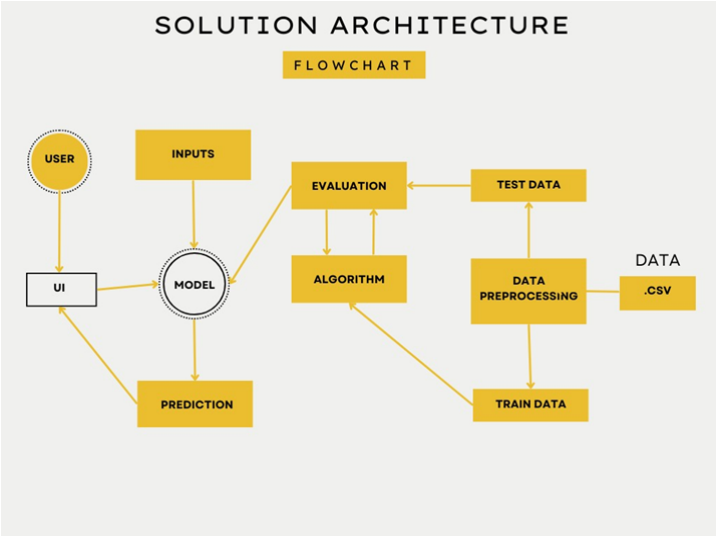
WALMART SALES FORECASTING

PROJECT DESCRIPTION:

Revolutionise Walmart's sales projections by leveraging advanced analytics, mining historical data, and implementing cutting-edge predictive models. This project aims to transcend conventional forecasting methods, introducing adaptability to swiftly changing market landscapes. The core objective is to equip Walmart decision-makers with nuanced insights drawn from diverse data sources, fostering a holistic understanding of sales trends, inventory dynamics, and external influencers.

The emphasis is on creating a scalable system that seamlessly accommodates escalating data volumes and user demands. By integrating diverse forecasting models and harnessing sophisticated algorithms, the project strives for heightened accuracy and real-time adaptability. Walmart seeks to redefine its position in the retail landscape by embracing a data-driven paradigm, ensuring strategic decision-making aligns seamlessly with the dynamic nature of the retail sector. This initiative encapsulates a vision of an agile, responsive, and forward-looking sales forecasting system, poised to propel Walmart into the vanguard of innovation within the retail industry.

Architecture:



The structure of the software is as follows:

- **User Inputs:** This component provides a user interface for the user to input data into the system.
- **Data Preprocessing:** This component cleans and prepares the data for training and evaluation. This may include tasks such as removing outliers, scaling the data, and converting the data to a format that is compatible with the chosen machine learning algorithm.
- **Model:** This component represents the machine learning model. The model is trained on a set of data and then used to make predictions on new data. Evaluation: This component evaluates the performance of the model on a held-out test set. This helps to ensure that the model is able to generalize to new data.
- **Prediction:** This component uses the trained model to make predictions on new data

Pre-requisites:

To complete this project, knowledge of the following software, concepts and packages is required.

Python packages

o Open anaconda prompt as administrator

o Type “pip install numpy” and click enter.

o Type “pip install pandas” and click enter.

o Type “pip install scikit-learn” and click enter.

o Type “pip install matplotlib” and click enter.

o Type “pip install seaborn” and click enter.

o Type “pip install pmdarima” and click enter.

Project Flow:

1. Define Objectives and Scope:

Clearly define the objectives of the sales forecasting project. Determine the scope, including the time period and specific products or categories.

2. Data Collection:

Gather historical sales data from Walmart's stores, both online and brick-and-mortar. Include relevant features such as date, product details, promotions, and external factors like holidays.

3. Data Cleaning and Preprocessing:

Clean the data by handling missing values, outliers, and ensuring data consistency.
Preprocess the data by converting date columns, encoding categorical variables, and scaling if necessary.

4. Data Splitting:

Split the dataset into training and testing sets. Consider using a time-based split to ensure the model is trained on historical data and tested on more recent data.

5. Model Selection:

Choose appropriate models for sales forecasting. Common choices include time series models like ARIMA, machine learning models like Random Forests or Gradient Boosting, and deep learning models if the dataset is large.

6. Model Training:

Train the selected models on the training dataset. Fine-tune hyperparameters and evaluate model performance using appropriate metrics.

7. Model Evaluation:

Evaluate the models on the testing dataset using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared.

8. Ensemble Learning (Optional):

Consider implementing ensemble learning techniques if individual models show variability. Techniques like stacking or blending models can improve overall accuracy.

9. Hyperparameter Tuning (Optional):

Fine-tune hyperparameters to optimize model performance. Use techniques like grid search or random search.

Project Structure:

Importing Libraries:

Import the necessary libraries as shown below

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import statsmodels.api as sm

from sklearn.preprocessing import MinMaxScaler

import pickle

from os import path

from sklearn import metrics

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from xgboost import XGBRegressor

from keras.models import Sequential

from keras.layers import Dense

from scikeras.wrappers import KerasRegressor
```

Read the Dataset:

The dataset format might be in .csv, .excel files, .txt, .json, ,etc. So, the dataset can be read with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

All the datasets are used in the same way.

data = pd.read_csv('train.csv')

stores = pd.read_csv('stores.csv')

features = pd.read_csv('features.csv')

After reading the datasets we will be viewing them

Training Dataset

data.shape

(421570, 5)

data.tail()

	Store	Dept	Date	Weekly_Sales	IsHoliday
421565	45	93	10/26/2012	2487.80	False
421566	45	94	10/26/2012	5203.31	False
421567	45	95	10/26/2012	56017.47	False
421568	45	97	10/26/2012	6817.48	False
421569	45	98	10/26/2012	1076.80	False

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 421570 entries, 0 to 421569

Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	Store	421570 non-null	int64
1	Dept	421570 non-null	int64
2	Date	421570 non-null	object
3	Weekly_Sales	421570 non-null	float64
4	IsHoliday	421570 non-null	bool

dtypes: bool(1), float64(1), int64(2), object(1)

memory usage: 13.3+ MB

Dataset containing info of Stores:

stores.shape

(45, 3)

stores.tail()

	Store	Type	Size
40	41	A	196321
41	42	C	39690
42	43	C	41062

43	44	C	39910
44	45	B	118221

stores.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 45 entries, 0 to 44

Data columns (total 3 columns):

Column Non-Null Count Dtype

--- ------ -------------- -----

0 Store 45 non-null int64

1 Type 45 non-null object

2 Size45 non-null int64

dtypes: int64(2), object(1)

memory usage: 1.2+ KB

Dataset containing additional data of Stores:

features.shape

(8190, 12)

features.tail()

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
8185	41	7/26/2013	67.56	3.582	497.670000	1454.290000	6.30	4.000000	2418.00	172.460809	7.826821	False

8186	42	7/26/2013	83.32	3.865	7032.371786	3384.176594	0.17	3292.935886	756.79	172.460809	7.826821	False
8187	43	7/26/2013	79.13	3.620	43.370000	3384.176594	1.18	3292.935886	531.35	172.460809	7.826821	False
8188	44	7/26/2013	83.62	3.669	134.310000	3384.176594	1.00	3292.935886	199.75	172.460809	7.826821	False
8189	45	7/26/2013	76.06	3.804	212.020000	851.730000	2.06	10.880000	1864.57	172.460809	7.826821	False

features.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 8190 entries, 0 to 8189

Data columns (total 12 columns):

#	Column	Non-Null Count	Dtype
0	Store	8190 non-null	int64
1	Date	8190 non-null	object
2	Temperature	8190 non-null	float64
3	Fuel_Price	8190 non-null	float64
4	MarkDown1	8190 non-null	float64
5	MarkDown2	8190 non-null	float64

6 MarkDown3 8190 non-null float64

7 MarkDown4 8190 non-null float64

8 MarkDown5 8190 non-null float64

9 CPI 8190 non-null float64

10 Unemployment 8190 non-null float64

11 IsHoliday 8190 non-null bool

dtypes: bool(1), float64(9), int64(1), object(1)

memory usage: 712.0+ KB

Handling missing values of features dataset:

```
features["CPI"].fillna(features["CPI"].median(),inplace=True)

features["Unemployment"].fillna(features["Unemployment"].median(),inplace=True)

 for i in range(1,6):

features["MarkDown"+str(i)] = features["MarkDown"+str(i)].apply(lambda x: 0 if x < 0 else x)

features["MarkDown"+str(i)].fillna(value=0,inplace=True)
```

Merging Training Dataset and merged stores-features Dataset:

```
data = pd.merge(data,stores,on='Store',how='left')
data = pd.merge(data,features,on=['Store','Date'],how='left')
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values(by=['Date'],inplace=True)
data.set_index(data.Date, inplace=True)
data['IsHoliday_x'].isin(data['IsHoliday_y']).all()
True
```

```
data.drop(columns='IsHoliday_x',inplace=True)

data.rename(columns={"IsHoliday_y" : "IsHoliday"}, inplace=True)
data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 421570 entries, 2010-02-05 to 2012-10-26
Data columns (total 16 columns):
#   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Store        421570 non-null  int64
 1   Dept         421570 non-null  int64
 2   Date         421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64
 4   Type         421570 non-null  object
 5   Size         421570 non-null  int64
 6   Temperature  421570 non-null  float64
 7   Fuel_Price   421570 non-null  float64
 8   MarkDown1    421570 non-null  float64
 9   MarkDown2    421570 non-null  float64
 10  MarkDown3    421570 non-null  float64
 11  MarkDown4    421570 non-null  float64
 12  MarkDown5    421570 non-null  float64
 13  CPI          421570 non-null  float64
 14  Unemployment  421570 non-null  float64
 15  IsHoliday    421570 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(10), int64(3), object(1) memory usage: 51.9+ MB
```

Splitting Date Column:

```
data['Year'] = data['Date'].dt.year

data['Month'] = data['Date'].dt.month
data['Week'] = data['Date'].dt.week
```

Outlier Detection and Abnormalities:

```
agg_data = data.groupby(['Store', 'Dept']).Weekly_Sales.agg(['max', 'min', 'mean', 'median', 'std']).reset_index()
agg_data.isnull().sum()
```

```
Store    0
Dept     0
max      0
min      0
mean     0
median           0
std       37
dtype: int64
```

```
store_data = pd.merge(left=data,right=agg_data,on=['Store', 'Dept'],how ='left')

store_data.dropna(inplace=True)

data = store_data.copy()
del store_data

data['Date'] = pd.to_datetime(data['Date'])

data.sort_values(by=['Date'],inplace=True)
data.set_index(data.Date, inplace=True)
```

```
data['Total_MarkDown'] =
data['MarkDown1']+data['MarkDown2']+data['MarkDown3']+data['MarkDown4']+data['MarkDown5']
```

```
data.drop(['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5'], axis =
1,inplace=True)
```

```
numeric_col =
['Weekly_Sales','Size','Temperature','Fuel_Price','CPI','Unemployment','Total_MarkDown']
```

```
data_numeric = data[numeric_col].copy()
```

```
data.shape
```

```
(421533, 20)
```

```
data = data[(np.abs(stats.zscore(data_numeric)) < 2.5).all(axis = 1)]
```

```
data.shape
```

```
(377857, 20)
```

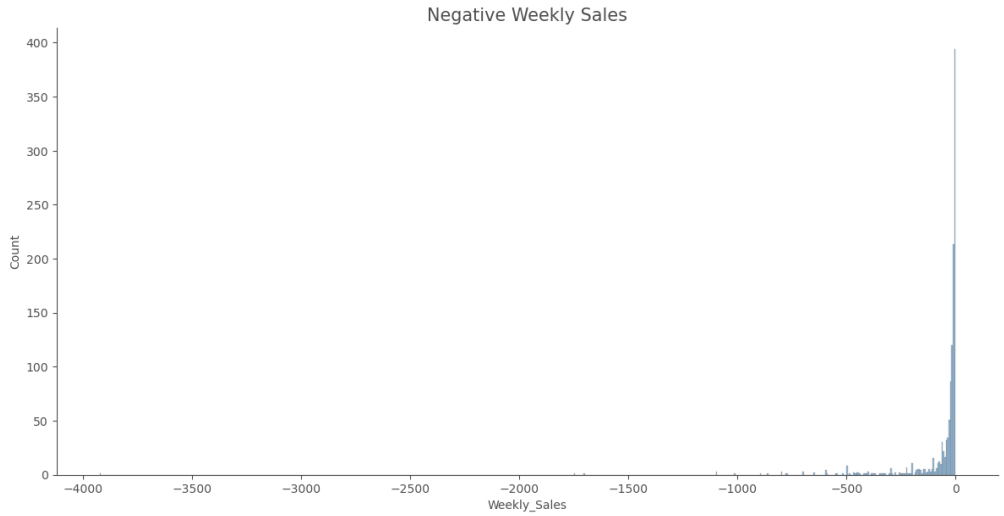
Negative Weekly Sales:

```
y = data["Weekly_Sales"][data.Weekly_Sales < 0]
```

```
sns.displot(y,height=6,aspect=2)
```

```
plt.title("Negative Weekly Sales", fontsize=15)
```

```
plt.savefig('negative_weekly_sales.png')
plt.show()
```




```
data=data[data['Weekly_Sales']>=0]
```

```
data.shape
```

(376657, 20)

```
data['IsHoliday'] = data['IsHoliday'].astype('int')
```

```
data
```

```
data.to_csv('preprocessed_walmart_dataset.csv')
```

Data Visualisation:

Average Monthly Sales:

```
plt.figure(figsize=(14,8))
```

```
sns.barplot(x='Month',y='Weekly_Sales',data=data)
```

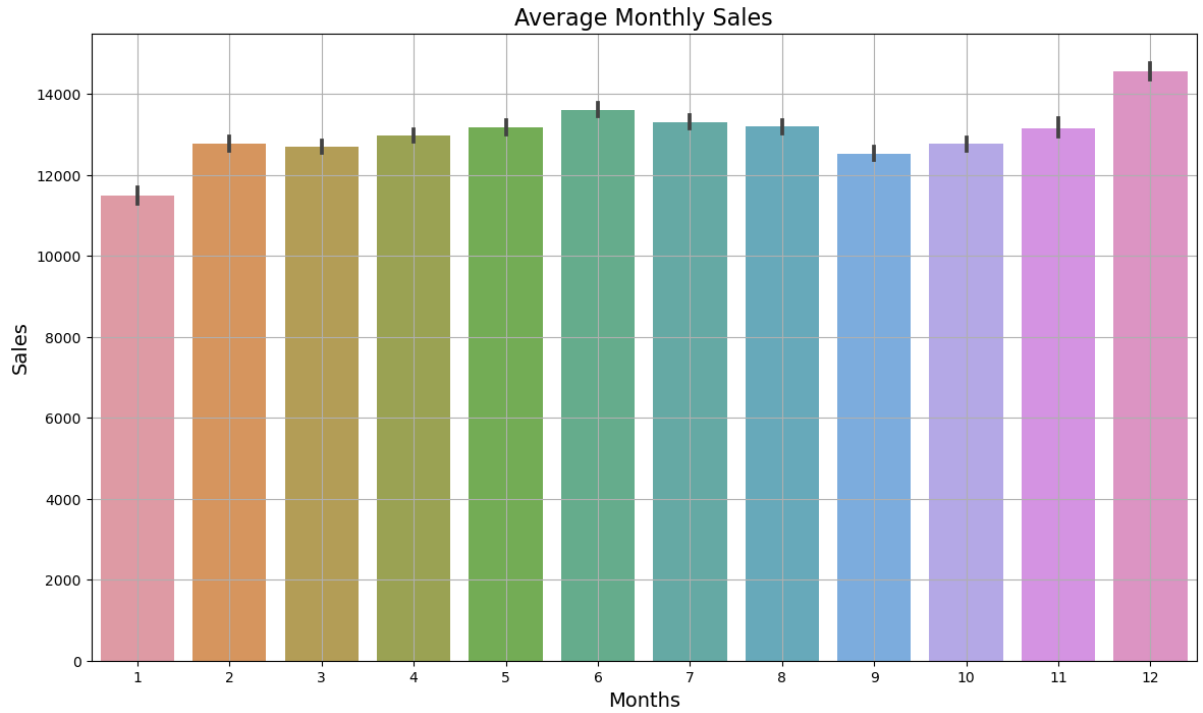
```
plt.ylabel('Sales',fontsize=14)
```

```
plt.xlabel('Months',fontsize=14)
```

```
plt.title('Average Monthly Sales',fontsize=16)
```

```
plt.savefig('avg_monthly_sales.png')
```

```
plt.grid()
```



Monthly Sales for Each Year:

```
data_monthly = pd.crosstab(data["Year"], data["Month"],
values=data["Weekly_Sales"],aggfunc='sum')

data_monthly

fig, axes = plt.subplots(3,4,figsize=(16,8))

plt.suptitle('Monthly Sales for each Year', fontsize=18)

k=1

for i in range(3):

        for j in range(4):

    sns.lineplot(ax=axes[i,j],data=data_monthly[k])

    plt.subplots_adjust(wspace=0.4,hspace=0.32)

        plt.ylabel(k,fontsize=12)

        plt.xlabel('Years',fontsize=12)

