Proyecto 1- Entrega 2

Inteligencia de negocios

Jonathan Rivera

Julian Castro

Alejandro Gomez

1. Proceso de automatización del proceso de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de API

1.1 Creación de pipelines

-En primer lugar retomamos el modelo de la etapa 1 que obtuvo las mejores métricas, este fue el modelo con vectorización tf-idf, usando el algoritmo de SVM.

-El siguiente paso consistió en automatizar el modelo por medio de creación de pipelines. Para este paso se crearon dos pipelines para la creación de dos endpoints distintos en la API a desarrollar: un pipeline para la predicción de datos y otro para el entrenamiento de un modelo.

```
# Crear pipeline
pipeline = Pipeline([
    ('drop_duplicates', DropDuplicates()),
    ('lang_filter', LanguageFilter()),
    ('preprocessor', TextPreprocessor(stopwords=stopwords.words('spanish'))),
    ('stem_lemmatize', StemLemmatize()),
    ('tfidf', TfidfVectorizer()),
    ('svm', SVC(kernel='linear', random_state=5))
])

joblib.dump(pipeline, 'modelo_entrenamiento.joblib')

# Entrenar el pipeline
pipeline.fit(X2_train, y2_train)

joblib.dump(pipeline, 'modelo_prediccion.joblib')

# Exportar el modelo
predictions = pipeline.predict(X2_test)
```

-Como podemos ver en la imagen, todos los pasos de preparación y transformación de los datos se incluyeron en el pipeline, incluyendo también como pasos finales la vectorización y el algoritmo para la predicción de reviews.

-Para crear dos modelos distintos (predicción y entrenamiento), primero se creo el archivo del pipeline del entrenamiento una vez creado el pipeline con todos sus pasos, posteriormente se entreno el modelo con los datos proporcionados en la etapa 1, a partir de este proceso, procedimos a generar el pipeline para la predicción de datos, que contiene el modelo ya entrenado.

-Los procesos de preparación y transformación de datos en el pipeline son clases personalizadas, ya que la librería de scikit-learn no incluye estas transformaciones.

```python
# Transformador para filtrado de idioma
class LanguageFilter(BaseEstimator, TransformerMixin):
    def __init__(self, language='es'):
        self.language = language

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return X[X.apply(lambda text: detect(text) == self.language)]

# Transformador para eliminación de duplicados
class DropDuplicates(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return X.drop_duplicates()
```

```python
class DropNullValues(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return X.dropna()

# Transformador para preprocesamiento de texto
class TextPreprocessor(BaseEstimator, TransformerMixin):
    def __init__(self, stopwords=None):
        self.stopwords = stopwords if stopwords else set(stopwords.words('spanish'))

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return X.apply(self.preprocess_text)

    def preprocess_text(self, text):
        words = word_tokenize(text)
        words = [word.lower() for word in words]
        words = [re.sub(r'[^\\w\\s]', '', word) for word in words if re.sub(r'[^\\w\\s]', '', word) != '
        words = [unicodedata.normalize('NFKD', word).encode('ascii', 'ignore').decode('utf-8', 'ignore')
        words = [num2words(word, lang='es') if word.isdigit() else word for word in words]
        words = [word for word in words if word not in self.stopwords]
        return ' '.join(words)
```

```python
# Transformador para stemming y lematización
class StemLemmatize(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.stemmer = SnowballStemmer("spanish")

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        print(X)
        print()
        return X.apply(self.stem_and_lemmatize)

    def stem_and_lemmatize(self, text):
        stems = [self.stemmer.stem(word) for word in word_tokenize(text)]
        doc = nlp(" ".join(stems))
        lemmas = [token.lemma_ for token in doc]
        return ' '.join(stems + lemmas)
```

-Como podemos ver el proceso de transformación de datos consiste en los mismos pasos aplicados para la obtención del modelo de la etapa 1: Quedarnos solo con las reseñas en español, quitar los datos duplicados, los datos nulos, remover palabras y caracteres innecesarios (non ascii, stopwords, dígitos, mayusculas), y hacer stemming y lematización.

