

Documentation.

Introduction:

This file represents the process by which the programming was conducted to develop a model capable of predicting each store and its departments weekly sales with acceptable accuracy. the model was developed using the python programming language. A popular and effective tool in the field of artificial intelligence (AI).

A primary set of features was developed using data analysis tools. However, while developing the model this set proved to lack the ability to provide the necessary accuracy leading to a trial-and-error process which was guided by the weights provided by the eli5 function with the XGBRegressor, during that process new features were added and some were removed to reach the highest possible accuracy, which in its turn led to a significant increase of accuracy which made the model satisfactory. Figure 1 shows the code used to generate the weight of each feature which was used to guide the trial and error process.

```
#XGB regressor
from xgboost import XGBRegressor
gbm = XGBRegressor(random_state=42, n_jobs=-1, n_estimators=400, max_depth=15, learning_rate=0.35)
gbm.fit(X_train,y_train)
train.head(10)
```

```
#permutation importance
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(gbm, random_state=1).fit(X_train, y_train)
features = eli5.show_weights(perm, top=len(X_train.columns), feature_names = X_test.columns.tolist())

features_weights = eli5.show_weights(perm, top=len(X_train.columns), feature_names = X_test.columns.tolist())
features_weights
f_importances = pd.Series(dict(zip(X_test.columns.tolist(), perm.feature_importances_))).sort_values(ascending=False)
f_importances
```

Figure 1: Code of eli5 and XGBRegressor used to generate the weight of each feature.

Code segments:

The code was divided into two parts the first represents the code needed to predict the weekly sales for each department, meanwhile the second represents the predictive model for the weekly sales for each store.

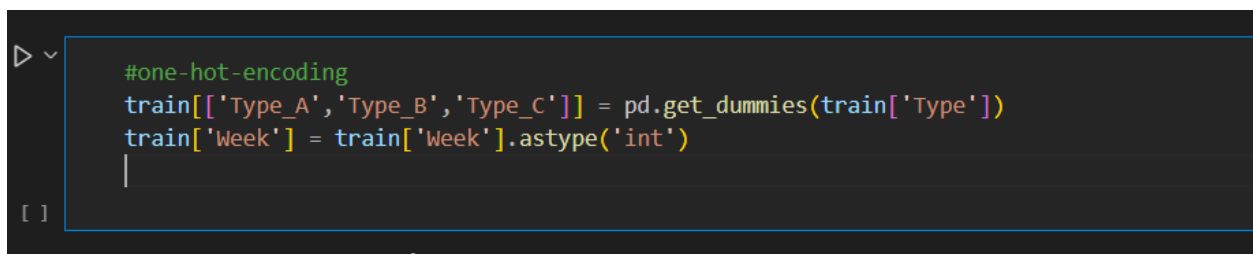
Each section consisted of different segments to enhance readability, and reusability, the segments in both sections can be divided as follow:

Pre-processing:

First part.

In this segment the goal was to develop a data frame that is suitable to use inside a regression model, it also included adding necessary information from different files, for the first section the prepared data had a much larger number of rows since it detailed the weekly sales for each department making the number of rows around 400k.

Two categorical data were transformed to fit the regression model in the first section, the first was the store type which was transformed using `get.dummies()` function since it only had 3 types as shown in the following figure.



```
#one-hot-encoding
train[['Type_A', 'Type_B', 'Type_C']] = pd.get_dummies(train['Type'])
train['Week'] = train['Week'].astype('int')
|
```

Figure 2: one hot encoding using `get.dummies()` for the store type.

Unlike the store type the department hot-encoding include had 99 item, the department ID was transformed using the category encoders model to the department the code used is shown in Figure 3.

```
def enc(x):
    df1 = pd.DataFrame(x, columns=['Store', 'Dept', 'IsHoliday', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment', 'Type', 'Size', 'week'])
    encoded_dept = encoder.transform(df1['Dept'])
    df1 = df1.join(encoded_dept)
    df1 = pd.get_dummies(df1)
    df1 = df1.reindex(columns = X.columns, fill_value=0)
    df1 = df1[['Dept_0', 'Dept_1', 'Dept_2', 'Dept_3', 'Dept_4', 'Dept_5', 'Dept_6', 'Store', 'Size', 'week',
               'IsHoliday', 'Type_A', 'Type_B', 'Type_C', 'Temperature', 'Fuel_Price', 'Unemployment', 'CPI']]
    return df1
```

Figure 3: Code used to transform the categorical date into binary data.

A date transformer function was developed to automate the conversion from regular date to its component which were (day, week, month, and year) As shown in Figure 4.

```
def date_transformer(data_frame, date_column):
    data_frame[date_column]=pd.to_datetime(train['Date'])
    data_frame['Day'] = data_frame['Date'].dt.day
    data_frame['Week'] = data_frame['Date'].dt.isocalendar().week
    data_frame['Month'] = data_frame['Date'].dt.month
    data_frame['Year'] = data_frame['Date'].dt.year
    return data_frame
date_transformer(train,"Date")
```

Figure 4: Date transformer function

Second part.

In the second part the preprocessing included taking the summation of the weekly sales of each department within a store to calculate the weekly sales for each store. A function was developed to calculate the summation of a set of columns and add it as a new column in the data frame as shown in Figure 5.

```
#function for combining the sum of a set of features
def join_sum(data_frame,list_of_columns):
    df_md = data_frame[list_of_columns]
    Markdown_sum = df_md.sum(axis=1)
    data_frame["Markdown_sum"] = Markdown_sum
```

Figure 5: a function to add the summation of a set of columns to the data frame.

The summation of the weekly sales means the elimination of the department feature which makes the transformation only include the store type using `get.dummies()` function.

The preprocessing also included the code used to calculate the weight of features which helped in the manual feature selection, as was discussed in the introduction.

Optimization:

The optimization was conducted using different estimators within each estimator a series of parameters was optimized, and the optimization process for the first part was conducted as follows:

- Two estimators were used the first is `RandomForestRegressor`, and `DecisionTreeRegressor`.
- The cross-validation function used was `RandomizedSearchCV` since it was faster than `GridSearchCV` and since the latter is impossible to use with the large dataset which we have in part 1.
- `RandomForestRegressor` proved to be a much better suit for our data giving a significantly higher score (coefficient of determination) compared to `DecisionTreeRegressor`.
- The parameters used for the optimization of the `RandomForestRegressor` were (`max_depth`, `min_samples_split`, and `min_samples_leaf`)

- Best parameters were ('min_samples_split': 3, 'min_samples_leaf': 7, 'max_depth': 14)

As for the second part the optimization was conducted using three different estimators, the addition was support vector regressor which did not provide a satisfactory result and the final results showed that RandomForestRegressor provided the best score. The results were reached after applying the following steps.

- three estimators were used the first is RandomForestRegressor, SVR, and DecisionTreeRegressor.
- The cross-validation function used was GridSearchCV since the data was much smaller compared to part 1.
- RandomForestRegressor proved to be a much better suit for our data giving a significantly higher score (coefficient of determination) compared to DecisionTreeRegressor which was also significantly better than SVR.
- The parameters used for the optimization of the RandomForestRegressor were (max_depth, criterion, and n_estimators)
- Best parameters were ('criterion': 'friedman_mse', 'max_depth': 43, 'n_estimators': 75)

The final model:

The best parameters for each part were used to develop the final model using the RandomForestRegressor for both parts, for the second part the grid function was used to perform cross-validation 5 times.

After training both models on the training data sets the score (coefficient of determination) was calculated to make sure the model achieved the desired accuracy, following that the models were transformed into pickle files so it can be used in the deployment.

Model Deployment:

The models for both parts were saved using the pickle library, and in the deployment code they were unpacked to be used for predictions also the deployment model was connected to GUI to make it easier for the user to apply a prediction, nonetheless, if the user wants to predict a set of weeks, the user has to prepare a dataset with the needed parameters.

The model deployment contained a function that worked to ensure that the number of features in the input is the same as required for the model to predict, those functions worked to transform the input categorical data to valid numerical data for the model. The following figure shows the function with a default parameter x.

```
10 #This function will predict the weekly sales for a department in a store.
11 def predict(x= [[1,1,False,42.31,2.572,211.096358,8.106,'A',151315,5]]):
12     df1 = pd.DataFrame(x,columns=['Store','Dept','IsHoliday','Temperature','Fuel_Price','CPI','Unemployment','Type','Size','Week'])
13     encoder=pickle.load(open('enc.pkl','rb'))
14     encoded_dept = encoder.transform(df1['Dept'])
15     df1 = df1.join(encoded_dept)
16     df1 = pd.get_dummies(df1)
17     model = pickle.load(open('model.pkl','rb'))
18     columns=list(model.feature_names_in_)
19     df1 = df1.reindex(columns = columns, fill_value=0)
20     df1 = df1[columns]
21
22     return model.predict(df1)
23
```

Figure 6: function to standardized input so it works with the model by changing categorical data to numerical

The following figure show the code used to develop the GUI using the streamlit app which is a framework application in python that is used to develop user-friendly interface, it takes input from

the user and if the department is put to zero it redirects the function to store prediction instead of department's prediction.

```
37 #This function takes input from the user and it will be redirected to either models depending on the inputs.
38 def main():
39     st.title('Store Sales prediction')
40     st.write('If you want to predict the store sales please keep the next two cells as zero.')
41     store=int(st.number_input("What is your store number(0-45)?"))
42     dept=st.number_input("What is your Department number(0-99)?")
43     isholiday=st.selectbox("Is there a holiday?",('True','False'))
44     temp=st.number_input("What is the temperature? ")
45     fuel=st.number_input("What is the fuel price?")
46     cpi=st.number_input("What is the CPI ?")
47     unem=st.number_input("What is the Unemployment?")
48     type=st.selectbox("What is the Type of your store?",('A','B','C'))
49     size=st.number_input("What is the size?")
50     week=st.number_input("What is the week (1-52)?")
51     weekly_sales=''
52     if st.button('weekly_sales'):
53         if store and dept==0:
54             weekly_sales=predict2(x=[[isholiday,temp,fuel,cpi,unem,type,size,week]])
55         else:
56             weekly_sales=predict(x=[[store,dept,isholiday,temp,fuel,cpi,unem,type,size,week]])
57     st.success(weekly_sales)
58
59
60
61 if __name__ == '__main__':
62     main()
```

Figure 7: the code used to develop the GUI

Graphical User Interface (GUI):

The GUI is made to be easy and friendly, in which entering the data is divided into boxes, in each box a point of the required data for the model is entered, the model in the backend calculates the prediction for one point at a time, the GUI also gives enough information so it eases the process of making a new prediction whether it is for a store or a department.

Stream lit :

To display the GUI run the python file from the terminal to display your local host webpage by giving it the path of your file as shown in figure 8. And it will automatically redirect you to the web page as shown in figure 9.

```
Anaconda Powershell Prompt (anaconda3)

(base) PS C:\Users\LENOVO> cd .\Downloads\
(base) PS C:\Users\LENOVO\Downloads> cd .\Project\
(base) PS C:\Users\LENOVO\Downloads\Project> streamlit run salesProject.py

You can now view your Streamlit app in your browser.

URL: http://localhost:8501
```

Figure 8: the terminal for leading the path of your file

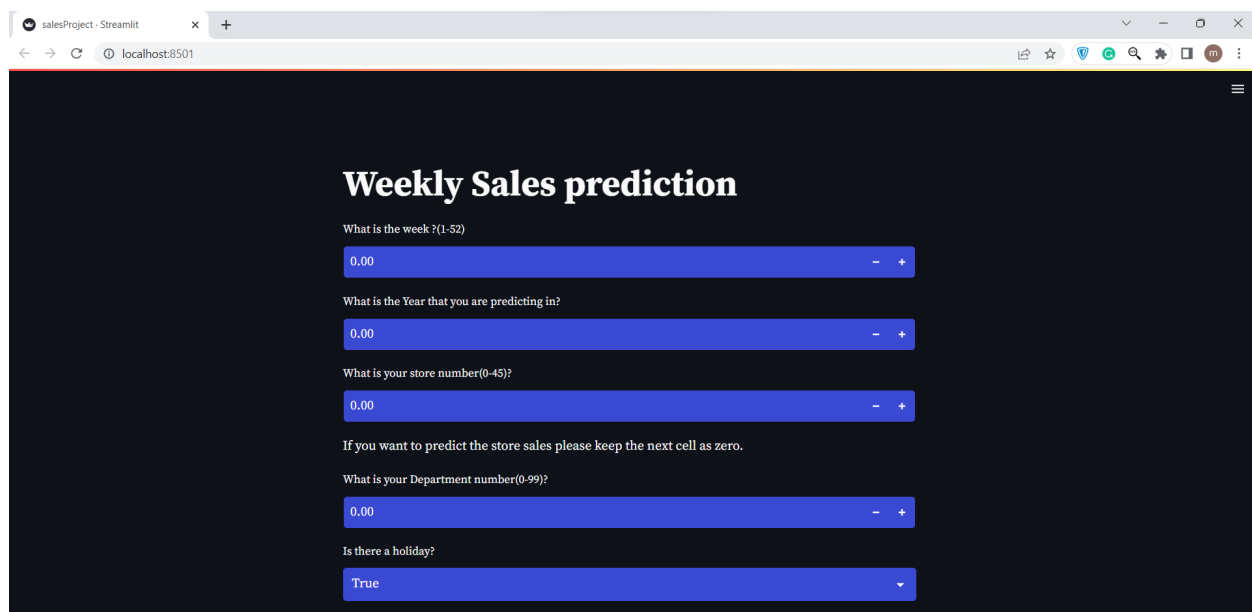


Figure 9: the local host webpage it's redirected to

