decisiones, ¿De dónde proviene la mayor cantidad de comentarios positivos? ¿Cuales son los usuarios más subjetivos?





# Google Play Store Apps

- Presenta 2 bases de datos con reviews por app y comentarios por separado.
- Diferentes tipos de datos a mano, más una evaluación ya realizada del tipo de comentario y lo subjetivo que es.
- Revisamos ambas bases antes de tomar una ruta de trabajo



# 01

# EDA



# EDA BBDD Ratings

**Buscamos los valores únicos por columna, los ordenamos:**

```
clean_df1['Genres'].sort_values().unique()  
clean_df1['Category'].sort_values().unique()
```

**hacemos una limpieza y orden de los valores en la columna de generos:**

```
clean_df1['Genres'].sort_values().unique()  
clean_df1['Genres'] = clean_df1['Genres'].apply(lambda x: x.split(';')[0])  
clean_df1.loc[clean_df1['Genres']=="Educational",'Genres'] = "Education"  
clean_df1['Genres'] = clean_df1['Genres'].replace("Music & Audio","Music")  
frequencies = clean_df1['Genres'].value_counts()
```

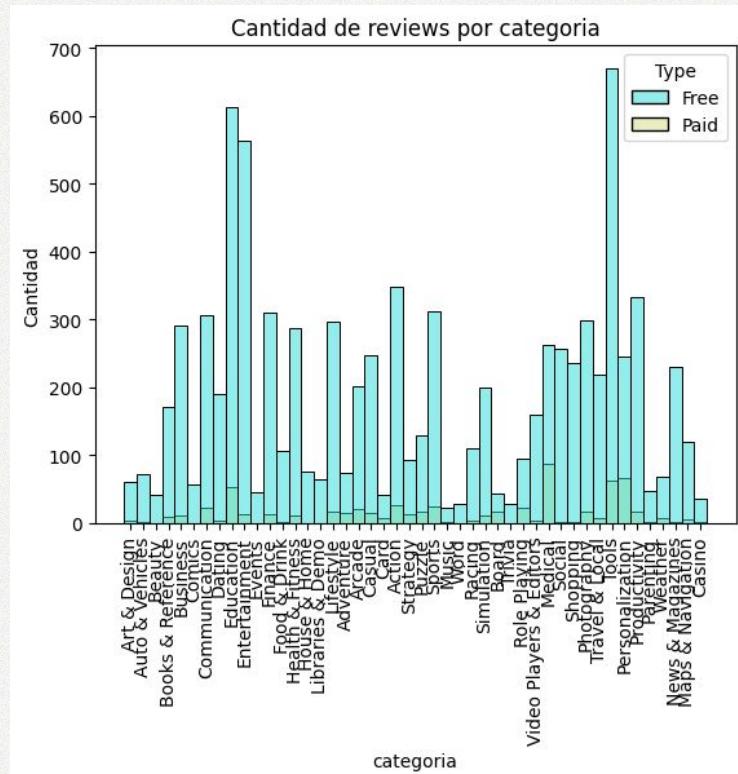


# EDA BBDD Ratings

**En una revisión final, vemos cuantos reviews hay por categoría:**

```
histplot(clean_df1, x='Genres', hue='Type', palette='rainbow')  
plt.xticks(rotation=90)  
plt.title('Cantidad de reviews por categoría')  
plt.xlabel('categoría')  
plt.ylabel('Cantidad')
```

# Cantidad de Reviews por categoría



\*BBDD con  
comentarios.