        k+=1
```

plt.savefig('monthly_sales_every_year.png')

plt.show()

Average Weekly Sales Store wise:

plt.figure(figsize=(20,8))

sns.barplot(x='Store',y='Weekly_Sales',data=data)

plt.grid()

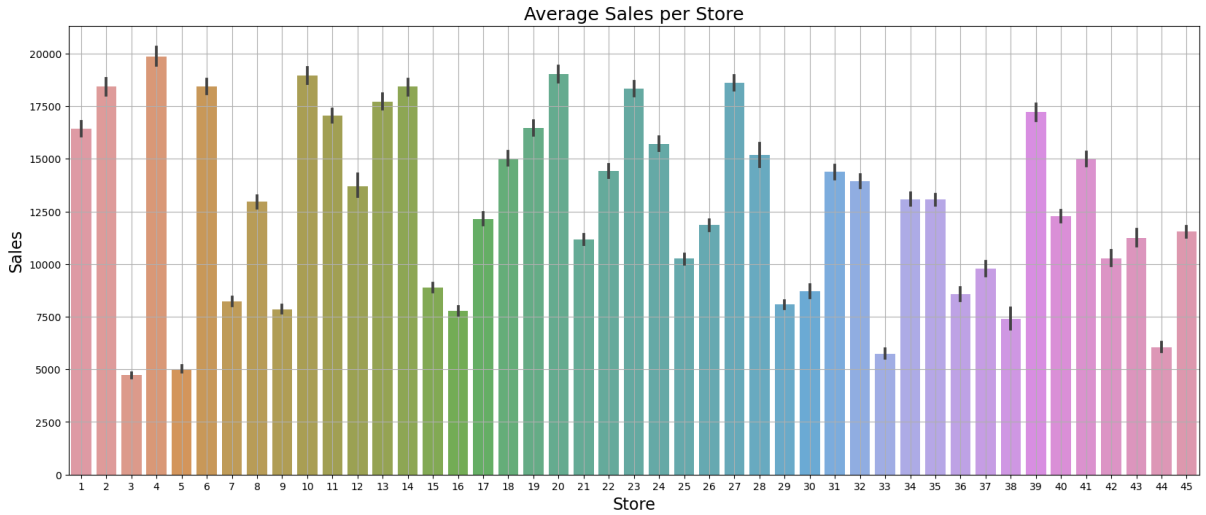
plt.title('Average Sales per Store', fontsize=18)

plt.ylabel('Sales', fontsize=16)

plt.xlabel('Store', fontsize=16)

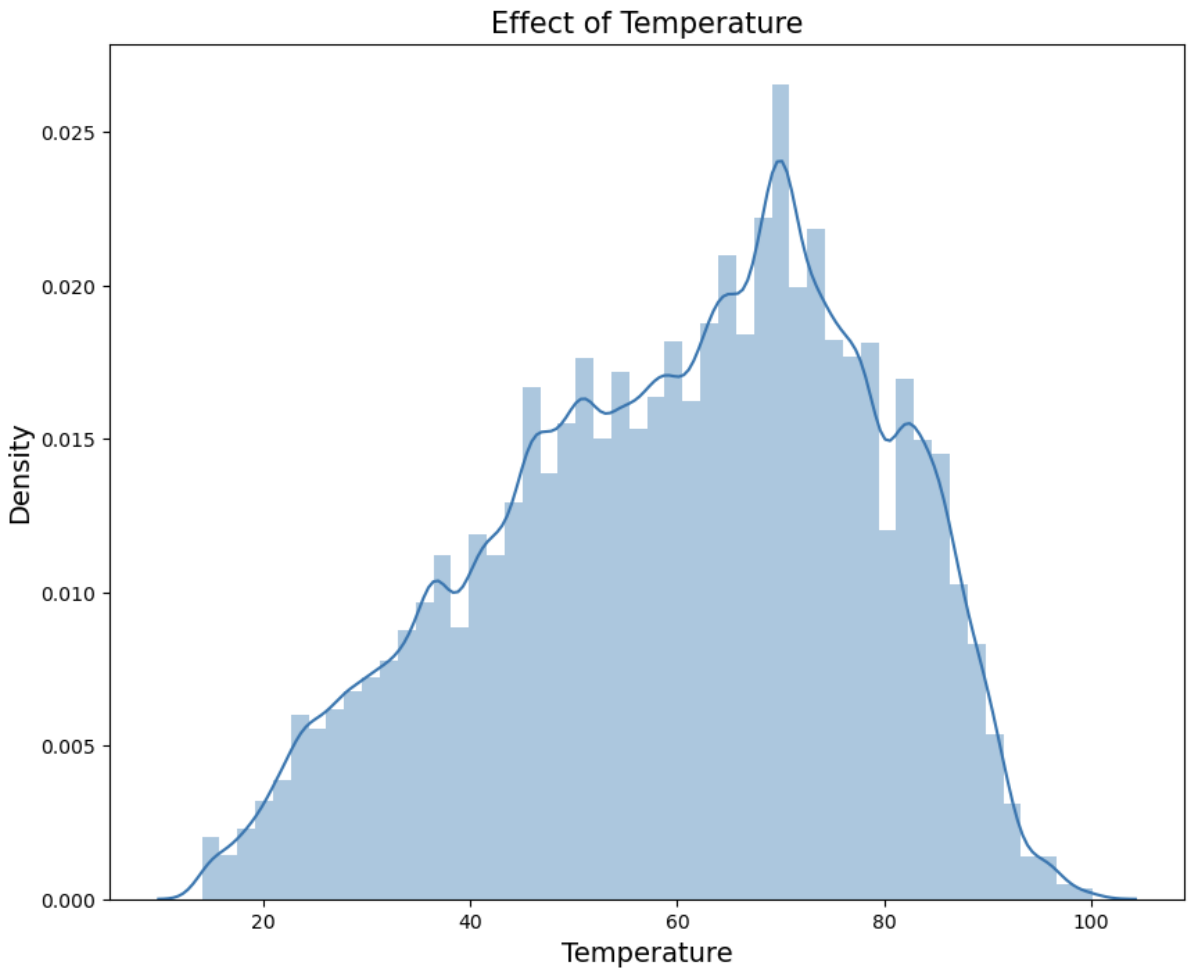
plt.savefig('avg_sales_store.png')

plt.show()



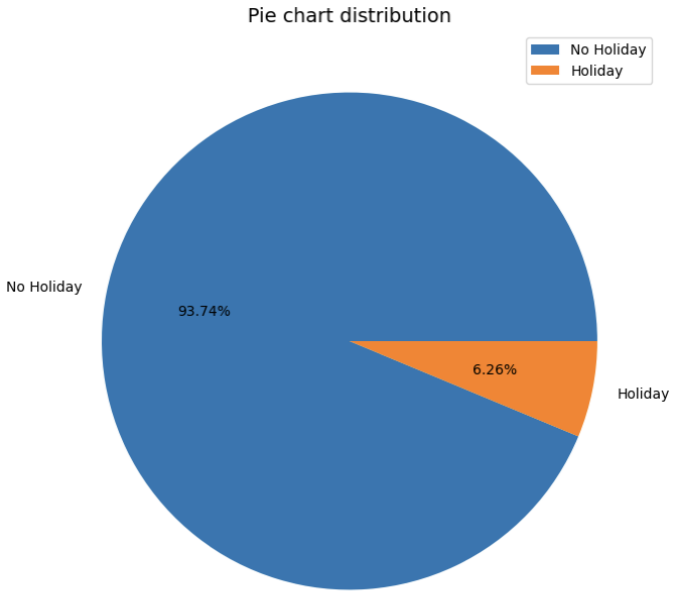
Sales Vs Temperature:

```
plt.figure(figsize=(10,8))

sns.distplot(data['Temperature'])

plt.title('Effect of Temperature',fontsize=15)

plt.xlabel('Temperature',fontsize=14)

plt.ylabel('Density',fontsize=14)

plt.savefig('effect_of_temp.png')

plt.show()
```



Holiday Distribution:

```
plt.figure(figsize=(8,8))

plt.pie(data['IsHoliday'].value_counts(),labels=['No Holiday','Holiday'],autopct='%0.2f%%')

plt.title("Pie chart distribution",fontsize=14)

plt.legend()

plt.savefig('holiday_distribution.png')

plt.show()
```

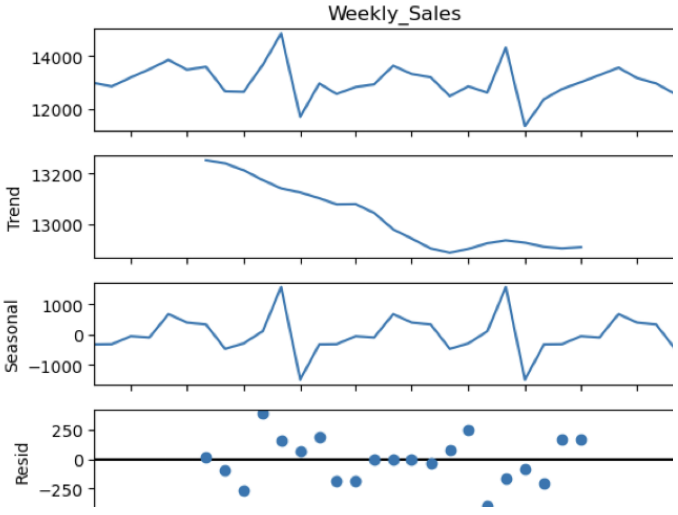


Time Series Decompose:

sm.tsa.seasonal_decompose(data['Weekly_Sales'].resample('MS').mean(), model='additive').plot()

plt.savefig('seasonal_decompose.png')

plt.show()



One-hot-encoding:

```
cat_col = ['Store','Dept','Type']

data_cat = data[cat_col].copy()

data_cat.tail()
```

Date	Store	Dept	Type
2012-10-26	45	95	B
2012-10-26	45	58	B
2012-10-26	45	23	B
2012-10-26	45	85	B
2012-10-26	45	98	B

```
data_cat = pd.get_dummies(data_cat,columns=cat_col)

data_cat.head()

data.shape
```

(376657, 20)

```
data = pd.concat([data, data_cat],axis=1)

data.shape
```

(376657, 149)

```
data.drop(columns=cat_col,inplace=True)
```

```
data.drop(columns=['Date'],inplace=True)

data.shape

(376657, 145)
```

Data Normalization:

```
num_col
['Weekly_Sales','Size','Temperature','Fuel_Price','CPI','Unemployment','Total_MarkDown','max','min','mean','median','std']

minmax_scale = MinMaxScaler(feature_range=(0, 1))

def normalization(df,col):

  for i in col:

        arr = df[i]

        arr = np.array(arr)

        df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))

  return df

data.head()

data = normalization(data.copy(),num_col)

data.head()
```

Correlation between features of dataset:

```
plt.figure(figsize=(15,8))


corr = data[num_col].corr()


sns.heatmap(corr,vmax=1.0,annot=True)


plt.title('Correlation Matrix',fontsize=16)


plt.savefig('correlation_matrix.png')
plt.show()
```

Recursive Feature Elimination:

```
feature_col = data.columns.difference(['Weekly_Sales'])
feature_col


Index(['CPI', 'Dept_1', 'Dept_10', 'Dept_11', 'Dept_12', 'Dept_13', 'Dept_14',
       'Dept_16', 'Dept_17', 'Dept_18',
       ...
       'Type_B', 'Type_C', 'Unemployment', 'Week', 'Year', 'max', 'mean',
       'median', 'min', 'std'],
      dtype='object', length=144)
 '''
param_grid={'n_estimators':np.arange(10,25)}
tree=GridSearchCV(RandomForestRegressor(oob_score=False,warm_start=True),param_grid,cv=5)
tree.fit(data_train[feature_col],data_train['Weekly_Sales'])
'''\nparam_grid={'n_estimators':np.arange(10,25)}\ntree=GridSearchCV(RandomForestRegressor(oob_score=False,warm_start=True),param_grid,cv=5)\ntree.fit(data_train[feature_col],data_train['Weekly_Sales'])\n'
 #tree.best_params_
 radm_clf = RandomForestRegressor(oob_score=True,n_estimators=23)
radm_clf.fit(data[feature_col], data['Weekly_Sales'])
 pkl_filename = "feature_elim_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
```

```
  with open(pkl_filename, 'wb') as file:
        pickle.dump(radm_clf, file)
print("Saved model to disk")
else:
  print("Model already saved")
```

Saved model to disk

```
indices = np.argsort(radm_clf.feature_importances_)[::-1]
feature_rank = pd.DataFrame(columns = ['rank', 'feature', 'importance'])
 for f in range(data[feature_col].shape[1]):
        feature_rank.loc[f] = [f+1,
                data[feature_col].columns[indices[f]],
                radm_clf.feature_importances_[indices[f]]]
 feature_rank
```

	rank	feature	importance
0	1	mean	4.899501e-01
1	2	median	4.380400e-01
2	3	Week	1.967489e-02
3	4	Temperature	8.771803e-03
4	5	CPI	5.885204e-03

...	...	...	...
139	140	Dept_51	2.360081e-10
140	141	Dept_45	2.115956e-10
141	142	Dept_78	4.336395e-12
142	143	Dept_39	8.461573e-15
143	144	Dept_43	2.175517e-15

144 rows × 3 columns

```
x=feature_rank.loc[0:22,['feature']]

x=x['feature'].tolist()

print(x)
```

['mean', 'median', 'Week', 'Temperature', 'CPI', 'max', 'Fuel_Price', 'min', 'Unemployment', 'std', 'Month', 'Total_MarkDown', 'Dept_16', 'Dept_18', 'IsHoliday', 'Size', 'Dept_3', 'Year', 'Dept_1', 'Dept_9', 'Dept_11', 'Dept_5', 'Dept_7']

```
X = data[x]

Y = data['Weekly_Sales']

data = pd.concat([X,Y],axis=1)

data

data.to_csv('final_data.csv')
```

Data Splitted into Training, Validation, Test

```
X = data.drop(['Weekly_Sales'],axis=1)
Y = data.Weekly_Sales
```

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.20, random_state=50)
```

EXTRA TREES METHOD:

```
 from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
 # Initialize the Extra Trees regressor
et_regressor = ExtraTreesRegressor(n_estimators=100)
 # Fit the regressor on the training data
et_regressor.fit(X_train, y_train)
 # Make predictions on the test set
y_pred = et_regressor.predict(X_test)
 # Calculate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
 print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Value: {r2}")
```

Mean Squared Error (MSE): 0.0008816651590708112

Mean Absolute Error (MAE): 0.014999600508864733

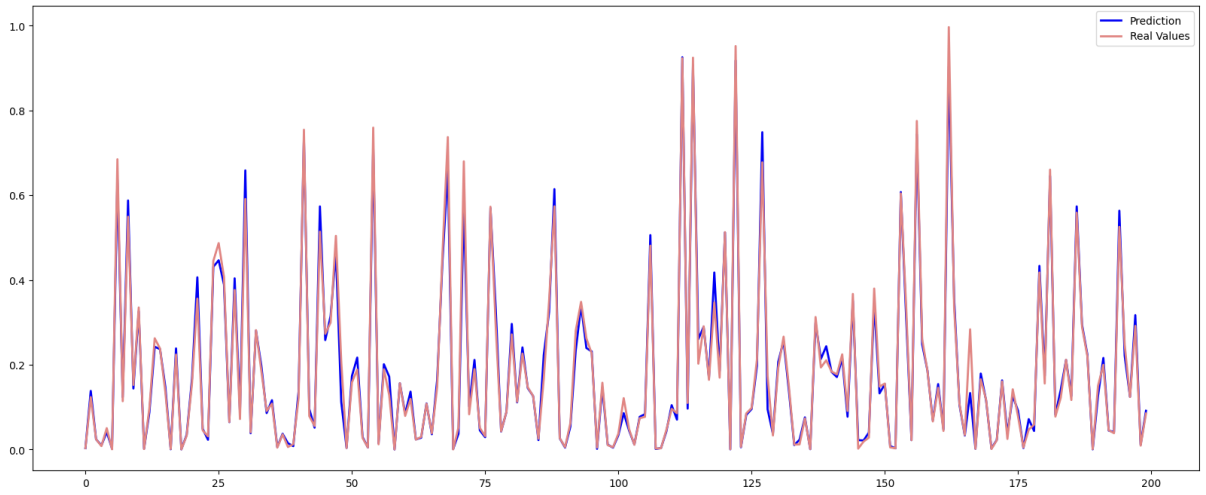
Root Mean Squared Error (RMSE): 0.029692846934418586

R-squared (R2) Value: 0.9803685869647454

```
er = ExtraTreesRegressor()
er.fit(X_train, y_train)
er_acc = er.score(X_test,y_test)*100
print("Accuracy - ",er_acc)
```

Accuracy - 98.04031174361452

```
import matplotlib.pyplot as plt
 # Create a figure with a larger size
plt.figure(figsize=(20, 8))
 # Plot the first 200 predicted values in blue
plt.plot(y_pred[:200], label="Prediction", linewidth=2.0, color='blue')
 # Plot the first 200 actual values in light coral
plt.plot(y_test[:200].values, label="Real Values", linewidth=2.0, color='lightcoral')
 # Add a legend to the plot
plt.legend(loc="best")
 # Save the plot as an image
plt.savefig('extra_trees_real_pred.png')
 # Display the plot
plt.show()
```



LightGBM (Light Gradient Boosting Machine):

```
 import lightgbm as lgb

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score,r2_score

import matplotlib.pyplot as plt

train_data = lgb.Dataset(X_train, label=y_train)

params = {

        'objective': 'regression',  # for regression task

        'metric': 'mse',  # mean squared error as the evaluation metric

    'boosting_type': 'gbdt',  # gradient boosting decision tree

    'num_leaves': 31,  # number of leaves in each tree

    'learning_rate': 0.05,

    'feature_fraction': 0.9,
```

```
}
 # Train the LightGBM model
num_round = 100  # Number of boosting rounds (you can adjust this)
bst = lgb.train(params, train_data, num_round)
 # Make predictions on the test set
y_pred = bst.predict(X_test, num_iteration=bst.best_iteration)
 # Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
explained_var = explained_variance_score(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R2) Value: {r2}")
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"Explained Variance: {explained_var}")
 # Create a DataFrame to compare actual and predicted values
lgbm_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lgbm_df.to_csv('lgbm_real_pred.csv')
 # Create a plot to visualize the results (first 200 data points)
plt.figure(figsize=(20, 8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0, color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0, color='lightcoral')
plt.legend(loc="best")
plt.savefig('lgbm_real_pred.png')
plt.show()
```

bst.save_model('lgbm_model.txt')

print("Saved model to disk")

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.067988 seconds.