-Una vez con los modelos exportados, procedimos a crear la API con los endpoints necesarios, para su posterior uso por parte de la interfaz de usuario.

1.2 Diseño de API

-Para la creación de la API usamos el framework de Django con Python. La decisión se tomo basado en las muchas ventajas que ofrece Django: creación y despliegue de APIs de manera sencilla, módulos para el manejo de rutas, manejo de ORM que permite la creación de modelos en una base de datos local fácilmente manipulable por Django.

-Para el diseño de la API pensamos un poco en como seria la aplicación web que montaríamos, llegamos a la conclusión de que un esbozo primero consistiría de autenticación, generación de predicciones, y generación de modelos entrenados.

- A partir de lo anterior definimos los siguientes endpoints:

    -Endpoint de tipo POST que recibe reviews en texto y devuelve las predicciones con un rating asociado.

-Endpoint de tipo POST que recibe reviews en un archivo CSV y devuelve el CSV con una columna adicional de predicciones para las reviews. Este endpoint fue modificado después de la reunión con el equipo de estadística, para que también devolviera métricas y estadísticas de las predicciones generadas.

-Endpoint de tipo POST que recibe un archivo CSV para el entrenamiento de un modelo basado en el pipeline de entrenamiento, estos CSV tienen que incluir una columna con reviews y otra con los ratings de las reviews.

-Endpoint de tipo POST para autenticar a un usuario, recibe el correo y contraseña del usuario y devulve el resultado de la operación de autenticación, exitosa si el usuario existe y su contraseña es correcta, error si fallan los datos recibidos.

-Endpoint de tipo POST para el registro de un usuario, recibe la información de el usuario a registrarse y devuelve una solicitud exitosa en caso de que el correo no este registrado.

-Endpoint de tipo GET para obtener el histórico de las reviews predecidas por un usuario. Este endpoint se definió después de la reunión de validación con el grupo de estadística.

```python
@csrf_exempt
def prediccionCsv(request):
    return pcsv(request)
@csrf_exempt
def prediccionTxt(request):
    return ptxt(request)
@csrf_exempt
def entrenamiento(request):
    return cm(request)
@csrf_exempt
def login(request):
    return lg(json.loads(request.body))
@csrf_exempt
def register(request):
    return rg(json.loads(request.body))
@csrf_exempt
def lastReviews(request):
    return lr(json.loads(request.body))
```

Endpoints definidos en Django

-Con los endpoints definidos, creamos la lógica para cada una de las operaciones contempladas con estos.

1.3 Creación de API

-Para el endpoint de predicción de reviews textuales se definió la siguiente lógica:

```python
def prediccionTexto(req):
        data = req.body.decode('utf-8')
        data_json = json.loads(data)
        correo = data_json['correo']
        usuario = getUsuario(correo)
        directorio_actual = os.path.dirname(__file__)
        ruta = os.path.join(directorio_actual,'modelo_prediccion.joblib')
        pipeline = load(ruta)
        # Importante: json.loads(data) si el contenido es JSON
        reviews = data_json['reviews']
        reviews_list = reviews.split("|$|")
        df = pd.DataFrame(reviews_list, columns=['Reviews'])
        df['Rating']= pipeline.predict(df['Reviews'])
        reviews_ratings_dict = df.to_dict(orient='records')
        print(reviews_ratings_dict)
        reviews_usuario = json.loads(usuario.reviews)
        print(reviews_usuario)
        for review in reviews_ratings_dict:
            reviews_usuario[review['Reviews']]=  review['Rating']
        usuario.reviews = json.dumps(reviews_usuario)
        usuario.save()
        # Devolver el diccionario como una respuesta JSON
        return JsonResponse({'status': 'success', 'data': reviews_ratings_dict})
```