# EDA BBDD Ratings

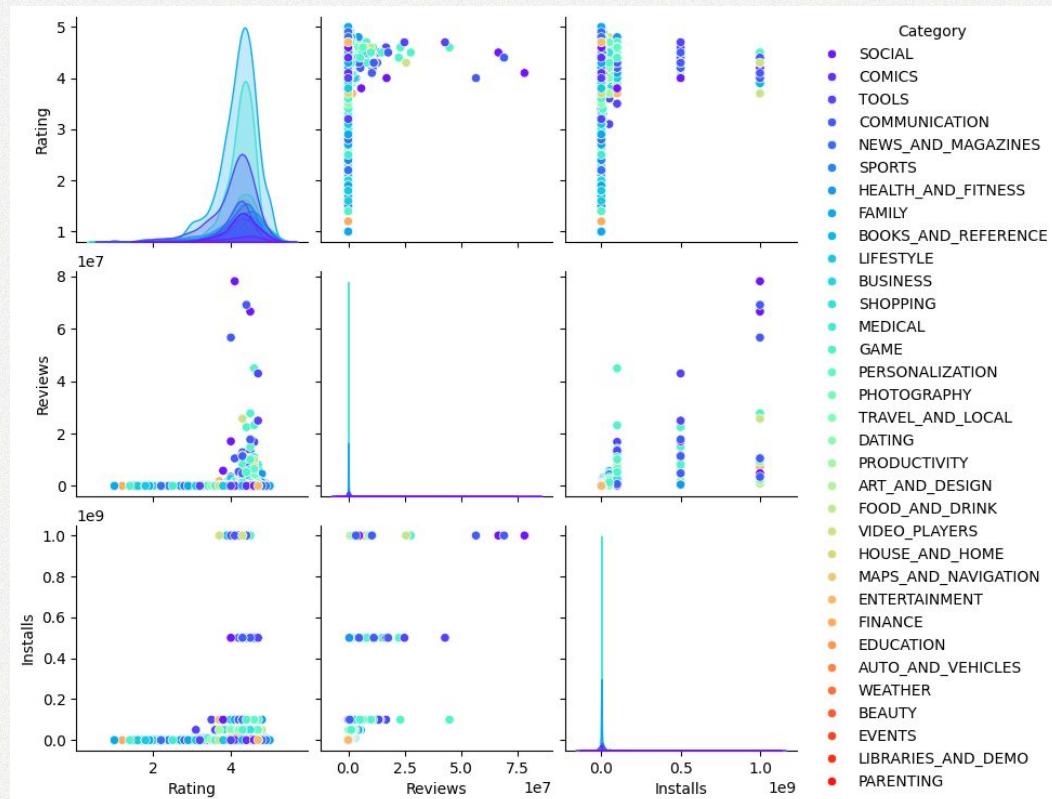
**Limpiamos valores como la cantidad de instalaciones, las fechas convertidas a datetime:**

```
clean_df1['Installs'] = clean_df1['Installs'].str.replace(',', '').str.replace('+', '')  
clean_df1['Installs'] = clean_df1['Installs'].astype(int)  
clean_df1['Reviews'] = clean_df1['Reviews'].astype(int)  
clean_df1['Last Updated'] = pd.to_datetime(clean_df1['Last Updated'])
```

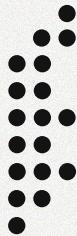
**Eliminamos los reviews duplicados y nos quedamos solo con los mas recientes:**

```
idx_most_recent = clean_df1.groupby('App')['Last Updated'].idxmax()  
filtered_df = clean_df1.loc[idx_most_recent]
```

# Algunos cruces de la información



\*BBDD con ratings  
de aplicaciones



# Conclusiones

## BBDD

Observamos que en esta bbdd la información tiene algunos cruces interesantes, sin embargo el potencial de predicción al que podemos llegar nos parece menos interesante que el de la segunda base.

## Cruces

Vemos algunos potenciales cruces de información, sobretodo en los géneros de las aplicaciones y cómo esto impacta las descargas y reviews de una app.

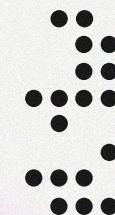
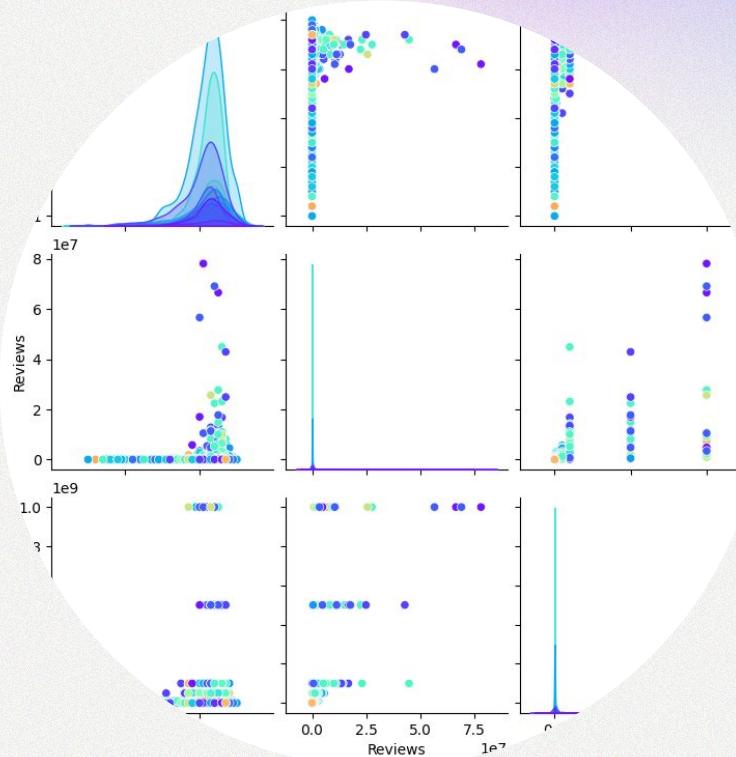
## Potencial

