Importing Modules

```
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 keras.wrappers.scikit_learn import KerasRegressor
```

Importing Datasets

```
#data = pd.read_csv('/content/drive/MyDrive/walmart-recruiting-store-sales-forecasting/train.csv')
#stores = pd.read_csv('/content/drive/MyDrive/walmart-recruiting-store-sales-forecasting/stores.csv')
#features = pd.read_csv('/content/drive/MyDrive/walmart-recruiting-store-sales-forecasting/features.csv')

data = pd.read_csv('datasets/train.csv')
stores = pd.read_csv('datasets/stores.csv')
features = pd.read_csv('datasets/features.csv')
```

Training Dataset

```
data.shape
```

```
(421570, 5)
```

```
data.tail()
```

```
        Store  Dept        Date  Weekly_Sales  IsHoliday
421565     45    98  2012-09-28        508.37      False
421566     45    98  2012-10-05        628.10      False
421567     45    98  2012-10-12       1061.02      False
421568     45    98  2012-10-19        760.01      False
421569     45    98  2012-10-26       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   Size    45 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  ...  CPI  Unemployment
IsHoliday
8185     45  2013-06-28        76.05  ...  NaN           NaN
False
8186     45  2013-07-05        77.50  ...  NaN           NaN
False
8187     45  2013-07-12        79.37  ...  NaN           NaN
False
8188     45  2013-07-19        82.84  ...  NaN           NaN
False
8189     45  2013-07-26        76.06  ...  NaN           NaN
False

[5 rows x 12 columns]
```

```
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     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 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(),inpl
ace=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)

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
```

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
```

```
data.head()
```

```
            Store  Dept       Date  ...         CPI Unemployment
IsHoliday
Date                                ...                         

2010-02-05      1     1 2010-02-05  ...  211.096358        8.106
False
2010-02-05     29     5 2010-02-05  ...  131.527903       10.064
False
2010-02-05     29     6 2010-02-05  ...  131.527903       10.064
False
2010-02-05     29     7 2010-02-05  ...  131.527903       10.064
False
2010-02-05     29     8 2010-02-05  ...  131.527903       10.064
False

[5 rows x 16 columns]
```

Splitting Date Column

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3:
FutureWarning: Series.dt.weekofyear and Series.dt.week have been
deprecated.  Please use Series.dt.isocalendar().week instead.
  This is separate from the ipykernel package so we can avoid doing
imports until
```

```
data.head()
```

```
            Store  Dept       Date  Weekly_Sales  ... IsHoliday  Year
Month  Week
Date                                              ...
2010-02-05      1     1 2010-02-05      24924.50  ...     False  2010
2     5
2010-02-05     29     5 2010-02-05      15552.08  ...     False  2010
2     5
2010-02-05     29     6 2010-02-05       3200.22  ...     False  2010
2     5
2010-02-05     29     7 2010-02-05      10820.05  ...     False  2010
2     5
2010-02-05     29     8 2010-02-05      20055.64  ...     False  2010
2     5

[5 rows x 19 columns]
```

Outlier Detection and Abnormalities

Outliers

```
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.head()
```

```
            Store  Dept       Date  ...          mean    median
std
Date                                 ...

2010-02-05      1     1 2010-02-05  ...  22513.322937  18535.48
9854.349032
2010-02-05      9    97 2010-02-05  ...    372.655556    371.05
290.954675
2010-02-05      9    85 2010-02-05  ...    876.629441    824.04
307.436056
2010-02-05      8    80 2010-02-05  ...   9188.915105   9161.97
756.223236
2010-02-05      9    55 2010-02-05  ...   8607.050490   7571.60
3874.176095

[5 rows x 24 columns]
```

```
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)
```

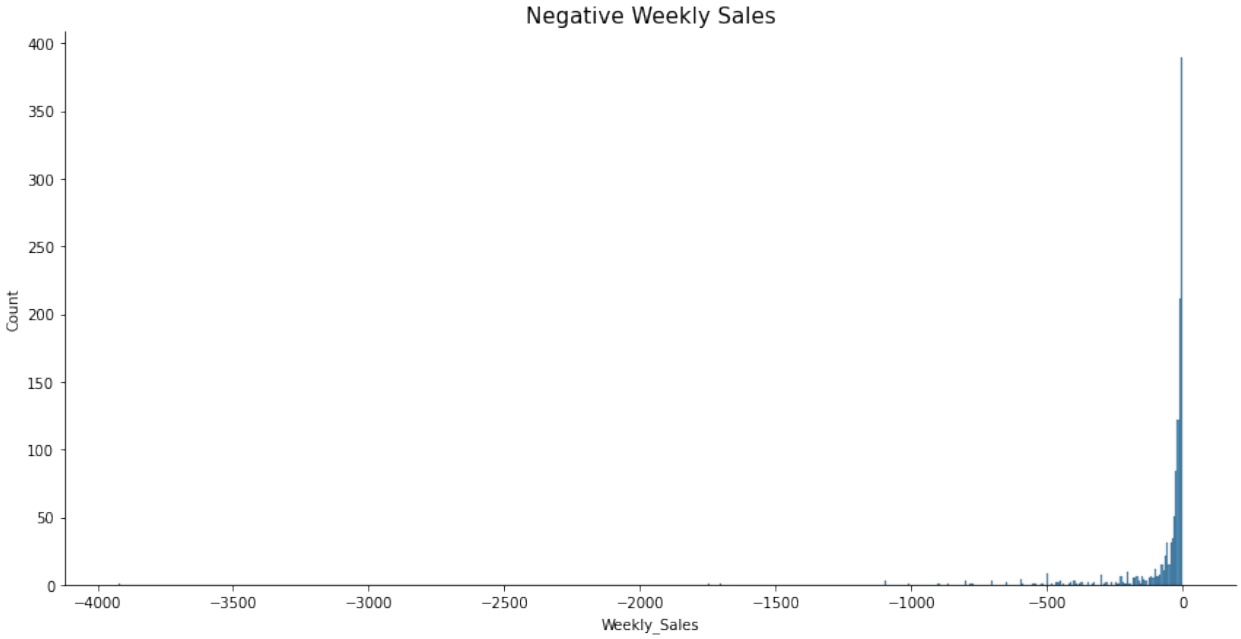
```
data = data[(np.abs(stats.zscore(data_numeric)) < 2.5).all(axis = 1)]
data.shape
```

```
(375438, 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('plots/negative_weekly_sales.png')
plt.show()
```




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

data.shape

(374247, 20)