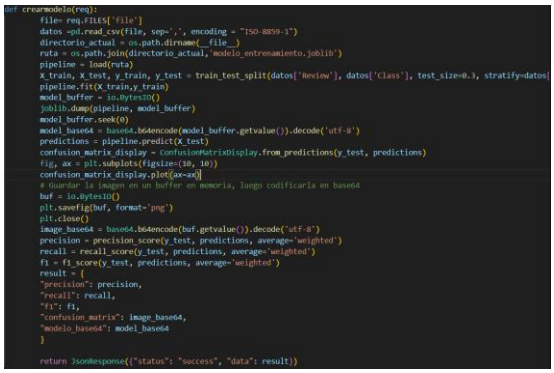
-Para esta función primero obtenemos el usuario que esta realizando la predicción, con su correo que se envía en la petición, esto con el motivo de poder almacenar las predicciones hechas por el usuario. Posteriormente carga el pipeline del modelo de predicciones, que se encuentra importado en la API. Se parsean las predicciones enviadas por el usuario en la solicitud, y se crea un dataframe a partir de estas. Finalmente se predicen con el modelo, que nos arrojara en una nueva columna del dataframe las predicciones. Guardamos las predicciones con sus ratings en el modelo de Django del usuario, y posteriormente procedemos a enviarlas como respuesta de la solicitud.

-Para el endpoint de predicción por medio de archivos csv se definió la siguiente lógica:

```python
def prediccionCsv(req):
    file = req.FILES['file']
    directorio_actual = os.path.dirname(__file__)
    ruta = os.path.join(directorio_actual, 'modelo_prediccion.joblib')
    pipeline = load(ruta)
    # Leer el archivo CSV
    data = pd.read_csv(file, sep=',', encoding="ISO-8859-1")
    # Realizar predicciones
    data['Rating'] = pipeline.predict(data['Review'])
    # Calcular estadísticas
    stats = {
        'maximo': int(data['Rating'].max()),
        'minimo': int(data['Rating'].min()),
        'promedio': float(data['Rating'].mean()),
        'desviacion_estandar': float(data['Rating'].std()),
        'cantidad_de_registros': int(len(data))
    }
    # Crear gráfico de la distribución de los ratings
    plt.figure(figsize=(8, 6))
    plt.hist(data['Rating'], bins=10, color='skyblue', edgecolor='black')
    plt.title('Distribución de Ratings')
    plt.xlabel('Rating')
    plt.ylabel('Frecuencia')
    plt.grid(True)
    plt.tight_layout()
    # Guardar el gráfico en un buffer de Bytes
    buffer = io.BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    # Codificar el gráfico en base64
    graph_base64 = base64.b64encode(buffer.getvalue()).decode('utf-8')
    # Adjuntar los datos a un diccionario
    response_data = {
        'stats': stats,
        'graph': graph_base64,
        'csv_data': data.to_dict(orient='records')  # Convertir DataFrame a diccionario
    }
```

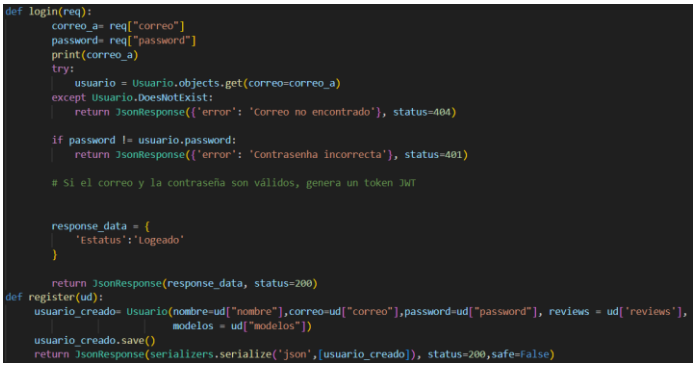
-Aquí extraemos el csv del request, lo convertimos en dataframe de pandas y generamos las predicciones de las reviews en una nueva columna. Después de esto sacamos las estadísticas principales de las predicciones, y un grafico con la distribución de los ratings predecidos. Todo lo anterior junto con el archivo csv actualizado, se va a enviar como respuesta del request.