Observamos un posible modelo predictivo para categorías y evaluaciones. Vemos la posibilidad de prever\* que tipo de evaluación una aplicación tendrá según su categorización, nombre y descargas.

\*En este caso sin embargo, creemos que dada la base de datos, es una predicción que estaría sesgada por los datos mismos, considerando que es difícil predecir sin conocer el desempeño de la potencial aplicación y como los comentarios impactan sobre el rating.

# Decidimos trabajar con la siguiente base

Personalmente, creemos que un modelo de NLP nos permitirá acceso a información nueva y rica respecto a los comentarios de una posible aplicación digital.





# ¿Qué vimos en estas bases?



## Potencial

Poder interpretar con precisión comentarios de productos digitales nos permite clasificarlos de mejor manera.

Esto nos da acceso a una nueva dimensión de la información, no sólo la evaluación de 1 a 5, sumamos una valoración del producto en base a las palabras y que tan ligado a lo subjetivo se encuentra el comentario.



02

# Preprocesado

# Base inicial

Podemos observar la estructura inicial que tenía nuestra base, previo que trabajaríamos y la limpiaramos.

	App	Translated_Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive	1.00	0.533333
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive	0.25	0.288462
2	10 Best Foods for You		NaN	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive	0.40	0.875000
4	10 Best Foods for You	Best idea us	Positive	1.00	0.300000

**Primero definimos los datos a utilizar y eliminamos los valores NaN:**

```
data=pd.concat([df2.Translated_Review,df2.Sentiment,df2.Sentiment_Subjectivity],axis=1)  
data.dropna(axis=0,inplace=True)
```

# Preprocesado

	Translated_Review	Sentiment	Sentiment_Subjectivity
0	I like eat delicious food. That's I'm cooking ...	Positive	0.533333
1	This help eating healthy exercise regular basis	Positive	0.288462
3	Works great especially going grocery store	Positive	0.875000
4	Best idea us	Positive	0.300000
5	Best way	Positive	0.300000
6	Amazing	Positive	0.900000
8	Looking forward app,	Neutral	0.000000
9	It helpful site ! It help foods get !	Neutral	0.000000
10	good you.	Positive	0.600000
11	Useful information The amount spelling errors ...	Positive	0.100000

**Esto nos entrega la base con la que finalmente trabajaremos**



# Preprocesado

**Transformamos la data de texto en valores numéricos:**

```
data.Sentiment=[0 if i=="Positive" else 1 if i=="Negative" else 2 for i in data.Sentiment]
```

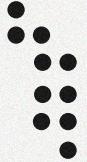
**Eliminamos caracteres especiales y todo el texto queda sin mayúsculas:**

```
import re
first_text=data.Translated_Review[0]
text=re.sub("[^a-zA-Z]","",first_text) #changing characters with space
text=text.lower()
```

**Quitamos las palabras de parada y separamos todas las palabras en cada comentario:**

```
import nltk

from nltk.corpus import stopwords
text = nltk.word_tokenize(text)
```



# Preprocesado

**Aplicamos Lemmatizer para poder hacer mejor cruce entre cada comentario:**

```
nltk.download('wordnet')

lemma=nltk.WordNetLemmatizer()
text=[lemma.lemmatize(i) for i in text]
text=" ".join(text)
```

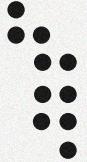
**Con este paso, pasamos las palabras a su origen, esto nos permite que el model cree mejor cruce entre las palabras y sepa identificarlas con mayor precisión**



# Preprocesado

**Nuestro proceso completo de preprocesado se ve algo así:**

```
text_list=[]
for i in data.Translated_Review:
    text=re.sub("[^a-zA-Z]","",i)
    text=text.lower()
    text=nltk.word_tokenize(text)
    lemma=nltk.WordNetLemmatizer()
    text=[lemma.lemmatize(word) for word in text]
    text=" ".join(text)
    text_list.append(text)
```



