Resumen del proyecto de análisis de compra de casas en Barcelona

Introducción: En este proyecto, se llevó a cabo un análisis de ventas de casas en Barcelona utilizando datos obtenidos de la página web Idealista a través de la plataforma Kaggle. Los datos recopilados incluyeron información sobre el precio, área en metros cuadrados, presencia de ascensor, distrito, vecindario, cantidad de habitaciones y la distancia al metro en euros por metro. El objetivo principal del proyecto fue entrenar un modelo capaz de predecir si una propiedad fuera una buena opción de compra o no, basándose en estas variables, En este proyecto de análisis de ventas de casas en Barcelona, se utilizó una base de datos obtenida de la página web Idealista a través de la plataforma Kaggle. Estos datos representan una recopilación de información sobre propiedades en venta, incluyendo detalles como el precio, el área en metros cuadrados, la presencia de ascensor, el distrito, el vecindario, la cantidad de habitaciones y el valor del metro en euros. El objetivo principal del proyecto fue desarrollar un modelo capaz de predecir si una propiedad fuera una buena opción de compra o no, considerando estas variables.

Metodología:

Preparación de los datos:

- Se importaron las librerías necesarias, como numpy, pandas, matplotlib, seaborn y plotly, para el análisis y visualización de datos.
- Se utilizó la función "read_csv" de pandas para cargar los datos del archivo "datos_precios_casas.csv".
- Se seleccionaron las columnas relevantes para el análisis y la construcción del modelo.
- Se dividió el conjunto de datos en características (X) y variable objetivo (y) utilizando la función "train_test_split" de sklearn.

A continuación, se utilizó la función "read_csv" de pandas para cargar los datos desde el archivo "datos_precios_casas.csv". Una vez cargados los datos, se seleccionaron las columnas relevantes para el análisis y construcción del modelo, que incluyeron las variables de interés: precio, área en metros cuadrados, presencia de ascensor, distrito, vecindario y cantidad de habitaciones. Estas columnas se asignaron al DataFrame "data_f".

Luego, se dividió el conjunto de datos en características (X) y variable objetivo (y) utilizando la función "train_test_split" de sklearn. Esta división es esencial para el entrenamiento y evaluación del modelo posteriormente. El conjunto de entrenamiento representó el 70% de los datos, mientras que el conjunto de prueba abarcó el 30% restante. Además, se estratificó la división según la variable objetivo "compra" para asegurar una distribución equilibrada en ambos conjuntos.

Preprocesamiento de los datos:

Se aplicó una escala de características utilizando el escalador MinMaxScaler de sklearn para estandarizar los valores de las características.

Se estandarizó tanto el conjunto de entrenamiento como el conjunto de prueba.

Se creó un nuevo DataFrame con los datos proporcionados por el usuario para realizar predicciones.

En esta etapa, se aplicó una técnica de preprocesamiento llamada escala de características o normalización a los datos. Para ello, se utilizó la clase "MinMaxScaler" de sklearn. El escalado de características se realizó para estandarizar los valores de las variables y evitar sesgos o diferencias de escala que puedan afectar al rendimiento del modelo.

Primero, se instanció el objeto "scaler" de la clase MinMaxScaler. Luego, se aplicó el escalado a los datos del conjunto de entrenamiento utilizando el método "fit_transform" del escalador. Esto ajustó el escalador a los datos de entrenamiento y, al mismo tiempo, transformó las características a una escala específica.

Posteriormente, se recreó el conjunto de entrenamiento con las columnas estandarizadas, reemplazando el DataFrame original. Esto aseguró que las características del conjunto de entrenamiento estuvieran escaladas correctamente.

El mismo proceso de escala se aplicó al conjunto de prueba, utilizando el método "transform" del escalador. Nuevamente, se recreó el DataFrame del conjunto de prueba con las columnas estandarizadas.

Entrenamiento del modelo:

Se instanció y entrenó un modelo de regresión logística utilizando la clase LogisticRegression de sklearn.

Se utilizó el conjunto de entrenamiento escalado para ajustar el modelo.

Se realizaron predicciones utilizando el conjunto de prueba.

En esta fase, se instanció y entrenó un modelo de regresión logística utilizando la clase "LogisticRegression" de sklearn. La regresión logística es un algoritmo de aprendizaje supervisado utilizado para la clasificación de datos.

Se instanció un objeto "lr" de la clase LogisticRegression, y se configuraron los parámetros, como el número máximo de iteraciones y la semilla aleatoria (random_state). Luego, se utilizó el método "fit" para entrenar el modelo con los datos del conjunto de entrenamiento escalados (X_train).

Evaluación del modelo:

Se utilizó la matriz de confusión y el informe de clasificación para evaluar el rendimiento del modelo.

Se asignaron etiquetas descriptivas ("COMPRA RECOMENDABLE" y "COMPRA NO RECOMENDABLE") a las predicciones.

Una vez entrenado el modelo, se realizaron predicciones sobre el conjunto de prueba utilizando el método "predict" del modelo. Las predicciones resultantes se almacenaron en la variable "y_pred_lr".