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

data
```

```
            Store  Dept       Date  ...     median          std
Total_MarkDown
Date                                 ...
2010-02-05      1     1 2010-02-05  ...  18535.480  9854.349032
0.00
2010-02-05      9    97 2010-02-05  ...    371.050   290.954675
0.00
2010-02-05      9    85 2010-02-05  ...    824.040   307.436056
0.00
2010-02-05      8    80 2010-02-05  ...   9161.970   756.223236
0.00
2010-02-05      9    55 2010-02-05  ...   7571.600  3874.176095
0.00
...           ...   ...        ...  ...        ...          ...
...
```

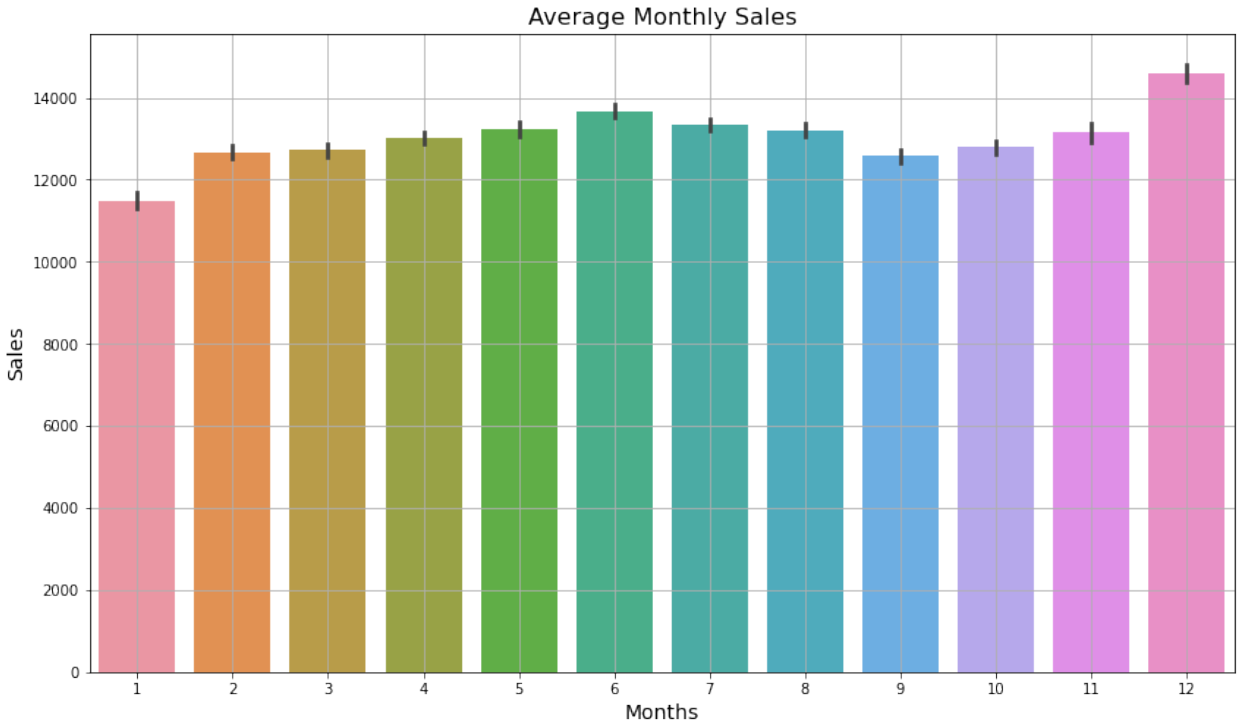
```
2012-10-26      2    26 2012-10-26  ...   8762.990  2825.107609
9678.80
2012-10-26     38    23 2012-10-26  ...     31.365    34.065601
502.88
2012-10-26     27     6 2012-10-26  ...   6798.780  5178.928257
10969.27
2012-10-26     36    40 2012-10-26  ...  10329.180  1043.930131
1260.55
2012-10-26     45    98 2012-10-26  ...    619.410   371.286705
5247.26

[374247 rows x 20 columns]
```

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

Data Visuallizations

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('plots/avg_monthly_sales.png')
plt.grid()
```

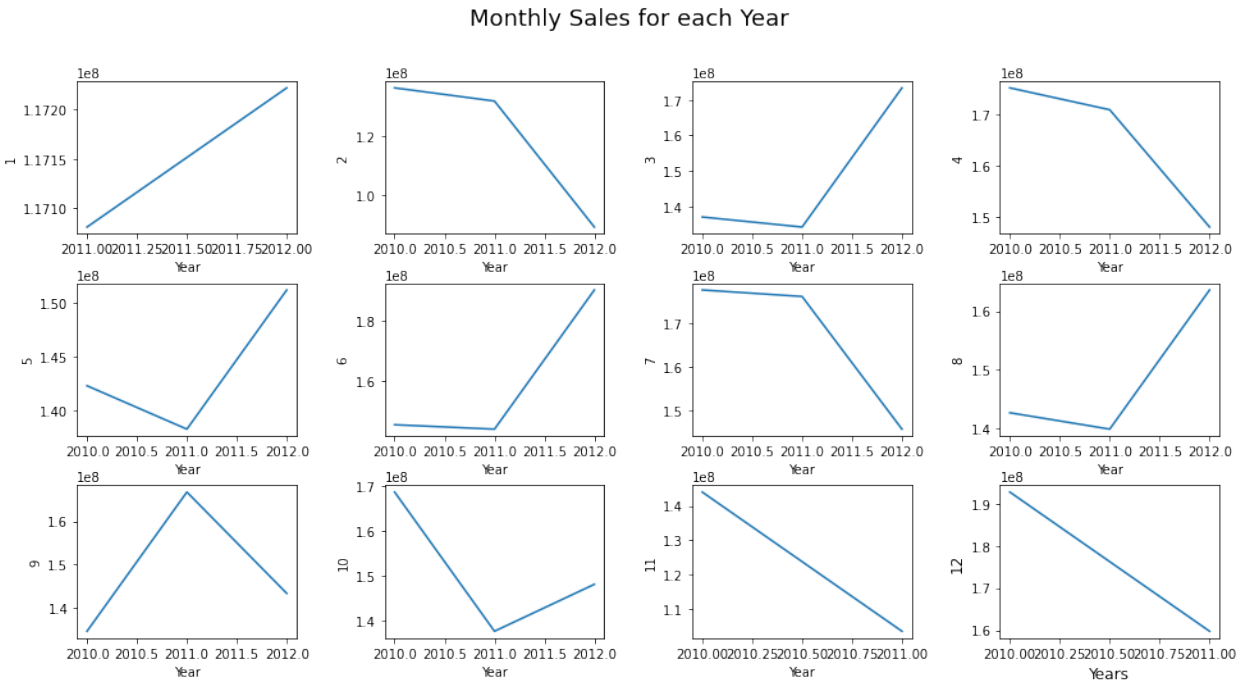


Monthly Sales for Each Year

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

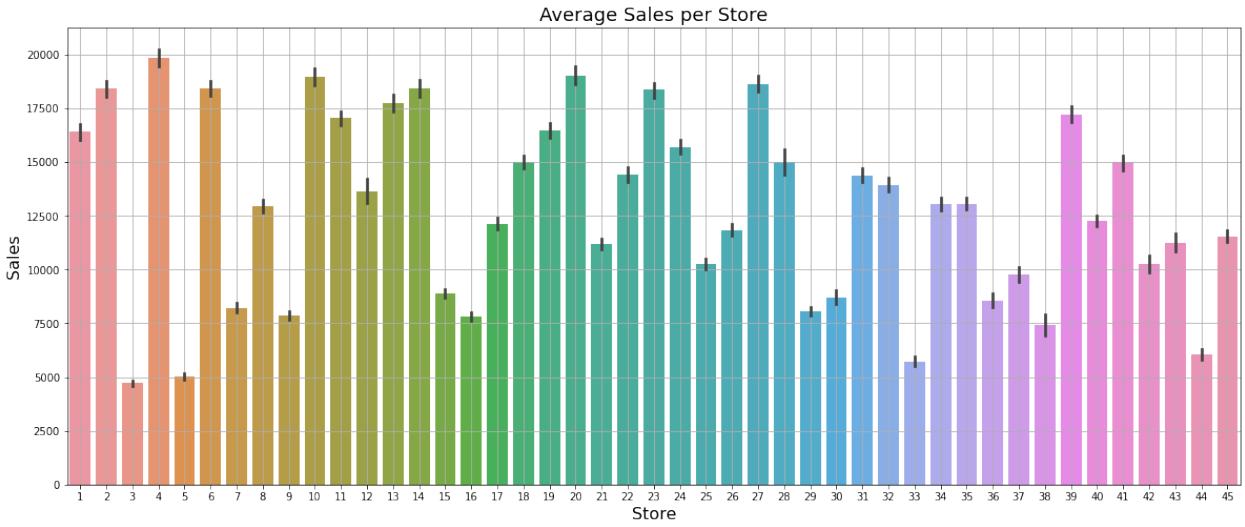
```
Month            1             2   ...            11            12
Year                                ...
2010           NaN  1.365986e+08   ...  1.440445e+08  1.927286e+08
2011  1.170809e+08  1.320987e+08   ...  1.034907e+08  1.597524e+08
2012  1.172222e+08  8.915290e+07   ...           NaN           NaN

[3 rows x 12 columns]
```

```
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('plots/monthly_sales_every_year.png')
plt.show()
```