# Tokenization

**Tokenization con CountVectorizer de sklearn:**

```
cou_vec=CountVectorizer(max_features=max_features,stop_words="english")
sparse_matrix=cou_vec.fit_transform(text_list).toarray()
all_words=cou_vec.get_feature_names_out()
```

**Visualizamos las palabras con mayor presencia:**

```
from wordcloud import WordCloud
plt.subplots(figsize=(8,8))
wordcloud=WordCloud(background_color="white",width=1024,height=768).generate(
".join(all_words[100:]))"
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



# Frecuencia de palabras en la BBDD



\*BBDD con  
comentarios.

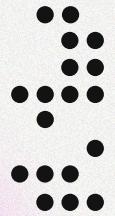


# Texto procesado

## Antes y después del tratamiento de texto

I like eat delicious food. That's I'm cooking food myself, case "10 Best Foods" helps lot, also "Best Before (Shelf Life)"

i like eat delicious food that s i m cooking food myself case best food help lot also best before shelf life



# 03

# Modelado



# Modelado

**Armamos set de entrenamiento con el contenido previo:**

```
y=data.iloc[:,1].values  
X=sparce_matrix  
from sklearn.model_selection import train_test_split  
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=42)
```

**Primer modelo, Random Forest:**

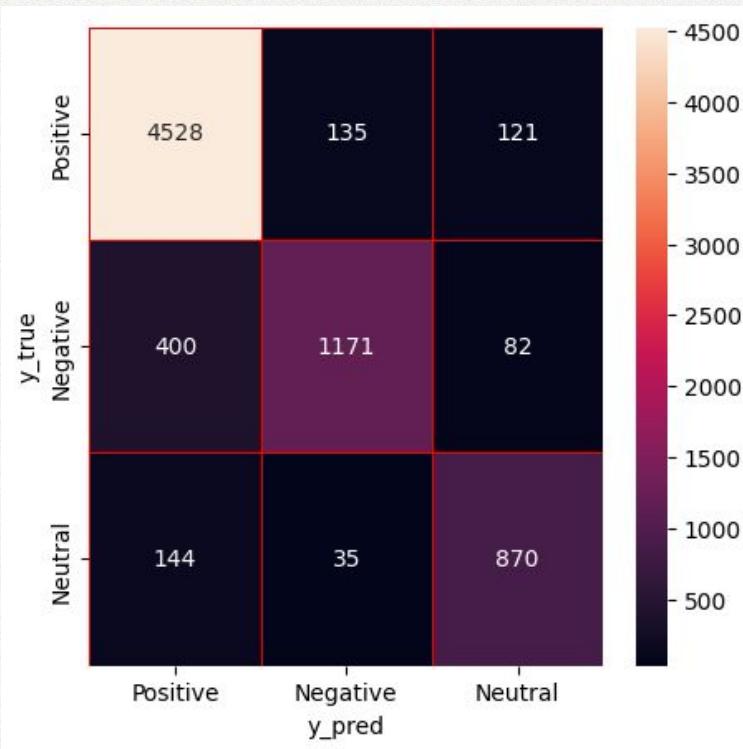
```
rf = RandomForestClassifier(n_estimators = 10, random_state=42)  
rf.fit(X_train,y_train)  
print("accuracy: ",rf.score(X_test,y_test))
```

**Precisión media en validación cruzada:**

**0.8777609934423115**



# Matriz de confusión modelo RF





# Modelado

## **Modelo de Regresión Logística:**

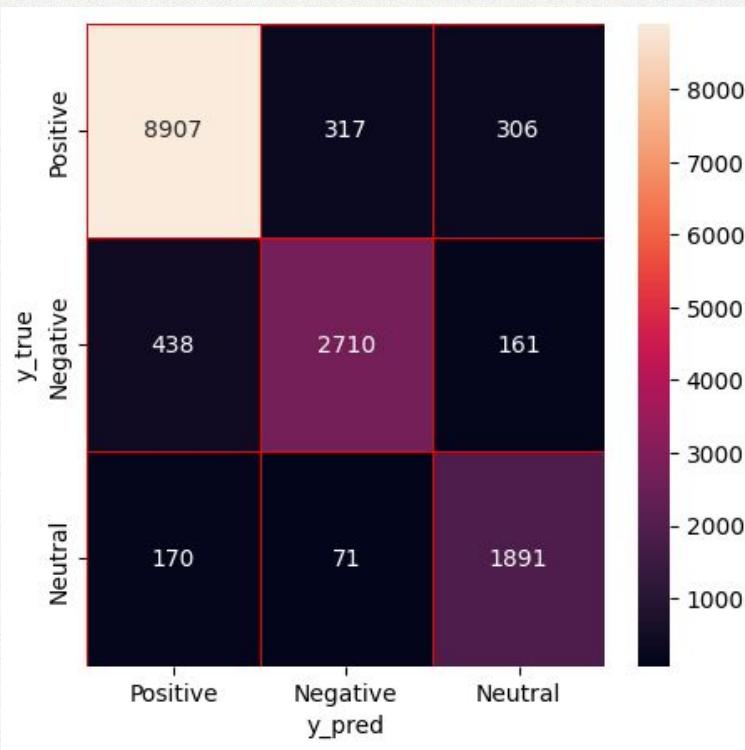
```
from sklearn.linear_model import LogisticRegression  
lr=LogisticRegression()  
lr.fit(X_train,y_train)
```

## **Precisión media en validación cruzada:**

**0.9022777369581191**



# Matriz de confusión modelo LR





# Optimización del Modelado

## Búsqueda de los mejores hyperparameters:

```
def find_best_logistic_regression_params(X_train, y_train):  
    Define the parameter grid including different solvers  
    param_grid = { 'C': [0.001, 0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2'],  
                  'solver': ['lbfgs', 'newton-cg', 'sag', 'saga'] }
```

Create a logistic regression classifier  
`logistic_regression = LogisticRegression(verbose=1)`

Create GridSearchCV object  
`grid_search = GridSearchCV(estimator=logistic_regression, param_grid=param_grid, cv=5)`

Fit the grid search to find the best parameters  
`grid_search.fit(X_train, y_train)`



# Optimización del Modelado

## Modelo de Regresión Logística Optimizado:

```
Optimized_Ir = LogisticRegression(C= 1, penalty= 'l2', solver='newton-cg', max_iter=200)  
Optimized_Ir.fit(X_train,y_train)
```

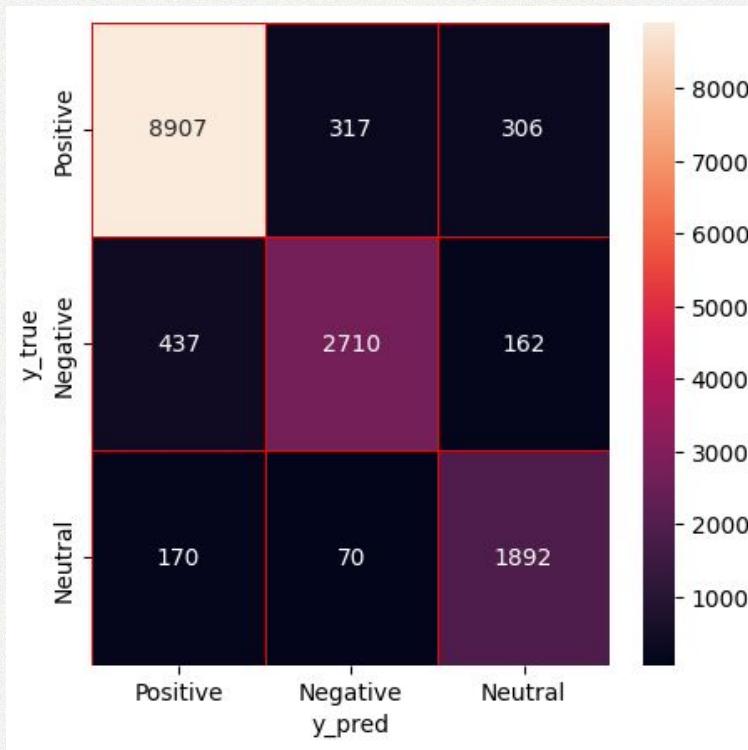
## Precisión media en validación cruzada:

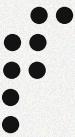
0.9023445327633425

**Nuestra ganancia es menor. Decidimos mantener este modelo de todos modos, ya que presenta sobre 90% de precisión**



# Matriz de confusión modelo Optimizado





# +90%

Es nuestro porcentaje de precisión para la evaluación de Sentimiento de comentario





# Modelado N2

**Creamos un segundo modelo, que esta vez nos muestre lo subjetivo de cada comentario:**

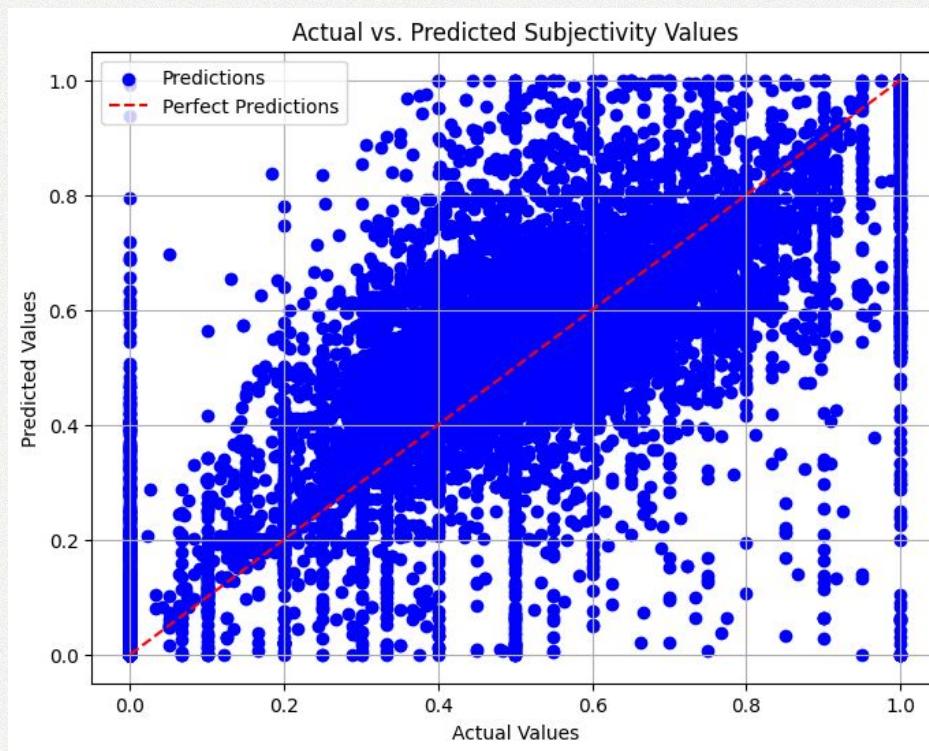
```
y_subjective=data.iloc[:,2].values  
X=sparce_matrix  
from sklearn.model_selection import train_test_split  
X_train,X_test,y_subjective_train,y_subjective_test=train_test_split(X,y_subjective,test_size=0.4,random_state=42)
```

**Entrenamos nuestro segundo modelo:**