Para evaluar el rendimiento del modelo, se utilizaron métricas como la matriz de confusión y el informe de clasificación. La matriz de confusión proporciona una visión general de las predicciones correctas e incorrectas, mientras que el informe de clasificación ofrece métricas más detalladas, como precisión, recall y puntuación F1 para cada clase.

Guardado del modelo: Una vez completado el entrenamiento y la evaluación del modelo, se guardó el modelo entrenado utilizando la función "dump" de la biblioteca joblib. El modelo se guardó en un archivo llamado "modelo_entrenado.joblib". Esta acción permite conservar el modelo para su uso futuro sin necesidad de volver a entrenarlo.

Uso del modelo entrenado:

- Se cargó el modelo entrenado utilizando la función "load" de joblib.
- Se proporcionaron nuevos datos para hacer predicciones sobre la opción de compra de una casa.
- Se realizó la predicción utilizando el modelo cargado y los nuevos datos

Para utilizar el modelo entrenado, se cargó el archivo "modelo_entrenado.joblib" utilizando la función "load" de joblib. Esto permitió cargar el modelo en la memoria para realizar predicciones posteriores.

Se proporcionaron nuevos datos, ingresados por el usuario, para realizar predicciones sobre la opción de compra de una casa. Los datos se organizaron en un nuevo DataFrame llamado "new_df" y se escaló utilizando el mismo escalador MinMaxScaler.

A continuación, se utilizó el modelo cargado para realizar predicciones sobre el nuevo DataFrame "new_df" utilizando el método "predict". Las predicciones se asignaron a la variable "y_new".

Implementación y versionado:

El proyecto se dockerizó para facilitar su implementación y portabilidad.

Se utilizó Google Cloud para alojar la implementación dockerizada del proyecto.

GitHub se utilizó para el versionado y control de cambios durante el desarrollo del proyecto.

La implementación en la nube se actualiza automáticamente cuando se realizan cambios en el repositorio de GitHub.

El proyecto se dockerizó para facilitar su implementación y portabilidad. Docker permite crear contenedores ligeros y autónomos que incluyen todas las dependencias necesarias para ejecutar la aplicación sin problemas en diferentes entornos.

Para asegurar un control de versiones y colaboración eficiente, se utilizó GitHub. GitHub permite almacenar y gestionar el código fuente del proyecto, realizar un seguimiento de los cambios y facilitar la colaboración entre diferentes miembros del equipo.

La implementación del proyecto se realizó en Google Cloud, una plataforma en la nube que ofrece una variedad de servicios para alojar, gestionar y desplegar aplicaciones. La implementación en la nube se actualizó automáticamente cuando se realizaron cambios en el repositorio de GitHub, lo que facilitó el despliegue y actualización del proyecto de manera eficiente.

Conclusiones:

En este proyecto de análisis de ventas de casas en Barcelona, se utilizó un conjunto de datos obtenido de la página web Idealista a través de Kaggle. El objetivo fue desarrollar un modelo de regresión logística capaz de predecir si una propiedad era una buena opción de compra o no,

utilizando variables como el precio, el área en metros cuadrados, la presencia de ascensor, el distrito, el vecindario, la cantidad de habitaciones y el valor del metro en euros.

Se realizaron diversas etapas de procesamiento y análisis de datos, incluyendo la preparación de los datos, el preprocesamiento mediante la escala de características, el entrenamiento del modelo y la evaluación de su rendimiento. Además, se implementó un proceso de guardado y carga del modelo entrenado, lo que permitió utilizarlo posteriormente para realizar predicciones sobre nuevos datos.

El proyecto fue dockerizado para facilitar su implementación y se utilizó GitHub para el control de versiones y colaboración en el desarrollo del proyecto. La implementación final se realizó en Google Cloud, aprovechando sus servicios en la nube para alojar y desplegar la aplicación.

En resumen, este proyecto demuestra cómo se puede utilizar un modelo de regresión logística para predecir la opción de compra de casas en Barcelona, considerando diversas variables relevantes. La implementación en la nube y el uso de herramientas como Docker y GitHub facilitaron el desarrollo y despliegue del proyecto.

Bibliografía:

Datos obtenidos de Kaggle: https://www.kaggle.com/datasets/jorgeglez/barcelona-idealista-housingprices

Página web de Idealista: https://www.idealista.com/venta-viviendas/barcelona-provincia/mapa

Documentación de las bibliotecas utilizadas:

Pandas: https://pandas.pydata.org/

Scikit-learn: https://aprendeia.com/libreria-scikit-learn-de-python/

Matplotlib: https://aprendeconalf.es/docencia/python/manual/matplotlib/

Seaborn: https://www.analyticslane.com/2018/07/20/visualizacion-de-datos-con-seaborn/

Documentación de Docker: enlace a la documentación de Docker

Documentación de GitHub: Torfer26/hkgn (github.com)

Documentación de Google Cloud: https://hkgn-erkt535vha-uc.a.run.app