grocery-sales-forecasting-before-running

October 5, 2023

1 Machine Learning Techniques for Sales Forecasting

1.1 Importing Libraries

[ ]:
```
%pip install xgboost
%pip install statsmodels
%pip install pandas numpy statsmodels
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.linear_model import Ridge
from xgboost import XGBRegressor
from sklearn.linear_model import Lasso
from statsmodels.tsa.arima.model import ARIMA
from sklearn.linear_model import BayesianRidge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import panel as pn
pn.extension()
import hvplot.pandas
from statsmodels.tsa.stattools import adfuller
```

1.2 Importing Datasets & Read all csv files

1. item_categories.csv - item_category_name, item_category_id
2. items.csv - item_name, item_id, category_id
3. sales_train.csv - date, date_block_num, shop_id, item_id, item_price, item_cnt_day
4. shops.csv - shop_name, shop_id
5. test.csv - ID, shop_id, item_id

[ ]:
```
#importing data
item_categories = pd.read_csv('./data-set/item_categories.csv')
items = pd.read_csv('./data-set/items.csv')
sales_train = pd.read_csv('./data-set/sales_train.csv')
shops = pd.read_csv('./data-set/shops.csv')
test = pd.read_csv('./data-set/test.csv')
```

[ ]:
```
#checking the shape of the data
print("Shape of item_categories:", item_categories.shape)
print("Shape of items:", items.shape)
print("Shape of sales_train:", sales_train.shape)
print("Shape of shops:", shops.shape)
print("Shape of test:", test.shape)
```

[ ]:
```
#checking the columns of the data
print("\n\nColumns of item_categories:\n")
print(item_categories.info())

print("-----------------------------------------------------")

print("\n\nColumns of items:\n")
print(items.info())

print("-----------------------------------------------------")

print("\n\nColumns of sales_train:\n")
print(sales_train.info())

print("-----------------------------------------------------")

print("\n\nColumns of shops:\n")
print(shops.info())

print("-----------------------------------------------------")

print("\n\nColumns of test:\n")
print(test.info())
```

[ ]:
```
#checking the head and tail of the data

print("\n\nHead of item_categories:\n")
print(item_categories.head())

print("\n\nTail of item_categories:\n")
print(item_categories.tail())

print("-----------------------------------------------------")

print("\n\nHead of items:\n")
print(items.head())

print("\n\nTail of items:\n")
print(items.tail())

print("-----------------------------------------------------")

print("\n\nHead of sales_train:\n")
print(sales_train.head())

print("\n\nTail of sales_train:\n")
print(sales_train.tail())

print("-----------------------------------------------------")

print("\n\nHead of shops:\n")
print(shops.head())

print("\n\nTail of shops:\n")
print(shops.tail())

print("-----------------------------------------------------")

print("\n\nHead of test:\n")
print(test.head())

print("\n\nTail of test:\n")
print(test.tail())
```

1.3 Data Preprocessing & Feature Engineering

[ ]:
```
#merging the data for better understand the data
```

[ ]:
```
#Merge sales_train.csv with items.csv on the "item_id" column
sales_with_items = sales_train.merge(items, on='item_id', how='left')
print("\n\nHead of sales_with_items:\n")
```

```
print(sales_with_items.head(20))
print(sales_with_items.shape)
```

[ ]:
```
#Merge the result with item_categories.csv on the "category_id"
sales_with_items_and_categories = sales_with_items.merge(item_categories,␣
↪right_on='item_category_id', left_on='category_id', how='left')
print("\n\nHead of sales_with_items_and_categories:\n")
print(sales_with_items_and_categories.head(20))
print(sales_with_items_and_categories.shape)
```

[ ]:
```
# Check if the two columns are the same
if sales_with_items_and_categories['item_category_id'].
↪equals(sales_with_items_and_categories['category_id']):
    # If they are the same, drop one of the columns
    sales_with_items_and_categories.drop(columns=['item_category_id'],␣
↪inplace=True)
```

[ ]:
```
print("\n\nHead of sales_with_items_and_categories:\n")
print(sales_with_items_and_categories.head(20))
print(sales_with_items_and_categories.shape)
```

[ ]:
```
#Merge the result with shops.csv on the "shop_id"
final_dataset = sales_with_items_and_categories.merge(shops, on='shop_id',␣
↪how='left')
print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#checks the columns of the final dataset
print("\n\nColumns of final_dataset:\n")
print(final_dataset.info())
```