```
from sklearn.ensemble import RandomForestRegressor  
subjective_rfr = RandomForestRegressor(n_estimators = 30, random_state=42)  
subjective_rfr.fit(X_train,y_subjective_train)
```

**accuracy: 0.6902306149704379**

# Predictión vs reales modelo RFR



\*La linea roja es el punto ideal, aca se nota el 69%



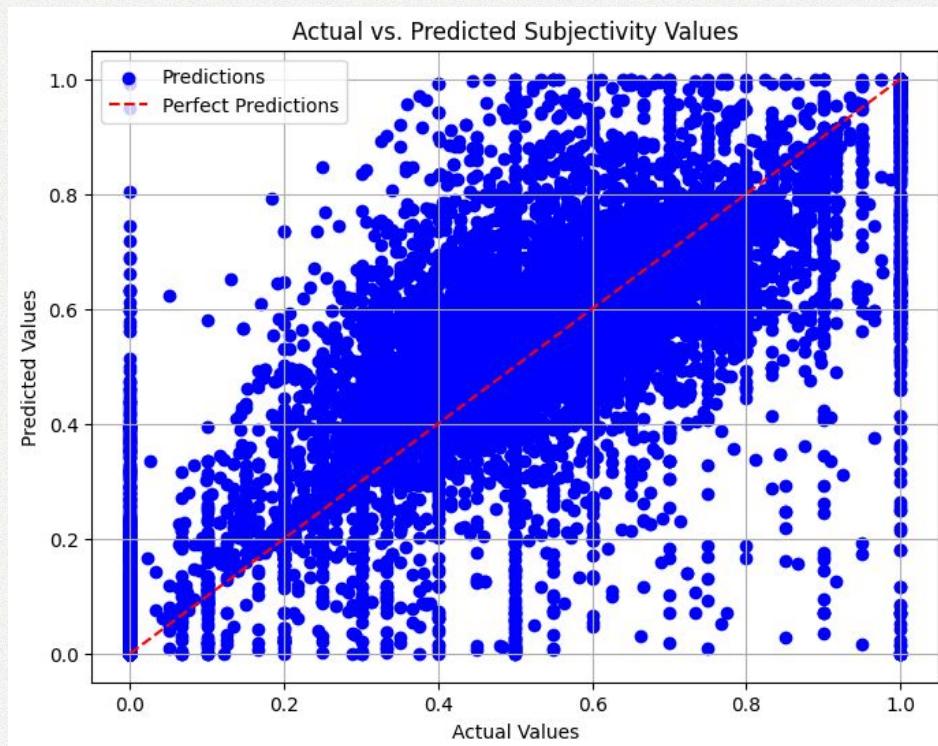
# Optimización del Modelado N2

**Leves ajustes al modelo:**

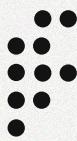
```
subjective_rfr2 = RandomForestRegressor(n_estimators = 60, random_state=42)