-Para el endpoint de entrenamiento de modelos se definió la siguiente lógica:



-Aquí tomamos el csv proporcionado por el usuario, lo cargamos como dataframe, hacemos la división en datos de entrenamiento y prueba (para poder sacar las métricas de desempeño), y procedemos a entenar el modelo con los datos de entrenamiento. Una vez entrenado el modelo predecimos para los datos de prueba y comparamos predicciones con datos reales para obtener las métricas de desempeño. Finalmente devolvemos las métricas de desempeño (incluyendo la matriz de confusión) junto con el modelo en formato joblib.

-Para los endpoints de login y registro definimos la siguiente lógica:



-Cuando un usuario manda solicitud de registros, se crea un nuevo objeto del modelo de Django Usuario, que permite la persistencia de los usuarios del sistema. Para el registo se saca el correo y la conteaseña del request, se verifica que el usuario exista y que esa sea su contraseña, y se devuelve una solicitud exitosa dado el caso.

-Finalmente para el endpoint del histórico de reviews, se obtiene el objeto de la persistencia del Usuario que mando la solicitud, y se saca el atributo de las ultimas reviews predecidas para el usuario.

2. Desarrollo de la aplicación y justificación

2.1 Apoyo con el equipo de estadistica

-Para definir el usuario respecto al cual se iba a desarrollar la aplicación nos apoyamos con el equipo de estadística, conformado por Sofia Vargas y Lorena Figueroa, aquí un registro de las reuniones y lo hablado con el grupo:

Reunión de resultados etapa 1 e inicio etapa 2, Fecha: 13 de abril

-El grupo de estadística expreso satisfacción con la presentación de resultados y conclusiones de la etapa 1, remarcaron en mejorar la comunicación de los resultados de una manera mas intuitiva por medio de estadísticas. Se charlo sobre la creación de la API, propusimos los endpoints para las API de generar predicciones a partir de texto y CSV, y el endpoint de creación de modelos. El grupo de estadística propuso generar endpoints adicionales relacionados a la obtención de datos históricos de predicciones generadas por los usuarios, y consecuente con esto endpoints para la creación y autenticación de usuarios. Llegamos a la conclusión de genera los endpoints propuestos por nuestro equipo y los propuestos por el equipo de estadística. Finalmente propusimos una reunión para el lunes 15 de abril, para definir el usuario de la aplicación y el desarrollo de esta.

Reunión de definición usuario y aplicación, Fecha: 15 de abril

-Junto con el grupo de estadística definimos el usuario de la aplicación. Ambos estábamos de acuerdo que considerando las funcionalidades del modelo, el mejor usuario posible para generar una aplicación a partir de este, serian los administradores de los hoteles, u otras figuras que se encarguen de analizar la satisfacción de los clientes de los hoteles para la implementación de estrategias que ayuden con los objetivos de negocio. El grupo nos sugirió que el uso de la aplicación debía ser simple, con solo un par de botones para cada funcionalidad, y el despliegue de los resultados de cada funcionalidad de manera minimalista y simplificada para su entendimiento. La pagina también debía manejar el registro y autenticación de los usuarios para guardar sus predicciones. Cada funcionalidad de la aplicación debería estar separada de las otras para fuera mas facil para el usuario manejarla. Finalmente se cuadro una reunión para el viernes 19 de abril para validar la implementación de la aplicación web.

Reunión de validación de la APP, Fecha: 19 de abril

-El equipo quedo satisfecho con la aplicación presentada, elogiaron la facil interacción con esta, el diseño minimalista y ordenado, y la creatividad para el despliegue de resultados, no sugirieron cambio alguno.

2.2 Definición de usuario de la aplicación

-De acuerdo a lo hablado con el grupo de estadística el usuario definido para la pagina fue administradores de hoteles u otros miembros de las empresas que se encarguen de el análisis de satisfacción del usuario y la implementación de estrategias para el cumplimiento de los objetivos de negocio basado en el análisis hecho. La aplicación desarrollada proporciona un conjunto de funcionalidades clave que contribuyen directamente a los objetivos de negocio establecidos:

En primer lugar, la capacidad de predecir ratings de reviews textuales permite a los usuarios identificar áreas de mejora al determinar los factores que caracterizan a los lugares con calificaciones bajas. Esto proporciona una comprensión cualitativa de las descripciones asociadas con los peores lugares, lo que ayuda a los administradores a focalizar sus esfuerzos de mejora en áreas específicas.