[ ]:
```
#prints the date and date_block_num column to check whether they are related
columns_to_print = ['date', 'date_block_num']
print(final_dataset[columns_to_print])
```

[ ]:
```
# Rename the column
final_dataset.rename(columns={'date_block_num': 'month_num'}, inplace=True)
```

[ ]:
```
#Rename the column
final_dataset.rename(columns={'item_cnt_day': 'item_cnt_month'}, inplace=True)
```

[ ]:
```
print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#checks the columns of the final dataset
print("\n\nColumns of final_dataset:\n")
print(final_dataset.info())
```

[ ]:
```
#export the final dataset to csv file
final_dataset.to_csv('./data-set/output/final_dataset_without_cleaning.csv',␣
↪index=False)
```

[ ]:
```
#Data Cleaning

#checking for missing values
print("\n\nMissing values in final_dataset:\n")
print(final_dataset.isnull().sum())
```

[ ]:
```
#checking for null values
print("\n\nNull values in final_dataset:\n")
print(final_dataset.isnull().sum())
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#handles the missing values in final_dataset
final_dataset['item_name'].fillna('Unknown', inplace=True)
final_dataset['item_category_name'].fillna('Unknown', inplace=True)
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#removes duplicates rows in final_dataset
final_dataset.drop_duplicates(inplace=True)
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#checks and solves the data type of the columns
print("\n\nData types of final_dataset:\n")
print(final_dataset.dtypes)
```

[ ]:
```
# #seems like item_cnt_month should be int64
final_dataset['item_cnt_month'] = final_dataset['item_cnt_month'].
↪astype('int64')
```

[ ]:
```
print(final_dataset.dtypes)
```

[ ]:
```
#prints item_cnt_month column to check whether it is int64
print(final_dataset['item_cnt_month'].head(30))
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#removes -1 and 307980 from item_cnt_month column because it is an outlier
#it is not possible to sell -1 and 307980 items in a day because 307980 is the␣
↪total number of items sold in a day
#which means that the data is incorrect
#and -1 is not possible

final_dataset = final_dataset[(final_dataset['item_cnt_month'] > 0) &␣
↪(final_dataset['item_cnt_month'] < 307980)]

print(final_dataset.shape)
```

[ ]:
```
#outlier treatment

#checks for outliers in the item_cnt_month column
print("\n\nOutliers in item_cnt_month column:\n")
print(final_dataset[final_dataset['item_cnt_month'] > 1000])

#removes the outliers in the item_cnt_month column
final_dataset = final_dataset[final_dataset['item_cnt_month'] < 1000]

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#deal with the incorrect data in the item_price column
#the item_price should not be negative
#the item_price should not be zero
#the item_price should not be greater than 100000

final_dataset = final_dataset[(final_dataset['item_price'] > 0) &␣
↪(final_dataset['item_price'] < 100000)]
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#handles special characters and formatting in the data set
final_dataset['item_name'] = final_dataset['item_name'].str.
↪replace('[^A-Za-z0-9 - - ]+', ' ')
```

[ ]:
```
print(final_dataset.shape)
```

[ ]:
```
#removes the noise in the item_name column
final_dataset['item_name'] = final_dataset['item_name'].str.replace('  ', ' ')
```

[ ]:
```
print(final_dataset.head())
```

[ ]:
```
#creates a new column called revenue
```

```
final_dataset['revenue'] = final_dataset['item_cnt_month'] *␣
 ↪final_dataset['item_price']
```

[ ]:
```
print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#creates a new column called revenue_per_item
final_dataset['revenue_per_item'] = final_dataset['revenue'] /␣
 ↪final_dataset['item_cnt_month']


print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#checks whether the revenue_per_item column and revenue column are the same

if final_dataset['revenue_per_item'].equals(final_dataset['revenue']):
    # If they are the same, drop one of the columns
    final_dataset.drop(columns=['revenue_per_item'], inplace=True)

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#creates a new column called date num
final_dataset['date_num'] = final_dataset['date'].str[:2]
```

[ ]:
```
print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#creates a new column called year num
final_dataset['year_num'] = final_dataset['date'].str[6:]
```

[ ]:
```
print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
print(final_dataset.shape)
print(final_dataset.info())
```

[ ]:
```
# rearrange the columns
final_dataset = final_dataset[['date', 'date_num', 'year_num', 'month_num',␣
 ↪'shop_id', 'shop_name', 'item_id', 'item_name', 'category_id',␣
 ↪'item_category_name', 'item_price', 'item_cnt_month', 'revenue']]
```

```
print(final_dataset.shape)
print(final_dataset.info())
```

[ ]:
```
#data profiling

#descriptive statistics
print("\n\nDescriptive statistics of final_dataset:\n")
print(final_dataset.describe())
```

[ ]:
```
#data enrichment

#creates a new column called month name
final_dataset['month_name'] = final_dataset['month_num'].replace({0: 'January',␣
 ↪1: 'February', 2: 'March', 3: 'April', 4: 'May', 5: 'June', 6: 'July', 7:␣
 ↪'August', 8: 'September', 9: 'October', 10: 'November', 11: 'December', 12:␣
 ↪'January', 13: 'February', 14: 'March', 15: 'April', 16: 'May', 17: 'June',␣
 ↪18: 'July', 19: 'August', 20: 'September', 21: 'October', 22: 'November', 23:
 ↪ 'December', 24: 'January', 25: 'February', 26: 'March', 27: 'April', 28:␣
 ↪'May', 29: 'June', 30: 'July', 31: 'August', 32: 'September', 33: 'October'})

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#removes month_num column

final_dataset.drop(columns=['month_num'], inplace=True)

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#rearange the columns

final_dataset = final_dataset[['date', 'date_num', 'month_name', 'year_num',␣
 ↪'shop_id', 'shop_name', 'item_id', 'item_name', 'category_id',␣
 ↪'item_category_name', 'item_price', 'item_cnt_month', 'revenue']]

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#data binning

#found the bins using the following code
print(final_dataset['item_price'].max())
```

```
print(final_dataset['item_price'].min())

#creates a new column called price range
final_dataset['price_range'] = pd.cut(final_dataset['item_price'], bins=[-1,␣
↪100, 200, 300, 400, 500, 600, 700, 800, 900, 100000], labels=['0-100',␣
↪'100-200', '200-300', '300-400', '400-500', '500-600', '600-700', '700-800',␣
↪'800-900', '900-100000'])
```

```
[ ]: print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

```
[ ]: #log transformation

#creates a new column called log_revenue
final_dataset['log_revenue'] = np.log(final_dataset['revenue'])
```

```
[ ]: print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

```
[ ]: #encoding

#encodes the year_num column to 0, 1, 2

final_dataset['year_num'] = final_dataset['year_num'].replace({'2013': 0,␣
↪'2014': 1, '2015': 2})


print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

```
[ ]: #grouping and aggregation

#grouping the data set by shop_id and year_num and aggregating the␣
↪item_cnt_month column using sum

grouped_by_shop_id_and_year_num = final_dataset.groupby(['shop_id',␣
↪'year_num']).agg({'item_cnt_month': 'sum'})

print("\n\nHead of grouped_by_shop_id_and_year_num:\n")
print(grouped_by_shop_id_and_year_num.head(60))
print(grouped_by_shop_id_and_year_num.shape)
```

```
[ ]: #creates a new column called scaled_revenue
```

```
final_dataset['scaled_revenue'] = (final_dataset['revenue'] -
↪final_dataset['revenue'].min()) / (final_dataset['revenue'].max() -
↪final_dataset['revenue'].min())

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

[ ]:
```
#change month_name column to numeric

final_dataset['month_name'] = final_dataset['month_name'].replace({'January':
↪1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June':6, 'July': 7,
↪'August': 8, 'September': 9, 'October': 10, 'November':11, 'December': 12})

print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

1.4 Data Exploration & Analysis

[ ]:
```
#correlation

numeric_columns = final_dataset.select_dtypes(include=['number'])
print("\n\nCorrelation of final_dataset:\n")
print(numeric_columns.corr())
```

[ ]:
```
#checks for missing values
print("\n\nMissing values in final_dataset:\n")
print(final_dataset.isnull().sum())

#checks for null values
print("\n\nNull values in final_dataset:\n")
print(final_dataset.isnull().sum())
```

[ ]:
```
#Descriptive analytics

# Summary Statistics
print("\nDescriptive statistics of final_dataset:")
print(final_dataset.describe())
```

[ ]:
```
#seasonality analysis

grouped_by_month_name = final_dataset.groupby(['month_name']).
↪agg({'item_cnt_month': 'sum'})

print("\n\nHead of grouped_by_month_name:\n")
print(grouped_by_month_name)
```

```
print(grouped_by_month_name.shape)
```

[ ]:
```
#performing seasonal decomposition
decomposition = sm.tsa.seasonal_decompose(grouped_by_month_name,␣
 ↪model='additive', period=1)

#plotting the seasonal decomposition
fig = decomposition.plot()
plt.show()

#plotting the item_cnt_month column
plt.figure(figsize=(20, 10))
plt.plot(final_dataset['item_cnt_month'])
plt.title('Item Count Per Month')
plt.xlabel('Month')
plt.ylabel('Item Count')
plt.show()
```

[ ]:
```
#regulatory analytics

grouped_by_shop_id_and_year_num = final_dataset.groupby(['shop_id',␣
 ↪'year_num']).agg({'item_cnt_month': 'sum'})

print("\n\nHead of grouped_by_shop_id_and_year_num:\n")
print(grouped_by_shop_id_and_year_num.head(60))
```

[ ]:
```
#Variable Identification

# Identify numerical and categorical variables
numerical_vars = final_dataset.select_dtypes(include=['int64', 'float64']).
 ↪columns
categorical_vars = final_dataset.select_dtypes(include=['object', 'category']).
 ↪columns

# Print the list of numerical and categorical variables
print("Numerical Variables:")
print(numerical_vars)

print("\nCategorical Variables:")
print(categorical_vars)
```

[ ]:
```
# univariate analysis

for column in final_dataset.columns:
    variable_type = final_dataset[column].dtype

    summary_stats = final_dataset[column].describe()
```

```python
    plt.figure(figsize=(10, 6))

    # For numerical variables, create a histogram
    if variable_type in ['int64', 'float64']:
        sns.histplot(data=final_dataset, x=column, kde=True)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')

    # For categorical variables, create a bar plot
    else:
        sns.countplot(data=final_dataset, x=column)
        plt.title(f'Counts of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')

    plt.show()

    # Print summary statistics
    print(f"Summary Statistics for {column}:")
    print(summary_stats)
```

[ ]:

```python
#bivariate analysis

#can analysis by changing var1 and var2
var1 = 'item_price'
var2 = 'item_cnt_month'

var1_type = final_dataset[var1].dtype
var2_type = final_dataset[var2].dtype

# Scatter Plot for Numerical vs. Numerical
if var1_type in ['int64', 'float64'] and var2_type in ['int64', 'float64']:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=final_dataset, x=var1, y=var2)
    plt.title(f'Scatter Plot: {var1} vs. {var2}')
    plt.xlabel(var1)
    plt.ylabel(var2)
    plt.grid(True)
    plt.show()

# Box Plot for Categorical vs. Numerical
elif var1_type in ['object', 'category'] and var2_type in ['int64', 'float64']:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=final_dataset, x=var1, y=var2)
    plt.title(f'Box Plot: {var1} vs. {var2}')
```

```
    plt.xlabel(var1)
    plt.ylabel(var2)
    plt.grid(True)
    plt.show()

# Bar Plot for Categorical vs. Categorical
elif var1_type in ['object', 'category'] and var2_type in ['object',␣
 ↪'category']:
    crosstab = pd.crosstab(final_dataset[var1], final_dataset[var2])
    crosstab.plot(kind='bar', stacked=True, figsize=(10, 6))
    plt.title(f'Bar Plot: {var1} vs. {var2}')
    plt.xlabel(var1)
    plt.ylabel('Count')
    plt.grid(True)
    plt.show()

# Print correlation for Numerical vs. Numerical
if var1_type in ['int64', 'float64'] and var2_type in ['int64', 'float64']:
    correlation = final_dataset[[var1, var2]].corr().iloc[0, 1]
    print(f'Correlation between {var1} and {var2}: {correlation:.2f}')
```

[ ]:
```
#Exploratory Data Analysis (EDA)

print("Dataset Overview:")
print(final_dataset.info())

print("\nSummary Statistics for Numerical Variables:")
print(final_dataset.describe())

print("\nMissing Values:")
print(final_dataset.isnull().sum())

numerical_columns = ['month_name', 'year_num', 'shop_id', 'item_id',␣
 ↪'category_id', 'item_price', 'item_cnt_month', 'revenue', 'log_revenue',␣
 ↪'scaled_revenue']

for column in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(data=final_dataset, x=column, kde=True, bins=20)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()

# Visualize relationships between variables with a correlation matrix for␣
 ↪numerical variables
correlation_matrix = final_dataset[numerical_columns].corr()
```

```python
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap for Numerical Variables")
plt.show()

# Explore categorical variables with bar plots
categorical_columns = ['shop_name', 'item_name', 'item_category_name',␣
↪'price_range']

for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(data=final_dataset, x=column)
    plt.title(f'Counts of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

[ ]:

```python
#inferential analysis

np.random.seed(42)
data = np.random.normal(loc=70, scale=10, size=100)

# Create a DataFrame from the generated data
df = pd.DataFrame({'measurement': data})

# Calculate the sample mean and standard deviation
sample_mean = df['measurement'].mean()
sample_std = df['measurement'].std()

# Define a hypothetical population mean for comparison
population_mean = 75

# Perform a t-test to compare the sample mean with the population mean
t_statistic, p_value = stats.ttest_1samp(df['measurement'], population_mean)

# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Sample Standard Deviation: {sample_std:.2f}")
print(f"Population Mean: {population_mean}")
print(f"T-Statistic: {t_statistic:.2f}")
print(f"P-Value: {p_value:.4f}")

# Determine statistical significance
alpha = 0.05  # Significance level (adjust as needed)
if p_value < alpha:
```

```
    print("Reject the null hypothesis: The sample mean is statistically␣
 ↪different from the population mean.")
else:
    print("Fail to reject the null hypothesis: There is no significant␣
 ↪difference between the sample mean and the population mean.")
```

[ ]:
```
#dianostic analytics

# Generate a hypothetical dataset
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 3 * X + 2 + np.random.randn(100, 1)

# Create a DataFrame from the generated data
df = pd.DataFrame({'X': X.flatten(), 'y': y.flatten()})

# Diagnostic Plots
plt.figure(figsize=(12, 6))

plt.tight_layout()
plt.show()
```

[ ]:
```
#qualitative analytics

category_counts = final_dataset['item_category_name'].value_counts()
print(category_counts)

cross_tab = pd.crosstab(final_dataset['shop_name'],␣
 ↪final_dataset['item_category_name'])
print(cross_tab)

category_frequency = (final_dataset['price_range'] == 'Low').sum()
print(f"Frequency of 'Low' price range: {category_frequency}")

average_price_per_category = final_dataset.
 ↪groupby('item_category_name')['item_price'].mean()
print(average_price_per_category)

category_counts.plot(kind='bar', figsize=(10, 6))
plt.title('Item Category Counts')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#stationarity analysis
```

```
[ ]: # Convert the date column to datetime format
final_dataset['date'] = pd.to_datetime(final_dataset['date'], format='%d.%m.%Y')

monthly_data = final_dataset.groupby(final_dataset['date'].dt.to_period('M')).
 ↪agg({
    'item_cnt_month': 'sum',
}).reset_index()

def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')

    if result[1] <= 0.05:
        print("Stationary (Reject the null hypothesis)")
    else:
        print("Non-Stationary (Fail to reject the null hypothesis)")

item_cnt_month_series = monthly_data['item_cnt_month']

plt.figure(figsize=(12, 6))
plt.plot(item_cnt_month_series)
plt.title('Monthly Item Count Over Time')
plt.xlabel('Date')
plt.ylabel('Item Count')
plt.show()

adf_test(item_cnt_month_series)
```

```
[ ]: numerical_columns = final_dataset.select_dtypes(include=['number'])

#calculating the mean, median and standard deviation for numerical variables
print("\n\nMean of final_dataset:\n")
print(numerical_columns.mean())

print("\n\nMedian of final_dataset:\n")
print(numerical_columns.median())

print("\n\nStandard Deviation of final_dataset:\n")
print(numerical_columns.std())
```

```
[ ]: print("\n\nHead of final_dataset:\n")
print(final_dataset.head(20))
print(final_dataset.shape)
```

```
print(final_dataset.info())
```

[ ]:
```
#export the final dataset to csv file
final_dataset.to_csv('./data-set/output/final_dataset_with_cleaning.csv',␣
↪index=False)
```