subjective_rfr2.fit(X_train,y_subjective_train)
print("accuracy: ",subjective_rfr2.score(X_test,y_subjective_test))
```

**accuracy: 0.6940935374752735**

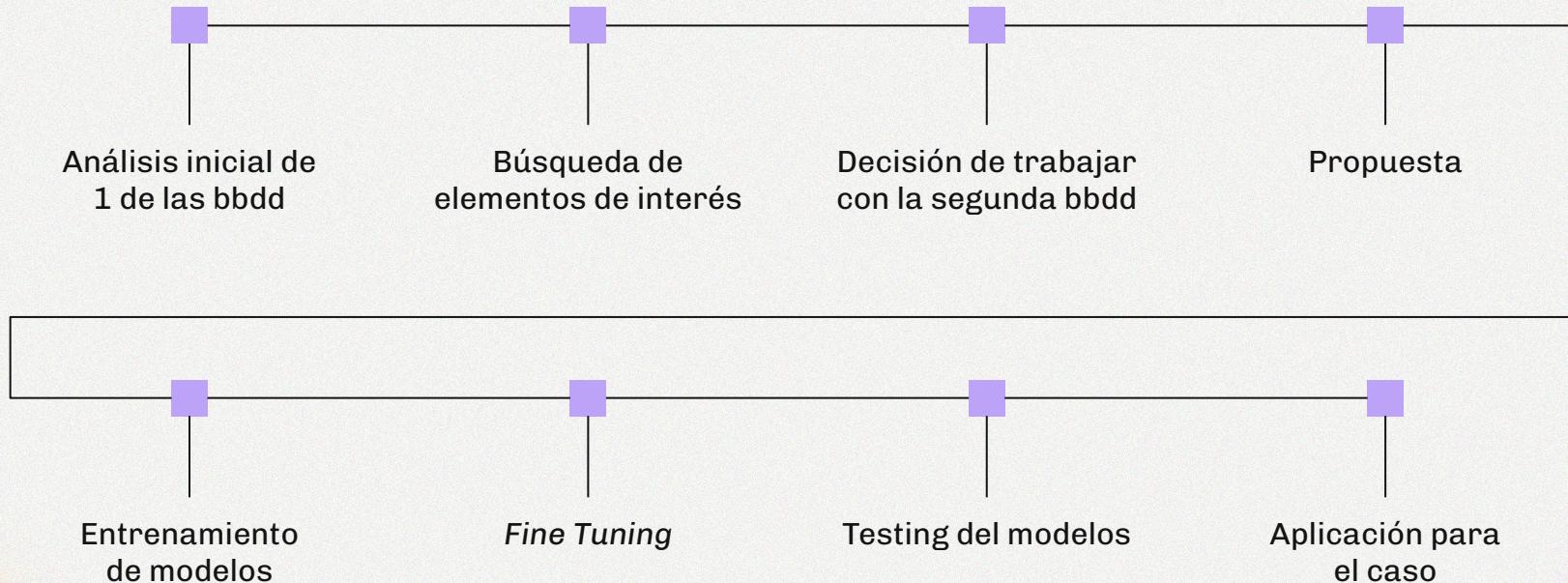
# Predictión vs reales modelo RFR



\*La linea roja es el punto ideal, aca se nota el 69%

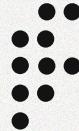


# Proceso de búsqueda

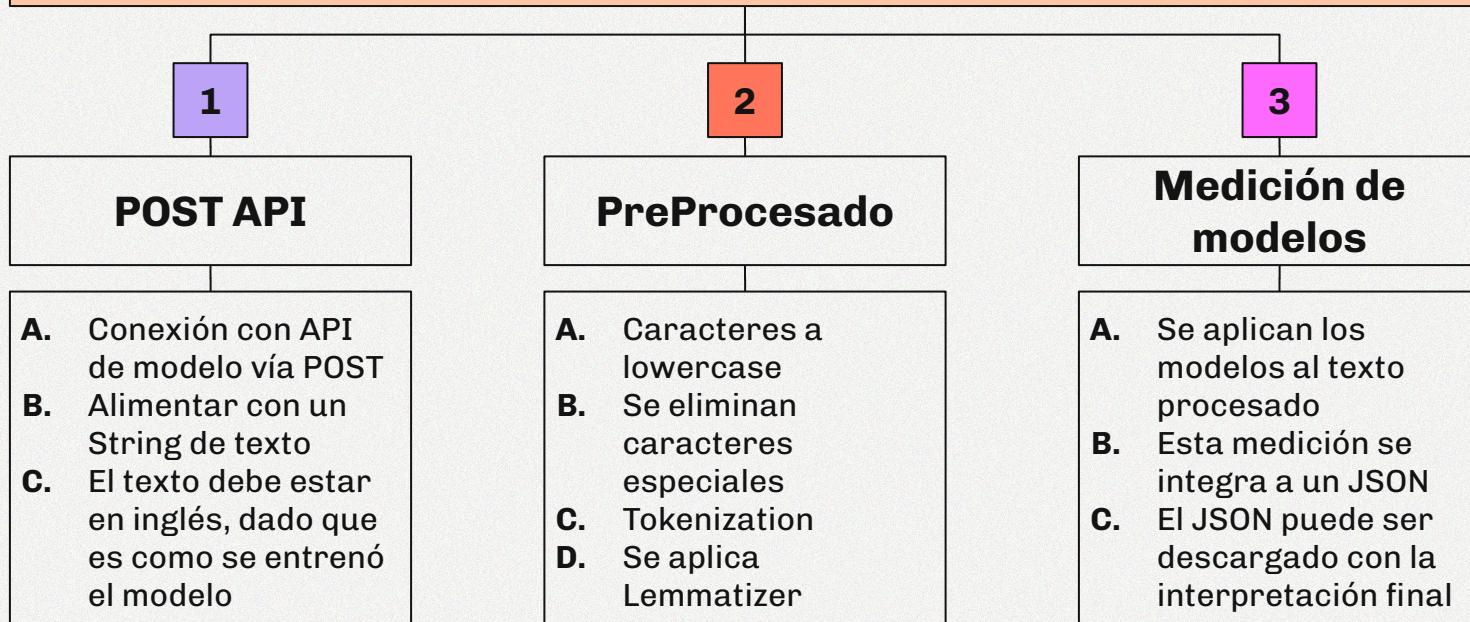


04

# Aplicación



# Lectura de nuevos Comentarios





# Preprocesado

**El resumen de lo que ocurre a lo entregado via POST:**

```
def preprocess_text(text):

    # Step 1: Remove non-alphabetic characters and convert to lowercase
    text = re.sub("[^a-zA-Z]", " ", text.lower())

    # Step 2: Tokenize the text
    tokens = word_tokenize(text)

    # Step 3: Lemmatize tokens
    lemma = WordNetLemmatizer()
    tokens = [lemma.lemmatize(word) for word in tokens]

    # Step 4: Join tokens back into a string
    processed_text = ' '.join(tokens)

return processed_text
```



# Medición

## La aplicación de ambos modelos:

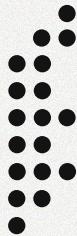
```
def predict_sentiment_and_subjectivity(preprocessed_text):
    # Transform the preprocessed text into a numeric vector
    numeric_vector = count_vectorizer.transform([preprocessed_text])

    # Make prediction using logistic regression model for sentiment analysis
    sentiment_prediction =
        logistic_regression_model.predict(numeric_vector)[0]

    if sentiment_prediction == 0:
        sentiment_label = 'Positive'
    elif sentiment_prediction == 1:
        sentiment_label = 'Negative'
    else:
        sentiment_label = 'Neutral'

    # Make prediction using random forest regression model for subjectivity analysis
    subjectivity_prediction =
        random_forest_model.predict(numeric_vector)[0]

    # Return the predictions
    return sentiment_label,
           subjectivity_prediction
```



# Conclusiones

## Foco inicial

Inicialmente creamos un modelo que nos permite crear predicciones de texto, si es positivo, negativo o neutro y la subjetividad con la que está escrito.

## Modelo

Ambos modelos son complementarios, lo cual nos permite obtener una información valiosa con un solo POST.

## Aplicación

Creemos que esto se puede aplicar como punto intermedio, entre la publicación de comentarios y el traspaso a BBDD.



# MUCHAS GRACIAS

Matías Karmelic

