Import statements ¶

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# For prediction models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM,Input
from tensorflow.keras.callbacks import EarlyStopping
# Metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, mean_absolute_percentage_error
#plt.style.use("five thirty eight")
import seaborn as sns
pal=sns.color_palette()
pal_list=list(pal)
# For Images
import torch
import datetime as dt
import os
import tqdm
```

Import datasets to Python env

[103]:
```
items=pd.read_csv(r'C:\Users\HP\Capstone Project\items.csv')
resturants=pd.read_csv(r'C:\Users\HP\Capstone Project\resturants.csv')
sales=pd.read_csv(r'C:\Users\HP\Capstone Project\sales.csv')
```

Examine datasets

[104]:
```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109600 entries, 0 to 109599
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   date        109600 non-null  object 
 1   item_id     109600 non-null  int64  
 2   price       109600 non-null  float64
 3   item_count  109600 non-null  float64
dtypes: float64(2), int64(1), object(1)
memory usage: 3.3+ MB
```

[105]:
```
sales.date=pd.to_datetime(sales.date)
items.head()
```

[105]:

	id	store_id	name	kcal	cost
0	1	4	Chocolate Cake	554	6.71
1	2	4	Breaded Fish with Vegetables Meal	772	15.09
2	3	1	Sweet Fruity Cake	931	29.22
3	4	1	Amazing Steak Dinner with Rolls	763	26.42
4	5	5	Milk Cake	583	6.07

Merge datasets into a single dataset that includes the date, item id, price, item count,

item names, kcal values, store id, and store name

[106]:
```
merged_data = pd.merge (sales,items, left_on="item_id",right_on="id",how="left")
merged_data.head()
```

[106]:

	date	item_id	price	item_count	id	store_id	name	kcal	cost
0	2019-01-01	3	29.22	2.0	3	1	Sweet Fruity Cake	931	29.22
1	2019-01-01	4	26.42	22.0	4	1	Amazing Steak Dinner with Rolls	763	26.42
2	2019-01-01	12	4.87	7.0	12	1	Fantastic Sweet Cola	478	4.87
3	2019-01-01	13	4.18	12.0	13	1	Sweet Frozen Soft Drink	490	4.18
4	2019-01-01	16	3.21	136.0	16	1	Frozen Milky Smoothy	284	3.21

[107]:
```
data=pd.merge(merged_data,resturants, left_on='store_id',right_on='id')
```

[108]:
```
data.head(2)
```

[108]:

	date	item_id	price	item_count	id_x	store_id	name_x	kcal	cost	id_y	name_y
0	2019-01-01	3	29.22	2.0	3	1	Sweet Fruity Cake	931	29.22	1	Bob's Diner