1.5 Model Development, Error Analysis & Comparison

[ ]:
```
#prepare the data for modeling
df = pd.read_csv('./data-set/sales_train.csv')
#rename item_cnt_day column
df.rename(columns={'item_cnt_day': 'item_count'}, inplace=True)
#removes duplicates
df.drop_duplicates(inplace=True)
#outlier treatment
df = df[(df['item_count'] > 0) & (df['item_count'] < 307980)]
df = df[df['item_count'] < 1000]
#handles incorrect data
df = df[(df['item_price'] > 0) & (df['item_price'] < 100000)]
#converts date column to datetime format
df['date'] = pd.to_datetime(df['date'], format='%d.%m.%Y')
#convert date to year-month format
df['year-month'] = df['date'].dt.strftime('%Y-%m')
#drop date column and item_price column
df.drop(columns=['date', 'item_price'], inplace=True)
# group features
df_train_group = df.groupby(['year-month', 'shop_id', 'item_id']).sum().
↪reset_index()
# pivot table
df = df_train_group.pivot_table(index=['shop_id', 'item_id'],␣
↪columns='year-month', values='item_count', fill_value=0).reset_index()

print(df.head(10))
print(df.shape)
print(df.info())
```

[ ]:
```
#export the final dataset to csv file
df.to_csv('./data-set/output/dataset_for_modeling.csv', index=False)
```

[ ]:
```
# Create X and y variables for train and test sets
X = df[df.columns[:-1]]
y = df[df.columns[-1]]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

[ ]:
```
#creating evaluation metrics
scores_and_names = []

# Create a function to evaluate the model
def evaluate_the_model(y_true, y_pred, model_name, model):

    # Calculate the MAE
    mae = mean_absolute_error(y_true, y_pred)
    print(f"MAE for {model_name}: {mae:.5f}")

    # Calculate the MSE
    mse = mean_squared_error(y_true, y_pred)
    print(f"MSE for {model_name}: {mse:.5f}")

    # Calculate the RMSE
    rmse = np.sqrt(mse)
    print(f"RMSE for {model_name}: {rmse:.5f}")

    # Plot the predictions vs. the actual values
    plt.figure(figsize=(12, 6))
    plt.plot(y_true, label='Actual Values')
    plt.plot(y_pred, label='Predicted Values')
    plt.title(f'Predictions vs. Actual Values ({model_name})')
    plt.xlabel('Observation')
    plt.ylabel('Item Count')
    plt.legend()
    plt.show()

    scores_and_names.append((model_name, rmse))
```

1.5.1 linear regression

[ ]:
```
# Create a Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Linear Regression', lin_reg)
```

1.5.2 Logistic Regression

[ ]:
```
# Create a logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Logistic Regression', log_reg)
```

1.5.3 SVM

[ ]:
```
# create a support vector machine model
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Support Vector Machine', svm)
```

1.5.4 Decision Tree

[ ]:
```
# create a decision tree model
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Decision Tree', dt)

#visualize the decision tree
feature_names = list(X.columns)
plt.figure(figsize=(20, 10))
plot_tree(dt, filled=True, feature_names=feature_names, max_depth=2,␣
 ↪fontsize=10)
plt.show()
```

1.5.5 random forest

[ ]:
```
# create a random forest model
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Random Forest', rf)
```

1.5.6 Stochastic Gradient Descent

[ ]:
```python
#create a stochastic gradient descent model
sgd_reg = SGDRegressor()
sgd_reg.fit(X_train, y_train)
y_pred = sgd_reg.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Stochastic Gradient Descent', sgd_reg)
```

1.5.7 xtra tree

[ ]:
```python
#create a extra trees model
et = ExtraTreesRegressor()
et.fit(X_train, y_train)
y_pred = et.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Extra Trees', et)
```

1.5.8 XGBoost

[ ]:
```python
#create a xgboost model
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)

evaluate_the_model(y_test, y_pred, 'XGBoost', xgb)
```

1.5.9 ridge regression

[ ]:
```python
#create ridge regression model
ridge = Ridge()
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Ridge Regression', ridge)
```

1.5.10 lasso regression

[ ]:
```python
#create lasso regression model
lasso = Lasso()
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Lasso Regression', lasso)
```

1.5.11 ARIMA

[ ]:
```
#create ARIMA model
arima = ARIMA(y_train, order=(1, 1, 1))
model = arima.fit()
y_pred = model.predict(start=len(y_train), end=len(y_train) + len(X_test) - 1,␣
 ↪exog=X_test)

evaluate_the_model(y_test, y_pred, 'ARIMA', arima)
```

1.5.12 ADABOOST

[ ]:
```
#create adaboost model
ada = AdaBoostRegressor()
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)

evaluate_the_model(y_test, y_pred, 'AdaBoost', ada)
```

1.5.13 BayesianRidge

[ ]:
```
# create bayesian ridge model
br = BayesianRidge()
br.fit(X_train, y_train)
y_pred = br.predict(X_test)

evaluate_the_model(y_test, y_pred, 'Bayesian Ridge', br)
```

1.5.14 KNN

[ ]:
```
# create a knn model
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train.values, y_train.values)
y_pred = knn.predict(X_test.values)

evaluate_the_model(y_test.values, y_pred, 'K-Nearest Neighbors', knn)
```

1.5.15 Compare Models

[ ]:
```
results = pd.DataFrame(scores_and_names, columns=['Model', 'RMSE'])
```

[ ]:
```
#sort the results
results.sort_values(by='RMSE', ascending=True, inplace=True)
```

[ ]:
```
#print the results in tabel format
print(results)
```

[ ]:
```
#print the best model from the results with model name and score
print(f"\nBest Model: {results.iloc[0, 0]}")
print(f"RMSE: {results.iloc[0, 1]:.5f}")
```

1.6 Data Visualization

[ ]:
```
#line chart
plt.figure(figsize=(12, 6))
plt.plot(final_dataset['date'], final_dataset['revenue'])
plt.title('Revenue Over Time')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.show()
```

[ ]:
```
#bar chart
plt.figure(figsize=(12, 6))
plt.bar(final_dataset['shop_name'], final_dataset['revenue'])
plt.title('Revenue by Shop')
plt.xlabel('Shop Name')
plt.ylabel('Revenue')
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#pairplot
sns.pairplot(final_dataset[['item_price', 'item_cnt_month', 'revenue']])
plt.show()
```

[ ]:
```
#boxplot
plt.figure(figsize=(12, 6))
sns.boxplot(data=final_dataset, x='item_category_name', y='revenue')
plt.title('Revenue by Item Category')
plt.xlabel('Item Category')
plt.ylabel('Revenue')
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#scatter chart
plt.figure(figsize=(12, 6))
sns.scatterplot(data=final_dataset, x='item_price', y='revenue')
plt.title('Revenue vs. Item Price')
plt.xlabel('Item Price')
plt.ylabel('Revenue')
plt.show()
```

[ ]:
```
#histogram
plt.figure(figsize=(12, 6))
sns.histplot(data=final_dataset, x='item_price', kde=True, bins=20)
```

```
plt.title('Distribution of Item Count Per Month')
plt.xlabel('Item Count')
plt.ylabel('Frequency')
plt.show()
```

[ ]:
```
#area plot
plt.figure(figsize=(12, 6))
plt.stackplot(final_dataset['date'], final_dataset['revenue'])
plt.title('Revenue Over Time')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.show()
```

[ ]:
```
#heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

[ ]:
```
# Time Series Plot
plt.figure(figsize=(10, 6))
final_dataset['date'] = pd.to_datetime(final_dataset['date'])
sns.lineplot(x='date', y='revenue', data=final_dataset)
plt.title('Time Series Plot of Revenue')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.show()
```

[ ]:
```
#sales data per store (pie chart)

#group the data by shop_name and sum the revenue column
grouped_by_shop_name = final_dataset.groupby(['shop_name']).agg({'revenue':␣
 ↪'sum'})
plt.figure(figsize=(20, 20))
plt.pie(grouped_by_shop_name['revenue'], labels=grouped_by_shop_name.index,␣
 ↪autopct='%1.1f%%')
plt.title('Sales Data Per Store')
plt.show()
```

[ ]:
```
#mean monthly sales
grouped_by_month_name = final_dataset.groupby(['month_name']).agg({'revenue':␣
 ↪'mean'})
sns.lineplot(x=grouped_by_month_name.index, y=grouped_by_month_name['revenue'])
plt.title('Mean Monthly Sales')
plt.xlabel('Month')
plt.ylabel('Mean sales')
```

```
plt.xticks(rotation=45)
plt.show()
```

[ ]:
```
#mean sales compression across the years
grouped_by_year_num = final_dataset.groupby(['year_num']).agg({'revenue':␣
↪'mean'})
sns.lineplot(x=grouped_by_year_num.index, y=grouped_by_year_num['revenue'])
plt.title('Mean Sales Compression Across The Years')
plt.xlabel('Year')
plt.ylabel('Mean Sales')
plt.show()
```

[ ]:
```
#Montly sales Mean, Median, and Standard Deviation
grouped_by_month_name = final_dataset.groupby(['month_name']).agg({'revenue':␣
↪['mean', 'median', 'std']})
plt.figure(figsize=(12, 6))
sns.lineplot(x=grouped_by_month_name.index,␣
↪y=grouped_by_month_name['revenue']['mean'], label='Mean')
sns.lineplot(x=grouped_by_month_name.index,␣
↪y=grouped_by_month_name['revenue']['median'], label='Median')
sns.lineplot(x=grouped_by_month_name.index,␣
↪y=grouped_by_month_name['revenue']['std'], label='Standard Deviation')
plt.title('Monthly Sales Mean, Median, and Standard Deviation')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

[ ]:
```
#popular item categories
grouped_by_item_category_name = final_dataset.groupby(['item_category_name']).
↪agg({'revenue': 'sum'})
grouped_by_item_category_name.sort_values(by='revenue', ascending=False,␣
↪inplace=True)
plt.figure(figsize=(12, 6))
plt.pie(grouped_by_item_category_name['revenue'][:10],␣
↪labels=grouped_by_item_category_name.index[:10], autopct='%1.1f%%')
plt.title('Popular Item Categories')
plt.show()
```

[ ]:
```
#average sales per item category
grouped_by_item_category_name = final_dataset.groupby(['item_category_name']).
↪agg({'revenue': 'mean'})
grouped_by_item_category_name.plot(kind='bar', figsize=(12, 6))
plt.title('Average Sales Per Item Category')
plt.xlabel('Item Category')
```

```
plt.ylabel('Sales')
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#average monthly sales per year
grouped_by_year_num_and_month_name = final_dataset.groupby(['year_num',␣
 ↪'month_name']).agg({'revenue': 'mean'})
sns.lineplot(data=grouped_by_year_num_and_month_name, x='month_name',␣
 ↪y='revenue', hue='year_num')
plt.title('Average Monthly Sales Per Year')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.legend(title='Year', loc='upper right', labels=['2013', '2014', '2015'])
plt.show()
```

[ ]:
```
#average store sales
grouped_by_shop_name = final_dataset.groupby(['shop_name']).agg({'revenue':␣
 ↪'mean'})
grouped_by_shop_name.plot(kind='bar', figsize=(12, 6))
plt.title('Average Store Sales')
plt.xlabel('Store')
plt.ylabel('Sales')
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#average store sales - year wise
grouped_by_shop_name_and_year_num = final_dataset.groupby(['shop_name',␣
 ↪'year_num']).agg({'revenue': 'mean'})
sns.lineplot(data=grouped_by_shop_name_and_year_num, markers=True,␣
 ↪dashes=False, x='shop_name', y='revenue', hue='year_num')
plt.title('Average Store Sales - Year Wise')
plt.xlabel('Store')
plt.ylabel('Sales')
plt.legend(title='Year', loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation=90)
plt.show()
```

[ ]:
```
#relationship between item price and revenue
plt.figure(figsize=(12, 6))
sns.scatterplot(data=final_dataset, x='item_price', y='revenue')
plt.title('Relationship Between Item Price and Revenue')
plt.xlabel('Item Price')
plt.ylabel('Revenue')
plt.show()
```

```
[ ]: #relationship between item price and revenue - year wise
plt.figure(figsize=(12, 6))
sns.scatterplot(data=final_dataset, x='item_price', y='revenue', hue='year_num')
plt.title('Relationship Between Item Price and Revenue - Year Wise')
plt.xlabel('Item Price')
plt.ylabel('Revenue')
plt.legend(title='Year', loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```

```
[ ]: #relationship: month of year vs sales
grouped_by_month_name = final_dataset.groupby(['month_name']).agg({'revenue':
    'sum'})
sns.lineplot(x=grouped_by_month_name.index, y=grouped_by_month_name['revenue'])
plt.title('Month of Year vs Sales')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.show()
```

```
[ ]: #export the final dataset to csv file
final_dataset.to_csv('./dashboard_dataset.csv', index=False)
```