Además, la función de predecir ratings de reviews en un CSV amplía esta capacidad al permitir el análisis de grandes conjuntos de datos de reviews. Esto es crucial para identificar tendencias a lo largo del tiempo y en diferentes contextos, lo que puede proporcionar una visión más completa de las áreas de mejora y de las características de atracción.

La capacidad de arrojar métricas de las predicciones, como la precisión, el recall y el F1-score, ayuda a los usuarios a comprender la calidad y el rendimiento de los modelos de predicción. Esto es fundamental para evaluar la efectividad de las estrategias implementadas y ajustarlas según sea necesario.

Además, la funcionalidad de crear un modelo para la predicción de ratings con datos proporcionados para el entrenamiento permite a los usuarios desarrollar modelos predictivos personalizados que se ajusten a las necesidades específicas de sus negocios. Esto les brinda la capacidad de identificar características únicas que pueden ser clave para la satisfacción del cliente y la mejora continua.

En última instancia, la aplicación está diseñada para empoderar a los administradores de hoteles y otros profesionales responsables del análisis de satisfacción del usuario y la implementación de estrategias. Les proporciona herramientas para identificar áreas de mejora, entender las características que más atraen a los clientes y anticipar la recepción de futuros visitantes. Con esta información, los usuarios pueden formular estrategias proactivas de mejora, implementar planes de acción basados en análisis de datos y promover el turismo en Colombia mediante la diversificación y el crecimiento de la oferta turística en el país.

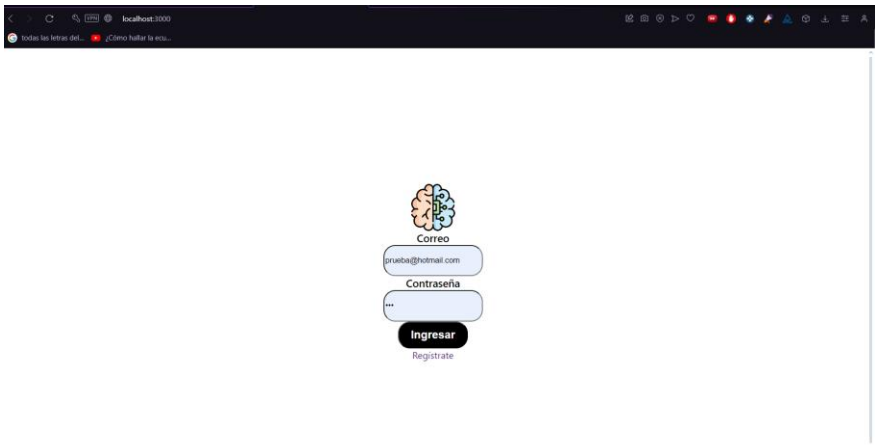
2.3 Desarrollo de la aplicación

-Para el desarrollo de la interfaz web que se comunica con la API que creamos previamente, se optó por utilizar el framework de React junto con JavaScript debido a varias ventajas clave. React ofrece una estructura robusta que facilita la creación de interfaces de usuario interactivas y dinámicas. Al estar diseñado para ser eficiente y flexible, permite a los desarrolladores construir páginas complejas con un rendimiento optimizado. Además, JavaScript es ampliamente conocido por su escalabilidad y compatibilidad con diversos navegadores, lo que resulta ideal para proyectos que requieren una alta interactividad y una buena experiencia de usuario en una variedad de dispositivos y entornos. Esta combinación también se integra de manera eficiente con otras herramientas y bibliotecas, haciendo del desarrollo un proceso más ágil y efectivo.