1	2019-01-01	4	26.42	22.0	4	1	Amazing Steak Dinner with Rolls	763	26.42	1	Bob's Diner

[11]:
```
(data.item_id == data.id_x).all()
(data.store_id == data.id_y).all()
```

[11]:
```
True
```

[109]:
```
data.rename(columns = {"name_x" : "item_name", "name_y" : "restaurant_name"}, inplace=True)
```

[110]:
```
data.head(2)
```

[110]:

	date	item_id	price	item_count	id_x	store_id	item_name	kcal	cost	id_y	restaurant_name
0	2019-01-01	3	29.22	2.0	3	1	Sweet Fruity Cake	931	29.22	1	Bob's Diner
1	2019-01-01	4	26.42	22.0	4	1	Amazing Steak Dinner with Rolls	763	26.42	1	Bob's Diner

[111]:
```
data.drop(columns = ["id_x","id_y","cost"], inplace=True)
```

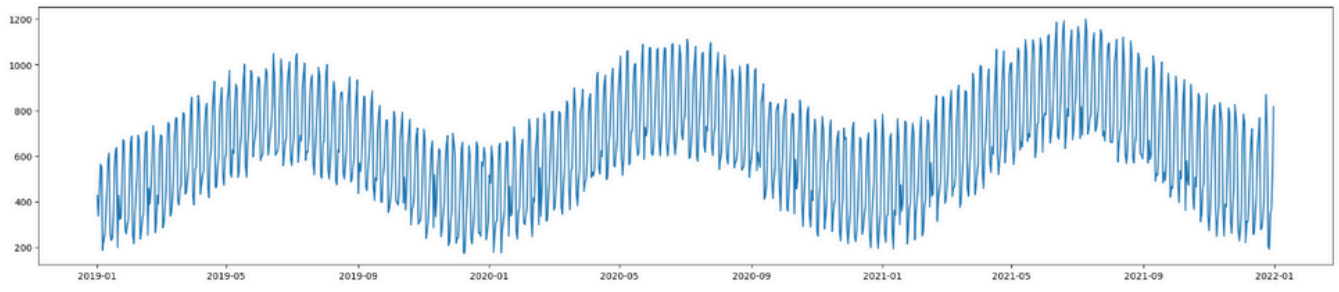
[112]:
```
sales.head(2)
```

[112]:

	date	item_id	price	item_count
0	2019-01-01	3	29.22	2.0
1	2019-01-01	4	26.42	22.0

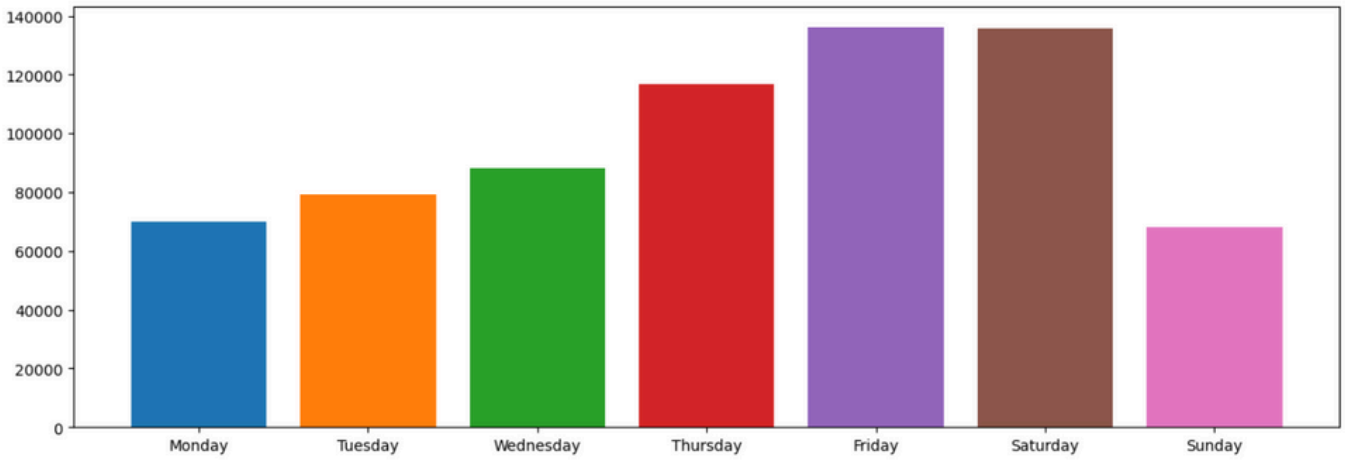
Study overall sales date-wise to understand the sales pattern

[16]:
```
grpd = sales.groupby(['date'])[['item_count']].sum()
plt.figure(figsize=(25,5))
plt.plot(grpd)
plt.show()
```



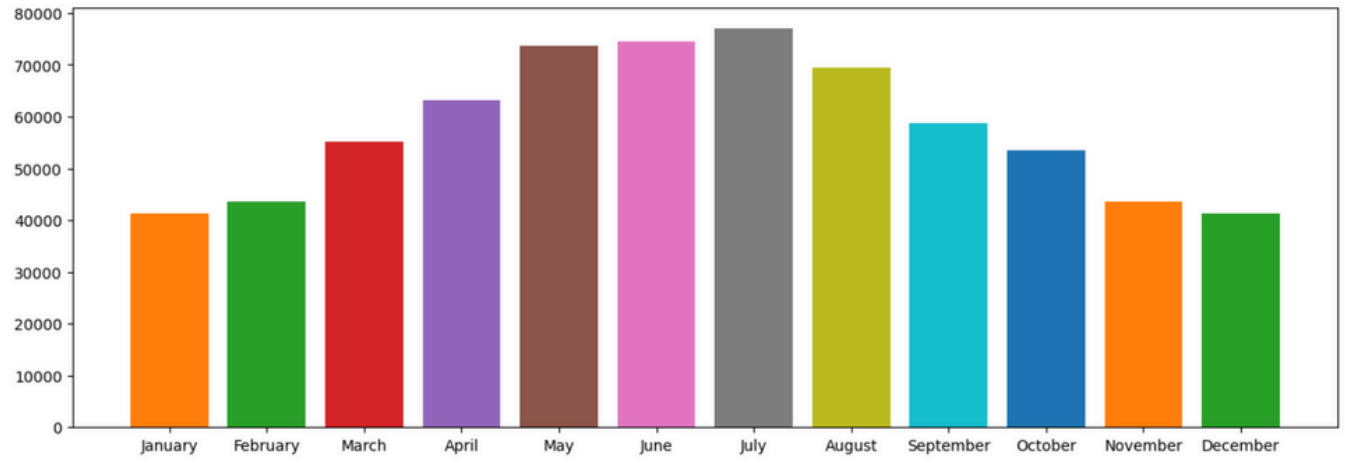
Find our how the sale fluctuates different days of the week

[113]:
```
import calendar
data['weekday'] = data.date.dt.day_name()
day_names = list(calendar.day_name)
grpd = data.groupby(['weekday'])[['item_count']].sum()
grpd=grpd.loc[day_names].squeeze()
plt.figure(figsize=(15,5))
plt.bar(height=grpd.values,x=grpd.index,color=pal)
plt.show()
```



Fluctuations of sales month wise

[114]:
```
import calendar
data['month'] = data.date.dt.month_name()
months = list(calendar.month_name)
grpd = data.groupby(['month'])[['item_count']].sum()
grpd = grpd.reindex(months).squeeze()
plt.figure(figsize=(15,5))
plt.bar(height=grpd.values,x=grpd.index,color=pal)
plt.show()
```



Identify the distributions of sales across different quarters averaged over years. Identify any noticable pattern in sales

[115]:
```
data['quarter']  = data.date.dt.quarter
data
```

[115]:

	date	item_id	price	item_count	store_id	item_name	kcal	restaurant_name	weekday	month	quarter
0	2019-01-01	3	29.22	2.0	1	Sweet Fruity Cake	931	Bob's Diner	Tuesday	January	1
1	2019-01-01	4	26.42	22.0	1	Amazing Steak Dinner with Rolls	763	Bob's Diner	Tuesday	January	1
2	2019-01-01	12	4.87	7.0	1	Fantastic Sweet Cola	478	Bob's Diner	Tuesday	January	1
3	2019-01-01	13	4.18	12.0	1	Sweet Frozen Soft Drink	490	Bob's Diner	Tuesday	January	1
4	2019-01-01	16	3.21	136.0	1	Frozen Milky Smoothy	284	Bob's Diner	Tuesday	January	1
...	...	...	...	...	...	...	...	...	...	...	...
109595	2021-12-31	49	1.39	0.0	3	Awesome Smoothy	78	Sweet Shack	Friday	December	4
109596	2021-12-31	52	5.68	0.0	3	Original Sweet Milky Soft Drink	535	Sweet Shack	Friday	December	4
109597	2021-12-31	77	7.70	0.0	3	Blue Ribbon Frozen Milky Cake	636	Sweet Shack	Friday	December	4
109598	2021-12-31	81	5.11	0.0	3	Fantastic Milky Smoothy	383	Sweet Shack	Friday	December	4
109599	2021-12-31	86	6.50	0.0	3	Original Milky Cake	595	Sweet Shack	Friday	December	4

109600 rows × 11 columns

[116]:
```
data['year'] = data.date.dt.year
data['quart-year'] = "Q" + data.quarter.astype(str) + "_" + data.year.astype(str)
data
```

[116]:

	date	item_id	price	item_count	store_id	item_name	kcal	restaurant_name	weekday	month	quarter	year	quart-year
0	2019-01-01	3	29.22	2.0	1	Sweet Fruity Cake	931	Bob's Diner	Tuesday	January	1	2019	Q1_2019
1	2019-01-01	4	26.42	22.0	1	Amazing Steak Dinner with Rolls	763	Bob's Diner	Tuesday	January	1	2019	Q1_2019
2	2019-01-01	12	4.87	7.0	1	Fantastic Sweet Cola	478	Bob's Diner	Tuesday	January	1	2019	Q1_2019
3	2019-01-01	13	4.18	12.0	1	Sweet Frozen Soft Drink	490	Bob's Diner	Tuesday	January	1	2019	Q1_2019
4	2019-01-01	16	3.21	136.0	1	Frozen Milky Smoothy	284	Bob's Diner	Tuesday	January	1	2019	Q1_2019
...	...	...	...	...	...	...	...	...	...	...	...	...	...
109595	2021-12-31	49	1.39	0.0	3	Awesome Smoothy	78	Sweet Shack	Friday	December	4	2021	Q4_2021
109596	2021-12-31	52	5.68	0.0	3	Original Sweet Milky Soft Drink	535	Sweet Shack	Friday	December	4	2021	Q4_2021
109597	2021-12-31	77	7.70	0.0	3	Blue Ribbon Frozen Milky Cake	636	Sweet Shack	Friday	December	4	2021	Q4_2021
109598	2021-12-31	81	5.11	0.0	3	Fantastic Milky Smoothy	383	Sweet Shack	Friday	December	4	2021	Q4_2021
109599	2021-12-31	86	6.50	0.0	3	Original Milky Cake	595	Sweet Shack	Friday	December	4	2021	Q4_2021

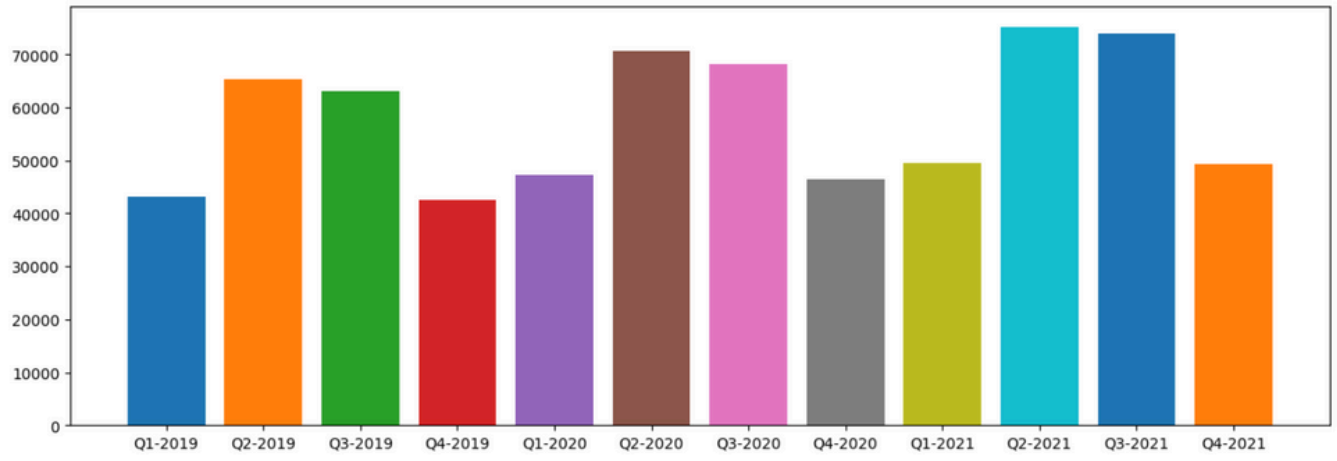
109600 rows × 13 columns

[117]:
```
data['month_name'] = data.date.dt.month_name()
#data['quart-year'] = "Q" + data.quarter.astype(str) + "_" + data.year.astype(str)
data
```

[117]:

	date	item_id	price	item_count	store_id	item_name	kcal	restaurant_name	weekday	month	quarter	year	quart-year	month_name
0	2019-01-01	3	29.22	2.0	1	Sweet Fruity Cake	931	Bob's Diner	Tuesday	January	1	2019	Q1_2019	January
1	2019-01-01	4	26.42	22.0	1	Amazing Steak Dinner with Rolls	763	Bob's Diner	Tuesday	January	1	2019	Q1_2019	January
2	2019-01-01	12	4.87	7.0	1	Fantastic Sweet Cola	478	Bob's Diner	Tuesday	January	1	2019	Q1_2019	January
3	2019-01-01	13	4.18	12.0	1	Sweet Frozen Soft Drink	490	Bob's Diner	Tuesday	January	1	2019	Q1_2019	January
4	2019-01-01	16	3.21	136.0	1	Frozen Milky Smoothy	284	Bob's Diner	Tuesday	January	1	2019	Q1_2019	January
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
109595	2021-12-31	49	1.39	0.0	3	Awesome Smoothy	78	Sweet Shack	Friday	December	4	2021	Q4_2021	December
109596	2021-12-31	52	5.68	0.0	3	Original Sweet Milky Soft Drink	535	Sweet Shack	Friday	December	4	2021	Q4_2021	December
109597	2021-12-31	77	7.70	0.0	3	Blue Ribbon Frozen Milky Cake	636	Sweet Shack	Friday	December	4	2021	Q4_2021	December
109598	2021-12-31	81	5.11	0.0	3	Fantastic Milky Smoothy	383	Sweet Shack	Friday	December	4	2021	Q4_2021	December
...	2021-12-													

[118]:
```
# Check and update the 'quart-year' column in your DataFrame
data['quart-year'] = "Q" + data['quarter'].astype(str) + "-" + data['year'].astype(str)
# Create the order list for the quarters and years
order = ['Q{}-{}'.format(j, i) for i in range(2019, 2022) for j in range(1, 5)]
# Group by 'quart-year' and sum 'item_count'
grpd = data.groupby(['quart-year'])[['item_count']].sum()
# Reindex the grpd DataFrame to match the order with the quarters and years
grpd = grpd.reindex(order).squeeze()
# Plot the data using a bar plot
plt.figure(figsize=(15, 5))
plt.bar(x=grpd.index, height=grpd.values, color=pal)
plt.show()
```

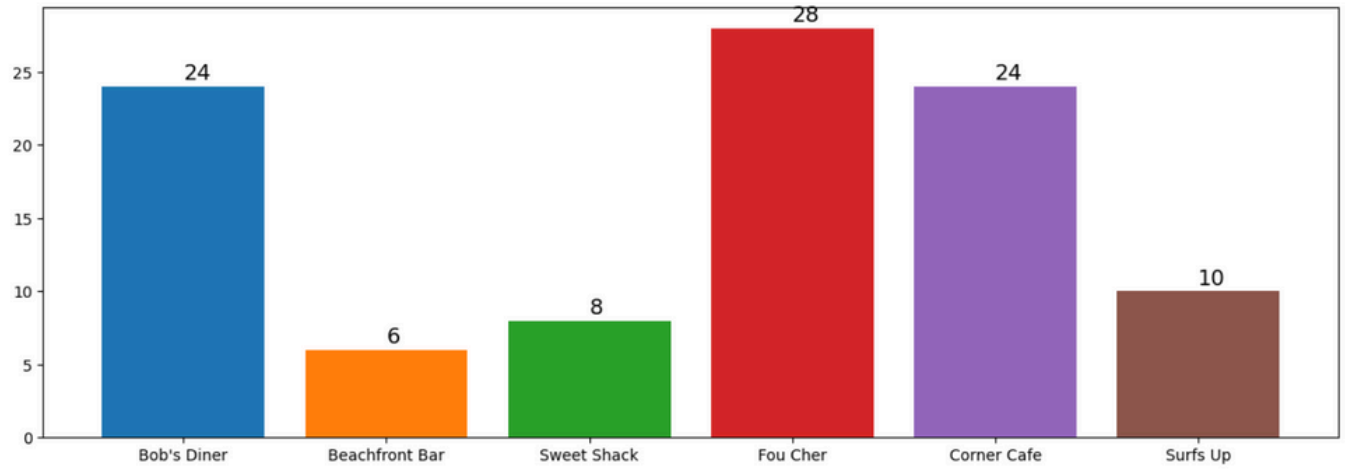


[119]:
```
data.groupby('store_id')[['item_count']].sum()
```

[119]:

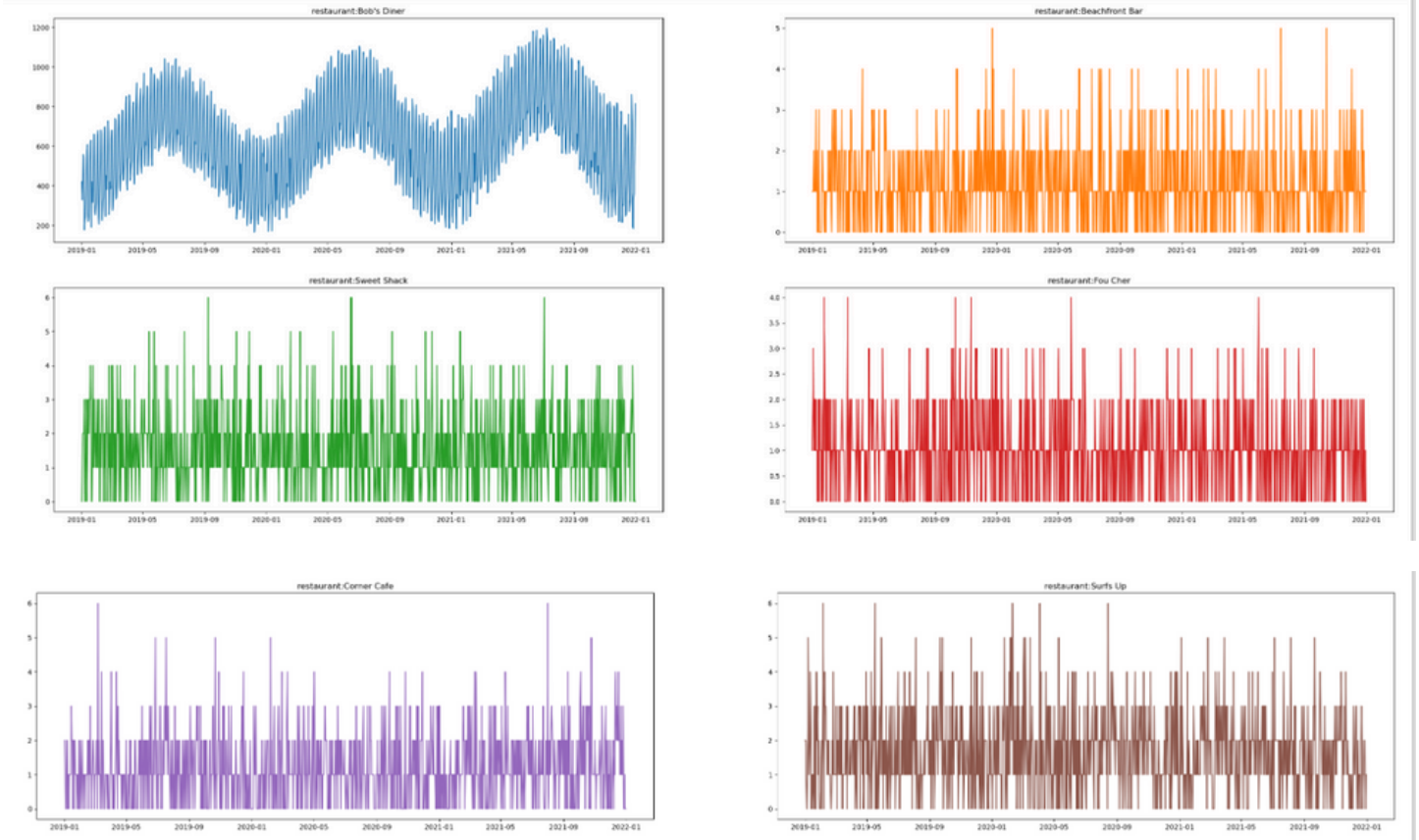
store_id	item_count
1	687527.0
2	1305.0
3	1736.0
4	1106.0
5	1310.0
6	1803.0

[120]:
```
items_served=items.groupby(['store_id']).agg({'id' : "nunique"}).squeeze()
plt.figure(figsize = (15,5))
plt.bar(items_served.index,items_served.values,color=pal)
plt.xticks(range(1,7),resturants.name)
for i in range(1,7):
    plt.annotate(items_served[i],xy = (i,items_served[i]+0.45),size=14)

plt.show()
```



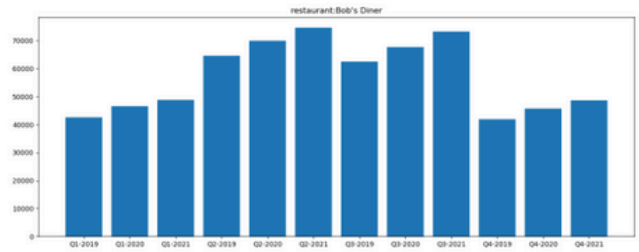
Store wise sales

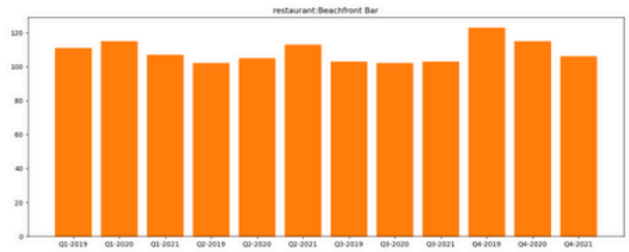
[121]:
```
plot_data = data.pivot_table(index='date', columns='store_id', values='item_count', aggfunc='sum')

f, ax = plt.subplots(3, 2, figsize=(35, 20))

r, c = 0, 0
for i in range(1, 7):
    ax[r, c].plot(plot_data.loc[:, i], color=pal_list[i - 1])
    ax[r, c].set_title('restaurant:{}'.format(resturants.loc[resturants.id == i, 'name'].values[0]))
    c += 1
    if c == 2:
        c = 0
        r += 1

plt.show()
```

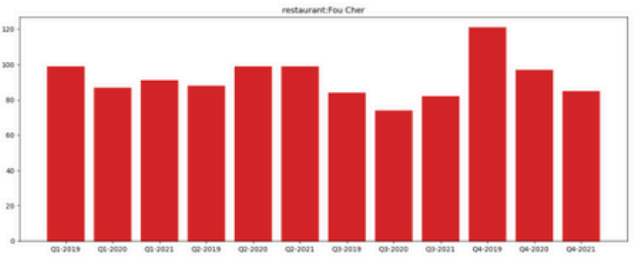


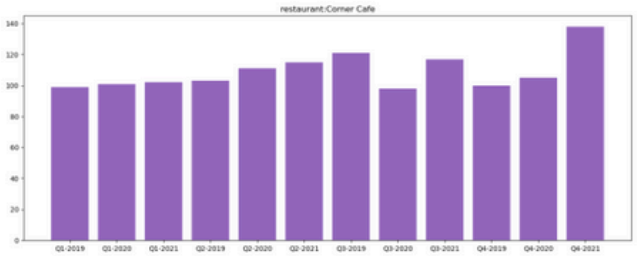
Year wise sales for each store

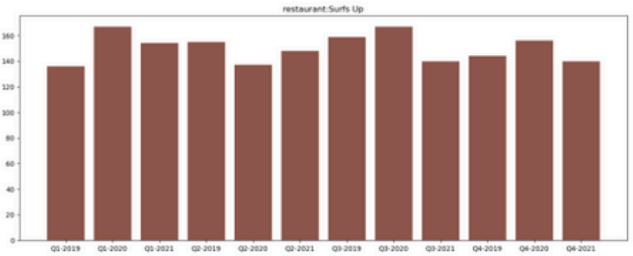
[137]:
```
plot_data = data.pivot_table(index='quart-year', columns='store_id', values='item_count', aggfunc='sum')

f, ax = plt.subplots(3, 2, figsize=(35, 20))

r, c = 0, 0
for i in range(1, 7):
    ax[r, c].bar(x=plot_data.index,height=plot_data.loc[:,i], color=pal_list[i - 1])
    ax[r, c].set_title('restaurant:{}'.format(resturants.loc[resturants.id == i, 'name'].values[0]))
    c += 1
    if c == 2:
        c = 0
        r += 1

plt.show()
```












```
# Develop a linear regression, random forest and xgboost model to predict for the model with the given data
```

```
# Create required features for development of these models like day of the week, quarter of the year, day of the month etc
# Use last six months data as test data
# Compute the RMSE values for each of the models to compare their performances
# Also use these models to forecast for 1 year
```

[27]:
```
data.head(2)
```

[27]:

	date	item_id	price	item_count	store_id	item_name	kcal	restaurant_name	weekday	month	quarter	year	quart-year	month_name
0	2019-01-01	3	29.22	2.0	1	Sweet Fruity Cake	931	Bob's Diner	Tuesday	January	1	2019	Q1-2019	January
1	2019-01-01	4	26.42	22.0	1	Amazing Steak Dinner with Rolls	763	Bob's Diner	Tuesday	January	1	2019	Q1-2019	January

[138]:
```
time_series = data.groupby(['date']).agg({'item_count' : 'sum','weekday' : lambda x : x.unique()[0],
                                          'quarter' : lambda x : x.unique()[0], 'year' : lambda x : x.unique()[0],
                                           'month_name' : lambda x : x.unique()[0], 'year' : lambda x : x.unique()[0],
                                          'year' : lambda x : x.unique()[0] })
```

[139]:
```
time_series.head(2)
```

[139]:

date	item_count	weekday	quarter	year	month_name
2019-01-01	427.0	Tuesday	1	2019	January
2019-01-02	337.0	Wednesday	1	2019	January

Add some more features : Day of the year, Day of the month and week of the year

[140]:
```
time_series['day_year'] = time_series.index.day_of_year
time_series['day_month'] = time_series.index.day
time_series['week_num'] = time_series.index.isocalendar().week.astype(int)
time_series
```

[140]:

date	item_count	weekday	quarter	year	month_name	day_year	day_month	week_num
2019-01-01	427.0	Tuesday	1	2019	January	1	1	1
2019-01-02	337.0	Wednesday	1	2019	January	2	2	1
2019-01-03	445.0	Thursday	1	2019	January	3	3	1
2019-01-04	564.0	Friday	1	2019	January	4	4	1
2019-01-05	552.0	Saturday	1	2019	January	5	5	1
...	...	...	...	...	...	...	...	...
2021-12-27	192.0	Monday	4	2021	December	361	27	52
2021-12-28	344.0	Tuesday	4	2021	December	362	28	52
2021-12-29	371.0	Wednesday	4	2021	December	363	29	52
2021-12-30	527.0	Thursday	4	2021	December	364	30	52
2021-12-31	817.0	Friday	4	2021	December	365	31	52

1096 rows × 8 columns

Convert categorical to numeric using ordinal encoder as here data is ordered in terms of week day and month name

[141]:
```
from sklearn.preprocessing import OrdinalEncoder
```

[135]:
```
day_names
```

[135]:
```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

[142]:
```
ord_enc=OrdinalEncoder(categories=[months,day_names])
ord_enc.fit(time_series[['month_name','weekday']])
time_series[['month_name','weekday']] = ord_enc.transform(time_series[['month_name','weekday']])
```

```
# Generate test dataset
```

[143]:
```
train = time_series[time_series.index < '2021-07-01']
test = time_series[time_series.index >= '2021-07-01']
y_var='item_count'
x_vars = time_series.drop(columns = 'item_count').columns
X_train = train [x_vars]
Y_train = train[y_var]
X_test = test[x_vars]
Y_test = test[y_var]
```

Linear Regression

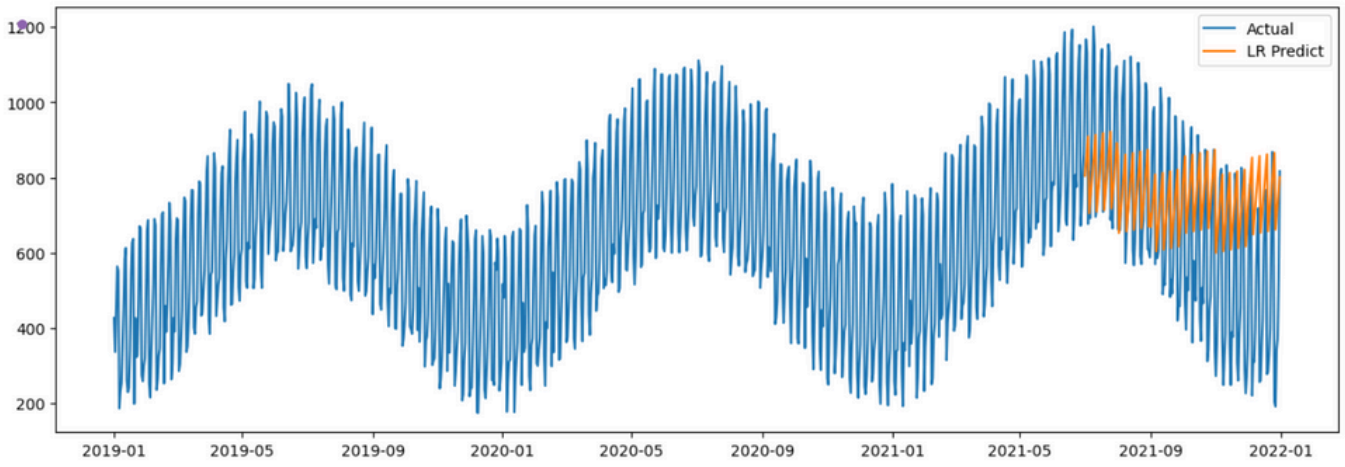
[144]:
```
lr = LinearRegression()
lr.fit(X_train,Y_train)
```

[144]:
```
LinearRegression
LinearRegression()
```

[145]:
```
test['lr_pred'] = lr.predict(X_test)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_14084\3421247276.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['lr_pred'] = lr.predict(X_test)
```

[146]:
```
plt.figure(figsize=(15,5))
plt.plot(time_series.item_count, label="Actual")
plt.plot(test.lr_pred, label="LR Predict")
plt.legend()
plt.show()
```



[147]:
```
lr_rmse=mean_squared_error(y_true = test.item_count, y_pred = test.lr_pred, squared=False)
lr_mae=mean_absolute_error(y_true=test.item_count,y_pred=test.lr_pred)
lr_r2_score=r2_score(y_true=test.item_count,y_pred=test.lr_pred)
print("Linear Regression :\nRMSE : {:.2f}\nMAE : {:.2f}\nR2_Score : {:.2f}".format(lr_rmse,lr_mae,lr_r2_score))
```

```
Linear Regression :
RMSE : 244.21
MAE : 196.04
R2_Score : 0.08
```

[148]:
```
results = pd.DataFrame([lr_rmse,lr_mae,lr_r2_score*100],index=['RMSE','MAE','R2_Score'],columns=['Value'])
results
```

[148]:

	Value
RMSE	244.210844
MAE	196.037167
R2_Score	7.649488

Random Forest

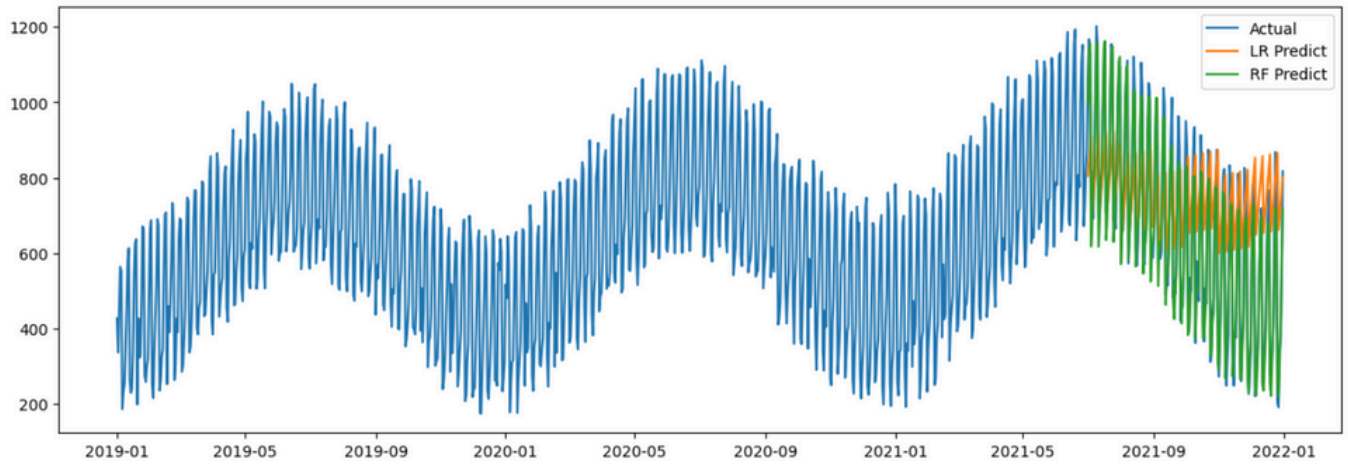
[40]:
```
rf=RandomForestRegressor(n_estimators=100,max_depth=20)
rf.fit(X_train,Y_train)
```

[40]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=20)
```

[41]:
```
test['rf_pred'] = rf.predict(X_test)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_14084\582745053.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['rf_pred'] = rf.predict(X_test)
```

[42]:
```
plt.figure(figsize=(15,5))
plt.plot(time_series.item_count, label="Actual")
plt.plot(test.lr_pred, label="LR Predict")
plt.plot(test.rf_pred, label="RF Predict")
plt.legend()
plt.show()
```



[43]:
```
rf_rmse=mean_squared_error(y_true = test.item_count, y_pred = test.rf_pred, squared=False)
rf_mae=mean_absolute_error(y_true=test.item_count,y_pred=test.rf_pred)
rf_r2_score=r2_score(y_true=test.item_count,y_pred=test.rf_pred)
print("Linear Regression :\nRMSE : {:.2f}\nMAE : {:.2f}\nR2_Score : {:.2f}".format(rf_rmse,rf_mae,rf_r2_score))
```

```
Linear Regression :
RMSE : 59.58
MAE : 47.25
R2_Score : 0.95
```

[44]:
```
results["Random Forest"] = [rf_rmse,rf_mae,rf_r2_score*100]
results.round(2)
```

[44]:

	Value	Random Forest
RMSE	244.21	59.58
MAE	196.04	47.25
R2_Score	7.65	94.50

XGBoost

[45]:
```
xgb = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators=1000,
                       early_stopping_rounds=50,
                       objective='reg:linear',
                       max_depth=3,
                       learning_rate=0.01)
xgb.fit(X_train, Y_train,
        eval_set=[(X_train,Y_train),(X_test,Y_test)],
        verbose=100)
```

```
[0]    validation_0-rmse:660.07829    validation_1-rmse:709.01212
[100]  validation_0-rmse:262.51190    validation_1-rmse:311.53220
[200]  validation_0-rmse:121.36888    validation_1-rmse:171.11193
C:\Users\HP\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [09:59:45] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-windows\src\objective\regression_obj.cu:227: reg:linear is now deprecated in favor of reg:squarederror.
  warnings.warn(smsg, UserWarning)
[300]  validation_0-rmse:71.72975     validation_1-rmse:105.03640
[400]  validation_0-rmse:52.50055     validation_1-rmse:78.37923
[500]  validation_0-rmse:44.19461     validation_1-rmse:71.59216
[590]  validation_0-rmse:40.38314     validation_1-rmse:71.95490
```

[45]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, objective='reg:linear', ...)
```

[46]:
```
# Generate predictions using the trained XGBoost model
test['xgb_pred'] = xgb.predict(X_test)
xgb_rmse=mean_squared_error(y_true = test.item_count, y_pred = test.xgb_pred, squared=False)
xgb_mae=mean_absolute_error(y_true=test.item_count,y_pred=test.xgb_pred)
xgb_r2_score=r2_score(y_true=test.item_count,y_pred=test.xgb_pred)
print("Linear Regression :\nRMSE : {:.2f}\nMAE : {:.2f}\nR2_Score : {:.2f}".format(xgb_rmse,xgb_mae,xgb_r2_score))
```

```
Linear Regression :
RMSE : 71.18
MAE : 58.45
R2_Score : 0.92
```

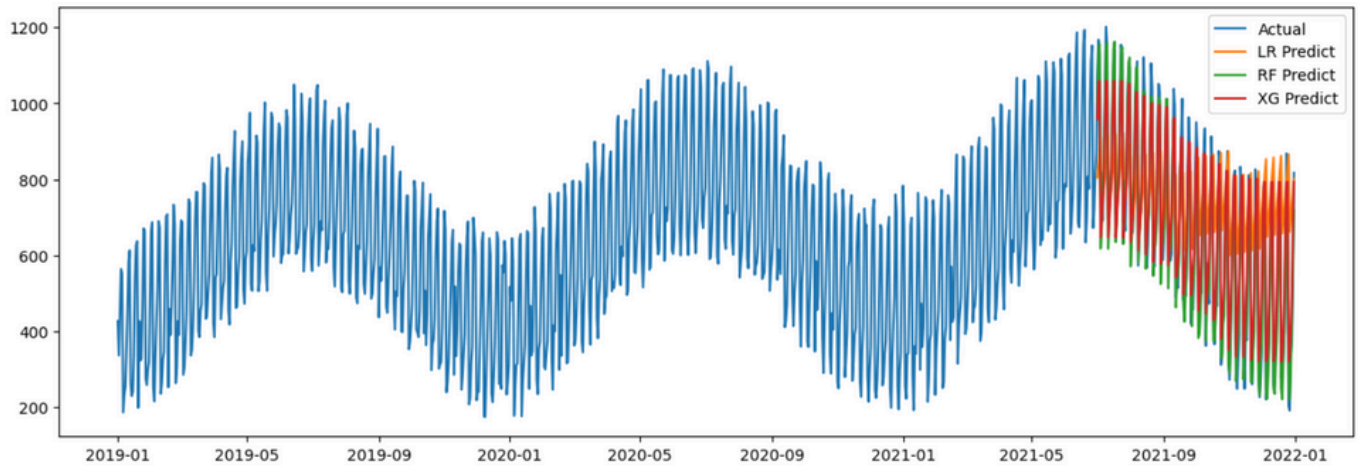
```
C:\Users\HP\AppData\Local\Temp\ipykernel_14084\914599684.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['xgb_pred'] = xgb.predict(X_test)
```

[47]:
```
results["XGBoost"] = [xgb_rmse,xgb_mae,xgb_r2_score*100]
results.round(2)
```

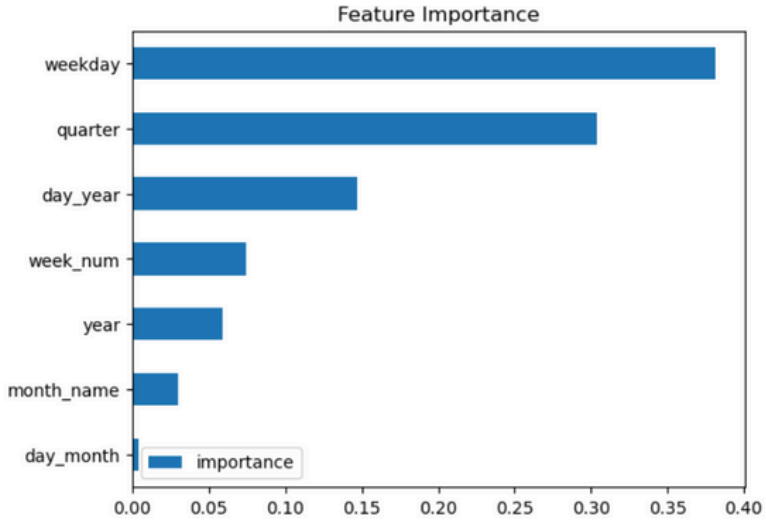
[47]:

	Value	Random Forest	XGBoost
RMSE	244.21	59.58	71.18
MAE	196.04	47.25	58.45
R2_Score	7.65	94.50	92.15

[48]:
```
plt.figure(figsize=(15,5))
plt.plot(time_series.item_count, label="Actual")
plt.plot(test.lr_pred, label="LR Predict")
plt.plot(test.rf_pred, label="RF Predict")
plt.plot(test.xgb_pred, label="XG Predict")
plt.legend()
plt.show()
```



[49]:
```
importance=pd.DataFrame(data=xgb.feature_importances_,
                        index=xgb.get_booster().feature_names,
                        columns=['importance'])
importance.sort_values('importance').plot(kind='barh',title='Feature Importance')
plt.show()
```




```
# The best model is RF (Random Foerst). It can be used for predictions
```

[50]:
```
time_series.index.max()
```

[50]:
```
Timestamp('2021-12-31 00:00:00')
```

[52]:
```
future_predictors = pd.DataFrame(pd.date_range(start="2022-01-01",end="2022-12-31"),columns=['date'])
future_predictors.index = pd.to_datetime(future_predictors.date)
```

[53]:
```
X_train.columns
```

[53]:
```
Index(['weekday', 'quarter', 'year', 'month_name', 'day_year', 'day_month',
       'week_num'],
      dtype='object')
```

[54]:
```
future_predictors.date.dt.isocalendar().week
```

[54]:
```
date
2022-01-01    52
2022-01-02    52
2022-01-03     1
2022-01-04     1
2022-01-05     1
              ..
2022-12-27    52
2022-12-28    52
2022-12-29    52
2022-12-30    52
2022-12-31    52
Name: week, Length: 365, dtype: UInt32
```

Forecasting using Deep Learning

[70]:
```
time_ser=data.groupby('date').agg({'price':'sum'})
time_ser
#data.head(2)
```

[70]:

	price
date	
2019-01-01	1176.37
2019-01-02	1176.37
2019-01-03	1176.37
2019-01-04	1176.37
2019-01-05	1176.37
...	...
2021-12-27	1176.37
2021-12-28	1176.37
2021-12-29	1176.37
2021-12-30	1176.37
2021-12-31	1176.37

1096 rows × 1 columns

Define train and test series

[72]:
```
train=time_ser[time_ser.index < '2021-07-01']
test=time_ser[time_ser.index >= '2021-07-01']
```

Data scaling

[73]:
```
scaler = MinMaxScaler()
scaler.fit(train)
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

Generate synthetic data

[75]:
```
n_features = 1
length=12
generator = TimeseriesGenerator(scaled_train,scaled_train,length=length,batch_size=1)
```

Build and train LSTM model

[77]:
```
model = Sequential()
model.add(LSTM(100,activation='relu',input_shape=(length,n_features)))
model.add(Dense(1))
model.compile(optimizer='adam',loss='mse')
```

```
C:\Users\HP\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[78]:
```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

[79]:
```
validation_generator = TimeseriesGenerator(scaled_test,scaled_test,length=length,batch_size=1)
```

Fit Model

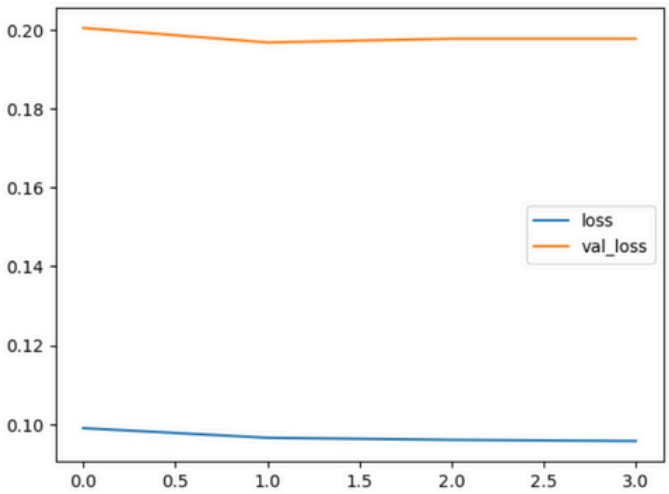
[81]:
```
model.fit(generator, epochs=20, validation_data=validation_generator, callbacks=[early_stop])
```

```
Epoch 1/20
 16/900 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 3.9700e-05
C:\Users\HP\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
900/900 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - loss: 0.0864 - val_loss: 0.2005
Epoch 2/20
900/900 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - loss: 0.0986 - val_loss: 0.1968
Epoch 3/20
900/900 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - loss: 0.1146 - val_loss: 0.1978
Epoch 4/20
900/900 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - loss: 0.1010 - val_loss: 0.1978
```

[81]:
```
<keras.src.callbacks.history.History at 0x2afd930b350>
```

[82]:
```
losses = pd.DataFrame(model.history.history)
losses.plot()
```

[82]:
```
<Axes: >
```



Make Predictions

[85]:
```
true_predictions = scaler.inverse_transform(test_predictions)
test['Predictions']=true_predictions
```

[86]:
```
# Assuming test_predictions is the predictions made by your model on the test data
test_predictions = model.predict(test)

# Inverse transform the scaled predictions
true_predictions = scaler.inverse_transform(test_predictions)

# Add the 'Predictions' column to the test DataFrame
test['Predictions'] = true_predictions
```

Evaluate

[96]:
```
print(current_batch.shape)  # Check the shape
```

```
(1, 12, 1)
```

[99]:
```
from keras.models import Sequential
from keras.layers import LSTM, Input

model = Sequential()
model.add(Input(shape=(12, 1)))  # Specify the input shape here
model.add(LSTM(units=50))        # Add the LSTM layer without the input_shape argument
```

[101]:
```
import numpy as np

# Assuming model and test are predefined
test_predictions = []

first_eval_batch = scaled_train[-length:]
current_batch = first_eval_batch.reshape((1, length, n_features))

for i in range(len(test)):
    # Make the prediction
    current_pred = model.predict(current_batch)[0]

    # Store the prediction
    test_predictions.append(current_pred)

    # Reshape the prediction to match the input shape
    current_pred = np.expand_dims(current_pred, axis=0)
    current_pred = np.expand_dims(current_pred, axis=-1)  # Ensure current_pred has 3 dimensions

    # Update current_batch by removing the first time step and adding the new prediction
    current_batch = np.append(current_batch[:, 1:, :], current_pred, axis=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 451ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 214ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 229ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 217ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 234ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 267ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 270ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 278ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 292ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 286ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 285ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 285ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 304ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 387ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 331ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 335ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 336ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 345ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 361ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 333ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 362ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 399ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 377ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 383ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 404ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 410ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 427ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 410ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 592ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 589ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 600ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 616ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 612ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 639ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 612ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 641ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 627ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 625ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 651ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 668ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 664ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 674ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 676ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 686ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 650ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 801ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 827ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 632ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 698ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 653ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 666ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 671ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 633ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 590ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 839ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 763ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 585ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 924ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 751ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 716ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 748ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 815ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 732ms/step
```