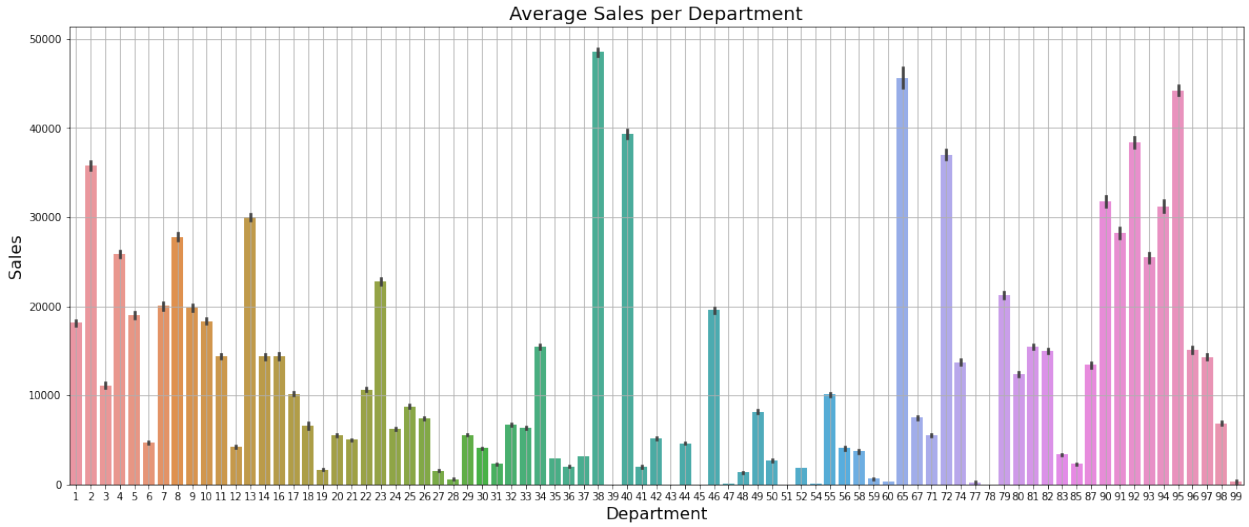


Average Weekly Sales Store wise

```
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('plots/avg_sales_store.png')
plt.show()
```



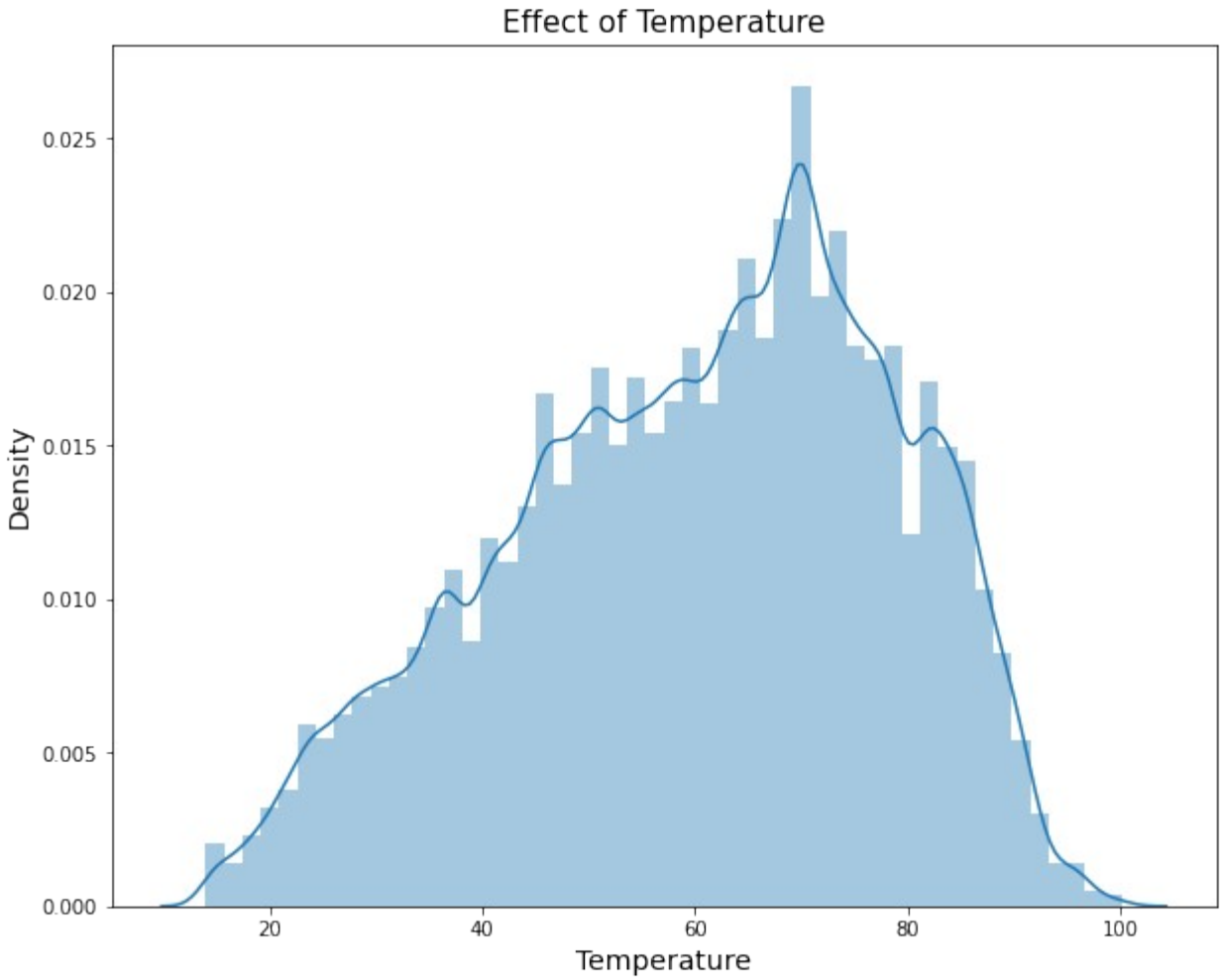
```
plt.figure(figsize=(20,8))
sns.barplot(x='Dept',y='Weekly_Sales',data=data)
plt.grid()
plt.title('Average Sales per Department', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Department', fontsize=16)
plt.savefig('plots/avg_sales_dept.png')
plt.show()
```