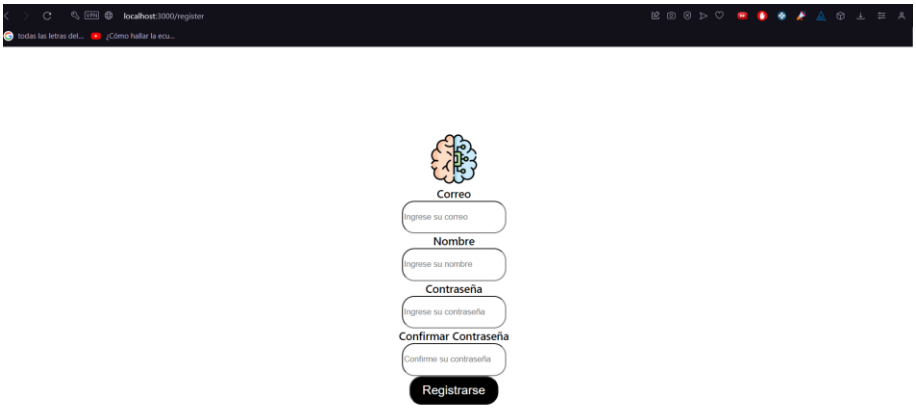
3. Resultados

-La pagina se planteo de la siguiente manera:

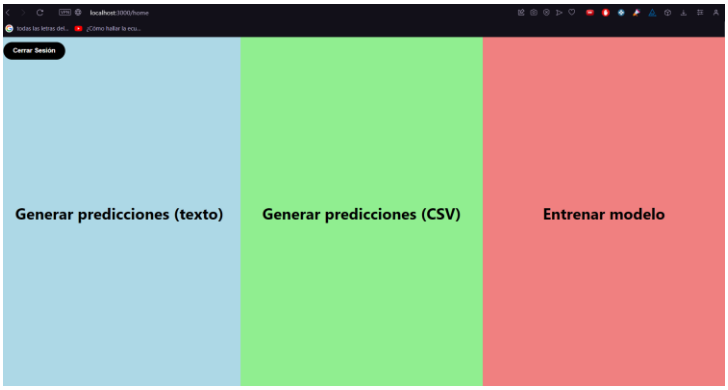
Primero el usuario se encuentra con la pantalla de logueo:



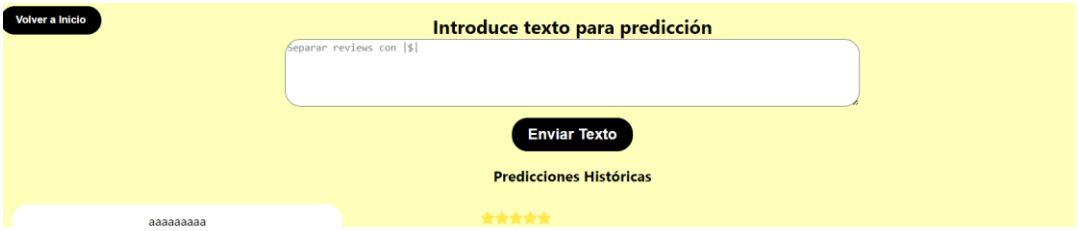
-Si no posee cuenta puede darle al botón de registrarse y lo llevara al formulario de registro:

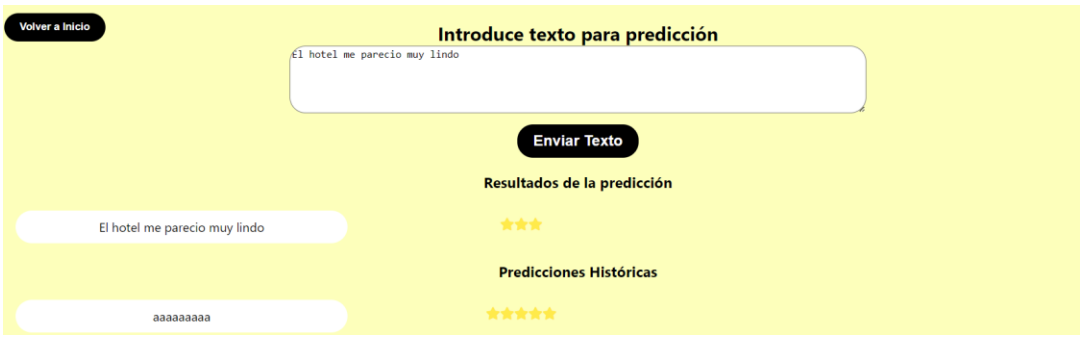


-Una vez registrado puede iniciar sesión que lo llevara a la pantalla de inicio donde puede escoger que funcionalidad utilizar de la pagina, esta pantalla también posee un botón de cerrar sesión :



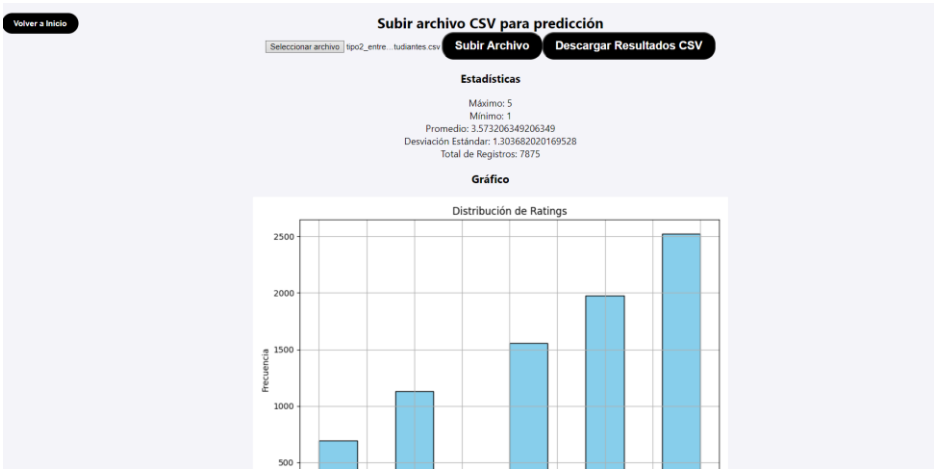
-En la pantalla de Generar predicciones texto nos encontramos con una opción para ingresar reviews en un input, y un histórico de las predicciones hechas en caso de que el usuario ya haya usado esta funcionalidad:





-En la pantalla de Generar predicciones csv aparece una opción para subir el archivo csv y generar un archivo nuevo con las predicciones:

-Una vez se generen las predicciones, se actulizara la pantalla con los resultados y la opción para descargar el archivo con predicciones:



-Finalmente en la pantalla de entrenamiento encontraremos esto:

-Despues de cargar el archivo obtendremos los resultados:

4. Trabajo en equipo

Rol dentro del equipo	**Integrantes**
Líder del Proyecto	Julián Castro Jonathan Rivera Alejandro Gómez Sofia Vargas Lorena Figueroa
Ingeniero de datos	Alejandro Gomez
Ingeniero de software responsable del diseño de la aplicación y resultados	Jonathan Rivera Julian Castro
Ingeniero de software responsable de desarrollar la aplicación final	Jonathan Rivera Julian Castro
Integrante	Tareas
Jonathan Rivera	Diseño de api, implementación de endpoints, implementación de front
Julián Castro	Diseño de api, implementación de lógica de endpoints, Diseño de front
Alejandro Gómez	Creación de pipeline y transformaciones personalizadas, Diseño de api, mejoras estéticas front

Integrante	**Horas**	**Distribución del trabajo**
Jonathan Rivera	26	33%
Julián Castro	26	33%

Alejandro Gómez	26	33%

-El trabajo fue altamente equitativo e integrado, con colaboración mutua para todas las tareas a desarrollar a pesar de la división y repartición de esta.

-Los mayores retos estuvieron relacionados a dos categorías: la primera fue la integración técnica entre sistemas distintos, tuvimos que resolver varios problemas de compatibilidad de datos entre el pipeline, la API y el front. Y en segundo lugar, intentar crear un sistema que fuera lo suficientemente intuitivo y simplista para que una persona sin conocimiento técnico de modelos pudiera utilizar la aplicación. Con el primer problema aprendimos que es importante leer bien la documentación de las tecnologías a utilizar y tener claro su funcionamiento, con el segundo aprendimos que cuando realmente se entiende un concepto, es fácil traerlo a términos poco técnicos.