You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 2668

[LightGBM] [Info] Number of data points in the train set: 301325, number of used features: 23

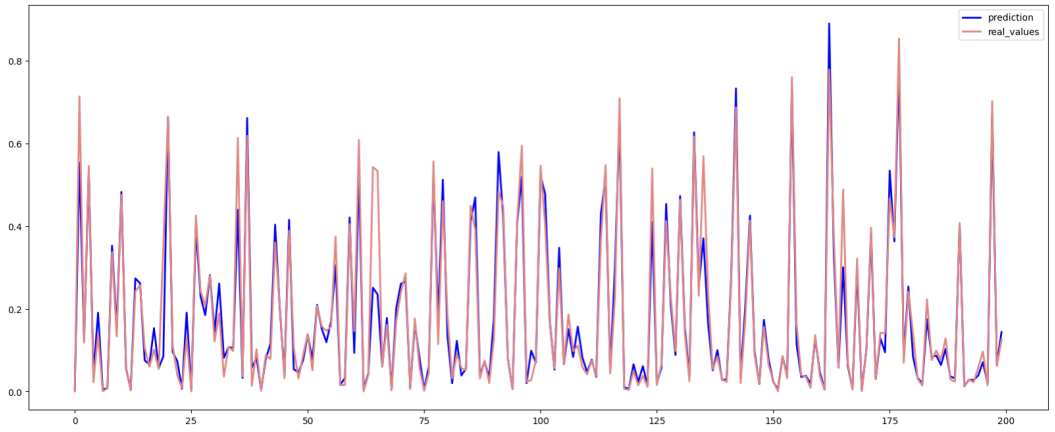
[LightGBM] [Info] Start training from score 0.179686

R-squared (R2) Value: 0.9551490411617666

Mean Squared Error: 0.002024916012821558

Mean Absolute Error: 0.024645265935532106

Explained Variance: 0.955149091340892



Linear Regression Model:

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler

```
# Initialize the linear regression model

lr = LinearRegression()

# Initialize the StandardScaler

scaler = StandardScaler()

# Fit and transform the training data with the scaler

X_train_scaled = scaler.fit_transform(X_train)

X_train=X_train_scaled

# Fit the linear regression model with the scaled data

lr.fit(X_train_scaled, y_train)

lr_acc = lr.score(X_test,y_test)*100

print("Linear Regressor Accuracy - ",lr_acc)

y_pred = lr.predict(X_test)

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print("R2" , metrics.explained_variance_score(y_test, y_pred)) lr_df =
pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

lr_df.to_csv('lr_real_pred.csv')

Lr_df

plt.figure(figsize=(20,8))

plt.plot(lr.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')

plt.legend(loc="best")

plt.savefig('lr_real_pred.png')

plt.show()
```

Linear Regressor Accuracy - -550.4425523482787

MAE 0.024645265935532106

MSE 0.002024916012821558

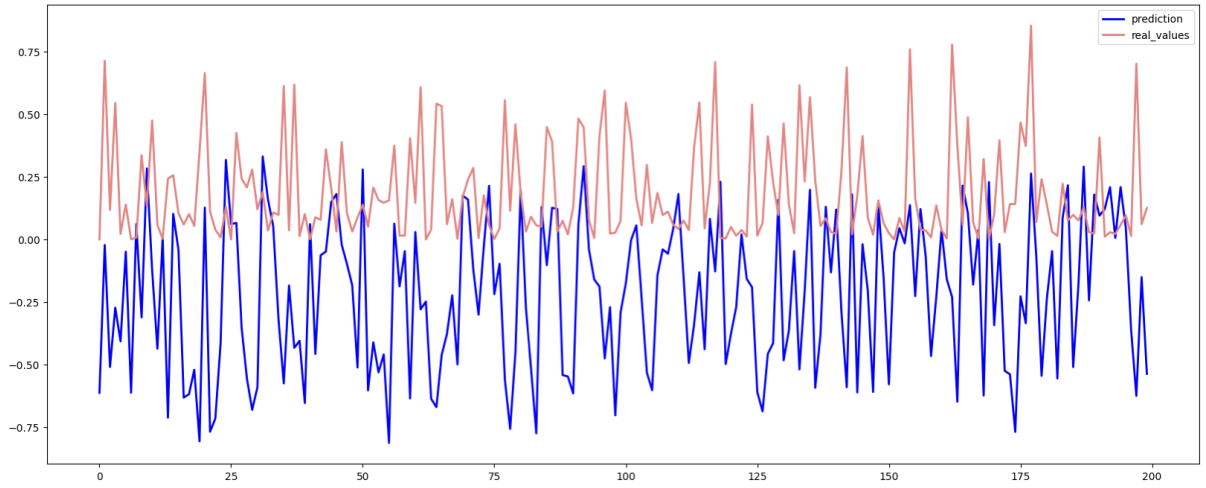
RMSE 0.04499906679945216

R2 0.955149091340892

Actual / Date	Predicted	
2010-10-22	0.000404	-0.612121
2010-05-07	0.713607	-0.021834
2011-09-23	0.118297	-0.509250
2010-08-06	0.545391	-0.271982
2012-08-17	0.022539	-0.406744
...	...	...
2011-09-30	0.137852	-0.534517

2011-10-07	0.055045	-0.544479
2011-02-18	0.338530	0.196864
2011-10-28	0.631689	-0.546636
2011-05-20	0.037323	-0.130799

75332 rows × 2 columns



Saving trained model:

```
pkl_filename = "linear_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
        pickle.dump(lr, file)
```

```
  print("Saved model to disk")
else:
  print("Model already saved")
```

Saved model to disk

ADABOOST CLASSIFIER:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import AdaBoostRegressor  # Import AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
 # Fit the classifier on the training data
ada_regressor = AdaBoostRegressor(n_estimators=100)  # You can choose a different number of estimators
 # Fit the regressor on the training data
ada_regressor.fit(X_train, y_train)
 # Make predictions on the test set
y_pred = ada_regressor.predict(X_test)
 # Calculate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
 print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

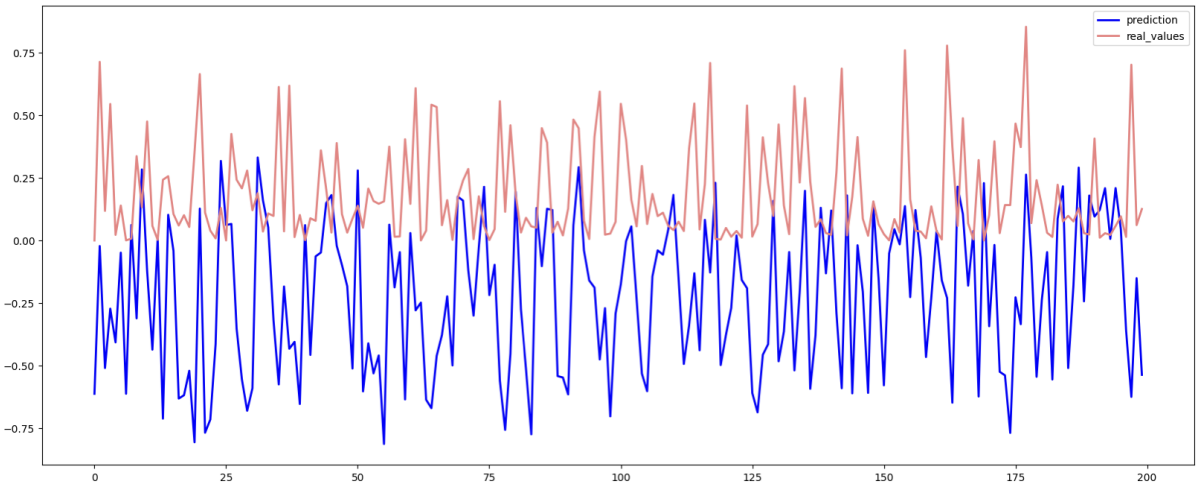
```
print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R-squared (R2) Value: {r2}")

 # Create a scatter plot of predicted vs. actual values

plt.figure(figsize=(20, 8))

plt.plot(y_pred[:200], label="Prediction", linewidth=2.0, color='blue')

plt.plot(y_test[:200].values, label="Real Values", linewidth=2.0, color='lightcoral')

plt.legend(loc="best")

plt.savefig('ada_real_pred.png')

plt.show()
```

Mean Squared Error (MSE): 0.005677102799603772

Mean Absolute Error (MAE): 0.04545054862894153

Root Mean Squared Error (RMSE): 0.07534655134512643

R-squared (R2) Value: 0.8735919767771264



Random Forest Regressor Model:

```
rf = RandomForestRegressor()
```

```
rf.fit(X_train, y_train)
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

rf_acc = rf.score(X_test,y_test)*100
print("Random Forest Regressor Accuracy - ",rf_acc)
Random Forest Regressor Accuracy -  97.88907135637824
y_pred = rf.predict(X_test)

from sklearn.ensemble import RandomForestRegressor

n_estimators = 100  # can change this value to your desired number of trees

rf_model = RandomForestRegressor(n_estimators=n_estimators)

number_of_trees = rf_model.n_estimators

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

Number of trees in the Random Forest: 100

MAE 0.015440129853658986

MSE 0.000934034751993661

RMSE 0.03056198213456812

R2 0.9787502531437008

rf_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

rf_df.to_csv('./predictions/rf_real_pred.csv')

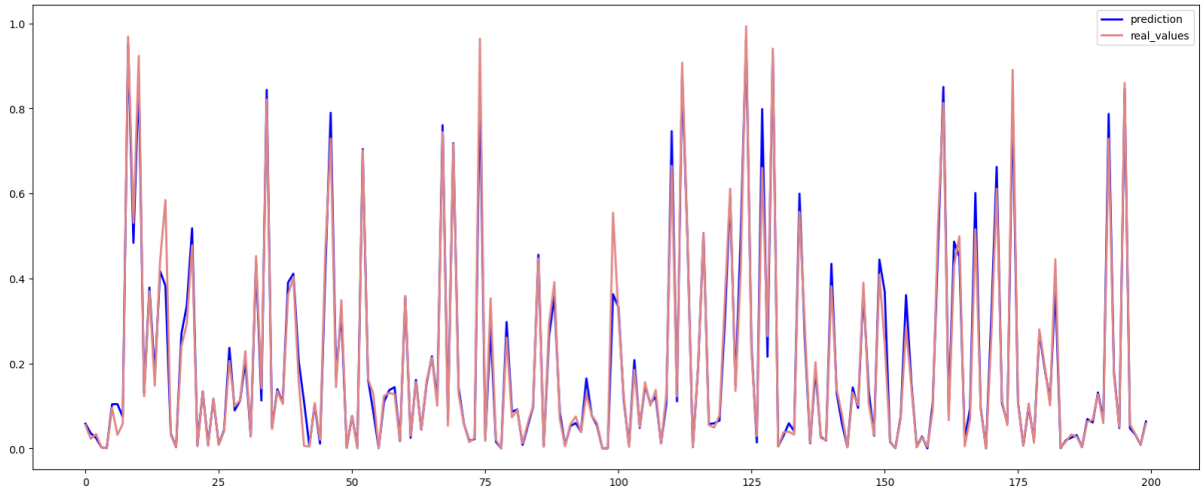
rf_df

Date	Actual	Predicted
2011-08-05	0.161661	0.124485
2010-07-09	0.364278	0.320277
2011-07-01	0.005003	0.012285
2012-01-06	0.015856	0.020360
2011-08-26	0.000318	0.000566
...	...	...
2011-01-28	0.169068	0.176886
2010-08-20	0.252860	0.272780
2010-11-26	0.265617	0.393226
2010-03-12	0.008865	0.015019