Sales Vs Temperature

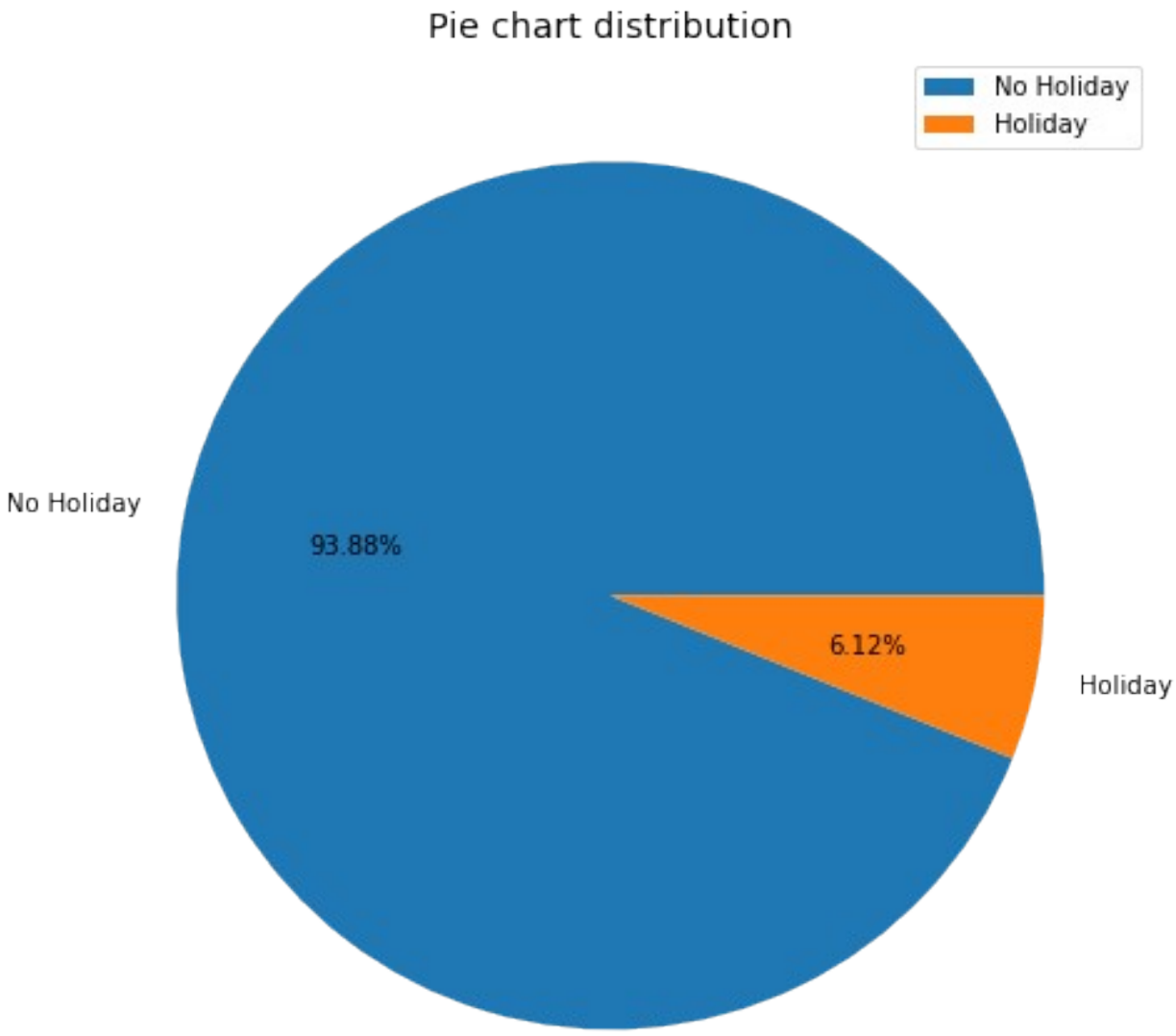
```
plt.figure(figsize=(10,8))
sns.distplot(data['Temperature'])
plt.title('Effect of Temperature',fontsize=15)
plt.xlabel('Temperature',fontsize=14)
plt.ylabel('Density',fontsize=14)
plt.savefig('plots/effect_of_temp.png')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



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('plots/holiday_distribution.png')
plt.show()
```



#Time Series Decompose

```
sm.tsa.seasonal_decompose(data['Weekly_Sales'].resample('MS').mean(),
model='additive').plot()
plt.savefig('plots/seasonal_decompose.png')
plt.show()
```

One-hot-encoding

```
cat_col = ['Store','Dept','Type']
data_cat = data[cat_col].copy()

data_cat.tail()
```

```
            Store  Dept Type
Date
2012-10-26      2    26    A
2012-10-26     38    23    C
2012-10-26     27     6    A
2012-10-26     36    40    A
2012-10-26     45    98    B
```

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

data_cat.head()
```

```
            Store_1  Store_2  Store_3  Store_4  ...  Dept_99  Type_A
Type_B  Type_C
Date                                           ...

2010-02-05        1        0        0        0  ...        0       1
0       0
2010-02-05        0        0        0        0  ...        0       0
```

```
1       0
2010-02-05        0        0        0        0  ...        0       0
1       0
2010-02-05        0        0        0        0  ...        0       1
0       0
2010-02-05        0        0        0        0  ...        0       0
1       0

[5 rows x 129 columns]
```

```
data.shape
```

```
(374247, 20)
```

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

```
data.shape
```

```
(374247, 149)
```

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

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

```
data.shape
```

```
(374247, 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()
```

```
            Weekly_Sales    Size  Temperature  ...  Type_A  Type_B
Type_C
Date                                           ...

2010-02-05      24924.50  151315        42.31  ...       1       0
0
```

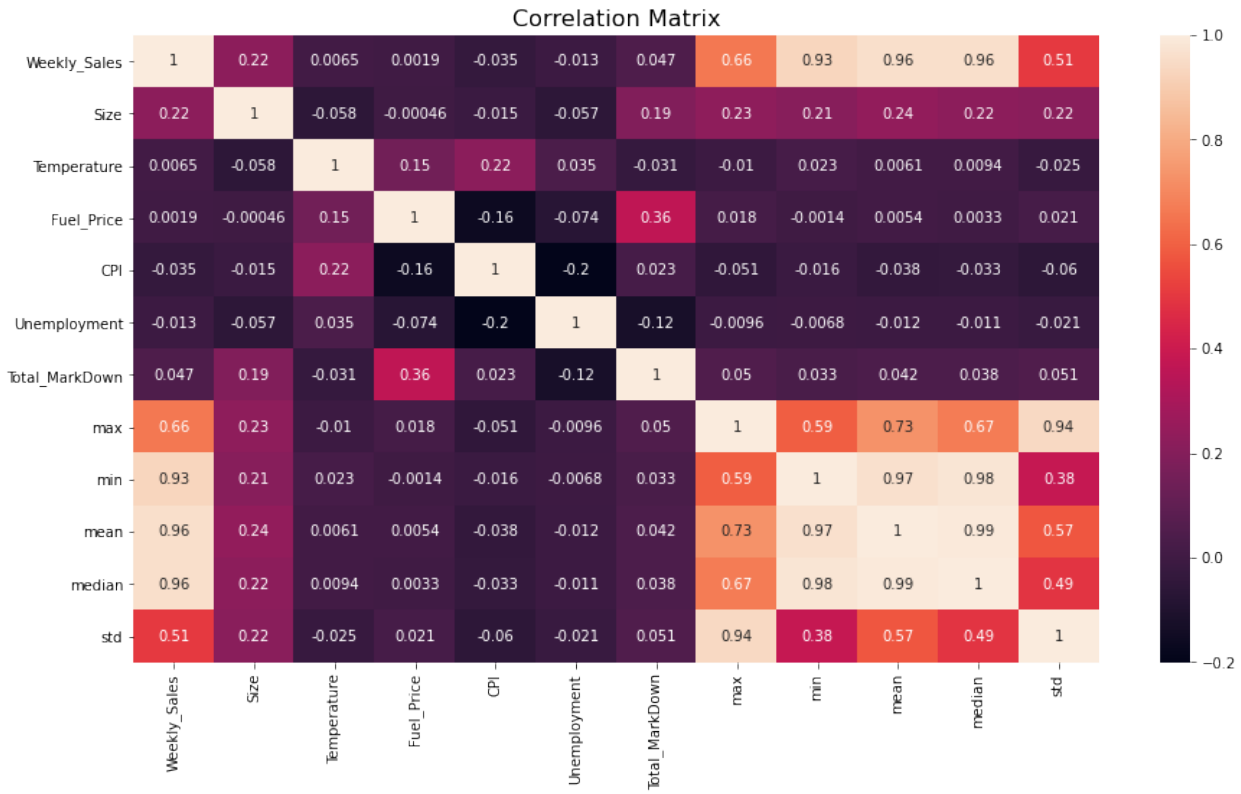
```
2010-02-05        668.48  125833        38.01  ...       0       1
0
2010-02-05        693.87  125833        38.01  ...       0       1
0
2010-02-05       8654.60  155078        34.14  ...       1       0
0
2010-02-05      11123.56  125833        38.01  ...       0       1
0

[5 rows x 145 columns]
```

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

data.head()
```

```
             Weekly_Sales      Size  Temperature  ...  Type_A  Type_B
Type_C
Date                                            ...

2010-02-05      0.342576  0.630267     0.328495  ...       1       0
0
2010-02-05      0.009188  0.492338     0.278565  ...       0       1
0
2010-02-05      0.009537  0.492338     0.278565  ...       0       1
0
2010-02-05      0.118953  0.650636     0.233627  ...       1       0
0
2010-02-05      0.152888  0.492338     0.278565  ...       0       1
0

[5 rows x 145 columns]
```

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('plots/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=Tru
e),param_grid,cv=5)
tree.fit(data_train[feature_col],data_train['Weekly_Sales'])
'''

#tree.best_params_
```

```
radm_clf = RandomForestRegressor(oob_score=True,n_estimators=23)
radm_clf.fit(data[feature_col], data['Weekly_Sales'])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/
_forest.py:815: UserWarning: Some inputs do not have OOB scores. This
probably means too few trees were used to compute any reliable oob
estimates.
  warn("Some inputs do not have OOB scores. "
```

```
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=23, n_jobs=None, oob_score=True,
                      random_state=None, verbose=0, warm_start=False)
```

```
pkl_filename = "./models/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       median  4.964671e-01
1      2         mean  4.317588e-01
2      3         Week  1.967699e-02
3      4  Temperature  8.925910e-03
4      5          max  6.038049e-03
..   ...          ...           ...
139  140      Dept_51  1.449926e-10
140  141      Dept_45  4.821496e-11
141  142      Dept_43  0.000000e+00
142  143      Dept_78  0.000000e+00
143  144      Dept_39  0.000000e+00
```

```
[144 rows x 3 columns]
```

```
x=feature_rank.loc[0:22,['feature']]
x=x['feature'].tolist()
print(x)
```

```
['median', 'mean', 'Week', 'Temperature', 'max', 'CPI', 'Fuel_Price',
'min', 'Unemployment', 'std', 'Month', 'Total_MarkDown', 'Dept_16',
'Dept_18', 'IsHoliday', 'Dept_3', 'Size', 'Dept_11', 'Year', 'Dept_9',
'Dept_1', 'Dept_5', 'Dept_56']
```

```
X = data[x]
Y = data['Weekly_Sales']

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

data
```

```
              median      mean  Week  ...  Dept_5  Dept_56
Weekly_Sales
Date                                  ...
2010-02-05  0.173215  0.208157     5  ...       0        0
0.342576
2010-02-05  0.004767  0.004499     5  ...       0        0
0.009188
2010-02-05  0.008968  0.009135     5  ...       0        0
0.009537
2010-02-05  0.086290  0.085594     5  ...       0        0
0.118953
2010-02-05  0.071542  0.080242     5  ...       0        0
0.152888
...              ...       ...   ...  ...     ...      ...             .
..
2012-10-26  0.082590  0.087055    43  ...       0        0
0.127259
2012-10-26  0.001617  0.001419    43  ...       0        0
0.000730
2012-10-26  0.064375  0.072181    43  ...       0        0
0.073391
2012-10-26  0.097114  0.098037    43  ...       0        0
0.140418
2012-10-26  0.007070  0.006234    43  ...       0        0
0.014800

[374247 rows x 24 columns]
```

```
data.to_csv('./datasets/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)
```

Linear Regression Model

```
lr = LinearRegression(normalize=False)
lr.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

```
lr_acc = lr.score(X_test,y_test)*100
print("Linear Regressor Accuracy - ",lr_acc)
```

```
Linear Regressor Accuracy -  92.28079698115758
```

```
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))
```

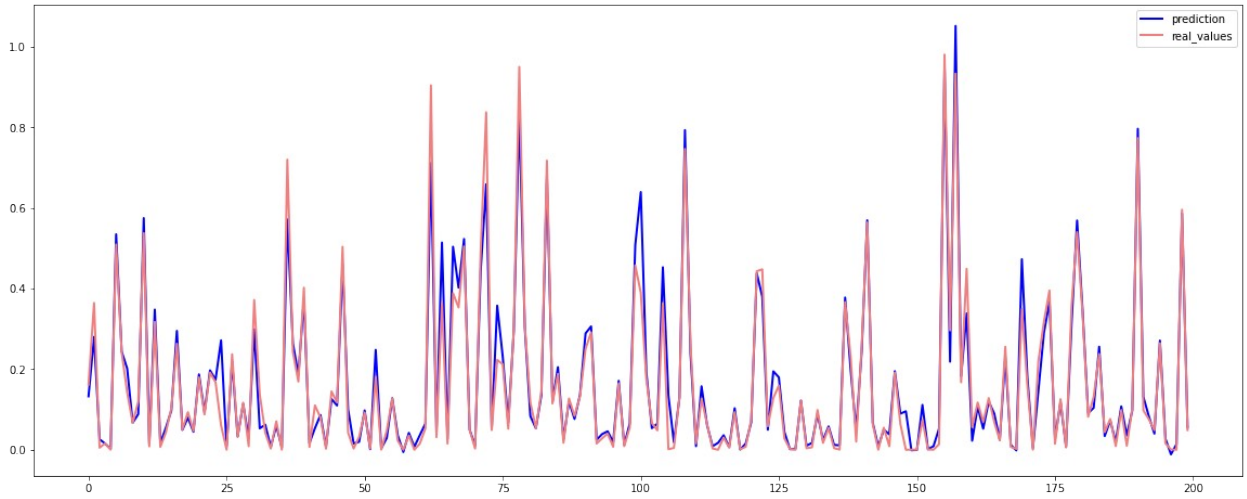
```
MAE 0.0300577149215146
MSE 0.0034851431916206577
RMSE 0.059035101351828455
R2 0.9228079866096734
```

```
lr_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lr_df.to_csv('./predictions/lr_real_pred.csv')
lr_df
```

```
              Actual  Predicted
Date
2011-08-05  0.161661   0.132555
2010-07-09  0.364278   0.280242
2011-07-01  0.005003   0.026085
2012-01-06  0.015856   0.015369
2011-08-26  0.000318   0.002072
...              ...        ...
2011-01-28  0.169068   0.236392
2010-08-20  0.252860   0.235591
2010-11-26  0.265617   0.321839
2010-03-12  0.008865   0.013607
```

```
2010-02-12  0.230510   0.235435

[74850 rows x 2 columns]
```

```
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('plots/lr_real_pred.png')
plt.show()
```



Saving trained model

```
pkl_filename = "./models/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")
```