```
2010-02-12    0.230510    0.258844
```

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))

plt.plot(rf.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')

plt.legend(loc="best")

plt.savefig('plots/rf_real_pred.png')
plt.show()
```



Saving trained model:

```
pkl_filename = "./models/randomforest_regressor.pkl"

if (not path.isfile(pkl_filename)):

  # saving the trained model to disk

  with open(pkl_filename, 'wb') as file:
```

```
    pickle.dump(rf, file)

  print("Saved model to disk")

else:
  print("Model already saved")
```

Saved model to disk

K Neighbors Regressor Model:

```
knn = KNeighborsRegressor(n_neighbors = 1,weights = 'uniform')
knn.fit(X_train,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
```

```
knn_acc = knn.score(X_test, y_test)*100
print("KNeigbhbors Regressor Accuracy - ",knn_acc)
```

KNeigbhbors Regressor Accuracy - 91.97260309962996

```
y_pred = knn.predict(X_test)
print("MAE" , metrics.mean_absolute_error(y_test, y_pred))
print("MSE" , metrics.mean_squared_error(y_test, y_pred))
print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

MAE 0.033122163743083126

MSE 0.003624289656000884

RMSE 0.060202073519114635

R2 0.9199211034808975

```
knn_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

knn_df.to_csv('./predictions/knn_real_pred.csv')

knn_df

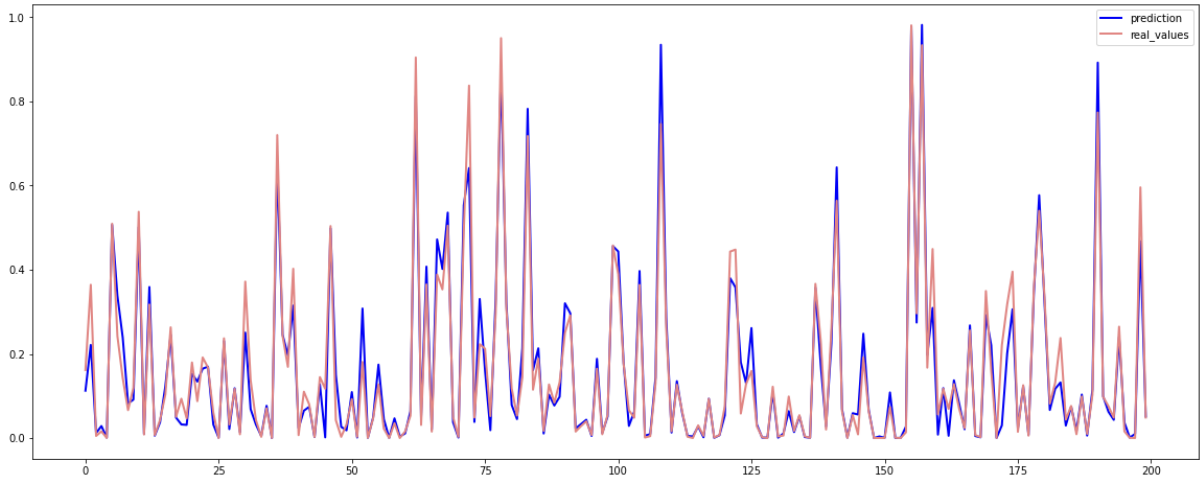
Date	Actual	Predicted
2011-08-05	0.161661	0.112559
2010-07-09	0.364278	0.221307
2011-07-01	0.005003	0.011921
2012-01-06	0.015856	0.028551
2011-08-26	0.000318	0.001063
...	...	...
2011-01-28	0.169068	0.229475
2010-08-20	0.252860	0.262688
2010-11-26	0.265617	0.203904
2010-03-12	0.008865	0.001663
2010-02-12	0.230510	0.287258

74850 rows × 2 columns

plt.figure(figsize=(20,8))

plt.plot(knn.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

```
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.savefig('plots/knn_real_pred.png')
plt.show()
```



Saving trained model:

```
pkl_filename = "./models/knn_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
          pickle.dump(knn, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

Saved model to disk

XGboost Model:

```
xgbr = XGBRegressor()
```

```
xgbr.fit(X_train, y_train)
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)

xgb_acc = xgbr.score(X_test,y_test)*100
print("XGBoost Regressor Accuracy - ",xgb_acc)
XGBoost Regressor Accuracy -  94.21152336133142
 y_pred = xgbr.predict(X_test)

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

MAE 0.026771808878560288

MSE 0.0026134394830486384

RMSE 0.051121810248157665

R2 0.9421152350249367

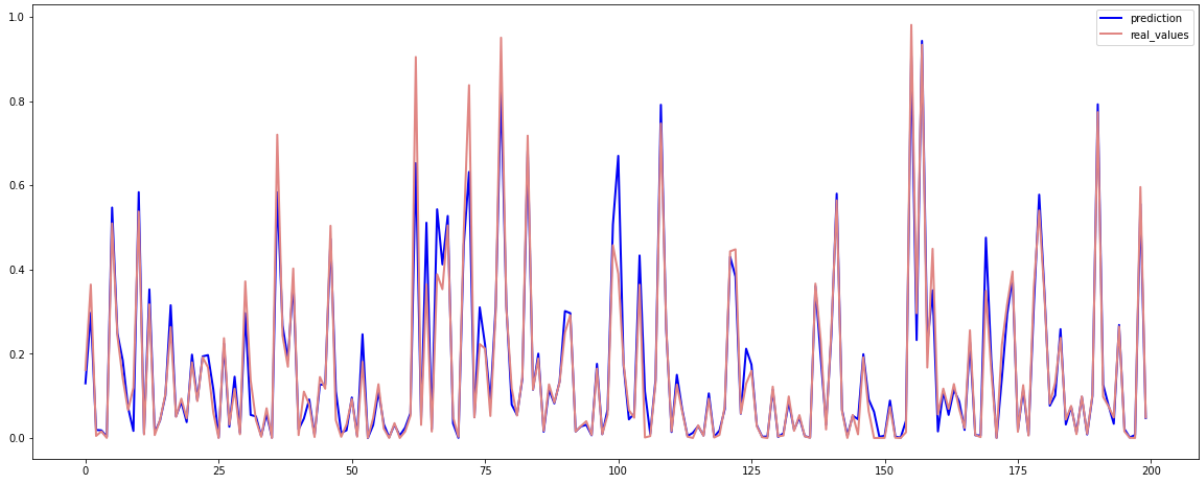
```
xgb_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

xgb_df.to_csv('./predictions/xgb_real_pred.csv')
xgb_df
```

Date	Actual	Predicted
2011-08-05	0.161661	0.129809
2010-07-09	0.364278	0.297181
2011-07-01	0.005003	0.019209
2012-01-06	0.015856	0.018191
2011-08-26	0.000318	0.002950
...	...	...
2011-01-28	0.169068	0.228197
2010-08-20	0.252860	0.234475
2010-11-26	0.265617	0.404794
2010-03-12	0.008865	0.011655
2010-02-12	0.230510	0.241285

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))

plt.plot(xgbr.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
```

```
plt.legend(loc="best")

plt.savefig('plots/xgb_real_pred.png')
plt.show()
```



Saving trained model:

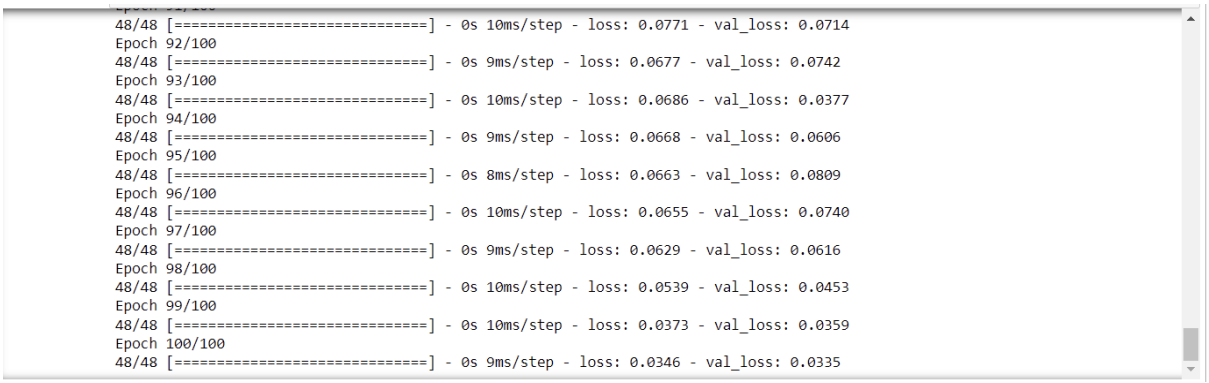
```
pkl_filename = "./models/xgboost_regressor.pkl"

if (not path.isfile(pkl_filename)):