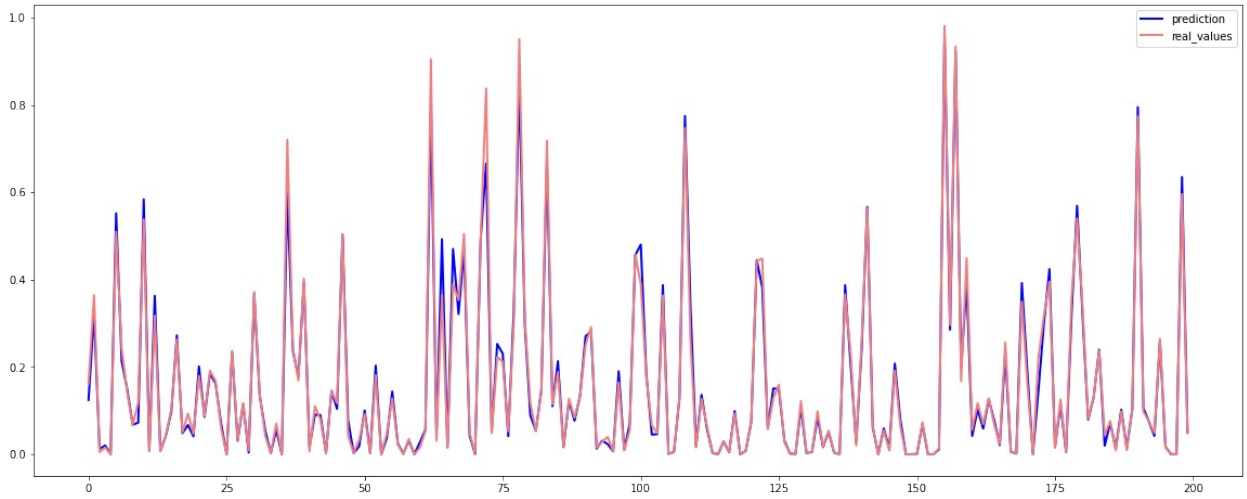
```
Saved model to disk
```

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)

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 0.015522536897538632
MSE 0.0009530632336469744
RMSE 0.030871722233250517
R2 0.9788909900125646

rf_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
rf_df.to_csv('./predictions/rf_real_pred.csv')
rf_df

              Actual  Predicted
Date
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 x 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
```

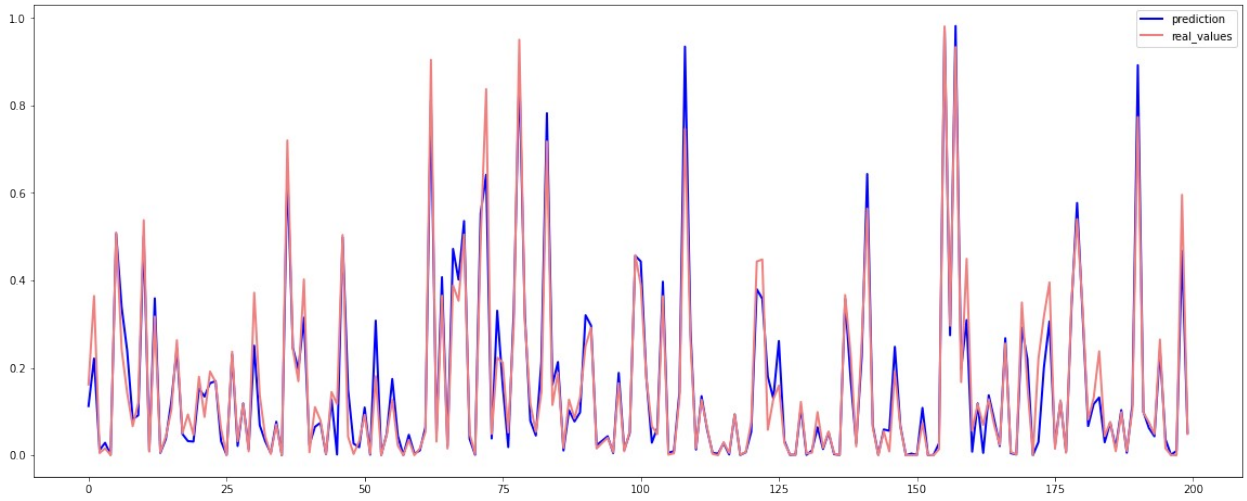
```
y_pred = knn.predict(X_test)

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
```

```
              Actual  Predicted
Date
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 x 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)
```

```
[10:05:22] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
```

```
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
```

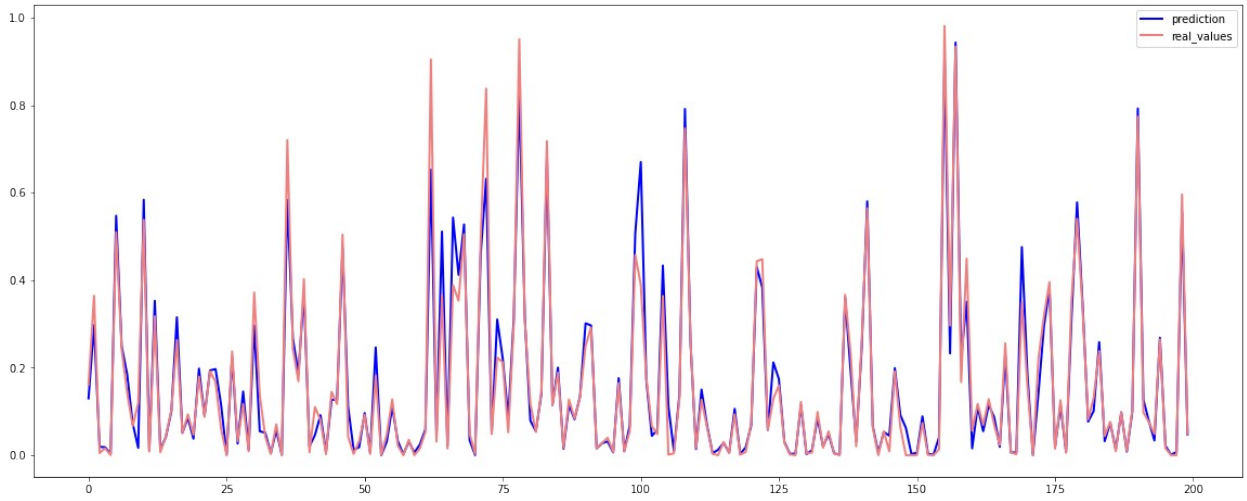
```
y_pred = xgbr.predict(X_test)

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
```

```
              Actual  Predicted
Date
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 x 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)
```

```
Epoch 1/100
48/48 [==============================] - 1s 16ms/step - loss: 2.1759 -
val_loss: 0.1723
Epoch 2/100
```

```
48/48 [==============================] - 0s 9ms/step - loss: 0.1516 -
val_loss: 0.1416
Epoch 3/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1533 -
val_loss: 0.1630
Epoch 4/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1878 -
val_loss: 0.1624
Epoch 5/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1763 -
val_loss: 0.1642
Epoch 6/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1693 -
val_loss: 0.1663
Epoch 7/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1625 -
val_loss: 0.1682
Epoch 8/100
48/48 [==============================] - 0s 8ms/step - loss: 0.1570 -
val_loss: 0.1213
Epoch 9/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1202 -
val_loss: 0.1276
Epoch 10/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1542 -
val_loss: 0.1131
Epoch 11/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1427 -
val_loss: 0.1106
Epoch 12/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1344 -
val_loss: 0.1099
Epoch 13/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1380 -
val_loss: 0.1664
Epoch 14/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1880 -
val_loss: 0.5794
Epoch 15/100
48/48 [==============================] - 0s 10ms/step - loss: 0.3423 -
val_loss: 0.1388
Epoch 16/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1292 -
val_loss: 0.1596
Epoch 17/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1268 -
val_loss: 0.1295
Epoch 18/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1304 -
```

```
val_loss: 0.1333
Epoch 19/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1295 -
val_loss: 0.2775
Epoch 20/100
48/48 [==============================] - 0s 10ms/step - loss: 0.2301 -
val_loss: 0.2046
Epoch 21/100
48/48 [==============================] - 0s 9ms/step - loss: 0.2367 -
val_loss: 0.3250
Epoch 22/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1744 -
val_loss: 0.1674
Epoch 23/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1615 -
val_loss: 0.1243
Epoch 24/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1045 -
val_loss: 0.1121
Epoch 25/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1015 -
val_loss: 0.1421
Epoch 26/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1001 -
val_loss: 0.0917
Epoch 27/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0973 -
val_loss: 0.1618
Epoch 28/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1522 -
val_loss: 0.5053
Epoch 29/100
48/48 [==============================] - 0s 9ms/step - loss: 0.2153 -
val_loss: 0.0903
Epoch 30/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0958 -
val_loss: 0.1895
Epoch 31/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1977 -
val_loss: 0.0632
Epoch 32/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0990 -
val_loss: 0.0604
Epoch 33/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1358 -
val_loss: 0.0957
Epoch 34/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0804 -
val_loss: 0.0529
```

```
Epoch 35/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1231 -
val_loss: 0.1682
Epoch 36/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1405 -
val_loss: 0.1330
Epoch 37/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1361 -
val_loss: 0.1381
Epoch 38/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1079 -
val_loss: 0.0541
Epoch 39/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1234 -
val_loss: 0.0700
Epoch 40/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0697 -
val_loss: 0.0565
Epoch 41/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0859 -
val_loss: 0.0647
Epoch 42/100
48/48 [==============================] - 1s 10ms/step - loss: 0.0870 -
val_loss: 0.0888
Epoch 43/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0679 -
val_loss: 0.0434
Epoch 44/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0878 -
val_loss: 0.2287
Epoch 45/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1899 -
val_loss: 0.0502
Epoch 46/100
48/48 [==============================] - 1s 11ms/step - loss: 0.1181 -
val_loss: 0.0784
Epoch 47/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0654 -
val_loss: 0.0465
Epoch 48/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0715 -
val_loss: 0.1129
Epoch 49/100
48/48 [==============================] - 1s 10ms/step - loss: 0.0885 -
val_loss: 0.2324
Epoch 50/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1350 -
val_loss: 0.0591
Epoch 51/100
```

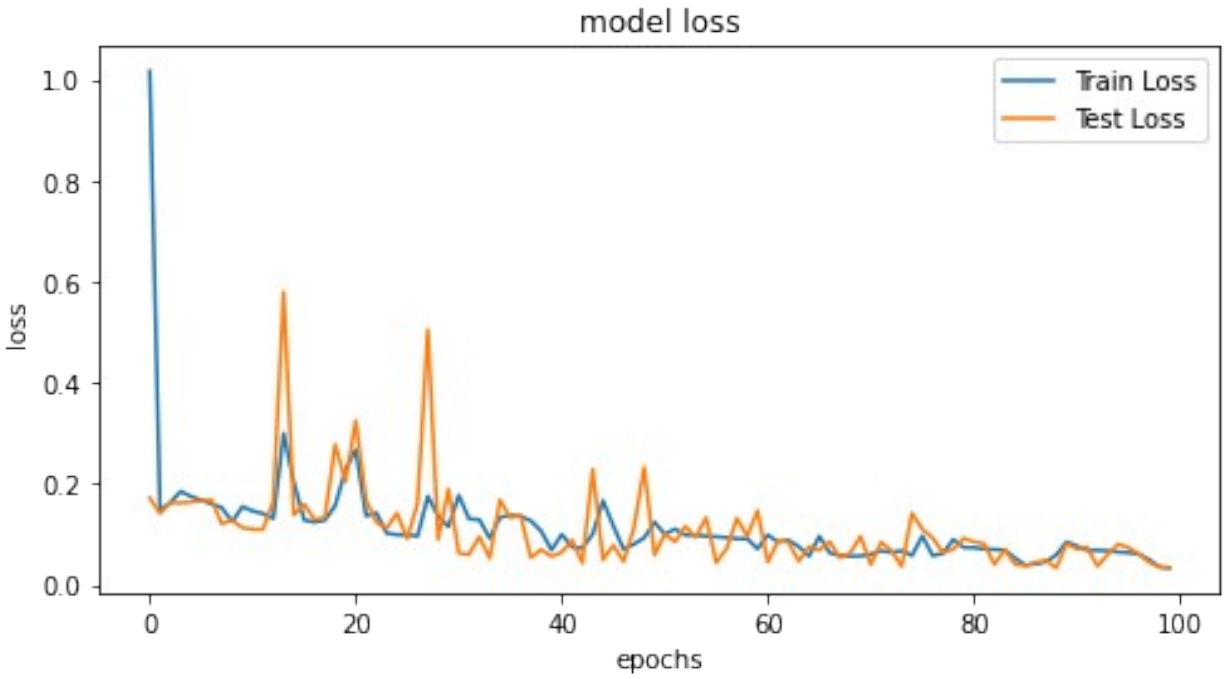
```
48/48 [==============================] - 0s 10ms/step - loss: 0.0944 -
val_loss: 0.1007
Epoch 52/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1255 -
val_loss: 0.0858
Epoch 53/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0987 -
val_loss: 0.1160
Epoch 54/100
48/48 [==============================] - 0s 9ms/step - loss: 0.1008 -
val_loss: 0.0941
Epoch 55/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0954 -
val_loss: 0.1333
Epoch 56/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0961 -
val_loss: 0.0439
Epoch 57/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0927 -
val_loss: 0.0720
Epoch 58/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0910 -
val_loss: 0.1318
Epoch 59/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0949 -
val_loss: 0.0959
Epoch 60/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0639 -
val_loss: 0.1470
Epoch 61/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1141 -
val_loss: 0.0455
Epoch 62/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0839 -
val_loss: 0.0875
Epoch 63/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0862 -
val_loss: 0.0844
Epoch 64/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0939 -
val_loss: 0.0472
Epoch 65/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0544 -
val_loss: 0.0757
Epoch 66/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0887 -
val_loss: 0.0690
Epoch 67/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0628 -
```

```
val_loss: 0.0858
Epoch 68/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0629 -
val_loss: 0.0538
Epoch 69/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0588 -
val_loss: 0.0633
Epoch 70/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0514 -
val_loss: 0.0961
Epoch 71/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0649 -
val_loss: 0.0409
Epoch 72/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0683 -
val_loss: 0.0844
Epoch 73/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0664 -
val_loss: 0.0680
Epoch 74/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0644 -
val_loss: 0.0372
Epoch 75/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0495 -
val_loss: 0.1424
Epoch 76/100
48/48 [==============================] - 0s 10ms/step - loss: 0.1059 -
val_loss: 0.1102
Epoch 77/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0649 -
val_loss: 0.0932
Epoch 78/100
48/48 [==============================] - 1s 11ms/step - loss: 0.0659 -
val_loss: 0.0638
Epoch 79/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0801 -
val_loss: 0.0707
Epoch 80/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0749 -
val_loss: 0.0925
Epoch 81/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0758 -
val_loss: 0.0853
Epoch 82/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0719 -
val_loss: 0.0818
Epoch 83/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0705 -
val_loss: 0.0408
```

```
Epoch 84/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0679 -
val_loss: 0.0694
Epoch 85/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0634 -
val_loss: 0.0413
Epoch 86/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0409 -
val_loss: 0.0387
Epoch 87/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0377 -
val_loss: 0.0444
Epoch 88/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0426 -
val_loss: 0.0489
Epoch 89/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0581 -
val_loss: 0.0336
Epoch 90/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0876 -
val_loss: 0.0813
Epoch 91/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0771 -
val_loss: 0.0714
Epoch 92/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0677 -
val_loss: 0.0742
Epoch 93/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0686 -
val_loss: 0.0377
Epoch 94/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0668 -
val_loss: 0.0606
Epoch 95/100
48/48 [==============================] - 0s 8ms/step - loss: 0.0663 -
val_loss: 0.0809
Epoch 96/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0655 -
val_loss: 0.0740
Epoch 97/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0629 -
val_loss: 0.0616
Epoch 98/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0539 -
val_loss: 0.0453
Epoch 99/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0373 -
val_loss: 0.0359
Epoch 100/100
```

```
48/48 [==============================] - 0s 9ms/step - loss: 0.0346 -
val_loss: 0.0335
```

```
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
```

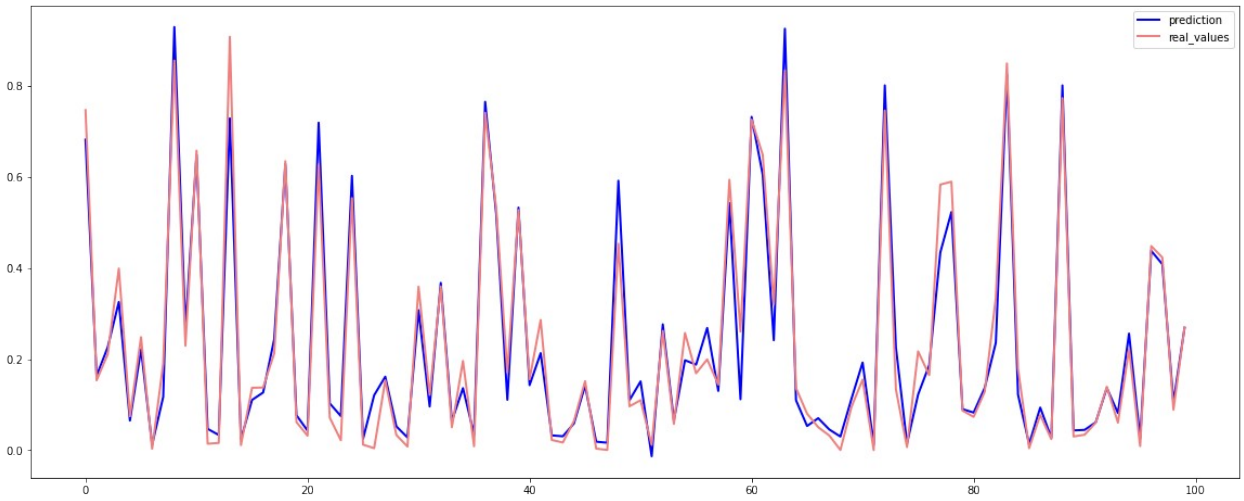
Date	Actual	Predicted
2011-08-05	0.161661	0.124761
2010-07-09	0.364278	0.289382
2011-07-01	0.005003	0.034531
2012-01-06	0.015856	0.024284
2011-08-26	0.000318	0.015496
...	...	...
2011-01-28	0.169068	0.233344
2010-08-20	0.252860	0.236093
2010-11-26	0.265617	0.342386
2010-03-12	0.008865	0.023427
2010-02-12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

```
[74850 rows x 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")
```

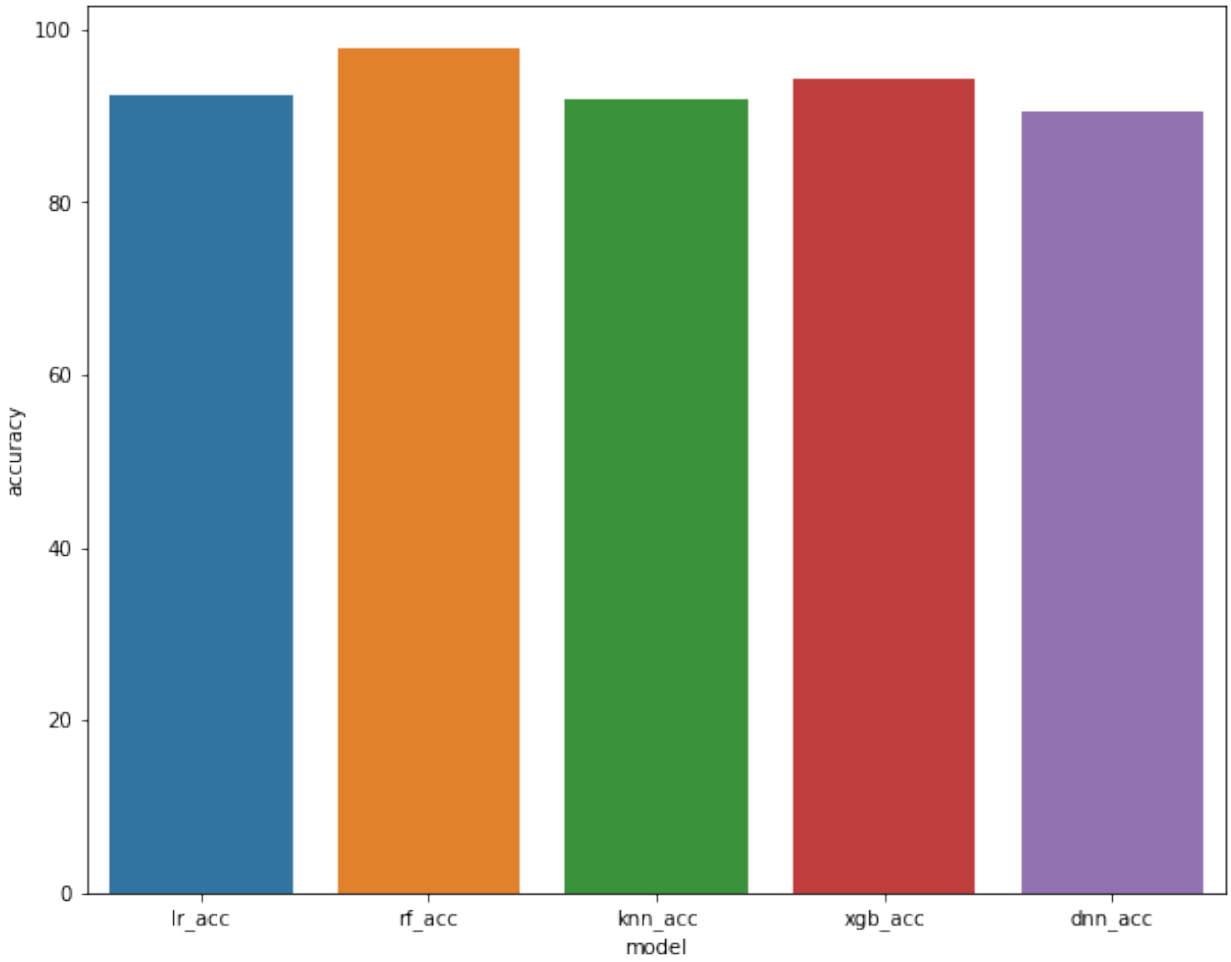
```
Saved model to disk
```

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()
```


100
80
60
40
20
0
accuracy
lr_acc
rf_acc
knn_acc
xgb_acc
dnn_acc
model