  # saving the trained model to disk

  with open(pkl_filename, 'wb') as file:

    pickle.dump(xgbr, file)

  print("Saved model to disk")

else:
  print("Model already saved")
```

Saved model to disk

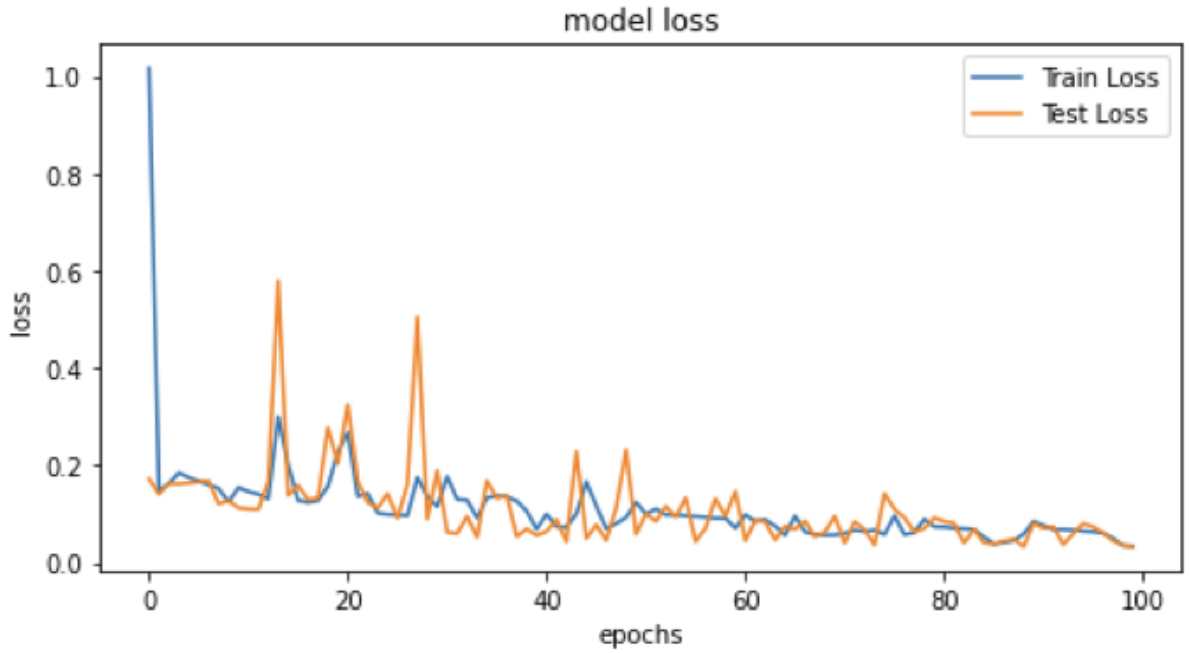
Custom Deep Learning Neural Network:

```
def create_model():
```

```
  model = Sequential()

  model.add(Dense(64, input_dim=X_train.shape[1],
kernel_initializer='normal',activation='relu'))

  model.add(Dense(32, kernel_initializer='normal'))

  model.add(Dense(1, kernel_initializer='normal'))

  model.compile(loss='mean_absolute_error', optimizer='adam')

  return model

estimator_model = KerasRegressor(build_fn=create_model, verbose=1)

history = estimator_model.fit(X_train, y_train, validation_split=0.2, epochs=100,
batch_size=5000)
```




```
plt.figure(figsize=(8,4))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val_loss'], label='Test Loss')

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epochs')

plt.legend(loc='upper right')

plt.savefig('plots/dnn_loss.png')

plt.show()
```



dnn_acc = metrics.r2_score(y_pred, y_test)*100

print("Deep Neural Network accuracy - ",dnn_acc)

Deep Neural Network accuracy - 90.50328742871066

y_pred = estimator_model.predict(X_test)

2340/2340 [==============================] - 3s 977us/step

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print("R2" , metrics.explained_variance_score(y_test, y_pred))

MAE 0.033255980538121045

MSE 0.0038670810150368187

RMSE 0.062185858641951856

R2 0.9144106847304281

dnn_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

dnn_df.to_csv('./predictions/dnn_real_pred.csv')

dnn_df

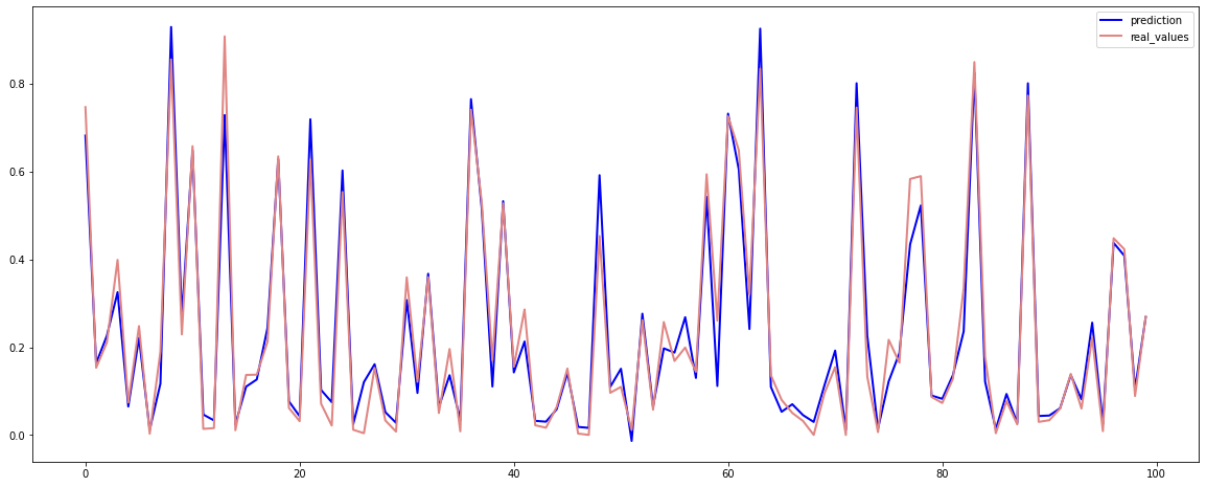
Date	Actual	Predicted
2011-08-05	0.161661	0.124761
2010-07-09	0.364278	0.289382
2011-07-01	0.005003	0.034531
2012-01-06	0.015856	0.024284
2011-08-26	0.000318	0.015496
...	...	...
2011-01-28	0.169068	0.233344
2010-08-20	0.252860	0.236093
2010-11-26	0.265617	0.342386
2010-03-12	0.008865	0.023427
2010-02-12	0.230510	0.242022

74850 rows × 2 columns

plt.figure(figsize=(20,8))

```
plt.plot(estimator_model.predict(X_test[200:300]), label="prediction",
linewidth=2.0,color='blue')

plt.plot(y_test[200:300].values, label="real_values", linewidth=2.0,color='lightcoral')

plt.savefig('plots/dnn_real_pred.png')

plt.legend(loc="best")
```

```
4/4 [==============================] - 0s 5ms/step

<matplotlib.legend.Legend at 0x7fc2eb110890>
```




```
filepath = './models/dnn_regressor.json'

weightspath = './models/dnn_regressor.h5'

if (not path.isfile(filepath)):

  # serialize model to JSON

  model_json = estimator_model.model.to_json()

  with open(filepath, "w") as json_file:

    json_file.write(model_json)

  print("Saved model to disk")

else:

  print("Model already saved")
```

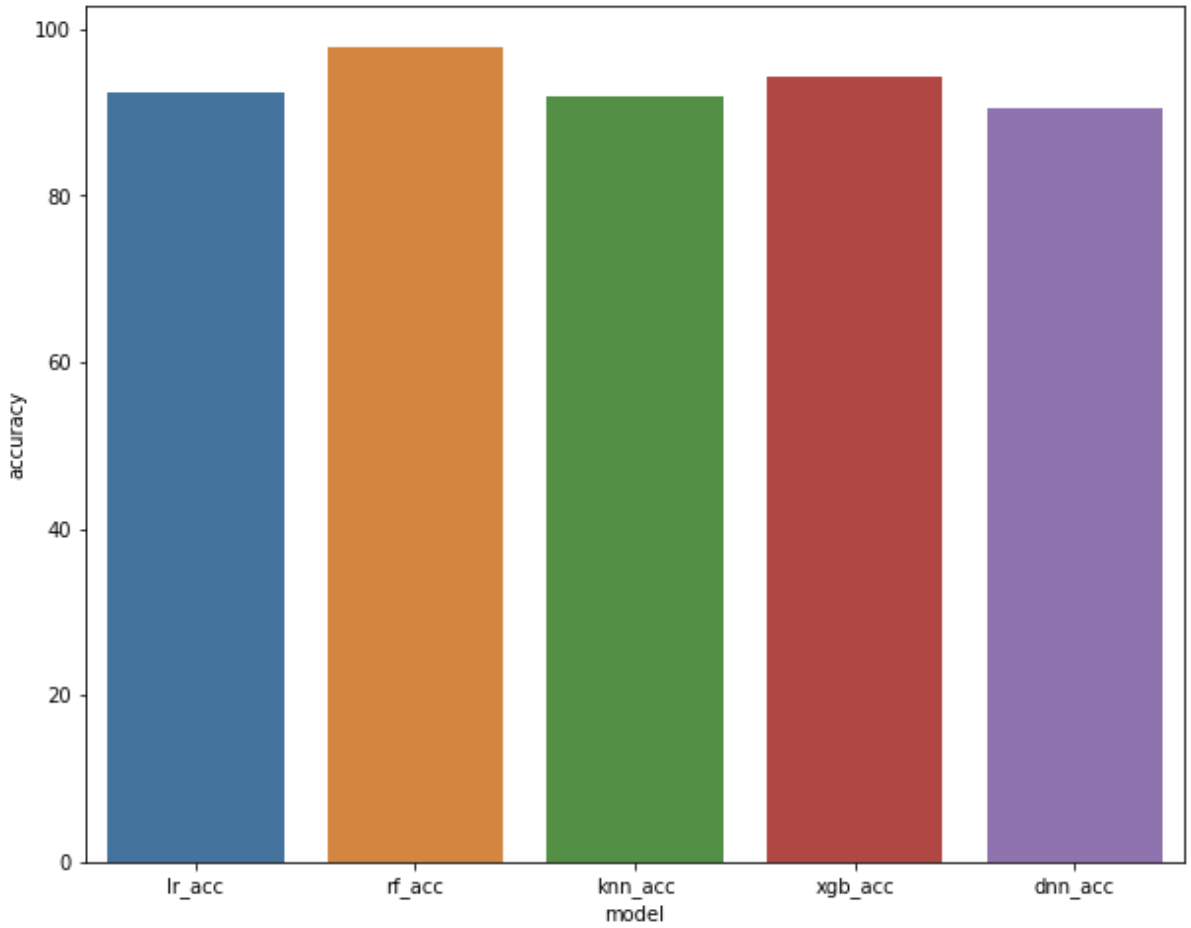
Saved model to disk

Comparing Models:

```
acc = {'model':['lr_acc','rf_acc','knn_acc','xgb_acc','dnn_acc'],'accuracy':[lr_acc,rf_acc,knn_acc,xgb_acc,dnn_acc]}

acc_df = pd.DataFrame(acc)

acc_df
```

	model	accuracy
0	lr_acc	92.280797
1	rf_acc	97.889071
2	knn_acc	91.972603
3	xgb_acc	94.211523
4	dnn_acc	90.503287

```
plt.figure(figsize=(10,8))

sns.barplot(x='model',y='accuracy',data=acc_df)

plt.savefig('plots/compared_models.png')

plt.show()
```



BUILDING HTML PAGES:

HTML CODES:

```
<!DOCTYPE html>

<html >

<head>

    <meta charset="UTF-8">

    <title>Walmart Store Sales Forecasting</title>

    <link href='https://fonts.googleapis.com/css?family=Pacifico'
rel='stylesheet' type='text/css'>
```

```
    <link href='https://fonts.googleapis.com/css?family=Arimo'
rel='stylesheet' type='text/css'>

    <link href='https://fonts.googleapis.com/css?family=Hind:300'
rel='stylesheet' type='text/css'>

    <link
href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300'
rel='stylesheet' type='text/css'>

    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">

</head>

<body>

<div class="login">

    <h1>Walmart Store Sales Forecasting</h1>

    <!-- Main Input For Receiving Query to our ML -->

    <form action="{{url_for('predict')}}"  method="POST">

        <label for="store">Store:</label>

        <input type="text" id="store" name="store" placeholder="Store"
required="required"/>



        <label for="dept">Dept:</label>

        <input type="text" id="dept" name="dept" placeholder="Dept"
required="required"/>



        <label for="date">Date:</label>

        <input type="date" id="date" name="date">



        <label for="isholiday">IsHoliday:</label>

        <select id="isholiday" name="isholiday">
```

```
            <option value="0">False</option>

            <option value="1">True</option>

        </select>

        <br>

        <br>

        <button type="submit" class="btn btn-primary btn-block
btn-large">Predict</button>

    </form>

    <br>

    {% if output %}

    <h3> Sales should be <b style="color:red;">$ {{ output }}</b>
</h3>

    {% endif %}

</div>

</body>

</html>
```

Building the python code in flask:

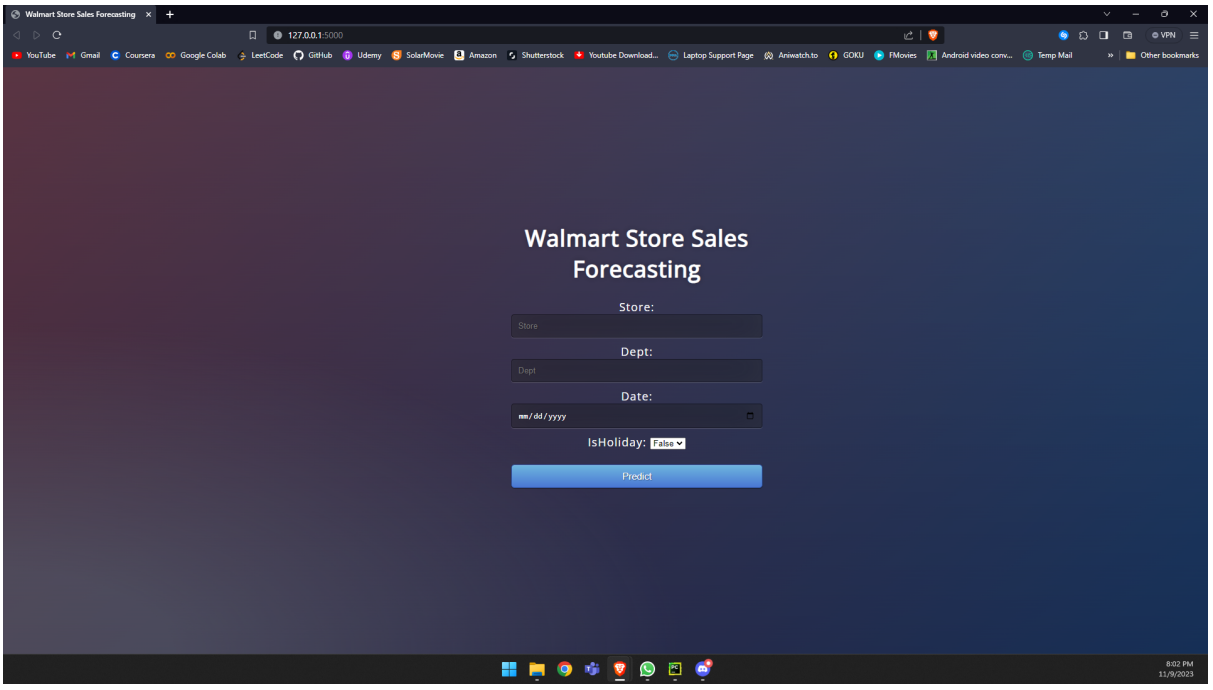
```
import pickle

import warnings

from sklearn.exceptions import InconsistentVersionWarning



import numpy as np

import pandas as pd

import datetime as dt

from flask import Flask, request, jsonify, render_template
```

```
app = Flask(__name__)

fet = pd.read_csv('all_features.csv')


warnings.simplefilter("error", InconsistentVersionWarning)

try:

    model = pickle.load(open('model.pkl', 'rb'), fix_imports=True,
encoding='bytes')

except InconsistentVersionWarning as w:

    print(w.original_sklearn_version)




@app.route('/')

def home():

    return render_template('index.html')




@app.route('/predict', methods=['POST'])

def predict():

    if request.method == 'POST':

        features = [x for x in request.form.values()]


        if features[3] == '0':

            features[3] = False

        else:
```

```
            features[3] = True



        df = fet[(fet['Store'] == int(features[0])) & (fet['IsHoliday']
== features[3]) & (fet['Date'] == features[2])]

        f_features = []

        d = dt.datetime.strptime(features[2], '%Y-%m-%d')

        c = 0

        if df['Type'][0] == 'C':

            c = 1

        else:

            c = 0



        if not features[3]:

            features[3] = 0

        else:

            features[3] = 1



        if df.shape[0] == 1:

            f_features.append(df['CPI'])

            f_features.append(d.date().day)

            f_features.append(int(features[1]))

            f_features.append(df['Fuel_Price'])

            f_features.append(features[3])

            f_features.append(d.date().month)

            f_features.append(df['Size'])
```

```
            f_features.append(int(features[0]))

            f_features.append(df['Temperature'])

            f_features.append(c)

            f_features.append(df['Unemployment'])

            f_features.append(d.date().year)



        final_features = [np.array(f_features)]

        output = model.predict(final_features)[0]

        return render_template('index.html', output=output)




if __name__ == "__main__":

    app.run(debug=True)
```

UI of application:



RUNNING APPLICATION:

OUTPUT: